Universidade Estadual de Campinas - UNICAMP
Faculdade de Engenharia Elétrica e de Computação

Transformada de Distância por Morfologia Matemática

Autor: **Francisco de Assis Zampirolli**
Orientador: **Prof. Roberto de Alencar Lotufo, Ph.D.**

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: **Engenharia de Computação**.

Banca Examinadora

Roberto de Alencar Lotufo, Ph.D.	DCA/FEEC/UNICAMP
Marcos Cordeiro d'Ornellas, Ph.D.	INF/UFSM
Roberto Hirata Jr., Ph.D.	SENAC/SP
Neucimar J. Leite, Ph.D.	IC/UNICAMP
Clésio Luiz Tozzi, Ph.D.	DCA/FEEC/UNICAMP

Junho de 2003
Campinas, SP - Brasil

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Z189f

Zampirolli, Francisco de Assis
Transformada de Distância por Morfologia Matemática / Francisco de Assis Zampirolli. – Campinas, SP: [s.n.], 2003.
Orientador: Roberto de Alencar Lotufo.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Transformada de Distância.
2. Transformada de Distância Euclidiana.
3. Morfologia Matemática.
4. Erosão Morfológica.
5. Padrões de Algoritmos.
6. Geração Automática de Códigos e Documentos.
I. Lotufo, Roberto de Alencar.
II. Universidade Estadual de Campinas.
Faculdade de Engenharia Elétrica e de Computação.
III. Título

Resumo

Esta tese consiste no estudo de algoritmos da transformada de distância que podem ser descritos usando erosões morfológicas caracterizadas por funções estruturantes. Estas erosões, juntamente com as dilatações, formam a base da morfologia matemática. Foram utilizadas as propriedades morfológicas de transformação idempotente, que ocorre com argumentos particulares na erosão, e de decomposição de função estruturante para melhorar o desempenho das implementações. As erosões foram implementadas nos padrões de varredura paralelo, seqüencial e por propagação. Este processo também pode ser feito para outros operadores morfológicos, como dilatação, reconstrução e transformada de distância. Os algoritmos clássicos da transformada de distância usando erosões foram reescritos (reimplementados) e classificados através destes padrões, produzindo códigos mais simples e eficientes.

Com base nesses estudos foram propostos dois algoritmos para a transformada de distância euclidiana: um por propagação direcional e outro multidimensional. Este segundo produz a transformada por várias erosões unidimensionais, onde as funções estruturantes usadas nas erosões pertencem a uma família de quatro funções estruturantes direcionais de dois pixels. O algoritmo de erosão, baseado na varredura por propagação, é muito simples de codificar e entender, sendo inclusive um dos mais rápidos na literatura.

Abstract

This thesis refers to distance transformation algorithms that can be written using a fundamental morphological operator. The performance of the algorithms is an important research problem. To improve them, some algebraic properties like indempontence and decomposition of structuring functions needs to be used. Morphological operators can be coded in parallel, sequential, or propagation sweeping patterns. Using these patterns, distance operators have been classified and rewritten in order to obtain neat and efficient coding.

With base in these studies was proposed directional propagation and multidimensional Euclidean distance transformation algorithms. In this second algorithm the distance transformation can be composed by several one dimensional erosions. The structuring functions used in the erosion belongs to a family of four directional one-dimensional two-point structuring functions. The erosion algorithm is based on a propagation scheme very simple to code and understand, yet being one of the fastest Euclidean distance transformation algorithms in the literature.

Agradecimentos

À orientação do Ph.D. Roberto de Alencar Lotufo dada no final do mestrado, nos trabalhos de especialização e durante as pesquisas do doutorado, sempre prestativo e profissional, contribuindo para o meu crescido acadêmico.

À Faculdade de Engenharia Elétrica e de Computação da UNICAMP. Em especial aos professores e funcionários do DCA pela acolhida.

Ao apoio financeiro, fundamental para se fazer pesquisa:

- Ao CNpQ, com a bolsa de Desenvolvimento Tecnológico Industrial, de agosto de 1997 a setembro de 1998, processo: $380443/1997-0$, onde, além de fazer a especialização em geração automática de documentos, fiz vários créditos do doutorado como aluno especial;
- À FAPESP pela bolsa de doutorado, de outubro de 1998 a janeiro de 2002, processo: $98/06641-6$;
- À Faculdade SENAC de Ciências Exatas e Tecnologia, pela liberação para as pesquisas de doutorado a partir de fevereiro de 2002.

A minha banca, em especial aos doutores Roberto Hirata Jr. e Marcos Cordeiro d'Ornellas pelas discussões e precisão nas correções do texto.

Aos companheiros Rubens Machado (pelas dicas de programação no ambiente *Adesso*) e Luciano Silva (pela ajuda na especificação da linguagem definida no apêndice).

Aos amigos Fábio Henrique Viduani Martinez e Marco Aurélio Stefanes pelas correções no texto.

Finalmente, e em especial, à Cristina, minha esposa, pelo incentivo dado nestes longos anos de pesquisa e muitas horas de isolamento (as vezes gerando frutos e outras não), mas sempre incentivando a concretizar este sonho.

Muito Obrigado!

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos

Capítulo 1

Introdução

O processamento de imagens tem-se tornado uma ferramenta decisiva na solução de problemas em diversas áreas da ciência e da tecnologia e um instrumento muito utilizado é a transformada de distância.

Algumas aplicações de processamento de imagens são descritas a seguir: caracterizar a forma e as dimensões de um tumor a partir de uma imagem de raios-X, ou de uma tomografia computadorizada é um auxílio importante na formulação de um diagnóstico médico; estimar a quantidade de grãos que serão colhidos numa safra agrícola, a partir de imagens aéreas ou de satélites, pode ajudar a controlar estoques em armazéns e financiamentos em bancos; desenvolver algoritmos de compressão de imagens são de grande importância para armazenamento, transmissão e processamento de imagens; reconhecer caracteres para a leitura de textos e verificar o controle de qualidade por inspeção visual; entre outras.

A *transformada de distância* (TD) associa os pixels de um objeto numa imagem binária a sua menor distância aos pixels de fundo da imagem. A TD é um instrumento importante em processamento de imagens pois pode ser usada em muitas outras transformações, como: dilatação, erosão, caminho mínimo entre dois pixels, esqueleto, *SKIZ* (*SKeleton of Influence Zone*), diagrama de Voronoi, casamento de formas, dimensão *fractal*,

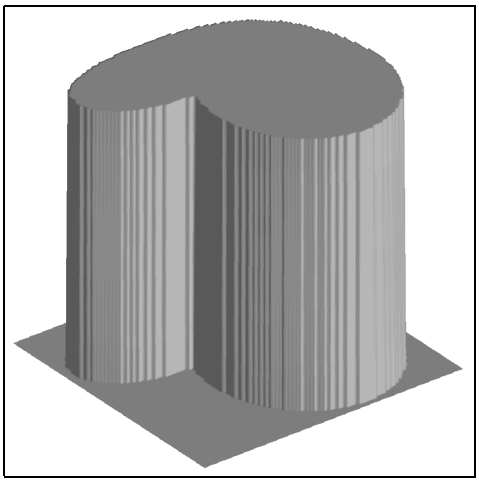
etc. [Dan80, Vin91, Rag92, SC94, Egg98, MCSM02]. Além de ser um operador básico em processamento de imagens, o seu estudo auxilia no estudo de outros algoritmos similares, como o *watershed* e a *IFT* (*Image Floresting Transform*). Assim, melhorar a eficiência da TD, melhora-se a eficiência de algoritmos similares. A TD pode também ser formada por uma seqüência de operações locais, usando vizinhanças pequenas, inclusive unidimensionais, tornando os algoritmos mais simples e eficientes de codificar e entender. Existe uma relação íntima entre a TD e a morfologia matemática e o objetivo desta tese é estudar esta relação, contribuindo também com algoritmos novos para a TD.

1.1 Transformada de distância

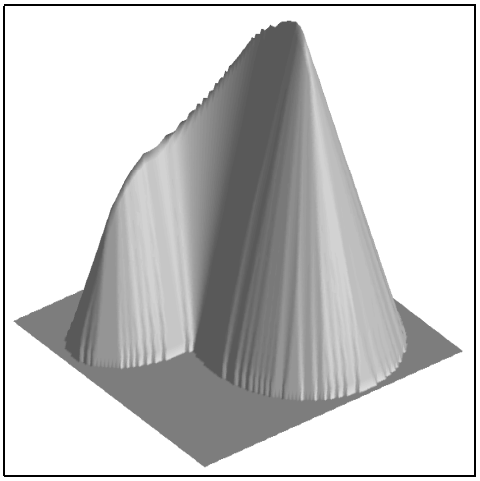
Uma *imagem binária* bidimensional é uma função f que mapeia os elementos (ou *pixels*) de um espaço $\mathbb{E}$ em $\{0, 1\}$, onde $\mathbb{E}$ é usualmente uma matriz ou grade hexagonal. Os pixels com valor 1 na imagem f formam os *objetos* e os pixels com valor 0 formam o *fundo* da imagem. A posição de um pixel é dada pela sua posição na matriz. Assim, a linha r e a coluna c do pixel são associados ao ponto do plano cartesiano (r, c). Desta maneira, qualquer função distância definida no plano cartesiano induz uma função distância no domínio das imagens [BGKW95].

Para uma dada função distância, a *transformada de distância* (TD) atribui aos pixels de um objeto de uma imagem binária a distância mínima entre estes pixels e os pixels de fundo. A Figura 1.1 ilustra a TD aplicada na imagem f e também alguns contornos, onde todos os pixels de um contorno têm a mesma distância ao fundo da imagem.

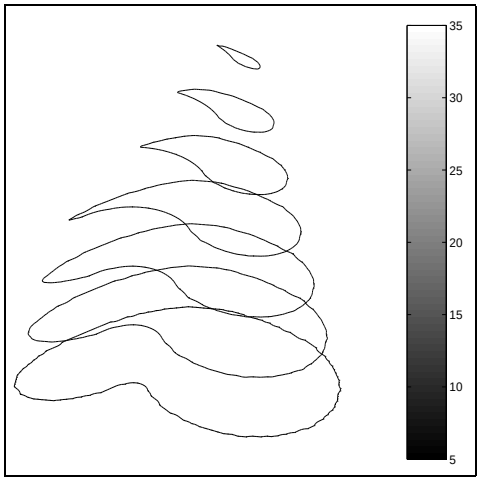
A *métrica* mais natural para computar a distância na maioria das aplicações é a *euclidiana*, principalmente devido a sua propriedade de invariância à rotação. Tendo em vista a dificuldade de construir algoritmos eficientes para a *transformada de distância euclidiana* (TDE), muitos



f



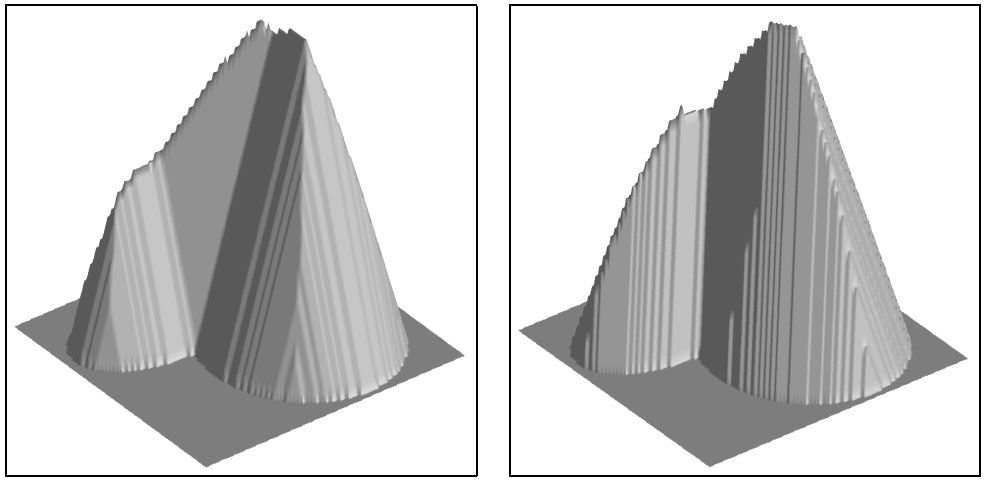
TD



contornos

Figura 1.1: Imagem de entrada f e o resultado da aplicação da transformada de distância (métrica euclidiana), com alguns contornos onde a legenda são as distâncias.

pesquisadores desenvolveram algoritmos aproximados. A métrica mais popular para a TD é conhecida como *chanfrada* (*chamfer*) [Bor86] e será usada nesta tese como *transformada de distância chanfrada* (TDC). Sua popularidade se deve a simplicidade, velocidade e aproximação razoável para a TDE. A Figura 1.2 ilustra as métricas mais antigas utilizadas na TD e são casos particulares da TDC: *city-block* e *chessboard*. A TDE é ilustrada na imagem à direita da Figura 1.1. Outro tipo de TD é a transformada que procura ser euclidiana, porém esbarra no problema da conexidade do diagrama de Voronoi discreto, este tipo é descrito neste documento como *transformada de distância euclidiana aproximada* (TDEA). Danielsson foi o primeiro a criar um algoritmo do tipo TDEA [Dan80].



city-block *chessboard*

Figura 1.2: Métricas usadas na TD: *city-block* e *chessboard*.

Além das métricas utilizadas na TD é interessante saber o tipo (ou *padrão*) de algoritmo utilizado para computar a TD. Uma introdução aos padrões de algoritmos de processamento de imagens é apresentada a seguir.

1.2 Padrões de algoritmos

O uso de padrões de algoritmos auxilia a classificação das transformações de processamento de imagens. O objetivo destes padrões é identificar quais algoritmos implementam o mesmo operador e usam diferentes arquiteturas, possibilitando um melhor entendimento da área. Existem vários padrões de algoritmos [D'O01], porém neste documento são analisados os *padrões de varredura*: *paralelo*, *seqüencial* e *por propagação*.

Existem dois padrões de algoritmos em processamento de imagens conhecidos na literatura desde a década de 60: os *paralelos* (ou *iterativos*) e os *seqüenciais* (ou *recursivos*) [RP66, RP68]. Desde aquela época, a preocupação em desenvolver operações eficientes sempre foi objetivo de pesquisa, como descrito no artigo [RP66]:

> *Programas de computador para processamento de imagens digitalizadas receberam atenção crescente nos últimos anos. Muitos dos trabalhos feitos neste campo envolveram performance de operações "locais"... Como vários pesquisadores mostraram, uma variedade grande de transformações de processamento de imagens pode ser realizada aplicando tais operações independentemente, ou "em paralelo", para cada elemento da imagem... É sugerido que operações locais sejam executadas em uma seqüência definida, usando em cada passo os resultados obtidos anteriormente... É mostrado que as aproximações paralelas e seqüenciais são matematicamente equivalentes...*

Nos algoritmos paralelos, os pixels são processados independentemente da ordem de varredura da imagem, dependendo apenas dos valores dos pixels da imagem de entrada e da vizinhança usada. Isto não ocorre nos algoritmos seqüenciais, onde a imagem de saída depende tanto dos pixels da imagem de entrada quanto dos valores calculados anteriormente, além da ordem de varredura na imagem.

Os algoritmos paralelos possuem a característica de serem ineficientes em computadores seqüenciais [Vin92]. O mesmo não ocorre com os algoritmos seqüenciais, onde transformações como a transformada de distância podem ser encontradas em apenas uma varredura na *ordem raster* e uma outra na *ordem anti-raster* [RP66] [1].

Para obter transformações eficientes em máquinas seqüenciais é apresentado um estudo da equivalência entre as erosões morfológicas paralela e seqüencial, que é uma aplicação da equivalência definida por Rosenfeld e Pfaltz [RP66]. Em vez de realizar uma erosão paralela usando uma função estruturante "grande"(por exemplo, com o tamanho do domínio equivalente a duas vezes o da imagem a ser processada) é apresentado uma erosão seqüencial equivalente usando uma função estruturante "pequena"(por exemplo, 3×3, 1×3 ou 1×2), que decompõe a função estruturante "grande"e obtém o mesmo resultado na transformação.

Além das varreduras paralelas e seqüenciais, outras técnicas de varreduras têm sido introduzidas [Vin92]. Neste texto é usada a *varredura por propagação* (ou *padrão por propagação*). Um algoritmo implementado utilizando este padrão usa o fato de que nem todos os pixels são modificados na operação. Com isso, o algoritmo coloca apenas os pixels que potencialmente influenciam no resultado da operação em uma estrutura de dados – *fila*, *conjunto* ou *fila hierárquica*. O conjunto destes pixels é chamado de *fronteira*. O padrão por propagação é útil apenas em casos de operações sucessivas, como para computar a TD usando sucessivas erosões, isto é, quando a função estruturante foi decomposta. Para este caso particular (devido à propriedade de idempotência de erosões particulares), após a criação da fronteira, o algoritmo entra num laço que só termina quando o mesmo estiver vazio. Dentro deste laço, um pixel é retirado por vez da fronteira para analisar os pixels vizinhos e decidir quais destes vizinhos irão fazer parte da nova fronteira. Este processo se repete até não existir nenhum conjunto fronteira.

[1] As varreduras *raster* e *anti-raster* serão definidas na Seção 3.2.

Na literatura, o uso de operações por propagação é confuso pois em algumas vezes usa-se fila, em outras fila hierárquica. Para o caso da erosão, a fronteira é um conjunto que pode ser processado em qualquer ordem (análogo ao ocorrido no caso paralelo) e neste caso uma estrutura de conjunto (por exemplo, vetor) é suficiente para armazenar a fronteira.

Como contribuição, além de ser suficiente o uso do conjunto para armazenar a fronteira, é apresentado a condição para processar uma seqüência de erosões com elementos estruturantes diversos. Em vez de analisar a equivalência entre os padrões paralelo e seqüencial introduzido anteriormente, é feito uma análise da equivalência entre os padrões paralelo e por propagação. Esta análise contribui para a criação de novos algoritmos, principalmente para a TDE.

Existem entretanto situações em que se pode otimizar o processo de cópia da imagem no laço entre a operação com uma função estruturante e outra. Neste caso o uso da fila FIFO pode ser útil, como em alguns algoritmos de Vincent [Vin92] e também no algoritmo da TDE multidimensional introduzido na próxima seção.

1.3 Algoritmos rápidos para a TDE

Usando os padrões de algoritmos definidos na seção anterior e os trabalhos de Ragnemalm, Eggers e Mitchell *et. al.* [Rag92, Egg98, SM92, HM94], esta tese contribui com dois algoritmos para a *transformada de distância euclidiana* (TDE), introduzidos a seguir.

Os algoritmos mais eficientes para a TDE seguem o padrão por propagação, onde podem ser divididos em outros quatro tipos: propagação vetorial, propagação ordenada, propagação paralela (ou simplesmente, propagação) e geométrico. Os algoritmos da TD por *propagação vetorial* são difíceis de relacionar como a morfologia matemática, pois são decompostos em informações vetoriais do plano cartesiano discreto. O algoritmo de

Danielsson [Dan80] e a primeira parte do algoritmo de Cuisenaire [Cui99] são exemplos de algoritmos por propagação vetorial para a TDEA. Os algoritmos por *propagação ordenada* são melhores modelados como o problema de caminho mínimo usado em teoria dos grafos e implementados usando fila hierárquica. Os algoritmos por *propagação* (ou *propagação paralela*) têm relações diretas com a morfologia matemática e serão estudados neste documento. Os algoritmos *geométricos* usando intersecção de parábolas formam o último tipo de TD por propagação. Saito e Toriwaki [ST94] mostraram que para este tipo a TDE pode ser separável. Esta característica também ocorre quando a TDE é obtida por erosões morfológicas unidimensionais. Nesta linha existem vários outros artigos para a TDE, tais como o algoritmo definido por Meijster e Roerdink [MR00], um dos mais eficientes reportados na literatura, e também tratado nesta tese.

Como contribuição de algoritmos para a TDE, esta tese apresenta os seguintes algoritmos:

***TD por propagação direcional*:** semelhante em vários aspectos ao proposto por Eggers [Egg98] para a métrica euclidiana, porém mais simples de implementar e entender. Este algoritmo usa erosões por propagação por funções estruturantes direcionais. Além disso, computa a TDE se as funções estruturantes especiais definidas por Huang e Mitchell [HM94] forem utilizadas. Este algoritmo será apresentado na Seção 4.4.

Na *erosão por propagação*, apenas os pixels de fronteira são processados, o que diminui a varredura redundante presente no algoritmo paralelo. Quando apenas a direção de propagação é considerada diminui-se ainda mais o processamento desnecessário [ZL00].

***TDE multidimensional*:** usa decomposições unidimensionais (1D) das funções estruturantes formadas por dois pixels direcionais. Este algoritmo é separável e dimensões elevadas podem ser computadas a partir do algoritmo 1D, exibindo o mesmo comportamento de paralelismo de

dimensões elevadas dos algoritmos de FFT [LZ01]. Este algoritmo será apresentado na Seção 4.5 e suas principais características são resumidas a seguir:

- Transformada de distância euclidiana exata e discreta;
- Bem adaptado para dimensões mais altas;
- Simplicidade de código quando comparado a outros algoritmos eficientes da TDE;
- Usa somente operações de comparação e adição;
- Um dos mais rápidos algoritmos conhecidos para computadores seqüenciais;
- Paralelização eficiente.

1.4 Resumo das principais contribuições

As principais contribuições desta tese são:

- Novo enfoque de padrões de algoritmos morfológicos: paralelo, seqüencial e por propagação;
- Estudo da equivalência entre erosões paralelas, seqüenciais e por propagação;
- Classificação, projeto e implementação de algoritmos clássicos da transformada de distância usando esses padrões;
- Construção de novos algoritmos da transformada de distância.

Para os padrões de algoritmos concluí-se que não existe a necessidade de usar uma *fila* no padrão por propagação – pois esta pode ser substituída por um *conjunto* – que armazena os pixels de *fronteira* de um objeto numa imagem digital.

Como exemplo da equivalência entre operações paralelas e seqüenciais usando uma vizinhança local definida por Rosenfeld e Pfaltz [RP66], são apresentados estudos entre erosões paralelas, seqüenciais e por propagação.

Na classificação, projeto e implementação de algoritmos clássicos da TD, a escrita e o entendimento destes algoritmos são simplificados, mostrando que podem ser construídos através do encadeamento de erosões morfológicas, destacando o estudo do algoritmo por propagação definido por Vincent [Vin92].

Este trabalho contribui com dois novos algoritmos para a transformada de distância euclidiana: um por propagação direcional e outro multidimensional. Com isso, são obtidos algoritmos simples, onde o segundo pode ser considerado como um dos mais eficientes para imagens reais.

1.5 Conteúdo deste documento

O Capítulo 2 apresenta a dilatação e a erosão caracterizadas por funções estruturantes. Estas definições são feitas para as imagens binárias e em níveis de cinza. Ainda neste capítulo é apresentada uma notação para descrever os algoritmos apresentados nos capítulos subseqüentes. O Capítulo 3 apresenta erosões morfológicas, definidas em imagens em níveis de cinza, nos padrões paralelo, seqüencial e por propagação. Neste capítulo também é apresentado um estudo da equivalência entre estes padrões, com o objetivo de obter transformações eficientes. Além disso, é apresentado um resumo de padrões de algoritmos da erosão morfológica. O Capítulo 4 contém a classificação de algoritmos da TD e algoritmos novos para a TDE por propagação direcional e multidimensional. Finalmente, o Capítulo 5 apresenta os resultados e as conclusões da tese.

Referências Bibliográficas

[BGKW95] H. Breu, J. Gil, D. Kirkpatrick e M. Werman. Linear time Euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995.

[Bor86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[Cui99] O. Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. Tese de Doutorado, Université Catholique de Louvain, Bélgica, 1999.

[Dan80] P.E. Danielsson. Euclidean distance mapping. *Computer Vision, Graphics and Image Processing*, 14:227–248, 1980.

[D'O01] M.C. D'Ornellas. *Algorithmic Patterns for Morphological Image Processing*. Tese de Doutorado, University of Amsterdam, Amsterdam, 2001.

[Egg98] H. Eggers. Two fast Euclidean distance transformations in $\mathbb{Z}^2$ based on sufficient propagation. *Computer Vision, Graphics and Image Processing*, 69(1), 1998.

[HM94] C.T. Huang e O.R. Mitchell. A Euclidean distance transform using grayscale morphology decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:443–448, 1994.

[LZ01] R.A. Lotufo e F.A. Zampirolli. Fast multidimensional parallel Euclidean distance transform based on mathematical morphology. No *Brazilian Symposium on Computer Graphics and Image Processing*, páginas 100–105, Florianopolis, Brasil, Outubro 2001.

[MCSM02] E.T.M. Manoel, L.F. Costa, J. Streicher e G.B. Muller. Multiscale fractal characterization of three-dimensional gene expression data. No *Brazilian Symposium on Computer Graphics and Image Processing*, Fortaleza, RN, Brasil, Outubro 2002.

[MR00] A. Meijster e J.B.T.M. Roerdink. A general algorithm for computing distance transforms in linear time. No *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, páginas 331–340. Palo Alto, USA, Junho 2000.

[Rag92] I. Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *Computer Vision, Graphics and Image Processing: Image Understanding*, 56(3), 1992.

[RP66] A. Rosenfeld e J.L. Pfalz. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, 13(4):471–494, Outubro 1966.

[RP68] A. Rosenfeld e J.L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[SC94] Y. Sharaiha e N. Christofides. Graph-theoretic approach to distance transformations. *Pattern Recognition Letters*, 15(10), 1994.

[SM92] F.Y.-C. Shih e O.R. Mitchell. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing*, 1:197–204, 1992.

[ST94] T. Saito e J. I. Toriwaki. New algorithms for Euclidean distance transformations of an n-dimensional digitized picture with applications. *Pattern Recognition*, 27:1551–1565, 1994.

[Vin91] Luc Vincent. Exact euclidean distance function by chain propagations. No *IEEE Int. Computer Vision and Pattern Recog. Conference*, páginas 520–525, Maui, HI, Junho 1991.

[Vin92] L. Vincent. Morphological algorithms. No E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, capítulo 8, páginas 255–288. Marcel Dekker, New York, 1992.

[ZL00] F.A. Zampirolli e R.A. Lotufo. Algoritmos rápidos para a transformada distância euclidina baseados em morfologia matemática. No *Workshop on Artificial Intelligence and Computer Vision*, Atibaia, SP, Brasil, Novembro 2000.

Capítulo 2

Operadores Morfológicos

Uma forma elegante de resolver problemas de processamento de imagens é através da utilização de uma base teórica consistente. Uma destas teorias é a *morfologia matemática* criada na década de 60 por Jean Serra e George Matheron na *École Nationale Superiéure des Mines de Paris*, em Fontainebleau, França. Esta teoria diz que é possível fazer transformações entre reticulados completos[1], os quais são chamados de *operadores morfológicos*. Na morfologia matemática existem quatro classes básicas de operadores: dilatação, erosão, anti-dilatação e anti-erosão, chamadas de *operadores elementares*. Banon e Barrera [BB93] provaram que todos os operadores morfológicos invariantes por translação podem ser obtidos a partir de combinações de operadores elementares juntamente com as operações de união e intersecção. Além disso, quando um reticulado possui uma *família sup-geradora*, estes operadores podem ser caracterizados por *funções estruturantes*. Usando estes operadores elementares é possível construir uma *linguagem formal*, a *linguagem morfológica*, e sua implementação é chamada *máquina morfológica* [BB92]. Um exemplo de uma máquina morfológica é a *MMach* [BBL94].

[1]Um conjunto qualquer com uma relação de ordem é um reticulado completo se todo subconjunto não vazio tem um supremo e um ínfimo. Para detalhes da teoria dos reticulados veja [Bir67].

A teoria dos reticulados estudada em morfologia matemática é abrangente e o leitor interessado pode consultar, por exemplo, os trabalhos de Serra, Banon e Barrera [Ser88, BB94]. Banon e Barrera [BB94] também fizeram estudos de caracterização de operadores morfológicos por funções estruturantes. Zampirolli [Zam97] fez um estudo de *operadores baseados em grafos*. Porém as distâncias em um grafo estão relacionadas aos curtos das arestas ou vértices.

Serão apresentados neste capítulo estudos de casos da dilatação e da erosão caracterizadas por funções estruturantes nos reticulados das imagens binárias e em níveis de cinza.

2.1 Erosão e dilatação binária

Existem várias formas de escrever a erosão e a dilatação binária. Uma das mais antigas é a *subtração de Minkowski* e a *soma de Minkowski*, respectivamente. Seja $\mathbb{E}$ um grupo Abeliano[2]. Sejam X e B dois subconjuntos de $\mathbb{E}$. A *subtração de Minkowski* é um subconjunto de $\mathbb{E}$ definida da seguinte forma:

$$X \ominus B = \{x \in \mathbb{E} : B_x \subset X\}, \tag{2.1}$$

onde B_x é a *translação* de B por x [BB94]. Esta translação pode ser interpretada como a vizinhança B de x [Hei94]. Em morfologia matemática, o conjunto B é chamado de *elemento estruturante*. A notação da subtração de Minkowski também é particular, $\varepsilon_B(X)$ – lê-se erosão de X por B. Analogamente, a *soma de Minkowski* é definida da seguinte forma:

$$X \oplus B \;\; = \;\; \cup\{X_y : y \in B\}. \tag{2.2}$$

[2]Um conjunto com uma operação de adição e as propriedades comutativa, associativa, existência do elemento neutro e existência do oposto é um grupo Abeliano.

A notação usada em morfologia matemática para a soma de Minkowski é $\delta_B(X)$ – lê-se dilatação de X por B.

Seja X^c o *complemento* de X. Então a *subtração de Minkowski* pode ser obtida através da *soma de Minkowski* e vice-versa [WB88], isto é,

$$X \ominus B = (X^c \oplus B)^c.$$

Serão apresentadas a seguir algumas definições para deixar conceitualmente consistentes as implementações que serão descritas a partir do próximo capítulo e a teoria de morfologia matemática.

Sejam $\mathbb{E}$ e K dois conjuntos quaisquer finitos e não vazios. Um *operador* ψ entre $\mathbb{E}$ e K é definido como um mapeamento de $\mathbb{E}$ em K e denotado $\psi : \mathbb{E} \to K$ ou $\psi \in K^{\mathbb{E}}$.

O conjunto $\mathcal{P}(\mathbb{E})$ de todos os subconjuntos de $\mathbb{E}$, com a relação usual de inclusão $\subset$ entre subconjuntos, é um reticulado completo e é chamado de conjunto das partes de $\mathbb{E}$. As operações de ínfimo e supremo são, respectivamente, as operações de intersecção e união entre subconjuntos. Um elemento $X \subset \mathcal{P}(\mathbb{E})$ pode representar, por exemplo, uma *imagem binária* se $X \in \{0,1\}^{\mathbb{E}}$ – o valor de um elemento de $\mathbb{E}$ é 1 se ele pertence a X e 0, caso contrário.

