Representação e Descrição de Imagens

Cleber M. Barboza N.USP 3286353
Fernando Mario de O. Filho N.USP 3286395
João Vitor B. Soares N.USP 3286461

Professora Nina S. T. Hirata

12 de julho de 2002

1 Introdução

Representação e descrição de imagens é a área de processamento de imagens que estuda formas de representar elementos de uma imagem de maneira conveniente para posterior uso ou tratamento.

Existem basicamente duas formas de representação de imagens, segundo [2]:

- A *representação externa*, que consiste em representar uma região em termos de suas características externas, ou seja, sua fronteira.
- A *representação interna*, que consiste em representar uma região segundo suas características internas, como os *pixels* que a formam.

Em nosso trabalho pretendemos estudar formas de representação externa dos objetos de uma figura, mais especificamente, aproximações da fronteira de tais objetos por polígonos. Tal assunto é muito relevante no contexto da extração de informação de uma imagem, já que desta forma podemos, após ter codificado a fronteira dos diversos objetos como polígonos, aplicar diversos outros algoritmos interessantes sobre os polígonos resultantes.

Por exemplo, após termos aproximado o contorno de um objeto por um polígono, podemos resolver de forma eficiente o problema de encontrar o fecho convexo de tal objeto, ou o de determinar se um ponto é ou não interior ao objeto. Para saber mais sobre estas e outras aplicações, consulte O'Rourke [5, 6].

1.1 Tópicos abordados

Para realizar nosso objetivo teremos de estudar os seguintes tópicos:

1. Segmentação de imagens em regiões.
2. Algoritmos para encontrar a fronteira digital dos objetos de uma imagem.
3. Algoritmos para descrição de fronteiras por polígonos.

2 Segmentação

Para obter a descrição das fronteiras de objetos da imagem por polígonos, queremos inicialmente obter uma segmentação da imagem e, a partir dela, criar os polígonos pela aproximação das fronteiras entre os segmentos obtidos.

Segmentações identificam as áreas de uma imagem que aparecem uniformemente ao observador, e subdivide a imagem, explicitando essas áreas. Entre as abordagens comuns para segmentação, estão:

- Segmentação por tresholding: A imagem é segmentada com base em tresholds, normalmente escolhidos a partir do histograma da imagem. Cada regiao R da segmentação corresponde aos pixels cujos valores estão entre dois valores T_1 e T_2, isto é:

$$R = \{P : T_1 \leq f(P) \leq T_2\}$$

  Onde f é a função que descreve a imagem.

- Segmentação baseada em contornos: As segmentações baseadas em contornos tentam encontrar as regiões de rápida mudança de valores que definam fronteiras entre regiões. A abordagem usada é a de aplicar operadores de gradientes à imagem e observar sua magnitude. Locais com alta magnitude do gradiente podem corresponder a transições entre regiões.Assim, a partir do gradiente da imagem, a fronteira entre regiões pode ser encontrada.

- Segmentação baseada em regiões.

Nas próximas seções, abordaremos com mais profundidade dois tipos de segmentação baseadas em regiões: watershed (divisor de águas) e region growing (crescimento de regiões).

2.1 Aplicações

A segmentação de imagens pode ser muito útil nas mais diversas áreas: médica, militar, exploração espacial, indústria cinematográfica etc.

2.2 Watershed

O conceito de watershed e catchment basins (bacia hidrográfica) é bem conhecido em topografia. As linhas de divisão de águas são as fronteiras entre bacias hidrográficas de uma região.

Dessa maneira, podemos interpretar o gradiente da imagem como uma superfície topográfica, em que os níveis de cinza representam as altitudes.

As regiões de contorno da imagem (de alto gradiente) correspondem às linhas de divisão de águas, enquanto que as regiões interiores (de baixo gradiente), às bacias hidrográficas. As bacias hidrográficas da superfície topográfica são homogêneas no sentido de que todos os pixels que pertencem a uma mesma bacia são conectados com as regiões de mínima altitude por um caminho simples de pixels em que a altitude é monotonicamente decrescente. Assim, tais bacias representam as regiões de segmentação da imagem.

