[PSI3471-2022 aula 13] A aula de hoje não tem lição de casa.

Morfologia Matemática

1. Erosão e dilatação

Como vimos, filtragem (figura 1) é uma transformação de imagem onde a cor $B(p)$ de um pixel p da imagem de saída B é escolhida em função das cores $W(p)$ da vizinhança (janela) W de p na imagem de entrada A. Já vimos vários filtros, por exemplo: média móvel, filtro mediano, filtro linear, filtro gaussiano, filtro laplaciano, etc. Nesta aula, veremos os filtros morfológicos. Os filtros básicos da Morfologia Matemática são erosão e dilatação. Eles calculam basicamente o menor e o maior elemento (respectivamente) dentro da janela. Max-pooling, muito usado en redes neurais convolucionais, é uma dilatação.

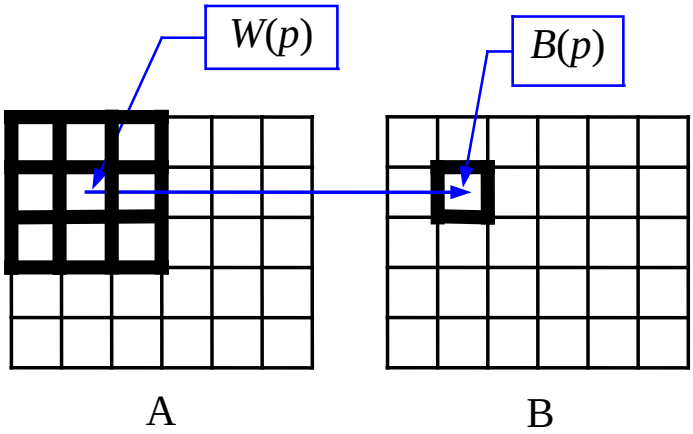


Figura 1: Filtro.

Em Morfologia Matemática, costuma-se usar o termo “operador” no lugar de “filtro”. Além disso, costuma-se usar o termo “elemento estruturante” no lugar de “janela”. Um elemento estruturante é representado em OpenCV (e em muitas outras bibliotecas) como uma imagem W onde os pixels não-nulos de W fazem parte do elemento estruturante e os pixels zero não fazem parte.

O valor de saída no pixel p da erosão da imagem A por um elemento estruturante W é definido como o menor valor de A dentro de W transladado para p:

$$(A \ominus W)(p) = \min_{q:W(q)\neq 0} \left[A(p+q)\right]$$

onde $p+q$ indica soma vetorial.

O valor de saída no pixel p da dilatação da imagem A por um elemento estruturante W é definido como o maior valor de A dentro de W rotacionado por 180 graus e transladado para p:

$$(A \oplus W)(p) = \max_{q:\hat{W}(q)\neq 0} \left[A(p+q)\right]$$

onde $\hat{W}$ indica a imagem W rotacionada por 180 graus (ou reflexão horizontal e vertical).

Nota: Maioria dos autores define a dilatação com rotação de 180 graus do elemento estruturante [wikiMorpho, Gonzalez2002]. Porém, a implementação de OpenCV não efetua esta rotação. Muitas vezes, o elemento estruturante é invariante por rotação de 180 graus (retângulo, círculo, elipse, cruz, etc). Evidentemente, nestes casos, não faz diferença efetuar ou não rotação de 180 graus.

1.1 Exemplo numérico de erosão e dilatação para imagens binárias

7 5 0 0 0 0 0 0 0 255 0 0 0 255 255 255 0 0 255 255 255 0 0 0 255 255 0 0 255 255 255 0 0 0 0 0 0	3 3 0 255 0 0 255 255 0 0 0 3 3 0 0 0 255 255 0 0 255 0	7 5 0 0 0 0 0 0 0 0 0 0 0 0 255 0 0 0 255 255 0 0 0 0 255 0 0 0 0 255 0 0 0 0 0 0 0	7 5 0 0 255 0 0 0 255 255 255 0 0 255 255 255 255 0 255 255 255 255 0 255 255 255 255 0 255 255 255 255 0 0 0 0 0
(a) Imagem *A*.	(b) Elemento estruturante *W* (superior) e após rotação de 180 graus (inferior).	(c) $(A \ominus W)$	(d) $(A \oplus W)$

Figura 2: Exemplo de erosão e dilatação para imagens binárias.

Considere a imagem binária *A* da figura 2a, com 7 linhas e 5 colunas. Vamos fazer erosão e dilatação dessa imagem por um elemento estruturante *W* em forma de "L" (figura 2b-superior), com os pixels não-nulos marcados em amarelo.

Para calcular a erosão $(A \ominus W)$, deve-se procurar o menor elemento transladando o elemento estruturante *W* para cada pixel de *A*. Por exemplo, quando a janela *W* está nas duas posições marcadas em amarelo (figura 2a), os menores elementos de *A* dentro das duas janelas são 0 e 255. Assim, na imagem de saída (figura 2c), essas posições recebem valores 0 e 255.

Para calcular a dilatação $(A \oplus W)$, primeiro deve-se rotacionar o elemento estruturante *W* de 180 graus. A figura 2b-inferior mostra $\hat{W}$, o resultado dessa rotação, com os pixels não-nulos em ciano. Agora, deve-se transladar $\hat{W}$ para cada pixel de *A* e calcular o maior elemento dentro da janela. Por exemplo, quando a janela $\hat{W}$ está nas duas posições marcadas em ciano (figura 2a), os maiores elementos dentro das duas janelas são 0 e 255. Assim, na imagem de saída (figura 2d), essas posições recebem valores 0 e 255.

Para que não haja "invasão" dos pixels fora do domínio de imagem, ao calcular erosão deve-se considerar que todos pixels fora do domínio da imagem *A* possuem o maior valor possível (255). Da mesma forma, ao calcular dilatação, deve-se considerar que todos os pixels fora do domínio da imagem *A* possuem o menor valor possível (zero).

1.2 Exemplo numérico de erosão e dilatação para imagens em níveis de cinza

3 3 34 56 94 28 19 69 . . 96 72 18	1 3 1 1 0 1 3 0 1 1	3 3 34 34 56 28 19 19 96 72 18	3 3 34 56 94 28 28 69 96 96 72	3 3 56 94 94 28 69 69 96 72 18
(a) Imagem *A*.	(b) Elemento estruturante *W* (superior) e após rotação de 180 graus (inferior).	(c) $(A \ominus W)$	(d) Dilatação sem fazer rotação 180 graus de W.	(e) Dilatação $(A \oplus W)$ fazendo rotação 180 graus de W.

Figura 3: Exemplo de erosão e dilatação para imagens em níveis de cinza.

Considere a imagem *A* da figura 3a, com 3 linhas e 3 colunas. Vamos fazer erosão e dilatação dessa imagem por um elemento estruturante *W* (figura 3b-superior).

Para calcular a erosão $(A \ominus W)$, deve-se procurar o menor elemento transladando o elemento estruturante *W* para cada pixel de *A*. O pixel zero de *W* não faz parte do elemento estruturante. Assim, quando a janela *W* está nas posições coloridas da imagem 2a, o menor elemento de *A* dentro da janela amarela é 34, verde é 19 e ciano é 96 (pois 72 está fora da janela). Assim, os valores desses pixels na imagem de saída (figura 3c) recebem os valores 34, 19 e 96.

A rotina de OpenCV calcula dilatação sem fazer rotação de 180 graus do elemento estruturante (figura 3b-superior). Calculando os maiores elementos das janelas coloridas (o pixel da direita não faz parte da janela), obtemos 56 no amarelo, 69 no verde e 96 no ciano. São os valores desses pixels na imagem de saída da figura 3d.

A maioria dos autores calcula dilatação $(A \oplus W)$ fazendo rotação de 180 graus do elemento estruturante (figura 3b-inferior). Os maiores elementos das janelas coloridas da figura 3a são (o pixel da esquerda não faz parte da janela): 94 no amarelo, 69 no verde e 96 no ciano. São esses os valores desses pixels na imagem de saída da figura 3e.