Seja uma *imagem digital*, ou simplesmente *imagem*, como sendo uma função do reticulado $K^{\mathbb{E}}$. Assim, se f é uma imagem então $f \in K^{\mathbb{E}}$. O conjunto $\mathbb{E}$ representa o *domínio da imagem* podendo ser *unidimensional* se $\mathbb{E} \subset \mathbb{Z}$, onde $\mathbb{Z}$ representa o conjunto dos números inteiros e *bidimensional*, se $\mathbb{E} \subset \mathbb{Z} \times \mathbb{Z}$. Existem também imagens tridimensionais, imagens com domínio em uma grade hexagonal, imagens representadas por grafos, etc. Sem perda de generalidade, seja a origem de uma imagem definida no pixel zero no caso unidimensional (pixel mais à esquerda) e $(0,0)$ no caso bidimensional (topo esquerdo da imagem). O conjunto K representa o *contra-domínio da imagem* ou os valores que os pontos da imagem podem assumir. Para as imagens binárias considere $K = \{0, k\}$, onde k é um valor diferente de 0, por exemplo,

$k = 1$ ou $k = \infty$. Para as imagens em níveis de cinza considere o intervalo $K = [0, k]$, onde $k = 255$ para imagens com níveis de cinza representadas no tipo *byte*. Para representação computacional será definida uma imagem em uma matriz com domínio em $\mathbb{E}$ e o tipo desta matriz representa os níveis de cinza [Vin92]. Observe que esta representação se refere às imagens tratadas neste texto, ou seja, imagens unidimensionais ou bidimensionais, dos tipos binários ou em níveis de cinza. Para imagens representadas por grafos, por exemplo, a estrutura de dados é mais complexa. Apesar desta variedade de possibilidades de tipos de imagens, todos os algoritmos apresentados nesta tese são genéricos quanto ao domínio $\mathbb{E}$. O que muda são as implementações nas linguagens de programação de propósito geral, como *C* e *MATLAB*.

2.1.1 Decomposição de elementos estruturantes

As seguintes propriedades das operações de Minkowski produz um método para a *decomposição de elemento estruturante* [WB88]:

$$\begin{aligned} \varepsilon_{A\oplus B}(X) &= \varepsilon_A(\varepsilon_B(X)) \quad e \\ \delta_{A\oplus B}(X) &= \delta_A(\delta_B(X)). \end{aligned}$$

Por exemplo, uma dilatação por um elemento estruturante 3×3 é equivalente a realizar duas dilatações unidimensionais 1×3 e 3×1. Outro exemplo é obter o elemento estruturante *octagonal* a partir de dois elementos estruturantes 3×3. Como ilustrados, respectivamente, nas duas somas de Minkowski da Figura 2.1.

Como conseqüência desta decomposição, seja a *soma de Minkowski generalizada* definida como segue:

$$B_G = B_1 \oplus \cdots \oplus B_k,$$

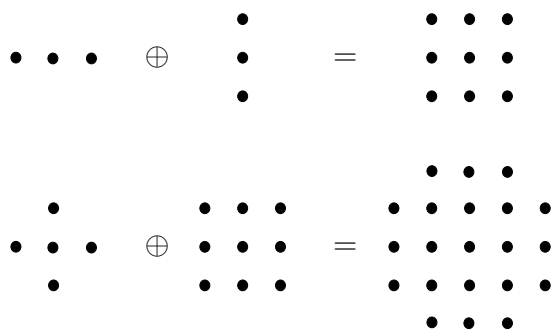


Figura 2.1: Exemplos de decomposições de elementos estruturantes.

onde $\{B_1, \cdots, B_k\}$ são elementos que decompõem B_G. Assim,

$$\varepsilon_{B_G}(f) = \varepsilon_{B_k}(\cdots(\varepsilon_{B_1}(f))\cdots) \quad e \tag{2.3}$$

$$\delta_{B_G}(f) = \delta_{B_k}(\cdots(\delta_{B_1}(f))\cdots). \tag{2.4}$$

Quando existe um único B que decompõe B_G, escrevemos,

$$B_G = \underbrace{B \oplus \cdots \oplus B}_{k \ vezes},$$

ou simplesmente $B_G = kB$.

É evidente que tais procedimentos de decomposição são úteis com respeito a implementações da erosão e da dilatação [Hei94], como serão apresentados nos próximos capítulos.

2.2 Tipos de elementos estruturantes

Observe que não foi definido o domínio da imagem nas Equações 2.1 e 2.2. Para tratar deste caso, serão definidas as *transformações infinitas*, onde o domínio da imagem e o domínio da transformação não são necessariamente os mesmos. Uma transformação muito utilizada na prática são

as *transformações limitadas*, onde a imagem e o resultado da transformação pertencem ao mesmo domínio $\mathbb{E}$.

Uma aplicação de *transformação infinita* são os operadores *invariantes por translação* (i.t.), isto é, ψ é invariante por translação se e somente se

$$\psi(B_x) = (\psi(B))_x,$$

onde $B_x = B + x = \{y + x,\ y \in B\}$ é a *translação* de B por $x \in \mathbb{E}$.

Uma *função estruturante* b será definida por $b \in \mathbb{Z}^B$. Assim, em morfologia matemática também é possível aplicar as Equações 2.1 e 2.2 em funções estruturantes em vez de elementos estruturantes. Porém, é comum usar *elemento estruturante* para transformações morfológicas em imagens binárias (ou conjuntos) e *função estruturante* para transformações morfológicas em imagens em níveis de cinza. Observe que um elemento estruturante pode ser visto como o domínio da função estruturante, pois $b \in \mathbb{Z}^B$, e as transformações morfológicas binárias podem ser vistas como um caso particular de transformações em níveis de cinza ($\{0,1\}^{\mathbb{E}} = K^{\mathbb{E}}$).

Algumas possíveis escolhas para funções estruturantes, considerando $x \in \mathbb{E}$ e $B \subseteq \mathbb{E} \oplus \mathbb{E}$, são [BB94]:

I. *transformação infinita*

$$b(x) = B_x;$$

II. *transformação limitada*

$$b(x) = B_x \cap \mathbb{E}.$$

Veja na Figura 2.2 um exemplo de dilatação binária limitada.

As funções estruturantes *quase invariantes por translação* (q.i.t.) são exemplos de transformações limitadas cujo domínio é $\mathbb{E}$. Mais detalhes de funções estruturantes i.t. e q.i.t. podem ser encontrados em [BB94].

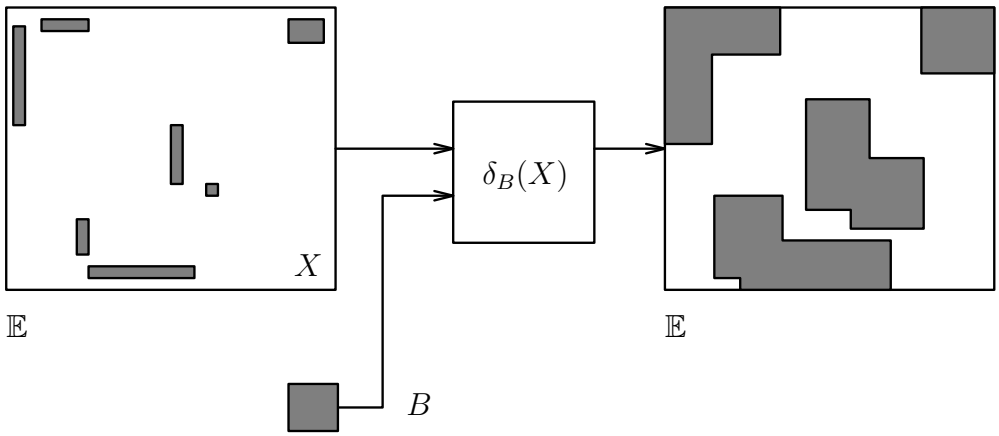


Figura 2.2: Dilatação binária limitada de X por B.

2.3 Erosão e dilatação em níveis de cinza

Nesta seção serão apresentadas a erosão e a dilatação no reticulado das imagens em níveis de cinza $K^{\mathbb{E}}$.

Seja $\mathbb{E}$ um retângulo finito de $\mathbb{Z} \times \mathbb{Z}$, onde $\mathbb{Z}$ é o conjunto dos números inteiros e $K \subset \mathbb{Z}$ é o conjunto finito dos níveis de cinza. Então $K^{\mathbb{E}}$ pode representar todas as possíveis *imagens em níveis de cinza*. A coleção $K^{\mathbb{E}}$ com a relação de ordem parcial $\leq$ dada por, $\forall f_1, f_2 \in K^{\mathbb{E}}$, $f_1 \leq f_2 \Leftrightarrow f_1(x) \leq f_2(x)$ e $\forall x \in \mathbb{E}$, é um reticulado completo. As operações de ínfimo e supremo são as operações de mínimo e máximo usuais.

Para tornar viável as implementações das transformações morfológicas em níveis de cinza, serão definidos os operadores $\dot{-}$ e $\dot{+}$ que limitam as transformações no intervalo K da seguinte forma [Hei91]:

Definição 2.1 *Seja K um intervalo $[0,k]$ de $\mathbb{Z}$ com $k>0$. Sejam v e t inteiros, então $t \to t\dot{-}v$ em K é definido por*

$$t\dot{-}v = \begin{cases} 0 & se\ t<k\ e\ t-v\leq 0, \\ t-v & se\ t<k\ e\ 0\leq t-v\leq k, \\ k & se\ t<k\ e\ t-v>k, \\ k & se\ t=k. \end{cases}$$

Analogamente, o operador $t \to t\dot{+}v$ em K é definido por

$$t\dot{+}v = \begin{cases} 0 & se\ t=0, \\ 0 & se\ t>0\ e\ t+v\leq 0, \\ t+v & se\ t>0\ e\ 0\leq t+v\leq k, \\ k & se\ t>0\ e\ t+v>k. \end{cases}$$

◇

Existem várias formas de escrever a erosão e a dilatação em níveis de cinza. Serão definidas a seguir a erosão e a dilatação por uma função estruturante que serão as mais usadas no decorrer do texto.

A erosão e a dilatação podem ser descritas, respectivamente, pelas expressões [Hei91]: $\forall f \in K^{\mathbb{E}}$, $\forall x \in \mathbb{E}$ e $b \in \mathbb{Z}^B$,

$$\varepsilon_b(f)(x) = \min\{f(y)\dot{-}b(y-x) : y \in B_x \cap \mathbb{E}\} \quad e \tag{2.5}$$

$$\delta_b(f)(x) = \max\{f(y)\dot{+}b(x-y) : y \in B_x \cap \mathbb{E}\}, \tag{2.6}$$

onde B_x é a *translação* de B por x. Veja na Figura 2.3 uma ilustração da dilatação em níveis de cinza.

2.4 Diferentes formas de escrever a erosão

Serão apresentados a seguir três exemplos diferentes de escrever a erosão morfológica. Estas erosões estão classificadas no padrão paralelo, descrito

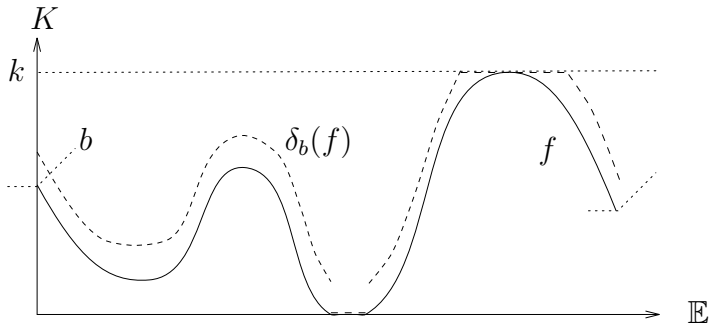


Figura 2.3: Dilatação de uma imagem em níveis de cinza f por uma função estruturante b.

com detalhes no próximo capítulo.

Exemplo 1

Na Equação 2.5, é apresentada a primeira forma de escrever a erosão morfológica aplicada em imagens em níveis de cinza. Esta definição é a mais conhecida na literatura e é ilustrada na Figura 2.4. O valor da erosão no ponto x é o mínimo de $f(y)\dot{-}b(y-x)$ na janela dada pelo elemento estruturante centrado em x.

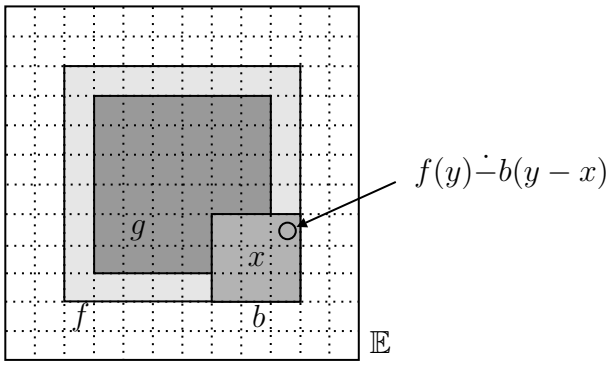


Figura 2.4: Primeiro exemplo de erosão.

Exemplo 2

Outra forma de escrever a erosão de uma imagem f por uma função estruturante b é apresentada na equação a seguir e ilustrada na Figura 2.5: $\forall f \in K^{\mathbb{E}}$ e $b \in \mathbb{Z}^B$,

$$\varepsilon_b(f) \quad = \quad \min\{(f_y \cap \mathbb{E}) \dot{-} b(y) : y \in B\}. \tag{2.7}$$

A principal diferença deste exemplo para o exemplo anterior é relativa ao parâmetro f_y, que tem como retorno uma imagem transladada por y.

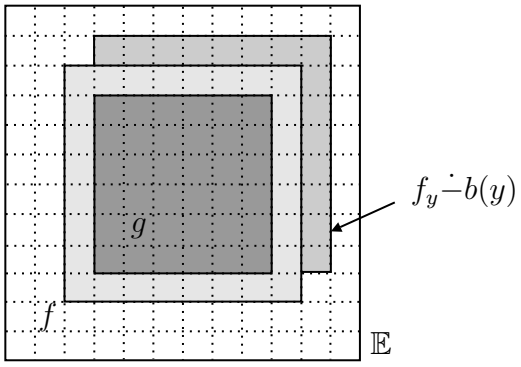


Figura 2.5: Segundo exemplo de erosão.

Exemplo 3

Neste último exemplo, a erosão de cada pixel x da imagem g é obtida fazendo a redução através da operação de *mínimo* de todas as sub-imagens de f subtraídas de b (veja também Figura 2.6): $\forall f \in K^{\mathbb{E}}$ e $b \in \mathbb{Z}^B$, $\forall x \in \mathbb{E}$

$$\varepsilon_b(f)(x) \quad = \quad \min\{(f(x)_{y:\ \forall y \in B} \cap \mathbb{E}) \dot{-} b\}. \tag{2.8}$$

Observe neste exemplo que $f' = f(x)_{y,\ \forall y \in B} \cap \mathbb{E}$ retorna uma sub-imagem de f com centro no pixel x tendo o mesmo domínio de b.

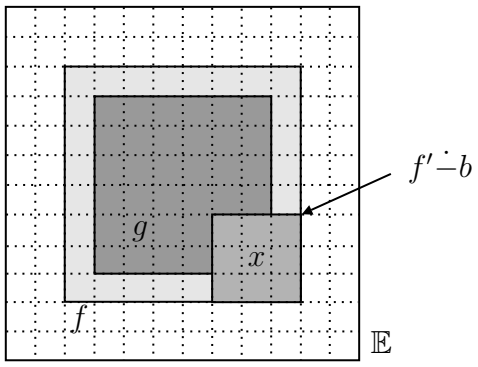


Figura 2.6: Terceiro exemplo de erosão.

2.5 Notação

Será apresentada nesta seção a notação utilizada para descrever os algoritmos neste documento a partir do próximo capítulo. Esta notação foi inspirada na *notação Z* e foi usada para gerar código de forma automática, como descrito no Apêndice A. Z é utilizada para a especificação de problemas em engenharia de software através de expressões matemáticas [Pre02].

Para declarar uma variável pertencente a um tipo será utilizado o formato:

$nome\_variavel : tipo$;

onde *nome_variavel* pode ser letras ou números. Outra possibilidade é fazer

$nome\_variavel \in tipo$;

Para declarar variáveis do mesmo tipo seja:

$nome\_variavel\_1, \ldots, nome\_variavel\_n : tipo$; // declarações

onde n é um número natural e os comentários seguem "//". Para declarar uma imagem f seja:

$f : \mathbb{E} \longrightarrow K$ *ou* $f \in K^{\mathbb{E}}$.

Para definir uma transformação será utilizada a notação apresentada no Algoritmo 1. Onde s define um símbolo para representar o algoritmo; D_i representa o domínio da transformação, com entrada $e_i \in D_i,\ i = 1, \ldots, n$; I_i representa o retorno, com $s_i \in I_i,\ i = 1, \ldots, m$, onde m e n são inteiros positivos; e *expressões* são as expressões matemáticas implementadas.

Algoritmo 1 *Especificação de transformação*

$s : D_1 \times \cdots \times D_n \longrightarrow I_1 \times \cdots \times I_m$
$\quad s(e_1, \ldots, e_n) = (s_1, \ldots, s_m)$

expressões;

Por exemplo, o Algoritmo 2 apresenta a soma de duas imagens com varreduras explícitas, isto é, pixel-por-pixel. Esta operação é a soma usual sem a preocupação com o tratamento de saturação.

Algoritmo 2 Soma duas imagens

$\_ + \_ : K^{\mathbb{E}} \times K^{\mathbb{E}} \longrightarrow \mathbb{Z}^{\mathbb{E}}$
$\qquad +(f_1, f_2) = f_1 + f_2$

$\forall x \in \mathbb{E}$
$\quad (f_1 + f_2)(x) = f_1(x) + f_2(x)$;

Outra forma de fazer um algoritmo para somar duas imagens com saturação em $K = [0, 255]$ é definida no Algoritmo 3, onde $\wedge$ calcula o mínimo entre $f1(x) + f2(x)$ e 255.

As notações deste documento para representar as transformações de morfologia matemática são as usadas na literatura e estão descritas na tabela de símbolos, no início deste texto. Nesta tabela também estão definidos símbolos próprios para representar algoritmos como a *erosão por propagação*, ou ε^p. Além disso, quando uma função estruturante b é aplicada em uma imagem f é comum utilizar $\varepsilon_b^p(f)$. Para deixar clara a seqüência de argumentos deste

Algoritmo 3 Soma duas imagens com saturação

$\_\bar{+}\_ : K^{\mathbb{E}} \times K^{\mathbb{E}} \longrightarrow \mathbb{Z}^{\mathbb{E}}$
$\quad \bar{+}(f_1, f_2) = f_1 \bar{+} f_2$

$\forall x \in \mathbb{E}$
$\quad (f_1 \bar{+} f_2)(x) = (f_1(x) + f_2(x)) \wedge 255;$

operador também utilizamos $\varepsilon^p(f, b)$. Por outro lado, quando um algoritmo com uma grande quantidade de argumentos junto ao símbolo é especificado, usamos simplesmente uma letra para representar este operador. Por exemplo, em vez de usar $f_1 \bar{+} f_2$ no Algoritmo 3 pode ser usado simplesmente g.

Referências Bibliográficas

[BB92] J. Barrera e G.J.F. Banon. Expressiveness of the morphological language. No *Image Algebra and Morphological Image Processing III*, páginas 264–274, San Diego, California, 1992. SPIE.

[BB93] G.J.F. Banon e J. Barrera. Decomposition of mappings between complete lattices by mathematical morphology, Part I: general lattices. *Signal Processing*, 30:299–327, 1993.

[BB94] G.J.F. Banon e J. Barrera. *Bases da morfologia matemática para analise de imagens binárias*. IX Escola de Computação, Recife, Brasil, 1994.

[BBL94] J. Barrera, G.F. Banon e R.A. Lotufo. A mathematical morphology toolbox for the KHOROS system. No *Image Algebra and Morphological Image Processing V*, páginas 241–252, Bellingham, Julho 1994. SPIE.

[Bir67] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, 1967.

[Hei91] H.J.A.M. Heijmans. Theoretical aspects of gray-level morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):568–581, Junho 1991.

[Hei94] H.J.A.M. Heijmans. *Morphological Image Operators*. Academic Press, Boston, 1994.

[Pre02] R.S. Pressman. *Engenharia de Software*. McGraw-Hill, Rio de Janeiro, 5ed. edition, 2002.

[Ser88] J. Serra, editor. *Image Analysis and Mathematical Morphology - Volume II: Theoretical Advances*. Academic Press, London, 1988.

[Vin92] L. Vincent. Morphological algorithms. No E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, capítulo 8, páginas 255–288. Marcel Dekker, New York, 1992.

[WB88] X. Wang e G. Bertrand. An algorithm for a generalized distance transformation based on Minkowski operations. *9th ICPR*, páginas 1163–1167, 1988.

[Zam97] F.A. Zampirolli. Operadores morfológicos baseados em grafos de vizinhanças – uma extensão da MMach toolbox. Dissertação de Mestrado, Unversidade de São Paulo, São Paulo, Brasil, Abril 1997.

Capítulo 3

Padrões de Algoritmos da Erosão

No capítulo anterior foi apresentada a teoria de operadores elementares caracterizados por funções estruturantes. Como exemplo, foram definidos operadores no reticulado das imagens binárias e no reticulado das imagens em níveis de cinza. Uma importante propriedade desses operadores é a decomposição de função estruturante.

Dando continuidade aos estudos de operadores morfológicos, será apresentada neste capítulo a erosão caracterizada por funções estruturantes focando as implementações. Assim, serão usados três padrões de algoritmos: paralelo, seqüencial e por propagação.

Existem resultados na morfologia matemática para a decomposições ótimas de funções estruturantes [HAS00]. Porém, serão apresentadas nesta tese apenas algumas decomposições para a viabilidade da transformada de distância usando erosões.

No próximo capítulo serão classificados os principais algoritmos da transformada de distância usando as erosões paralela, seqüencial e por propagação definidas a seguir.

3.1 Padrão paralelo

Será apresentado nesta seção o padrão paralelo, onde a ordem de varredura na imagem não influencia no resultado obtido.

3.1.1 Erosão paralela

Será apresentada a seguir a caracterização da erosão morfológica por função estruturante no reticulado $K^{\mathbb{E}}$ das imagens em níveis de cinza.

A erosão por uma função estruturante, definida na Equação 2.5, pode ser reescrita pelo Algoritmo 4.

Algoritmo 4 Erosão paralela

$\varepsilon : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}}$
$\quad \varepsilon(f, b) = \varepsilon_b(f)$

$\forall x \in \mathbb{E}$ // em paralelo
$\quad \varepsilon_b(f)(x) = \bigwedge_{\forall y \in B_x \cap \mathbb{E}} \{f(y) \dot{-} b(y - x)\};$

Observe que este algoritmo é genérico, isto é, independe da dimensão e do tipo da imagem.

A notação utilizada para representar os algoritmos neste documento é a descrita na Seção 2.5. Esta notação é semelhante à notação Z e é possível gerar código a partir destes algoritmos. Isso será descrito com detalhes no Apêndice A – *Ambiente mmil*.

3.1.2 Decomposição paralela de função estruturante

A aplicação direta do Algoritmo 4 por uma função estruturante "grande"[1] b_G, apresentado na Subseção 3.1.1, é ineficiente em máquinas seqüenciais. Uma solução é a *decomposição de função estruturante* grande em funções estruturantes "pequenas", como apresentada mais adiante nesta seção. Para justificar esta afirmação, considere as próximas definições.

Sejam $B_i \subseteq \mathbb{E} \oplus \mathbb{E}$ contendo a origem e $b_i \in \mathbb{Z}^{B_i}$, onde $i = 1, \ldots, k$. Então a *soma de Minkowski em níveis de cinza* de b_i por b_j [Ser82, SM92] é definida como [2]: $\forall x \in B_i \oplus B_j$,

$$(b_i \oplus b_j)(x) = \max\{b_i(y) + b_j(x - y) : \ y \in (B_j + x)\}, \tag{3.1}$$

onde $j = 1, \ldots, k$ e $B_i \oplus B_j$ é a *soma de Minkowski* em conjunto[3] (imagens binárias), veja Figura 3.1.

Como conseqüência, é possível fazer a *soma de Minkowski generalizada em níveis de cinza* como segue:

$$b_G = b_1 \oplus \cdots \oplus b_k, \tag{3.2}$$

onde $b_1, \cdots, b_k$ são funções que decompõem b_G. Quando existe um b tal que,

$$b_G = \underbrace{b \oplus \cdots \oplus b}_{k \ vezes}, \tag{3.3}$$

[1] Como apresenado no Seção 1.2, dizemos função estruturante "grande" (por exemplo, com o tamanho do domínio equivalente a duas vezes o da imagem a ser processada) e função estruturante "pequena" (por exemplo, 3×3, 1×3 ou 1×2) que decompõe a função estruturante "grande".

[2] Observe que nesta parte B_i não é a translação de B por i, mas apenas a notação usual de um conjunto qualquer.

[3] Note que é usado o mesmo símbolo para soma de Minkowski para imagens binárias (representada por letras maiúsculas) e para imagens em níveis de cinza (representadas por letras minúsculas).

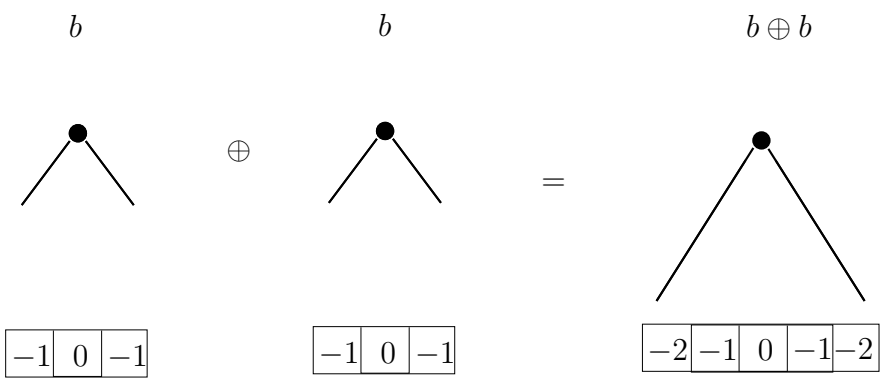


Figura 3.1: Soma de Minkowski em níveis de cinza, onde as origens estão no centro de cada função estruturante.

é escrito simplesmente $b_G = kb$.

Uma propriedade da erosão é [Ser82, SM92]:

$$\varepsilon_{b_G}(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots). \tag{3.4}$$

Note que é mais eficiente trabalhar com a decomposição de b_G em erosões paralelas. Isto também se aplica para os padrões seqüenciais e por propagação, como serão apresentados nas próximas seções. Por exemplo, seja $b_G = kb_i$ de dimensão 3×3 e f de dimensão $n \times n$. Então $\varepsilon_{b_G}(f)$ requer

$$(2k+1)(2k+1)n^2 = (2k+1)^2n^2$$

acessos à memória, enquanto $\varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots)$ requer $9kn^2$ acessos.

A Figura 3.2 ilustra um exemplo de aplicação da erosão paralela de uma imagem f por uma função estruturante b.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

f

$$\begin{matrix} -1 & -1 & -1 \\ -1 & \mathbf{0} & -1 \\ -1 & -1 & -1 \end{matrix}$$

b

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 5 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$\varepsilon_b(f)$

Figura 3.2: Ilustração da erosão paralela $\varepsilon_b(f)$ da imagem de entrada f pela função estruturante b, com origem no centro.

3.2 Padrão seqüencial

Será apresentado nesta seção o *padrão seqüencial*. A principal característica deste padrão é a sua eficiência pois basta varrer uma imagem duas vezes para conseguir bons resultados. Este tipo de algoritmo é conhecido na literatura desde a década de 60 [RP66, Dia69, Dan80, YTF81, WB88, WB92, WHCR95, BHJ97, D'O01]. Como foi visto na seção anterior, nos algoritmos paralelos os pixels são processados independentemente da ordem de varredura da imagem, dependendo apenas dos valores dos pixels da imagem de entrada e da vizinhança usada. Nos algoritmos seqüenciais a imagem de saída depende não somente dos pixels da imagem de entrada, mas também dos valores calculados anteriormente e da ordem de varredura na imagem.

3.2.1 Erosão seqüencial

Será apresentada nesta subseção a erosão morfológica caracterizada por funções estruturantes no reticulado $K^{\mathbb{E}}$ das imagens em níveis de cinza. Os operadores que serão apresentados são diferentes daqueles vistos até agora no que diz respeito à ordem de varredura da imagem e na forma de decompor a função estruturante.

Algoritmos seqüenciais podem ser classificados conforme a ordem de varredura dos pixels acessados de uma imagem. As duas ordens mais comuns são as *raster* e *anti-raster*, como exemplo, veja Figuras 3.3a e 3.3b, respectivamente[4].

Considere uma imagem f com domínio $\mathbb{E} \subset \mathbb{Z} \times \mathbb{Z}$ de dimensões $m \times n = |\mathbb{E}|$, onde m é a *largura* ou o número de colunas da imagem e n é a *altura* ou o número de linhas da imagem. Um *pixel* em $\mathbb{E}$ é denotado pelo par ordenado

[4]D'Ornellas chamou esta varredura *raster* e *anti-raster* e varredura horizontal, definindo também a varredura vertical [D'O01].

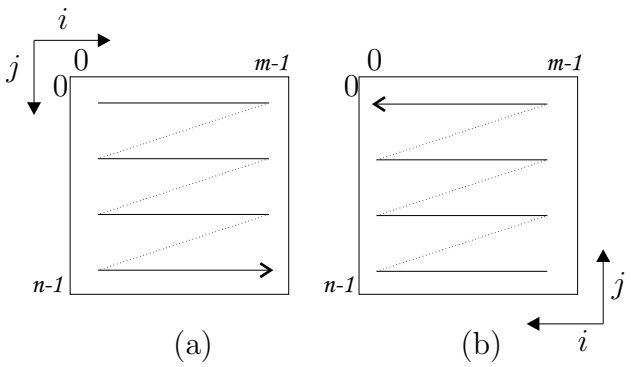


Figura 3.3: Ordem de varredura em uma imagem de duas dimensões: (a) *raster* horizontal e (b) *anti-raster* horizontal.

