Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas

NEUCIMAR JERÔNIMO LEITE

DCC/IMECC/UNICAMP
Caixa Postal 6065
13081-970 Campinas - SP
neucimar@dcc.unicamp.br

Abstract. In this paper we present some computational models for iterative cellular automaton for image processing applications. We introduce some concepts such as memory splitting, conditional functions, dynamic neighborhood and supervisor automaton. The models defined lead to a parallel language structure that can express low-level image processing in a clear and concise way. The language allows a transparent description of the algorithms and can be easily expandable to reflect the needs of people working in different branches of image processing.

1 Introdução

A máquina de Turing é um modelo de computação que define um sistema seqüencial como sendo constituído de um conjunto de estados finitos. Os dados de entrada são *símbolos* que podem ser lidos ou modificados durante a evolução do sistema. Estas transformações podem ser descritas em termos de um programa ou seqüência de instruções que define o resultado final obtido quando da execução da última instrução do programa. Nós diferenciamos, assim, dois níveis de abstração na descrição de um algoritmo: um representado por um *modelo de computação formal* e o outro, por uma *estrutura de programação* constituída de dados e instruções reais.

A descrição dos modelos formais é muito importante para a análise do tempo de execução de algoritmos paralelos de processamento de imagens. Por exemplo, em análise de imagens, os modelos de Rosenfeld [Rosenfeld-83] definem um autômato como um conjunto de células cuja memória é proporcional ao logaritmo do número de células da rede. Neste caso, um grande número de operadores característicos de análise de imagens, tais como histograma, rotulação, transformada do eixo medial, cálculo da área, do perímetro, diâmetro de uma região, etc, tem uma complexidade de cálculo proporcional ao diâmetro da matriz celular.

De maneira geral, os modelos formais, expressando operações efetuadas por um autômato celular são importantes para a compreensão e descrição das classes de algoritmos executadas por sistemas maciçamente paralelos. No entanto, o alto grau de abstração (como o modelo de Rosenfeld) e/ou especificidade (modelos limitados a arquiteturas específicas não conduz a uma estrutura de linguagem flexível que exprima os diversos algoritmos de processamento de imagens de forma *natural* e *transparente*.

Nós apresentamos, a seguir, alguns modelos formais de autômatos celulares iterativos para processamento de imagens. Estes autômatos estão relacionados com arquiteturas paralelas SIMD do tipo MPP, DAP, CLIP, CAAP [Fountain-87], etc. Os modelos discutidos apresentam diferentes recursos computacionais e consideram alguns problemas relacionados com a descrição dos algoritmos de processamento de baixo nível de imagens. Estes modelos conduzem a uma estrutura de linguagem paralela que possibilita, entre outras, uma representação natural das operações executadas em processadores matriciais, assim que uma descrição transparente destas operações.

Este artigo está organizado da seguinte forma. A seção 2 define os modelos dos autômatos para processamento de imagens. A seção 3 apresenta a estrutura da linguagem decorrente dos modelos lógicos definidos, e dá alguns exemplos de representação algorítmica. Finalmente a seção 4 tece algumas considerações gerais sobre o trabalho.

2 Autômato celular para processamento de imagens

Visando definir alguns modelos formais para processamento de imagens, nós retomamos, mais precisamente, a definição geral de autômato celular apresentada por Rosenfeld [Rosenfeld-79]. Um autômato celular é um conjunto **C** de células que, num dado instante, se encontram num determinado estado. Uma configuração do autômato é uma aplicação de

C em **Q**, onde **Q** é o conjunto de todos os estados possíveis. Uma configuração **e** especifica o estado **e(c)** de cada célula c, c ∈ **C**. Cada uma destas células pode ler o estado de n células vizinhas. A evolução do autômato é caracterizada por uma *função de transição* δ. Definidas a configuração inicial e a função δ, o autômato se comporta como um sistema determinista em que para uma configuração dos estados das células, e uma dada transformação, existe uma única configuração definindo o estado final do autômato.