1.3 Sobrecarga de operadores de erosão e dilatação

Como vimos, em Morfologia Matemática, erosão e dilatação são usualmente denotadas por $\ominus$ e $\oplus$, e a rotação 180 graus por "^". Para que possamos escrever expressões semelhantes em C++, vamos "sobrecarregar" (overload) os operadores:

```
+     ==> dilatação
-     ==> erosão (operador com 2 operandos. Ex: a-b)
~     ==> rotação de 180 graus.
```

Nota: Não é possível usar ^ para denotar rotação 180 graus, pois ^ é um operador binário em C++ (com 2 operandos, tipo a^b), enquanto que a rotação 180 graus deve ser um operador unário (com 1 operando, como ~a).

Além disso, vamos "consertar" dilatação de OpenCV, para que calcule rotação de 180 graus do elemento estruturante. Os operadores abaixo fazem isso:

```
Mat_<GRY> operator~(Mat_<GRY> a) //rotacao de 180 graus
  { Mat_<GRY> d; flip(a,d,-1); return d; }

Mat_<GRY> operator+(Mat_<GRY> a, Mat_<GRY> b) //dilatacao
  { Mat_<GRY> d; dilate(a,d,~b); return d; }

Mat_<GRY> operator-(Mat_<GRY> a, Mat_<GRY> b) //erosao
  { Mat_<GRY> d; erode(a,d,b); return d; }
```

Algoritmo 1: Operadores "~", "+" e "-".

Nota: Tentei fazer a mesma coisa em Python e não descobri uma forma simples de fazê-lo. Se alguém souber como fazer, avise-me. Acho que a forma mais simples seria escrever "d=cv2.erode(a,b)", "d=cv2.flip(a,-1)", etc.

1.4 Exemplo de erosão e dilatação para imagens binárias

Incluindo esses operadores no programa, podemos testar erosão e dilatação:

```
1  // erodilb1.cpp pos2020
2  #include <cekeikon.h>
3
4  Mat_<GRY> operator~(Mat_<GRY> a) //rotacao de 180 graus
5    { Mat_<GRY> d; flip(a,d,-1); return d; }
6
7  Mat_<GRY> operator+(Mat_<GRY> a, Mat_<GRY> b) //dilatacao
8    { Mat_<GRY> d; dilate(a,d,~b); return d; }
9
10 Mat_<GRY> operator-(Mat_<GRY> a, Mat_<GRY> b) //erosao
11   { Mat_<GRY> d; erode(a,d,b); return d; }
12
13 int main() {
14   Mat_<GRY> e33(3,3, 255);
15   Mat_<GRY> ecruz=
16     (Mat_<GRY>(3,3) <<   0,255,  0,
17                        255,255,255,
18                          0,255,  0);
19   Mat_<GRY> a; le(a,"letram.bmp");
20   imp(a-e33,   "ero33.bmp");
21   imp(a-ecruz, "ero_cruz.bmp");
22   imp(a+e33,   "dil33.bmp");
23   imp(a+ecruz, "dil_cruz.bmp");
24 }
25
```

Algoritmo 2: Erosão e dilatação para imagens binárias.

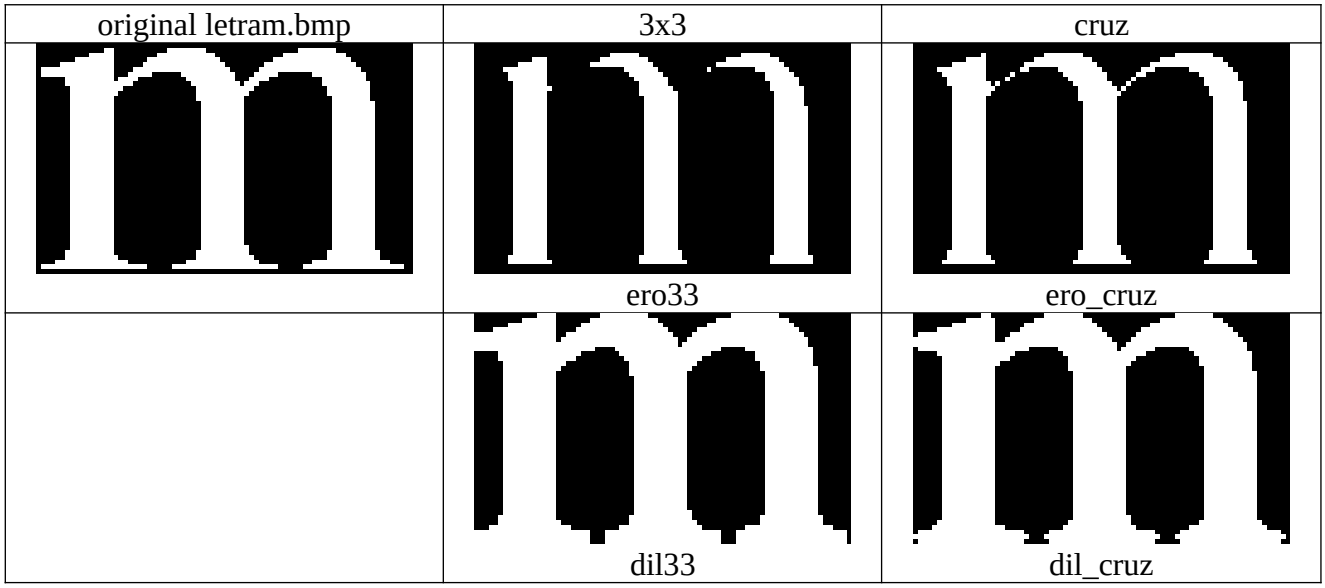


Figura 4: Erosão e dilatação para imagens binárias

A figura 4 mostra o resultado de aplicação de erosão e dilatação na imagem “letram.bmp”. Quando o objeto é branco e o fundo é preto, erosão e dilatação fazem o que se espera dos seus nomes (erosão emagrece e dilatação engorda o objeto). Porém, quando o objeto é preto sobre fundo branco, erosão irá engordar o objeto e dilatação irá emagrecer o objeto, contrário ao que os seus nomes indicam.

Na linha 15, estamos criando o elemento estruturante 3x3 quadrado. Na linha 16, criamos o elemento estruturante 3x3 em forma de cruz.

Nas linhas 21-24, aplicamos erosão e dilatação usando esses dois elementos estruturantes e imprimimos os resultados.

Na atual biblioteca Cekeikon, já estão implementados os operadores utilizados na Morfologia Matemática. Assim, o algoritmo 3 pode ser reescrito utilizando os operadores prontos da biblioteca, bastando escrever “using namespace Morphology”.

```
// erodilb.cpp pos2020
#include <cekeikon.h>
int main() {
  using namespace Morphology;
  Mat_<GRY> e33(3,3, 255);
  Mat_<GRY> ecruz=
    (Mat_<GRY>(3,3) <<   0,255,  0,
                       255,255,255,
                         0,255,  0);
  Mat_<GRY> a; le(a,"letram.bmp");
  imp(a-e33,   "ero33.bmp");
  imp(a-ecruz, "ero_cruz.bmp");
  imp(a+e33,   "dil33.bmp");
  imp(a+ecruz, "dil_cruz.bmp");
}
```

Algoritmo 3: Erosão e dilatação para imagens binárias, usando os operadores da biblioteca Cekeikon.

