Universidade Tecnológica Federal do Paraná - UTFPR
Campus Curitiba
Departamento Acadêmico de Eletrônica - DAELN
Curso de Engenharia Eletrônica
Prof. Gustavo B. Borba e Prof. Humberto Gamba

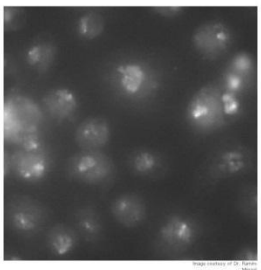
Notas de aula e prática MATLAB #07

Arquivos necessários [disponíveis no zip]

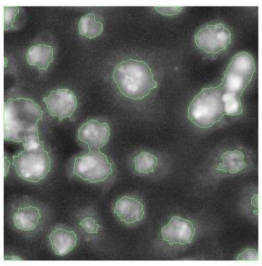
1. whitecells4.png [Adaptada de [JR] Capítulo 5, Figura 2, p. 71 e [GD] Figura 9.11, https://www.cambridge.org/br/academic/subjects/engineering/biomedical-engineering/digital-image-processing-medical-applications?format=HB]
2. exMorph4.bmp
3. visualize.m
4. rice.png [disponível no MATLAB]
5. gradient_with_text.tif [http://www.ogemarques.com/wp-content/uploads/2014/11/Tutorial_Images.zip]

7a) Segmentação por limiarização global

Segmentar uma imagem pode ser definido como a tarefa de separar os objetos de interesse do fundo da imagem. Nos exemplos a seguir são apresentados os resultados de algoritmos desenvolvidos especificamente para a segmentação das imagens de microscopia [SE] e de pílulas farmacêuticas [[MMea], Figura 8].



Células vistas em um microscópio.



Resultado da segmentação (cortorno verde).

Pílula farmacêutica.

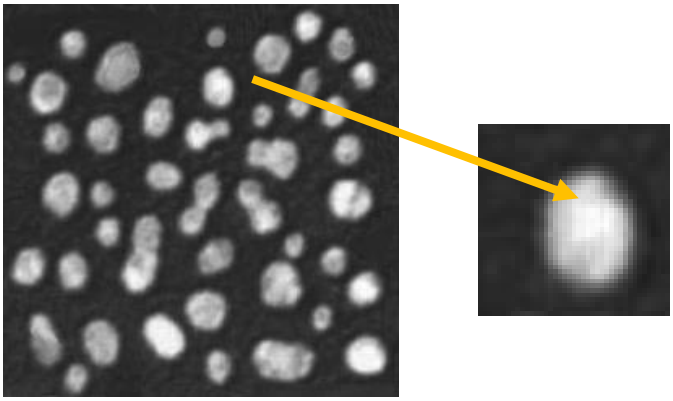
Resultado da segmentação (contorno preto).

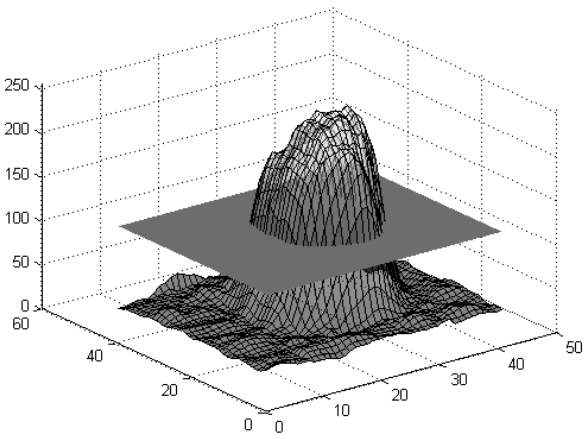
Após a segmentação, o objeto ou conjunto de objetos podem ser submetidos a algoritmos para a extração e análise dos parâmetros de interesse para a aplicação em questão. Na imagem das células, por exemplo, a quantidade pode ser o parâmetro de interesse. No controle de qualidade de uma pílula farmacêutica, o formato do contorno pode ser usado para a identificação de imperfeições nas laterais.

O método mais direto de segmentação é a limiarização (*thresholding*), também chamada de binarização da imagem. Todos os pixels com nível de cinza maior que um valor especificado são transformados em 1, todos os outros são transformados em 0. Isto dá origem a uma imagem chamada de binária (*binary*), lógica (*logical*) ou preto e branco (*bw*). Na imagem segmentada, espera-se que uma das duas categorias de pixels (0 ou 1) corresponda aos objetos da imagem original e a outra categoria corresponda ao fundo da imagem original.

A seguir é mostrado um exemplo de uma imagem de microscopia [[JR], Capítulo 5, Figura 2, p. 71 e [GD] Figura 9.11] na qual as células podem ser separadas do fundo através da segmentação por limiarização. No gráfico que mostra uma região da imagem como uma superfície, o valor de limiar 128 é mostrado como um plano.

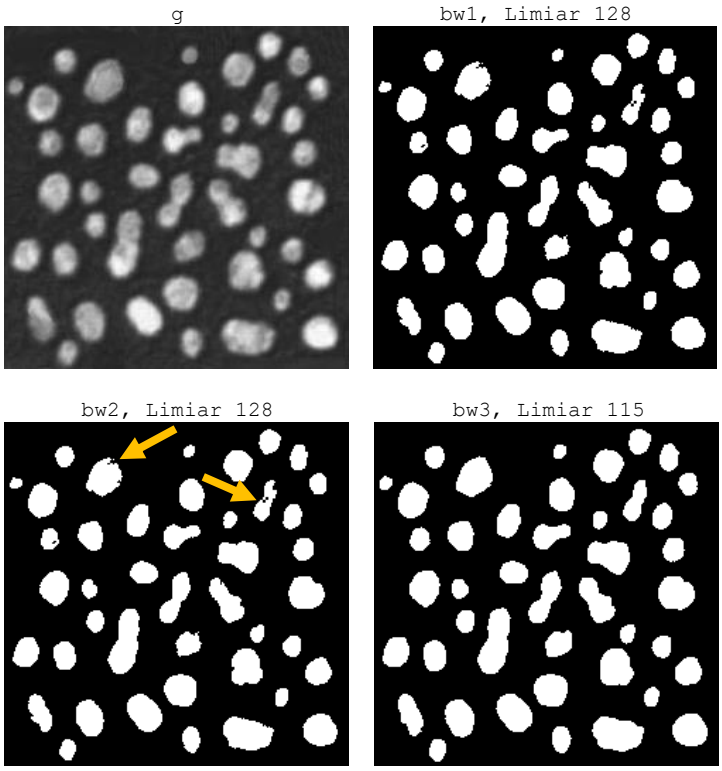
```
% blobs_surface [script]
clc, clear, close all

g = imread('whitecells4.png');

%Coord do canto sup esq da região
%imin = 30
%jmin = 110
%Coord do canto inf dir da região
%imax = 70
%jmax = 150
reg = g(30:70,110:150);
%Plano em 128,
%de mesmas dimensões da região
z = ones(41,41).*128;

%Display
figure, imshow(g)
figure, imshow(reg)
figure
surf(double(reg), 'FaceAlpha', 0.5)
zlim([0 255])
colormap('gray')
hold on
surf(z, 'EdgeColor', 'none')
hold off
```

