Capítulo 8

Máquinas Morfológicas

O paradigma central da Morfolgia Matemática é a decomposição de operadores em termos dos operadores elementares (i.e., dilatações, erosões, anti–dilatações e anti–erosões) e das operações de composição, união e interseção.

Esta dinâmica de procedimento pode ser expressa através de uma linguagem formal: a Linguagem Morfológica. As frases desta linguagem serão exatamente as decomposições possíveis para os operadores.

A aplicação da teoria da Morfologia Matemática a problemas reais de análise de imagens requer o desenvolvimento de instrumentos adequados: as Máquinas Morfológicas. Um programa em uma máquina morfológica será equivalente a uma frase da linguagem morfológica.

Neste capítulo, apresentamos a Linguagem Morfológica e discutimos a arquitetura de uma máquina morfológica típica.

8.1 Linguagem morfológica

Um aspecto importante da Morfologia Matemática é a descrição de operadores entre subconjuntos pelo uso de uma linguagem formal [BarBan92], chamada de *Linguagem Morfológica* (LM).

A fim de definir uma *linguagem formal*, precisamos definir uma *gramática* (i.e., um conjunto de regras que definem a sintaxe) e uma *semântica* (i.e., um modelo de interpretação para a gramática). A Tabela 8.1 apresenta uma gramática formal para a LM, usando uma metalinguagem na forma de Backus–Naur [Pagan81]. Nesta tabela, o metasímbolo ↓ {...} significa um rebaixamento de meia linha.

Tabela 8.1 – GRAMÁTICA DA LM.

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

As sentenças que seguem são alguns exemplos de frases da LM:

$$\psi_1 ::= \epsilon_{a_1} \wedge \delta^{\mathrm{a}}{}_{b_1}$$

$$\psi_2 ::= \delta_{a_1} \wedge \epsilon^{\mathrm{a}}{}_{b_1}$$

$$\psi_3 ::= (\epsilon_{a_1} \wedge \delta^{\mathrm{a}}{}_{b_1}) \vee (\epsilon_{a_2} \wedge \delta^{\mathrm{a}}{}_{b_2})$$

$$\psi_4 ::= (\delta_{a_1} \wedge \epsilon^{\mathrm{a}}{}_{b_1}) \vee (\delta_{a_2} \wedge \epsilon^{\mathrm{a}}{}_{b_2})$$

$$\psi_5 ::= \delta_a \epsilon_a$$

$$\psi_6 ::= (\epsilon_a \vee \epsilon_b)(\delta_a \wedge \delta_b).$$

A Figura 8.1 apresenta a árvore sintática para a frase ψ_3.

A fim de definir formalmente uma semântica para uma gramática, devemos estabelecer um conjunto de funções de interpretação que mapeam as frases primitivas no domínio de interpretação. A interpretação é criada recursivamente e a ordem de execução das primitivas em uma frase é estabelecida pela gramática [GenNil88]. A Tabela 8.2 apresenta a definição formal de uma semântica para a gramática apresentada na Tabela 8.1.

Tabela 8.2 – SEMÂNTICA DA LM.

$$\mathcal{I}[a] \equiv f \in \mathcal{P}(E)^E$$
$$\mathcal{I}[\delta_a] \equiv \psi \in \Psi : \psi(X) = \{y \in E : (\mathcal{I}[a])^{\mathrm{t}}(y) \cap X \neq \emptyset\} \quad (X \in \mathcal{P}(E))$$
$$\mathcal{I}[\epsilon_a] \equiv \psi \in \Psi : \psi(X) = \{y \in E : \mathcal{I}[a](y) \subset X\} \quad (X \in \mathcal{P}(E))$$
$$\mathcal{I}[\delta^{\mathrm{a}}{}_a] \equiv \psi \in \Psi : \psi(X) = \{y \in E : (\mathcal{I}[a])^{\mathrm{t}}(y) \cap X \neq \emptyset\}^{\mathrm{c}} \quad (X \in \mathcal{P}(E))$$
$$\mathcal{I}[\epsilon^{\mathrm{a}}{}_a] \equiv \psi \in \Psi : \psi(X) = \{y \in E : \mathcal{I}[a](y) \subset X\}^{\mathrm{c}} \quad (X \in \mathcal{P}(E))$$
$$\mathcal{I}[(\phi)] \equiv \mathcal{I}[\phi]$$
$$\mathcal{I}[\phi_1 \vee \phi_2] \equiv \psi \in \Psi : \psi(X) = \mathcal{I}[\phi_1](X) \cup \mathcal{I}[\phi_2](X) \quad (X \in \mathcal{P}(E))$$
$$\mathcal{I}[\phi_1 \wedge \phi_2] \equiv \psi \in \Psi : \psi(X) = \mathcal{I}[\phi_1](X) \cap \mathcal{I}[\phi_2](X) \quad (X \in \mathcal{P}(E))$$
$$\mathcal{I}[\phi_1 \phi_2] \equiv \psi \in \Psi : \psi(X) = \mathcal{I}[\phi_1](\mathcal{I}[\phi_2](X)) \quad (X \in \mathcal{P}(E))$$

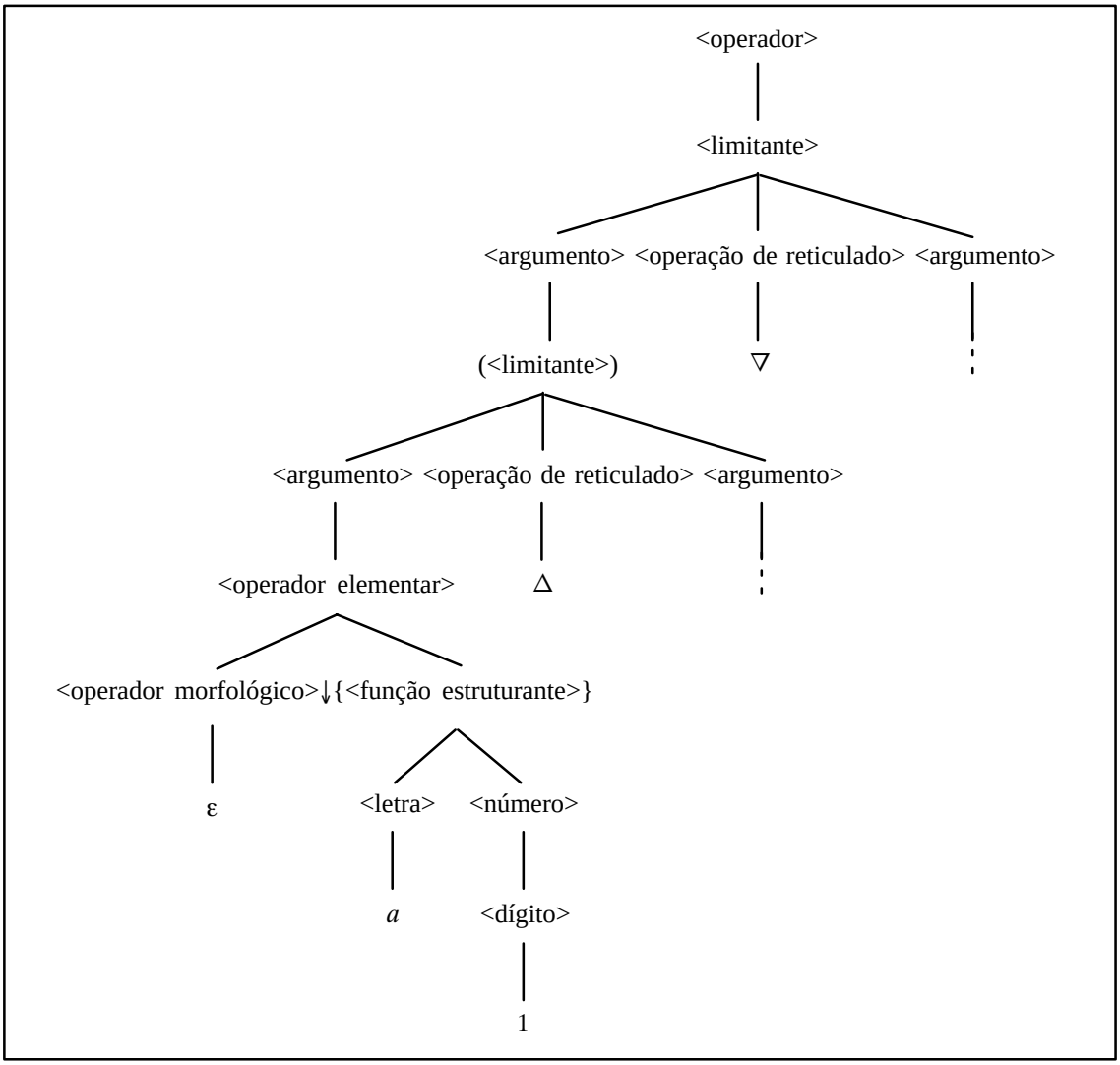


Fig. 8.1 – Árvore sintática de uma frase.