Nota: OpenCV usa operadores "+" e "-" entre duas imagens para fazer soma e subtração pixel a pixel. Escrevendo "using namespace Morphology", o significado desses operadores muda, passando a efetuar dilatação e erosão. Se você quiser usar os operadores “+” ou “-” do OpenCV (dentro do escopo de “using namespace Morphology”), pode escrever por exemplo:

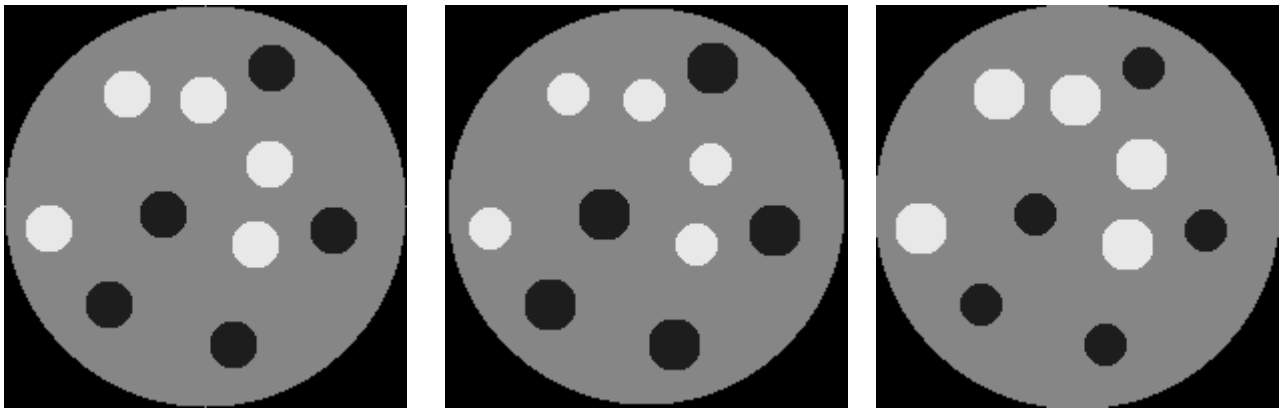
```
{ using cv::operator-;
  b = a-b;
}
```

A operação “a-b” será subtração pixel a pixel.

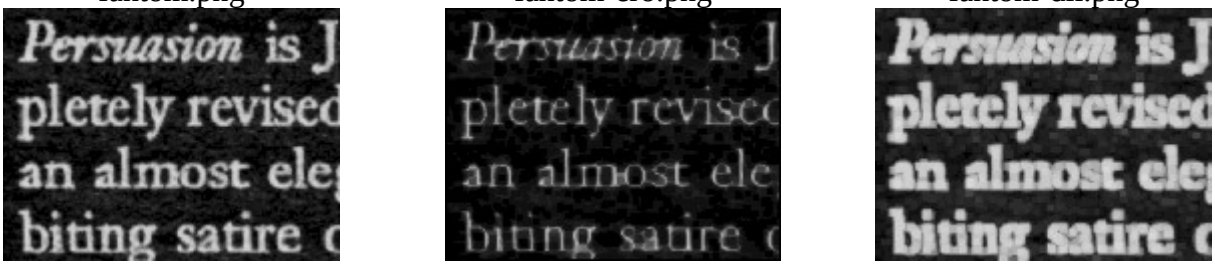
1.5 Exemplo de erosão e dilatação para imagens em níveis de cinza

```
// erodilg.cpp pos2020
#include <cekeikon.h>
using namespace Morphology;

int main() {
  Mat_<GRY> ee(3,3,255);

  Mat_<GRY> a; le(a,"fantom.png");
  imp(a-ee, "fantom-ero.png");
  imp(a+ee, "fantom-dil.png");

  le(a,"pers.png");
  imp(a-ee, "pers-ero.png");
  imp(a+ee, "pers-dil.png");
}
```

Algoritmo 4: Erosão e dilatação para imagens em níveis de cinza.



fantom.png fantom-ero.png fantom-dil.png



pers.png pers-ero.png pers-dil.png

Figura 5: Erosão e dilatação 3x3 de imagens em níveis de cinza.

Repare no “fantom-ero.png” que erosão diminuiu círculos claros (objeto claro sobre fundo escuro) mas aumentou círculos escuros (objeto escuro sobre fundo claro).

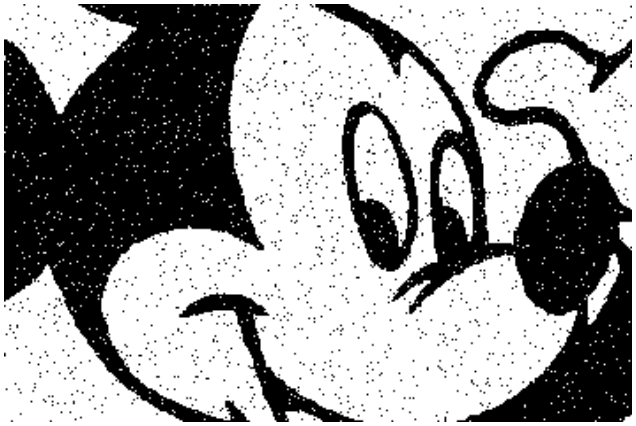
2. Abertura e fechamento

2.1 Abertura e fechamento para imagens binárias

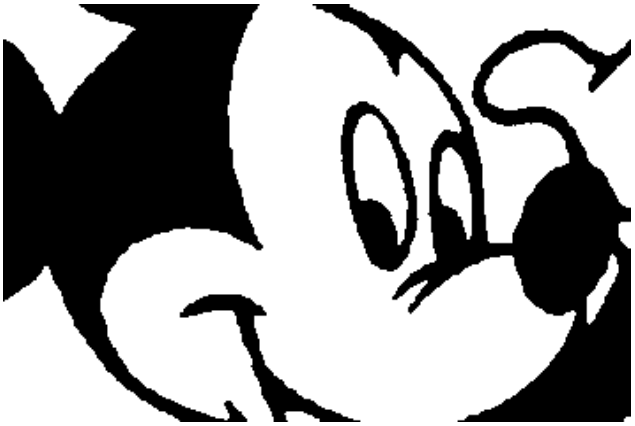
Abertura é erosão seguido por dilatação, usando o mesmo elemento estruturante nas duas operações. Fechamento é dilatação seguido por erosão, usando o mesmo elemento estruturante em ambas operações.

```
 1 //mickeyr.cpp 2015
 2 #include <cekeikon.h>
 3 using namespace Morphology;
 4 int main()
 5 { Mat_<GRY> a;
 6   le(a,"mickeyr.bmp");
 7   Mat_<GRY> ee(2,2,255);
 8   Mat_<GRY> abert=a-ee+ee;
 9   imp(abert,"mickeyabert.bmp");
10   Mat_<GRY> fech=a+ee-ee;
11   imp(fech,"mickeyfech.bmp");
12 }
```

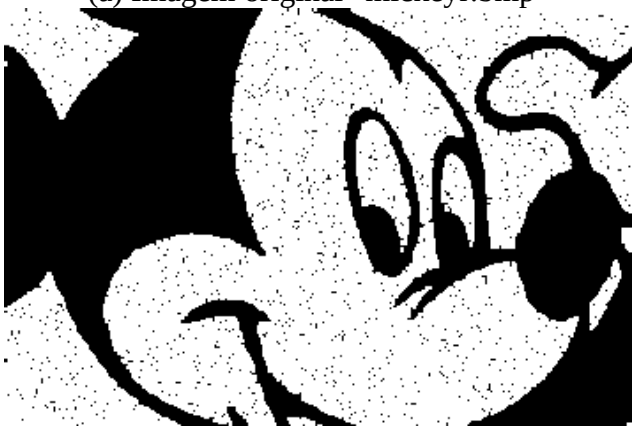
Algoritmo 5: Abertura e fechamento para imagens binárias.



(a) Imagem original “mickeyr.bmp”



(b) Filtro mediano 3x3



(c) Abertura 2x2

(d) Fechamento 2x2

Figura 6: Abertura e fechamento para imagens binárias.

O algoritmo 5 é implementação de abertura e fechamento usando elemento estruturante 2x2. A figura 6 mostra a entrada e as saídas desse programa, aplicando abertura e fechamento a uma imagem ruidosa. A figura 6b mostra o filtro mediano 3x3 aplicado na imagem original para comparação.

Aplicando abertura (linha 8) numa imagem ruidosa (figura 6a), obtemos imagem de saída (figura 6c) onde ruído tipo “sal” foi eliminado mas o ruído tipo “pimenta” foi preservado. Abertura é erosão seguida de dilatação. Quando o programa efetua erosão, o ruído tipo “sal” desaparece, enquanto que o ruído tipo “pimenta” fica maior. Quando o programa efetua dilatação (na imagem que sofreu erosão), ruído tipo “pimenta” diminui de tamanho e a maioria volta ao tamanho e posição originais, enquanto que não tem como o ruído tipo “sal” voltar a aparecer.

A mesma lógica funciona para o fechamento (figura 6d).

Por que abertura e fechamento recebem estes nomes? Veja a figura 7, onde podemos interpretar preto como água e branco como terra. Aplicando fechamento, lagos e baias foram “fechados” pela terra. Aplicando abertura, ilhas, penínsulas e lagos foram “abertas” para o mar.