Uma função de transição é uma aplicação δ de $\mathbf{Q}^{n+1} \rightarrow \mathbf{Q}$. Aplicando-se uma tal função a uma configuração **e** do autômato, a nova configuração **e'** é tal que $\mathbf{e'(c)} = \delta[\mathbf{e(c)}, \mathbf{e}(c_1),...,\mathbf{e}(c_n)]$, $c_1, c_2,...,c_n$ é o conjunto das células vizinhas de c.

Consideremos o conjunto $\Delta = \{\delta_1, \delta_2,...,\delta_p\}$, onde δ_i é uma aplicação de $\mathbf{Q}^{n+1}$ em **Q**. Nós definimos um programa do autômato como sendo uma seqüência do tipo $(\delta_{i_1}, \delta_{i_2},...,\delta_{i_r})$, com $\delta_{iq} \in \Delta$. A configuração final é obtida aplicando-se $\delta_{i_1}, \delta_{i_2},...,$ δ_{i_r}, sucessivamente [Leite 92].

A potência computacional de um autômato $\mathbf{A}_i$ depende das operações δ realizáveis e é caracterizada pelo conjunto das aplicações $\Delta_i \subset \Delta$. Um autômato $\mathbf{A}_j$, caracterizado por Δ_j, é mais potente que $\mathbf{A}_i$ se $\Delta_i \subset \Delta_j$ e $\Delta_i \neq \Delta_j$.

Nós definimos um *autômato geral* para processamento de imagens como sendo um autômato representado pelo conjunto $\mathbf{Q} = P \cup P'$ de estados, e pelo conjunto Δ de funções primitivas. P é o conjunto dos estados correspondentes à imagem inicial e P' é a memória auxiliar das células.

O problema que se coloca quando da implementação de um autômato celular (ou de uma máquina que o simule) refere-se ao bom compromisso entre:

- o custo material associado ao tamanho da memória e à complexidade do conjunto de funções Δ.
- a complexidade computacional do autômato.

A seguir, nós abordaremos estes aspectos a partir de alguns exemplos de modelos de autômatos que são casos especiais do autômato geral definido acima. Para isto consideraremos o problema que consiste a restringir o conjunto das funções executáveis enquanto tentamos aumentar o conjunto dos algoritmos efetivamente implementáveis pelo autômato.

2.1 Autômato minimal

Inicialmente, observemos que o autômato mais simples, do ponto de vista da memória, é o *autômato minimal* cuja memória é $\mathbf{Q} = P$.

Um exemplo interessante deste tipo de autômato é dado pela Transformada Simples de Golay [Golay-69]. Esta transformada representa operações sobre imagens considerando apenas um plano de memória que se comporta, ao mesmo tempo, como memória de entrada e saída do sistema.

Evidentemente, uma restrição muito severa, do ponto de vista da memória, reduz o custo das células em detrimento dos recursos funcionais da arquitetura. Por exemplo, no caso em que esta memória não é suficiente para guardar as coordenadas das células na rede, é impossível ao autômato identificá-las de maneira unívoca e, conseqüentemente, executar algoritmos que considerem a distribuição espacial dos objetos na imagem, tais como o cálculo do centro de gravidade, a localização dos pontos de máximo em operações de correlação, transformada de distância, etc.

2.2 Autômato com planos de memória

Uma maneira de restringir o conjunto das funções δ_i é de particionar **Q** em diferentes subconjuntos (planos de memória) e de considerar apenas certas restrições de funções a estes planos de memória.

Um *autômato com planos de memória* é um autômato geral tal que $\mathbf{Q} = P \cup Q^1 \cup Q^2 \cup...\cup Q^j$, ($P, Q^1, Q^2,...,Q^j$ são os planos de memória do autômato), e tal que uma função $\delta_i \in \Delta$ é:

- uma aplicação do tipo $X^{n+1} \rightarrow Y$ com $X, Y \in \{P, Q^1,...Q^j\}$. Nós representamos esta aplicação por $\delta_i(X \rightarrow Y)$.
- ou uma aplicação do tipo $XxY \rightarrow Z$ com X, Y, $Z \in \{P, Q^1,...,Q^j\}$. Nós representamos esta aplicação por $\delta_i((X,Y) \rightarrow Z)$.

A função δ_i, cuja execução depende de parâmetros $p_1, p_2,...,p_j$, é indicada por $\delta_i(; p_1, p_2,...,p_j)$. Esta representação paramétrica será utilizada na especificação de funções de convolução, operações morfológicas, reconhecimento de padrões, etc.