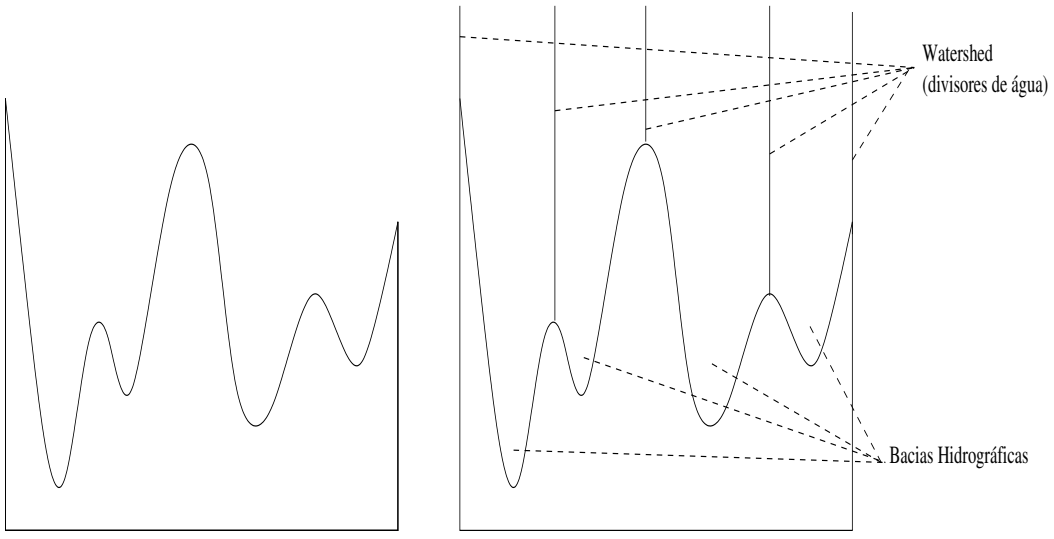

Figura 1: Segmentação watershed em uma imagem de dimensão 1

Existem várias maneiras de se realizar a segmentação watershed. Mas todas utilizam a mesma idéia básica. Pensando numa imagem como uma superfície topográfica, podemos inundá-la a partir das regiões interiores. Dessa maneira, iremos particionar a imagem em diferentes conjuntos: **as bacias hidrográficas e as linhas divisores de águas.**

Se aplicarmos essa transformação no gradiente da imagem, as bacias serão correspondentes às regiões de níveis de cinza homogêneas.

Entretanto, na prática se realizarmos esse tipo de transformação, iremos obter uma super-segmentação, devido aos ruídos ou irreugularidades locais no gradiente da imagem. Veja figura 2.

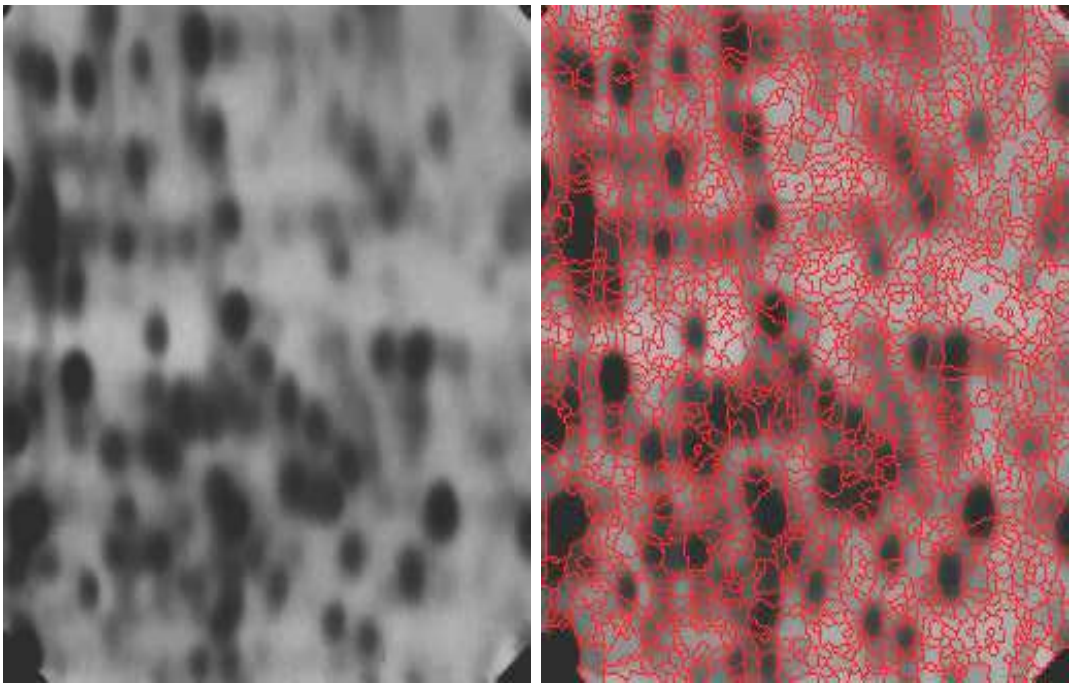

Figura 2: Imagem e a sua segmentação watershed

A maneira mais eficiente de se evitar a super-segmentação é, antes de aplicar a transformação watershed, escolher um conjunto de **marcas**, definido automaticamente ou não, para o gradiente da imagem. Veja figura 3.

Essas **marcas** são regiões conexas, cuja finalidade é indicar qual objeto deverá ser segmentado. Na figura à esquerda, foram escolhidas algumas **marcas**[1] iniciais através de algum critério. A partir delas, obtivemos a imagem segmentada à direita.

Portando, realizar a segmentação watershed consiste basicamente de dois passos:

- **Passo 1**

  Encontrar as **marcas** e **um critério de segmentação** (o critério ou a função que será usada para dividir as regiões - na maior parte das vezes são utilizados ou o contraste ou o gradiente, mas não necessáriamente).