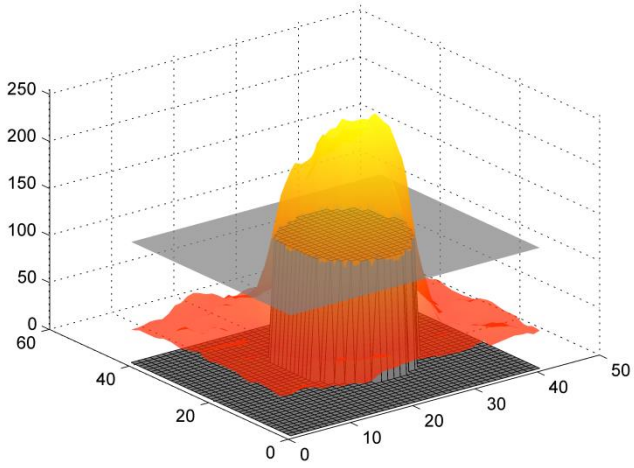




No MATLAB, os operadores relacionais podem ser usados para a limiarização de uma imagem. Também existe a função `im2bw`. Este tipo de limiarização, na qual um mesmo limiar (*threshold*) é aplicado em todos os pixels da imagem, é denominada *limiarização global*.

```
% blobs_th_global [script]
clc, clear, close all

g = imread('whitecells4.png');

% Usando operador relacional
% IMPORTANTE: bw1 é
% da classe logical
bw1 = g > 128;

% Usando a função im2bw
% O limiar deve ser um
% número entre 0 e 1
% IMPORTANTE: bw2, bw3 são
% da classe logical
%Limiar 128 (0.5)
bw2 = im2bw(g, 128/255);
%Limiar 115 (0.45)
bw3 = im2bw(g, 115/255);

% Display
figure, imshow(g)
title('g')
figure, imshow(bw1)
title('bw1, Limiar 128')
figure, imshow(bw2)
title('bw2, Limiar 128')
figure, imshow(bw3)
title('bw3, Limiar 115')

% A região do exemplo anterior
reg = g(30:70,110:150);
regbw = bw2(30:70,110:150);
% O plano de limiarização em 128
z = ones(41,41).*128;
% Display
% Mostra a imagem como uma
% superfície usando pseudocores
% para facilitar a visualização
figure
surf(double(reg), 'FaceAlpha', 0.7)
shading('interp')
zlim([0 255])
colormap('autumn')
hold on
% +regbw transforma de logical
% para double.
% Mostra os pixels '1' da imagem
% limiarizada com o valor 128,
% isto é, no mesmo nível do plano
% de corte que limiarizou a imagem.
surf((+regbw)*128,...
    'FaceColor', [0.5 0.5 0.5])
% Mostra o plano de corte que
% limiarizou a imagem
surf(z, 'EdgeColor', 'none',...
    'FaceColor', [0.5 0.5 0.5],...
    'FaceAlpha', 0.7)
hold off
```



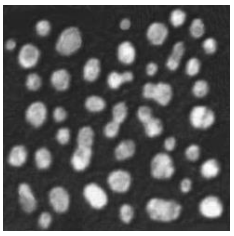


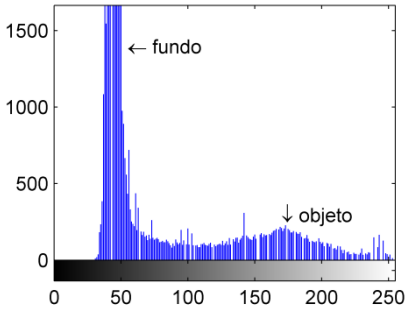
Observe que estamos encontrando o valor de limiar mais adequado de forma manual. Como o valor 128 (0.5) resultou na não detecção de alguns pixels das células, como apontado pelas setas, dimunuímos o valor do limiar para 115 (0.45).

7b) Limiarização automática

Existem diferentes métodos para encontrar automaticamente um valor de limiar [[JP], Capítulo 3 é uma boa referência e em [GLea] há uma boa lista e implementação em Java]. O mais popular é o de *Otsu*, nome do prof. que desenvolveu o método [NO]. Como este método baseia-se no histograma da imagem, vamos primeiro obter o histograma da nossa imagem exemplo e fazer algumas observações.

```
% blobs_histogram [script]
clc, clear, close all

g = imread('whitecells4.png');

figure
subplot(2,1,1)
imshow(g)
subplot(2,1,2)
imhist(g)
text(55,1400,'\leftarrow fundo',...
     'HorizontalAlignment','left')
text(170,300,'\downarrow objeto',...
     'HorizontalAlignment','left')
```





Note que o histograma possui dois grupos (concentrações de pixels) que se destacam. Este tipo de histograma é chamado de *bimodal*. Isso é o esperado, já que a imagem possui dois grupos de pixels bem distintos: escuros para o fundo e claros para as células. Então, o grupo da esquerda no histograma, dos pixels mais escuros da imagem, corresponde ao fundo. O da direita, dos pixels mais claros da imagem, corresponde às células. O método de Otsu procura o nível de cinza que melhor separa, segundo o critério estatístico da variância, estes dois grupos. O procedimento é o seguinte [[SS], Capítulo 3, Tópico 3.8.2]:

1. Para um determinado limiar t:
   - A variância do grupo de pixels de valor menor ou igual a t é $\sigma_1^2(t)$
   - A variância do grupo de pixels de valor maior que t é $\sigma_2^2(t)$
   - A probabilidade do grupo de pixels de valor menor ou igual a t é $q_1(t)$
   - A probabilidade do grupo de pixels de valor maior que t é $q_2(t)$
2. O valor ótimo de limiar é aquele que minimiza a variância interna dos grupos de pixels (*within-class variance* ou *intraclass variance*) σ_w^2, sendo:

$$\sigma_W^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Os valores de $q_1(t)$, $q_2(t)$, $\sigma_1^2(t)$ e $\sigma_1^2(t)$ podem ser obtidos a partir do próprio histograma normalizado $P(i)$ da imagem. I é o nível de cinza máximo (255 para imagens de 8 bits):

$$q_1(t) = \sum_{i=1}^{t} P(i) \qquad \mu_1(t) = \sum_{i=1}^{t} i\frac{P(i)}{q_1(t)}$$

$$q_2(t) = \sum_{i=t+1}^{I} P(i) \qquad \mu_2(t) = \sum_{i=t+1}^{I} i\frac{P(i)}{q_2(t)}$$