Uma aplicação do tipo $\delta_i(X \rightarrow Y)$ corresponde à uma *função de vizinhança* cujo resultado, armazenado em Y, depende do estado da célula e do estado das células vizinhas no plano X. Exemplos destas aplicações são o cálculo do *max*, da *média* e da *mediana* da vizinhança.

Uma aplicação do tipo $\delta_i((X,Y) \rightarrow Z)$ corresponde a uma *função pontual* entre dois planos X e Y, com o resultado armazenado em Z. Exemplos destas aplicações são as funções monodiádicas do tipo *inversão, incremento*, e as funções diádicas tais como *adição, subtração* e *funções boleanas.*

Devido à restrição das funções de transição, visando limitar sua complexidade computacional, um autômato com planos de memória é menos potente que um autômato geral (se os dois autômatos pos-

suem mesma memória **Q**).

2.3 Autômato com funções condicionais

Um autômato com planos de memória é um autômato tal que, a cada instante, todas as células executam a mesma função δ, sem nenhum nível de autonomia. Uma maneira de aumentar a funcionalidade do autômato é através da noção de função condicional.

Um *autômato com função condicional* é um autômato com planos de memória contendo um plano S distinto, $S \in \{Q^1, Q^2, ..., Q^j\}$, e tal que a execução das funções de transição pode ser condicionada ao estado deste plano. Uma função δ_i(; if S=k) é uma função que só é executada por uma célula c se o valor de S for igual a k.

Este modelo executa um controle local de atividades e inclui o conceito de programação orientada por dados. Este aspecto é importante quando se deseja, por exemplo, discriminar regiões de uma imagem e tratar cada uma destas regiões diferenciadamente. Suponhamos que a imagem a ser processada se encontra no plano P e que cada célula contém um plano S de $log_2(l)$ bits. Neste caso, podemos utilizar uma função de vizinhança $\delta_i(P \rightarrow S)$ para classificar a imagem original em l classes. Um programa do tipo

$$
\begin{array}{l}
(\delta_1(P \rightarrow P \text{ ; if } S = 0) \\
\delta_2(P \rightarrow P \text{ ; if } S = 1) \\
\quad \vdots \qquad\qquad \vdots \\
\delta_l(P \rightarrow P \text{ ; if } S = l\text{-}1))
\end{array}
$$

especifica um processamento a ser aplicado à imagem em função de um parâmetro ($\delta_i(P \rightarrow S)$) calculado a partir dos valores da imagem original e das células da vizinhança.

O modelo de autômato com funções condicionais, apresentado aqui, é uma generalização das máscaras presentes em arquiteturas celulares como DAP, MPP, CLIP7 e CAAP [Fountain-87] que contêm um registro binário indicando o estado ativo/inativo das células, no momento da execução de uma dada operação.

Um autômato com funções condicionais é mais potente que um autômato com planos de memória (se os dois autômatos têm mesma memória **Q** e conjunto de funções equivalentes).

2.4 Autômato com vizinhança dinâmica

O domínio da função de transição dos modelos definidos anteriormente é composto pelos $(n + 1)$ pontos correspondentes à célula e aos seus vizinhos. No entanto, diversos algoritmos de processamento de imagens necessitam de apenas um subconjunto destes vizinhos como domínio da operação. Além disto, esta parte da vizinhança não é necessariamente definida a priori, isto é, ela pode evoluir dinamicamente. Uma maneira de se abordar este tipo de algoritmo é guardar, a cada instante, o subconjunto das células que constituem o domínio de uma dada função. Estas células são ditas *ativas*.

Nós definimos um *autômato com vizinhança dinâmica* como sendo um autômato com planos de memória, contendo um plano especial **V**. Cada célula apresenta, neste plano, $(n+1)$ bits que indicam quais os pontos ativos da vizinhança. Uma função de vizinhança cujo domínio depende de **V** é representada por δ_i(; on V).

