PROCESSAMENTO DE IMAGENS APLICADO A AGROINDUSTRIA

Pedro Luiz de Paula Filho

Objetivos

- Capturar vídeo
- Transformações Pontuais
  - Operações Algébricas em Imagens
  - Outras operações
- Transformações Locais
  - Filtros passa-baixa
  - Filtros passa-alta (realce de borda)
- Merge
- ROI

Capturar vídeo OpenCv

- Um vídeo nada mais é do que um conjunto de frames (imagens).
- Para capturar essas "imagens" deve-se fazer a conexão com o dispositivo de captura (webcam / vídeo gravado)
- Para tanto, usa-se a classe VideoCapture, existem três métodos construtores:
  - VideoCapture()
  - VideoCapture(const string& filename)
  - VideoCapture(int device)

Nome do vídeo (avi), ou sequência de imagens (img_00, img_01, ...)

Id do dispositivo de captura, 0→ camera 1, 1→camera 2, ...

Classe VideoCapture - OpenCv

- Métodos:
  - open → abre o dispositivo de captura
  - isOpened →retorna **true** se já está aberto
  - release → libera o dispositivo de captura
  - grab → pega o próximo frame do vídeo
  - retrive → decodifica e retorna o próximo frame
  - read → pega, decodifica e retorna o próximo frame
  - get → retorna as propriedades do VideoCapture
  - set → altera as propriedades do VideoCapture

http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture

```
int main(){
    int tecla;
    VideoCapture camera(0);
    VideoCapture video("robo.mpg");
    Mat frameCamera;
    Mat frameVideo;
    namedWindow("Captura", 1);
    namedWindow("Video", 1);

    do {
      camera.read(frameCamera);
      video.read(frameVideo);

      imshow("Captura", frameCamera);
      imshow("Video", frameVideo);
      tecla = waitKey(5);
    } while (tecla != 27);
 return 0;
}
```

Realce de Imagens

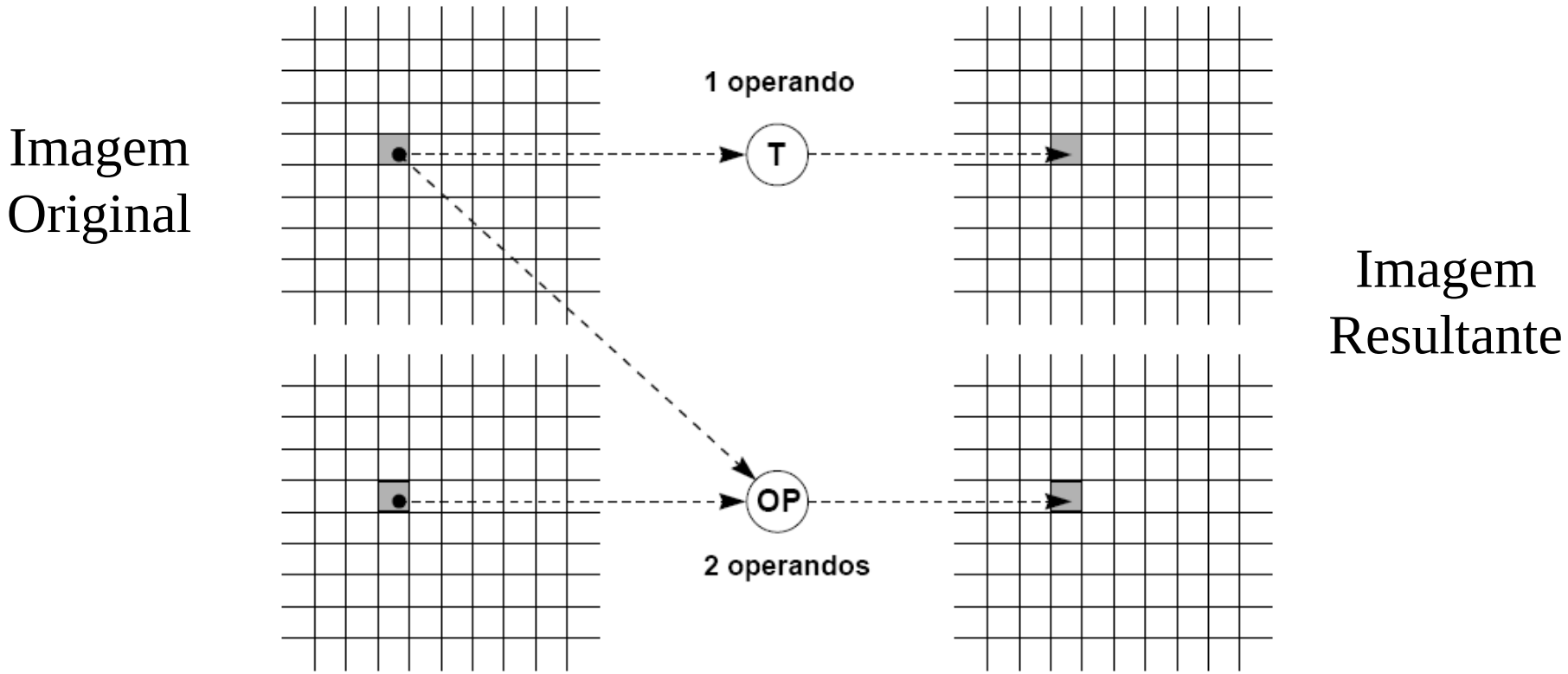
- Através do processamento de uma imagem busca obter um resultado mais apropriado para uma determinada aplicação.
- "Melhorar" uma imagem visualmente, baseado na resposta do sistema visual humano.
- Trabalha nos domínios :
  - Espacial (pixels)
  - Freqüência da imagem (transformadas)

Domínio Espacial

- Trabalha-se diretamente com os pixels da imagem
- Usa a intensidade do "pixel" ou de sua vizinhança no processamento
- Operações são realizadas:
  - Ponto a ponto (transformações pontuais)
  - Por regiões (transformações locais)

Transformações Pontuais

- Cada ponto na imagem de entrada gera um ponto na imagem de saída.