Deixe $\mathcal{I}$ denotar as funções de interpretação dos subconjuntos do conjunto de frases gerado pela gramática em $\mathcal{P}(E)^{\mathcal{P}(E)}$. A Figura 8.2 ilustra a interpretação semântica de uma frase ψ avaliada em X, que corresponde a uma dilatação invariante por translação de X pelo losângulo (a cruz) 3×3 centrado na origem.

Uma característica importante das frases da LM é que elas são construídas por cadeias de operadores elementares, ligados pelas operações de união, interseção e composição de operadores (ver Capítulo 3).

A LM é um linguagem completa (i.e., qualquer operador entre subconjuntos pode ser representado como uma frase desta linguagem) e expressiva (i.e., muitos operadores úteis podem ser construídos usando relativamente poucos operadores elementares). Podemos garantir que a LM é completa, pois qualquer operador entre subconjuntos pode ser representado por formas canônicas [BanBar90], similares àquelas apresentadas para a decomposição de operadores i.t., e essas formas são frases válidas da LM.

Fig. 8.2 – Semântica de uma frase avaliada num subconjunto.

8.2 Elementos estruturantes primitivos

Nos capítulos anteriores mostramos como os operadores elementares podem ser usados para construir uma extensa classe de operadores. Nesta seção, mostramos como esses operadores elementares podem ser decompostos em termos de uma pequena subfamília de operadores elementares. Na próxima seção, mostramos como essa propriedade pode ser explorada para o desenvolvimento de Máquinas Morfológicas mais eficientes.

Chamamos de *quadrado elementar* o quadrado 3×3, isto é, o subconjunto $\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$ e de *elemento estruturante primitivo* um subconjunto qualquer do quadrado elementar. Verificamos que qualquer subconjunto B pode ser representado pela união de translados $B_i + u_i$ de elementos estruturantes primitivos B_i, isto é, $B = \bigcup_i B_i + u_i$. A Figura 8.3 ilustra esta propriedade.

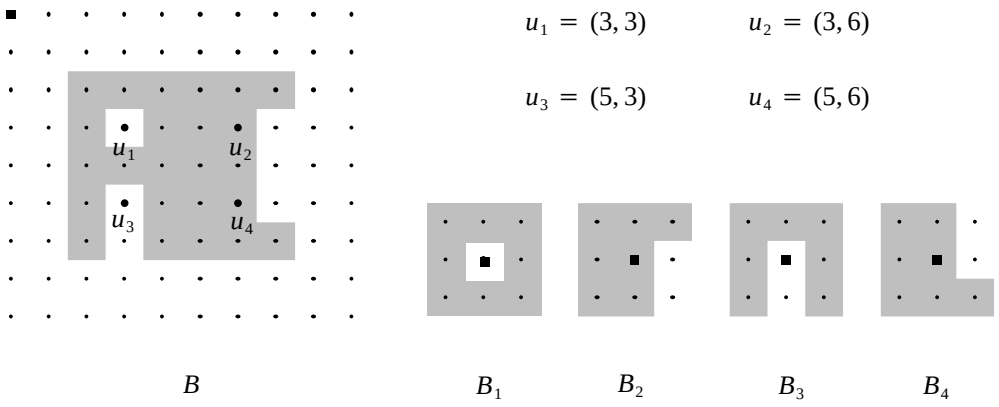


Fig. 8.3 – Decomposição em termos de subconjuntos do quadrado elementar.

Observe que o translado $B + u$ de qualquer subconjunto B por qualquer vetor u pode ser ralizado por composições de dilatações de B pelos elementos estruturantes primitivos $\begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 1 \\ 0 & 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 & 0 \\ 1 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 1 & 0 \\ 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix}$ e $\begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 \\ 0 & 1 & 0 \end{bmatrix}$, denotados, respectivamente, por L (leste), O (oeste), N (norte) e S (sul). Por exemplo, $B + (2, 3) = \delta_{3L}(\delta_{2S}(B))$.

Estes fatos e as propriedades $\delta_{B_1}\delta_{B_2} = \delta_{B_1 \oplus B_2}$ e $\delta_{B_1} \vee \delta_{B_2} = \delta_{B_1 \cup B_2}$, estudadas no Capítulo 4, garantem que qualquer dilatação i.t. pode ser realizada através de composições e uniões de dilatações por elementos estruturantes primitivos. Por exemplo, a dilatação pelo elemento estruturante B da Figura 8.3 pode ser expressa por

$$\delta_B = \delta_{3L}\delta_{3S}\delta_{B_1} \vee \delta_{6L}\delta_{3S}\delta_{B_2} \vee \delta_{3L}\delta_{5S}\delta_{B_3} \vee \delta_{6L}\delta_{5S}\delta_{B_4}.$$

Observe ainda, que esta estratégia pode ser usada para sintetizar qualquer forma B, pois $\delta_B(\{o\}) = B$.

Em particular, pode–se provar que qualquer dilatação por um subconjunto convexo pode ser construida *sequencialmente,* através de composições de dilatações por elementos estruturantes primitivos [Xu91]. Um subconjunto de $\mathbf{Z}^2$ é convexo se ele é a interseção de todos os semi planos a 0, 45, 90 e 135 graus que o contêm.

Por exemplo, a dilatação pelo elemento estruturante B da Figura 8.4 pode ser expressa por $\delta_B = \delta_{B_1}\delta_{B_2}...\delta_{B_7}$.

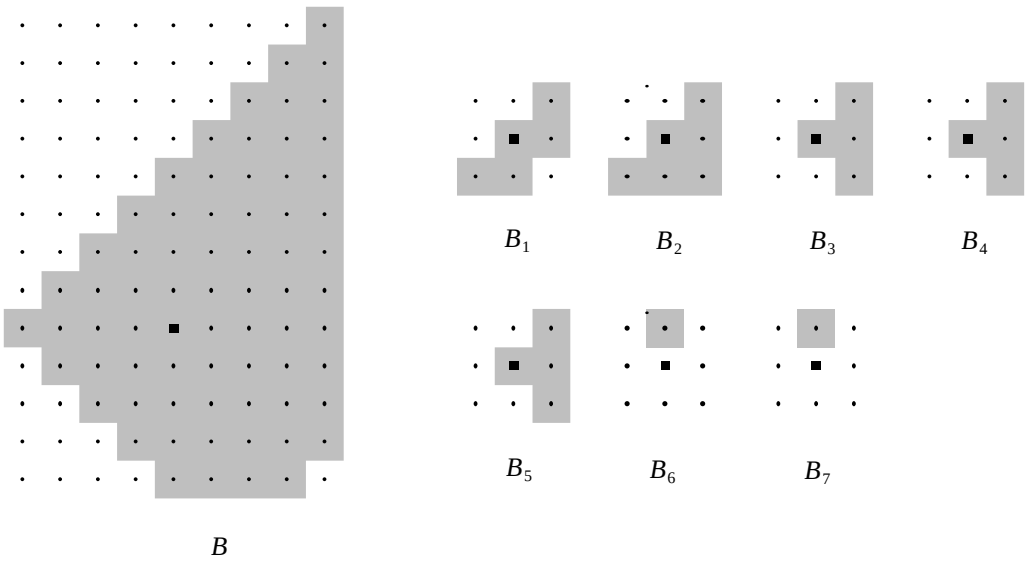


Fig. 8.4 – Decomposição de um subconjunto convexo.

Seja E um subconjunto de $\mathbf{Z}^2$. Um *operador elementar* é dito *localmente condicionalmente invariante por translação* ou *localmente c.i.t.*, se existem dois subconjuntos, um subconjunto B de $E \oplus E^t$, e um subconjunto M de E, tal que sua função estruturante b seja definida por, para todo y em E,

$$b(y) = \begin{cases} (B + y) \cap E & \text{se } y \in M \\ \emptyset & \text{c.c.}. \end{cases}$$

O subconjunto M é denominado *máscara do operador.*

A dilatação, a erosão, a anti–dilatação e a anti–erosão localmente c.i.t. que têm b como função estruturante serão denotadas, respectivamente, por $\delta_{B,M}$, $\epsilon_{B,M}$, $\delta^a{}_{B,M}$ e $\epsilon^a{}_{B,M}$.

Seja $(M_i)_{i\in I}$ uma partição de E, seja $(B_i)_{i\in I}$ uma família de $\mathcal{P}(E \oplus E^t)$ e seja b a função estruturante definida por $b(y) = (B_i + y) \cap E$ se $y \in M_i$, então, pela Proposição 3.12,

$$\delta_b = \bigvee_{i \in I} \delta_{B_i,M_i}.$$

Desta forma, as dilatações localmente c.i.t. podem ser usadas como protótipos para a construção de qualquer dilatação.

Ainda, em muitas situações práticas, uma dilatação pode ser construída a partir de dilatações localmente c.i.t. que têm elementos estruturantes primitivos. Esta propriedade decorre do fato que

$$\delta_{B_{i_1},M_i} \vee \delta_{B_{i_2},M_i} = \delta_{B_{i_1} \cup B_{i_2},M_i}$$

e que, em muitas situações práticas,

$$\delta_{B_{i_1},M_i}\delta_{B_{i_2},M_i} = \delta_{B_{i_1} \oplus B_{i_2},M_i}.$$

Trocando o operador dilatação pelo operador erosão e a operação de união pela operação de interseção, encontramos resultados duais para todas as propriedades apresentados nesta seção.