- **Passo 2**

  Realizar a tranformação watershed com base nos dois elementos do passo 2

2.3 Algoritmo Watershed

A seguir, apresentaremos um algoritmo, que foi deselvolvido por Luc Vicent e Pierre Soille, para realizar a segmentação watershed. A idéia básica do algoritmo é a seguinte[2]: no ponto mais baixo de cada bacia

[1] "manchas" delimitadass pelas linhas brancas

[2] Devemos pensar na imagem como uma superfície topográfica

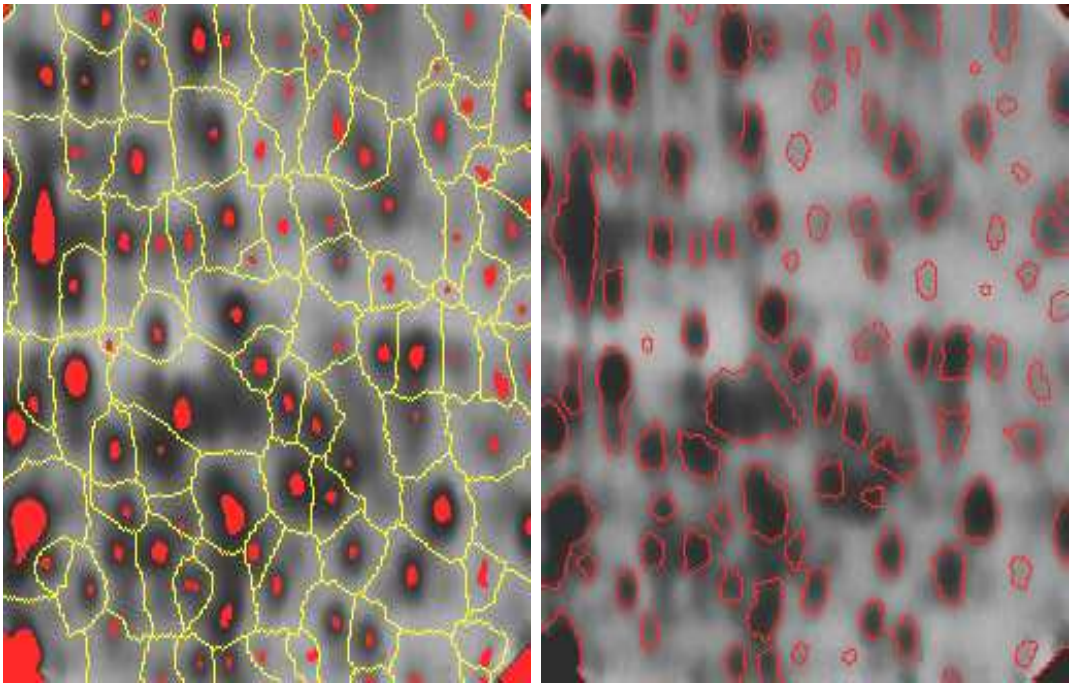

Figura 3: Imagem e a sua segmentação watershed utilizando marcas

hidrográfica, imaginasse que há um furo. Em seguida, simula-se a formação das poças de água através da imersão dessa superfície um tanque d'água de modo que a superfície seja inundada através daqueles furos. Conforme as bacias hidrográficas vão enchendo e se juntando, marca-se os pontos de encontro de uma bacia com a outra. O conjunto desses pontos marcados formam as linhas de partição das águas procuradas.

O algoritmo usa estruturas FIFO em uma de suas partes, e uma técnica de ordenação por acesso direto de endereços ("sorting by address calculation") para melhorar sua eficiência.

Por praticidade, dividimos o algoritmos e duas partes: ordenaçãp e imersão.

2.3.1 Ordenação

Nessa parte, os pontos da imagem são ordenados de acordo com seus valores de níveis de cinza. O resultado é guardado em um vetor que contém os endereços dos pontos da imagem.

A ordenação é constituída de duas etapas:

- Cálculo do histograma dos níveis de cinza da imagem.
- Com o histograma, pode-se criar um vetor ordenado de pontos, onde cada posição do vetor contém o endereço de um ponto da imagem.

ORDENAÇÃO(IMAGEM:f)

```
1  Seja H um vetor de k posições (número de níveis de cinza)
2  Seja hmin um inteiro que contém o menor nível de cinza da imagem
3  Seja hmax um inteiro que contém o maior nível de cinza da imagem
4  H[ ] ← 0 /* Inicializando o vetor H */
5  hmin ← hmax ← f(0) /* Inicializando hmin e hmax */
6  para cada ponto p da imagem faça
7     H[f(p)] + + /* Processa o histograma */
8     se f(p) < hmin então
9        hmin ← f(p)
10    se f(p) > hmax então
11       hmax ← f(p)
```

```
12 Seja HC um vetor com k posições para armazenar o histrograma cumulativo da imagem
13 Seja ims um vetor com k posições
14 HC[ ]← 0
15 para i ← (hmin + 1) até (hmax + 1) faça
16    HC[i] ← HC[i − 1] + H[i − 1] /* Histograma cumulativo */
17 para  cada ponto p da imagem faça
18    ims[HC[f(p)]] ← p /* Coloca o endereço de p em ims
19    HC[f(p)] + +
20 devolva ims, hmin, hmax, HC
```