Transformações Pontuais

- Dada uma imagem A(x,y) que produz outra imagem B(x,y), pode-se definir matematicamente como:

$$B(x, y) = f[A(x, y)]$$

- Existem *n* formas de fazer transformações pontuais, identificadas pela função *f* ( )

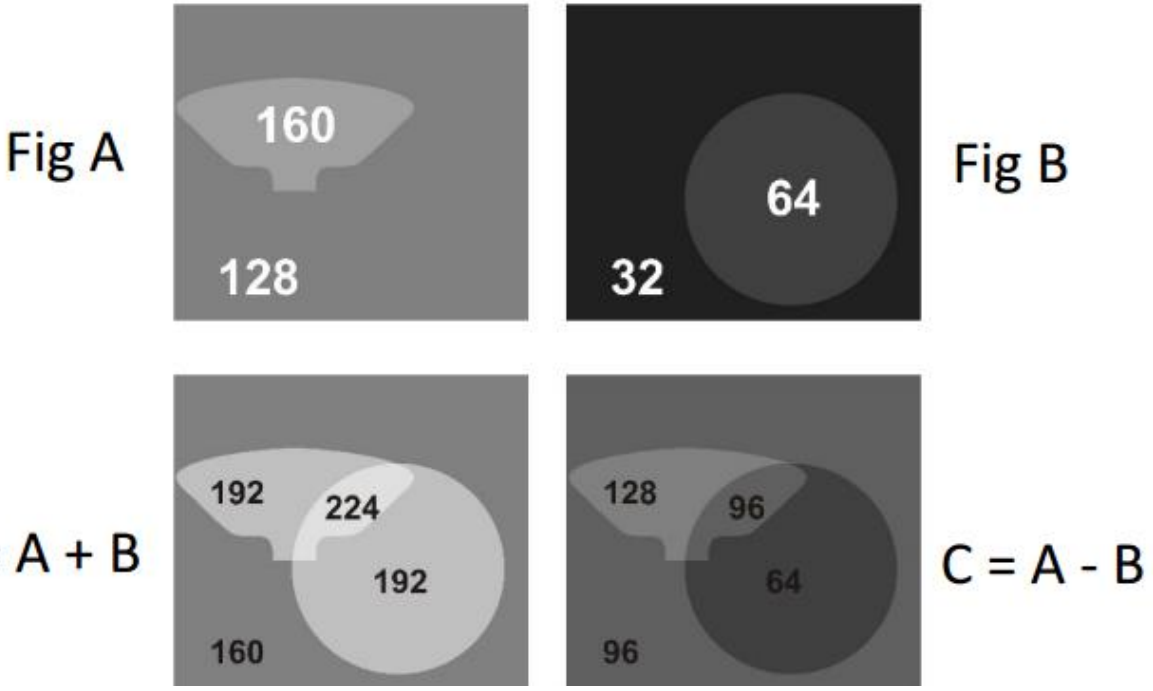
Operações Algébricas

- Pode-se fazer soma, subtração, multiplicação, divisão e combinação linear sobre imagens.
- As operações consistem em "combinar" duas imagens A e B de entrada, gerando uma terceira imagem C de saída.
- Cada ponto de C é o resultado de uma operação algébrica entre pontos e das imagens A e B respectivamente

$$C(x,y) = A(x,y) + B(x,y)$$
$$C(x,y) = A(x,y) - B(x,y)$$
$$C(x,y) = A(x,y) * B(x,y)$$
$$C(x,y) = A(x,y) / B(x,y)$$

Operações Algébricas

- Essenciais para ajustar e suavizar imagens em aplicações com imagens muitos ruidosas.
- Todas as operações aritméticas em imagens digitais são executadas pixel a pixel.



Adição de Imagens

- Permite obter a média de diversas imagens de uma mesma cena/objeto
- Pode ser utilizada para reduzir / eliminar ruídos aleatórios.
- Obtém o efeito fotográfico de super imposição ou exposição dupla, que consiste em bater várias fotografias no mesmo filme.
- A operação é executada pixel a pixel, tratando a imagem como uma matriz de números, e cada elemento da matriz é adicionado para o elemento correspondente da segunda matriz, dessa forma produzindo uma terceira matriz que armazena o resultado da adição dos elementos.

Adição de Imagens

- Na soma de duas imagens com intensidade variando de 0-255 tem-se como resultado valores na faixa de 0 a 510. Para resolver isso pode-se simplesmente dividir o resultado por dois
- Em geral, quando tem-se N imagens, faz-se a adição e divide-se o resultado por N.
- Outra forma é com base no resultado da soma fazer uma normalização, baseado na seguinte fórmula:

$$valor = 255 * \frac{\sum - \min()}{\max() - \min()}$$

Adição de Imagens

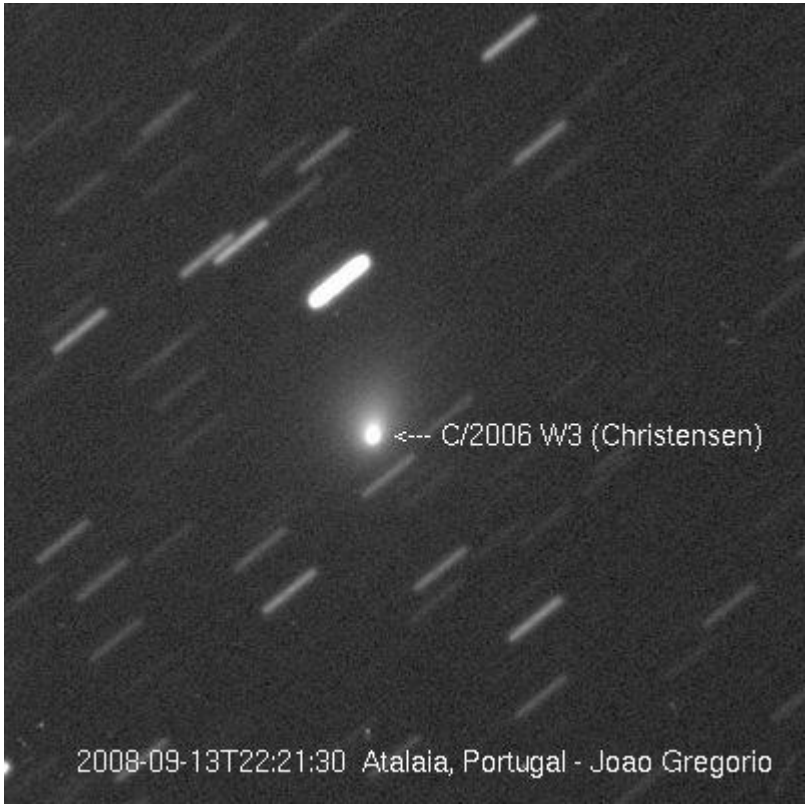


Imagem composta pela soma de 18 imagens de 2 minutos centradas em um cometa. Como o cometa tem movimento fica o efeito das estrelas "riscadas".

http://www.atalaia.org/encontro.php?id=298

Adição no OpenCv (add)

- Soma pixels imagens (P&B / Coloridas - **saturação**)
- Sintaxe:

  void add(InputArray src1, InputArray src2, OutputArray dst, InputArray mask=noArray(), int dtype=-1)

- Outra forma é fazer soma simples
  - dst = src1 + src2;
  - dst += src1; → **Equivale a dst = dst + src1**
  - dst = src1 + 50; → aumenta o brilho em 50

```
Mat orig1, orig2, dest1, dest2;
orig1 = imread("baixo.bmp", CV_LOAD_IMAGE_GRAYSCALE);
orig2 = imread("cima.bmp", CV_LOAD_IMAGE_GRAYSCALE);
add(orig1, orig2, dest1);
dest2 = orig1 + orig2;
```

Adição no OpenCv (addWeighted)

- Calcula uma soma ponderada de duas imagens
- Sintaxe:

void addWeighted(InputArray src1, double alpha, InputArray src2, double beta, double gamma, OutputArray dst, int dtype=-1)

  - Equivale a:
    - **dst = src1*alpha + src2*beta + gamma;**

```
addWeighted(orig1, 0.5, orig2, 0.5, 0, dest6);
//dest6 = orig*.5+orig2*.5+0;
```

Subtração de Imagens

- Mais amplamente utilizada e interessante que a adição, onde ela é usada para encontrar diferenças entre duas imagens, mesmo na presença de ruído.
- Usa-se uma imagem "mestre", e subtraí se uma nova imagem e as diferenças são destacadas.
- Pode-se usar em vídeo para detectar movimentos, direções, velocidades, ...

Subtração de Imagens

- Como resultado da subtração, os pixels podem variar de -255 a +255, logo, deve-se escaloná-los, adicionando 255 e dividindo por 2 cada pixel, ou usar o método de normalização descrito na adição.

www.inf.ufsc.br/~visao/threshold.html

Subtração no OpenCv (subtract)

- Diminui pixels imagens (P&B/Coloridas - **saturação**)
- Sintaxe:

  void subtract(InputArray src1, InputArray src2, OutputArray dst, InputArray mask=noArray(), int dtype=-1)

- Outra forma é fazer subtração simples
  - dst = src1 - src2;

orig1 – orig2

orig2 – orig1

```
subtract(orig1, orig2, dest3);
dest4 = orig2 - orig1;
```

Multiplicação de Imagens

- Importante para a correção de sombreamento
- Uma das principais dificuldades com a multiplicação é a faixa extrema de valores que podem ser gerados (de 0 a mais de 65000), pois na normalização, pode-se gerar uma significativa perda de informações.