A Figura 8.5 ilustra a construção de uma erosão adaptativa a partir de erosões localmente c.i.t.. A imagem apresentada corresponde a vista em perspectiva de uma estrada. Devido ao efeito da perspectiva, os trechos da estrada que estão mais próximos do observador aparecem maiores do que os que estão mais afastados. Para que esta mudança de escala seja considerada, o domínio da imagem é particionado, aqui, em três regiões e a cada uma delas é associada um elemento estruturante, que tem a mesma forma dos demais, porém tamanho diferente. Com isto, a erosão respeita de maneira aproximada o efeito de perspectiva, alargando a pista proporcionalmente ao tamanho dos elementos estruturantes.

Lembrando, ainda, que as anti–dilatação e as anti–erosão podem ser construídas, respectivamente, a partir das dilatações e das erosões, todos esses resultados podem ser usados para construir anti–dilatações e anti–erosões.

8.3 Descrição de uma Máquina Morfológica

Ainda na década de sessenta, Klein e Serra projetaram a primeira máquina morfológica conhecida: o Texture Analyser [KleSer72]. Desde então, uma família de máquinas similares foram desenvolvidas: desde softwares para computadores convencionais [Läy84; Gratin88; BaBaLo94] até implementações em silício [HuDeBo88; KlePey89] ou tecnologias ópticas [HuJeSa89].

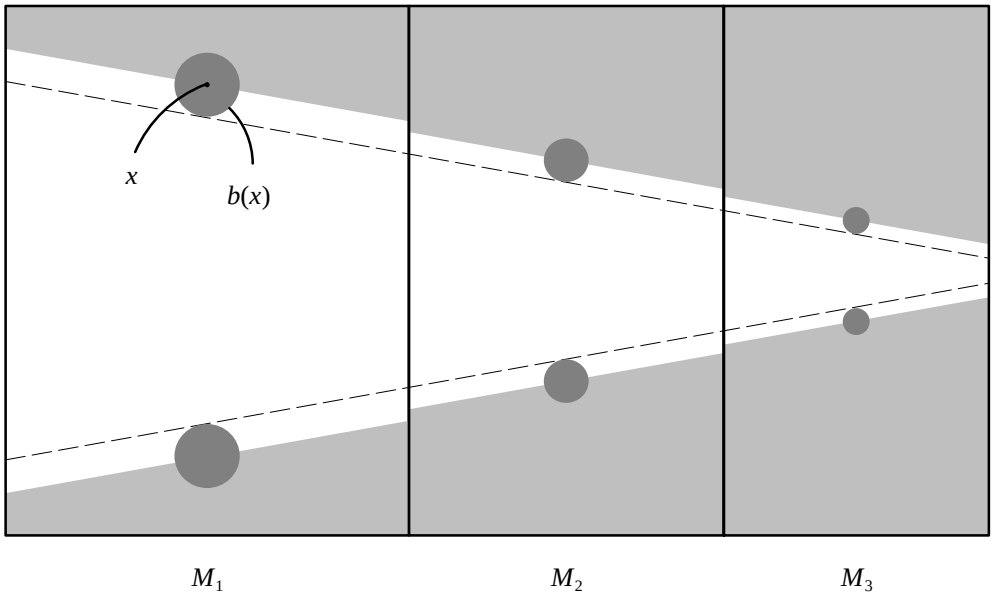


Fig. 8.5 – Erosão adaptativa.

Diremos que uma linguagem formal *L1* é *equivalente* a uma linguagem formal *L2*, se para cada frase de *L1* existe uma frase de *L2* com a mesma semântica e, inversamente, se para cada frase de *L2* existe uma frase de *L1* com a mesma semântica. Hoje em dia, uma *Maquina Morfológica* (*MMach*) é conceituada como uma implementação de uma linguagem formal equivalente a LM.

O usuário enxerga uma MMach como uma linguagem de programação, que dispõe de funções primitivas (i.e., interseção, união, complementação, dilatação, erosão, anti–dilatação e anti–erosão) e estruturas de controle (do, while, if). Um *programa da MMach* nesta linguagem corresponde a uma frase da LM.

O tipo de dado a ser transformado por uma MMach é a imagem binária. Outros tipos de dados auxiliares que aparecem são os elementos estruturantes primitivos e os números inteiros não negativos. Os elementos estruturantes primitivos são os parâmetros das dilatações e erosões localmente c.i.t., enquanto os números inteiros são úteis para o controle de procedimentos iterativos.

Usualmente, por uma questão de eficiência, os operadores de dilatação e erosão são implementados a partir dos respectivos operadores localmente c.i.t., caracterizados por elementos estruturantes primitivos. A partir da dilatação e da erosão, assim criadas, são formadas, respectivamente, a anti–dilatação e a anti–erosão.

Normalmente, estas máquinas dispõem internamente também de recursos para realizar operações sobre os elementos estruturantes primitivos, como a rotação em torno da origem central e a complementação. Observe que a transposição é exatamente uma rotação de 180 graus em torno da origem

A característica central da arquitetura de uma MMach típica é a existência de um processador dedicado para efetuar dilatações, erosões, união, interseção, complementação e comparação de igualdade entre imagens. Este processador é conhecido como *processador morfológico.* A arquitetura desta máquina conta também com dispositivos para aquisição, visualização e armazenamento volátil ou permanente de imagens, além de um processador central que controla todos esses recursos.

A comparação entre imagens é um recurso importante para a implementação de procedimentos iterativos com um número de iterações indefinido a priori. Neste caso, o número de iterações é estabelecido a partir de algum critério de convergência de uma sequência de imagens.

A Figura 8.6 apresenta a arquitetura de uma MMach [Beuche87]. Esta máquina conta com cinco planos

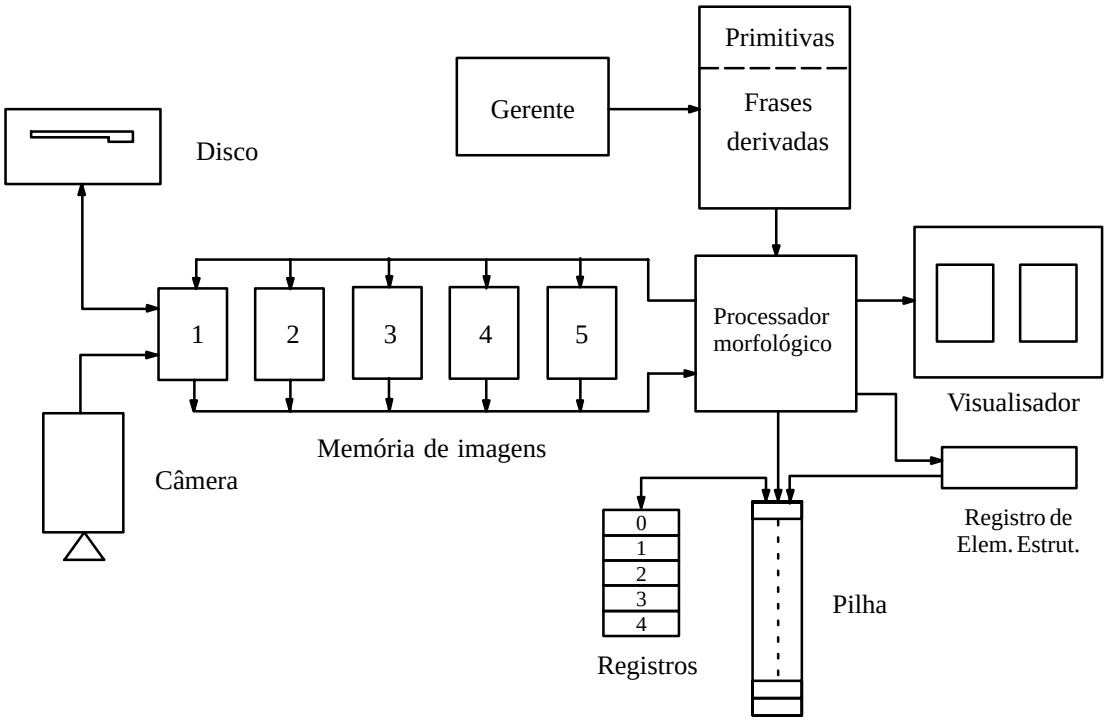


Fig. 8.6 – Arquitetura de uma máquina morfológica.

de memória para o armazenamento de imagens, um processador morfológico, uma pilha para o armazenamento de numeros inteiros e instruções, cinco registros auxiliares que trocam dados com a pilha e um registro para armazenar um elemento estruturante primitivo, além de dispositivos para a aquisição, armazenamento permanente e vizualização de imagens. Todos estes recursos são controlados por um microprocessador de 16 bits.