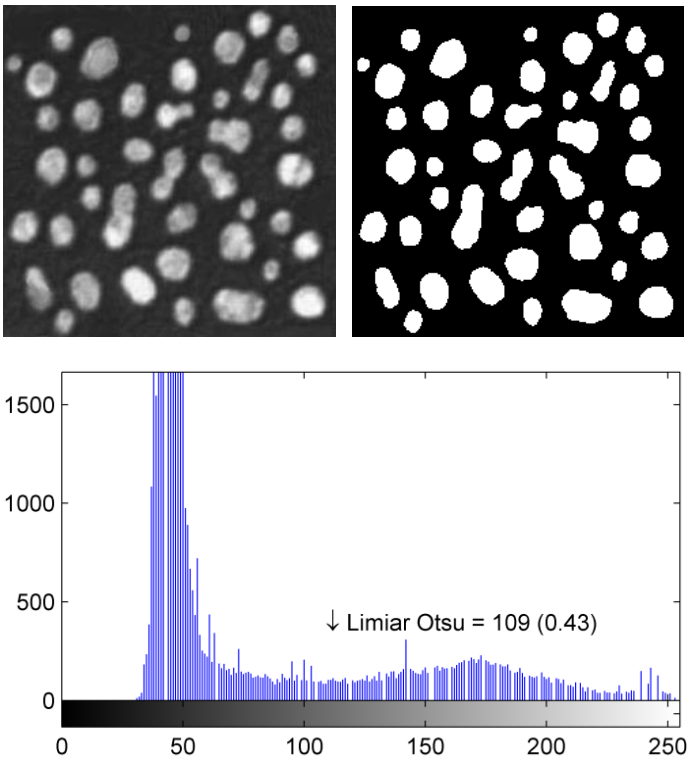
$$\sigma_1^2(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 P(i)/q_1(t)$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i-\mu_2(t)]^2 P(i)/q_2(t)$$

Este procedimento pode ser implementado computacionalmente fazendo-se a busca exaustiva do menor σ_w^2 para todos os t possívies, de 0 até 255. Porém, existe uma forma mais eficiente de implementação, que utiliza um conjunto de equações disponívies em [[SS], Capítulo 3, Tópico 3.8.2].

A função do MATLAB que implementa a limiarização pelo método de Otsu é a `graythresh`.

```
% blobs_th_global_auto [script]
clc, clear, close all

g = imread('whitecells4.png');

th = graythresh(g);
bw = im2bw(g, th);

% Nível de cinza do th
% na faixa [0 255]
thp = th*255;

%Display
figure, imshow(g)
figure, imshow(bw)
figure, imhist(g)
text(thp,400,['\downarrow' ,...
    ' Limiar Otsu = ',...
    num2str(thp) ,...
    ' (' num2str(th,2) ')'])
```



7c) Operações morfológicas

Morfologia significa "estudo da forma, estrutura, aparência etc. da matéria" [http://aulete.uol.com.br/morfologia]. Então, as operações morfológicas tem o objetivo de analisar e manipular a forma (*shape*) dos objetos na imagem. Vamos abordar a morfologia matemática aplicada às imagens binárias, que é o uso mais comum, mas existem também definições das mesmas operações morfológicas vistas aqui, para aplicação em imagens em níveis de cinza [[BB] Tópico 10.4 explica com clareza, [OM] Tópico 13.7, [GW] Tópico 9.6] e coloridas [CD],[LVA]. Em [MS] há curiosidades sobre a história da morfologia matemática.

As operações básicas, ou primitivas, das quais muitas outras são derivadas são a *dilatação (dilation)* e a *erosão (erosion)*. As operações de *fechamento (closing)* e *abertura (opening)* são obtidas a partir da combinação de dilatação e erosão e também são bastante utilizadas. Existem muitas outras operações baseadas em morfologia, como a hit-or-miss (para detectar um formato pré-definido), boundary extraction, hole filling, convex hull, thinning, thickening, skeletons, pruning e reconstrução morfológica [[GW], Tópico 9.5].

A base matemática da morfologia é a teoria de conjuntos. Um conjunto de pixels binários denominado *elemento estruturante* (*structuring element*) varre a imagem (outro conjunto de pixels binários) e as operações morfológicas são realizadas entre o elemento estruturante e os pixels da imagem contidos no elemento estruturante. Este processo de 'varrer a imagem com o elemento estruturante' é similar àquele que realizamos em um box filter, por exemplo. Mas ao invés de aplicar

uma convolução, aplicamos a operação morfológica na máscara (elemento estruturante). O elemento estruturante pode ter um formato arbitrário e depende da aplicação.

A descrição matemática formal das operações morfológicas, na maioria da vezes utilizando notação de conjuntos, pode ser encontrada nas referências [[BB] Tópicos 10.1, 10.2 e 10.3 são uma boa sugestão]. A seguir, vamos descrevê-las textualmente.

Algumas definições:

1) O *ponto de referência* do elemento estruturante é o equivalente em morfologia ao ponto central das janelas de convolução quadradas. Este ponto de referência também é chamado de *ponto central* do elemento estruturante. Mas como este ponto central muitas vezes não é central, geometricamente falando, vamos usar o termo *hot spot*, como em [[BB], Tópico 10.2.1]. Chamaremos o elemento estruturante de S. No MATLAB, o *hot spot* é denominado *origin* e às vezes *center*.

2) Estamos aplicando morfologia em imagens binárias (bw). Vamos convencionar que os pixels dos objetos possuem sempre valor '1' (pixels brancos). Logo, os pixels do fundo possuem sempre valor '0' (pixels pretos). Vamos chamar a imagem de entrada de B. A mesma idéia pode ser usada para os pontos do elemento estruturante: um ponto ativo do elemento estruturante tem valor '1'. Logo, um ponto não ativo (irrelevante) do elemento estruturante tem valor '0'.

3) A *vizinhança-4* (4-neighborhood) de um pixel é definida pelos dois pixels adjacentes à ele na horizontal e pelos dois pixels adjacentes à ele na vertical.

4) *A vizinhança-8* (8-neighborhood) de um pixel é definida pela vizinhança-4, mais os 4 pixels diagonais à ele.