Multiplicação de Imagens

* **0.5**

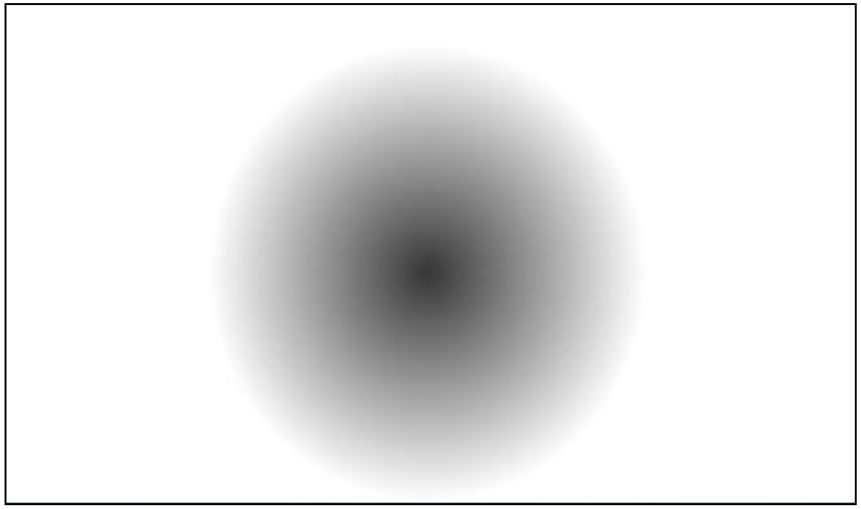
=

Realce das baixas magnitudes

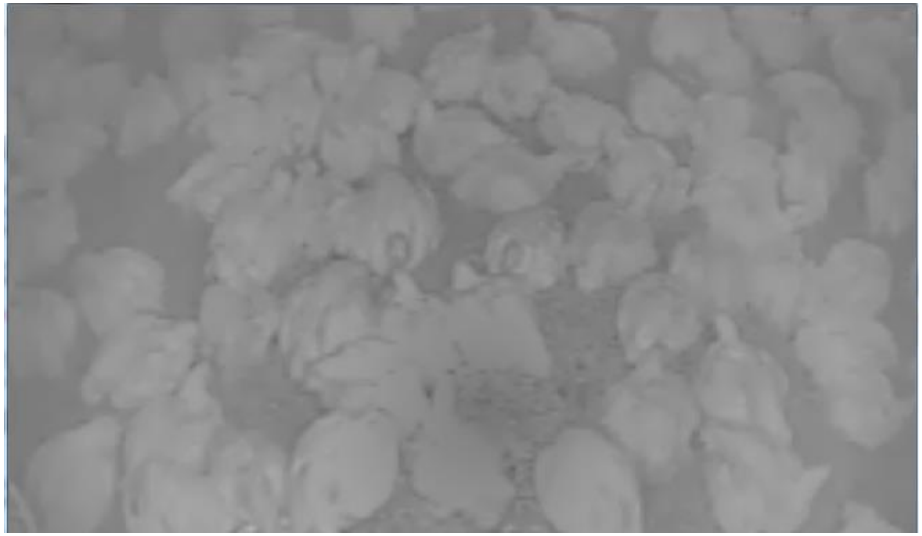
http://www.inf.ufrgs.br/~danielnm/docs/OperacoesAritmeticasImagens.pdf

Multiplicação de Imagens

*



=



Multiplicação no OpenCv (multiply)

- Calcula o produto entre os pixels de 2 imagens
- Sintaxe:

  void multiply(InputArray src1, InputArray src2, OutputArray dst, double scale=1, int dtype=-1)

- Ou ainda:
  - dst = src1 * constante; → **não aceita multiplicação de 2 objetos Mat**

```
multiply(orig1, orig2, dest7);
//dest7 = orig1 * 100;
```

Divisão de Imagens

- A divisão como a multiplicação também é importante para a correção de sombreamento
- Ela pode ser utilizada na remoção de fundos quando detectores lineares ou câmeras são usadas.

http://www.inf.ufrgs.br/~danielnm/docs/OperacoesAritmeticasImagens.pdf

Divisão no OpenCv (divide)

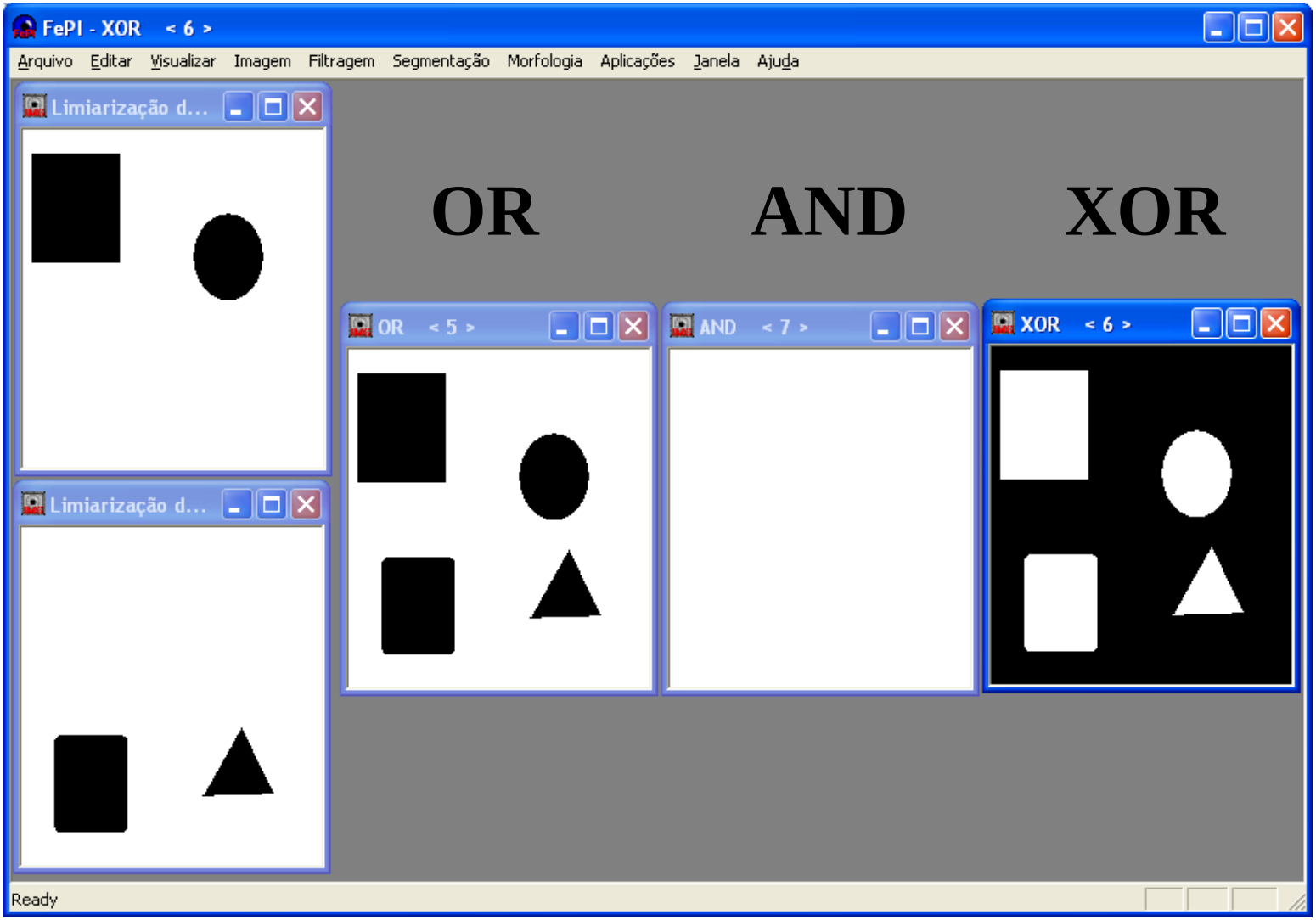
- Calcula a divisão entre os pixels de 2 imagens
- Sintaxe:

  void divide(InputArray src1, InputArray src2, OutputArray dst, double scale=1, int dtype=-1)

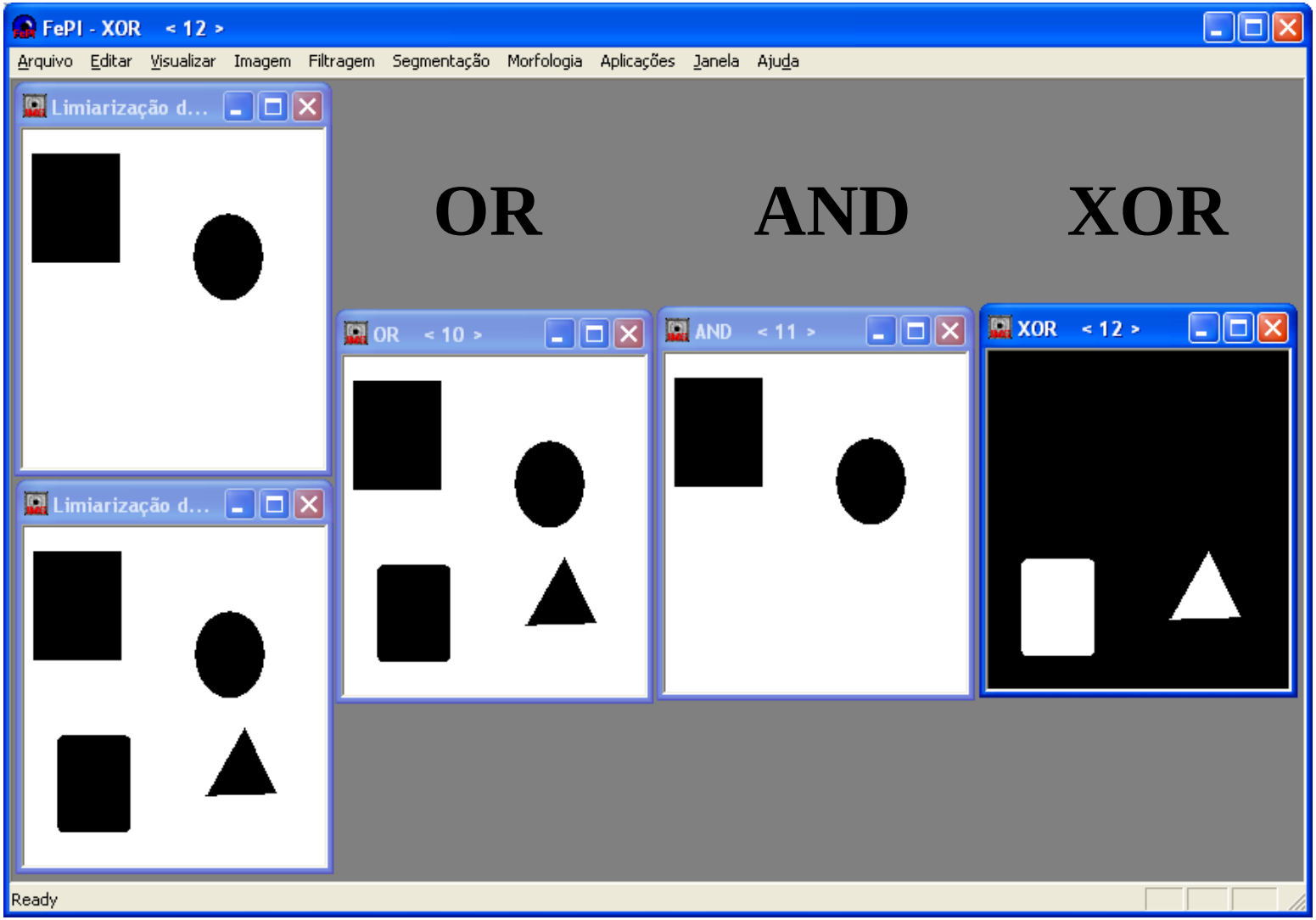
- Ou ainda:
  - dst = src1 / constante;
  - dst = sr1/src2

```
//divide(orig1, orig2, dest8);
dest8 = orig1 / orig2;
```