A dinâmica de uso típico desta máquina envolve: a aquisição pela câmera de uma imagem binária externa; o armazenamento da imagem adquirida no plano de imagem 1; a visualização da imagem adquirida; a programação da máquina para extrair a informação desejada; a execução do programa implementado; o armazenamento em disco dos resultados intermediários e final.

A execução do programa, que é armazenado na pilha, promove a troca de dados entre a pilha e os registros, define os elementos estruturantes usados e estabelece o fluxo de imagens entre os planos de memória e o processador morfológico.

A Figura 8.7 apresenta a estrutura interna do processador morfológico, que é composto por seis processadores internos. Não existem muitas variantes de arquitetura para os processadores de união, interseção, complementação e comparação de igualdade. As nuances mais interessantes aparecem nos processadores de dilatação e erosão.

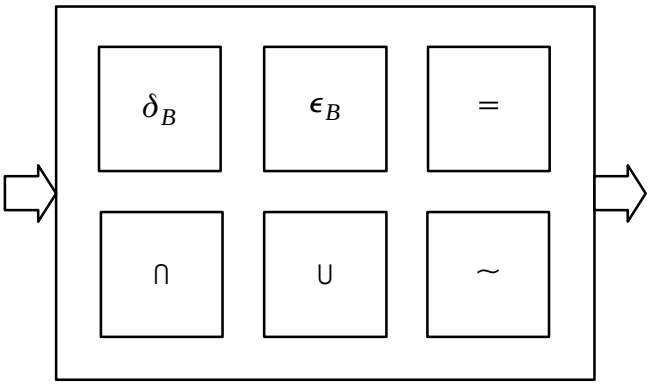


Fig. 8.7 – Processador morfológico.

Os processadores de dilatação e erosão usualmente são compostos por uma família de processadores idênticos, chamados, respectivamente, *processadores primitivos de dilatação e processadores primitivos de erosão*, que realizam, respectivamente, dilatações e erosões c.i.t. por elementos estruturantes primitivos. Esses processadores são organizados em "*pipeline*", em *paralelo*, ou em configurações *híbridas* ("pipeline"–paralelo) e podem atuar simultaneamente sobre a mesma imagem ou sobre imagens distintas. Em algumas arquiteturas conhecidas, são disponíveis recursos de programação que permitem ao usuário redefinir a disposição dos processadores primitivos. A Figura 8.8 apresenta um processador de dilatação, composto por uma família de kl processadores primitivos, organizados em k "pipelines" paralelos, cada um composto por l processadores primitivos.

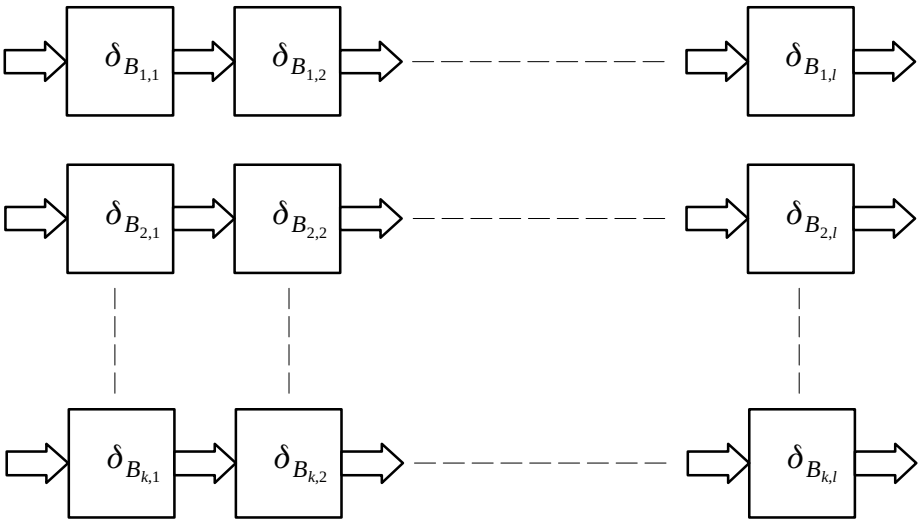


Fig. 8.8 – Processadores de dilataçao em "pipelines" paralelos.

Alguns exemplos de processadores morfológicos são o processador do "Cellular Computer" [Sternb82], desenvolvido na Universidade de Michigan, o processador do sistema MORPHO-PERICOLOR [Bilode86] e o CHIP de tecnologia VLSI [KlePey89], ambos desenvolvidos na École des Mines de Paris.

Os processadores primitivos de diltação e erosão podem ser implementados em duas classes de arquiteturas: uma baseada em translações e uniões ou interseções e outra baseada em operações lógicas de vizinhança. A primeira são implementações das expressões, para todo X e Y em $\mathcal{P}(E)$,

$$\delta_B(Y) = (\bigcup_{b \in B} (Y + b)) \cap E \quad \text{e} \quad \epsilon_B(Y) = (\bigcap_{b \in B} (Y - b)) \cap E$$

e a segunda das expressões, para todo X e Y em $\mathcal{P}(E)$,

$$\delta_B(Y) = \{x \in E : (B^t + x) \cap Y \neq \emptyset\} \text{ e } \epsilon_B(X) = \{y \in E : (B + y) \cap E \subset X\}.$$

Na Figura 8.9, apresentamos um processador primitivo de dilatação e erosão que tem uma arquitetura baseada em translações e uniões ou interseções de planos de bits. Cada ponto do elemento estruturante vai causar uma translação da imagem original e a união ou interseção destas translações será a imagem de saída. O processador conta com um dispositivo para controle de translação, uma matriz, com as mesmas dimensões das imagens, de portas lógicas OR/AND que atuam em paralelo e três planos de bits: um para as imagens transladadas, um para o elemento estruturante e um para o acúmulo dos resultados, intermediários e final. Uma característica dessa arquitetura é que todas as operações primitivas usadas (translações e uniões ou interseções) são operações globais, isto é, se aplicam simultamente sobre toda a imagem.

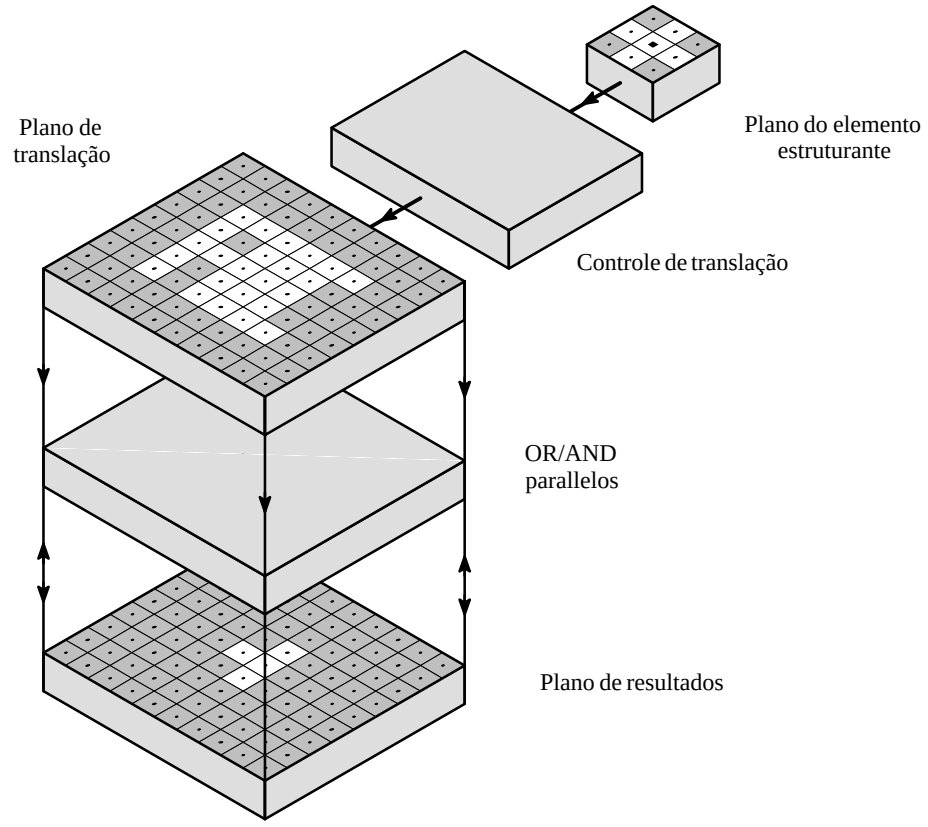


Fig. 8.9 – Processador primitivo baseado em operações globais.

Na Figura 8.10 apresentamos um processador primitivo de dilatação e erosão que tem uma arquitetura baseada no deslocamento de uma matriz 3×3, representando o elemento estruturante primitivo, sobre a imagem e na comparação dos valores lógicos dos elementos dessa matriz com os valores dos correspondentes elementos de matrizes 3×3 extraídas da imagem.