2.3.2 Imersão

Nessa parte, temos o processo que simula a imersão da superfície topográfica (gradiente da imagem) em um tanque de d'água. Os pontos localizados em uma mesma bacia hidrográfica recebem um único **rótulo**[3]. Esses rótulos permitirão identificar as diferentes bacias. Já aqueles pontos que se localizam no encontro das diferentes bacias reberão um rótulo especial: **wshed**.

Assim, todos os pontos inundados até uma altura h (nível de cinza) estarão rotulados. Como teremos um vetor ordenado segundo os níveis de cinza (veja 2.3.1), poderemos acessar todos os pontos de altura $h+1$ diretamente. Inicialmente, os pontos com altura $h+1$ receberão um rótulo temporário: **mask**, e aqueles que têm pelo menos um vizinho já rotulado receberão um novo rótulo: **inqueue** e serão colocados em uma fila.

O algoritmo usa aquela fila para fazer uma busca na imagem e rotular todos os pontos com altura $h+1$. A rotulação é feita processando-se as **zonas de influência geodésica**[4] das bacias já rotuladas e que possuem pontos com o valor **mask**. Com isso, as bacias vão crescendo, e os pontos com rótulos **mask** são assimilados, recebendo o rótulo das bacias que os assililaram.

IMERSÃO(IMAGEM:f)

```
 1 Seja g a imagem de saida /* |g| = |f| */
 2 Seja mask ← −2
 3 Seja wshed ← 0
 4 Seja init ← −1 /* Valor dos pontos iniciais de g */
 5 Seja inqueue ← −3 /* Valor para os pontos enfileirados */
 6 para  cada ponto p da imagem g  faça
 7    g(p) ← ini /* Inicializando g */
 8    dist(p) ← 0 /* Inicializando distância geodésica g */
 9 Seja rótulo-atual ← −2
10 (ims, hmin, hmax, HC) ← ORDENAÇÃO(f)
11 para h ← hmin até hmax faça
12    /* SKIZ geodésico do nível h − 1 dentro do nível h */
13    se h ≠ 0  então
14       start ← HC[h − 1]
15    senão
16       start ← 0
17    para i ← start até (HC[h] − 1) faça
18       p ← ims[i]
19       g(p) ← mask
20       para  cada ponto r vizinho de p  faça
21          se g(r) > 0 ou g(r) = wshed  então
22             QUEUE-ADD(W,r)
23             dist(p) ← 1
```

[3]Seja f uma imagem. A imagem rotulada g é dada por: $g(p) = g(q) \iff p$ e q fazem parte de um mesmo objeto (no nosso caso, mesma bacia hidrográfica).

[4]Sejam A uma imagem, e $B_1, B_2, ..., B_k$ suas componentes conexas (regiões que fazem parte de um mesmo objeto segundo algum critério). A **zona de influência geodésica** B_i corresponde a todos os pontos de A ques estão mais próximos de B_i do que qualquer outra componente B_j. Os pontos de A que não pertencem a nenhuma zona de influência geodésica pertencem ao "esqueleto" das zonas de influência ("Skeleton by Influence Zones") ou **SKIZ**.

```
24            g(p) ← inqueue
25      distância-atual ← 1
26      QUEUE-ADD(W,ponto-ficticio)
27      enquanto TRUE faça
28        p ← QUEUE-FIRST(W)
29        se p = ponto-ficticio então
30          se QUEUE-EMPTY(W) então
31            BREAK
32          senão
33            QUEUE-ADD(W,ponto-ficticio)
34            distância-atual++
35            p ← QUEUE-FIRST(W);
36        para cada ponto r vizinho de p faça
37          se (dist(r) < distância-atual) e (g(r) > 0 ou g(r) = wshed) então
38            se g(p) > 0 então
39              se g(p) = mask ou g(p) = wshed então
40                QUEUE-ADD(W,r)
41                g(p) ← g(r)
42              senão
43                se g(p) ≠ g(r) então
44                  g(p) ← wshed
45          senão
46            se g(p) = mask então
47              g(p) ← wshed
48          senão
49            se g(r) = mask e dist(r) = 0 então
50              dist(r) ← distância-atual
51              QUEUE-ADD(W,r)
52      para j ← start até (HC[i] − 1) faça
53        g(p) ← ims[j]
54        se g(p) = mask então
55          g(p) ← ++rótulo-atual
56          QUEUE-ADD(W,p)
57          enquanto QUEUE-EMPTY(W) = FALSE faça
58            r ← QUEUE-FIRST(W)
59            para cada ponto q vizinho de r faça
60              se g(q) = mask então
61                QUEUE-ADD(W,q)
62                g(q) ← rótulo-atual
63  devolva g
```

2.4 Crescimento de Regiões

A segmentação por crescimento de regiões baseia-se em achar, na imagem, grupos de pixels que obedeçam a certos critérios de uniformidade.