5) Notação:

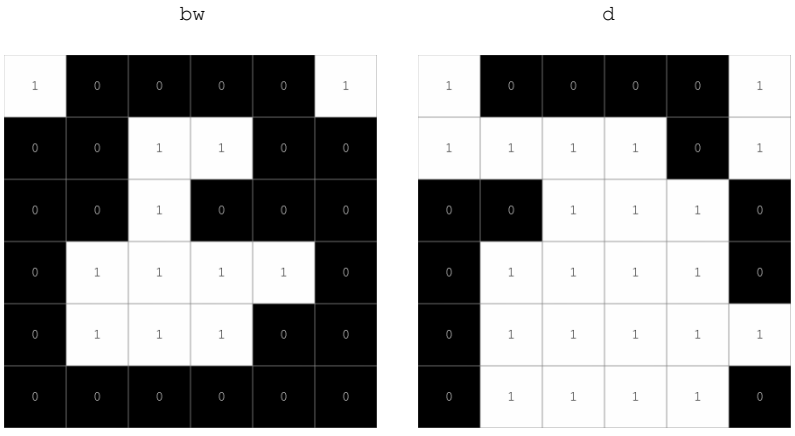
B⊕S - **Dilatação** da imagem B pelo elemento estruturante.
BϴS - **Erosão** da imagem B pelo elemento estruturante S.
B•S - **Fechamento** da imagem B pelo elemento estruturante S.
B○S - **Abertura** da imagem B pelo elemento estruturante S.

6) Assim como nas operações baseadas em vizinhança que vimos em outros tópicos da disciplina, a saída das operações morfológicas deve ser escrita em uma nova imagem, inicialmente preenchida com zeros.

Função MATLAB para criar um elemento estruturante: `strel`. Funções MATLAB para implementar as quatro operações morfológicas citadas: `imdilate, imerode, imclose, imopen`.

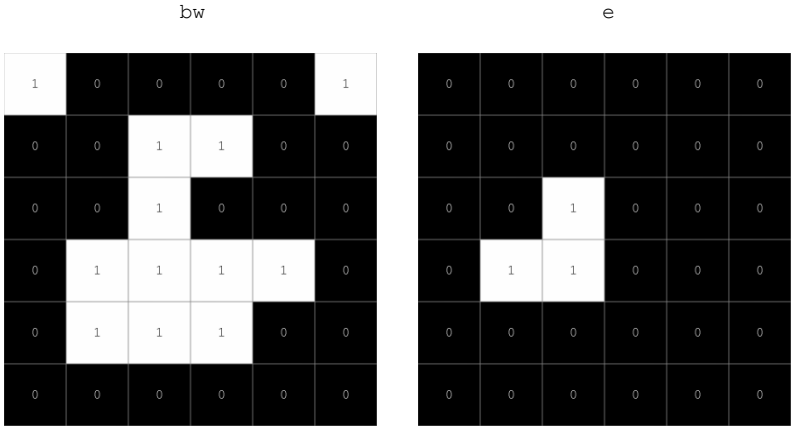
Dilatação (B⊕S). Cada vez que o hot spot de S encontra um pixel '1' de B, S é replicado na imagem de saída.

```
% dilation_bw6 [script]
clc, clear, close all
imtool close all

bw = [ 1 0 0 0 0 1
       0 0 1 1 0 0
       0 0 1 0 0 0
       0 1 1 1 1 0
       0 1 1 1 0 0
       0 0 0 0 0 0 ];

bw = logical(bw);

% O hot spot é dado por
% floor((size(NHOOD)+1)/2)
% Logo, o hotspot do el est
% abaixo é em (1,1)
NHOOD = [ 1 0
          1 1 ];
SE = strel('arbitrary', NHOOD);
d = imdilate(bw, SE);

imtool(bw)
imtool(d)
```



Erosão (BϴS). O pixel '1' de B correpondente ao hot spot de S só é mantido na imagem de saída se *todos* os pixels '1' de S possuem pixel '1' correspondente em B.

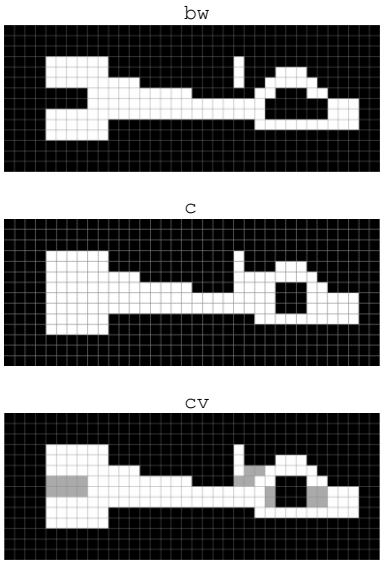
```
% erosion_bw6 [script]
clc, clear, close all
imtool close all

bw = [ 1 0 0 0 0 1
       0 0 1 1 0 0
       0 0 1 0 0 0
       0 1 1 1 1 0
       0 1 1 1 0 0
       0 0 0 0 0 0];

bw = logical(bw);

% O hot spot é dado por
% floor((size(NHOOD)+1)/2)
% Logo, o hotspot do el est
% abaixo é em (1,1)
NHOOD = [ 1 0
          1 1 ];
SE = strel('arbitrary', NHOOD);
e = imerode(bw, SE);

imtool(bw)
imtool(e)
```



Fechamento (B•S). B•S = (B⊕S)ϴS. É a dilatação seguida da erosão, isto é, aplicar a dilatação na imagem B usando S e depois aplicar a erosão na imagem dilatada usando S.

A operação de fechamento atua nos objetos da imagem principalmente das seguintes formas: funde quebras estreitas, elimina pequenos buracos, preenche espaços de contornos.

```
% closing_bw36 [script]
clc, clear, close all
imtool close all

bw = imread('exMorph4.bmp');
se = strel('square', 3);
c = imclose(bw, se);
cv = visualize(bw, c);

figure
t = tiledlayout(3,1);
t.TileSpacing = 'tight'; t.Padding='compact';
nexttile
imshow(bw,'InitialMagnification','fit'),
title('bw')
nexttile
imshow(c,'InitialMagnification','fit'),
title('c')
nexttile
imshow(cv,'InitialMagnification','fit'),
title('cv')
```



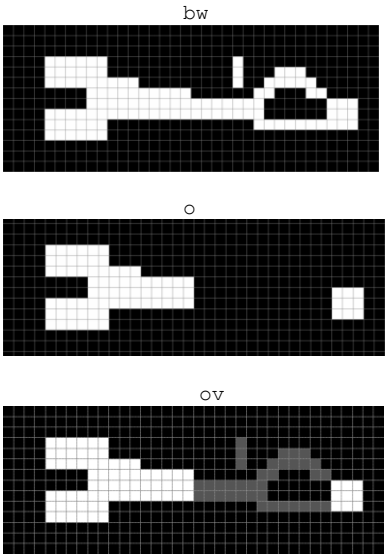

```
function v = visualize(bwo, bwp)
% v: saida uint8
% bwo: bw original
% bwp: bw processada
% 0 em bwo -> 0 em bwp: 0 em v,
% 1 em bwo -> 1 em bwp: 1 em v,
% 0 em bwo -> 1 em bwp: 170 em v,  (cinza claro)
% 1 em bwo -> 0 em bwp: 85 em v,   (cinza escuro)

v = uint8(bwo & bwp)*255;    % 1 em bwo -> 1 em bwp: 1 em v
idx = (bwo==0 & bwp==1);     % 0 em bwo -> 1 em bwp: 170 em v
v(idx) = 170;
idx = (bwo==1 & bwp==0);     % 1 em bwo -> 0 em bwp: 85 em v
v(idx) = 85;
```

Função visualize no arquivo *visualize.m*

Abertura (B○S). B○S = (BϴS)○S. É a erosão seguida da dilatação, isto é, aplicar a erosão na imagem B usando S e depois aplicar a dilatação na imagem erodida usando S.