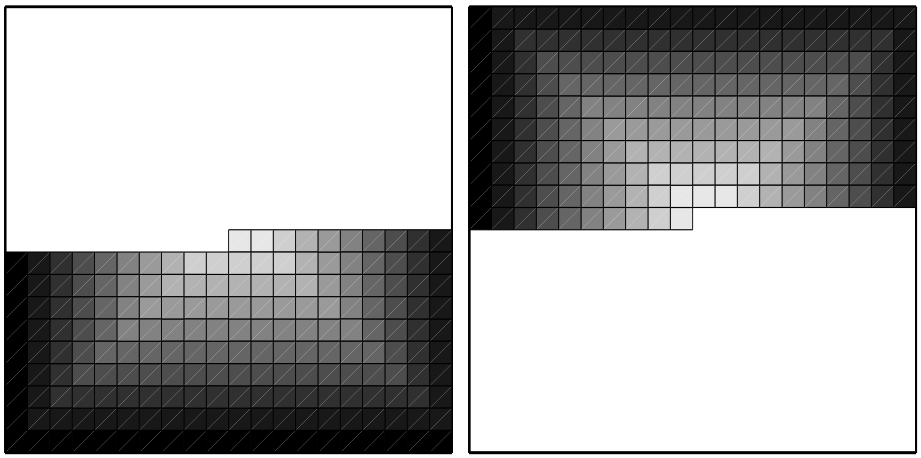
(i, j), onde $0 \leq i \leq m-1$ e $0 \leq j \leq n-1$.

No caso de uma varredura horizontal em uma imagem f, a ordem *raster* é definida como uma visita aos pixels de f da esquerda para a direita e de cima para baixo, isto é, $\{(0,0),(0,1),\cdots,(0,n-1),\ (1,0),(1,1),\cdots,(1,n-1),\ \cdots,(m-1,0),(m-1,1),\cdots,(m-1,n-1)\}$, veja Figura 3.3a. Esta seqüencia é definida como $S^+ = \{0,\cdots,mn-1\}$. A varredura na ordem *anti-raster* em f é definida na direção inversa, da direita para a esquerda e de baixo para cima, isto é, $\{(m-1,n-1),(m-1,n-2),\cdots,(m-1,0),\cdots,(0,n-1),(0,n-2),\cdots,(0,0)\}$. Esta seqüencia é definida como $S^- = \{mn-1,\cdots,0\}$, veja Figura 3.3b.

Existe uma bijeção $\ell^+ : \mathbb{E} \rightarrow S^+$, definida por $\ell^+(i,j) = in + j$, onde $(i,j) \in \mathbb{E}$ e n é a largura da imagem. Uma função inversa de ℓ^+ é dada por $(\ell^+)^{-1}(r) = (\lfloor r/n \rfloor, r\ rem\ n)$, onde $r \in \{1,2,\ldots,|S^+|\}$, $\lfloor r/n \rfloor$ é a divisão inteira e $r\ rem\ n$ é o resto da divisão de r por n, respectivamente.

Dado um elemento estruturante $B \subseteq \mathbb{E} \oplus \mathbb{E}$ com origem, seja B^+ uma vizinhança para a ordem *raster* e seja B^- uma vizinhança para a ordem *anti-raster*. Considerando o centro de B como o centro dos eixos de coordenadas cartesianas $(i,j) \in \mathbb{E}$, onde i é a abscissa (posição para direita) e j é a ordenada (posição para baixo), como ilustrados na Figura 3.3a, seja: $B^+ =$

$\{(i,j) \in B \mid j > 0$ ou se $j = 0$ então $i \geq 0\}$. Similarmente $B^- = \{(i,j) \in B \mid j < 0$ ou se $j = 0$ então $i \leq 0\}$. Note que $B^+ \cup B^- = B$. Similarmente, igual decomposição pode ser aplicada para uma função estruturante $b \in \mathbb{Z}^B$, isto é, $b^+ \in \mathbb{Z}^{B^+}$ e $b^- \in \mathbb{Z}^{B^-}$. Um exemplo desta decomposição é mostrada na Figura 3.4. Isto significa que qualquer função estruturante com domínio em $\mathbb{E} \oplus \mathbb{E}$ com origem pode ser decomposta em funções estruturantes *raster* e *anti-raster*.



b_G^+ b_G^-

Figura 3.4: Exemplos de funções estruturantes utilizadas nas varreduras *raster* e *anti-raster*.

Seja $\varepsilon_{b^+}^+(f)$ a *erosão seqüencial na ordem raster*, como definida na Equação 2.5, porém colocando os resultados parciais na própria função f. De forma mais precisa: para $x \in \mathbb{E}$ na ordem *raster*,

$$\varepsilon_{b^+}^+(f)(x) = \min\{\varepsilon_{b^+}^+(f)(y) \dot{-} b(y-x) : y \in B_x^+ \cap \mathbb{E}\}. \tag{3.5}$$

Analogamente, seja $\varepsilon_{b^-}^-(f)(x)$ a *erosão seqüencial na ordem raster* de f, definida como: para $x \in \mathbb{E}$ na ordem *anti-raster*,

$$\varepsilon_{b^-}^-(f)(x) = \min\{\varepsilon_{b^-}^-(f)(y) \dot{-} b(y-x) : y \in B_x^- \cap \mathbb{E}\}. \tag{3.6}$$

Baseados nas Equações 3.5 e 3.6 acima, serão apresentados os algoritmos da *erosão seqüencial na ordem raster* e da *erosão seqüencial na ordem anti-raster*, Algoritmos 5 e 6, respectivamente. Nestes algoritmos os conjuntos *seq* S^+ e *seq* S^- representam as varreduras *raster* e *anti-raster*, respectivamente. De forma análoga é definida a dilatação seqüencial e por este motivo não será apresentada aqui.

Algoritmo 5 Erosão seqüencial na ordem *raster*

$\varepsilon^+ : K^{\mathbb{E}} \times K^{B^+} \longrightarrow K^{\mathbb{E}}$
$\quad \varepsilon^+(f, b^+) = \varepsilon^+_{b^+}(f)$

$\varepsilon^+_{b^+}(f) = f;$
$\forall x \in seq\ S^+$ // seqüencial *raster*
$\quad \varepsilon^+_{b^+}(f)(x) = \bigwedge_{\forall y \in B^+_x \cap \mathbb{E}} \{\varepsilon^+_b(f)(y) \dot{-} b^+(y - x)\};$

A Figura 3.5 ilustra uma iteração intermediária da erosão seqüencial na ordem *raster*, $\varepsilon^+_{b^+}(f)(x)$, da imagem de entrada f pela função estruturante b^+. O pixel x, que está sendo processado, é o em negrito e está recebendo o valor $\mathbf{1} = \bigwedge\{0\dot{-}(-1), 1\dot{-}(-1), 2\dot{-}(-1), 0\dot{-}(-1), 5\dot{-}(0)\} = \varepsilon^+_{b^+}(f)(x)$.

Algoritmo 6 Erosão seqüencial na ordem *anti-raster*

$\varepsilon^- : K^{\mathbb{E}} \times K^{B^-} \longrightarrow K^{\mathbb{E}}$
$\quad \varepsilon^-(f, b^-) = \varepsilon^-_{b^-}(f)$

$\varepsilon^-_{b^-}(f) = f;$
$\forall x \in seq\ S^-$ // seqüencial *anti-raster*
$\quad \varepsilon^-_{b^-}(f)(x) = \bigwedge_{\forall y \in B^-_x \cap \mathbb{E}} \{\varepsilon^-_b(f)(y) \dot{-} b^-(y - x)\};$

$$
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 5 & 5 & 5 & 0 & 0 \\
0 & 0 & 5 & 5 & 5 & 0 & 0 \\
0 & 0 & 5 & 5 & 5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

f

$$
\begin{array}{ccc}
 & \mathbf{0} & -1 \\
-1 & -1 & -1 \\
 & b^+ &
\end{array}
$$

$$
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 2 & 3 & 0 & 0 \\
0 & 0 & \mathbf{1} & 5 & 5 & 0 & 0 \\
0 & 0 & 5 & 5 & 5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

$\varepsilon^+_{b^+}(f)$

Figura 3.5: Ilustração de uma iteração intermediária da erosão seqüencial *raster*, $\varepsilon^+_{b^+}(f)(x)$, da imagem de entrada f pela função estruturante b^+, com origem em negrito. O pixel x, que está em negrito com valor $\mathbf{1}$, é o que está sendo processado.

3.2.2 Decomposição seqüencial de função estruturante

De forma análoga ao ocorrido nos algoritmos paralelos, a aplicação direta da erosão por uma função estruturante grande b_G é ineficiente. Contudo, é possível fazer a *decomposição seqüencial da função estruturante* b_G para obter implementações rápidas, como apresentado a seguir.

Um resultado de Rosenfeld e Pfaltz [RP66] é apresentado no teorema a seguir:

Teorema 3.1 *Aplicar um operador seqüencial com uma vizinhança local em uma imagem f é equivalente a aplicar uma seqüência de transformações paralelas com igual vizinhança local em f.* ■

A prova deste resultado pode ser vista no apêndice do trabalho [RP66] e algumas aplicações serão apresentadas a seguir, mais especificamente, serão apresentados casos onde se obtém a equivalência entre as erosões paralelas e seqüenciais.

Primeiro caso: $\varepsilon_{b_G}(f) = \varepsilon^-_{b^-}(f) \wedge \varepsilon^+_{b^+}(f)$

Seja b uma função estruturante qualquer contendo a origem b_o com o valor zero. Para o caso de b_o ser diferente de zero, basta subtrair b de b_o ($b = b - b_o$) e nos dois lados da igualdade acima calcular a erosão por b_o:

$$\varepsilon_{b_o}(\varepsilon_{b_G}(f)) = \varepsilon_{b_o}(\varepsilon^-_{b^-}(f) \wedge \varepsilon^+_{b^+}(f)).$$

Seja $b = b^+ \vee b^-$ a decomposição de b nas ordens *raster* b^+ e *anti-raster* b^-. Observe que as origens de b^+ e b^- também têm valores zeros.

Seja uma *função estruturante estável* uma função que não muda dentro de uma janela finita:

$$kb = (k+1)b, \tag{3.7}$$

por exemplo, dentro da janela $k \times k$.

Sejam b^+ e b^- funções estruturantes estáveis, $f \in K^{\mathbb{E}}$ e

$$b_G = k(b^+ \vee b^-) = kb^+ \vee kb^-, \tag{3.8}$$

então,

$$\varepsilon_{b_G}(f) = \varepsilon_{kb^+ \vee kb^-}(f) = \varepsilon_{kb^+}(f) \wedge \varepsilon_{kb^-}(f) = \varepsilon^-_{b^-}(f) \wedge \varepsilon^+_{b^+}(f). \tag{3.9}$$

Assim, a erosão paralela $\varepsilon_{b_G}(f)$ pode ser obtida pelos algoritmos *raster* e *anti-raster* $\varepsilon^-_{b^-}(f)$ e $\varepsilon^+_{b^+}(f)$, para uma função estruturante qualquer. A Figura 3.6 ilustra esta equivalência para uma função estruturante unidimensional. Na figura são ilustradas a imagem de entrada f de tamanho 1×10; a função estruturantes b de tamanho 1×5 com origem no centro; a erosão *raster* $\varepsilon^+_{b^+}(f)$ e b^+; a erosão *anti-raster* $\varepsilon^-_{b^-}(f)$ e b^-; e a equivalência $\varepsilon_{kb^+ \vee kb^-}(f) = \varepsilon^+_{b^+}(f) \wedge \varepsilon^-_{b^-}(f)$, com $b_G = kb^+ \vee kb^-$. Foi utilizado neste exemplo $k = 10$.

A desvantagem desta equivalência está nas funções estruturantes bidimensionais, onde ocorre o inconveniente ilustrado na Figura 3.7. Observe que na união $b_G = 2b^- \vee 2b^+$ existem dois espaços em branco, que possuem valores $-\infty$. Este problema se propaga com o aumento das somas de Minkowski. Assim, não é possível calcular a TD utilizando esta equivalência.

Outra ilustração da equivalência para o caso bidimensional é apresentada na Figura 3.8. Na figura são ilustradas a imagem de entrada f de tamanho 10×10, com os níveis de cinza gerados de forma aleatória[5]; a função estruturantes b de tamanho 3×3, também com níveis de cinza gerados de forma aleatória e com origem no centro; a erosão *raster* $\varepsilon^+_{b^+}(f)$; a erosão *anti-raster* $\varepsilon^-_{b^-}(f)$; e a equivalência $\varepsilon_{kb^+ \vee kb^-}(f) = \varepsilon^+_{b^+}(f) \wedge \varepsilon^-_{b^-}(f)$, com $b_G = kb^+ \vee kb^-$. Foi utilizado neste exemplo $k = 27$.

[5]Observe que à direita de cada imagem possui uma escala dos níveis de cinza existentes em cada imagem.

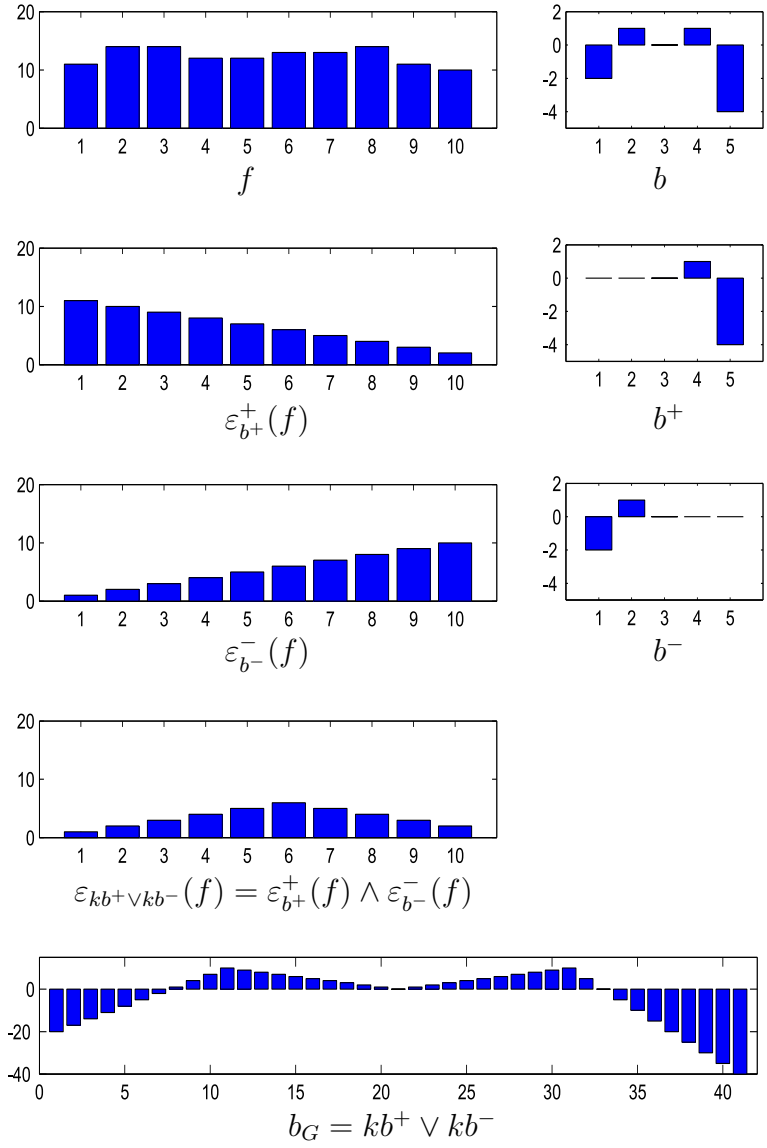


Figura 3.6: Ilustração da equivalência entre algoritmos paralelos e seqüenciais para funções estruturantes unidimensionais.

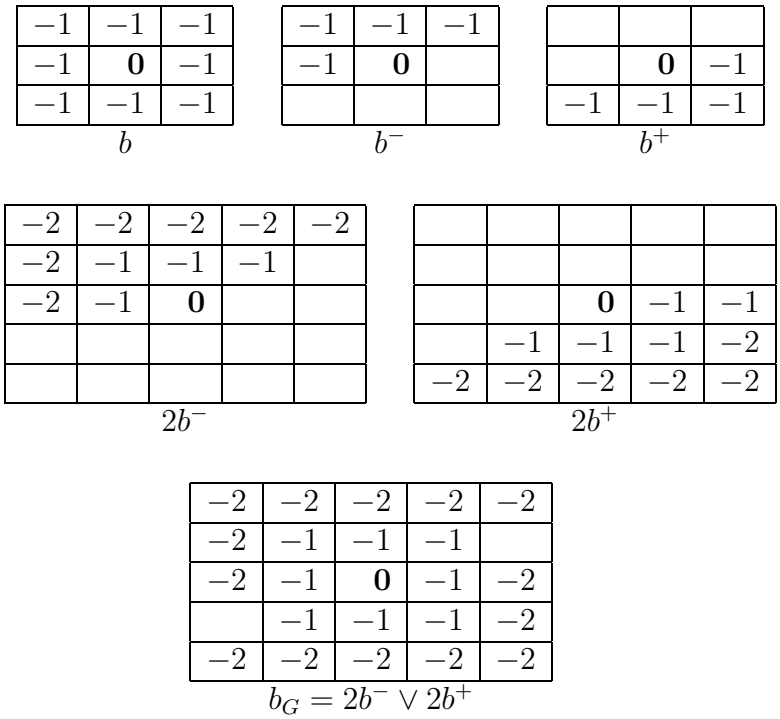


Figura 3.7: Ilustração da patologia ocorrida no primeiro caso da equivalência das erosões paralelas e seqüenciais usando $b_G = 2b^- \vee 2b^+$. Os espaços assumem $-\infty$.

Como sugestões para pesquisas futuras, seria interessante estudar a relação entre k e as dimensões e níveis de cinza de f e b, e não simplesmente assumir k suficientemente grande ou tendendo para o infinito para obter a equivalência entre erosões paralelos e seqüenciais[6].

Segundo caso: $\varepsilon_{b_G}(f) = \varepsilon^-_{b^-}(\varepsilon^+_{b^+}(f))$

Neste segundo caso, só existe uma função estruturante paralela equivalente para *funções estruturantes estáveis* (que não mudam dentro de uma janela finita). Quando isto ocorre, também vale a igualdade:

$$b_G = kb = kb^+ \oplus kb^-. \tag{3.10}$$

Como estudo de casos de funções estruturantes estáveis, veja a seguinte definição.

Seja $2p < q < p \leq 0$ conforme a função estruturante b mostrada na Figura 3.9 [Bor86], então

$$\varepsilon^+_{b^+}(f) = \varepsilon_{kb^+}(f) \quad e \quad \varepsilon^-_{b^-}(f) = \varepsilon_{kb^-}(f).$$

Portanto, é possível obter um operador paralelo equivalente a dois operadores seqüenciais. Por exemplo, seja k a maior distância possível numa imagem $f \in [0,k]^{\mathbb{E}}$. Se $b_G = kb$, então

$$\varepsilon_{kb}(f) = \varepsilon_{k(b^+ \oplus b^-)}(f) = \varepsilon_{kb^+ \oplus kb^-}(f) = \varepsilon_{kb^-}(\varepsilon_{kb^+}(f)) = \varepsilon^-_{b^-}(\varepsilon^+_{b^+}(f)). \tag{3.11}$$

Assim, a erosão paralela $\varepsilon_{b_G}(f)$ pode ser obtida pelos algoritmos *raster* e *anti-raster* $\varepsilon^-_{b^-}(\varepsilon^+_{b^+}(f))$, para uma função estruturante definida pela Figura 3.9.

[6]Um trabalho interessante que estuda as condições para que dilatações e erosões em espaço-escala morfológico sejam idempotentes foi realizado por Leite e Teixeira [LT00]. Neste trabalho também existe um resultado para o número de iterações necessárias, entre escalas, para que a idempotência seja calcula. Este resultado seria o ponto de partida para determinar a relação entre k e as dimensões e níveis de cinza de f e b.

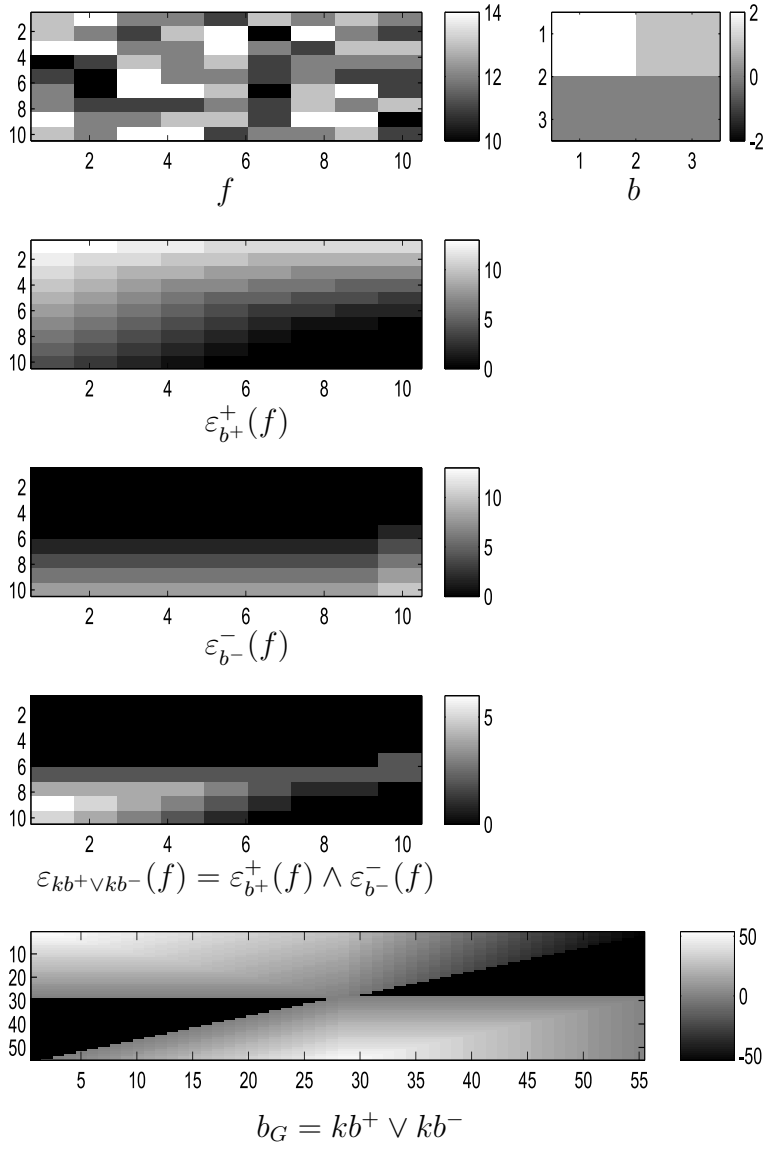


Figura 3.8: Ilustração da equivalência entre algoritmos paralelos e seqüenciais para funções estruturantes bidimensionais.

q	p	q
p	**0**	p
q	p	q

b

Figura 3.9: Função estruturante b para a Equação 3.11, onde $2p < q < p \leq 0$.

As funções estruturantes estáveis são apropriadas para calcular a TD e serão estudas com detalhes no próximo capítulo.

3.3 Padrão por propagação

Algoritmo por *propagação* é um outro padrão muito estudado devido à boa eficiência dos algoritmos [VV88, Vin92, Vin93, BHJ97, FLA01, D'O01]. Como introduzida na Seção 1.2, a idéia deste padrão é processar apenas alguns pixels, inseridos em uma estrutura de dados, como *fila*, *conjunto* ou *fila hierárquica*. Com isso, o algoritmo coloca apenas os pixels que potencialmente influenciam no resultado da operação nesta estrutura. Um trabalho interessante de Vliet e Verwer [VV88] descreve o padrão por propagação para erosões, dilatações e esqueletos, porém não possui o formalismo usado em morfologia matemática.

O padrão por propagação é útil apenas em casos de operações sucessivas, onde uma função estruturante b_G pode ser decomposta como $b_G = b_1 \oplus \cdots \oplus b_k$ e $b_1 \geq \cdots \geq b_k$. Quando isto ocorre, é equivalente fazer uma operação usando b_G ou uma seqüência de operações usando os $b_i's$.

Este *padrão por propagação* possui características dos padrões paralelo e seqüencial, apresentados anteriormente. Do *padrão paralelo*, possui a característica de poder ser implementado numa arquitetura paralela para os pixels de propagação. Para o *padrão seqüencial*, em vez de possuir varreduras *raster* e *anti-raster*, o padrão por propagação é formado por uma seqüência

de padrões paralelos.

Como exemplo, é possível computar a TD usando sucessivas erosões por propagação com as funções estruturantes decompostas. Para este caso particular, após a criação da fronteira, o algoritmo entra num laço que só termina quando o mesmo estiver vazio. Dentro deste laço, um pixel por vez é retirado da fronteira para analisar seus pixels vizinhos e decidir quais destes vizinhos irão entrar numa nova fronteira. Este processo se repete até não existir nenhum conjunto fronteira. Os detalhes deste processo são apresentados no próximo capítulo.

Vincent [Vin92] abordou o problema de varredura por propagação de forma diferente, definindo os algoritmos *baseados em contorno* e subdividindo-os em duas famílias: os algoritmos *por propagação chains and loops* e os algoritmos *por propagação baseados em fila* [Vin92].

Nos algoritmos por propagação *chains and loops*, propostos em 1989 por Schmitt [Sch89], os vetores elementares que contornam a fronteira de uma imagem binária são armazenados. As transformações ocorrem usando esses vetores através de regras de propagação. Essas regras são determinadas por ângulos formados pelos pares de vetores elementares adjacentes. Por exemplo, para a grade hexagonal existem seis regras para o operador de dilatação [Vin92]. Portanto, não é necessário percorrer toda a imagem para saber quais pixels propagar, mas sim percorrer a fronteira dos objetos da imagem e verificar nas regras quais pixels são propagados.

Esse processo tem o mesmo princípio ocorrido no *padrão por propagação* usado neste documento, ou seja, percorrer apenas a fronteira dos objetos. Porém, a diferença entre estes dois padrões é que no padrão por propagação percorrem-se os pixels que contornam os objetos, colocando-os em uma estrutura de dados e entrando em um laço que analisa os vizinhos destes pixels. Estes vizinhos são determinados pela função estruturante. No padrão *chain and loop* não existe a noção de vizinhança de um pixel definida por uma função estruturante, o que dificulta ou impossibilita as transformações

genéricas ocorridas. Por exemplo, o número de regras para cada operação cresce consideravelmente com o aumento da vizinhança.

Na família de *algoritmos por propagação baseados em fila*, definida por Vincent [Vin92], os pixels de fronteira dos objetos em imagens são armazenados em uma estrutura de dados de *fila FIFO – First-In-First-Out*. Esta estrutura possui os métodos de incluir um pixel no final da fila, remover um pixel do início da fila e verificar se a fila está vazia. Esta família de algoritmos baseados em fila também foi usada por D'Ornellas [D'O01].

O uso de filas *FIFO* é uma otimização da varredura por propagação para casos particulares onde é possível trabalhar com a mesma imagem de entrada e saída e com operações idempotentes. Estes pontos serão detalhados na Subseção 4.3.3.

As maiores contribuições desta tese, no que diz respeito ao estudo de padrões de algoritmos, foram alcançadas para o *padrão por propagação*, onde foram obtidas implementações simples e eficientes de algoritmos da TD reportados na literatura. Além disso, estas implementações foram construídas através dos operadores elementares de morfologia matemática usando apenas um conjunto para armazenar a fronteira, como serão apresentadas no próximo capítulo. No final dessa seção é apresentado a condição para o uso de sucessivas erosões por funções estruturantes não crescentes.

3.3.1 Erosão por propagação

Será apresentada nesta subseção a *erosão por propagação* caracterizada por função estruturante. Não será apresentada a dilatação por propagação, pois é semelhante à erosão e foi defina por Barrera e Hirata [BHJ97].

Seja $f \in K^{\mathbb{E}}$ e $B \subseteq \mathbb{E} \oplus \mathbb{E}$ com origem. A *fronteira de propagação*, ou simplesmente *fronteira*, da erosão de f por $b \in \mathbb{Z}^B$ é o conjunto $\partial(f, b)$

definido por:

$$\partial(f,b) = \{x \in \mathbb{E} : \exists y \in B_x, f(y) > f(x)\dot{-}b(x-y)\}. \tag{3.12}$$

No Algoritmo 7 é apresentado o cômputo da fronteira de f usando uma vizinhança definida pela função estruturante b. Esta fronteira é armazenada no conjunto U, valor de saída de $\partial(f,b)$. Observe que antes de inserir um pixel no conjunto de fronteira, é feita uma verificação para que o pixel não seja inserido mais de uma vez desnecessariamente.

Algoritmo 7 Fronteira

$\partial : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow \mathbb{E}$
$\quad \partial(f,b) = U$

$U = \emptyset$;
$\forall x \in \mathbb{E}$
$\quad \forall y \in B_x \cap \mathbb{E}$
$\quad\quad$ **if** $f(y) > f(x)\dot{-}b(x-y)$ **e** $x \notin U$
$\quad\quad\quad U = U \cup \{x\}$;

A *erosão por propagação* de f por $b \in \mathbb{Z}^B$ é definida como:

$\forall x \in \partial(f,b),\ \forall y \in B_x \cap \mathbb{E}$,

$$\varepsilon_b^p(f)(y) = \min\{f(x)\dot{-}b(x-y)\}. \tag{3.13}$$

No Algoritmo 8 é apresentado o código que calcula a *erosão por propagação* e a nova fronteira tendo como entrada a imagem f, a função estruturante b e a fronteira U calculada por $\partial(f,b)$. Um exemplo de erosão por propagação é mostrado na Figura 3.10.

Algoritmo 8 Erosão por propagação

$\varepsilon^p : K^{\mathbb{E}} \times \mathbb{Z}^B \times \mathbb{E} \longrightarrow K^{\mathbb{E}} \times \mathbb{E}$
$\quad \varepsilon^p(f, b, U) = (g, U')$

$g = f;$
$U' = \emptyset;$
$\forall x \in U$
$\quad \forall y \in B_x \cap \mathbb{E}$
$\quad\quad$ **if** $g(y) > f(x) \dot{-} b(x - y)$
$\quad\quad\quad g(y) = f(x) \dot{-} b(x - y);$
$\quad\quad\quad$ **if** $y \notin U'$
$\quad\quad\quad\quad U' = U' \cup \{y\};$

3.3.2 Decomposição por propagação de função estruturante

De forma análoga ao ocorrido nos algoritmos paralelos e seqüenciais, a aplicação direta da erosão por uma função estruturante grande b_G é ineficiente. Contudo, é possível fazer a *decomposição por propagação da função estruturante* b_G para obter implementações rápidas, como apresentada a seguir.

É possível generalizar o Algoritmo 8 da *erosão por propagação* usando uma seqüência de erosões com funções estruturantes não crescentes, isto é, satisfazendo o critério $b_1 \geq b_2 \geq \cdots \geq b_k$, onde a relação de ordem é definida pelo reticulado $\mathbb{Z}^{B_i}$, onde $B_i \subseteq \mathbb{E} \oplus \mathbb{E}$ com origem. Esta *erosão por propagação generalizada* é mostrada no Algoritmo 9.