Existem vários algoritmos que usam crescimento de regiões para segmentar uma imagem. A maioria deles começa com pequenas regiões que devem crescer – agregando a si alguns pixels (normalmente adjacentes à região) que obedeçam a algum critério de similaridade. As regiões iniciais, a partir das quais o crescimento é feito, costumam ser chamadas de sementes e são determinantes no resultado da segmentação.

Um método simples que é capaz de particionar uma imagem em regiões consiste em varrer a imagem em alguma ordem (por exemplo, a *raster*, em que as linhas são percorridas de cima para baixo e cada linha é percorrida da esquerda para a direita) e, para cada pixel encontrado que não pertencer à alguma região, deve-se criar uma nova região usando-o como semente. Para criar a região a partir do pixel, considera-se inicialmente a região constituída apenas pelo pixel. Então, devem ser sucessivamente agregados à região pixels adjacentes que obedecerem ao seguinte critério: a diferença absoluta entre o pixel que está fora da

região e um pixel que está na região e é adjacente a ele é pequena. Pixels adjacentes devem ser agregados até que nenhum pixel obedeça ao critério.

No método anteriormente descrito, no lugar de usar pixels como sementes, é preferível escolher pequenas regiões, evitando assim a escolha de um pixel inicial que faça parte do ruído da imagem, o que pioraria o resultado da segmentação.

A escolha das sementes é normalmente muito importante para o resultado da segmentação, portanto resultados melhores podem ser obtidos com um pré-processamento da imagem que faça uma escolha adequada de sementes. Também é comum que o usuário escolha as sementes, porém dessa forma a segmentação não fica totalmente automatizada. Após feita alguma escolha prévia de sementes, pode ser feito o crescimento simultâneo das regiões, ou seja, a cada passo da construção da segmentação, deve-se decidir qual região deverá crescer. Dependendo do resultado que desejarmos, poderemos usar as seguintes idéias para a escolher a região que deverá crescer antes:

- escolher a região de menor área
- escolher a região de menor raio.
- escolher a região que tenha pixels adjacentes de valores mais próximos à média dos valores dos pixels da região

Para a criação das sementes, podemos usar uma segmentação inicial mais simples, como o método baseado em tresholding, descrito anteriormente.

Outro método de segmentação que usa regiões começa com uma partição da imagem em conjuntos de pequenas regiões homogêneas. Após isso, regiões vizinhas sao sucessivamente unidas enquanto forem suficientemente uniformes. Esse último método tem a vantagem de ser menos sensível a ruídos. Uma partição inicial poderia ser simplesmente a divisão da imagem em quadrados de tamanho 2×2 ou 4×4 pixels.

Com esse método, para decidir se duas regiões vizinhas devem ser unidas, podemos:

- extrair informações estatísticas de cada uma e ver se são parecidas
- medir a "significação das bordas" entre as regiões: faz-se um treshold do gradiente da imagem e calcula-se a fração da região de fronteira entre duas regiões que teve valor 1 após o treshold. Se essa fração for suficientemente pequena, as regiões adjacentes podem unidas, pois a fronteira entre elas é de pouco contraste.
- considerar a união das regiões e fazer um teste de homogeneidade. Se a união for homogênea, as regiões são unidas.

Outra abordagem é considerar inicialmente a imagem toda como uma única região (que não precisa ser homogênea) e subdividi-la repetidamente de tal forma que sejam criadas regiões homogêneas. Existem ainda técnicas que utilizam tanto a subdivisão quanto a agregação de regiões para fazer a segmentação.

Entre os critérios para decidir se uma região é homogênea, está o critério da diferença máxima entre o valor de um pixel e a média dos valores dos pixels de uma região, isto é, uma região R seria homogênea se:

$$max_{P \in R} |f(P) - m| < T$$

Onde m é a média dos pontos da região e T é algum limite estabelecido.

2.4.1 Algoritmo Exemplo

A seguir, será apresentado um exemplo de algoritmo que usa agrupamento de regiões:

1. O algoritmo começa com uma partição da imagem em regiões com pixels de mesmo valor que sejam contínuas e de maior tamanho possível.

   A partir dessa partição, deve ocorrer a união recursiva de regiões que tiverem fronteiras divisoras suficientemente fracas.

2. Regiões adjacentes R_i e R_j serão unidas se:

$$\frac{W}{min(l_i, l_j)} \geq T_1$$

Onde l_i, l_j são os comprimentos dos perímetros de R_i e Rj; T_1 é algum treshold preestabelecido e W é o comprimento das fronteiras entre as regiões em que a diferença absoluta entre os valores de R_i e R_j é pequeno, que correspondem às partes fracas da fronteira.