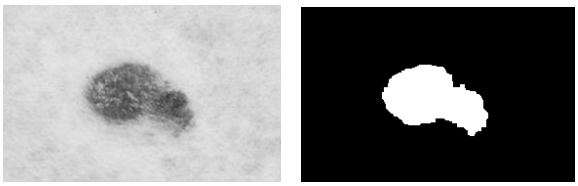
A operação de abertura atua nos objetos da imagem principalmente das seguintes formas: quebra ligações estreitas, elimina protrusões finas.

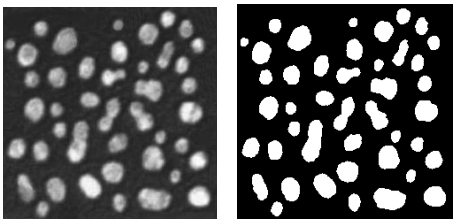
```
% opening_bw36 [script]
clc, clear, close all
imtool close all

bw = imread('exMorph4.bmp');
se = strel('square', 3);
o = imopen(bw, se);
ov = visualize(bw, o);

figure
t = tiledlayout(3,1);
t.TileSpacing = 'tight'; t.Padding='compact';
nexttile
imshow(bw,'InitialMagnification','fit'),
title('bw')
nexttile
imshow(o,'InitialMagnification','fit'),
title('o')
nexttile
imshow(ov,'InitialMagnification','fit'),
title('ov')
```



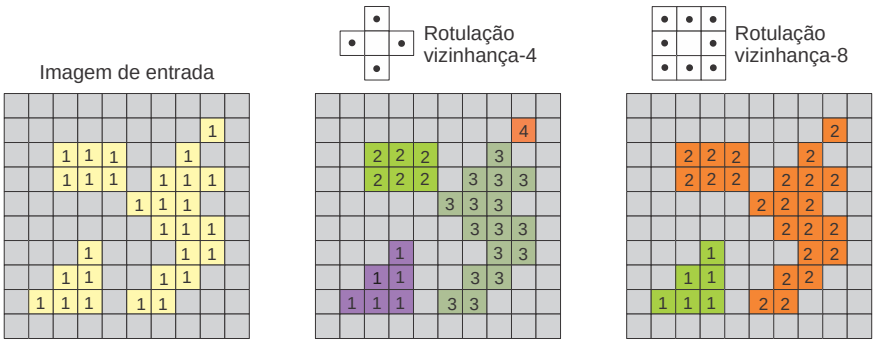
7d) Rotulação (labeling)

Considerando nossos exemplos da aula anterior (imagens abaixo) [esquerda: adaptada de [CS], Figura 6; direita: [JR], Capítulo 5, Figura 2, p. 71 e [GD] Figura 9.11], podemos fazer as perguntas: quantas lesões de pele existem na imagem e qual o seu tamanho? Sabemos que existe apenas uma porque a imagem original adquiriu apenas uma lesão e o nosso processo de segmentação garantiu que na imagem segmentada há um único grupo de pixels conectados, correspondentes ao único objeto. Então, para obter a área da lesão em pixels, basta contar o número de pixels de objeto na imagem segmentada. Mas se fizermos as mesmas perguntas para a imagem de microscopia contendo as células, a resposta não é direta, é necessário primeiro realizar a rotulação dos objetos da imagem segmentada.





O objetivo da rotulação (*labeling*) é discriminar (nomear) cada objeto da imagem binária. Neste contexto, os objetos são também denominados de *componentes conexos* (este é um termo que você não pode esquecer). Na nossa convenção, os componentes conexos possuem pixel '1' e fundo pixel '0'. Um componente conexo é um conjunto de pixels conectados pela vizinhança-4 ou vizinhança-8. A saída da rotulação é uma imagem na qual cada componente conexo é identificado por um valor de pixel único, como nos exemplos a seguir.

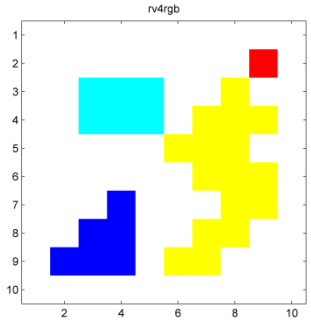


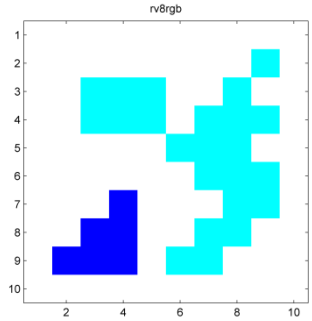
Na prática, a rotulação geralmente ocorre após uma segmentação e faz parte do processo de análise da imagem. As características a serem analisadas de cada componente conexo (objeto) dependem da aplicação. Dentre as mais diretas estão a contagem dos objetos na imagem e a medida da área em pixels, mas há uma grande quantidade de métodos disponíveis na literatura para a extração de outros atributos dos objetos binários. Por exemplo: centroide, perímetro, excentricidade e muitos outros [[SS] Tópico 3.6, [OM] Tópicos 18.3 e 18.4, [GW], Capítulo 11].

No código abaixo, a rotulação foi realizada com a função `bwlabel`, mas vale a pena saber: "The functions `bwlabel`, `bwlabeln`, and `bwconncomp` all compute connected components for binary images. `bwconncomp` replaces the use of `bwlabel` and `bwlabeln`. It uses significantly less memory and is sometimes faster than the other functions."
[https://www.mathworks.com/help/images/ref/bwconncomp.html]

```
% labeling_bw10 [script]
clc, clear, close all

bw = [ 0 0 0 0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0 0 1 0
       0 0 1 1 1 0 0 1 0 0 0
       0 0 1 1 1 0 1 1 1 0 0
       0 0 0 0 0 1 1 1 0 0 0
       0 0 0 0 0 0 1 1 1 0 0
       0 0 0 1 0 0 0 1 1 0 0
       0 0 1 1 0 0 1 1 0 0 0
       0 1 1 1 0 1 1 0 0 0 0
       0 0 0 0 0 0 0 0 0 0 0 ];

bw = logical(bw);

% A saída de bwlabel
% é da classe double
[rv4 nrv4] = bwlabel(bw, 4);
[rv8 nrv8] = bwlabel(bw, 8);

% Aplica pseudocores na
% imagem rotulada para
% visualização
rv4rgb = label2rgb(rv4);
rv8rgb = label2rgb(rv8);
disp(['n. obj em rv4 = ',...
    num2str(nrv4)])
disp(['n. obj em rv8 = ',...
    num2str(nrv8)])

% Display
figure, image(rv4rgb)
title('rv4rgb')
figure, image(rv8rgb)
title('rv8rgb')
```