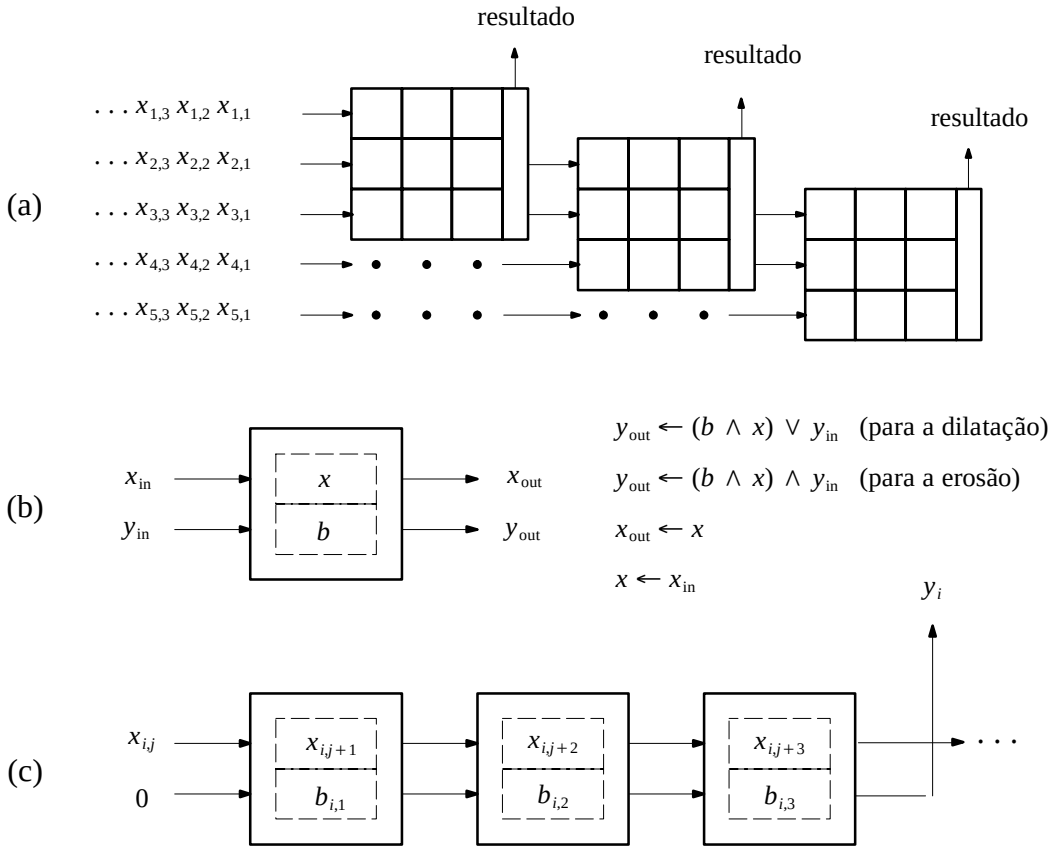


Fig. 8.10 – Processador primitivo baseado em operações de vizinhança.
(a) Fluxo de dados. (b) Célula elementar. (c) Conexão entre células.

A imagem fluirá em blocos de três linhas consecutivas, de forma que cada linha da imagem seja a linha central de exatamente um bloco, por um processador que efetua a comparação entre a matriz extraida da imagem e o elemento estruturante.

Este processador é denominado *processador celular*, pois é composto por 9 células de processamento interligadas.Cada uma das células recebe dois valores lógicos como entrada e produz dois valores lógicos de saída, seguindo as equações apresentadas na Figura 8.10 (b). A váriavel x corresponde ao valor do pixel da imagem que é transmitido para a saída da célula sem ser modificado, enquanto a variável y corresponde a ponderação do estado anterior dessa variável com o resultado da comparação entre o valor de um pixel da imagem e o valor de um ponto do elemento estruturante.

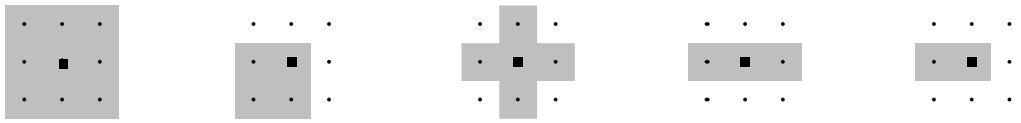
As 9 células são interligadas em três sistemas de três células cada um, conforme esquematizado na Figura 8.10 (c). A saída desses três sistemas vão alimentar a entrada de portas lógicas OR, no caso da dilatação, e portas lógicas AND, no caso da erosão, produzindo o resultado da ação do processador celular.

Para introduzir características de paralelismo à esta arquitetura, pode–se usar um conjunto de processadores celulares dispostos como esquematizado na Figura 8.10 (a). O limite do potencial de paralelismo dessa arquitetura é atingido quando é reservado um processador celular distinto para cada linha da imagem.

Uma característica dessa arquitetura é que as operações efetuadas pelos processadores celulares são operações de vizinhança, isto é, dependem apenas de pontos vizinhos. Este tipo de arquitetura pertence a classe das arquiteturas sistólicas, que são bem adptadas para implementações em CHIPS de tecnologia VLSI [Song84].

A decomposição sequencial de elementos estruturantes tem implicação na complexidade das implementações da dilatação e da erosão. Usualmente, a implementação de sequencias de dilatações e erosões é mais simples do que a equivalente implementação direta. Vamos verificar essa afirmação medindo a complexidade de implementações diversas.

Seja B um dos elementos estruturantes apresentados na Figura 8.11. Na Tabela 8.3 apresentamos um



B_1 B_2 B_3 B_4 B_5

Fig. 8.11 – Alguns elementos estruturantes típicos.

estudo da complexidade da implementação da dilatatação e da erosão por nB no processador da Figura 8.9 [Marago85]. A medida de complexidade adotada é, respectivamente, o número de operações OR–paralelos efetuadas ou, equivalentemente, o número de uniões entre planos de bits e o número de operações AND–paralelos ou, equivalentemente, o números de interseções entre planos de bits. A primeira coluna da tabela identifica o elemento estruturante B, a segunda apresenta a medida de complexidade para a implementação direta da dilatação por nB, a terceira a complexidade para a implementação sequencial de n dilatações por B e a quarta a complexidade para a implementação sequencial de n dilatações por B, onde cada dilatação por B é implementada como duas dilatações sequenciais.

Tabela 8.3 – COMPLEXIDADE DAS IMPLEMENTAÇÕES.

Elem. Estruturante	Implementação 1	Implementação 2	Implementação 3
B_1	$4n^2 + 4n$	$8n$	$4n$
B_2	$n^2 + 2n$	$3n$	$2n$
B_3	$2n^2 + 2n$	$4n$	–
B_4	$2n$	$2n$	–
B_5	n	n	–

Examinando a Tabela 8.3, observamos que a complexidade da implementação diminui a medida que a decomposição sequencial se acentua. Por exemplo, no caso da dilatação ou erosão por um quadrado 7×7, a primeira implementação envolve 48 operações, a segunda 24 e a terceira apenas 12. Notamos também que essa economia fica mais significativa a medida que o número de pontos do elemento estruturante a ser decomposto cresce.

Os processadores dedicados comparados com as máquinas convencionais são, por um lado, muito eficientes e, por outro lado, muito caros. Por essa razão existem muitos softwares para máquinas convencionais que emulam máquinas morfológicas. Normalmente, o uso de hardware dedicado fica restrito às aplicações que exigem resposta em tempo real, enquanto o uso de emulações ocorre nas outras aplicações, menos exigentes em termos de desempenho dos processadores.

Uma aplicação que normalmente exige resposta em tempo real é o controle de qualidade de produtos industrializados. Os produtos a serem inspecionados são dispostos sobre uma esteira rolante e passam por uma câmera estrategicamente colocada, que alimenta uma máquina morfológica. Esperasse que o sistema trate os dados e forneça um diagnóstico sobre o produto em um intervalo de tempo suficientemente pequeno para não afetar o fluxo de objetos pela esteira.

Uma classe de aplicações menos exigente com relação ao tempo de resposta é a análise de imagens geológicas, como, por exemplo, a estimativa da porcentagem de volumes ocos em uma rocha. A execução deste tipo de tarefa exige que um especialista observe por um microscópio óptico sistematicamente dezenas de lâminas e identifique em cada uma as cavidades presentes. Este trabalho pouco criativo normalmente consome várias horas do especialista, que poderiam ser usadas em tarefas mais nobres. Assim, uma resposta confiável após alguns minutos de processamento é considerado um resultado muito razoável.

Outro tipo de aplicação que não exige resposta em tempo real é o projeto de programas que solucionam problemas de tempo real. Quando um especialista em Morfologia Matemática recebe um problema novo de Análise de Imagens ele precisa fazer uma série de experimentos sobre as imagens adquiridas até poder propor uma solução satisfatória. Para este tipo de tarefa o tempo de resposta é considerado razoável quando não influe no rendimento do trabalho do especialista, o que é um requisito bem menos forte, por exemplo, do que não interromper o fluxo de produtos por uma esteira. Uma vez encontrada uma solução para o problema de Análise de Imagens, resta verificar se ela é viável, isto é, se, quando fosse implementada em uma máquina mais eficiente, atenderia os requisitos de desempenho. Isto é feito calculando–se a performance que o programa teria no ambiente real a partir da performance medida no ambiente de projeto e do conhecimento prévio da relação entre as performances das duas máquinas.