Operações Algébricas Binárias



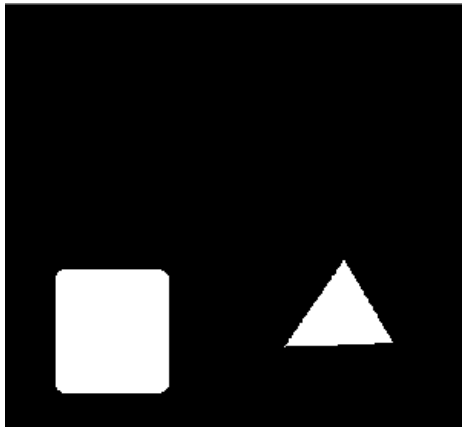
Operações Algébricas Binárias



Operações Algébricas Binárias

- **AND Lógico**
  - bitwise_and
    - void bitwise_and(InputArray src1, InputArray src2, OutputArray dst, InputArray mask=noArray())

```
bitwise_and(orig1, orig2, dest9);
```



orig1

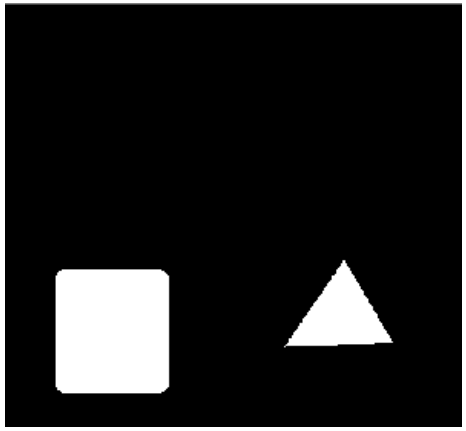
orig2

And

Operações Algébricas Binárias

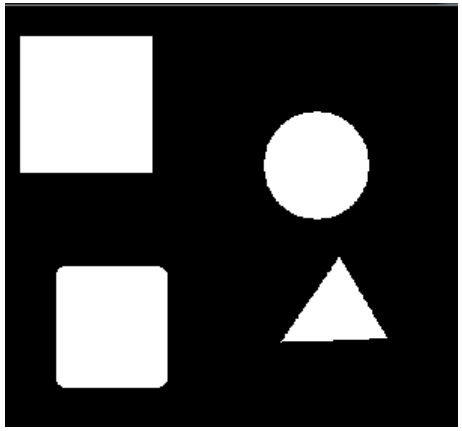
- **OR Lógico**
  - bitwise_or
    - void bitwise_or(InputArray src1, InputArray src2, OutputArray dst, InputArray mask=noArray())

```
bitwise_or(orig1, orig2, dest10);
```



orig1

orig2

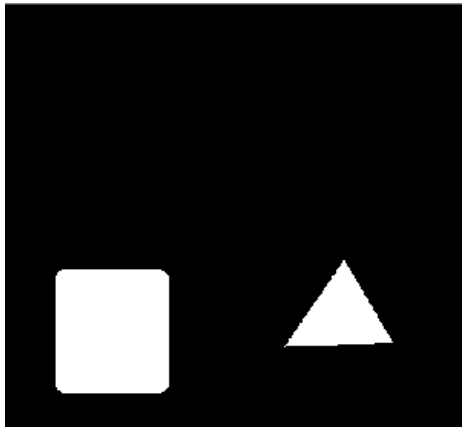


OR

Operações Algébricas Binárias

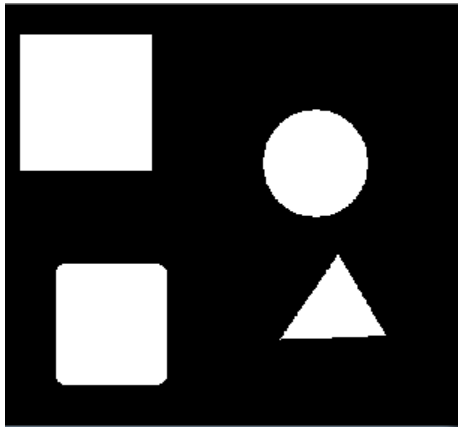
- **XOR Lógico**
  - bitwise_xor
    - void bitwise_xor(InputArray src1, InputArray src2, OutputArray dst, InputArray mask=noArray())

```
bitwise_xor(orig1, orig2, dest12);
```



orig1

orig2



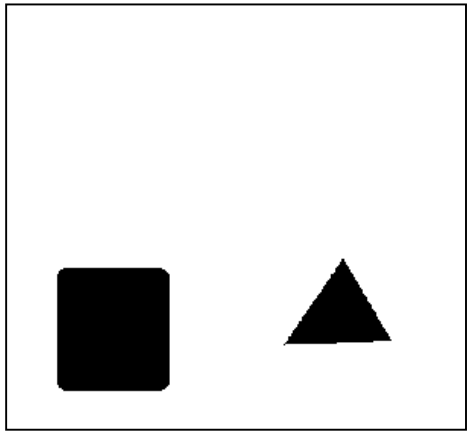
XOR

Operações Algébricas Binárias

- **Negação Lógica**
  - bitwise_not
    - void bitwise_not(InputArray src1, OutputArray dst, InputArray mask=noArray())

```
bitwise_not(orig1, dest11);
```