Pergunta: O que fazer para eliminar tanto ruídos “sal” como ruídos “pimenta”?

Imagem original "lago"

Abertura

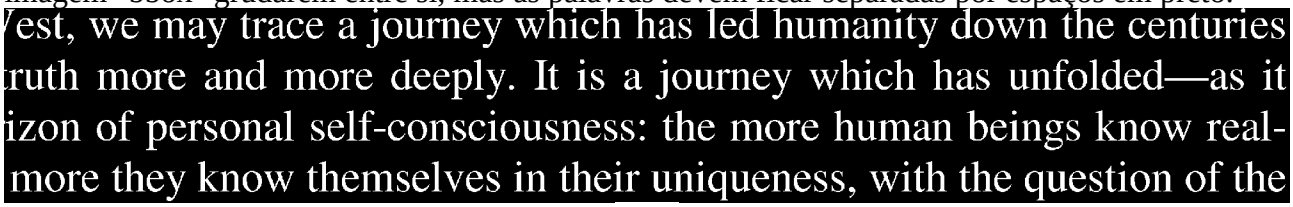
Fechamento

Figura 7: Abertura e fechamento por elemento estruturante 15x15.

```
#include <cekeikon.h>
using namespace Morphology;

int main()
{ Mat_<GRY> a;
  le(a,"lago.bmp");
  Mat_<GRY> ee(15,15,255);
  Mat_<GRY> abert=a-ee+ee;
  imp(abert,"labert.bmp");
  Mat_<GRY> fech=a+ee-ee;
  imp(fech,"lfech.bmp");
}
```

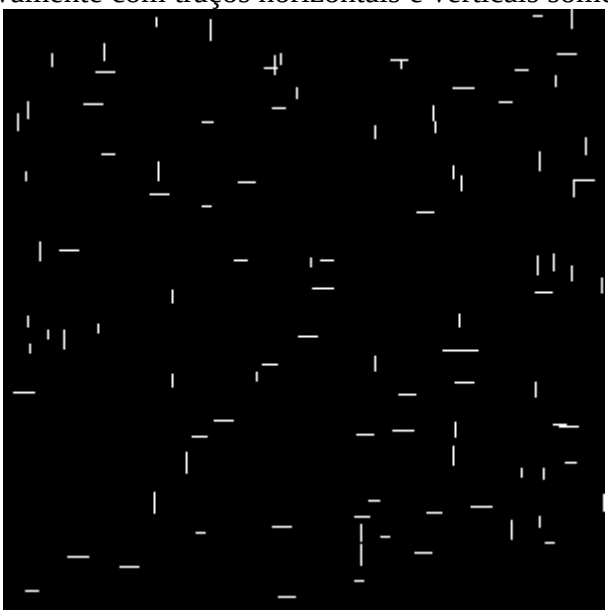
Algoritmo 6: Abertura e fechamento por elemento estruturante 15x15.

Exercício: Faça um programa com abertura/fechamento que faz as letras de uma palavra da imagem "bbox" grudarem entre si, mas as palavras devem ficar separadas por espaços em preto.



bbox

Exercício: A imagem horver.png contém traços horizontais e verticais. Escreva um programa que lê horver.png e separa os traços horizontais dos verticais usando morfologia matemática, gerando hor.png e ver.png respectivamente com traços horizontais e verticais somente.



horver.png

A função dilate do OpenCV calcula o maior valor dentro do elemento estruturante sem rotacionar o elemento estruturante. O operator+ que está definido no namespace Morphology rotaciona o elemento estruturante por 180 graus antes de calcular o máximo. Para ver a diferença, considere abertura por elemento estruturante em forma de "L":

```
#include <cekeikon.h>
using namespace Morphology;
int main()
{ Mat_<GRY> a;
  le(a,"mickeyr.bmp");
  Mat_<GRY> ee = (Mat_<GRY>(2,2) << 255,   0,
                                    255, 255);
  Mat_<GRY> abert=a-ee+ee;
  imp(abert,"abert1.bmp");
  Mat_<GRY> erosao;
  erode(a, erosao, ee);
  Mat_<GRY> abert2;
  dilate(erosao, abert2, ee);
  imp(abert2,"abert2.bmp");
}
```

Algoritmo 7: Diferença entre rotacionar ou não o elemento estruturante na abertura.

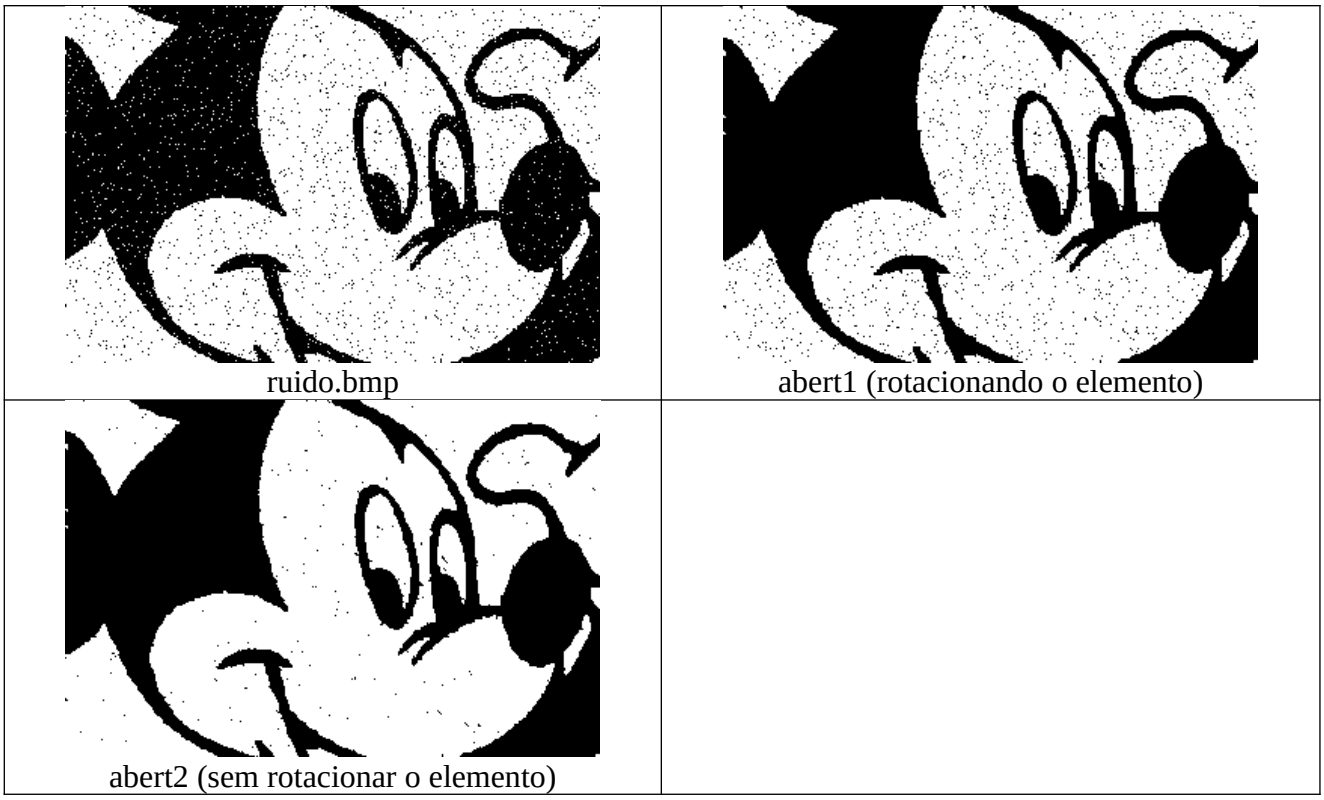


Figura 8: Diferença entre rotacionar ou não o elemento estruturante na abertura.

Abert1 foi gerado usando abertura com dilatação rotacionando o elemento estruturante 180 graus. Ruídos tipo "sal" foram eliminados sem alterar (muito) os ruídos tipo "pimenta". Emquanto isso, abert2 foi gerado usando abertura com dilatação do OpenCV (sem rotacionar o elemento estruturante). Repare que os ruídos tipo "pimenta" ficaram alterados (além de eliminar ruídos tipo "sal").