O plano de memória **V** é uma variável que pode ser inicializada estaticamente, através de uma função $\delta_i(V \rightarrow V; \mathbf{M})$ que afeta V com valor **M** (**M** pode ser um elemento estruturante em operações morfológicas, por exemplo), ou dinamicamente, a partir de uma função de transição do tipo $\delta_i(P \rightarrow V)$. Esta função define uma *conexão lógica* entre uma célula c e sua vizinhança. Dependendo da função de transição, o plano V pode fornecer diferentes informações relativas à vizinhança de c tais como direção do gradiente, posição dos k-vizinhos mais próximos, células com valor TRUE após uma operação lógica, etc. Este tipo de informação é importante na descrição de vários algoritmos de processamento de imagens, como nós veremos a seguir.

O autômato com vizinhança dinâmica representa um maior nível de autonomia local. Neste modelo, as células da rede são capazes de gerar a comunicação com seus vizinhos de maneira autônoma e dinâmica. De maneira geral, nós podemos utilisar V facilmente para representar uma configuração do autômato a partir de uma função de vizinhança δ_v que define uma conexão lógica entre as células da rede. Isto facilita, entre outras, a execução paralela de operações sobre diferentes regiões de uma imagem definidas de acordo com um certo critério de similaridade. Para ilustrar estes aspectos, consideremos o plano V representado pela estrutura bidimensional indicada na Fig. 1 (vizinhança 3x3):

Os valores de 1 a 8 indicam a posição das células vizinhas à célula 0. O plano de memória V tem um valor 1 nas posições relativas ao domínio de uma função δ_i. Suponhamos que uma imagem binária seja subdividida em um certo número de regiões e que se deseja extrair ou atribuir uma característica a cada uma destas regiões [Mérigot-91].

O critério de similaridade, considerado aqui, é definido em função das componentes conexas que

4	3	2
5	0	1
6	7	8

Figura 1: Estrutura de V numa vizinhança 3x3.

constituem as subregiões da imagem. Numa arquitetura estritamente SIMD este problema seria resolvido ou seqüencialmente, a nível de cada uma das regiões da imagem, ou em paralelo, através de uma comunicação complexa e custosa em termos de instrução e tempo de execução.

Se considerarmos os modelos de autômatos definidos anteriormente, a execução paralela das operações de concentração e distribuição de informações, nas diferentes regiões de uma imagem binária P, pode ser expressa através do seguinte algoritmo geral:

• *definição das componentes conexas: as células P=1 apontam para os vizinhos de mesmo estado.*

Esta instrução corresponde a uma função de vizinhança do tipo δ_v(P $\rightarrow$ V; if P = 1). Para um autômato 4-conectado, a aplicação desta função define uma configuração da arquitetura, como ilustrada na Fig. 2.

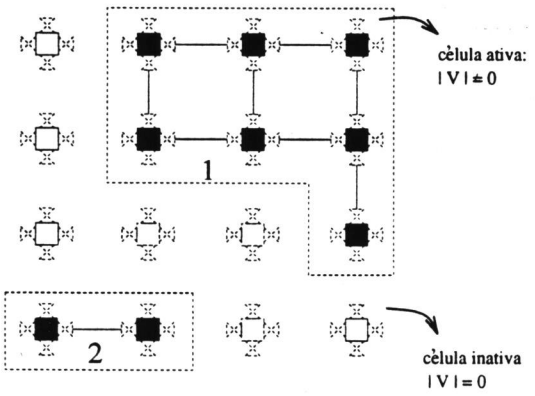

Figura 2: Exemplo de configuração lógica da rede.

• *execução de uma operação δ_p(; on V) de atribuição ou extração de características.*

δ_p é o programa do autômato que processa, em paralelo, todas as subregiões da imagem. Assim, se definimos o programa δ_p como sendo o operador absorvente *min*, aplicado a um plano de memória do autômato contendo uma concatenação x e y das células, no final de N^2 iterações desta função, sobre uma imagem NxN, todas as células de uma componente conexa terão um mesmo valor (o valor mínimo da concatenação de x e y), representando uma etiqueta distintiva da componente. Esta etiqueta pode ser usada posteriormente, no processamento de alto nível, para a identificação das diferentes subregiões da imagem [Weems-89].

Um autômato com vizinhança dinâmica é , assim, mais potente que um autômato com planos de memória (se os dois autômatos têm memória **Q** e conjunto de funções equivalentes).