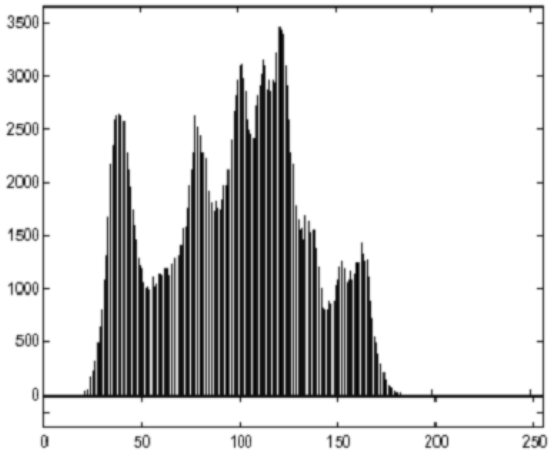
orig1

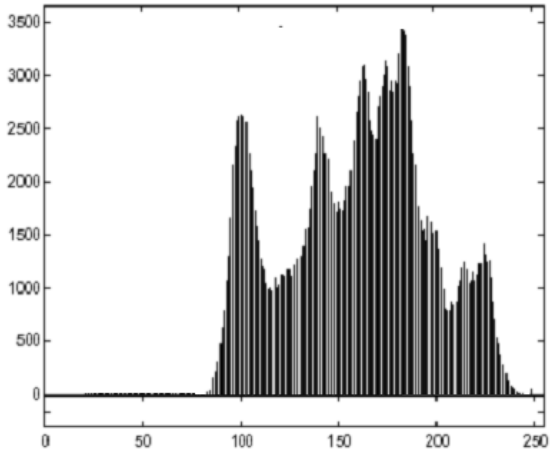


Negação

Brilho

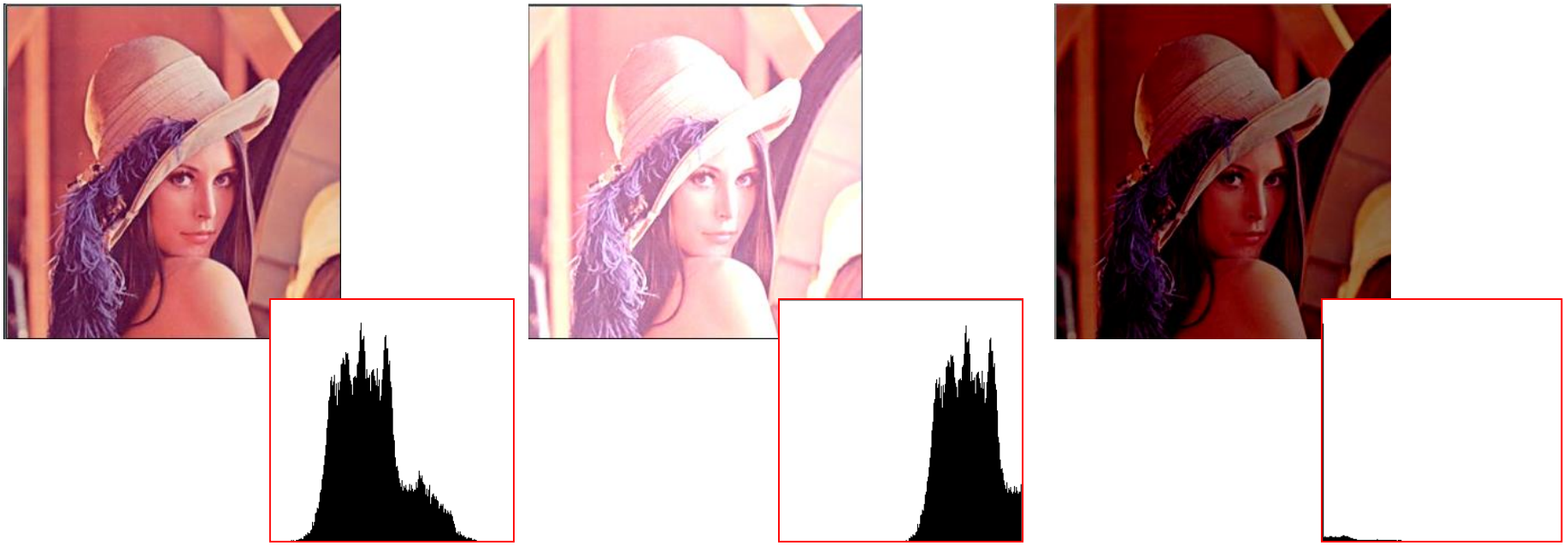
- Através da soma/subtração de um valor a intensidade de um pixel pode-se tornar uma imagem mais ou menos escura
- Deve-se controlar a saturação (a intensidade deve ficar na faixa 0-255)





Brilho

- A mudança do brilho equivale a aplicar a cada pixel da imagem um valor
  - Valor maior que 0 torna mais claro (ex. +100)
  - Valor menor que 0 torna mais escuro (ex. -100)



Brilho


```
for( int y = 0; y < image.rows; y++ ) {
  for( int x = 0; x < image.cols; x++ ){
    for( int c = 0; c < 3; c++ ) {
      new_image.at<Vec3b>(y,x)[c] =
         saturate_cast<uchar>((image.at<Vec3b>(y,x)[c])+brilho);
    }
  }
}
```

```
image.convertTo(new_image2, -1, 1, brilho);
```

convertTo - OpenCv

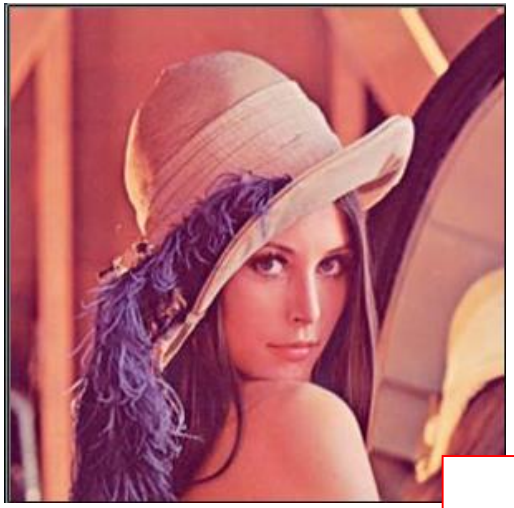
- Converte um Mat em outro tipo de dados com a opção do uso de escala
- Sintaxe:
  - void Mat::convertTo(OutputArray dst, int rtype, double alpha=1, double beta=0)
  - Rtype → tipo matriz saida (-1 → mesmo tipo origem)
  - Alpha (α) → fator de escala
  - Beta (β) → incrementa um valor na saída