2.2 Abertura e fechamento para imagens em níveis de cinza

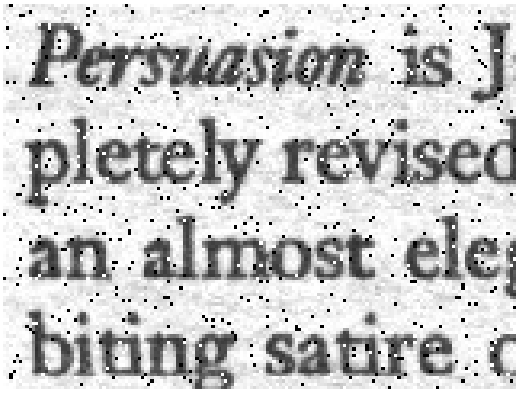
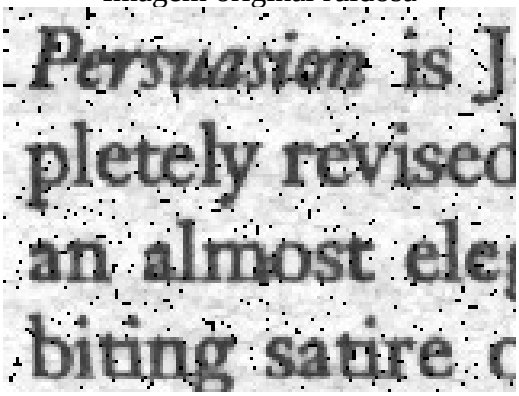
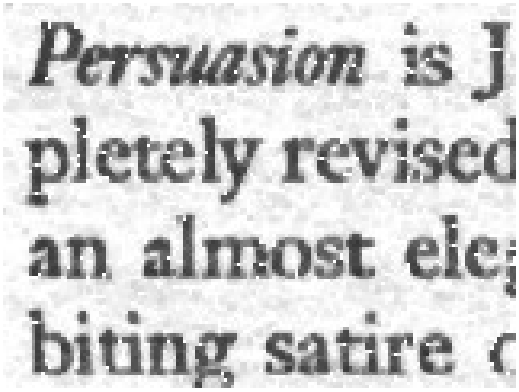


Imagem original ruidosa



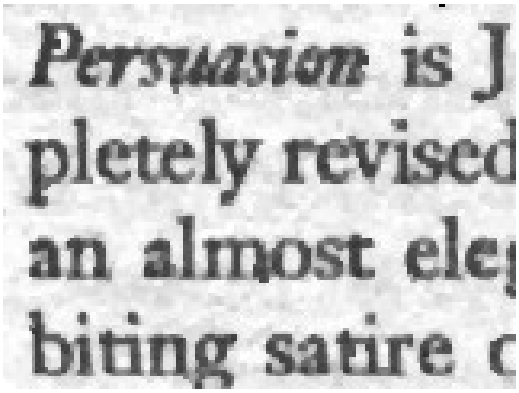
Abertura 2×2



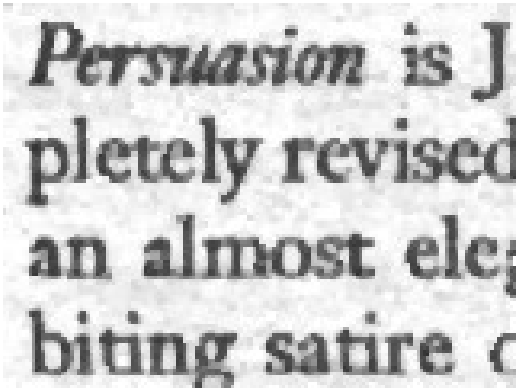
Fechamento 2×2

Figura 9: Abertura e fechamento numa imagem em níveis de cinzas.

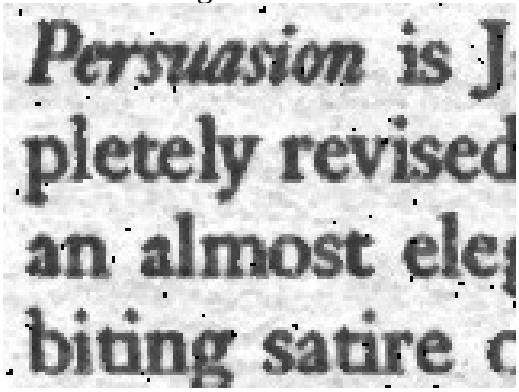
Abertura e fechamento funciona de forma semelhante para imagem em níveis de cinzas (figura 9), eliminando somente ruído tipo “sal” ou “pimenta”. Evidentemente, se aplicar abertura seguida de fechamento ou fechamento seguido de abertura, os dois tipos de ruído serão eliminados. A figura 10 ilustra isso. As saídas dos filtros mediana estão mostradas para comparação.



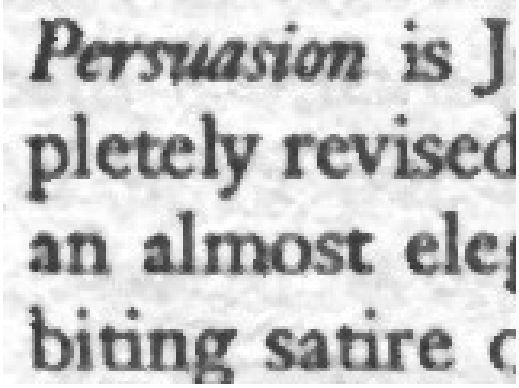
Abertura 2x2 seguida de fechamento 2x2.



Fechamento 2x2 seguido por abertura 2x2.



Mediana 2×2



Mediana 3×3

Figura 10: Abertura seguida de fechamento ou fechamento seguido de abertura elimina os dois tipos de ruídos. As saídas dos filtros mediana estão mostradas para comparação.

Exercício: Faça um programa que lê a imagem lennarui.pgm e elimina os ruídos da melhor forma possível, usando as operações morfológicas (erosão, dilatação, abertura e fechamento). A qualidade do filtro deve ser medida usando RMSE (root of mean square error). O programa do Cekeikon "kcek distg img1.pgm img2.pgm" calcula RMSE entre img1 e img2.

lennarui.pgm

3. Reconstrução morfológica

3.1 Bordas de imagens binárias

Vamos utilizar mais alguns operadores de “namespace Morphology”:

```
-     ==> imagem negativa (operador unário. Ex: -a).
&&    ==> minimo (intersecção ou and)
||    ==> maximo (união ou or)
^     ==> exclusive-or (xor para imagens binárias)
==    ==> verifica se duas imagens são iguais
!=    ==> verifica se duas imagens são diferentes
```

A implementação de alguns deles:

```
Mat_<GRY> operator-(Mat_<GRY> a)
{ Mat_<GRY> d; d=255-a; return d; }

Mat_<GRY> operator&&(Mat_<GRY> a, Mat_<GRY> b)
{ Mat_<GRY> d=min(a,b); return d; }

Mat_<GRY> operator||(Mat_<GRY> a, Mat_<GRY> b)
{ Mat_<GRY> d=max(a,b); return d; }

Mat_<GRY> operator^(Mat_<GRY> a, Mat_<GRY> b)
{ Mat_<GRY> d; bitwise_xor(a,b,d); return d; }
```

Nota: A implementação ^ acima só funciona para imagens binárias. Para que xor faça algo que faça sentido em imagens em níveis de cinza, possivelmente a implementação deve ser:

```
Mat_<GRY> operator^(Mat_<GRY> a, Mat_<GRY> b)
{ Mat_<GRY> d;
  { using cv::operator-;
    d = abs(a-b);
  }
  return d;
}
```

Com isso, podemos calcular as bordas de imagens binárias, como no exemplo abaixo (figura 11). É possível calcular bordas internas usando erosão e bordas externas usando dilatação.

Exercício: Escreva um programa que lê “letram.bmp” e gera as quatro imagens com as bordas como na figura 11.