A condição para que o Algoritmo 9 seja válido é que a fronteira calculada na iteração i com o uso de b_i deve conter (por excesso) a fronteira associada a b_{i+1}, pois ela será usada no cálculo da erosão por b_{i+1}: $\partial(g, b_i) \geq \partial(g, b_{i+1})$.

A fronteira de f em relação à função estruturante b_i pode ser escrita

Algoritmo 9 Erosão por propagação generalizada

$\varepsilon^{pg} : K^{\mathbb{E}} \times \mathbb{Z}^{B_1} \times \mathbb{Z}^{B_2} \times \cdots \times \mathbb{Z}^{B_k} \longrightarrow K^{\mathbb{E}}$
$\quad \varepsilon^{pg}(f, b_1, b_2, \cdots, b_k) = g$

$g = f;$
$U = \partial(f, b_1);$
$\forall i \in [1, \cdots, k]$
$\quad (g, U) = \varepsilon^{p}(g, b_i, U);$

como:

$$\begin{aligned} \partial(f, b_i) &= (f \oplus \breve{b}_i) - f \quad \Longleftrightarrow \\ (f \oplus \breve{b}_i - f) &\geq (f \oplus \breve{b}_{i+1} - f) \quad \Longleftrightarrow \\ f \oplus \breve{b}_i &\geq f \oplus \breve{b}_{i+1} \quad \Longleftrightarrow \\ b_i &\geq b_{i+1}, \end{aligned}$$

que é a condição para que o Algoritmo 9 seja válido. No caso em que a erosão é idempotente este algoritmo é aplicado até a estabilidade. Quando isto ocorre, é possível obter a TD, como será visto no próximo capítulo.

Observe que a fronteira usada na erosão por propagação é formada pelos pixels que têm pelo menos um vizinho que vai ser erodido.

Analisando numa imagem binária, a fronteira é formada pelos pixels do fundo que têm pelo menos um vizinho pertencente a um objeto, como ilustrado na Figura 3.10. A fronteira usada na dilatação é formada pelos pixels do objeto que têm pelo menos um pixel de fundo. Outras operações morfológicas, como abertura e fechamento, podem ser computadas usando as erosões e dilatações por propagação, porém tomando o devido cuidado para não trocas as fronteiras. Segundo Vliet e Verwer [VV88] as rotinas para trocar uma fronteira da erosão para uma fronteira da dilatação e vice-versa são facilmente construídas.

Como sugestões para trabalhos futuros, seria interessante implementar

$$\begin{array}{ccccccccc} k & k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k & k \\ k & k & k & k & [0] & k & k & k & k \\ k & k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k & k \end{array}$$

(a) f

$$\begin{array}{ccccccccc} k & k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k & k \\ k & k & k & k & [1] & k & k & k & k \\ k & k & k & [1] & 0 & [1] & k & k & k \\ k & k & k & k & [1] & k & k & k & k \\ k & k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k & k \end{array}$$

(a) $\varepsilon_{b_4}^{p}(f)$

Figura 3.10: (a) Imagem de entrada f, onde o valor entre colchetes pertence à fronteira $\partial(f, b_4)$; (b) erosão por propagação por b_4, onde os pixels entre colchetes pertencem à nova fronteira $\partial(\varepsilon_{b_4}^{p}(f), b_4)$. b_4 é a métrica *city-block* definida no próximo capítulo.

os resultados de Vliet e Verwer [VV88] também para imagens em níveis de cinza, incluindo o trabalho da dilatação por propagação definida por Barrera e Hirata [BHJ97].

3.4 Resumo do capítulo

Nesta seção será apresentado um pequeno resumo das principais características dos estudos feito neste capítulo, no que diz respeito aos padrões de algoritmos para a erosão morfológica. A extensão para a dilatação é análoga.

A Tabela 3.1 resume as características dos padrões da erosão: as **implementações** são simples nos três padrões, como foi apresentado neste capítulo; na **arquitetura**, o padrão por propagação pode ser implementado em máquinas paralelas na erosão por uma fronteira, ou seja, é uma seqüência de algoritmos paralelos; a implementação em **hardware** é complexa no padrão por propagação, pois exige uma estrutura de **armazenamento** para a fronteira; considerando uma função estruturante constante, a **eficiência**

	paralela	**seqüencial**	**propagação**
implementação	simples	simples	simples
arquitetura	paralelo	seqüencial	paralelo
hardware	simples	simples	complexo
armazenamento	não	não	sim
eficiência	baixa	alta	alta
função estruturante	genérica	restrita	$b_1 \geq \cdots \geq b_k$
calcula a TDE	sim	não	sim

Tabela 3.1: Resumo dos padrões de algoritmos da erosão.

no caso paralelo apresenta complexidade linear com relação ao número de pixels da imagem a ser processada, porém faz acessos desnecessários (no caso seqüencial, esses acessos desnecessários são eliminados) e no padrão por propagação a complexidade é linear com relação ao número de pixels da fronteira[7]; para melhorar a eficiência dos algoritmos foi realizada a decomposição de **função estruturante** nos padrões:

paralelo: se $b_G = b_1 \oplus \cdots \oplus b_k \Longrightarrow \varepsilon_{b_G}(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots)$,

seqüencial: equivalência restrita entre erosão paralela e seqüencial:

- $b_G = kb^+ \vee kb^- \implies b$ genérico, porém não aplicável à TD,
- $b_G = kb^+ \oplus kb^- \implies b$ restrito, porém aplicável à TD,

propagação: se $b_G = b_1 \oplus \cdots \oplus b_k$ e $b_1 \geq \cdots \geq b_k$
$\Longrightarrow \varepsilon_{b_G}(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots)$;

com estas restrições, apenas o padrão seqüencial não **calcula a TDE**, pois não suporta os $b_i's$ genéricos apresentados no próximo capítulo.

[7] A complexidade de sucessivas erosões por propagação, Algoritmo 9, é mais difícil de ser analisada usando funções estruturantes variáveis. Porém em casos particulares, como para computar a TD nas métricas chanfradas definidas no próximo capítulo, é fácil ver que este algoritmo também é linear com relação ao número de pixel da imagem a ser processada, pois um pixel é inserido na fronteira no máximo uma vez.

Referências Bibliográficas

[BHJ97] J. Barrera e R. Hirata Jr. Fast algorithms to compute the elementary operators of mathematical morphology. No *Brazilian Symposium on Computer Graphics and Image Processing*, páginas 163–170, São Paulo, Brasil, Outubro 1997.

[Bor86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[Dan80] P.E. Danielsson. Euclidean distance mapping. *Computer Vision, Graphics and Image Processing*, 14:227–248, 1980.

[Dia69] R.B. Dial. Shortest-path forest with topological ordering. *Communications of the ACM*, 12(11):632–633, 1969.

[D'O01] M.C. D'Ornellas. *Algorithmic Patterns for Morphological Image Processing.* Tese de Doutorado, University of Amsterdam, Amsterdam, 2001.

[FLA01] A.X. Falcão, R.A. Lotufo e G. Araújo. The image foresting transformation. Relatório Técnico, Universidade Estadual de Campinas, 2001.

[HAS00] R.F. Hashimoto. *Mudança de Estrutura de Representação de Operadores em Morfologia Matemática.* Tese de Doutorado, Universidade de São Paulo, São Paulo - Brasil, 2000.

[LT00] N.J. Leite e M.D. Teixeira. An idempotent scale-space approach for morphological segmentation. No *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, páginas 341–350, Palo Alto, USA, Junho 2000.

[RP66] A. Rosenfeld e J.L. Pfalz. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, 13(4):471–494, Outubro 1966.

[Sch89] M. Schmitt. *Des algorithmes morphologiques a l'intelligence artificielle.* Tese de Doutorado, Ecole National des Mines de Paris, France, 1989.

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

[SM92] F.Y.-C. Shih e O.R. Mitchell. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing*, 1:197–204, 1992.

[Vin92] L. Vincent. Morphological algorithms. No E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, capítulo 8, páginas 255–288. Marcel Dekker, New York, 1992.

[Vin93] L. Vincent. Morphological grayscale reconstruction in image analysis. *IEEE Transactions on Image Processing*, 2(2):176–201, 1993.

[VV88] L.J. van Vliet e B.J.H. Verwer. A contour processing method for fast binary neighbourhood opeations. *Pattern Recognition Letters*, 7(1):27–36, 1988.

[WB88] X. Wang e G. Bertrand. An algorithm for a generalized distance transformation based on Minkowski operations. *9th ICPR*, páginas 1163–1167, 1988.

[WB92] X. Wang e G. Bertrand. Some sequential algorithms for a generalized distance transformation based on minkowski operations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(11):1114–1121, Novembro 1992.

[WHCR95] D. Wang, V. Haese-Coat e J. Ronsin. Shape decomposition and representation using a recursive morphological operation. *Pattern Recognition*, 28(11):1783–1792, 1995.

[YTF81] S. Yokoi, Jun-I. Toriwaki e T. Fukumura. On generalized distande transformation of digitized pictures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(4), 1981.

Capítulo 4

Classificação da TD

Neste capítulo serão classificados os algoritmos clássicos da *transformada de distância* (TD) que podem ser implementados por erosões nos padrões paralelo, seqüencial e por propagação. Estas erosões apresentam melhor eficiência quando usados em decomposições de funções estruturantes. Como resultado desta classificação serão propostos dois novos algoritmos para a *transformada de distância euclidiana* (TDE): um direcional e outro multidimensional.

4.1 Definições

Métricas particulares serão apresentadas nesta seção. Com estas métricas usadas nas funções estruturantes é possível computar erosões morfológicas e obter a TD.

Sejam $x = (x_1, x_2) \in \mathbb{Z} \times \mathbb{Z}$ e $y = (y_1, y_2) \in \mathbb{Z} \times \mathbb{Z}$. $d(x, y)$ é uma *distância* entre x e y, se:

(i) $d(x, y) = d(y, x)$;

(ii) $d(x, y) \geq 0$;

(iii) $d(x,x) = 0$.

Se, além disso, *iv* e *v* abaixo também forem satisfeitas, então d é uma *métrica*.

(iv) $d(x,y) = 0 \Longleftrightarrow x = y$;

(v) $d(x,y) \leq d(x,z) + d(z,y), \forall z \in \mathbb{Z} \times \mathbb{Z}$.

Algumas métricas usuais para $d(x,y)$ são apresentadas a seguir [Bor86, Cui99]:

***City-block*:** $d_4(x,y) = |x_1 - y_1| + |x_2 - y_2|$;

***Chessboard*:** $d_8(x,y) = \max\{|x_1 - y_1|, |x_2 - y_2|\}$;

***Chanfrada A:B*:** $d_{A:B}(x,y) = A * \max\{|x_1 - y_1|, |x_2 - y_2|\} + (B - A) * \min\{|x_1 - y_1|, |x_2 - y_2|\}$, com $0 < A \leq B$ inteiros;

***Euclidiana*:** $d_E(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$.

A *função distância de um pixel x ao conjunto X* é definida como [Ser82]:

$$d(x,X) = \min\{d(x,y) : y \in X\}.$$

A *função distância* (denotada $\Psi_d(f)$ e chamada neste texto de *transformada de distância* ou simplesmente TD) é definida como:

$$\Psi_d(f)(x) = d(x, \{y \in \mathbb{E} : f(y) = 0\}).$$

Esta função atribui a cada pixel de um objeto numa imagem binária o valor da distância mínima ao fundo. Em outras palavras, atribui a cada pixel de um objeto a menor distância entre este pixel e um pixel de fundo.

Serão usadas neste texto imagens com níveis de cinza como um subconjunto dos números inteiros $\mathbb{Z}$. Para a métrica *euclidiana*, será usado o

quadrado da distância euclidiana $(d_E)^2 \in \mathbb{Z}$, para permanecer com imagens inteiras das funções e obter a *transformada de distância euclidiana* (TDE). Assim, serão apresentados algoritmos para $(\text{TDE})^2$ e para obter exatamente a métrica euclidiana, deve-se calcular a raiz quadrada de todos os níveis de cinza gerados pela transformação. Portanto, ao dizer que um algoritmo obtém a métrica euclidiana, na verdade está obtendo o quadrado da métrica euclidiana.

4.2 Tipos de classificações da TD

A TD pode ser classificada de várias formas [Cui99]: pela métrica mais pobre (*city-block*) à mais exata (*euclidiana*), pela complexidade, pela eficiência, pela ordem de varredura, etc. Na tese do Cuisenaire [Cui99], existe uma classificação da TD da métrica mais pobre à mais exata, sendo divididas como *TD euclidiana aproximada* e *TD euclidiana exata*, respectivamente.

Na introdução desta tese, Seção 1.1, foi feito uma classificação da TD quanto à métrica usada, podendo ser *transformada de distância chanfrada* (TDC), incluindo as métricas onde a decomposição da função estruturante é constante. Exemplos de métricas que produzem a TDC: *city-block*, *chessboard*, *octagonal*, *chanfrada 3:4*, *chanfrada 5:7:11*, etc. A métrica mais natural para computar a distância na maioria das aplicações é a *euclidiana*, principalmente em função de sua propriedade de rotação invariante. Porém, devido a dificuldades de construir algoritmos eficientes para a *transformada de distância euclidiana* (TDE), muitos pesquisadores desenvolveram algoritmos aproximados (TDEA).

Uma classificação da TD quanto à ordem de varredura dos pixels na imagem foi feita na Seção 1.2. Três padrões de algoritmos foram definidos: paralelo, seqüencial e por propagação. Estes padrões estão descritos nas Subseções 4.3.1, 4.3.2 e 4.3.3, respectivamente.

Finalmente, na Seção 1.3, a TD por propagação foi subdividida em ou-

tros quatro tipos: propagação vetorial, propagação ordenada, propagação (ou propagação paralela) e geométrico. Vale observar que alguns algoritmos pertencem a mais de um tipo apresentado, como o do Ragnemalm [Rag92] e o do Eggers [Egg98] que são por propagação vetorial e também por propagação paralela. Mais informações para esta classificação são apresentadas a seguir.

TD por propagação vetorial

Os algoritmos da TD por propagação vetorial são difíceis de relacionar com a morfologia matemática, pois são decompostos em informações vetoriais do plano cartesiano discreto.

Cuisenaire [Cui99] chamou de *TD por propagação vetorial* os algoritmos que em vez de manipularem as distâncias (por exemplo, $d_4(x, y)$, $d_8(x, y)$, ...), manipulam os vetores distância (por exemplo, $(|x_1 - y_1|, |x_2 - y_2|)$, onde $x = (x_1, x_2) \in \mathbb{Z} \times \mathbb{Z}$ e $y = (y_1, y_2) \in \mathbb{Z} \times \mathbb{Z}$). O primeiro a implementar algoritmos seqüenciais para esta classe de TD foi Danielsson [Dan80]. Estes algoritmos não resultam na TDE, porém são eficientes (foram inspirados nos algoritmos seqüenciais definidos por Rosenfeld e Pfaltz [RP66]).

Ragnemalm [Rag92] modificou os algoritmos seqüenciais da TD definidos por Danielsson. Ele definiu um novo algoritmo por propagação vetorial que encontra a TDE, onde utiliza duas estruturas de filas para armazenar os pixels de propagação e várias condições de propagação. Para cada direção de propagação existe um teste, gerando um algoritmo complexo.

Eggers [Egg98] melhorou o algoritmo de Ragnemalm diminuindo os vários testes de direção, melhorando a velocidade do algoritmo de propagação, mas aumentando a complexidade da codificação.

Os trabalhos de TD por propagação vetorial, como os de Danielsson [Dan80], Ragnemalm [Rag92] e Eggers [Egg98], não serão reescritos por erosão morfológica pois necessitam manipular informações vetoriais e as erosões definidas neste texto são as clássicas, ou seja, manipulam distâncias e não vetores. Porém, inspirado no trabalho do Eggers, será proposto um

algoritmo que usa erosões e as informações de direção de propagação, como será apresentado na Seção 4.4.

O algoritmo do Danielsson [Dan80] e a primeira parte do algoritmo do Cuisenaire [Cui99] não calculam a TDE devido a não conexidade do diagrama de Voronoi no caso discreto. O *diagrama de Voronoi* divide um plano formados por polígonos desconexos e as uniões destes polígonos formam novamente o plano. A não conexidade do diagrama de Voronoi sempre ocorre próximo aos cantos dos polígonos de Voronoi, descrito inicialmente por Danielsson e corrigido por Cuisenaire na segunda parte do seu algoritmo da TDE. Yamada [Yam84] foi o primeiro a propor um algoritmo exato (que também é por propagação) que resolve este problema. A Figura 4.1 ilustra duas imagens onde ocorrem o erro nos algoritmos definidos por Danielsson. Ambas imagens contém três pixels ($p1$, $p2$ e $p3$) de valores zeros e os restantes com valores uns. A imagem à esquerda ilustra a métrica *city-block* e a imagem à direita a métrica *chessboard*. A região ao redor do pixel $p2$ é desconectado do pixel q pelas regiões geradas pelos pixels $p1$ e $p3$, onde $r1$ pertence à região (polígono) propagada a partir de $p1$ e $r2$ pela região propagada a partir de $p3$. Assim, na figura à esquerda q deveria receber $\sqrt{8}$ e não $\sqrt{9}$ ($q - p1 = (3, 0)$, $q - p2 = (2, 2)$, $q - p3 = (0, 3)$). Na figura à direita q deveria receber $\sqrt{144 + 25}$ e não $\min\{\sqrt{169 + 1}, \sqrt{121 + 49}\}$ ($q - p1 = (13, 1)$, $q - p2 = (12, 5)$ e $q - p3 = (11, 7)$). A TDE (como o diagrama de Voronoi) não é uma propriedade local, a informação de vizinhança de um pixel não é propagada para pixels distantes. Portanto esse problema da não conexidade ocorre pois são propagadas informações locais, vizinho-por-vizinho.

TD por propagação ordenada

Os algoritmos por *propagação ordenada* são melhores modelados como o problema de caminho mais curto usado em teoria dos grafos, implementados usando fila hierárquica. Não é possível modelar este problema por erosões morfológicas.

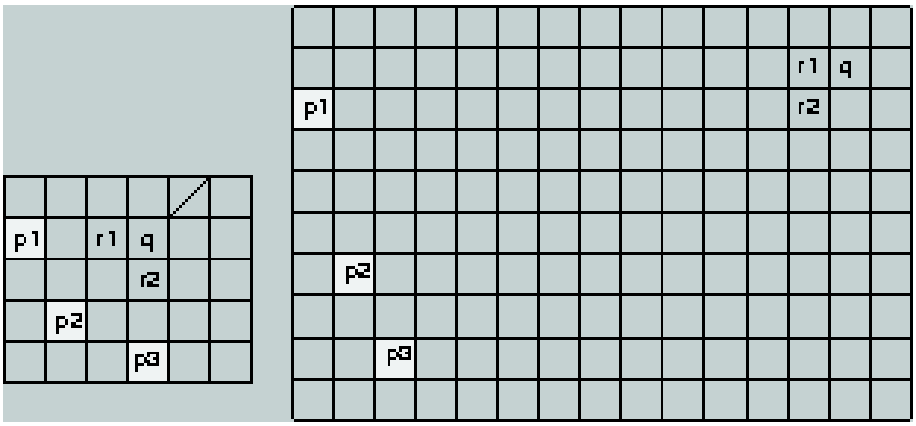


Figura 4.1: Erros ocorridos nos algoritmos de Danielsson: para métrica *city-block* (esquerda) e *chessboard* (direita) (fonte [Cui99]).

Sharaiha e Christofides [SC94] propuseram uma aproximação *baseada em grafos* para a TD. A TD baseada em grafos pode ser reduzida ao problema de floresta de caminho mínimo, onde as raízes das árvores são determinadas pelos pixels de fronteira dos objetos da imagem. Sharaiha e Christofides descreveram uma TD *chanfrada* baseada em grafos e apontaram as principais vantagens de usar a teoria dos grafos em vez de usar a aproximação pixel-por-pixel usada em processamento de imagens. A saber: o tamanho do grafo é normalmente menor que o tamanho da imagem; cada pixel é normalmente processado uma única vez; o algoritmo pode ser estendido para dimensões maiores e para diferentes grades topológicas (por exemplo, grade hexagonal).

Outro trabalho importante da TD por propagação ordenada foi definido por Lotufo *et. al.* [LFZ02], onde definiu o algoritmo *IFT* (*Image Floresting Transform*). A *IFT* tem como entrada uma imagem em níveis de cinza, uma outra imagem com marcadores e uma função estruturante para determinar a vizinhança. Como saída, o algoritmo retorna uma imagem de custos (para algumas entradas especiais esta imagem de custos é exatamente a TD), uma imagem representando a propagação de cada marcador, ou *watershed*, e o caminho mínimo até o marcador mais próximo.

Um estudo de *transformações baseadas em grafos* pode ser encontrado em Zampirolli [Zam97], onde se criou um conjunto de ferramentas baseadas em grafos na *toolbox MMach* [BBL94], usando o ambiente *KHOROS*, inclusive incluindo a TD por propagação ordenada.

TD por propagação

Os algoritmos da TD por *propagação* (ou *propagação paralela*) têm relações diretas com erosões morfológicas e serão estudados nas próximas seções.

Os trabalhos de Ragnemalm [Rag92], Eggers [Egg98] e Yamada [Yam84] apresentados anteriormente no tipo de propagação vetorial também são por propagação paralela. A chave destes algoritmos é: usar duas estruturas de armazenamento para o mapa de custos (*custo_velho* e *custo_novo*); propagar todos os pixels em paralelo atualizando *custo_novo* apenas com os dados de *custo_velho*; depois que tudo foi atualizado, trocar *custo_novo* por *custo_velho* e fazer outra propagação paralela; continuar fazendo isto até não existir mais propagação. Estes algoritmos não são baseados explicitamente em erosão, porém suas implementações são bastante semelhantes ao algoritmo da TDE usando erosão por propagação apresentado na Subseção 4.3.3. O trabalho de Eggers [Egg98] propôs várias melhorias ao trabalho de Ragnemalm [Rag92] fazendo um algoritmo que realiza menos acessos aos pixels. A idéia é fazer uso da direção da propagação para restringir a varredura da vizinhança apenas aos pixels que podem ser modificados. Usando as informações contidas no algoritmo do Eggers é possível alterar o algoritmo da TDE utilizando erosão por propagação para que faça uso da direção da propagação, como apresentado na Seção 4.4.

Outro trabalho importante para esta tese foi feito por Vincent [Vin92], que implementou a TD por propagação a qual está apresentada no Algoritmo 11, Subseção 4.3.3. Este algoritmo da TD usa uma estrutura de fila e poderia também ser usado para calcular a TDE, porém há a necessidade

de trocar a função estruturante a cada erosão e fazer cópia de imagens. Isto poderia ser feito colocando na fila um *token* indicando o fim de cada erosão. A cada retirada do *token*, seria trocado a função estruturante. A reescrita do Algoritmo 11 é apresentada no Algoritmo 13, porém sem o uso de fila.

TD geométrica

Os algoritmos *geométricos* para a TD usando intersecção de parábolas formam o último tipo de TD por propagação. Como exemplo, Saito e Toriwaki [ST94] mostraram que a TDE é separável. Este problema é exatamente o que faz a morfologia matemática, quando o problema é decomposto em elementos unidimensionais. Nesta linha existem vários outros artigos: como o que valida o espaço contínuo [EBS01]; o algoritmo definido por Meijster e Roerdink [MR00], um dos mais eficientes reportados na literatura; e também o proposto neste documento, veja Seção 4.5.

4.3 Padrões de algoritmos da TD usando erosões

Shih e Mitchell [SM92] mostraram que a TD pode ser computada através de uma erosão morfológica por uma função estruturante dada pelo negativo da distância à origem. Mais tarde, Huang e Mitchell [HM94] chegaram em uma computação eficiente para a TDE decompondo a função estruturante euclidiana por uma seqüência de funções estruturantes diferentes de tamanho 3×3.

O trabalho de Mitchell *et. al.* [SM92, HM94] abriu uma porta para estudar e classificar a diversidade de algoritmos da TD disponíveis na literatura. Devido ao fato de existirem muitos caminhos para implementar eficientemente a erosão morfológica, Zampirolli e Lotufo [ZL00] propuseram uma classificação de algoritmos da TD, onde são analisados quais algoritmos da

erosão e quais esquemas de decomposição de função estruturante foram utilizados na literatura.

4.3.1 TD paralela

TD usando erosão paralela

Shih e Mitchell [SM92] propuseram um algoritmo para a TD usando a erosão morfológica aplicada em uma imagem binária f por uma função estruturante b_G.

Veja na Figura 4.2 um exemplo da TD usando erosão para o caso unidimensional. Assim, seja a seguinte equação para a TD:

$$\Psi_d(f) = \varepsilon_{b_G}(f). \tag{4.1}$$

O valor na origem de b_G é zero e nos outros pixels é dado pelo negativo da distância à origem, isto é,

$$b_G(x) = -d(x, \mathcal{O}), x \in \mathbb{E} \oplus \mathbb{E}, \tag{4.2}$$

onde $\mathcal{O} = (0, 0)$. O tipo da TD (*city-block*, *chessboard*, *octagonal*, *chanfrada*, *euclidiana*, etc) vai depender da métrica utilizada pela função estruturante.

Uma propriedade da erosão específica para a TD é ser *idempotente*, isto é, ao aplicar a erosão por b_G novamente, o resultado não é modificado [LT00][1]:

$$\varepsilon_{b_G}(\varepsilon_{b_G}(f)) = \varepsilon_{b_G}(f). \tag{4.3}$$

Esta propriedade é útil, por exemplo, quando se deseja trabalhar com a

[1]Um trabalho interessante que estuda as condições para que dilatações e erosões em espaço-escala morfológico sejam idempotentes foi realizado por Leite e Teixeira [LT00]. Este trabalho será comentado nas conclusões desta tese como trabalhos futuros para a TD.

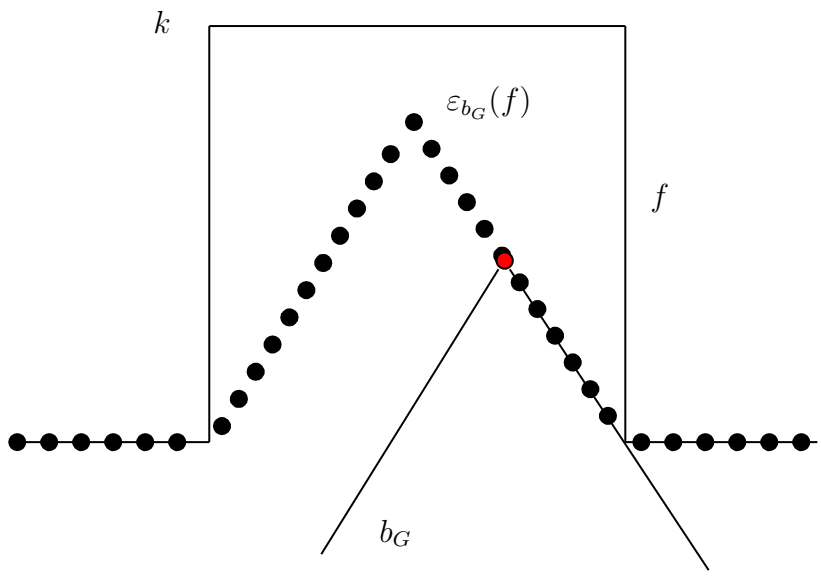


Figura 4.2: TD usando erosão unidimensional da imagem f por b_G.

decomposição da função estruturante b_G para obter algoritmos eficientes e estáveis, como serão apresentados a seguir.

Pelos estudos de decomposição paralela de função estruturante apresentados na Subseção 3.1.2 e pela Equação 4.1 é definido a seguinte equação para a *transformada de distância* [Ser82, SM92]:

$$\Psi_d(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots), \tag{4.4}$$

onde $b_G = b_1 \oplus \cdots \oplus b_k$ e o valor de k deve ser o maior valor de distância que pode ocorrer na imagem f. Como ilustração, veja Figura 4.3.

Se for aplicado k vezes o Algoritmo 4 da erosão, apresentado na Subseção 3.1.1, então a TD é computada. Além disso, a mesma saída é encontrada se for aplicado recursivamente este algoritmo até a estabilização. Este segundo caso tem como resultado a TD por causa da propriedade de operação idempotente, Equação 4.3, e é apresentado no Algoritmo 10.

Outros tipos de funções estruturantes aplicáveis a Equação 4.4 estão

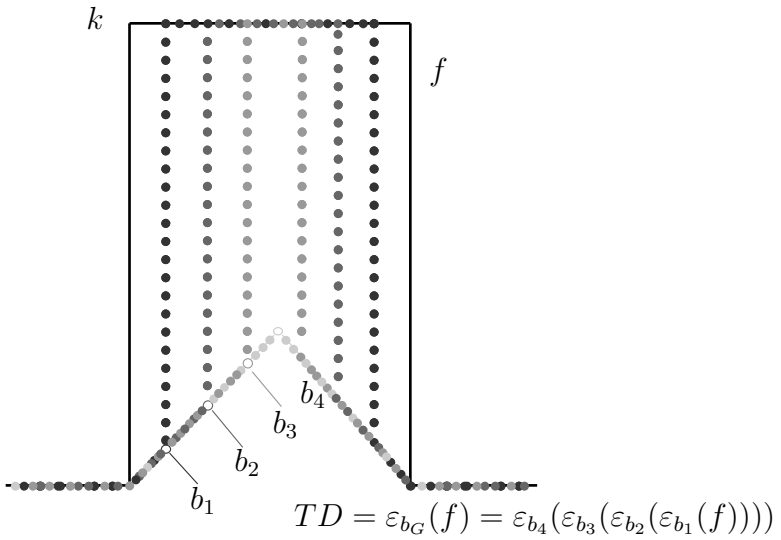


Figura 4.3: TD usando erosões por decomposições de funções estruturantes unidimensionais.

ilustrados nas Figuras 4.4a, 4.4b e 4.4c. Estes exemplos mostram funções estruturantes $b_i = b$, onde $b_G = kb$ para as métricas *city-block*, *chessboard* e *octagonal*, respectivamente. Estas métricas estão ilustradas também nas Figuras 4.7a, 4.7b e 4.7c, respectivamente.

$$\begin{array}{ccc} & -1 & \\ -1 & \mathbf{0} & -1 \\ & -1 & \end{array} \qquad \begin{array}{ccc} -1 & -1 & -1 \\ -1 & \mathbf{0} & -1 \\ -1 & -1 & -1 \end{array} \qquad \begin{array}{ccccc} & & -2 & & \\ & -2 & -1 & -2 & \\ -2 & -1 & \mathbf{0} & -1 & -2 \\ & -2 & -1 & -2 & \\ & & -2 & & \end{array}$$

(a) b_4 (b) b_8 (c) $b_O = b_4 \oplus b_8$

Figura 4.4: Funções estruturantes usadas nas decomposições das métricas (a) *city-block*, (b) *chessboard* e (c) *octagonal*, onde as origens estão em negrito.