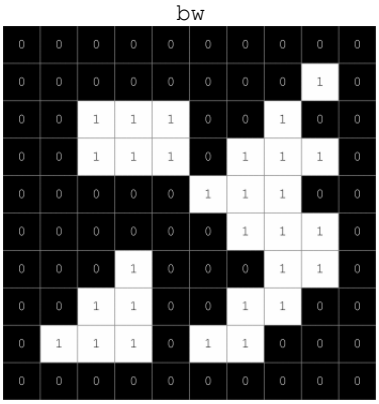
```
>> rv4
rv4 =
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     4     0
     0     0     2     2     2     0     0     3     0     0
     0     0     2     2     2     0     3     3     3     0
     0     0     0     0     0     3     3     3     0     0
     0     0     0     0     0     0     3     3     3     0
     0     0     0     1     0     0     0     3     3     0
     0     0     1     1     0     0     3     3     0     0
     0     1     1     1     0     3     3     0     0     0
     0     0     0     0     0     0     0     0     0     0
>> rv8
rv8 =
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     2     0
     0     0     2     2     2     0     0     2     0     0
     0     0     2     2     2     0     2     2     2     0
     0     0     0     0     0     2     2     2     0     0
     0     0     0     0     0     0     2     2     2     0
     0     0     0     1     0     0     0     2     2     0
     0     0     1     1     0     0     2     2     0     0
     0     1     1     1     0     2     2     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

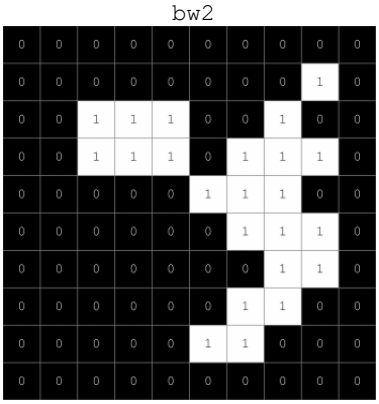




Uma vez que os objetos foram rotulados, é possível obter as coordenadas dos pixels de cada um utilizando a função `find`. Para medir a área de um objeto em pixels, basta contar as suas coordenadas. A partir das coordenadas pode-se também manipular o objeto desejado. No código a seguir, por exemplo, eliminamos o menor objeto da imagem.

```
% labeling_area_bw10 [script]
clc,clear, close all
imtool close all

bw = [ 0 0 0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0 1 0
       0 0 1 1 1 0 0 1 0 0
       0 0 1 1 1 0 1 1 1 0
       0 0 0 0 0 1 1 1 0 0
       0 0 0 0 0 0 1 1 1 0
       0 0 0 1 0 0 0 1 1 0
       0 0 1 1 0 0 1 1 0 0
       0 1 1 1 0 1 1 0 0 0
       0 0 0 0 0 0 0 0 0 0 ];

bw = logical(bw);
[rv8 nobj] = bwlabel(bw, 8);
imshow(bw), title('bw')
impixelregion

% Aloca vetor para armazenar o número de pixels
% de cada objeto
area = zeros(1,nobj);
% Aloca cell array para armazenar os índices
% lineares de cada objeto
linearIdx = cell(1,nobj);

% Para cada objeto
for k = 1:nobj
    % Coordenadas lineares dos pixels do objeto k
    linearIdx{k} = find(rv8==k);
    % Número de pixels do objeto k
    area(k) = length(linearIdx{k});
    fprintf('Area em pixels do obj %d = %d\n', k,
area(k));
end

% Elimina o objeto menor
bw2 = bw;
[valor idx] = min(area);
bw2(linearIdx{idx}) = 0;
figure, imshow(bw2), title('bw2')
impixelregion
```





Esta é uma das maneiras de se obter as coordenadas e a área dos objetos binários no MATLAB e pode ser utilizada também em outros ambientes ou linguagens, desde que a imagem rotulada esteja disponível. No MATLAB, se a rotulação for realizada com a função `bwconncomp` ao invés de `bwlabel`, o `find` não é necessário. Isto porque `bwconncomp` retorna uma struct na qual um dos campos é um cell array com os índices lineares de cada pixel de cada objeto. Uma maneira mais direta ainda de obter a área é através da função `regionprops`, que pode retornar também várias outras propriedades dos objetos.

Pontanto, reforçando... as seguintes funções são muito úteis e versáteis:

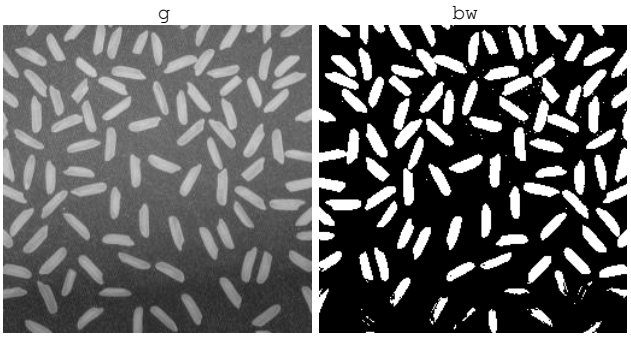
bwconncomp
https://www.mathworks.com/help/images/ref/bwconncomp.html

regionprops
https://www.mathworks.com/help/images/ref/regionprops.html

7e) Transformada top-hat

A segmentação por limiarização global pode ser comprometida em situações nas quais a iluminação na imagem não é uniforme (homogênea), como no exemplo a seguir (é um exemplo bem conhecido: https://www.mathworks.com/help/images/image-enhancement-and-analysis.html e https://www.mathworks.com/help/images/ref/imtophat.html. Talvez também exista uma versão desse exemplo no blog do Steve Eddins [SE]).