2.5 Autômato celular supervisionado

Os autômatos definidos acima são limitados computacionalmente devido, sobretudo, à impossibilidade de extrairem, eficientemente, informações globais de uma imagem. Esta característica é importante, por exemplo, no controle da execução de operações paralelas a partir de testes globais efetuados sobre uma imagem. Estes testes são úteis na implementação de diversos algoritmos envolvendo, por exemplo, o cálculo de histograma, limiarização, max-min, fechamento transitivo de algoritmos iterativos, etc.

Nós definimos um *autômato celular supervisionado* como sendo um autômato geral ao qual associamos um *processador supervisor*. Este processador executa uma função de controle correspondente ao histograma dos estados do autômato, ou seja, ele conhece o número de células que, num dado instante, se encontram no estado q, $\forall\ q \in \mathbf{Q}$.

Este modelo exprime um paralelismo global em que um cálculo é efetuado por cada célula sem interação com as células vizinhas. Do ponto de vista teórico, este modelo é importante para a redução da complexidade computacional de algoritmos iterativos, executados numa matriz celular. Por exemplo, os algoritmos paralelos de afinamento ou de definição do número de componentes conexas de uma imagem cujos tempos de execução, sem supervisão global, são de $O(N^2)$ e $O(N)$, numa matriz de NxN células, terão uma complexidade de O(espessura máxima do objeto), no caso de um autômato supervisionado.

3 A representação algorítmica

Os modelos lógicos definidos acima podem ser associados naturalmente a uma estrutura de programação - linguagem - representando, em termos de um programa, o nível algorítmico de uma transformação. O conceito mais importante associado à estrutura de linguagem apresentada aqui refere-se aos *operadores maciçamente paralelos* (operadores que transformam milhares de dados em paralelo). Estes operadores

correspondem às funções de transição δ, específicas ao processamento de imagens. Como vimos anteriormente, uma seqüência destas funções constitui um programa do autômato.

A representação geral deste programa pode ser dada por:

<programa> := *<declarações>* **BEGIN**
<funções de transição> **END**

As *declarações* indicam o tipo dos planos de memória utilizados; as *funções de transição* constituem os operadores maciçamente paralelos agindo sobre os dados. A potência do conjunto destas funções reflete o grau de adaptabilidade da linguagem.

3.1 Exemplos de programação

Os exemplos que seguem dão uma idéia geral da estrutura da linguagem proposta aqui. Estes exemplos mostram como o nível de abstração da linguagem exprime o paralelismo de dados, e como a relação direta entre as instruções e os modelos formais, definindo a linguagem,contribui para uma tradução mais natural, em linhas de códigos de um programa, dos conceitos lógicos de uma aplicação.

3.1.1 Teste geral de convergência

Nós apresentamos, inicialmente, um algoritmo geral de teste de convergência que pode ser utilizado, por exemplo, como teste de parada (fechamento transitivo) em transformações iterativas do tipo relaxação [Rosenfeld-82]. Nestes algoritmos, as decisões tomadas em paralelo, numa dada iteração do programa, dependem do resultado da imagem na iteração precedente. Este teste de parada pode ser descrito da seguinte forma. Seja $\delta_i \in \Delta$ uma função de vizinhança a ser reiterada até que exista um número de células, superior a um limiar n_0, tal que a diferença entre o valor atual da célula e o seu valor, na etapa precedente, seja inferior a um limiar ϵ. Este controle pode ser executado por um autômato supervisionado com planos de memória. Neste caso, o programa do autômato é:

```
PROGRAM convergência
IMAGE P, Q: integer

BEGIN

  *REPEAT
/* tranfere conteúdo de P para Q */
     transf(P → Q)
/* executa função δi ∈ Δ */
     δi(P → P)
 /* calcula a diferença entre P e Q e guarda resultado em Q */
     diff((P, Q) → Q)
/* threshold de Q com ε */
     thresh ((Q,ε) → Q)
  *UNTIL HIST(Q=0) > n0 /*executa programa
              até que número de células com Q = 0
              seja maior que n0 */
END
```

A instrução ***REPEAT** ··· ***UNTIL HIST** é a função de controle associada ao autômato supervisor. Neste caso específico, **HIST** conhece o número de células com plano de memória Q=0. A execução da função δ é interrompida quando este número é maior que n_0.