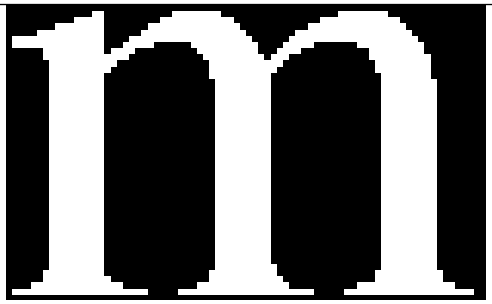
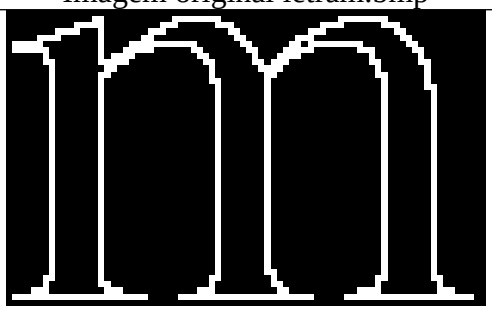
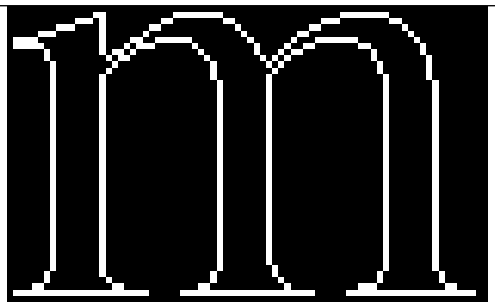


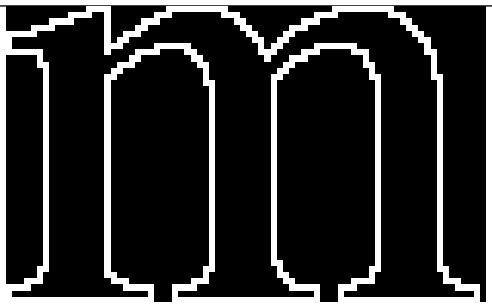
Imagem original letram.bmp



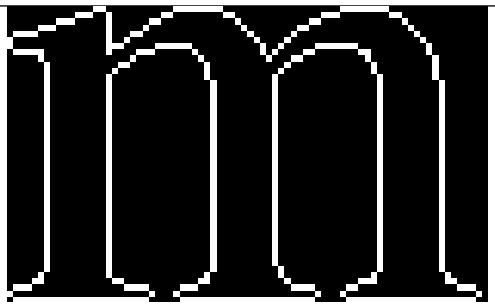
Borda interna - ee 3x3



Borda interna - ee cruz



borda externa - ee 3x3

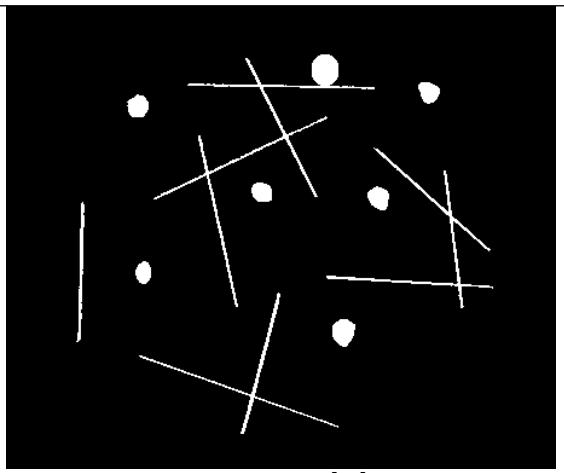


Borda externa - ee cruz

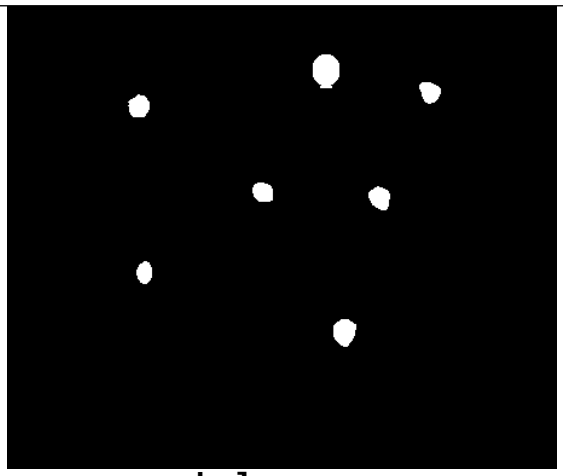
Figura 11: Bordas internas e externas usando erosão, dilatação e xor.

3.2 Traços finos

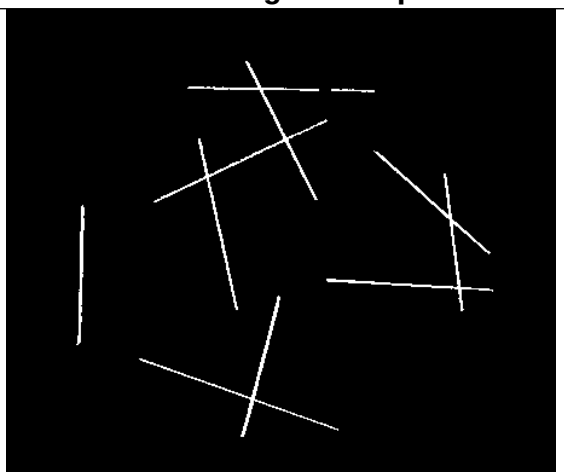
Exercício: Faça um programa que separa bolas e traços da imagem traco-grud.bmp. Use erosão -, dilatação +, máximo ||, mínimo && e ou-exclusivo ^.



traco-grud.bmp



bolas.pgm



tracos.pgm

Figura 12: Traços finos e bolas (alguns grudados às bolas).

Dica: Como podemos resolver este problema? Uma solução é descobrir uma erosão que consiga eliminar os traços mas somente diminuir o tamanho das bolas sem eliminá-las. Depois, se fizer dilatação usando o mesmo elemento estruturante, as bolas voltarão a ter o tamanho próximo do original.

3.3 Traços grossos

A imagem binária trabold.bmp consiste de traços grossos e bolas, sem que haja contato entre os traços e as bolas. Faça um programa baseado em morfologia matemática que elimina bolas, mantendo todos os traços intactos.

```
 1 //trabold.cpp pos2016
 2 #include <cekeikon.h>
 3 int main()
 4 { using namespace Morphology;
 5   Mat_<GRY> a; le(a,"trabold.bmp");
 6   Mat_<GRY> e1(1,31,255); // horizontal
 7   Mat_<GRY> e2(31,1,255); // vertical
 8   Mat_<GRY> e3(31,31,GRY(0));
 9   for (int i=0; i<31; i++) e3(i,i)=255; // diag principal
10   Mat_<GRY> e4(31,31,GRY(0));
11   for (int i=0; i<31; i++) e4(31-i,i)=255; // diag secundario
12   Mat_<GRY> b1=a-e1+e1; imp(b1,"b1.bmp");
13   Mat_<GRY> b2=a-e2+e2; imp(b2,"b2.bmp");
14   Mat_<GRY> b3=a-e3+e3; imp(b3,"b3.bmp");
15   Mat_<GRY> b4=a-e4+e4; imp(b4,"b4.bmp");
16
17   Mat_<GRY> c = b1 || b2 || b3 || b4; imp(c,"c.bmp");
18
19   Mat_<GRY> t; Mat_<GRY> e33(3,3,255);
20   do {
21     t=c.clone(); // t=c errado
22     c=(c+e33) && a;
23   } while (t!=c);
24   imp(c,"rec.bmp");
25   imp(c^a,"d.bmp");
26 }
```

Algoritmo 8: Traços grossos e bolas.