```
% otsu_rice [script]
clc, clear, close all

g = imread('rice.png');
figure, imshow(g);

t = graythresh(g);
bw = im2bw(g, t);
figure, imshow(bw)

% Observe que os grãos da parte
% inferior não são mantidos
% na imagem limiarizada
```



A transformada top-hat pode ser utilizada como uma etapa de pré-processamento da imagem, para uniformizar a iluminação do fundo (background), antes de aplicar a limiarização global. A transformada top-hat TH de uma imagem grayscale G é:

$$TH = G - (G \circ S)$$

onde '$\circ$' é a operação morfológica de abertura de G por um elemento estruturante S e '–' é uma subtração de imagens. Assim como nas operações morfológicas binárias, a abertura morfológica de uma imagem grayscale (G$\circ$S) é o resultado da erosão seguida da dilatação, isto é, aplicar a erosão na imagem G usando S e depois aplicar a dilatação na imagem erodida usando S. As figuras a seguir [[BB3], Figura 7.23 e Figura 7.24] ilustram as operações de dilatação e erosão morfológica para imagens grayscale (o elemento estruturante usado nas figuras é só um exemplo. Os elementos estruturantes mais usados na prática possuem apenas valores 1, 0 ou don't care (parece digital, mas é coincidência)).

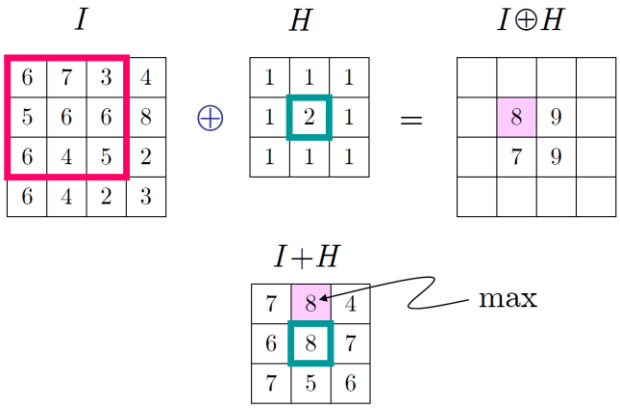


Fig. 7.23 Grayscale dilation $I \oplus H$. The 3×3 pixel structuring element H is placed on the image I in the upper left position. Each value of H is added to the corresponding element of I; the intermediate result $(I + H)$ for this particular position is shown below. Its maximum value $8 = 7 + 1$ is inserted into the result $(I \oplus H)$ at the current position of the filter origin. The results for three other filter positions are also shown.

Figura: Wilhelm Burger and Mark J. Burge,Digital Image Processing: An Algorithmic Introduction, 3rd ed., Springer, 2022, Fig. 7.23.
Site do livro: https://imagingbook.com/

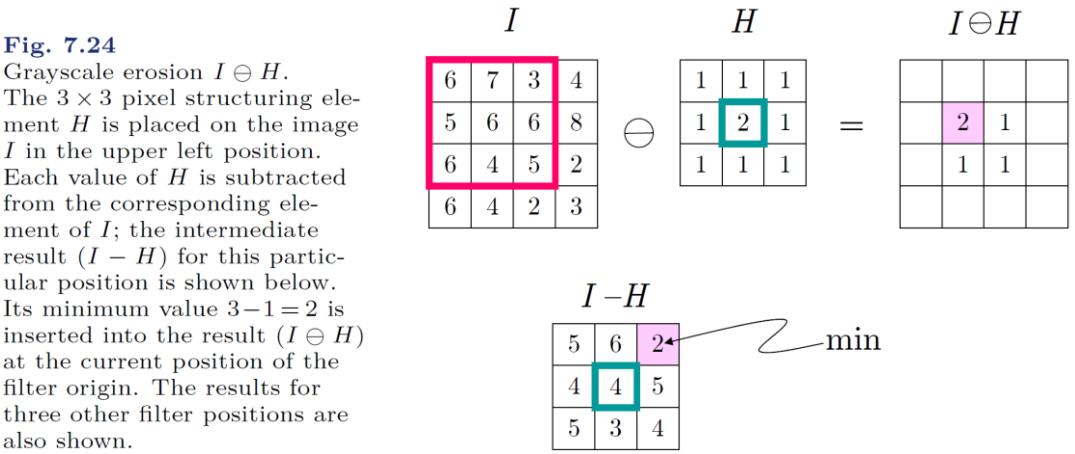
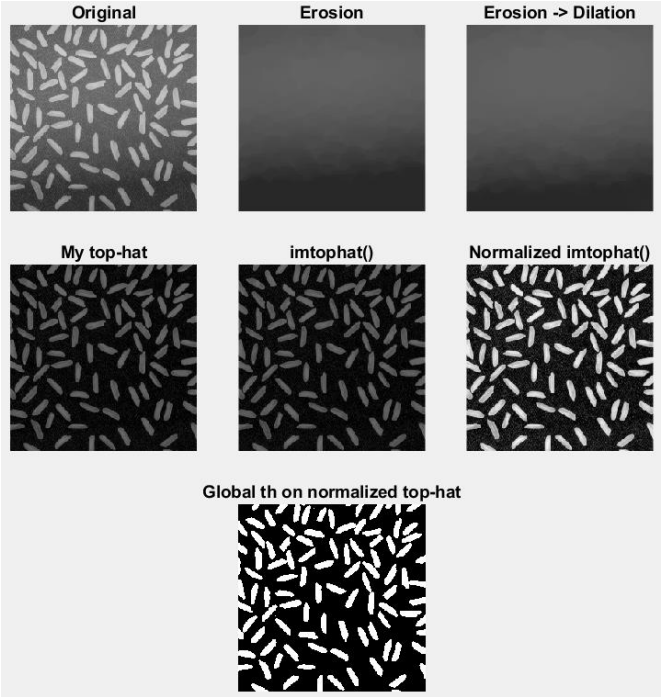


Fig. 7.24
Grayscale erosion $I \ominus H$. The 3×3 pixel structuring element H is placed on the image I in the upper left position. Each value of H is subtracted from the corresponding element of I; the intermediate result $(I - H)$ for this particular position is shown below. Its minimum value $3-1=2$ is inserted into the result $(I \ominus H)$ at the current position of the filter origin. The results for three other filter positions are also shown.

Figura: Wilhelm Burger and Mark J. Burge,Digital Image Processing: An Algorithmic Introduction, 3rd ed., Springer, 2022, Fig. 7.24.
Site do livro: https://imagingbook.com/

```
% tophat_rice [script]
clc, clear, close all

g = imread('rice.png');
figure
h = tiledlayout(3,3);
h.TileSpacing = 'tight';
h.Padding = 'compact';
nexttile, imshow(g); title('Original')