O núcleo dos softwares que emulam máquinas morfológicas são as funções que emulam o processador morfológico. Estas funções serão usadas intensivamente e, portanto, devem ser otimizadas ao máximo. Com essa motivação já foram propostos vários algoritmos para a implementação dessas funções. Esses algoritmos se dividem em duas classes: os algoritmos baseados em operações globais [PipTan89; LiaWon92; Gratin93] e os algoritmos baseados em operações de vizinhança [VliBen88; Schmit89; Ornell92].

Todos os algoritmos que serão apresentados usam a estrutura de dados matricial para representar as imagens ou partes delas. As matrizes serão implementadas como um vetor (ou “array”). Um elemento qualquer da matriz será disposto no vetor segundo a sua distância da origem, medida como o comprimento do caminho percorrido da origem até o elemento, caminhando sobre as linhas da matriz da esquerda para a direita e de cima para baixo (ver Seção 4.1). Considerando uma matriz A de $m \times n$ posições e um vetor V de mn elementos, temos $A(i,j) = V(in + j)$ ($i \in \{0,...,m-1\}$ e $j \in \{0,...,n-1\}$). Denominaremos *endereço de um elemento* a sua posição no vetor, por exemplo, o endereço do elemento $A(i,j)$ é $in + j$.

A forma convencional de representar uma imagem binária é como uma matriz cujos elementos são os pixels da imagem. A partir dessa estrutura de dados pode–se estabelecer algoritmos baseados em operações globais ou operações de vizinhança, simplesmente pela implementação direta das definições das operações e operadores [Barrer87]. Embora essas algoritmos sejam muito simples, eles se mostram pouco eficientes. A complexidade dos algoritmos que realizam as operações é proporcional às dimensões da imagem, enquanto a complexidade dos algoritmos que realizam os operadores de dilatação e erosão é proporcional ao produto das dimensões da imagem pelo cardinalidade do elemento estruturante.

Exercício 8.1 (algoritmos convencionais para operadores elementares) – Usando como estrutura de dados para representar as imagens uma matriz de pixels, implemente os operadores de dilatação e erosão por elementos estruturantes primitivos. □

Os algoritmos que não usam a estrutura de dados convencional serão chamados *algoritmos rápidos*. Dentre os algoritmos rápidos baseados em operações globais, o mais popular é aquele que representa vários pixels consecutivos de uma linha compactados em uma única palavra. A Figura 8.12 ilustra o uso desta estrutura de dados para representar uma imagem binária.

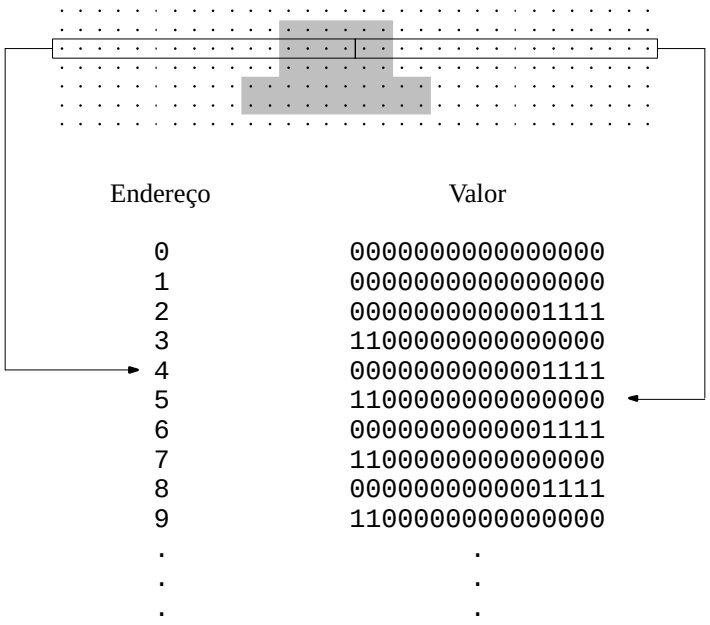


Fig. 8.12 – Representação compactada de uma imagem binária.

O fato que esse tipo de algoritmo explora é o paralelismo intrínsico dos processadores convencionais de palavras de 16, 32 ou 64 bits. Cada vez que uma palavra de n bits é processada ocorrem n operações lógicas em paralelo, uma para cada bit da palavra. Uma vez que nessa estrutura de dados cada bit representa um pixel, n pixels são tratados em paralelo.

Na Tabela 8.4 apresentamos o algoritmo que realiza a união entre imagens binárias, compactadas em palavras de 16 bits. A complexidade desse algoritmo é dezesseis vezes menor do que a complexidade de um algoritmo que representa cada pixel em uma palavra diferente.

Na Tabela 8.5 apresentamos um algoritmo que realiza a erosão pelo elemento estruturante $\begin{bmatrix} 0 & 0 & 0 \\ 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}$. Nesta Tabela, a notação *P SHR 1* representa a operação de translação da palavra *P* de 1 bit para a direita. A dinâmica desse algoritmo é a mesma do processador morfológico apresentado na Figura 8.9, ou seja, a translação horizontal de imagens por valores definidos pelo elemento estruturante e a interseção dessas translações. O ponto crítico deste algoritmo é a operação de translação entre palavras vizinhas: é preciso garantir que o primeiro pixel da palavra receba o último pixel da palavra vizinha (contando no sentido da translação) e que os demais recebam os bits vizinhos dentro da própria palavra. Este cuidado exige algumas operações lógicas adicionais, que impedem que a complexidade do algoritmo caia 16 vezes.

Tabela 8.4 – ALGORITMO DE UNIÃO DE DUAS IMAGENS BINÁRIAS.

- **Dados**
  - X e Y são as imagens binárias compactadas $m \times (n/16)$ a unir.
  - Z é a imagem binária compactada $m \times n$ resultante.
- **Laço principal**

```
Para i de 0 à m − 1
  Para j de 0 à n/16 − 1
    Z(in + j) ← X(in + j) OR Y(in + j)
  Fimpara
Fimpara
```

A extensão deste algoritmo para um elemento estruturante qualquer, que seja um subconjunto do quadrado elementar é simples, basta considerar cumulativamente todos os sentidos de translação definidos pelo elemento estruturante. Note que as translações na direção vertical não têm os problemas apontados paras as horizontais.

Tabela 8.5 – ALGORITMO DE EROSÃO POR UM SEGMENTO HORIZONTAL DE TAMANHO 1.

- **Dados**
  - X é a imagem binária compactada $m \times (n/16)$ original.
  - Z é a imagem binária compactada $m \times (n/16)$ resultante.
  - *carry* é uma variável auxiliar de 16 bits.
- **Laço principal**

```
Para i de 0 à m − 1
  carry ← 0
  Para j de 0 à n/16 − 1
    Z(in + j) ← ((X(in + j) SHR 1) OR carry) AND X(in + j)
    Se ((X(jn + i) AND 1) = 0) então
      carry ← 0
    Senão
      carry ← 2^15
    Fimse
  Fimpara
Fimpara
```

Exercício 8.2 (algoritmos rápidos para operadores elementares) – Usando como estrutura de dados para representar as imagens uma matriz de pixels compactados, implemente os operadores de dilatação e erosão por elementos estruturantes primitivos. □

Uma *fila* é uma estrutura de dados, que organiza os objetos sequencialmente segundo a ordem de inserção. O modelo intuitivo de uma fila é o de uma fila de espera em que as pessoas no início da fila são servidos primeiro e as pessoas que chegam entram no fim da fila. Existe uma ordem linear para filas que é a "ordem de chegada". Um possível conjunto de operações, definido para um tipo abstrato de dados FILA, é definido na Tabela 8.6 [Zivian93].

Tabela 8.6 – OPERAÇÕES DEFINIDAS PARA O TIPO ABSTRATO FILA.

- **Operações aplicadas ao objeto *X* do tipo FILA.**
  - vazio(*X*): retorna *true* se *X* é uma fila sem elementos e *false* caso contrário.
  - enfileira(*x*, *X*): insere o ítem que tem endereço *x* no final da fila *X*.
  - desinfileira(*X*): retorna o endereço do primeiro ítem da fila *X*.

Dentre os algoritmos rápidos baseados em operações de vizinhaça um dos mais interessantes é o que usa uma estrutura de dados híbrida para representar a imagem: matriz, para representar os pixels do interior, e fila de pixels, para representar os pixels da borda. A Figura 8.13 exemplifica a representação de uma imagem pela estrutura de dados híbrida.