3.1.2 Filtro seletivo de Nagao

O segundo programa ilustra a flexibilidade resultante da noção de vizinhança dinâmica. Trata-se de uma versão do filtro seletivo de Nagao [Nagao-79] que opera sobre uma vizinhança 5x5 da imagem subdividida em 9 janelas 3x3. Para cada uma destas janelas calcula-se um índice de homogeneidade (variância), e um valor representativo da janela (a média, por exemplo). O pixel central da vizinhança 5x5 é substituído pelo valor representativo associado à janela de índice mais homogêneo. Este algoritmo realiza uma filtragem da imagem com preservação de contornos. A representação do filtro de Nagao em pseudo-linguagem é:

```
BEGIN
     P ← imagem original
     Q ← variância
/* define a direção do vizinho de menor variância
*/
     min ← Q
     dir ← vizinho(0)
     para i=1 a 8 fazer
        se(Q[vizinho(i)] < min) então
           min ← Q[vizinho(i)]
/* guarda a direção do vizinho com menor /*
índice de homogeneidade */
           dir ← i
        fim_se
     fim_para
     Q ← média
/* lê valor na respectiva direção */
     P = Q[dir]
END
```

De um modo geral, podemos dizer que é difícil expressar este algoritmo de maneira simples e concisa, num autômato cujas células não comportam funções de endereçamento indireto. A especificação destas funções é feita seqüencialmente em linguagens

como IPC, GAL, *C, DAP-FORTRAN, etc [Fountain-87] que são linguagens seqüenciais extendidas a aplicações de processamento de imagens em arquiteturas do tipo autômato celular.

O conjunto de primitivas abaixo descreve o algoritmo de Nagao, na nossa estrutura de linguagem.

```
PROGRAM Nagao
IMAGE P,Q: integer
          V: pointer
BEGIN

/* calcula variância da vizinhança de P e armazena
resultado em Q. */
   variance(P → Q)
/* aponta para célula da vizinhança com valor
mínimo em Q */
   vmin(Q → V)
/* calcula a média da vizinhança de P. */
   conv(P → Q ; 1/9, 1/9, 1/9,
                 1/9, 1/9, 1/9,
                 1/9, 1/9, 1/9)
/* substitui imagem original pelo valor apontado
por V.*/
   transf(Q → P ; on V)

END
```

O plano de memória V, declarado como *apontador* (pointer), é empregado aqui para memorizar a direção das células cujo estado, no plano Q, é mínimo. Em seguida, a função *transf* utiliza esta informação para afetar, através de um enderaçamento indireto, o plano de memória P.

O paralelismo explícito das funções primitivas contribui para um maior nível de abstração na implementação deste algoritmo. O comando: "**procurar o máximo dos vizinhos no plano de memória Q e memorizar sua posição relativa**" é traduzida simplesmente pela função *min(Q → V)*. A representação desta função, próxima da descrição funcional do comando, justifica aqui a ausência de um raciocínio seqüêncial do tipo "**procurar vizinho na direção 1 e guardar direção se mínimo; procurar vizinho na direção 2 e guardar direção se mínimo;...**".

3.1.3 Reconexão de contornos

Este exemplo ilustra a flexibilidade de utilização do autômato com vizinhança dinâmica na implementação de operações de processamento de baixo nível de imagens que exigem uma certa autonomia funcional das células.

O exemplo aborda o problema da reconexão de contornos de uma imagem cujos objetos são diferenciados do fundo por uma técnica de segmentação qualquer. Se considerarmos que existe um contorno fechado definindo estes objetos, então uma operação de deteção de contornos, aplicada à imagem, deve extrair os pontos de borda das suas regiões homogêneas. Na prática, estes algoritmos não detectam corretamente os contornos (devido a ruídos, por exemplo), gerando resultados como o da Fig. 3a.