Outros tipos de decomposições para b_G são as métricas *chanfrada 3:4* (Figuras 4.5a e 4.7d) e *chanfrada 5:7:11* (Figuras 4.5b e 4.7e). O trabalho de Borgefors [Bor86] trouxe um estudo completo da aproximação destas e

de outras métricas com a métrica *euclidiana*. Por exemplo, se ao invés de usar $a = 3$ e $b = 4$ na métrica *chanfrada 3:4* usar $a = 2$ e $b = 3$ (ou seja, *chanfrada 2:3*) a diferença com a métrica euclidiana vai de 0.0809 para 0.1340 aplicada em uma mesma imagem de um certo tamanho [Bor86]. Os algoritmos apresentados por Borgefors são semelhantes aos de Rosenfeld e Pfaltz [RP66, RP68] e poderão ser reescritos por erosão, como apresentados a seguir.

$$\begin{array}{ccc} -4 & -3 & -4 \\ -3 & \mathbf{0} & -3 \\ -4 & -3 & -4 \end{array}$$

(a) $b_{3:4}$

$$\begin{array}{ccccc} & & -11 & & -11 & \\ & -11 & -7 & -5 & -7 & -11 \\ & & -5 & \mathbf{0} & -5 & \\ & -11 & -7 & -5 & -7 & -11 \\ & & -11 & & -11 & \end{array}$$

(b) $b_{5:7:11}$

Figura 4.5: Funções estruturantes usadas nas decomposições das métricas (a) *chanfrada 3:4* e (b) *chanfrada 5:7:11*, onde as origens estão em negrito.

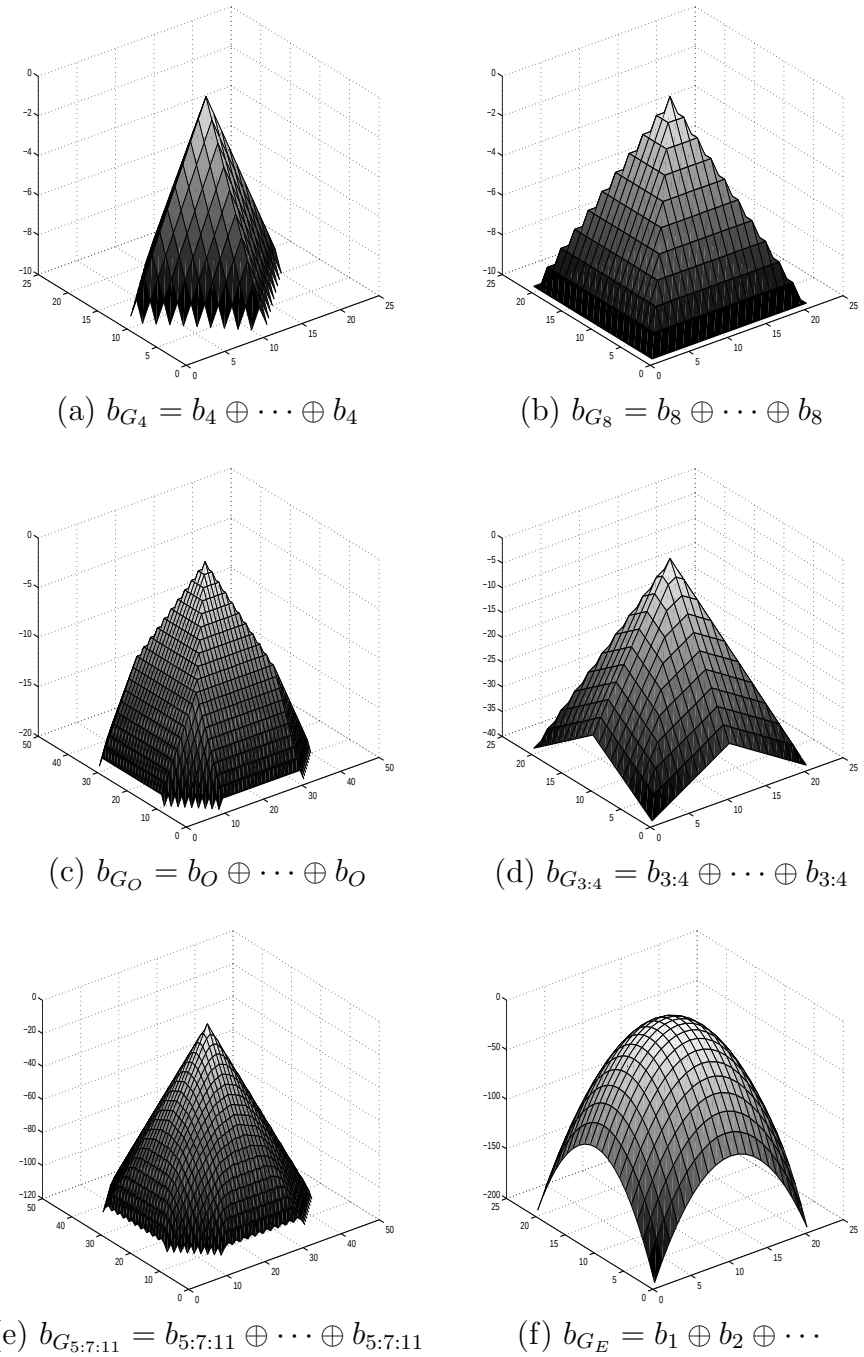
Huang e Mitchell [HM94] mostraram que a métrica euclidiana dada pela função estruturante b_{G_E} (Figura 4.7f) pode ser decomposta através de funções estruturantes 3×3 distintas, b_i, i inteiro positivo, veja Figura 4.6.

$-4i+2$	$-2i+1$	$-4i+2$
$-2i+1$	$\mathbf{0}$	$-2i+1$
$-4i+2$	$-2i+1$	$-4i+2$

Figura 4.6: b_i elemento estruturante usado na Equação 4.4 para obter a TDE, onde a origem está em negrito.

Reescrita de algoritmo paralelo da TD

Baseado na implementação da *erosão paralela*, Algoritmos 4 e 10, é possível reescrever os principais algoritmos da TD usando a teoria de ope-



(a) $b_{G_4} = b_4 \oplus \cdots \oplus b_4$ (b) $b_{G_8} = b_8 \oplus \cdots \oplus b_8$

(c) $b_{G_O} = b_O \oplus \cdots \oplus b_O$ (d) $b_{G_{3:4}} = b_{3:4} \oplus \cdots \oplus b_{3:4}$

(e) $b_{G_{5:7:11}} = b_{5:7:11} \oplus \cdots \oplus b_{5:7:11}$ (f) $b_{G_E} = b_1 \oplus b_2 \oplus \cdots$

Figura 4.7: Funções estruturantes obtidas através de 10 somas de Minkowski de suas respectivas decomposições.

Algoritmo 10 TD paralela

$TD : K^{\mathbb{E}} \times \mathbb{Z}^B \times \mathbb{Z}^B \times \cdots \longrightarrow K^{\mathbb{E}}$
$\quad TD(f, b_1, b_2, \ldots) = g$

$g = f;$
$i = 0;$
$\forall g \neq \varepsilon_{b_i}(g)$
$\quad g = \varepsilon_{b_i}(g);$
$\quad i++;$

radores morfológicos caracterizados por decomposição de função estruturante.

O primeiro algoritmo paralelo para a TD foi definido por Rosenfeld e Pfaltz e publicado em 1968 [RP68]. Outro trabalho clássico que usa esta implementação foi definido por Borgefors [Bor86]. Em ambos os casos a imagem de entrada assume valores zeros e uns e são descritos abaixo:

$$v_{i,j}^m = \min_{(k,l)\in mask}\{v_{i+k,j+l}^{m-1} + c(k,l)\}, \text{ até estabilizar,} \tag{4.5}$$

onde $v_{i,j}^m$ é o valor do pixel na posição (i, j) da imagem na iteração m, (k, l) é a posição na máscara $mask$ e $c(k, l)$ é o valor da máscara em (k, l) $(c(0, 0) = 0)$.

Note que as Equações 2.5 e 3.4 são equivalentes a Equação 4.5 resultando na mesma TD, considerando $b = b_i = -mask$, $i = 1, \cdots, k$. Estas conclusões são descritas com mais detalhes no trabalho de Shih e Mitchell de 1992 [SM92].

A melhor análise destas duas equações considera $x = (i, j)$ e $y = (i+k, j+l)$. A Equação 4.5 pode ser definida como $v^m(x) = \min\{v^{m-1}(y) \dot{-} b(y - x) : y \in B_x\}$, onde $x \in \mathbb{E}$. Note que, se $v^0 = f$ e $v^1 = \varepsilon_b(f)$, então é encontrado a definição da erosão da Equação 2.5. Como foi visto, para calcular a TD usando esta erosão é preciso considerar uma imagem com valores zeros e k (maior distância possível na imagem) e computar a erosão até a estabilização.

TDE em paralelo

Como exemplo de TDE usando uma *erosão paralela*, a Figura 4.8 mostra a função estruturante b_{G_E}, onde a origem está no topo do parabolóide. Esta função estruturante pode ser usada para calcular a TDE na imagem f fazendo $\varepsilon_{b_{G_E}}(f)$.

Como foi visto na Subseção 3.1.2, a função estruturante euclidiana b_{G_E} pode ser escrita como a *soma de Minkowski generalizada em níveis de cinza* de diversas funções estruturantes $b_i \in \{b_1, \cdots, b_k\}$. Usando a formação dada pela Figura 4.6 é possível obter $b_{G_E} = b_1 \oplus \cdots \oplus b_k$ e $\varepsilon_{b_{G_E}}(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots)$ =TDE, usando o Algoritmo 10.

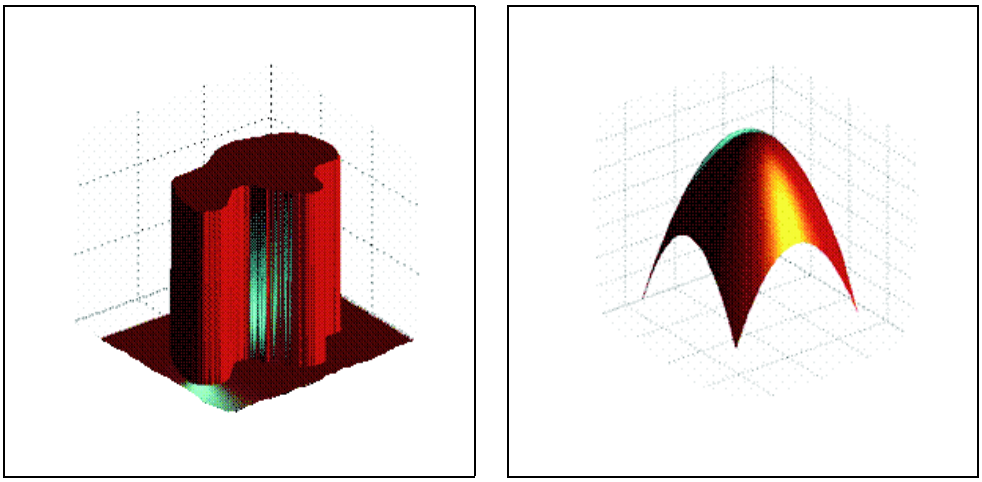
4.3.2 TD seqüencial

A erosão morfológica implementada de forma seqüencial foi vista na Seção 3.2. O algoritmo da TD seqüencial é exatamente o algoritmo da *erosão seqüencial*, Algoritmos 5 e 6, usando a decomposição $b_G = kb = kb^+ \oplus kb^-$. Veja um exemplo na Figura 4.9.

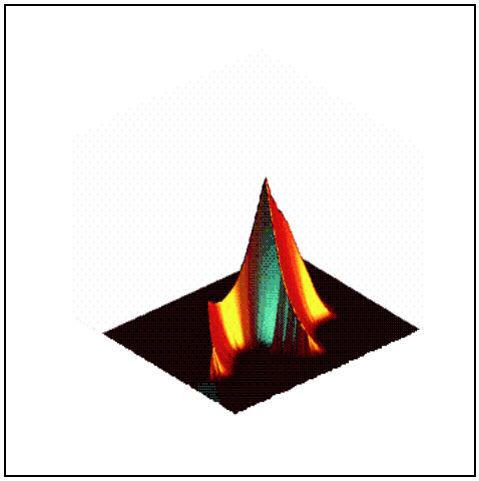
Note que não é possível usar uma decomposição com diferentes $b_i's$, como foram implementados nos padrões *paralelo* e *por propagação*. Portanto, não é possível calcular a métrica euclidiana usando apenas erosão seqüencial.

Reescrita de algoritmo seqüencial da TD

O primeiro algoritmo seqüencial da TD foi descrito por Rosenfeld e Pfaltz e publicado em 1966 [RP66]. Em seu algoritmo, a imagem de entrada contém somente 0/1, e o conjunto de pixels com valores zeros é não vazio. O algoritmo



f b_{G_E}



TDE

Figura 4.8: Imagem de entrada f, função estruturante euclidiana b_{G_E} e $TDE = \varepsilon_{b_{G_E}}(f)$.

-1	-1	-1
-1	**0**	-1
-1	-1	-1

b_8

-1	-1	-1
-1	**0**	

b_8^-

	0	-1
-1	-1	-1

b_8^+

k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	0	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k

(a) f

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	3	4
0	0	0	1	1	1	2	3	4
0	0	2	2	2	2	2	3	4
0	3	3	3	3	3	3	3	4

(b) $\varepsilon^+_{b_8^+}(f)$

4	3	3	3	3	3	3	3	4
4	3	2	2	2	2	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	1	0	1	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	2	2	2	2	3	4
4	3	3	3	3	3	3	3	4

(c) $\varepsilon^-_{b_8^-}(\varepsilon^+_{b_8^+}(f))$

Figura 4.9: (a) Imagem de entrada; (b) erosão *raster* por b_8^+ e (c) erosão *anti-raster* de (b) por b_8^-.

é dado por [2],

$$f_1(a_{i,j}) = \begin{cases} 0, \quad se\ a_{i,j} = 0, \\ \min(a_{i-1,j} + 1, a_{i,j-1} + 1), \quad se\ (i,j) \neq (1,1)\ e\ a_{i,j} = 1, \\ m + n, \quad se\ (i,j) = (1,1)\ e\ a_{i,j} = 1,\ e \end{cases} \tag{4.6}$$

$$f_2(a_{i,j}) = \min(a_{i,j}, a_{i+1,j} + 1, a_{i,j+1} + 1),$$

onde $a_{i,j}$ é o valor do pixel na posição (i, j) em uma imagem com m colunas e n linhas. Os valores $a_{i,j}$ fora da imagem não são definidos. f_1 é aplicada na ordem *raster* e, sobre o resultado, f_2 é aplicada na ordem *anti-raster*. Esta TD usa a métrica *city-block*. Se $a_{1,1} = 1$, o algoritmo faz $f_1(a_{1,1}) = m + n$, que é a maior distância possível na imagem. Este passo no algoritmo pode ser eliminado se for assumido uma imagem de entrada com valores zeros e k (k é a maior distância possível na imagem). Além disso, a primeira linha também pode ser eliminada se for incluída na função *min* três argumentos em vez de dois. Assim, $f_1(a_{i,j}) = min(a_{i,j}, a_{i-1,j} + 1, a_{i,j-1} + 1)$ pode ser substituída no passo f_1 do algoritmo acima.

Fazendo b^+ e b^- funções que decompõem a métrica *city-block* para as varreduras *raster* e *anti-raster*, respectivamente, como mostra a Figura 4.10, é possível reescrever a Equação 4.6 como:

$$f_1(a_{i,j}) = min\{a_{i,j} - b^+_{0,0}, a_{i+1,j} - b^+_{1,0}, a_{i,j+1} - b^+_{0,1}\},$$
$$f_2(a_{i,j}) = min\{a_{i,j} - b^-_{0,0}, a_{i-1,j} - b^-_{-1,0}, a_{i,j-1} - b^-_{0,-1}\}.$$

Se $x = (i, j)$ e $f(x) = a_{i,j}$, então $f_1(a_{i,j}) = \varepsilon^+_{b^+}(f)(x)$ e $f_2(a_{i,j}) = \varepsilon^-_{b^-}(f)(x)$, como definidos nas Equações 3.5 e 3.6, respectivamente. O algoritmo da TD de Rosenfeld e Pfaltz [RP66] é então equivalente a nossa *erosão*

[2] Observe que neste algoritmo o domínio da imagem está nos intervalos $1 \leq i \leq m$ e $1 \leq j \leq n$, diferente do definido na Seção 3.2.1, porém isto não afeta a nossa análise.

$$\begin{array}{cc} & -1 \\ -1 & \mathbf{0} \end{array} \qquad \begin{array}{cc} \mathbf{0} & -1 \\ -1 & \end{array}$$

$$b_4^- \qquad\qquad b_4^+$$

Figura 4.10: Decomposição *anti-raster* b_4^- e *raster* b_4^+ para a métrica *city-block*, com valor zero em negrito no centro.

seqüencial, Algoritmos 5 e 6. Borgefors [Bor86] também implementou um algoritmo seqüencial para a TD, o qual é também proposto aqui.

4.3.3 TD por propagação

Serão apresentados a seguir algoritmos da TD que usam a *erosão por propagação*, Algoritmo 8, e a *erosão por propagação generalizada*, Algoritmo 9.

Reescrita de algoritmo por propagação da TD

Vincent [Vin92] implementou a TD por propagação a qual está apresentada no Algoritmo 11. Este algoritmo usa uma estrutura de *fila* para armazenar a *fronteira*.

Nesse algoritmo, $FIFO\_add(x)$, $FIFO\_empty()$ e $FIFO\_first()$ são as primitivas de manipulação de fila que adiciona um elemento x na fila, verifica se a fila está vazia e retorna o primeiro elemento da fila, respectivamente.

Esse algoritmo pode ser reescrito como apresentado no Algoritmo 12, onde o primeiro $\forall$ coloca na fila todos os pixels de *fronteira* da imagem f. Este laço pode ser substituído pela função que encontra a fronteira, a qual coloca os pixels de fronteira no conjunto U como apresentado no código do Algoritmo 13. No laço $\forall\ FIFO\_empty() \neq \varnothing$ do Algoritmo 11, todos os

Algoritmo 11 TD por Vincent

$\mathrm{TD}^{Vinc} : \{0,1\}^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\mathrm{TD}^{Vinc}(f) = f$ // trabalha na mesma imagem

$\forall x \in \mathbb{E}$ // B é a vizinhança 4 ou 8 definido pela grade quadrada
if $f(x) = 1$ and $\exists y \in B_x \cap \mathbb{E} \mid f(y) = 0$
$FIFO\_add(x)$;
$f(x) = 2$; // distância iniciada com valor 2 para a fronteira
$\forall\ FIFO\_empty() \neq \varnothing$
$x = FIFO\_first()$;
$\forall y \in B_x \cap \mathbb{E}$
if $f(y) = 1$
$f(y) = f(x) + 1$;
$FIFO\_add(y)$;

Algoritmo 12 TD auxiliar

$\mathrm{TD}^{aux} : \{0,k\}^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\mathrm{TD}^{aux}(f) = f$ // $k = \infty$

$\forall x \in \mathbb{E}$
$\forall y \in B_x \cap \mathbb{E}$
if $f(y) > f(x) + 1$
$FIFO\_add(x)$;
$f(y) = 1$;
$\forall\ FIFO\_empty() \neq \varnothing$
$x = FIFO\_first()$;
$\forall y \in B_x \cap \mathbb{E}$
if $f(y) > f(x) + 1$
$f(y) = f(x) + 1$;
$FIFO\_add(y)$;

pixels de fronteira (da *fila*) são usados para computar a erosão por seus pixels vizinhos e ao mesmo tempo um novo pixel de fronteira é computado para verificar se será inserido na fila. Este processo é um caso particular do Algoritmo 8 - *Erosão por Propagação*, apresentado anteriormente, pois considera uma métrica fixa (*city-block* ou *chessboard*) em vez de usar uma função estruturante genérica b. Observe que não é necessário usar filas, mas dois conjuntos, que são trocados em cada iteração.

No Algoritmo 13 é apresentado uma generalização do algoritmo proposto por Vincent [Vin92].

Algoritmo 13 TD por propagação

$TD^p : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}}$
$TD^p(f, b) = g$

$g = \varepsilon_b(f)$;
$U = \partial(g, b)$;
$\forall\ U \neq \emptyset$ // até estabilizar
$(g, U) = \varepsilon^p(g, b, U)$;

TDE por propagação

O algoritmo da TDE por propagação é o Algoritmo 9 - *Erosão por propagação generalizada*, usando as decomposições de $b_{G_E} = b_1 \oplus \cdots \oplus b_k$, onde os $b_i's$ foram definidos na Figura 4.6 e b_{G_E} é ilustrado na Figura 4.7f. Este algoritmo, bastante simples, apresenta uma boa eficiência quando implementado em máquinas seqüenciais de propósito geral, como analisado na Seção 4.7 e resumido na Tabela 4.2.

4.4 TD por propagação direcional

Ragnemalm [Rag92] implementou a TDE utilizando duas estruturas de filas para armazenar os pixels de *fronteira* e várias condições de propagação, onde para cada direção de propagação existe um teste, gerando um algoritmo complexo. Eggers [Egg98] melhorou o algoritmo de Ragnemalm diminuindo os vários testes de direção, porém criando inúmeras variáveis, deixando o código mais eficiente, contudo não menos complexo. A idéia proposta por Eggers é fazer uso da direção da propagação para restringir a varredura da vizinhança apenas aos pixels que podem ser modificados. Inspirado neste princípio, é possível alterar o Algoritmo 9 - *Erosão por propagação generalizada*, para que as erosões também façam uso da direção da propagação, como será apresentado a seguir.

Considere uma outra versão para o cômputo da fronteira ∂, onde além de armazenar a coordenada x, armazena também a direção de propagação da erosão definida por um inteiro $k \in \{1, \ldots, 8\}$, cuja direção corresponde a propagação de x, conforme a Figura 4.11.

5	6	7
4		8
3	2	1

Figura 4.11: Direções de propagação.

Alterando o Algoritmo 9 - *Erosão por propagação generalizada*, para fazer uso da direção de propagação, é apresentado o Algoritmo 14 (agora U contém $(x, d) \in \mathbb{E}'$, onde $\mathbb{E}' = \mathbb{E} \times \{1, \ldots, 8\}$, x é a coordenada e d é a direção de propagação).

Os comandos ε^{init} e ε^{dir} que aparecem no Algoritmo 14 estão definidos nos Algoritmos 15 e 16, respectivamente.

No Algoritmo 16, $B'_x[k{-}{-}, k, k{+}{+}]$ é o subconjunto das direções: *anterior*,

Algoritmo 14 TD por propagação direcional

$\mathrm{TD}^{dir} : K^{\mathbb{E}} \times \mathbb{Z}^B \times \mathbb{Z}^B \times \cdots \longrightarrow K^{\mathbb{E}}$
$\mathrm{TD}^{dir}(f, b_1, b_2, \cdots) = g$

$[g, U] = \varepsilon^{init}(f, b_1)$;
$i = 2$;
$\forall U \neq \emptyset$
  $(g, U) = \varepsilon^{dir}(f, b_i, U)$;
  $i{+}{+}$;

Algoritmo 15 Erosão inicial

$\varepsilon^{init} : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}} \times \mathbb{E}'$ // $\mathbb{E}' = \mathbb{E} \times \{1, \ldots, 8\}$
$\varepsilon^{init}(f, b) = (g, U)$

$g = f$;
$U = \emptyset$;
$\forall x \in \mathbb{E}$
  $\forall y \in B_x \cap \mathbb{E}$
    if $g(x) > f(y) \dot{-} b(y - x)$
      $g(x) = f(y) \dot{-} b(y - x)$;
      if $x \notin U$
        $U = U \cup \{(x, d)\}$; // d é a direção de B

k e *posterior* a k, considerando o sentido horário, como ilustrado na Figura 4.11. A direção k é a direção de propagação de x obtida numa iteração anterior. Se $k = 1$, então *anterior* é 8, e se $k = 8$, *posterior* é 1. Na última linha do código, d é $k{-}{-}$, k ou $k{+}{+}$ dependendo da direção do vetor com origem x e destino y.

Algoritmo 16 Erosão por propagação direcional

$\varepsilon^{dir} : K^{\mathbb{E}} \times \mathbb{Z}^B \times \mathbb{E}' \longrightarrow K^{\mathbb{E}} \times \mathbb{E}' \quad // \;\; \mathbb{E}' = \mathbb{E} \times \{1, \ldots, 8\}$
$\varepsilon^{dir}(f, b, U) = (g, U')$

$g = f;$
$\forall\ (x, k) \in U$
  $\forall y \in B'_x[k{-}{-}, k, k{+}{+}] \cap \mathbb{E}$
    if $g(y) > f(x) \dot{-} b(x - y)$
      $g(y) = f(x) \dot{-} b(x - y);$
      $U = U \cup \{(y, d)\};$ // d é a direção de B

Se forem usadas as funções estruturantes $b'_i s$, como definidas na Figura 4.6, o Algoritmo 14 encontra a TDE, que é similar em vários aspectos ao proposto por Eggers [Egg98], porém mais simples e com desempenhos equivalentes, como analisados na Seção 4.7.

A Figura 4.12 ilustra uma iteração intermediária da TD por propagação direcional da imagem de entrada f pelas funções estruturantes $b_1, b_2, \ldots$ definidas pela Figura 4.6. A função estruturante que está sendo processado na iteração é b_2 com a direção de propagação 4, o que implica ao algoritmo processar apenas as direções $[3, 4, 5]$, conforme Figura 4.11. O pixel x, que está sendo processado, é o em negrito e está recebendo o valor $\mathbf{4} = \bigwedge\{1\dot{-}(-6), 1\dot{-}(-3), 1\dot{-}(-6)\} = \mathrm{TD}^{dir}(f, b_1, b_2, \ldots)(x)$.

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \end{array}$$

f

$$\begin{array}{ccc} -2 & -1 & -2 \\ -1 & \mathbf{0} & -1 \\ -2 & -1 & -2 \end{array}$$

b_1

$$\begin{array}{ccc} -6 & -3 & -6 \\ -3 & \mathbf{0} & -3 \\ -6 & -3 & -6 \end{array}$$

b_2

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 4 & 4 & 4 \\ 0 & 0 & 1 & \mathbf{4} & k & k & k \\ 0 & 0 & 1 & 4 & k & k & k \\ 0 & 0 & 1 & 4 & k & k & k \end{array}$$

$\mathrm{TD}^{dir}(f, b_1, b_2, \ldots)(x)$

Figura 4.12: Ilustração de uma iteração intermediária da TD por propagação direcional, $\mathrm{TD}^{dir}(f, b_1, b_2, \ldots)(x)$, da imagem de entrada f pelas funções estruturantes $b_1, b_2, \ldots$, onde o pixel x, com valor **4** em negrito, é o que está sendo processado e a direção propagada é 4, conforme Figura 4.11.

4.5 TDE multidimensional

Será apresentado nesta seção um algoritmo da TDE multidimensional formulado usando erosões morfológicas. A TD é composta de várias erosões unidimensionais (1D). As funções estruturantes usadas na erosão pertencem a uma família de quatro funções estruturantes direcionais formadas por dois pixels. Duas para erosões seqüencias e outras duas para erosões por propagação.

Como foi visto na Subseção 3.1.2, a função estruturante euclidiana b_{G_E} pode ser escrita como a soma de Minkowski em níveis de cinza de diversas funções estruturantes $b_i's \in \{b_1, \cdots, b_k\}$. Usando a formação dada pela Figura 4.6 é possível obter $b_{G_E} = b_1 \oplus \cdots \oplus b_k$ e $\varepsilon_{b_{G_E}}(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots)$.

Note que esses $b_i's$ podem ainda ser decompostos em quatro funções estruturantes direcionais de dois pixels, duas nas direções verticais, Norte (b_{Ni}), e Sul (b_{Si}), e duas nas direções horizontais, Leste (b_{Ei}), e Oeste (b_{Wi}):

$$b_{Ni} = \begin{bmatrix} -2i+1 \\ \mathbf{0} \end{bmatrix},$$

$$b_{Wi} = \begin{bmatrix} -2i+1 & \mathbf{0} \end{bmatrix}, \qquad b_{Ei} = \begin{bmatrix} \mathbf{0} & -2i+1 \end{bmatrix},$$

$$b_{Si} = \begin{bmatrix} \mathbf{0} \\ -2i+1 \end{bmatrix}. \tag{4.7}$$

A função estruturante para a TDE pode ser decomposta em:

$$\begin{aligned} b_E \;=\; & \cdots \oplus b_{N2} \oplus b_{N1} \oplus \cdots \oplus b_{S2} \oplus b_{S1} \oplus \cdots \\ & \cdots \oplus b_{W2} \oplus b_{W1} \oplus \cdots \oplus b_{E2} \oplus b_{E1}. \end{aligned} \tag{4.8}$$

Esta decomposição gera duas conseqüências importantes. Primeiro, a

TDE pode ser separada em mais erosões 1D simples, que traz independência para a computação de cada linha ou coluna da imagem. Segundo, as funções estruturantes são reduzidas para funções estruturantes direcionais de dois pixels, permitindo implementações simples e eficiente de algoritmos.

Isto significa que a TDE pode ser computada pela erosão de cada coluna da imagem por $b_{N1}, b_{N2}, \ldots$ até a estabilidade usando a propriedade idempotente, então pela erosão das colunas por $b_{S1}, b_{S2}, \ldots$, seguida pela erosão das linhas por $b_{E1}, b_{E2}, \ldots$, e finalmente, pela erosão das linhas por $b_{W1}, b_{W2}, \ldots$.

4.5.1 TDE unidimensional

Nos algoritmos paralelos, os pixels são processados independentemente da ordem de varredura. Os pixels alterados na transformação dependem apenas dos pixels da imagem de entrada e da função estruturante, como foi visto no Algoritmo 4 - *Erosão paralela*.

O Algoritmo 17 da TDE 1D apresentado a seguir é exato e um dos mais simples para o cômputo da distância euclidiana encontrados na literatura.

A ineficiência deste algoritmo ocorre devido à varredura desnecessária nas áreas da imagem onde a erosão não é afetada. A melhor eficiência pode ser verificada com algoritmos por propagação.

4.5.2 TDE por propagação unidimensional

A idéia do algoritmo de *erosão por propagação* está em processar somente os pixels que podem mudar na erosão. Este conjunto de pixels é chamado *fronteira* de f, denotado por $\partial(f, b)$. Neste caso, a fronteira para a próxima erosão é computada durante a erosão prévia. Como foi visto nos Algoritmos 8 e 13, da erosão por propagação e da TD por propagação, respectivamente.