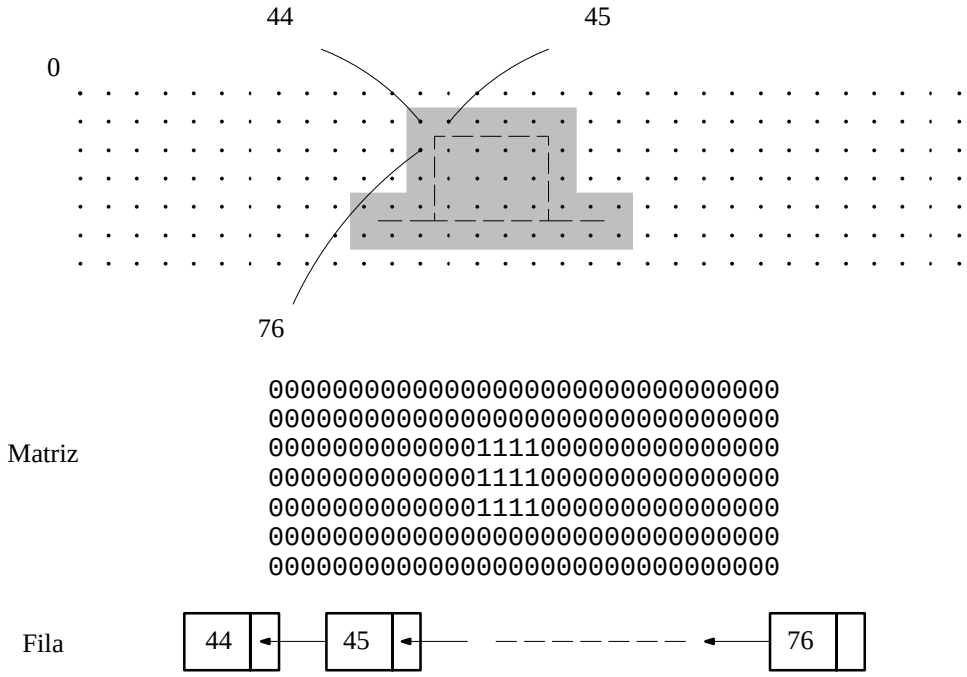


Fig. 8.13 – Representação híbrida de uma imagem binária.

O fato que esse tipo de algoritmo explora é que os operadores de dilatação e erosão por elementos estruturantes primitivos modificam apenas os pixels que se encontram na vizinhança da borda da imagem. Assim, basta computar o novo estado desses pixels para construir o operador.

Este tipo de algoritmo tem uma característica singular: a sua complexidade é proporcional ao número de pixels da borda da imagem tratada, isto é, quanto menos pixels existirem na borda mais eficiente será o algoritmo.

Se comparados com os respectivos algoritmos convencionais, os algoritmos de vizinhança para a dilatação e a erosão têm um ganho de eficiência significativo para um grande número de imagens, pois os primeiros têm complexidade proporcional ao número de pixels da imagem e os segundos têm complexidade proporcional ao número de pixels da borda da imagem.

Os algoritmos das operações de união, interseção e complementação para a estrutura híbrida é menos eficiente do que os respectivos algoritmos para imagens convencionais. Isso porque além de realizar uma

operação sobre a imagem, para computar o interior da imagem resultante, é preciso também realizar operações sobre filas, para computar os pixels das bordas da imagem resultante.

Apesar da estrutura híbrida levar a algoritmos para as operações menos eficientes que os convencionais, o ganho obtido nos operadores de dilatação e erosão garante um ganho de eficiência global do processador. O ganho de eficiência é acentuado sobremaneira na realização de operadores formados pela composição de dilatações e erosões

A Tabela 8.7 apresenta um algoritmo para a união de imagens representadas na estrutura de dados híbrida. O algoritmo é composto por dois blocos principais: computação do interior da imagem resultante e computação da borda da imagem resultante. A computação da borda também é dividida em dois blocos: computação dos pixels que não estão no interior, respectivamente, das imagens de entrada X e Y.

Tabela 8.7 – ALGORITMO DE UNIÃO DE DUAS IMAGENS BINÁRIAS.

- **Dados**
  - $\overset{o}{X}$ e $\overset{o}{Y}$ são o interior das imagens binárias $m \times n$ a unir.
  - $\overset{o}{Z}$ é o interior da imagem binária $m \times n$ resultante.
  - ∂X, ∂Y e ∂Z são as filas dos pixels das bordas das imagens.
  - *adress* é uma variável auxiliar de 16 bits.
- **Função**

```
filtrafila( ∂X, Y°, ∂Z)
  Enquanto vazio(∂X) = false faça
    adress ← desinfileira(∂X)
    Se Y°(adress) = 0 então
      enfileira(adress, ∂Z)
    Fimse
  Fimpara
```

- **Computa** $\overset{o}{Z}$

```
Para i de 0 à m − 1
  Para j de 0 à n − 1
    Z°(in + j) ← X°(in + j) OR Y°(in + j)
  Fimpara
Fimpara
```

- **Computa** ∂Z

```
filtrafila( ∂X, Y°, ∂Z)
filtrafila( ∂Y, X°, ∂Z)
```

A Tabela 8.8 apresenta um algoritmo de erosão por um quadrado elementar para imagens representadas na estrutura de dados híbrida. O algoritmo consiste, para cada pixel da fila de pixels de borda da imagem original, em eliminar os pixels do interior da imagem original que fazem partem da vizinhança do pixel e enfileira–los em uma nova fila, que conterá os pixels da borda da imagem resultante. Este algoritmo pode ser generalizado para erosões e dilatações por elementos estruturantes primitivos quaisquer [Ornell92].

Exercício 8.3 (algoritmos rápidos para operadores elementares) – Usando como estrutura de dados para representar as imagens um par formado por uma matriz, que representa os pixels do interior da imagem, e uma lista, que representa os pixels da borda da imagem, implemente os operadores de dilatação e erosão por elementos estruturantes primitivos. □

Exercício 8.4 (algoritmos para operadores primitivos) – Compare o desempenho das implementações realizadas dos Exercícios 8.1 a 8.3. □

Em todas as classes de algoritmos descritas, os elementos estruturantes constumam ser representados por uma estrutura de dados que contem a cardinalidade do conjunto e os valores das coordenadas de cada ponto. Para otimizar um pouco mais os algoritmos de dilatação e erosão, podemos armazenar as coordenasdas dos pontos em registradores e implementar um trecho de código específico para elementos estruturantes com número de pontos diferentes, de zero a nove.

O uso apenas de elementos estruturantes primitivos, além de ser suficiente para realizar qualquer operador elementar e, em muitos casos, levar a decomposições mais simples, permite também otimizar a implementação dos processadores primitivos de dilatação e erosão, pelo uso da informação a priori do tamanho limite dos elementos estruturantes. Por essas razões, a maior parte das MMach's conhecidas dispõem apenas de processadores primitivos de dilatação e erosão que realizam esta classe de operadores.

Tabela 8.8 – ALGORITMO DE EROSÃO PELO QUADRADO ELEMENTAR.

- **Dados**
  - $\overset{o}{X}$ é o interior das imagens binárias m x n original e resultante.
  - ∂X e ∂Y são as filas dos pixels das bordas, respectivamente, da imagem original e resultante.
- **Laço principal**

```
Enquanto vazio(∂X) = false faça
  adress ← desinfileira(∂X)
  Para todo i ∈ { − 1, 0, 1} faça
    Para todo j ∈ { − 1, 0, 1} faça
      Se X°(adress − in + j) = 1 então
        enfileira(adress, ∂Y)
        X°(adress − in + j) ← 0
      Fimse
    Fimpara
  Fimpara
Fimenquanto
```

Como os operadores que desejamos implementar são operadores c.i.t. e os elementos estruturantes são escolhidos como subconjuntos do quadrado elementar, chegamos a nove regiões da imagem com comportamento peculiar: uma central, quatro laterais e quatro cantos.

Mais precisamente, como foi visto na Seção 8.2, um operador elementar c.i.t. pelo quadrado elementar pode ser decomposto em união (no caso da dilatação) ou interseção (no caso da erosão) de nove operadores elementares (da mesma classe) localmente c.i.t. definidos por nove elementos estruturantes B_i e nove máscaras M_i especificando nove regiões do domínio da imagem.

A Figura 8.14 apresenta essas nove regiões e o valor, em cada uma delas, dos B_i. Para otimizar ainda mais o algoritmo, usualmente, cada uma dessas regiões têm tratamento especial.