A aplicação de métodos clássicos de reconexão de contornos [Ballard-82], a partir de uma arquitetura paralela do tipo autômato celular, conduz a resultados pouco satisfatórios devido à limitação do nível de autonomia das células. Isto faz com que segmentos de reta sejam introduzidos na imagem [Fountain-87] (ver Fig. 3b). Um algoritmo melhor adaptado a este tipo de problema deve considerar a curvatura dos contornos dos objetos. O autômato deve, por exemplo, fazer uma comparação entre as direções do gradiente das células representando os contornos dos objetos, já definidos, e das células vizinhas susceptíveis de pertencerem aos contornos.

Um algoritmo deste tipo [Rosenfeld-82] transforma iterativamente o estado dos pixels vizinhos que se encontram na mesma direção dos pontos do contorno, isto é, mais ou menos perpendiculares à direção do gradiente. Neste caso, ao encontrarmos um ponto *candidato ao contorno*, numa destas direções (ponto com gradiente superior a um limiar t_s, por exemplo), cuja inclinação do gradiente não difere muito da inclinação do ponto pertencente ao contorno (nós consideramos uma variação de $\pm 45°$ para uma vizinhança de 3x3), então este ponto é conectado ao contorno, se não, ele é desconsiderado. A aplicação deste algoritmo deve gerar um resultado semelhante ao ilustrado na Fig. 3c.

Para a descrição do programa, nós consideramos que o fundo da imagem tem valor *0* após as etapas iniciais de segmentação e deteção dos contornos dos objetos, e que os pixels reconhecidos como pertencentes aos contornos têm valor c. Os *candidatos ao contorno* têm valor t definido por um limiar t_s, $t_s < c$, eventualmente. Assim, para cada pixel P da imagem, as células do autômato executam as seguintes instruções:

- lê os valores de P dos dois vizinhos, na direção perpendicular ao seu gradiente $\vec{g}$.
- verifica se um destes pixels faz parte do contorno (P = c) e se sua direção $\vec{g}$ é igual à direção $\vec{g}$ de P.
  - se sim: então o pixel com P=t se torna P=c.
  - se não: repetir as operações acima conside-

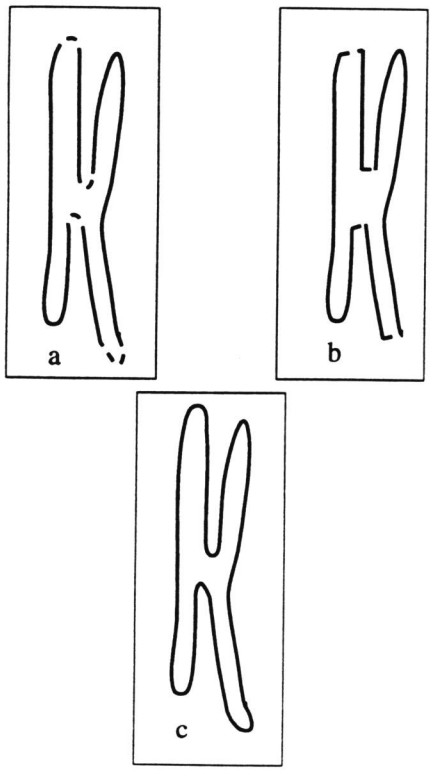

Figura 3: Exemplo de reconexão de contorno.

rando as direções $\vec{g}$ incrementadas de uma unidade ($\pm 45°$), em relação ao seu valor inicial.

Nós definimos aqui uma operação sobre a estrutura bidimensional V que consiste de uma rotação relativa, do seu conteúdo, de $\pm x$ posições. Esta operação é indicada por $V^{\pm x}$. Assim, por exemplo, a instrução $\delta_i(\ ;\ V^{+1})$ especifica o domínio de δ_i a partir de uma rotação de uma posição, no sentido horário, do plano de memória V. Considerando uma função $\uparrow$*gradient* que determina a direção de $\vec{g}$, na vizinhança 3x3, e que guarda esta informação em V, a parte principal do programa de reconexão de contornos é:

```
/* define direção do gradiente e guarda em V.*/
     ↑gradient(P → V)
/* lê P do vizinho numa das direções ⊥ a g. */
     transf(P → B; sur V+2)
/* compara g da célula com g do vizinho. */
     comp(V → S; sur V+2)
/* P=t se torna P=c se o vizinho é um contorno
e as direções de g são as mesmas
(os pixels P=c e P=0 não são modificados). */
     transf(P → B; si S=0 OU P=0)
     assign(P → P; c, si S=1 ET B=c)
```

Estas instruções são executadas iterativamente por todos os pontos vizinhos, perpendiculares às direções de $\vec{g}$. O algoritmo pára quando nenhuma alteração é verificada na imagem. Esta condição de estabilização pode ser detectada pelo autômato supervisor que avalia a evolução dos estados da imagem através de um controle semelhante àquele da seção 3.1.1.