Para as erosões usando a família de funções estruturantes decomposta

Algoritmo 17 TDE 1D clássica

$\text{TDE}^{1D} : K^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\quad \text{TDE}^{1D}(f) = g$
// f é imagem binária de entrada com valores 0 e M,
// onde M é o quadrado da distância máxima na imagem

$\forall\ c \subset \mathbb{E}$ // primeiro passo, erosões verticais da coluna c
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ r \subset \mathbb{E}$ // varre a coluna c
$\quad\quad\quad N(r, c) = \min(f(r, c), f(r - 1, c) + b)$
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ r \subset \mathbb{E}$
$\quad\quad\quad S(r, c) = \min(N(r, c), N(r + 1, c) + b)$
$\forall\ r \subset \mathbb{E}$ // segundo passo, erosões horizontais da linha r
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ c \subset \mathbb{E}$
$\quad\quad\quad E(r, c) = \min(S(r, c), S(r, c + 1) + b)$
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ c \subset \mathbb{E}$
$\quad\quad\quad g(r, c) = \min(E(r, c), E(r, c - 1) + b)$

// Considere $f(r - 1, c), N(r + 1, c), S(r, c + 1)$ e $E(r, c - 1)$ tratados
// na função min com valor M fora do domínio $\mathbb{E}$.

em dois pixels, é possível melhorar a eficiência do algoritmo por propagação. Estas melhorias são diferentes para cada um dos dois passos: a primeira *erosão seqüencial vertical* e a segunda *erosão por propagação horizontal*.

Parte 1

Para a primeira erosão seqüencial vertical será apresentado o Algoritmo 18[3].

Algoritmo 18 TDE 1D com erosão seqüencial vertical - parte 1

$\text{TDE}^{1D1} : K^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\text{TDE}^{1D1}(f) = g$
// f é imagem de entrada e saída, H é altura da imagem

$\forall$ $c \subset \mathbb{E}$ // coluna
b=1;
$\forall$ $r = 2..H$ // varre a coluna c na ordem *raster*
if $f(r,c) > f(r-1,c) + b$
$f(r,c) = f(r-1,c) + b$;
$b = b + 2$;
else;
$b = 1$;
$b = 1$;
$\forall$ $r = (H-1)..1$ // varre a coluna c na ordem *anti-raster*
if $f(r,c) > f(r+1,c) + b$
$f(r,c) = f(r+1,c) + b$;
$b = b + 2$;
else
$b = 1$;

Como a imagem de entrada é binária, a fronteira pode ser propagada na

[3]O Algoritmo 18 pode ser melhorado da seguinte forma: na varredura *raster*, quando $f(r,c) > f(r-1,c) + b$ for verdade, inicia-se um contador *count* (inicializado com 0) que finaliza quando a desigualdade assumir falso. A partir deste ponto inicia-se uma varredura *anti-raster* até *count*/2 utilizando o segundo **if** do algoritmo. Este processo se repete para todos os objetos e todas as colunas e é ilustrado na Figura 4.16.

mesma direção da função estruturante usando um processamento seqüencial de forma que o pixel modificado na transformação é uma função do pixel modificado previamente. Deste modo a erosão vertical pode ser computada usando uma única varredura *raster* e uma única varredura *anti-raster*.

Parte 2

Para a segunda parte, erosão por propagação horizontal, será apresentado o Algoritmo 19.

Nessa erosão horizontal, não é possível fazer um processamento seqüencial, mas é possível evitar a cópia das imagens e processar a imagem desde que a ordem de processamento seja a ordem oposta da propagação da fronteira. Quando se usa a direção Leste, a ordem *raster* deve ser Oeste, de forma que o pixel modificado não será usado naquela varredura. Isto é alcançado usando *fila FIFO - First-In-First-Out*, duas para cada direção. A eficiência deste algoritmo é melhorada pelo fato de que só os pixels de fronteira são processados até a estabilidade.

Note que as filas usadas neste algoritmo têm um tamanho de máximo na largura da imagem. Para melhor desempenho, estas filas podem ser facilmente implementadas por vetores de inteiro de comprimentos fixos.

Observe que nos Algoritmos 18 e 19 de TDE as erosão não estão definidas como funções separadas, como foi feito em todos os algoritmos até este ponto, pois o principal objetivo destes algoritmos é mostrar uma implementação eficiente. Estas duas versões estão no trabalho de Lotufo e Zampirolli [LZ01]. Contudo, no Algoritmo 18, o leitor pode ver que o laço $\forall \;\; r = 2..H$ faz a *erosão seqüencial* da coluna c na ordem *raster*, enquanto o laço $\forall \;\; r = (H-1)..1$ faz a *erosão seqüencial* da mesma coluna c na ordem *anti-raster*. O mesmo ocorre no Algoritmo 19, onde o laço **while** $notEmptyQueue(Eq)$ faz a erosão por propagação para a direita e **while** $notEmptyQueue(Wq)$ faz a *erosão por propagação* para a esquerda, até que as estruturas Eq e Wq fiquem vazias.

Algoritmo 19 TDE 1D com erosão por propagação horizontal - parte 2

$\text{TDE}^{1D2} : K^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\text{TDE}^{1D2}(f) = g$
// f é imagem de entrada e saída, W é a largura da imagem

```
∀ r ⊂ 𝔼, // linha
    ∀ c = (W − 1)..1
        insertQueue(Eq, c);
    ∀ c = 2..W
        insertQueue(Wq, c);
    b = 1;
    while (notEmptyQueue(Wq) ou notEmptyQueue(Eq))
        while notEmptyQueue(Eq)
            c = fromQueue(Eq)
            if f(r, c + 1) > f(r, c) + b
                f(r, c + 1) = f(r, c) + b;
                if c + 1 < W
                    insertQueue(Eq2, c + 1);
        while notEmptyQueue(Wq)
            c = fromQueue(Wq);
            if f(r, c − 1) > f(r, c) + b
                f(r, c − 1) = f(r, c) + b;
                if c − 1 > 1
                    insertQueue(Wq2, c − 1);
        b = b + 2;
        Wq = Wq2;
        Eq = Eq2;
```

Para melhor ilustrar o algoritmo da TDE multidimensional usando erosões serão apresentados a seguir alguns exemplos.

Exemplo 1

Neste primeiro exemplo é considerado apenas uma transformação 1D para melhor entendimento da erosão morfológica usada para computar a métrica euclidiana. A Figura 4.13 ilustra a erosão de f pela função estruturante b_{G_E}. A Figura 4.14 ilustrada a soma de Minkowski de b_1 por b_2. A generalização forma $b_{G_E} = b_1 \oplus \cdots \oplus b_k$. A Figura 4.15 ilustra a TDE obtida pela composição de erosões.

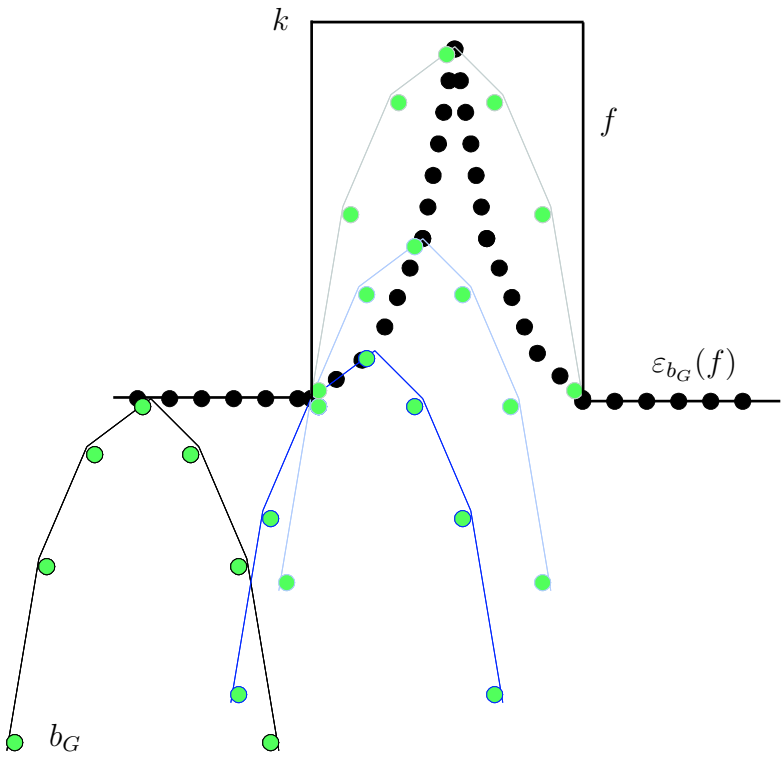


Figura 4.13: Ilustração da TDE 1D obtida da erosão pela função estruturante b_{G_E}.

$$b_i = \boxed{-2i+1} \boxed{\mathbf{0}} \boxed{-2i+1}$$

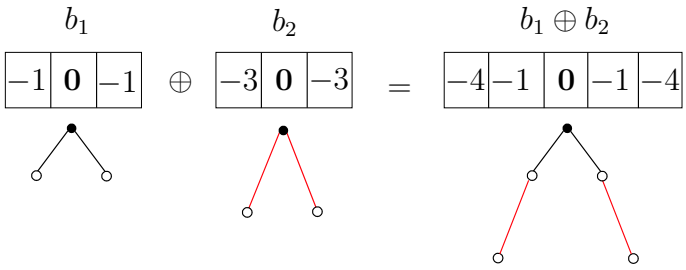


Figura 4.14: Soma de Minkowski em níveis de cinza para a métrica euclidiana, onde as origens estão no centro de cada função estruturante.

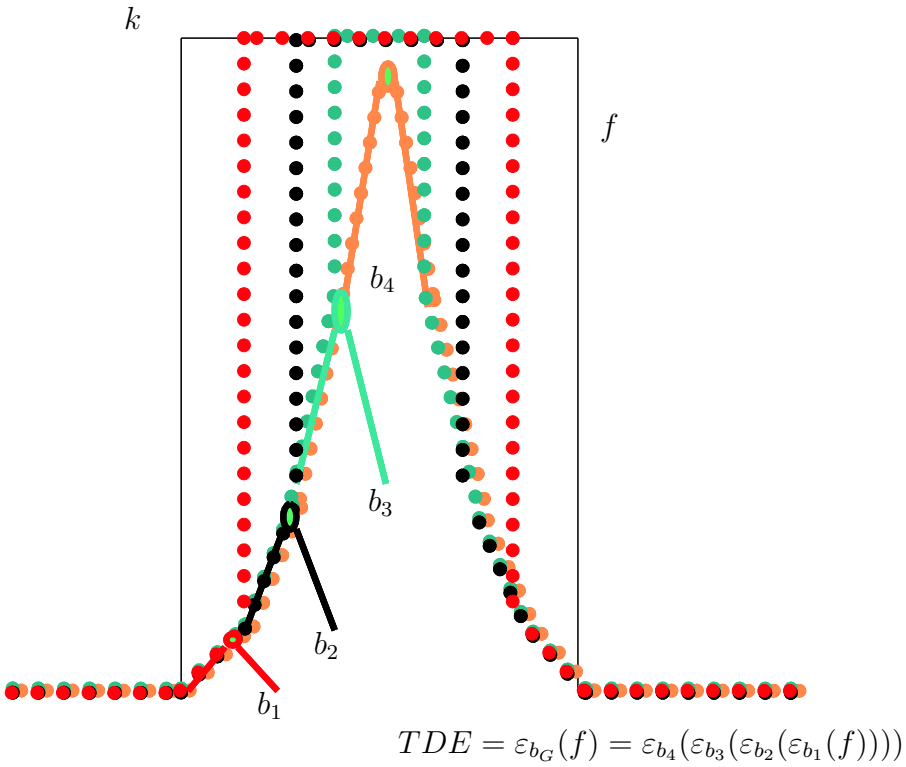
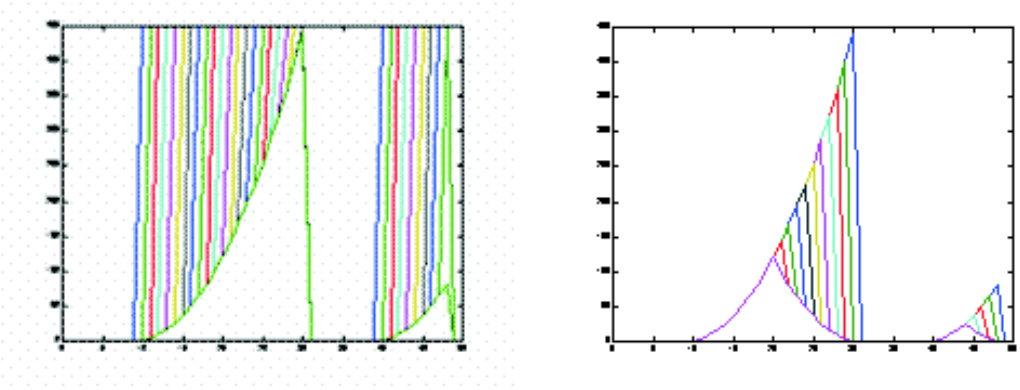


Figura 4.15: Ilustração da TDE 1D obtida de erosões por decomposição de função estruturante.

Exemplo 2

Para visualizar as transformações ocorridas nas quatro erosões do algoritmo multidimensional, a Figura 4.16 ilustra o primeiro passo do algoritmo, onde são computadas as erosões seqüenciais *raster* e *anti-raster*. Enquanto que a Figura 4.17 ilustra do segundo passo deste algoritmo, onde são realizadas as erosões por propagação.



raster *anti-raster*

Figura 4.16: Ilustração do primeiro passo do algoritmo multidimensional.

Exemplo 3

Finalmente, um exemplo simples usando uma imagem f de tamanho 4×4 é mostrado na Figura 4.18. O resultado da erosão vertical pela função estruturante B_v é apresentado na função f_v. Em seguida, f_1 é a primeira erosão por propagação horizontal e f_2 é a segunda erosão por propagação horizontal representando o quadrado da distância euclidiana. Nos resultados das erosões horizontais, os pixels inseridos na fila de propagação são marcados em negrito.

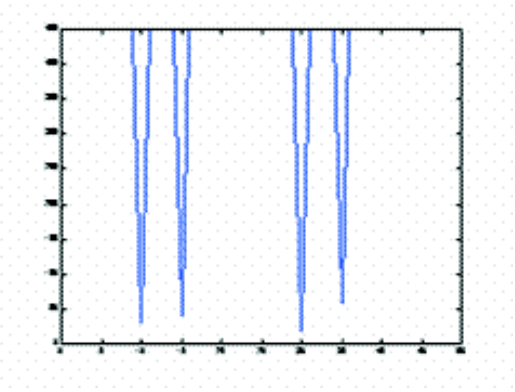
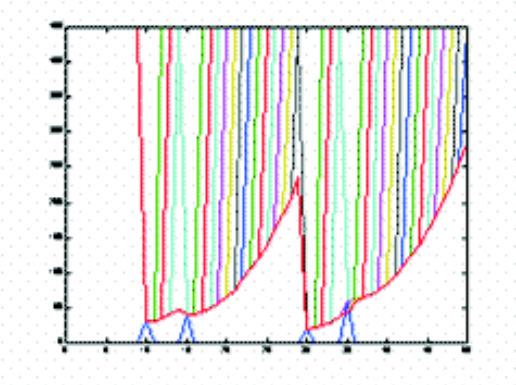
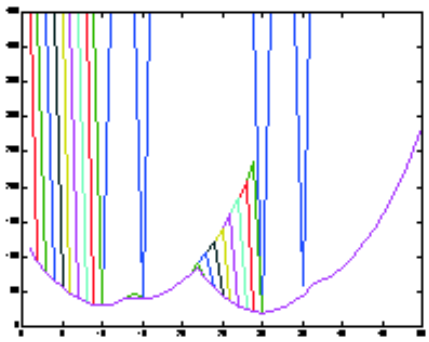


Imagem de Entrada



East



West

Figura 4.17: Ilustração do segundo passo do algoritmo multidimensional.

$$f = \begin{pmatrix} \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ 0 & \infty & \infty & \infty \end{pmatrix}, \qquad B_v = \begin{pmatrix} -5 \\ -3 \\ -1 \\ 0 \\ -1 \\ -3 \\ -5 \end{pmatrix},$$

$$f_v = f \ominus B_v = \begin{pmatrix} 9 & 4 & 0 & \infty \\ 4 & 1 & 1 & \infty \\ 1 & 0 & 4 & \infty \\ 0 & 1 & 9 & \infty \end{pmatrix},$$

$$B_{h1} = \begin{pmatrix} \text{-1} & 0 & \text{-1} \end{pmatrix}, \qquad f_1 = f_v \ominus B_{h1} = \begin{pmatrix} \mathbf{5} & \mathbf{1} & 0 & \mathbf{1} \\ \mathbf{2} & 1 & 1 & \mathbf{2} \\ 1 & 0 & \mathbf{1} & \mathbf{5} \\ 0 & 1 & \mathbf{2} & \mathbf{10} \end{pmatrix},$$

$$B_{h2} = \begin{pmatrix} -3 & 0 & -3 \end{pmatrix}, \qquad f_2 = f_1 \ominus B_{h2} = \begin{pmatrix} \mathbf{4} & 1 & 0 & 1 \\ 2 & 1 & 1 & 2 \\ 1 & 0 & 1 & \mathbf{4} \\ 0 & 1 & 2 & \mathbf{5} \end{pmatrix}.$$

Figura 4.18: Aplicação da TDE multidimensional.

	Algoritmos			**Métricas**					
DT	*Paralelo*	*Seqüencial*	*Propagação*	b_4	b_8	b_O	$b_{3:4}$	$b_{5:7:11}$	b_E
[RP66]		X		X					
[RP68]	X			X	X	X			
[Bor86]	X	X		X	X	X	X	X	X
[SM92]	X			X	X		X	X	X
[Vin92]	X	X	X	X	X	X			
[HM94]	X								X

Tabela 4.1: Classificação de algoritmos da TD e de suas decomposições de funções estruturantes.

4.6 Resumo da classificação da TD usando erosões

A Tabela 4.1 resume a classificação dos principais algoritmos da TD nos padrões paralelo, seqüencial e por propagação e suas decomposições de funções estruturantes $b_4, b_8, b_O, b_{3:4}, b_{5:7:11}$ e b_E. Esta tabela também foi apresentada no artigo [ZL00].

Note que não foram incluídos nesta classificação os algoritmos para a TD por propagação direcional e a TDE multidimensional pois são casos particulares dos padrões apresentados neste texto.

4.7 Análise de desempenho e comparações

Serão apresentadas nesta seção as análises e as comparações dos algoritmos da TDE paralelo, por propagação, por propagação direcional e multidimensional vistos nas Seções 4.3, 4.4 e 4.5, respectivamente.

Comparando o algoritmo da TDE obtido por erosões direcionais, proposto na Seção 4.4, com o algoritmo do Eggers [Egg98] é possível notar que este segundo é mais eficiente (veja Tabela 4.2), isto porque não existem cópias de imagens nas iterações por conseqüência dos vários testes existes. A simplici-

	Imagem 1	**Imagem 2**
Saito	80	60
Eggers	100	90
PMN	90	80
Meijster	121	120
LZ	90	101
TDE^{Dir}	190	210
TDE^{Pro}	230	240
TDE^{Par}	1012	701

Tabela 4.2: Tempo em mili-segundos do desempenho de diversos algoritmos da TDE.

dade do algoritmo direcional proposto nesta tese elimina estes testes, porém para o cálculo da TDE é necessário fazer a cópia dos pixels de fronteira em cada iteração, que torna o nosso algoritmo ineficiente. Isto não ocorre quando se usa funções estruturantes constantes, como para as métricas *chanfradas*.

A Tabela 4.2 apresenta um resumo dos testes feitos para o cálculo da TDE de diversos algoritmos da literatura e também dos algoritmos apresentados nesta tese. Para estes testes foi usado um *Pentium III* $-$ 800 MHz $-$ 256 MB. A **Imagem 1** e a **Imagem 2** são imagens 512×512 ilustradas na Figura 4.22 (a) e (b), respectivamente. As implementações dos três primeiros algoritmos apresentados na tabela (Saito [ST94], Eggers [Egg98] e PMN [Cui99]) foram extraídas da *home page* do Olivier Cuisenaire. O algoritmo Meijster é um dos mais recentes da literatura [MR00]. O algoritmo LZ é o multidimensional definido na Seção 4.5 e os restantes foram definidos nas Seções 4.3 e 4.4.

O bom desempenho dos algoritmos Saito [ST94] e PMN [Cui99] se deve ao fato de ambos usarem uma *lookup table* para armazenar o mapa da distância euclidiana [Cui99], veja Figura 4.19. Este recurso não foi incluído nos algoritmos propostos nesta tese, pois não iriam mais ser definidos explicitamete por erosões, que é a essência desta tese.

Meijster e Roerdink [MR00] definiu um dos algoritmos da TDE mais recentes, onde possui o primeiro passo igual ao algoritmo multidimensional

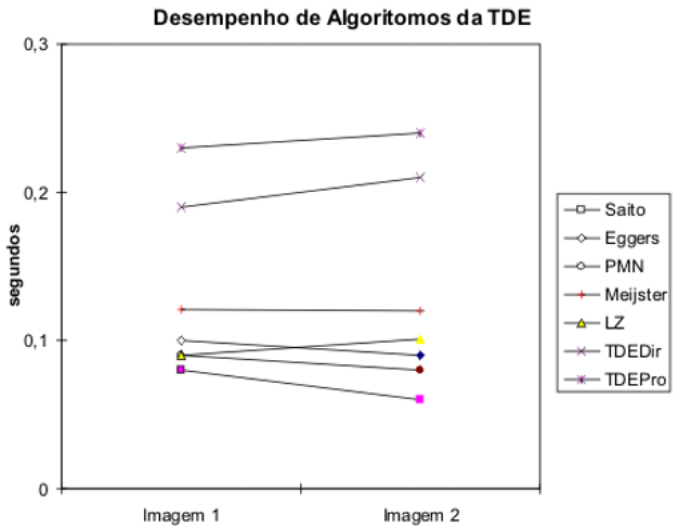


Figura 4.19: Gráfico ilustrando os desempenhos de algoritmos da TDE referente à Tabela 4.2.

proposto nesta tese. No segundo passo o algoritmo do Meijster e Roerdink usa uma multiplicação para cada pixel conferir a região dominante mais próxima. A vantagem do algoritmo multidimensional é que ele não exige multiplicação, só comparação e adição, que faz do algoritmo proposto um dos mais rápido na maioria das situações.

O primeiro passo do algoritmo multidimensional exige uma varredura *raster* e outra *anti-raster* com vizinhança de dois pixels cada. O desempenho do segundo passo é mais complexo. A melhor situação exige uma varredura *raster* e outra *anti-raster* com vizinhança de dois pixels, que acontece com uma imagem com colunas iguais, onde nenhuma propagação horizontal é exigida. Em um caso típico, a velocidade depende do número de vezes que o pixel entra na fila de propagação.

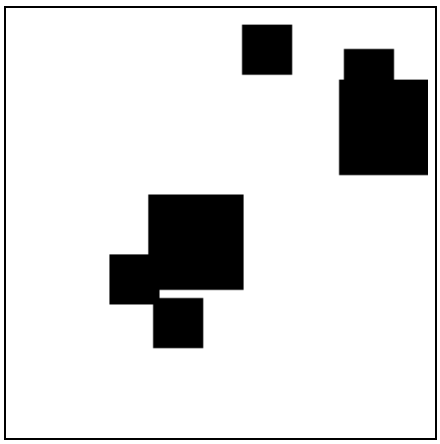
O algoritmo multidimensional proposto nesta tese tem uma patologia de pior caso que acontece com uma imagem quadrada com uma linha de fundo diagonal, veja Figura 4.22c. Nesta situação o número de vezes que um pixel é inserido na fila é aproximadamente $W/4$, onde W é a largura da imagem. Embora isto faça este algoritmo quadrático com relação às dimensões da imagem, isto dificilmente ocorre em imagens típicas. Uma análise mais precisa pode ser observada através da Figura 4.20, onde os valores mostram o número vezes que um pixel é inserido na fila no segundo passo do algoritmo multidimensional. Observe que no pior caso cada linha possui duas progressões aritméticas, assim o algoritmo possui complexidade $O(HW^2)$, onde W é a largura e H é a altura da imagem.

Para ilustrar o comportamento dos algoritmos com imagens de tamanhos diferentes, a imagem *box* (veja Figura 4.21) foi reproduzida por um fator de 4, 8, 16 e 64 e uma comparação de tempo foi feita usando a TDE definida por Meijster e Roerdink [MR00], veja Tabela 4.3.

Dos resultados apresentados nas Tabela 4.2 e 4.3, pode ser concluído que o algoritmo proposto tem um desempenho melhor que um dos mais rápidos da TDE e um desempenho comparável com algoritmos mais conhecidos da

4	4	3	3	2	2	1	1	0	0
4	3	3	2	2	1	1	0	0	0
3	3	2	2	1	1	0	0	0	1
3	2	2	1	1	0	0	0	1	1
2	2	1	1	0	0	0	1	1	2
2	1	1	0	0	0	1	1	2	2
1	1	0	0	0	1	1	2	2	3
1	0	0	0	1	1	2	2	3	3
0	0	0	1	1	2	2	3	3	4
0	0	1	1	2	2	3	3	4	4

Figura 4.20: Matriz representado o número de vezes que um pixel é inserido na fila no segundo passo do algoritmo multidimensional aplicado na imagem de pior caso (Figura 4.22c) de tamanho 10×10.



nbox

Figura 4.21: Imagem 256×256 utilizada para teste de eficiência para os algoritmos da TDE, onde a cor preta é 0 e a cor branca é $k \neq 0$.

	64K	256K	1M	4M
LZ	0.022	0.153	0.659	2.81
Meijster	0.028	0.181	0.763	3.08

Tabela 4.3: Tempo em segundos do algoritmo proposto e o do definido por Meijster e Roerdink [MR00] aplicados para a imagem *box* reproduzida por um fator de $4, 16, 32$ e 64.

distância euclidiana.

As experiências da Tabelas 4.3 foram feitas em um *notebook Pentium III,* $750MHz$*,* $128MB$*.*

Referências Bibliográficas

[BBL94] J. Barrera, G.F. Banon e R.A. Lotufo. A mathematical morphology toolbox for the KHOROS system. No *Image Algebra and Morphological Image Processing V*, páginas 241–252, Bellingham, Julho 1994. SPIE.

[Bor86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[Cui99] O. Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing.* Tese de Doutorado, Université Catholique de Louvain, Bélgica, 1999.

[Dan80] P.E. Danielsson. Euclidean distance mapping. *Computer Vision, Graphics and Image Processing*, 14:227–248, 1980.

[EBS01] E. Engbers, R.v.d. Boomgaard e A.W.M. Smeulders. Decomposition of separable concave structuring functions. *Journal of Mathematical Imaging and Vision*, 15(3), 2001.

[Egg98] H. Eggers. Two fast Euclidean distance transformations in $\mathbb{Z}^2$ based on sufficient propagation. *Computer Vision, Graphics and Image Processing*, 69(1), 1998.

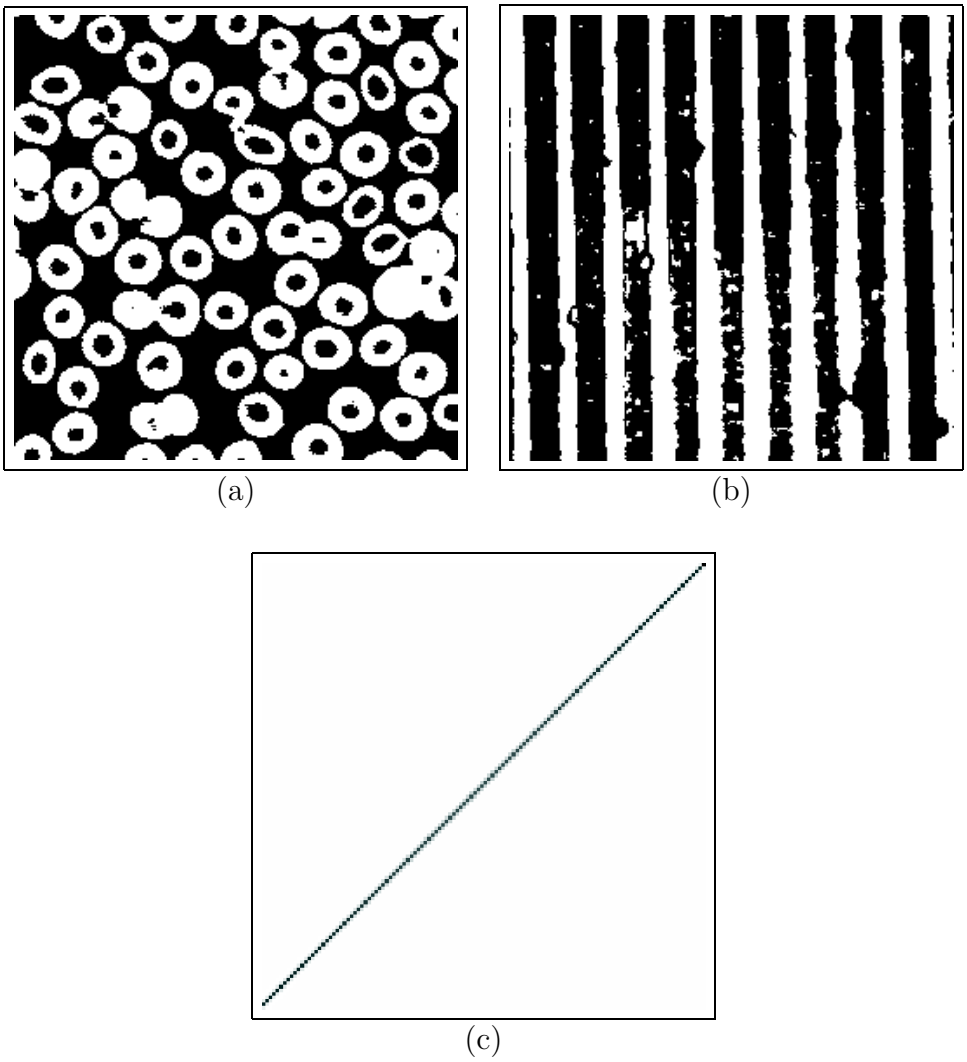


Figura 4.22: (a) e (b) imagens reais; (c) pior caso para a TDE multidimensional.

[HM94] C.T. Huang e O.R. Mitchell. A Euclidean distance transform using grayscale morphology decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:443–448, 1994.

[LFZ02] R.A. Lotufo, A.A. Falcão e F.A. Zampirolli. Ift-watershed from gray scale marker. No *Brazilian Symposium on Computer Graphics and Image Processing*, Fortaleza, RN, Brasil, Outubro 2002.

[LT00] N.J. Leite e M.D. Teixeira. An idempotent scale-space approach for morphological segmentation. No *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, páginas 341–350, Palo Alto, USA, Junho 2000.