3. Após isso, regiões adjacentes R_i e R_j devem ser unidas quando:

$$\frac{W}{l} \geq T_2$$

Onde l é o comprimento do perímetro da fronteira entre R_i e Rj; T_2 é algum treshold preestabelecido e W é o comprimento da parte fraca da fronteira entre as regiões, como anteriormente.

3 Descrição de fronteiras por polígonos

Nesta seção trataremos do problema de descrever a fronteira de objetos em uma imagem por polígonos. Apresentaremos basicamente dois métodos para a solução deste problema, cada um deles possui qualidades diferentes que podemos explorar.

Antes de começarmos nossa discussão, vamos falar um pouco sobre as hipóteses que fazemos. Supomos que a imagem original, provavelmente em níveis de cinza, já tenha sido segmentada e que cada objeto seja dado por um conjunto de pontos que descrevem sua fronteira. Em outras palavras, supomos que temos uma imagem binária que contém apenas um objeto, cujo contorno se encontra em preto e tanto o interior quanto o exterior se encontram em branco. É fácil ver que podemos representar tal imagem por uma matriz ou mais compactamente apenas através de uma lista dos pontos que a compõe.

3.1 Descrição através de polígonos convexos

Em muitos casos práticos é útil descrever um objeto em uma imagem por um polígono convexo, de preferência o menor[5] polígono convexo que o contém. Quando temos a imagem binária, resultado da segmentação da imagem original, isso é equivalente a encontrar o menor polígono convexo que contém todos os pontos (pretos) da imagem binária. Este método de descrição é especialmente interessante quando temos descontinuidades no contorno dos objetos: podemos garantir que nenhuma parte do objeto será perdida.

Vamos então tratar do problema de, dado um conjunto $S \subseteq \mathbf{R}^2$, encontrar o menor polígono convexo que contém todos os pontos de S. Tal polígono é chamado de fecho convexo dos pontos do conjunto S. Mais formalmente, dado um conjunto $S = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbf{R}^2$, o *fecho convexo* dos pontos de S, denotado por $\text{conv}(S)$, é o conjunto dos pontos $x \in \mathbf{R}^2$ para os quais existem reais não-negativos $\lambda_1, \lambda_2, \ldots, \lambda_n$ tais que $\lambda_1 + \lambda_2 + \cdots + \lambda_n = 1$ e

$$x = \lambda_1 p_1 + \lambda_2 p_2 + \cdots + \lambda_n p_n.$$

Podemos provar facilmente que o fecho convexo de um conjunto S de pontos é o menor polígono convexo que contém os pontos de S. É fácil ver que quando percorremos os vértices do fecho convexo de um conjunto de pontos no sentido anti-horário, viramos à esquerda a cada ponto. Esta é a idéia básica por trás do funcionamento do algoritmo de Graham, que calcula o fecho convexo de um conjunto de n pontos em tempo $O(n \log n)$. Vamos expor este algoritmo mais adiante.

Podemos determinar se dois segmentos consecutivos, $\overline{p_0 p_1}$ e $\overline{p_1 p_2}$, viram à esquerda ou à direita fazendo uso de produtos vetoriais. Dados dois vetores $u = (x_1, y_1)$ e $v = (x_2, y_2)$, o produto vetorial de u e v, denotado por $u \times v$, é dado por

$$u \times v = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1.$$

O sinal do produto vetorial entre u e v pode ser utilizado para determinar se u está no sentido horário ou anti-horário a partir de v. Quando $u \times v$ é positivo então u está no sentido horário a partir de v. Se $u \times v$ é

[5] No sentido de inclusão

negativo, então u está no sentido anti-horário a partir de v. Quando o produto é zero, ambos os vetores tem a mesma direção. Com isso, os segmentos $\overline{p_0p_1}$ e $\overline{p_1p_2}$ viram à esquerda em p_1 se e somente se $p_0p_2 \times p_0p_1 < 0$. Se $p_0p_2 \times p_0p_1 > 0$, então os segmentos viram à direita em p_1.

Com isso, temos o seguinte algoritmo para determinar o fecho convexo dos pontos de um conjunto S. O algoritmo recebe o conjunto S e devolve uma lista com os vértices do fecho convexo no sentido anti-horário. Utilizamos no algoritmo uma pilha P. Denotamos por $topo[P]$ o elemento inserido mais recentemente na pilha. Denotamos por $pred[P]$ o predecessor de $topo[P]$. A lista devolvida é a pilha, que contém do fim para o topo os vértices do fecho convexo no sentido anti-horário.