4 Conclusão

Nós apresentamos uma estrutura de linguagem definida a partir de alguns modelos formais de autômatos celulares iterativos com aplicações voltadas para processamento de imagens.

O "kernel" da linguagem é representado por um conjunto de funções primitivas aplicadas sucessivamente às células da rede. Estas primitivas podem ser consideradas como operadores cujo tempo de execução obedece ao critério de custo uniforme (cada primitiva é executada, a grosso modo, numa unidade de tempo específica), o que facilita a análise do tempo de execução de um programa.

As funções de transição, definidas como "kernel" da linguagem, resultam em considerações importantes tais como:

- a simplicidade e implementação eficaz das funções primitivas facilitam sua implementação em outras máquinas e permitem a um compilador gerar códigos otimizados sem que sejam necessárias técnicas complexas de otimização.
- a flexibilidade referente à introdução ou definição de outras primitivas contribui para uma extensão fácil da linguagem. Esta característica é importante quando se considera as necessidades dos usuários que trabalham nas diferentes áreas de processamento de imagens.
- a sintaxe das funções primitivas, próxima da área de aplicação, e o pequeno número de conceitos envolvidos facilitam a implementação e a reutilização de programas visto que as linhas de comando expressam mais visivelmente os conceitos lógicos relacionados.
- a estrutura da linguagem é bem adaptada a um ambiente de programação visual [Olson-91]. As funções de transição podem ser submetidas interativamente a um interpretador de comandos, seguidas de uma avaliação, execução e visualização imediatas das transformações efetuadas sobre imagens.

Finalmente, a definição da estrutura da linguagem a partir de modelos lógicos e seu nível de abstração com relação às características da arquitetura

possibilitam um grande nível de transportabilidade da linguagem. Estudos atuais referem-se à extensão destes modelos a outros tipos de arquitetura maciçamente paralela para processamento de imagens tais como as arquiteturas piramidais.

Agradecimento: O autor é muito grato à colaboração do Prof. Gilles Bertrand da École Supérieure d'Ingénieurs en Électrotechnique et Électronique de Paris pela ajuda na formalização dos conceitos presentes neste trabalho.

Esta pesquisa foi desenvolvida com suporte financeiro da CAPES.

5 Referências

D. H. Ballard, C. M. Brown, Computer Vision, *Prentice-Hall*, New Jersey, (1982).

T. J. Fountain. Processor Arrays - Architecture and Applications, *Academic Press*, London, (1987).

M. J. E. Golay, Hexagonal Parallel Pattern Transformations, *IEEE Trans. on Computers*, **18**, n.8, (1969) 733-740.

N. J. Leite, G. Bertrand, A Parallel Image Processing Language Based on Computational Models, *Proceedings of the 11th IAPR Inter. Conf. on Pattern Recognition*, The Hague, The Netherlands, (1992), 181-184.

M. Nagao, T. Matsuyama, Edge Preserving Smoothing, *Computer Graphics and Image Processing*, **9** (1979) 394-407.

T. J. Olson, N. G. Klop, MAVIS: A Visual Shell for Computer Vision and Image Processing, *Proceedings of Computer Arch. for Machine Perception*, (Paris 1991)

A. Rosenfeld, Picture Languages - Formal Models for Picture Processing, *Academic Press*, New York, (1979).

A. Rosenfeld, A. C. Kak, *Digital Picture Processing, vols. I e II*, Academic Press, Orlando, (1982),

A. Rosenfeld, Parallel Image Processing Using Cellular Arrays, *Computer*, January, (1983) 14-20.

C. C. Weems et alli, The Image Understanding Architecture, *International Journal of Computer Vision*, **2**, (1989) 251-282.