[LZ01] R.A. Lotufo e F.A. Zampirolli. Fast multidimensional parallel Euclidean distance transform based on mathematical morphology. No *Brazilian Symposium on Computer Graphics and Image Processing*, páginas 100–105, Florianopolis, Brasil, Outubro 2001.

[MR00] A. Meijster e J.B.T.M. Roerdink. A general algorithm for computing distance transforms in linear time. No *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, páginas 331–340. Palo Alto, USA, Junho 2000.

[Rag92] I. Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *Computer Vision, Graphics and Image Processing: Image Understanding*, 56(3), 1992.

[RP66] A. Rosenfeld e J.L. Pfalz. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, 13(4):471–494, Outubro 1966.

[RP68] A. Rosenfeld e J.L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[SC94] Y. Sharaiha e N. Christofides. Graph-theoretic approach to distance transformations. *Pattern Recognition Letters*, 15(10), 1994.

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

[SM92] F.Y.-C. Shih e O.R. Mitchell. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing*, 1:197–204, 1992.

[ST94] T. Saito e J. I. Toriwaki. New algorithms for Euclidean distance transformations of an n-dimensional digitized picture with applications. *Pattern Recognition*, 27:1551–1565, 1994.

[Vin92] L. Vincent. Morphological algorithms. No E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, capítulo 8, páginas 255–288. Marcel Dekker, New York, 1992.

[Yam84] H. Yamada. Complete euclidean distance transformation by parallel operation. páginas 69–71, 1984.

[Zam97] F.A. Zampirolli. Operadores morfológicos baseados em grafos de vizinhanças – uma extensão da MMach toolbox. Dissertação de Mestrado, Unversidade de São Paulo, São Paulo, Brasil, Abril 1997.

[ZL00] F.A. Zampirolli e R.A. Lotufo. Classification of the distance transformation algorithms under the mathematical morphology approach. No *Brazilian Symposium on Computer Graphics and Image Processing*, Gramado, RS, Brasil, Outubro 2000.

Capítulo 5

Conclusão

Os trabalhos de Mitchell e colaboradores [SM92, HM94] nos inspiraram a estudar e classificar algoritmos clássicos da transformada de distância (TD) com o enfoque de erosões morfológicas nos padrões paralelo, seqüencial e por propagação.

Como resultado destes estudos, foram apresentadas equivalências restritas entre os padrões paralelos e seqüenciais, sendo divididos em dois casos: no primeiro, se as somas de Minkowski de decomposições *raster* e *anti-raster* de funções estruturantes são estáveis em uma janela finita, então existe a equivalência entre a erosão paralela, pela união das somas de Minkowski das decomposições *raster* e *anti-raster*, e a intersecção das erosões seqüenciais pelas decomposições *raster* e *anti-raster*; no segundo caso, existe a equivalência quando uma seqüência de somas de Minkowski de funções estruturantes se estabilizam em uma janela finita, quando isto ocorre é possível obter a equivalência entre uma erosão paralela pela função estruturante resultante desta soma e a composição de duas erosões seqüenciais, uma *raster* e outra *anti-raster*. Neste segundo caso também é possível calcular a TD usando estas erosões seqüencias, porém não são aplicáveis para a métrica euclidiana. Para o padrão por propagação foi observado que, quando uma função estruturante é decomposta por uma seqüência não crescente de funções es-

truturantes, é possível obter a equivalência entre os padrões paralelo e por propagação. Todo esse estudo de equivalência foi útil para melhorar a eficiência da TD.

Na classificação da TD, foi apresenta uma nova formulação para a TD seqüencial e para a TD por propagação. Na TD por propagação também foi possível calcular a métrica euclidiana (TDE). Com esta classificação foi possível reescrever algoritmos clássicos da TD através de erosões morfológicas, destacando a reescrita do algoritmo do Vincent [Vin92].

Como resultados adicionais, dois novos algoritmos foram apresentados: TD por propagação direcional e TDE multidimensional. O primeiro algoritmo também encontra a TDE, quando é usado a decomposição de função estruturante definida por Huang e Mitchell [HM94]. O segundo algoritmo é baseado em erosões morfológicas por uma família de decomposições de funções estruturantes direcionais de tamanho 2. A decomposição unidimensional de função estruturante permite independência de linhas e colunas fazendo o algoritmo muito satisfatório para processamento paralelo e facilmente extensível para dimensões mais altas. O algoritmo de erosão por propagação usa uma fila de propagação de tamanho fixo permitindo implementações simples e eficientes. Como o algoritmo trabalha por erosão morfológica, apenas comparações e adições são usadas.

Foi confirmado que a *morfologia matemática* é muito satisfatória para a compreensão e projeto de algoritmos eficientes para a TD. Os algoritmos apresentados neste trabalho são os mais simples para a TD disponíveis na literatura, tanto para codificação quanto para compreensão. O algoritmo da TDE multidimensional apresentou desempenho de cerca de 9 acessos por pixel envolvendo somente adições, sendo um dos mais rápidos algoritmos da TDE conhecido na literatura.

Trabalhos futuros

Como sugestões para pesquisas futuras, seria interessante estudar a relação entre k e as dimensões e níveis de cinza de uma imagem $f \in [0, k]^{\mathbb{E}}$ e uma função estruturante b para determinar a equivalência restrita entre os padrões paralelos e seqüenciais da erosão e não simplesmente assumir k suficientemente grande ou tendendo para o infinito.

Um trabalho interessante que estuda as condições para que dilatações e erosões em espaço-escala morfológico sejam idempotentes foi realizado por Leite e Teixeira [LT00]. Neste trabalho também existe um resultado para determinar o número de iterações necessárias, entre escalas, para que a idempotência seja calcula. Este resultado seria o ponto de partida para determinar a relação entre k e as dimensões e níveis de cinza de f e b na equivalência restrita entre os padrões paralelos e seqüenciais da erosão.

Também como sugestões para trabalhos futuros, seria interessante implementar os resultados de Vliet e Verwer [VV88] de operadores por propagação para imagens em níveis de cinza, incluindo o trabalho da dilatação por propagação definida por Barrera e Hirata [BHJ97] e os resultados de dilatações e erosões em espaço-escala morfológico realizados por Leite e Teixeira [LT00], tudo isso para o espaço multidimensional. Como resultado deste estudo, seria possível implementar vários operadores existentes em morfologia matemática nos padrões paralelo, seqüencial e por propagação definidos nesta tese.

Referências Bibliográficas

[BHJ97] J. Barrera e R. Hirata Jr. Fast algorithms to compute the elementary operators of mathematical morphology. No *Brazilian Symposium on Computer Graphics and Image Processing*, páginas 163–170, São Paulo, Brasil, Outubro 1997.

[HM94] C.T. Huang e O.R. Mitchell. A Euclidean distance transform us-

ing grayscale morphology decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:443–448, 1994.

[LT00] N.J. Leite e M.D. Teixeira. An idempotent scale-space approach for morphological segmentation. No *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, páginas 341–350, Palo Alto, USA, Junho 2000.

[SM92] F.Y.-C. Shih e O.R. Mitchell. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing*, 1:197–204, 1992.

[Vin92] L. Vincent. Morphological algorithms. No E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, capítulo 8, páginas 255–288. Marcel Dekker, New York, 1992.

[VV88] L.J. van Vliet e B.J.H. Verwer. A contour processing method for fast binary neighbourhood opeations. *Pattern Recognition Letters*, 7(1):27–36, 1988.

Apêndice A

Ambiente mmil

> *Uma linguagem-fonte de sucesso está inclinada a ser implementada em várias máquinas-alvo. Se a linguagem sobreviver, os compiladores para a mesma necessitarão gerar código para várias gerações de máquinas-alvo... por conseguinte, compiladores reorientáveis estão inclinados a desempenhar algum papel. Logo, o projeto de linguagens intermediárias é importante, na medida em que confina detalhes específicos de máquina para um pequeno número de módulos.* *[ASU95]*

Neste apêndice será apresentado o ambiente de geração automática de códigos e documentos para operadores morfológicos, chamado *mathematical morphology intermediary language*, ou *mmil*. Este ambiente faz uso de elementos de alto desempenho que definem a nossa *linguagem intermediária*, também apresentada neste apêndice. O ambiente *mmil* foi desenvolvido no *Adesso*, um ambiente computacional de suporte ao desenvolvimento de aplicações científicas como uma caixa de ferramentas [Mac02].

Para explicar como funciona o ambiente *mmil*, este apêndice será dividido da seguinte forma: na Seção A.1 será apresentada a motivação para criação deste ambiente; na Seção A.2 será apresentada uma introdução ao ambiente desenvolvido, comentando as linguagens de programação utilizadas

e descrevendo a sua arquitetura, destacando a Figura A.2; na Seção A.3 será apresentada a *linguagem intermediária*; na Seção A.4 serão descritos os elementos utilizados pelos operadores que compõem a *linguagem intermediária* do ambiente *mmil*; nas Seções A.5, A.6 e A.7, serão apresentados alguns exemplos ilustrativos da programação usando os elementos definidos, mostrando nas duas primeiras seções os códigos correspondentes em *XML* e *MATLAB*.

Nos exemplos da Seção A.7 não serão mostrados os códigos *XML* e *MATLAB*, pois eles não são códigos para leitores humanos, mas para as máquinas interpretarem. Serão mostradas apenas as expressões matemáticas equivalentes, que podem ser geradas automaticamente através dos códigos em *XML*.

A.1 Motivação

Serão apresentadas nesta seção as várias linhas de pesquisa para o desenvolvimento de software para processamento de imagens. Serão analisados os paradigmas de programação e a organização da informação no processo de desenvolvimento de software; serão apresentados a evolução e os problemas de uma ferramenta particular de processamento de imagens, chamada inicialmente de *MMach*; serão apresentadas também as soluções encontradas na literatura para resolver parte destes problemas; finalmente, será apresentada a nossa solução para estes problemas.

A análise e o processamento de imagens por computador digital possibilita resolver problemas em várias áreas da ciência [BB94, GW92]. Porém, a criação e o uso de ferramentas de processamento de imagens pode muitas vezes impedir o sucesso de uma aplicação devido à complexidade, às limitações e à grande quantidade de transformações e parâmetros existentes [D'O01]. Em *transformações* serão incluídas todas as *operações* (soma, subtração, complemento, etc.), *operadores* (dilatação, erosão, etc.) e *transformações geométricas* (translação, rotação, etc.).

Programação genérica

O estado da arte em linguagens de programação é a *programação genérica* [Aus99]. Assim como o conceito de orientação a objetos revolucionou a programação no início da década de 90, a programação genérica está revolucionando a programação no início desta década, onde o objetivo é generalizar os algoritmos de forma a facilitar a sua reutilização.

O paradigma da *programação genérica* permite construir algoritmos abstratos encapsulando os tipos e estruturas de dados. A linguagem de programação *C++* possibilita a escrita destes tipos de algoritmos usando a biblioteca *STL* (*Standard Template Library*). O paradigma da *programação orientada a objetos* permite abstrair estruturas de dados encapsulando os algoritmos. Outro paradigma bem conhecido dos programadores é a *programação estruturada*, onde os esforços estão em implementar os mesmos para diferentes estruturas e tipos de dados e isto implica na multiplicação de esforços. A programação genérica busca distribuir as atividades de implementação de algoritmos usando diferentes estruturas e tipos de dados.

Essas três atividades de programação podem ser simbolizadas pelo eixo octagonal tridimensional, onde os esforços de construção de estruturas de dados estão no eixo X, dos tipos de dados estão no eixo Y e dos algoritmos estão no eixo Z. Segundo D'Ornellas *et. al.* [DCB+02], *um dos primeiros registros de iniciativas que buscaram organizar e estabelecer os reais limites e elementos envolvidos na programação geral coube à Wirth [Wir76] ao propor que houvesse uma separação semântica no desenvolvimento de programas, tratando algoritmos e estruturas de dados em separado e entendendo que a união de ambas é que viria a definir o programa.* Por esta análise, se os esforços de desenvolvimento forem separados, um esforço total de $X + Z$ é encontrado para os algoritmos e estruturas de dados. Com a evolução da programação percebeu-se que as atividades relacionadas ao tratamento de tipos de dados devem assumir uma atividade distinta. Assim, se essas três atividades forem tratadas em separado e se todas as possibilidades de imple-

mentação forem cobertas, os esforços para a programação são encontrados em $X + Y + Z$. Se isto não ocorrer, é obtido no pior caso um esforço total de $X * Y * Z$ para cobrir todas as possibilidades de implementação. Isto pode ser facilmente verificado na programação estruturada, onde são multiplicados os códigos quando são mudados as estruturas e tipos de dados. O paradigma da programação que ajuda a obter este esforço linear nas atividades de programação é chamado de *programação genérica*, onde as estruturas e tipos de dados estão encapsulados nos algoritmos.

Indo mais além nessa análise da evolução da programação, será analisado a construção de um software e não simplesmente as atividades de programação em uma linguagem específica, não descartando os esforços e as descobertas alcançadas na programação. Considerando um software formado por programas e documentações, os programas podem ser definidos em diferentes linguagens e os documentos podem ser escritos também usando diferentes editores. Uma sugestão para distribuir ainda mais as atividades para a construção de software seria definir mais uma ou duas dimensões no eixo octagonal tridimensional incorporando as atividades de linguagens de programação e editoração. Por exemplo, considere os eixos W para as linguagens de programação e K para os editores de texto. Assim, considere a união dos esforços para o desenvolvimento de um software distribuídos nos eixos X, Y, Z, W e K. Se foi possível, com muito esforço, construir um software na linguagem $C++ \in W$ em $X + Y + Z$, então, se for preciso mudar de linguagem, o software deverá ser reescrito na nova linguagem e torcer para que esta suporte também um esforço de $X + Y + Z$. Analogamente para a documentação escrita em $LaTeX \in K$. O ideal seria definir uma estrutura para armazenamento de informação incorporando algoritmos e documentação, como descrito na próxima seção, e que a compilação desta estrutura gerasse códigos em diferentes linguagens de programação e em diferentes formas de documentações.

Organização da informação

Uma boa metodologia no processo de desenvolvimento de software é definir uma estrutura de armazenamento que possibilite processar os seguintes conceitos [McG99]:

Conteúdo é a informação em si;

Estrutura define a organização da informação;

Apresentação associa a forma de consumir a informação.

É de consenso que se estas três partes são separadas uma da outra, uma melhor utilização das informações é alcançada. Uma boa ilustração desses conceitos é realizada no sistema de editoração conhecido como *LaTeX*. O conteúdo é armazenado em um arquivo texto, a estrutura é armazenada em um arquivo de estilo (*book, article, report*, etc.) e a saída é alcançada pelo processador *Tex*. Em contraste, se um autor escrever seu documento usando somente *Tex*, a reutilização do documento é perdida. Por outro lado, é possível converter arquivos *LaTeX* para outros formatos.

Recentemente, com a proliferação da Internet e a necessidade de ter uma ferramenta eficiente de manipulação da informação, surgiu a linguagem *XML* (*EXtensible Markup Language*), onde o conteúdo é armazenado em uma linguagem de marcação, tal como em *HTML*, mas com a possibilidade de criar novos *tags*. A estrutura é definida através de *esquema* (ou *scheme*), que restringe o conteúdo da informação, como aceitar apenas números inteiros em um certo campo. Finalmente, a apresentação é definida através de *folhas de estilo* (ou *stylesheets*), que governam a tradução da informação para um formato de saída.

A.1.1 *MMach*

A *morfologia matemática* é uma das subáreas do processamento de imagens e se baseia em teoria dos conjuntos através de quatro classes de operadores elementares: dilatação, erosão, anti-dilatação e anti-erosão.

Uma implementação de transformações para morfologia matemática foi iniciada em 1992 com o projeto *MMach* (*Morphological Machine*) [BBL94]. Durante a evolução deste projeto foi criada uma biblioteca na linguagem *C*, denominada *MMachLib* [LZHJB97]. Serão apresentadas a seguir as principais características desta biblioteca.

Hierarquia

A *MMachLib* foi concebida de forma hierárquica. Então, muitos dos operadores são formados pela composição de operadores elementares e operações básicas de união e intersecção. Esta característica facilita o desenvolvimento de software [BB94]. Tipicamente as bibliotecas de processamento de imagens têm centena de transformações. Este fato complica a manutenção e cresce a probabilidade de ocorrência de erros na programação, devido à proliferação de código. No caso da *MMachLib*, as transformações hierárquicas foram exploradas com o objetivo de diminuir o volume de código e minimizar a probabilidade de ocorrência destes erros.

Não-hierarquia

O surgimento de algoritmos não-hierárquicos, com implementações com menor tempo de processamento, obrigou a *MMachLib* possuir rotinas especializadas. Em particular, rotinas que utilizam estruturas de filas mostraram-se muito eficientes, como é o caso do *watershed* e da *transformada de distância*. Outro exemplo é a *abertura por área*, em imagens em níveis de cinza, que utiliza filas hierárquicas. Outros algoritmos que estão fornecendo bons resultados são os seqüenciais, devido a sua eficiência, porém possuem a

desvantagem de não serem intuitivos comparados com os algoritmos paralelos.

Obsolescência de recursos

Com o avanço da tecnologia, o software e o hardware estão ficando obsoletos num período de tempo cada vez mais curto. Sistemas operacionais como *MS-Windows* ou *Unix* e programas como *MATLAB* mudam de versão em média a cada três anos, ou menos. Quando isto ocorre, geralmente mudanças nas ferramentas que usam estas plataformas são necessárias. Na prática, quando um software não possui uma boa metodologia, tais mudanças significam muitas vezes implementar tudo de novo! Existe hardware específico para o processamento de imagens e para a morfologia matemática, hardware aceleradores nas *CPU's* convencionais como o *MMX* do *Pentium*, e também a possibilidade de se explorar eficientemente o uso do processamento em paralelo. Para um melhor aproveitamento do hardware, as empresas de software criam novos programas, com desempenho bem superior às versões anteriores. Assim sendo, é preciso atualizar freqüentemente os aplicativos e isto evidencia a necessidade de criar ferramentas capazes de minimizar a árdua tarefa de reescrita de código. Quando isto não ocorre, o custo de reescrita dos códigos pode ser alto e muitas vezes inviabilizar um projeto de hardware especial. O ideal seria com pouco esforço gerar novos códigos que executassem, de modo eficiente, nas diversas arquiteturas disponíveis.

A.1.2 Questões

Este apêndice contribui para responder de forma eficiente as seguintes questões: seria possível criar uma *linguagem intermediária* para escrever operadores morfológicos, e como resultado ter geração automática de código em várias linguagens de programação, junto com suas documentações?

A.1.3 Soluções encontradas na literatura

Apply e *Adapt*

Uma linguagem de programação independente de arquitetura para visão de baixo nível, chamada *Apply*, foi definida por Hamey *et. al.* [HWIC89]. Esta linguagem reduz o problema de escrita de algoritmos para visão de baixo nível, mas não pode ser usada em algoritmos globais, como *histograma* de imagens. A compilação da *Apply* converte um simples procedimento em uma implementação que pode ser executada eficientemente em linguagens como *C*. A linguagem *Adapt* foi definida em seguida por Webb [Web90], onde o processamento de imagens global foi incluído, baseado no modelo de divisão e conquista. Um estudo do desempenho da linguagem *Apply* foi feito por Walace *et. al.* [WWIC89] para vários problemas de visão computacional em máquinas paralelas.

Horus

Horus é um ambiente de processamento de imagens escrito na linguagem *C++* que usa a biblioteca *STL* (*Standard Template Library*) e é baseado na programação genérica e na programação orientada a objetos [KPS00]. Este software classifica as transformações de processamento de imagens em padrões: *pontual paralelo* (negação, complemento, etc); *binário paralelo* (adição, subtração, menor que, etc); *redução* (máximo, mínimo, etc); *convolução generalizada paralela* (convolução, erosão, dilatação, etc); *vizinhança paralela* (mediana, etc); *vizinhança recursiva paralela* (*transformada de distância*, implementações recursivas, etc); *geométrica* (rotação, reflexão, etc). Para cada padrão existe um algoritmo, que tem como parâmetro uma *string* que representa o índice de uma tabela contendo todas as transformações deste padrão. Isto define a programação genérica no *Horus*. Os tipos de dados no *Horus* são definidos através de seis elementos, definindo a

estrutura de dados genérica das imagens.

Horus com operadores morfológicos

Um estudo de padrões de algoritmos morfológicos foi feito por D'Ornellas [D'O01], onde três padrões de algoritmos foram definidos: paralelo, seqüencial e baseado em filas. Classificações semelhantes também foram estudas por outros pesquisadores [Vin92, BHJ97, ZL00]. D'Ornellas [D'O01] adicionou ao *Horus* estes padrões criando um ambiente de programação genérico e orientado a objetos para operadores de morfologia matemática, onde representações abstratas em algoritmos foram implementadas.

OLENA

Baseado nas conclusões de D'Ornellas [D'O01], Darbon *et. al.* [DGDL02] criou o ambiente *OLENA*, uma biblioteca dedicada a usuários de morfologia matemática com código fonte disponível. Este trabalho apresenta de forma simples um ambiente de processamento de imagens, onde escreve operadores morfológicos, como dilatação, erosão e *watershed*, usando programação genérica para diversos tipos de dados e dimensões da imagem.

A.1.4 Solução proposta neste apêndice - *mmil*

Será proposta neste texto a seguinte solução: a *MMachLib* seria reescrita em uma linguagem de programação para algoritmos morfológicos, chamada *linguagem intermediária*, fazendo uso de elementos de alto desempenho (definida neste trabalho). Por exemplo, elementos de acesso à fila, de acesso à vizinhança de um pixel, de acesso a todos os pixels da imagem, etc. Nesta linguagem a codificação de quase todos os algoritmos de morfologia matemática seria possível. Existem casos onde mais que uma implementação

para um operador morfológico são necessárias. Por exemplo, a *transformada de distância* pode ser implementada usando filas, ou usando varredura de vizinhança seqüencial, ou usando composições de erosões. O conjunto de transformações implementadas usando a *linguagem intermediária* juntamente com as regras de geração de código (definidas pelos *esquemas* e pelas *folhas de estilo*) formam o nosso ambiente *mmil* (*mathematical morphology intermediary language*). Dado uma arquitetura, uma estimativa do tempo de execução pode ser feita para cada elemento do ambiente *mmil* e baseada nesta estimativa a melhor implementação pode ser escolhida para cada função na biblioteca. A *linguagem intermediária* é compilada usando o melhor algoritmo para a plataforma em questão. Neste ambiente, com o surgimento de novas máquinas e de novas linguagens, a tarefa de migração do código da *MMachLib* consistiria apenas na adaptação das rotinas de geração de código para as novas linguagens ou máquinas.

Linguagem intermediária

Usualmente as transformações escritas em linguagens de programação são implementadas em dois passos. No primeiro passo, poderia especificar através da *linguagem morfológica*. No passo seguinte, poderia implementar em uma linguagem de programação de propósito geral, como *C*. Entretanto, é desejável implementar um sistema mais genérico e flexível de forma que os operadores morfológicos serão descritos em uma *linguagem intermediária*. Esta *linguagem intermediária* permitirá sua tradução eficiente para diversas linguagens e arquiteturas.

Por exemplo, existem várias formas de implementar um operador morfológico. A Figura A.1 ilustra três algoritmos para a erosão: paralelo, seqüencial e por propagação. Esta figura também ilustra a tradução de um comando da *linguagem intermediária* para comandos da linguagem *C*. A implementação que tiver o melhor desempenho numa dada arquitetura será a

escolhida para a tradução.

O principal resultado deste trabalho é a definição da *linguagem intermediária*. Seus elementos devem ser em número reduzido, poderosos na utilidade e que sejam facilmente traduzidos nas diversas arquiteturas disponíveis, mesmo as de alto desempenho. Também se deseja compilar a *linguagem intermediária* para gerar código em diversas linguagens, por exemplo *C*, *MATLAB*, *Python* e *Tcl/Tk*, em várias plataformas, como *MS-Windows* e *Unix*, e em várias arquiteturas, como *MMX* e cartões aceleradores de processamento de imagens.

Apesar do nome *linguagem intermediária* ser bem genérico, neste documento será usado este nome para descrever um passo intermediário entre a programação de alto nível, descrita por uma linguagem natural como a linguagem morfológica, e uma linguagem de programação de propósito geral.

Além disso, existem diversos tipos de usuários no desenvolvimento e no uso de um software de processamento de imagens. Para simplificar, serão definidos dois tipos de usuários. Os usuários que desenvolvem aplicativos, ou seja, usam um software de processamento de imagens para resolver problemas do mundo real, como segmentar imagens médicas usando o *MATLAB*. Em morfologia matemática estes usuários programam usando apenas a linguagem morfológica. Um outro tipo de usuário, que também será chamado projetista de software, é responsável pela criação de um software de processamento de imagens. A *linguagem intermediária* definida neste texto está voltada para este segundo tipo de usuário, para os projetistas de software para processamento de imagens.

A.2 Introdução ao ambiente *mmil*

Será apresentado neste apêndice um ambiente para a geração automática de código para operadores de morfologia matemática, chamado *mmil*. Este ambiente roda na linguagem *XML*, que usa as vantagens de armazenar os

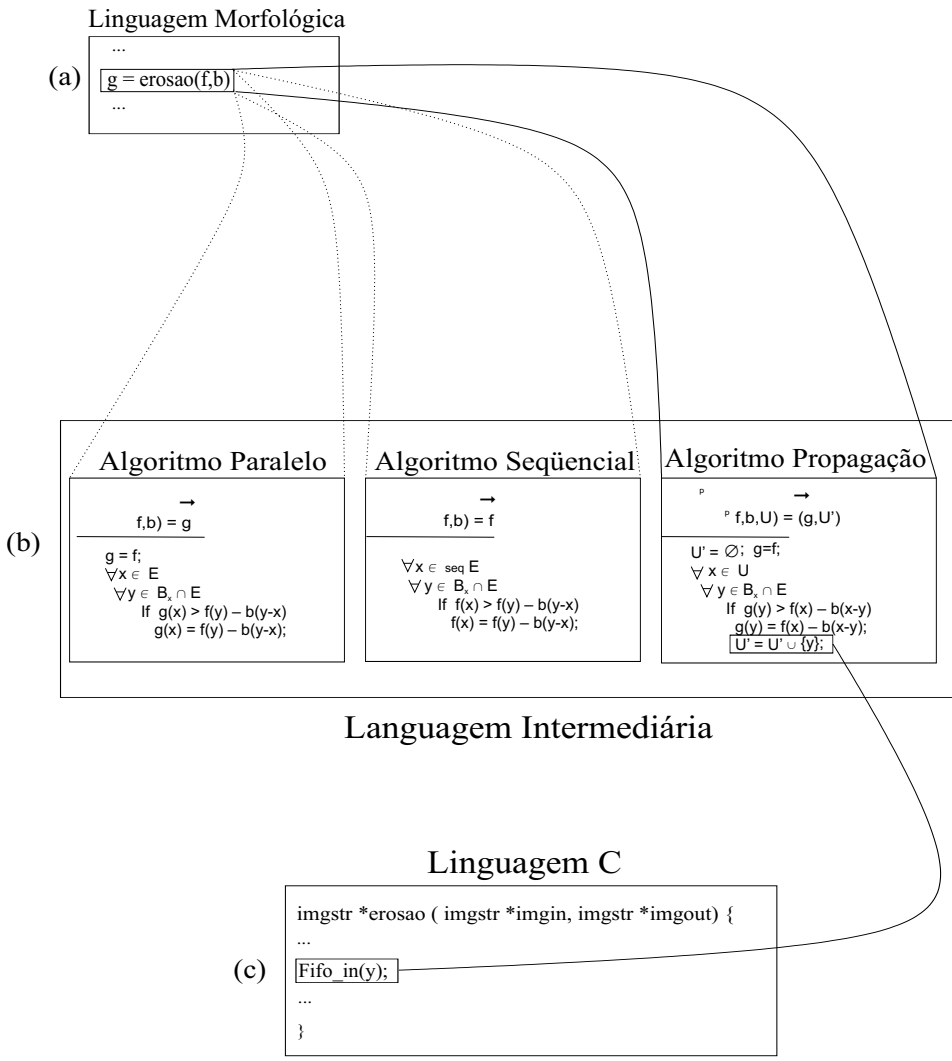


Figura A.1: Ilustração da relação entre: (a) *linguagem morfológica*, (b) *linguagem intermediária* e (c) *linguagem C*.

dados numa estrutura de árvore. A Figura A.2 mostra a arquitetura do ambiente *mmil*. Esta figura mostra um editor *GUI* (*Graphic User Interface*) para editar documentos *XML*. Em nossos estudos, estes documentos são operadores morfológicos implementados através dos elementos definidos na próxima seção. O conteúdo em *XML* é validado pelo *esquema*, que é uma estrutura contendo um conjunto de regras definidas para os atributos e elementos (nós da árvore) do *XML*. O conteúdo em *XML* é então processado pelas *folhas de estilo* gerando códigos em diversas outras linguagens (*C, MATLAB, PYTHON, TCL/TK*, etc), em diversas plataformas (*UNIX, LINUX, WINDOWS*, etc.), e também para gerar documentos (*LaTeX, HTML*, etc.).

Outra ferramenta usada na máquina de translação da *mmil* é a linguagem *TCL*, que trabalha junto com as *folhas de estilo* no processo.

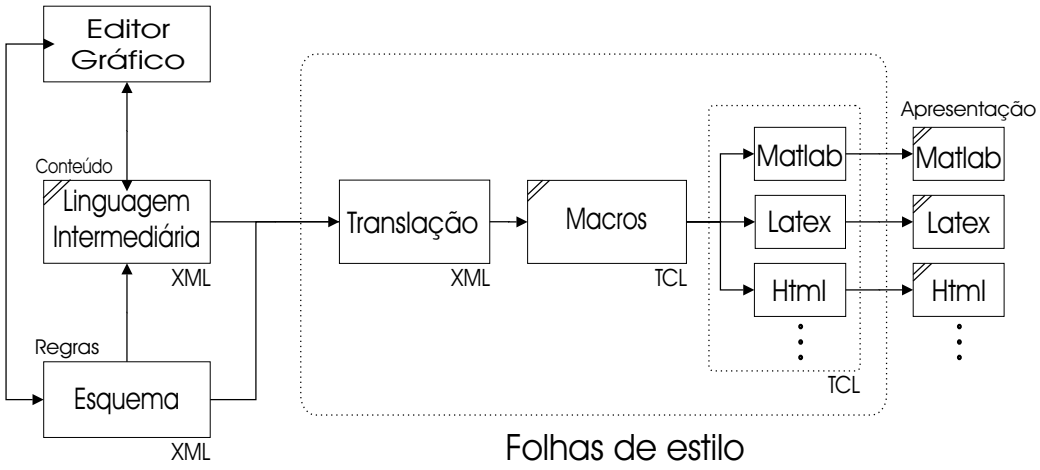


Figura A.2: Arquitetura da *mmil*.