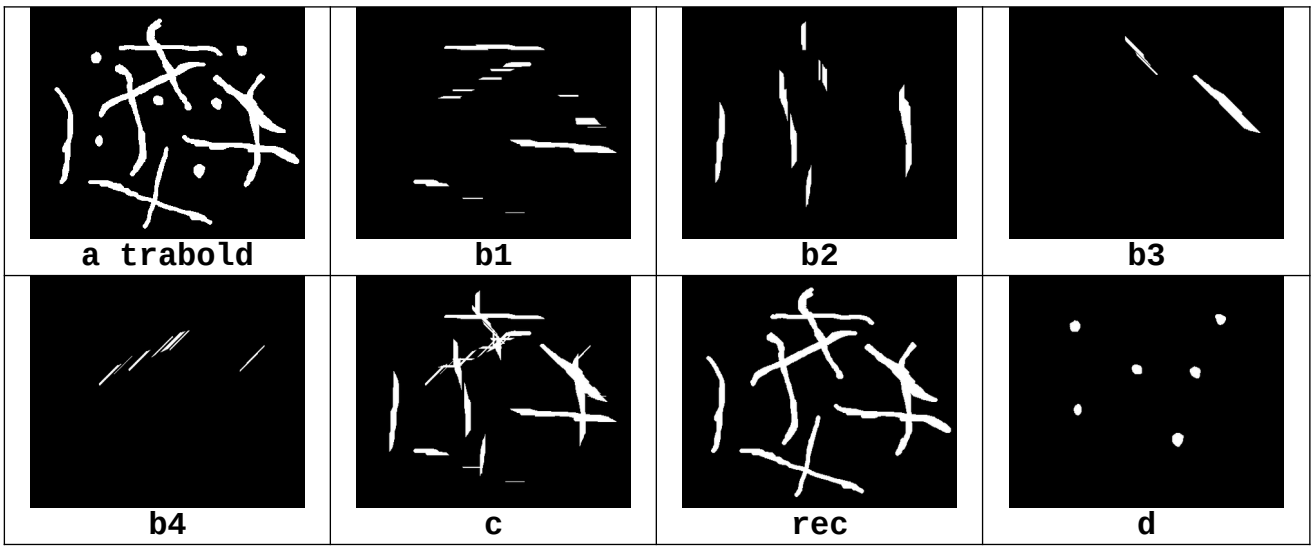


Figura 13: Traços grossos e bolas.

Aqui, não existe uma abertura capaz de eliminar os traços mantendo as bolas, pois há traços tão grossos quanto o diâmetro das bolas. Vamos usar uma ideia diferente. Vamos aplicar aberturas com elementos estruturantes compridos em diferentes ângulos, mais compridos do que o diâmetro de qualquer uma das bolas. Com isso, as aberturas irão eliminar todas as bolas, mas não conseguirão eliminar os traços com ângulo de inclinação semelhante à inclinação do elemento estruturante.

As linhas 6-11 criam quatro elementos estruturantes com 31 pixels de comprimento nas direções horizontal, vertical, diagonal principal e diagonal secundário. Fazendo abertura com eles, obtém as figuras *b1*, *b2*, *b3* e *b4* da figura acima. Repare que as bolas sumiram em todas essas imagens, mas algumas partes dos traços permaneceram. Calculando o máximo entre as saídas *b1*, *b2, b3* e *b4*, obtemos a imagem *c*. Se tivéssemos usado mais direções, a imagem *c* seria mais parecido com a imagem dos traços (sem bolas).

Vamos fazer crescimento de semente usando como sementes os pixels brancos da imagem *c*. Com isso conseguimos reconstruir os traços (imagem rec). Aqui, estamos supondo que os traços e as bolas não se tocam. Caso contrário, este processo não funcionaria, pois o crescimento de semente invadiria as bolas.

Podemos usar o crescimento de semente que vimos na aula de componentes conexos, baseado em fila ou pilha. Seria uma solução bem eficiente computacionalmente. Mas também é possível fazer um crescimento de semente (muito menos eficiente) usando as operações de Morfologia Matemática, chamada reconstrução morfológica [Vincent1993]. A vantagem da reconstrução morfológica é que ela pode ser utilizada para imagens em níveis de cinza, como veremos no próximo problema. A ideia é fazer uma dilatação 3x3 da semente, seguida por operação que elimina os pixels que saíram fora dos traços originais (calcula-se a operação de mínimo com a imagem original). Quando a imagem não se alterar mais fazendo dilatação seguida de mínimo, termina-se o programa.

3.4 Detecção de aneurismas

Qual seria a versão em níveis de cinzas do problema de separar traços grossos de bolas? A figura 14a mostra a imagem de fundo do olho com vasos sanguíneos e aneurismas. O problema é separar vasos sanguíneos (traços) das aneurismas (bolas) [Vincent1993].

```
 1 //aneurisma.cpp pos2016
 2 #include <cekeikon.h>
 3
 4 int main()
 5 { using namespace Morphology;
 6   Mat_<GRY> a; le(a,"an-ori.pgm");
 7
 8   Mat_<GRY> c(a.size(),0);
 9   for (double deg=0; deg<180.0; deg=deg+180.0/18.0) {
10     Mat_<GRY> e=strel(25,deg);
11     Mat_<GRY> b=a-e+e;
12     c = c || b;
13   }
14   imp(c,"an-c.pgm");
15
16   Mat_<GRY> t; Mat_<GRY> e33(3,3,255);
17   do {
18     t=c.clone(); // t=c errado
19     c=(c+e33) && a;
20   } while (t!=c);
21   imp(c,"an-rec.pgm");
22
23   { using cv::operator-;
24     Mat_<GRY> an = a - c;
25     imp(an,"an-an.pgm");
26   }
27 }
```

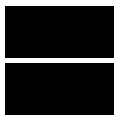
Algoritmo 9: Separar aneurismas de vasos sanguíneos.

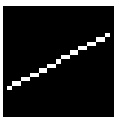
Aqui, a solução é praticamente idêntica à versão binária do problema.

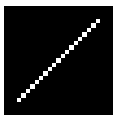
Em primeiro lugar, efetuam-se aberturas em diferentes direções usando elementos estruturantes longos e compridos maiores que o diâmetro de qualquer aneurisma. A função "strel(25,deg)" da linha 10 gera um elemento estruturante de comprimento 25 pixels com inclinação de *deg* graus. Estamos calculando 18 aberturas mudando ângulo a cada 10 graus. O máximo das 18 aberturas está na figura 14b. Veja que a imagem obtida já mostra quase perfeitamente os vasos sanguíneos, sem os aneurismas.

Função *strel* é de Cekeikon e não há equivalente em OpenCV. Foi implementada para ficar parecida com a mesma função em Matlab com opção *line*. Para que possam implementar a sua própria strel, mostro abaixo as saídas obtidas com diferentes parâmetros.

```
imp(strel(25, 0),"strel25_00.png");
imp(strel(25,30),"strel25_30.png");
imp(strel(25,45),"strel25_45.png");
```