FC-GRAHAM(S)
1 Seja p_0 o ponto de S com a menor coordenada x dentre os pontos que possuem a menor coordenada y
2 Seja $\langle p_0, p_1, \ldots, p_m \rangle$ os demais pontos de S, ordenados segundo seus ângulos polares no sentido anti-horário ao redor de p_0 (se mais de um ponto tem as mesmas coordenadas polares, despreze todos exceto o mais distante de p_0)
3 $P \leftarrow$ CRIA-PILHA()
4 EMPILHA(p_0, P)
5 EMPILHA(p_1, P)
6 EMPILHA(p_2, P)
7 **para** $i \leftarrow 3$ até m **faça**
8 **enquanto** os o ângulo formado pelos pontos $pred[P]$, $topo[P]$ e p_i não virar à esquerda **faça**
9 DESEMPILHA(P)
10 EMPILHA(p_i, P)
11 **devolva** P

Vale lembrar que o ângulo polar de um ponto p_i relativo à p_0 é o ângulo entre o vetor p_0p_i e o eixo x. Como os ângulos polares aumentam no sentido anti-horário, podemos fazer a ordenação da linha 2 utilizando produtos vetoriais para comparar os pontos. Note que o ponto p_0, escolhido na linha 1, é certamente um vértice do fecho convexo dos pontos de S.

As linhas 3-6 inicializam a pilha de modo a conter, do fim para o topo, os primeiros três pontos p_0, p_1 e p_2. O laço das linhas 7-10 executa uma iteração para cada ponto da subseqüência $\langle p_3, p_4, \ldots, p_m \rangle$. Nossa intenção é que após termos processado o ponto p_i, a pilha contenha, do fim para o topo, os vértices de $\text{conv}(\{p_0, p_1, \ldots, p_i\})$ no sentido anti-horário. O laço das linhas 8-9 remove vértices da pilha se decidimos que eles não fazem parte do casco convexo. Como dissemos anteriormente, quando percorremos os vértices do casco convexo no sentido anti-horário, viramos à esquerda em cada ponto. Se não viramos à esquerda nós então desempilhamos o topo da pilha. Uma demonstração formal de que o algoritmo está correto pode ser encontrada em [1].

Vamos agora analisar a eficiência de nosso algoritmo. A linha 1 pode ser executada em tempo $O(n)$. A linha 2 pode ser executada em tempo $O(n \log n)$ usando o MergeSort. As linhas 3 a 6 podem ser executadas em tempo $O(1)$. Para analisarmos o tempo gasto pelo laço das linhas 7-10, vejamos quantas iterações do laço das linhas 8-9 são executadas no total. Note que cada iteração deste laço desempilha um elemento da pilha. Como cada elemento é empilhado no máximo uma vez, cada elemento pode ser desempilhado no máximo uma vez. Portanto, no total o laço das linhas 8-9 toma tempo $O(m) = O(n)$, pois $m \leq n$. Como a linha 10 toma tempo $O(1)$, é fácil ver que o laço das linhas 7-10 toma no total tempo $O(n)$. Logo, nosso algoritmo é $O(n \log n)$.

3.2 Descrição através de polígonos quaisquer

Nesta seção apresentaremos um algoritmo que aproxima a fronteira do objeto não com um polígono convexo, que contém todos os pontos do objeto em seu interior, mas sim com um polígono cujos lados aproximam grupos de pontos que são "aproximadamente colineares". Nosso algoritmo não vai fornecer uma aproximação ótima (i.e, com o menor número possível de lados), mas teremos algumas garantias de qualidade para nossa aproximação.

Antes de mais nada, supomos que a fronteira do objeto tenha exatamente um pixel de espessura, o que pode ser conseguido utilizando-se diversas técnicas de *thinning* [2, 3]. Com isso, representaremos nossa fronteira através de uma lista de pontos $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, ordenados em ordem de busca em profundidade.

Com isso queremos dizer que consideramos a 8-vizinhança de cada ponto da fronteira e montamos o grafo de vizinhança correspondente. Escolhemos então um ponto arbitrário para ser o ponto (x_1, y_1) e então executamos uma busca em profundidade através dele. Como nossa fronteira é contínua, eventualmente encontraremos todos os pontos da fronteira e a ordem em que eles forem encontrados pela busca será a ordem que vamos utilizar. A Figura 4 ilustra este processo. Para saber mais sobre busca em profundidade e outros algoritmos elementares em grafos, consulte [1].

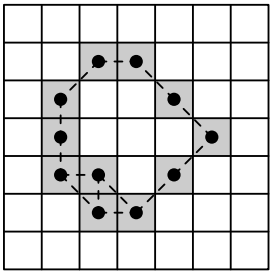

Figura 4: Uma imagem binária e seu grafo de vizinhança. Utilizamos a 8-vizinhança para fazer o grafo. Se começarmos a busca em profundidade a partir do ponto mais a direita, por exemplo, encontraremos todos os pontos da figura numa determinada ordem, que é a ordem em que estamos interessados.

Nosso algoritmo vai então agrupar pixels que são aproximadamente colineares[6] e aproximá-los por um lado do polígono. Utilizaremos para tanto um procedimento *split-and-merge*: examinamos grupos de pontos[7] e verificamos de eles são aproximadamente colineares segundo algum critério. Se um determinado grupo não é aproximadamente colinear, ele é dividido até que os grupos resultantes sejam aproximadamente colineares. Da mesma forma, dois grupos são unidos se o resultado é um grupo de pixels aproximadamente colinear.