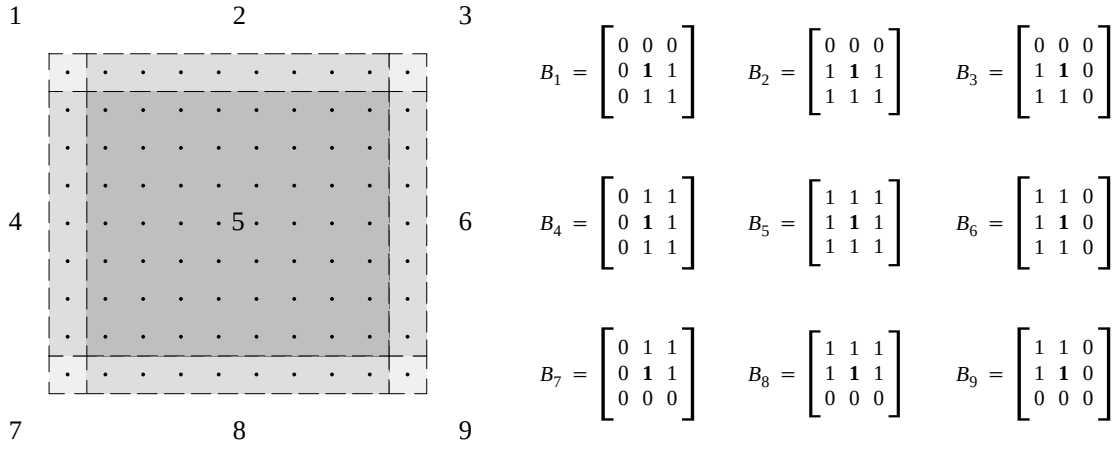


Fig. 8.14 – As nove regiões da imagem.

As MMach's, implementadas em hardware ou emuladas em software, vão ser enxergadas pelo usuário através de uma linguagem de programação. Nas máquinas morfológicas que temos conhecimento as funções primitivas dessa linguagem acionam diretamente o processador morfológico, porém, em máquinas mais sofisticadas, elas poderiam acionar uma camada inferior de software, que realizaria a decomposição dos operadores elementares em termos de erosões e dilatações localmente c.i.t. por elementos estruturantes primitivos e distribuiria a execução desses operadores pelos processadores primitivos.

Hoje em dia, o projeto de programas para resolver problemas de análise de imagens ainda exige que o especialista desenvolva muitos experimentos computacionais antes de chegar a um resultado satisfatório. A partir do resultado de um experimento, ele identifica problemas no seu programa, tenta corrigi–lo e efetua um novo experimento para verificar se o efeito da modificação foi o experado, e assim vai evoluindo, até chegar a uma solução. Esta dinâmica de trabalho impõe certos requisitos para as interfaces homem–máquina das MMach's.

A interface homem–máquina de uma MMach deve prover um ambiente que permita a edição e a execução ágil de programas, que podem depender de uma famílias de outros programas implementados em momentos anteriores da sessão de trabalho ou mesmo em outros sessões de trabalho. Ainda, essas facilidades de programação não podem penalizar sensivelmente o desempenho dos programas.

Dois tipos de ambientes que atendem a esses requisitos costumam ser adotados como interface homem–máquina para as MMach's: um baseado em uma linguagem interpretada e outro baseado em uma linguagem de programação visual.

As linguagens interpretadas são bem adaptadas para a implementação e execução ágeis de programas, especialmente aquelas linguagens que permitem o uso de qualquer programa existente como uma nova função. O fato dos programas não precisarem passar por uma fase de compilação e poderem reaproveitar facilmente programas existentes são pontos importantes que contribuem para o aumento da produtividade do desenvolvimento de software.

Um ponto negativo no uso de linguagens interpretadas é que os programas não ficam tão eficientes quanto poderiam. Por exemplo, um conjunto de comandos que se repete várias vezes é interpretado toda vez que é executado, o que naturalmente é uma causa de desperdício de tempo. A solução que se constuma adotar é usar linguagens que tanto podem ser compiladas como interpretadas, isto é, para cada função existente o programador pode escolher se quer compila–la ou interpreta–la. Com esse recurso o especialista

pode estabelecer um compromisso ideal entre a produtividade de desenvolvimento de software e a velocidade de execução dos programas, de forma a minimizar o tempo global de projeto (i.e., implementação e testes). Naturalmente, essas linguagens dispõem também de editores de texto integrados.

Uma outra propriedade importante das linguagens interpretadas é que todos os objetos criados durante a execução dos programas continuam disponíveis para o especialista. Assim, é possível acompanhar facilmente todos os estados desejados dos programas, o que é um recurso essencial para o projeto interativo de programas.

Existem várias MMach's que têm esse tipo de interface [Bilode86; Schmit86; Beuche87; Gratin88]. A Tabela 8.9 mostra a implementação de um programa, que realiza *n* erosões sucessivas por um mesmo elemento estruturante fixo e apresenta as imagens original e resultante, na linguagem interpretada que é a interface homem–máquina do sistema MORPHOPERICOLOR. Esta linguagem é similar ao Forth, na verdade, é um subconjunto do Forth incrementado com as funções de gerenciamento dos recursos do sistema (processador morfológico, aquisição, visualização, etc.). Este tipo de linguagem é bem adaptado para arquiteturas como a da Figura 8.6, porque implementações interessantes do Forth podem ser obtidas usando uma pilha como estrutura de dados central [Loelig81].

Tabela 8.9 – COMPOSIÇÃO DE *N* EROSÕES EM UMA LINGUAGEM INTERPRETADA.

```
:nerosão
    OUTPUT IS       /* declaração */
    INPUT IE
    PARAM N

    IE IS MOVE      /* procedimento */
    N 1
    DO
       IS IS ERODE
    LOOP
    IE DISPLAY
    IS DISPLAY
;
```

Outro tipo de interface bem adaptada para as MMach's são as linguagens de programação visual [Chang87; RaArSa90]. Este tipo de interface permite que o usuário interaja graficamente com o sistema para uso dos recursos e programação. O paradigma principal deste tipo de linguagem é descrever programas como grafos orientados. As arestas dos grafos representam os caminhos que os dados devem percorrer e os nós as funções que os transformam. A exemplo das linguagem procedurais, as linguagens visuais também dispõem de recursos avançados como mecanismos de controle e definição de subprogramas.

As linguagens visuais são bem adequadas para a programação rápida de pequenos programas compostos por chamadas de outros programas existentes e também permitem a fácil observação dos estados internos dos programas executados. Por outro lado, elas não são adequadas para o desenvolvimento de programas complexos que dependem de muitos outros programas visuais. A solução adotada é sempre usar uma linguagem procedural compilada em conjunto com uma linguagem visual. Com esses recursos em mãos, o especialista pode estabelecer uma relação de compromisso ideal entre as duas alternativas.

O sistema KHOROS [RaArSa90] é um sistema para processamento de imagens em ambientes padrões UNIX e XWINDOW, que tem uma interface de programação visual e usa a linguagem C como linguagem procedural.

Na estrutura do KHOROS os operadores são representados por funções, escritas em C, que são agrupadas em uma biblioteca e por programas principais que chamam estas funções. Muitas destas funções são criadas pelo encadeamento de outras funções da biblioteca e para cada função da biblioteca existe um programa principal, que chama apenas essa função. Nesta estrutura, a linguagem de programação visual tem o mesmo papel dos recursos de programação em batch do UNIX, isto é, controla a chamada de programas principais. Naturalmente, o KHOROS também dispõem de recursos avançados para a edição de programas em C, a geração de interfaces visuais e a instalação das novas funções na biblioteca.

A "toolbox" MMach [BaBaLo93], que desenvolvemos em colaboração com o professor Roberto de Alencar Lotufo, é uma máquina morfológica que usa a plataforma KHOROS. No apêncice A, apresentamos uma descrição mais completa dessa "toolbox".

A Figura 8.15 exemplifica o uso da "toolbox" MMach, apresentando um programa visual que executa *n* erosões sucessivas por um mesmo elemento estruturante e apresenta as imagens original e resultante.

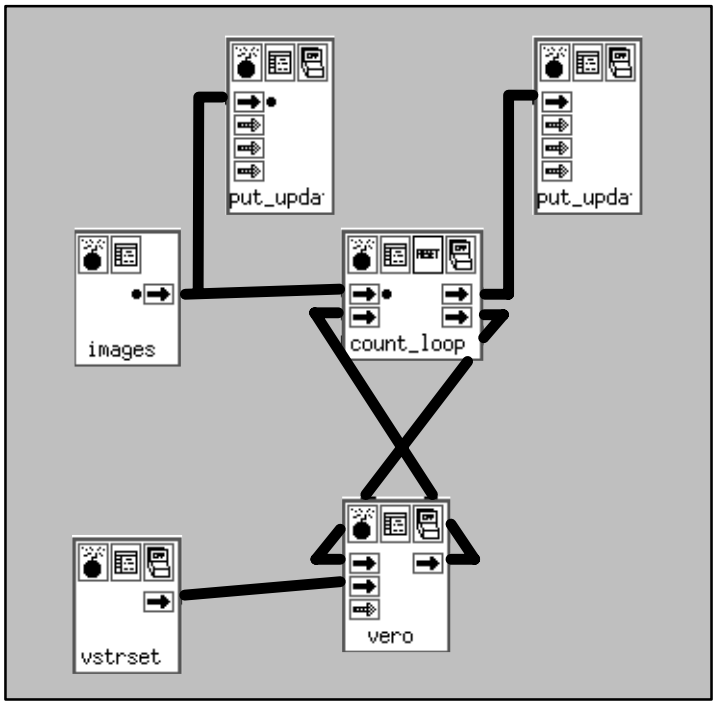


Fig. 8.15 – Composição de *n* erosões em uma linguagem visual.