Atualmente existem *folhas de estilo* para gerar códigos nas linguagens *MATLAB* e *C*. Também existe *folha de estilo* para gerar documentos em *LaTeX*.

A arquitetura da *mmil* foi desenvolvida no *Adesso*, um ambiente computacional de suporte ao desenvolvimento de aplicações científicas [Mac02].

Na Seção A.4 será apresentado um breve resumo dos elementos do modelo de informação do *Adesso* necessários para o desenvolvimento da linguagem intermediária.

A.3 Linguagem intermediária

Será apresentado nesta seção uma linguagem para algoritmos morfológicos, chamada *linguagem intermediária*, inspirado na *notação Z* [Spi88] e na *linguagem morfológica* [BB92].

A.3.1 Notação Z

A *notação Z* é utilizada para a especificação de problemas em engenharia de software através de notações matemáticas e possui uma ferramenta denominada *assistente de prova*, utilizada para prova automática das especificações [Pre02].

Como a *notação Z* modela problemas usando notação matemática, é natural usar em paralelo uma linguagem que possibilita editar símbolos matemáticos, como *LaTeX*. Neste sentido, um trabalho para servir de inspiração para mudança entre formatos é o conversor *Zed2XML* [CVM99], que transforma especificações Z escritas em *LaTeX* em documentos correspondentes em HTML [CVM99] e isto também pode ser realizado no ambiente proposto neste documento através das *folhas de estilo*, introduzidas na seção anterior. Existem também editores para a *notação Z*, que podem armazenar seus documentos em *LaTeX*, como o *ZCREATOR* [BC98] e visualizadores, como o *Z Browser* [Mik95].

A *notação Z* possui uma quantidade significativa de símbolos próprios, o que dificulta o aprendizado e a utilização. Por este motivo e baseado nos trabalhos existentes da *notação Z*, é possível criar um editor simplificado, que suporta a escrita dos padrões de algoritmos morfológicos estudados nos capítulos anteriores. Neste editor, além de poder visualizar as expressões

<operador>::=<operador elementar>|<limitante>|<composição>
<limitante>::=<argumento><operação de reticulado><argumento>
<argumento>::=<termo>|<composição>
<termo>::=<operador elementar>| (<limitante>)
<composição>::=<termo><termo>|<composição><termo>
<operador elementar>::=<operador morfológico>$_{\text{<função estruturante>}}$
<função estruturante>::=<letra>|<letra><número>
<número>::=<dígito>|<número><dígito>
<operação de reticulado>::= $\vee$ | $\wedge$
<operador morfológico>::= ε | δ | ε^a | δ^a
<letra>::= a | b | c | d
<dígito>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Tabela A.1: Gramática da linguagem morfológica [BB94].

matemáticas, é possível gerar códigos *LaTeX, HTML, C, MATLAB*, entre outros, como serão visto a seguir. Este editor seria uma proposta de continuação deste trabalho e não será discutido neste texto.

A.3.2 Linguagem morfológica

Existe na literatura uma linguagem formal para os operadores elementares de morfologia matemática [BB92, BB94], chamada *linguagem morfológica*. Veja na Tabela A.1 a gramática da linguagem morfológica. Esta linguagem tem como característica representar a dilatação, a erosão, a anti-dilatação e a anti-erosão por funções estruturantes. Porém, a linguagem intermediária definida neste documento, além de possuir esta característica, possui um vocabulário voltado para as diversas possibilidades de implementações destes operadores elementares.

A.4 Elementos da linguagem intermediária

A *notação Z* pode ser armazenada numa linguagem intermediária chamada *linguagem intercâmbio* (*interchange language*) [Har96] usando ele-

mentos (*tags*) para armazenar as informações de forma semelhante à linguagem *XML*. Analogamente, a linguagem *XML* é utilizada para armazenar a linguagem intermediária definida neste apêndice através de poucos elementos, como serão apresentados nesta seção.

Antes de definir os elementos da linguagem intermediária, será apresentado um breve resumo de onde estes elementos foram incluídos na estrutura do *Adesso*. Para mais detalhes deste ambiente consulte [Mac02]. Seja o elemento chamado *AdFunctions*, com os filhos definidos pelo elemento *AdFunction*. Veja Figura A.3 [1].

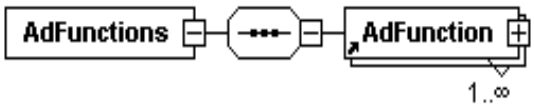


Figura A.3: Elemento *AdFunctions*.

O elemento *AdFunction* define as transformações implementadas, Figura A.4, e tem como filhos: *Platforms* – define as plataformas onde serão gerados os códigos; *Short* – descreve uma descrição da função; *Symbol* – associa um símbolo matemático; *Returns* – define os argumentos de retorno, veja Figura A.5; *Args* – define os argumentos de entrada através dos filhos definidos pelo elemento *Arg* contendo os atributos *name* e *type*, veja Figura A.6; e *Source* – descreve o código da função, veja Figura A.7.

A *linguagem intermediária*, onde são implementados os operadores morfológicos, é definida dentro do elemento *Code*, filho de *Source*, com atributo *lang* recebendo *"mmil"*, veja Figuras A.7 e A.8.

A *linguagem intermediária* é definida através de nove elementos: *Var* para variáveis, *Index* para acessar índices de vetor, *Set* para atribuições, *Oper* para transformações em geral, *Loop* para controles de repetição, *If* para operação condicional, *Call* para acessar outras funções já implementadas na *linguagem intermediária*, *Expand* e *Extract* para transformações que usam vizinhança.

[1] As Figuras A.3 até A.14 foram geradas pelo software *XMLSpy*.

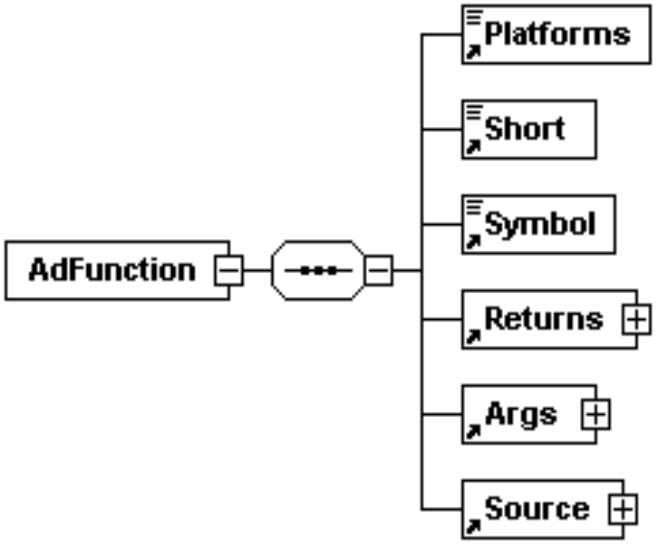


Figura A.4: Elemento *AdFunction*.

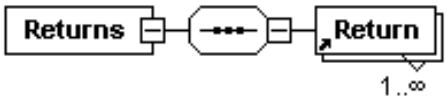


Figura A.5: Elemento *Returns*.

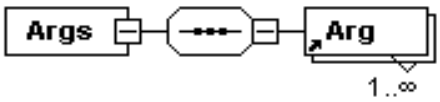


Figura A.6: Elemento *Args*.

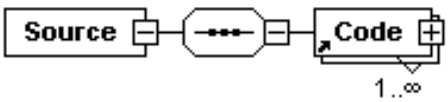


Figura A.7: Elemento *Source*.

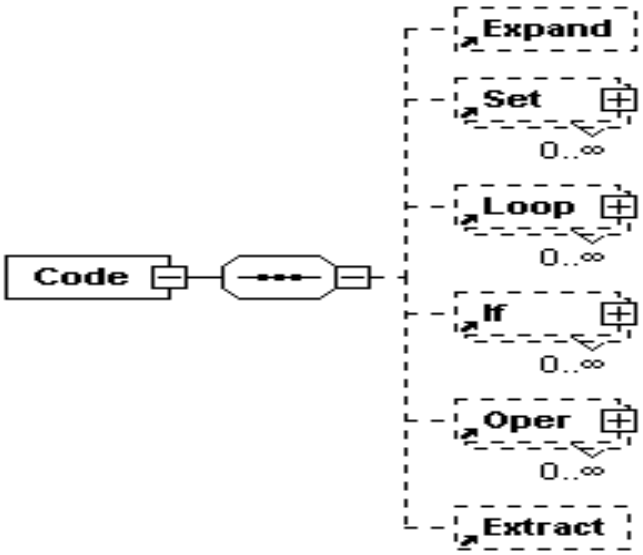


Figura A.8: Elemento *Code*.

Será descrito abaixo cada um destes elementos.

A.4.1 *Var*

Var é usado para acessar variável. Os seguintes casos podem ocorrer: o conteúdo é um valor escalar, uma matriz, ou um elemento de uma matriz (para o caso 2D). Neste último caso, um elemento de uma matriz é acessado pelo uso recursivo do elemento *Var* e pelo elemento *Index*. Veja Figura A.9.

Figura A.9: Elemento *Var*.

A.4.2 *Index*

Index é usado para acessar os elementos de uma matriz e é usado junto com o elemento *Var*. Veja Figura A.10.

Figura A.10: Elemento *Index*.

A.4.3 *Set*

Set é usado para especificar atribuições, onde existem duas partes, *left* e *right*. O conteúdo da parte *right* é associado a parte *left*. Associado à *left* existe o elemento *Var* e associado a *right* pode existir um dos filhos mostrados na Figura A.11.

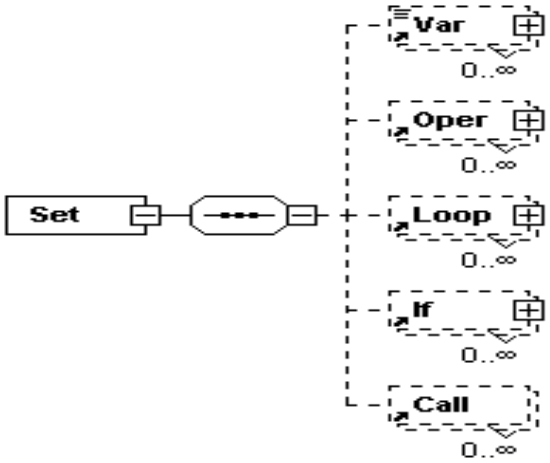


Figura A.11: Elemento *Set*.

A.4.4 *Oper*

Oper especifica uma transformação, que é dividida em três padrões: *pontual, global* e *geométrico*. O padrão *pontual* tem as operações: *adição* $(+)$, *subtração* $(-)$, *diferença* $(\neq)$, *igualdade* $(==)$, *união* $(\vee)$, *intersecção* $(\wedge)$, *negação* $(\sim)$, etc. O padrão *global* tem as operações *máximo* $(\bigvee)$, *mínimo* $(\bigwedge)$, etc. O padrão *geométrico* tem as transformações *translação* (B_x é a translação do conjunto B pelo vetor x), *find*, *fronteira* (∂), etc. Veja Figura A.12. Estas opções são definidas através dos seguintes elementos:

name especifica a operação;

i especifica um índice quando a operação é uma translação.

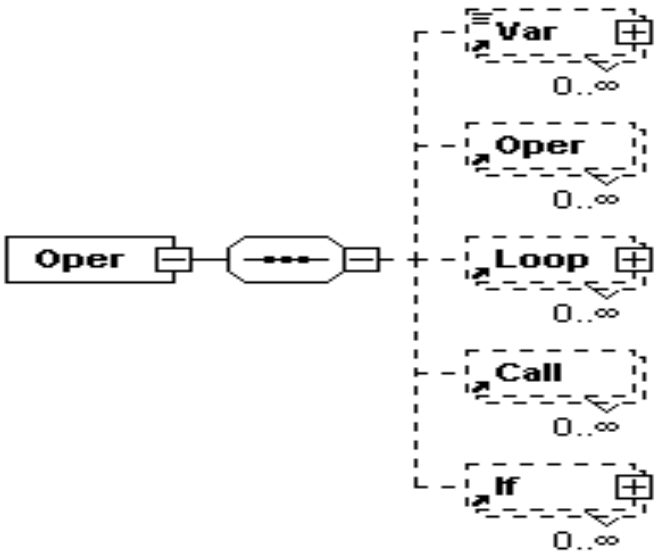


Figura A.12: Elemento *Oper*.

A.4.5 *Loop*

Loop este elemento faz referência a repetições *For* ou *While*, denotado por $\forall$. O laço é associado a um índice e a uma especificação de um domínio

através de atributos. O laço *While* requer uma expressão lógica. Veja Figura A.13. Para este elemento serão definidos os seguintes atributos:

name especifica o tipo de laço;

i especifica o índice para o laço;

D especifica o domínio de i;

Dse especifica o domínio da função estruturante;

oper especifica uma operação associada ao laço;

it especifica o índice se existir uma translação dentro de *Loop*;

Ds especifica o conjunto de armazenamento usado nos operadores por propagação;

s especifica o índice que vai percorrer o conjunto definido em *Ds*;

out especifica uma saída;

A.4.6 *If*

If é um comando condicional. Se a expressão lógica é *True*, o bloco associado ao elemento *If* é executado. Veja Figura A.14.

A.4.7 *Call*

Call é usado para referenciar uma operação implementada pela *mmil*. Serão definidos os seguintes elementos:

name especifica o nome da operação;

i1 especifica a primeira entrada;

i2 especifica a segunda entrada;

o1 especifica a saída.

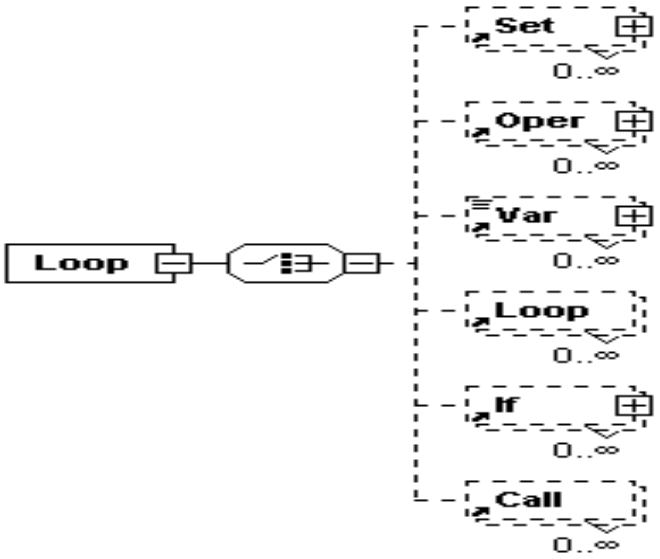


Figura A.13: Elemento *Loop*.

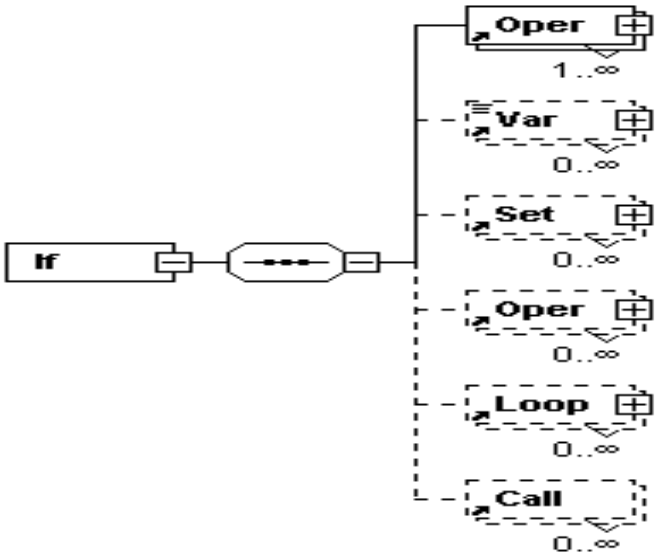


Figura A.14: Elemento *If*.

A.4.8 *Expand*

Expand expande uma imagem pela vizinhança passada como parâmetro. É passado também o valor a ser atribuído na borda expandida. Por exemplo, se for usado vizinhança 3×3, a imagem será expandida de uma linha e uma coluna ao redor da imagem. Para este elemento serão definidos os seguintes atributos:

Out especifica a saída expandida da imagem definida no atributo *In*;

Out1 especifica outra imagem com dimensões equivalentes à definida em *Out*, porém com valores definidos por *Value*;

In especifica a entrada;

SE especifica a vizinhança que define quanto vai expandir;

Value especifica o valor a ser atribuído à região expandida.

A.4.9 *Extract*

Extract extrai uma imagem pela vizinhança passada como parâmetro. Utilizado pelos operadores morfológicos que usam funções estruturantes. Para este elemento serão definidos os seguintes atributos:

Out especifica a saída;

In especifica a entrada;

SE especifica a vizinhança que define quanto vai diminuir a entrada.

A.5 Exemplo de uso dos elementos *Var* e *Index*

Nesta seção serão apresentados alguns exemplos ilustrativos da programação usando os elementos definidos anteriormente. Inicialmente, serão

mostrados exemplos simples com os elementos *Var* e *Index*. Em seguida, serão apresentados dois exemplos da operação de adição, mostrando o código *XML*, a expressão matemática e o código *MATLAB*. Finalmente, serão descritas três versões da erosão, dependendo da forma em que o laço é feito.

Exemplo A.1 *Este é um simples exemplo do uso do elemento Var.*

```
<Var>0</Var>
```

O retorno deste elemento é o valor zero. ◇

Exemplo A.2 *O valor de saída do elemento Var é a variável f que pode ser um valor escalar ou uma matriz, dependendo da forma em que foi definida.*

```
<Var>f</Var>
```

◇

Exemplo A.3 *Este exemplo usa os elementos Var e Index, devolvendo o conteúdo de um elemento de variável hist, definida por um elemento de f. Este exemplo pode ser reescrito simplesmente por hist*$(f(x))$.

```
<Var>
   hist
   <Index>
     <Var>f<Index>x</Index></Var>
   </Index>
</Var>
```

◇

A.6 Exemplos de adição em imagens

Nesta seção serão apresentados dois exemplos da operação de adição entre duas imagens com o mesmo domínio.

Exemplo A.4 *O exemplo abaixo faz a imagem g receber a adição da imagem $f1$ e $f2$. Estas três variáveis possuem o mesmo tamanho $\mathbb{E}$.*

```
...
<Set>
  <Var>g</Var>
  <Oper name="Add">
    <Var>f1</Var>
    <Var>f2</Var>
  </Oper>
</Set>
```

Este código em XML pode ser apresentado através da seguinte expressão:

$$g = f1 + f2,$$

ou ainda pelo seguinte código em MATLAB:

```
g = f1 + f2;
```

◇

Exemplo A.5 *Neste outro exemplo será mostrada uma outra versão para a adição de duas imagens. As três variáveis $f1, f2$ e g possuem o mesmo tamanho $\mathbb{E}$. Agora a varredura é explícita para o domínio $\mathbb{E}$:*

```
...
<Loop name="for" i="x" D="f1">
  <Set>
    <Var>g<Index>x</Index></Var>
```

```
    <Oper name="Add">
      <Var>f1<Index>x</Index></Var>
      <Var>f2<Index>x</Index></Var>
    </Oper>
  </Set>
</Loop>
```

Este código pode ser visto como a seguinte expressão (apresentada também no Algoritmo 2):

$\forall x \in \mathbb{E}$
$\quad g(x) = f_1(x) + f_2(x);$

ou ainda, pelo seguinte código em MATLAB:

```
for x=D(f1)
  g(x) = f1(x) + f2(x);
end
```

onde $\forall x \in \mathbb{E}$ *ou* $x = D(f1)$ *são todos os pixels da imagem* $f1$. ◇

Elemento *Call*

Exemplo A.6

```
<Call name="ero" i1="f" i2="b" o1="g"/>
```

Este exemplo é equivalente a $g = \varepsilon_b(f)$*, isto é,* g *recebe a erosão da imagem* f *pela função estruturante* b. ◇

A.7 Diferentes formas de implementar a erosão

Serão apresentadas três diferentes formas de implementar a erosão usando a linguagem intermediária através de três expressões matemáticas diferentes, porém equivalentes. A parte central deste exemplo é mostrar a flexibilidade do ambiente desenvolvido, particularmente do elemento *Loop*.

Nos exemplos a seguir não serão mostrados os códigos *XML* e *MATLAB*, pois não são códigos para leitores humanos, mas para as máquinas interpretarem. Serão mostradas apenas as expressões matemáticas equivalentes, que podem ser geradas automaticamente.

Exemplo A.7 *Seja $\mathbb{Z}$ o conjunto dos inteiros, $\mathbb{E} \subset \mathbb{Z}^2$ o domínio da imagem e $K = [0, k] \subset \mathbb{Z}$ um intervalo de números inteiros representando os possíveis níveis de cinza da imagem. O operador invariante por translação em níveis de cinza, $\varepsilon_b : K^{\mathbb{E}} \to K^{\mathbb{E}}$ ($K^{\mathbb{E}}$, lê-se conjuntos de funções de $\mathbb{E}$ em K), é definido como [Hei91]:*

$$\varepsilon_b(f)(x) = \min\{f(y) \dot{-} b(y - x) : y \in B_x \cap \mathbb{E}\},$$

onde $f \in K^{\mathbb{E}}$, $x \in \mathbb{E}$, $B \subseteq \mathbb{E} \oplus \mathbb{E}$ e B é chamado elemento estruturante), $B_x = \{y + x,\ y \in B\}$ (translação de B por x) e b é uma função estruturante definida em B com $b : B \to \mathbb{Z}$.

A equação acima, quando implementada na mmil, sem o tratamento de borda, tem a representação apresentada no Algoritmo 20 (veja Figura 2.4).

A expressão $\forall x \in \mathbb{E}$ é especificada pelo elemento Loop com o atributo for:

```
<Loop name="for" i="x" D="f">
```

O atributo i especifica um elemento do domínio de f, isto é, um elemento de $\mathbb{E}$. A expressão $\bigwedge_{\forall y \in B_x \cap \mathbb{E}}\{...\}$ é também especificada pelo elemento Loop, com um atributo adicional oper recebendo Min representando a operação de redução $\bigwedge$, que é a operação de mínimo:

Algoritmo 20 Primeiro algoritmo da erosão paralela

$\varepsilon^1 : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}}$
$\quad \varepsilon^1(f, b) = g$

$\forall x \in \mathbb{E}$
$\quad g(x) = \bigwedge_{\forall y \in B_x} \{f(y) \dot{-} b(y - x)\};$

```
<Loop name="for" i="y" Dse="b" oper="Min">
```

O atributo Dse acima indica que o laço irá varrer o domínio da função estruturante b. O parâmetro $b(y - x)$ é obtido pelos elementos Var, Index e Oper. Neste último caso, dois atributos são passados, um indicando a operação de translação e o outro indicando o índice de translação:

```
<Oper name="Transl" i="x">
  <Var>b<Index>y</Index></Var>
</Oper>
```

$\diamond$

Exemplo A.8

Neste outro exemplo, apresentado no Algoritmo 21, será obtido a erosão da imagem f pela função estruturante b, fazendo a intersecção da translação de f por y, subtraindo de b em y (veja Figura 2.5).

Algoritmo 21 Segundo algoritmo da erosão paralela

$\varepsilon^2 : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}}$
$\quad \varepsilon^2(f, b) = g$

$\forall x \in \mathbb{E}$
$\quad g = \bigcap_{\forall y \in B_x} \{f_y \dot{-} b(y)\};$

A principal diferença deste exemplo e do exemplo anterior, com relação ao código gerado, é com o parâmetro f_y, que tem como retorno uma imagem

transladada por y. Este tratamento diferenciado da operação de translação ocorre através do contexto onde o parâmetro f aparece no código XML.

$\diamond$

Exemplo A.9

Neste último exemplo, apresentado no Algoritmo 22, será obtido a erosão de cada pixel x da imagem g fazendo a redução, através da operação de mínimo de todas as sub-imagens de f subtraídas de b (veja também Figura 2.6).

Algoritmo 22 Terceiro algoritmo da erosão paralela

$\varepsilon^3 : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}}$
$\quad \varepsilon^3(f, b) = g$

$\forall x \in \mathbb{E}$
$\quad g = \bigwedge\{f(x)_{y,\ \forall y \in B} \dot{-} b\};$

Note que neste exemplo $f(x)_{y,\ \forall y \in B}$ retorna uma sub-imagem de f com centro no pixel x tendo o mesmo domínio de b.

$\diamond$

A.8 Conclusões e trabalhos futuros

Foi apresentado neste apêndice *mmil*, *mathematical morphology independent language*, um ambiente que escreve transformações morfológicas em *XML* e, ao ser compilado, gera de forma automática código em diversas linguagens de programação e também documentos. Além disso, foi visto que ao gerar documentos em *LaTeX*, é possível não mais trabalhar com linguagens de programação para descrever uma transformação morfológica, mas com expressões matemáticas, e a compilação destas expressões (armazenadas em *XML*) são códigos executáveis.

Trabalhos futuros

Para concluir este ambiente está faltando validar os códigos gerados para a linguagem de programação C, corrigir eventuais erros de geração de código, melhorar e completar a documentação do ambiente e a documentação gerada e finalmente testar a eficiência dos códigos gerados de forma automática. Por exemplo, é desejável saber quais das três operações de erosão implementadas na Seção A.7 é a mais eficiente na linguagem *C* ou *MATLAB*, rodando em um *PC* com *Windows 98* ou *LINUX*. Falta também fazer a otimização do código gerado.

Como resultado adicional deste ambiente, fazendo poucas modificações, é possível construir uma estrutura em *XML* contendo todas as informações necessárias de um artigo, a compilação é gerada num formato qualquer (como *pdf*) pronta para a submissão, e é possível também testar o pseudo-código (representados por expressões matemáticas) gerando códigos para a linguagem de programação desejada. Assim será eliminado o trabalho de implementar um pseudo-código contido em um artigo.

Referências Bibliográficas

[ASU95] A.V. Aho, R. Sethi e J.D. Ullman. *Compiladores: Princípios, Técnicas e Ferramentas*. LTC, Rio de Janeiro, Brasil, 1995.

[Aus99] M.H. Austern. *Generic Programming and the STL: using and extending the C++ Standard Template Library*. Addison-Wesley Publishing Company, London, 1999.

[BB92] J. Barrera e G.J.F. Banon. Expressiveness of the morphological language. No *Image Algebra and Morphological Image Processing III*, páginas 264–274, San Diego, California, 1992. SPIE.

[BB94] G.J.F. Banon e J. Barrera. *Bases da morfologia matemática para analise de imagens binárias*. IX Escola de Computação, Recife, Brasil, 1994.

[BBL94] J. Barrera, G.F. Banon e R.A. Lotufo. A mathematical morphology toolbox for the KHOROS system. No *Image Algebra and Morphological Image Processing V*, páginas 241–252, Bellingham, Julho 1994. SPIE.

[BC98] J. Bowen e D. Chippington. Z on the web using java. *Proc. 11th Int. Conf. on the Z Formal Method (ZUM)*, 1493:66–80, Setembro 1998.

[BHJ97] J. Barrera e R. Hirata Jr. Fast algorithms to compute the elementary operators of mathematical morphology. No *Brazilian Symposium on Computer Graphics and Image Processing*, páginas 163–170, São Paulo, Brasil, Outubro 1997.

[CVM99] P. Ciancarini, F. Vitali e C. Mascolo. Managing complex documents over the www: a case study for XML. Relatório Técnico UBLCS-99-06, Department of Computer Science, Mura Anteo Zamboni, 7, Italy, 1999.

[DCB+02] M.C. D'Ornellas, M. Carrard, T. Baldissera, G. Peccini e R. Disconzi. Programação genérica com C++ e STL. Relatório Técnico, Universidade Federal de Santa Maria, Santa Maria, RS, Brasil, 2002.

[D'O01] M.C. D'Ornellas. *Algorithmic Patterns for Morphological Image Processing*. Tese de Doutorado, University of Amsterdam, Amsterdam, 2001.

[DGDL02] J. Darbon, T. Géraud e A. Duret-Lutz. Generic implementation of morphological image operators. *ISMM*, 2002.

[GW92] R.C. Gonzalez e R.E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.

[Har96] A. Harry. *Formal Methods - Fact File: VDM and Z*. John Wiley and Son Ltd, 1996.

[Hei91] H.J.A.M. Heijmans. Theoretical aspects of gray-level morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):568–581, Junho 1991.

[HWIC89] L.G.C. Hamey, J.A. Webb e Wu I-Chen. An architecture independent programming language for low-level vision. *Computer Vision, Graphics and Image Processing*, 48(2):246–264, Junho 1989.

[KPS00] D. Koelma, E. Poll e F. Seinstra. Horus release 0.9.2. Relatório Técnico, University of Amsterdam, Amsterdam, 2000.

[LZHJB97] R.A. Lotufo, F.A. Zampirolli, R. Hirata Jr. e J Barrera. MMachLib functions and MMach operators. *Brazilian Workshop on Mathematical Morphology*, Fevereiro 1997.

[Mac02] R.C. Machado. Adesso - ambiente para desenvolvimento de software científico. Dissertação de Mestrado, Universidade Estadual de Campinas - UNICAMP, Campinas, SP, Brasil, 2002.

[McG99] S. McGrath. *XML: Aplicações Práticas*. Campus, Rio de Janeiro, 1999.

[Mik95] L. Mikusiak. Z browser: A tool for visualization of z specifications. *Proc. 9th Int. Conf. on the Z Formal Specification Notation (ZUM)*, 967:510–525, Setembro 1995.

[Pre02] R.S. Pressman. *Engenharia de Software*. McGraw-Hill, Rio de Janeiro, 5ed. edition, 2002.

[Spi88] J.M. Spivey. *Understanding Z: A Specification Language and Its Formal Semantics*. Cambridge University Press, 1988.

[Vin92] L. Vincent. Morphological algorithms. No E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, capítulo 8, páginas 255–288. Marcel Dekker, New York, 1992.

[Web90] J.A. Webb. Architecture-independent global image processing. No *10th International Conference on Pattern Recognition*, volume 2, páginas 623–628, Atlantic City, NJ, USA, Junho 1990.

[Wir76] N. Wirth. *Algorithms + Data Structures = Programs*. Prentice-Hall, 1976.

[WWIC89] R.S. Walace, J.A. Webb e Wu I-Chen. Machine-independent image processing: performance of apply on diverse architectures. *Computer Vision, Graphics and Image Processing*, 48(2):265–276, Junho 1989.

[ZL00] F.A. Zampirolli e R.A. Lotufo. Classification of the distance transformation algorithms under the mathematical morphology approach. No *Brazilian Symposium on Computer Graphics and Image Processing*, Gramado, RS, Brasil, Outubro 2000.

Índice Remissivo