Nosso algoritmo tem uma propriedade interessante em relação ao algoritmo exato para o problema, ou seja, em relação ao algoritmo que, segundo nosso critério para determinar se um conjunto de pontos é ou não aproximadamente colinear, encontra o polígono com menor número de lados que aproxima o objeto em questão. Temos então o seguinte resultado:

Proposição 1. *O número de segmentos encontrado pelo procedimento split-and-merge descrito acima é menor do que duas vezes o número de segmentos da solução ótima.*

Demonstração. Considere a seqüência de pontos $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ e considere os intervalos de uma partição ótima desta seqüência, $I_1, I_2, \ldots, I_r$. Com isso queremos dizer que o polígono que aproxima os pontos com um segmento para cada intervalo I_j, $j = 1, 2, \ldots, r$ tem o menor número de lados possível.

Considere agora os intervalos encontrados durante um procedimento *split-and-merge*, $J_1, J_2, \ldots, J_s$. Nenhum intervalo I_j pode conter dois intervalos J_k e J_{k+1}, de outra forma estes teriam sido unidos por nossa rotina *split-and-merge*. Entretanto, cada intervalo I_j pode conter um intervalo J_k dentro de si, com os intervalos J_{k-1} e J_{k+1} interceptando os intervalos I_{j-1} e I_{j+1}. Logo, para cada intervalo I_j podemos ter um intervalo J_k, além de um número de intervalos J_k igual ao número de pontos divisores entre os inervalos I_j. Logo, temos que $s \leq r + r - 1 = 2r - 1$. ■

Segundo [3], este limite é na prática muito menor. Existem, entretanto, exemplos que fazem com que o algoritmo gere polígonos com $2r - 1$ lados.

3.2.1 Descrição do algoritmo *split-and-merge*

O primeiro passo para a descrição do algoritmo *split-and-merge* é descrever como dividir os pontos em grupos, cada grupo sendo aproximado por um lado do polígono. A primeira simplificação que fazemos é a de aproximar um grupo de pontos aproximadamente colineares simplesmente pelo segmento que liga seus

[6] O significado de aproximadamente colineares ainda não está claro e será explicado mais adiante. Podemos usar vários diferentes critérios para determinar se um grupo de pixels é aproximadamente colinear e exporemos nosso critério mais a frente. Por enquanto, o leitor deve imaginar que um determinado critério foi fixado.

[7] Note que, dada a ordenação $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ dos pontos em questão, um grupo de pontos é apenas uma seqüência $(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_{i+k}, y_{i+k})$ para algum k.

pontos terminais, ao invés de procurar pela sua aproximação ótima. O segmento usado para aproximar os pontos $(x_j, y_j), (x_{j+1}, y_{j+1}), \ldots, (x_k, y_k)$ é

$$x(y_j - y_k) + y(x_k - x_j) + y_k x_j - y_j x_k = 0.$$

Com isso, temos o seguinte resultado:

Proposição 2. *Se um ponto (u, v) se encontra fora do segmento dado pela equação acima então sua distância do segmento é igual a d/L, onde*

$$d = u(y_j - y_k) + v(x_k - x_j) + y_k x_j + y_j x_k$$

e

$$L = \sqrt{(y_j - y_k)^2 + (x_k - x_j)^2}.$$

Demonstração. Veja [3]. ■

Um simples teste de colinearidade pode ser executado avaliando-se d/L para cada ponto do grupo. Uma distância máxima pode ser estabelecida tal que os pontos de um grupo são aproximadamente colineares se nenhum deles se encontra mais distante do segmento que une os terminais do grupo do que a distância previamente fixada. Existem casos em que este procedimento não fornece resultados interessantes, mas tais casos não ocorrem muito na prática. O leitor interessado pode consultar [3] para algumas modificações que podem ser feitas no algoritmo para adequá-lo a estes casos.

Com este critério para colinearidade, podemos descrever nosso algoritmo de forma bem simples. Basta considerar os pontos naquela ordem específica e dividí-los inicialmente em grupos de k_0 pontos cada. Verificamos se estes grupos são compostos de pontos aproximadamente colineares, se isto for verdade verificamos a possibilidade de uniões, se não dividimos os grupos nos pontos de máximo erro. Com isso teremos uma partição dos pontos em intervalos de pontos aproximadamente colineares, conseguindo nossa aproximação.

Uma descrição do algoritmo acima em pseudo-código pode ser encontrada em [3].

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.

[2] R. C. Gonzalez, R. E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Co., 1993.

[3] T. Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982.

[4] J. Sklansky, R. L. Chazin, B. J. Hansen, "Minimum-Perimeter Polygons of Digitized Silhouettes", *IEEE Trans. Comput.*, vol C-21, no. 3, pp. 260-268.

[5] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987.

[6] J. O'Rourke, *Computational Geometry in C*, Oxford University Press, 1998.