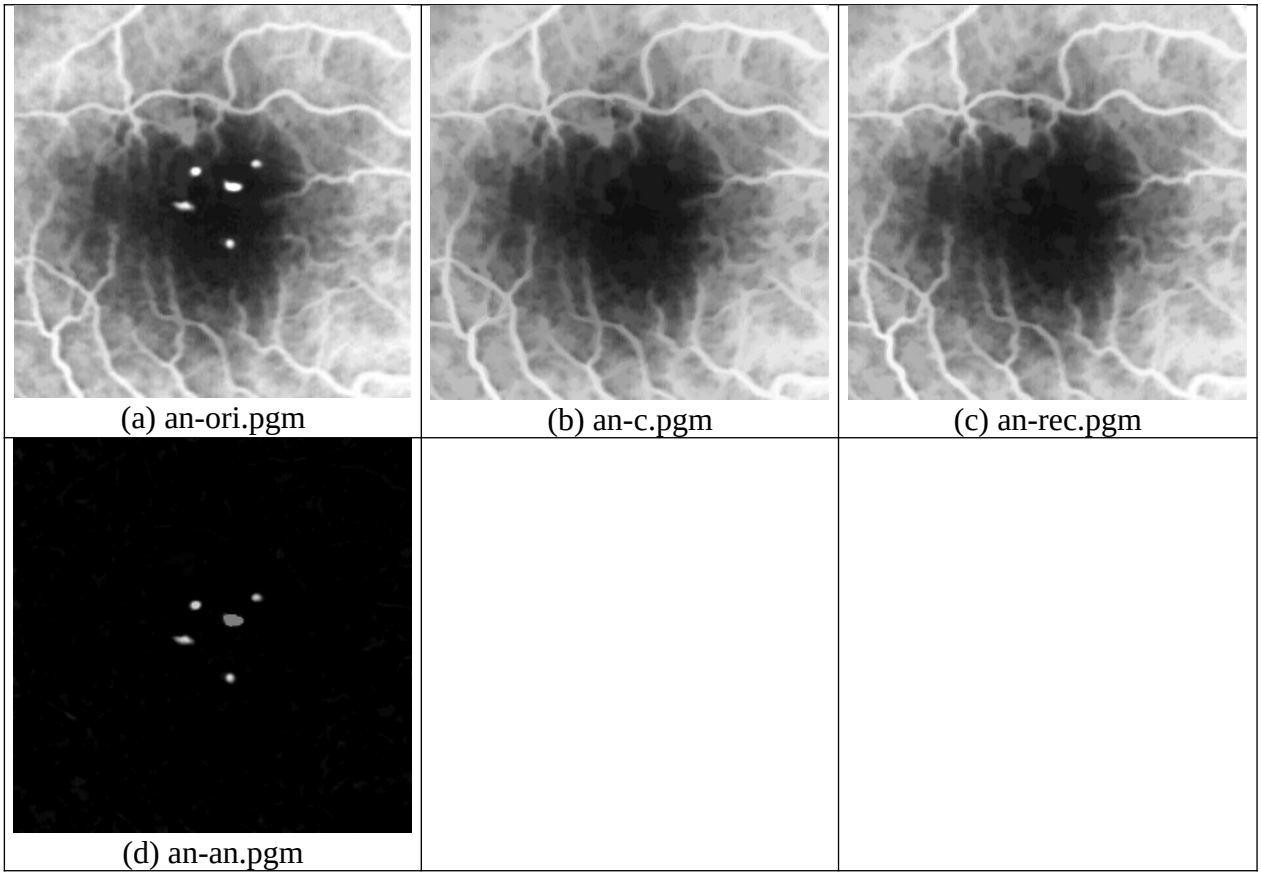

strel25_00


strel25_30


strel25_45

Para melhorar um pouco mais a qualidade dos vasos sanguíneos, efetua-se a reconstrução morfológica (espécie de crescimento de semente usando operações morfológicas), usando a imagem 14b como semente (linhas 16-21). Isto é, dilata-se a imagem 14b pelo elemento estruturante 3x3. Depois, calcula-se o mínimo com a imagem original. Repete-se o processo até que a imagem não se altere mais fazendo dilatação seguida pelo mínimo. O resultado desta operação está em figura 14c. A necessidade de fazer reconstrução morfológica pode não ser óbvio aqui, pois o resultado de máximo das aberturas já tem qualidade bastante boa. Porém, se você tivesse feito menos aberturas (por exemplo, somente 4 aberturas), haveria diferença bem grande entre a qualidade da imagem sem e com reconstrução morfológica.

Para obter os aneurismas, basta subtrair da imagem original a imagem obtida na figura 14d (linhas 23-26). “Using cv::operator-” indica que queremos fazer subtração pixel a pixel (operador "-" de OpenCV) e não erosão (operador “-” de “namespace Morphology”).



(a) an-ori.pgm (b) an-c.pgm (c) an-rec.pgm

(d) an-an.pgm

Figura 14: Separar aneurismas de vasos sanguíneos.

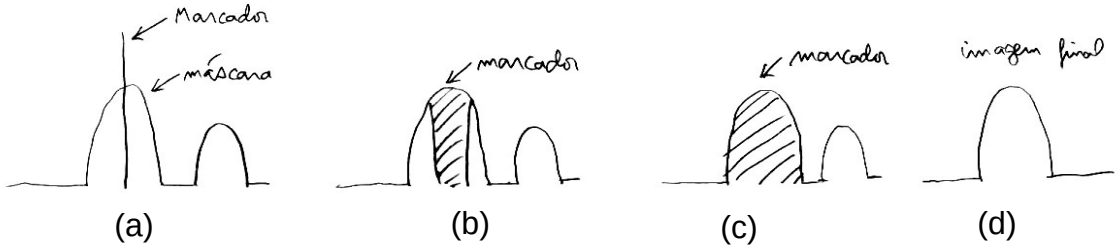
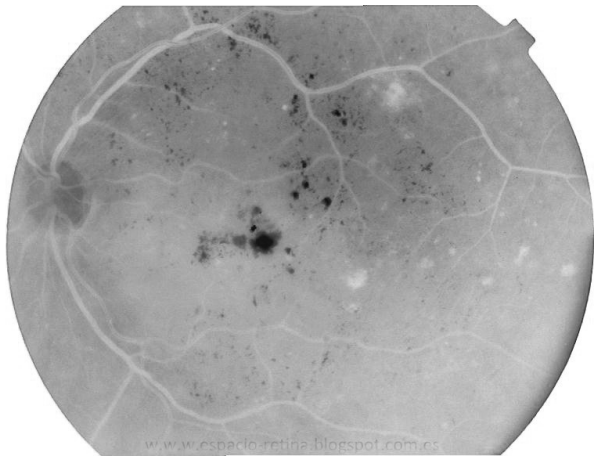


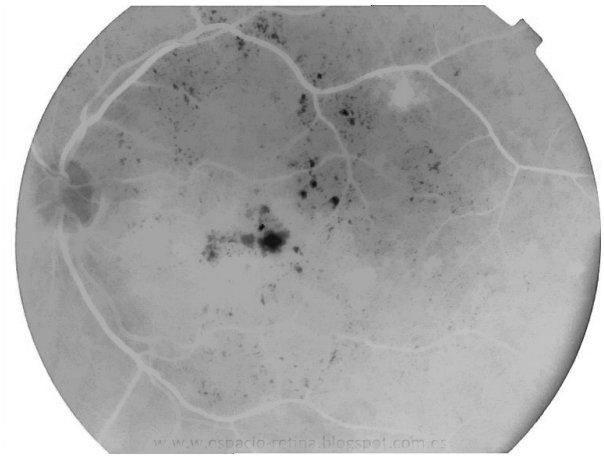
Figura 15: Reconstrução morfológica (crescimento de semente usando operações morfológicas).

A figura 15 ilustra a reconstrução morfológica. Em Morfologia, a semente é chamada de "marcador". A imagem onde a semente vai crescer é chamada de "máscara" (figura 15a). Efetua-se dilatação do marcador e depois calcula-se o mínimo com a máscara, obtendo figura 15b. Este processo repete-se até que a imagem não se altere com as operações de dilatação e mínimo (figura 15c). Neste momento, o marcador será o resultado de crescimento de semente (figura 15d).

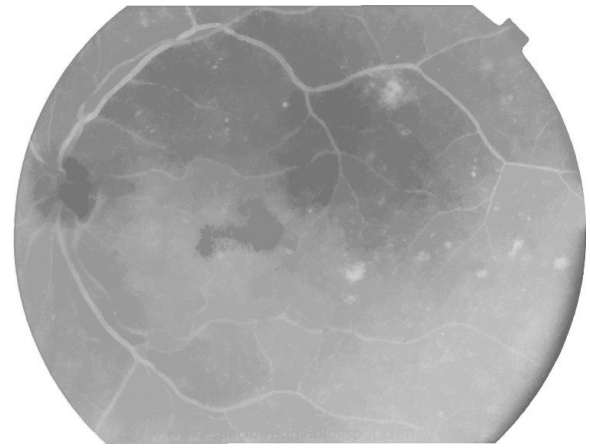
Exercício: Adapte o algoritmo 9 para ler a imagem fundoolhog.jpg, e gerar 2 imagens de saída: sem_mancha_clara.jpg e sem_mancha_escura.jpg. Procure apagar somente as manchas especificadas.



fundoolhog.jpg



sem_mancha_clara.jpg



sem_mancha_escura.jpg

Referências:

[WikiMorpho] https://en.wikipedia.org/wiki/Mathematical_morphology, acessado 27/03/2020.

[Gonzalez2002] R. C. Gonzalez, R. E. Woods, “Digital Image Processing, Second Edition,” Prentice-Hall, 2002.

[Vincent1993] L. Vincent, "Morphological Grayscale Reconstruction in Image Analysis: Applications and Efficient Algorithms," IEEE T. Image Processing, v. 2, n. 2, April 1993, pp. 176-201.