SE = strel('disk',15);
ge = imerode(g, SE);
nexttile, imshow(ge); title('Erosion')
ged = imdilate(ge, SE);
nexttile, imshow(ged);
title('Erosion -> Dilation')

gtophat = imsubtract(g, ged)
nexttile, imshow(gtophat);
title('My top-hat')
tophat = imtophat(g, SE);
nexttile, imshow(tophat)
title('imtophat()')
tophat_n = mat2gray(tophat);
nexttile, imshow(tophat_n);
title('Normalized imtophat()')

t = graythresh(tophat_n);
tophat_bw = im2bw(tophat_n, t);
nexttile([1 3]), imshow(tophat_bw);
title('Global th on normalized top-hat')
```

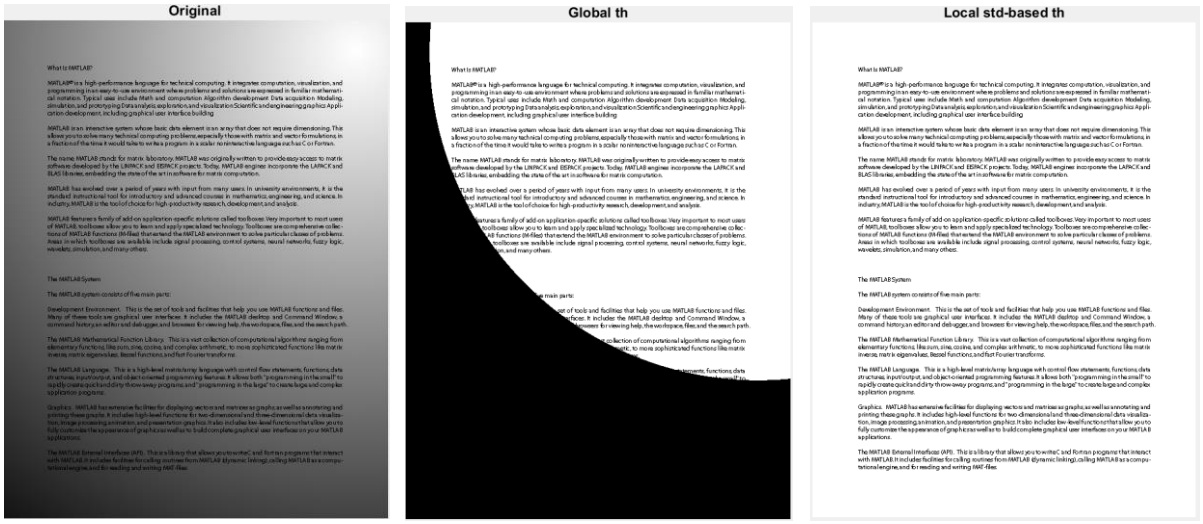


7f) Limiarização local (também chamada de limiarização adaptativa) baseada no desvio padrão

O código abaixo implementa uma **limiarização local** (adaptativa). Rodar o código e entender (é simples). Fonte: tutorial '**Adaptive Thresholding'** do livro [OM] (capítulo 15, páginas 350 até 352).

```
% local_stdbased_th [script]
% Tutorial 'Adaptive Thresholding',
% capítulo 15, páginas 382 até 384 do livro
% Oge Marques, Practical image and video
% processing using MATLAB, Wiley, 2011.
clc, clear, close all

g = imread('gradient_with_text.tif');
figure, imshow(g), title('Original')

% Global Th
ggth = im2bw(g, graythresh(g));
figure, imshow(ggth), title('Global th')

% Local Th
% Função blockproc é sem overlap
% Cria function handle
fun = @(myBlock) adaptThStd(myBlock.data);
gath = blockproc(g, [10 10], fun); %
processa
figure, imshow(gath)
title('Local std-based th')
```

```
function y = localThStd(x)
% x: bloco da imagem (subimagem)
% y: bloco processado

 if(std2(x))<1 %std baixo: é fundo
   %devolve um bloco de uns
   y = ones(size(x,1),size(x,2));
 else %std alto: é texto
    %Aplica Otsu no bloco e devolve
    y = im2bw(x, graythresh(x));
 end

end
```



Referências

[JR] Jadwiga Rogowska, Overview and fundamentals of image segmentation, In: Handbook of medical imaging: Processing and analysis management, Editor Isaac Bankman, Academic Press, 2000.

[GD] Geoff Dougherty, Digital Image Processing for Medical Applications, Cambridge University Press, 2009.

[OM] Oge Marques, Practical image and video processing using MATLAB, Wiley, 2011.

[GW] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, Digital image processing, Pearson Prentice Hall, 3rd ed, 2008.

[BB] Wilhelm Burger, Mark Burge, Digital image processing – an algorithmic introduction using Java, Springer, 2010.

[MMea] Miha Mozina, Dejan Tomazevic, Franjo Pernus, Bostjan Likar, Real-time image segmentation for visual inspection of pharmaceutical tablets, Machine Vision and Applications, v. 22, p. 145-156, 2011.

[SE] Steve Eddins, Cell segmentation, Blog: Steve on Image Processing, Imagem: Ramiro Massol, junho, 2006. Disponível em http://blogs.mathworks.com/steve/2006/06/02/cell-segmentation/,

[JP] James R. Parker, Algorithms for image processing and computer vision, John Wiley & Sons, 1997

[SS] Linda G. Shapiro, George C. Stockman, Computer Vision, Prentice Hall, 2001. Disponível em http://www.cse.msu.edu/~stockman/Book

[GLea] Gabriel Landini et al., Auto Threshold plugin para Fiji – ImageJ, 2011. Disponível em http://fiji.sc/Auto_Threshold.

[NO] Nobuyuki Otsu, A Threshold Selection Method from Gray-Level Histograms, IEEE Transactions on Systems, Man and Cybernetics, v. SMC-9, n. 1, 1979.
Disponível em
http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4310064&punumber=21
Link direto: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04310076

[MS] Georges Matheron and Jean Serra, Georges Matheron and Jean Serra, Slides, 2000. Disponível em http://cmm.ensmp.fr/~serra/pdf/birth_of_mm.pdf, via Wikipedia http://en.wikipedia.org/wiki/Mathematical_morphology.

[CD] Mary L. Comer, Edward J. Delp, Morphological operations for color image processing, Journal of Electronic Imaging, n.8, v. 3, p. 279-298, 1999. Disponível em ftp://jetty.ecn.purdue.edu/ace/delp/color-morph/paper.pdf

[LVA] G. Louverdis, M.I. Vardavoulia, I. Andreadis, A new approach to morphological color image processing, Pattern Recognition, v. 35, issue 8, p. 1733-1741, 2002.

[CS] Pablo G. Cavalcanti and Jacob Scharcanski, Macroscopic Pigmented Skin Lesion Segmentation and its Influence on the Lesion Classification and Diagnosis, In: Color Medical Image Analysis, Lecture Notes in Computational Vision and Biomechanics, Volume 6, 2013, pp 15-39, Edited by M. Emre Celebi and Gerald Schaefer, Springer Netherlands. Disponível em
https://www.researchgate.net/publication/260713485_Macroscopic_pigmented_skin_lesion_segmentation_and_its_influence_on_lesion_classification_and_diagnosis

[BB3] Wilhelm Burger and Mark J. Burge, Digital Image Processing: An Algorithmic Introduction, 3rd ed., Springer, 2022. Site do livro: https://imagingbook.com/