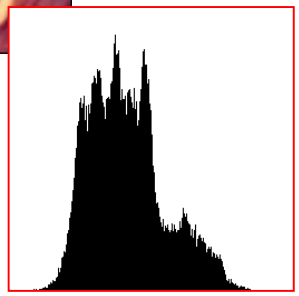
$$m(x,y) = saturate\_cast < rType > (\alpha(*this)(x,y) + \beta)$$

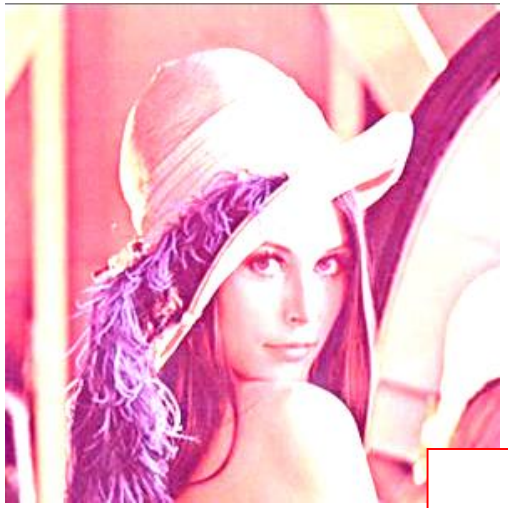
```
image.convertTo(new_image2, -1, 1, brilho);
```

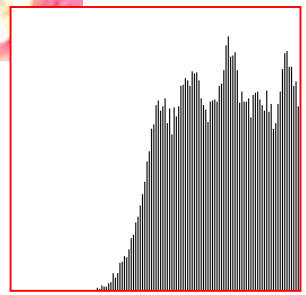
Contraste

- Melhora a qualidade das imagens sob os critérios subjetivos do olho humano
- Equivale a multiplicar cada pixel por um fator







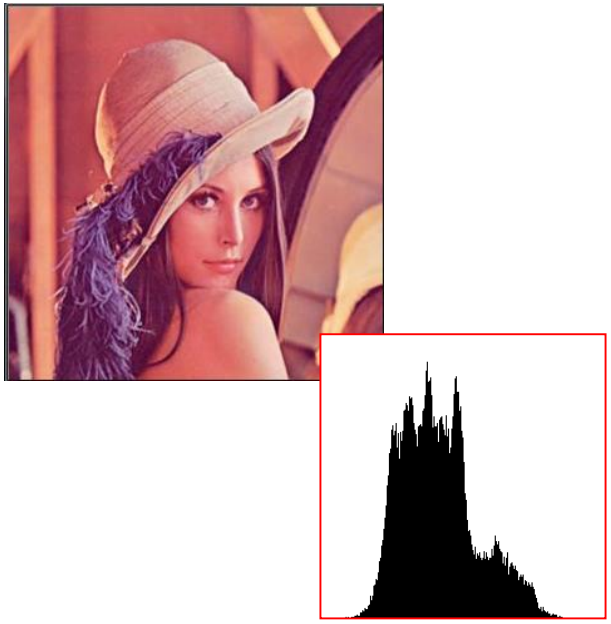


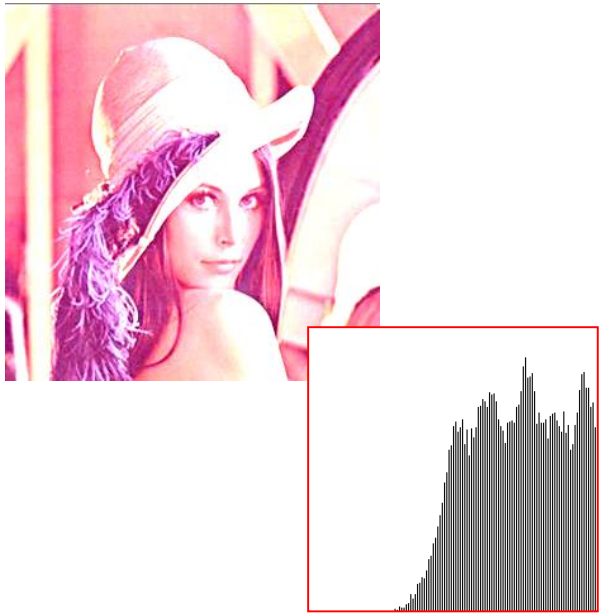
Contraste - OpenCv

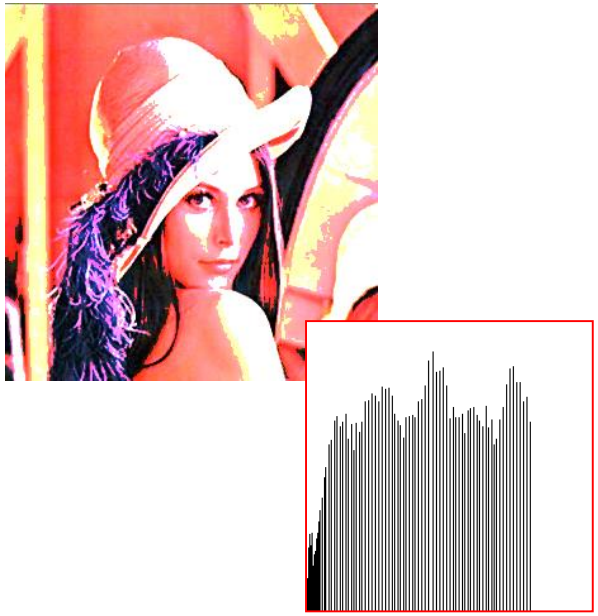

```
for( int y = 0; y < image.rows; y++ ) {
  for( int x = 0; x < image.cols; x++ ){
    for( int c = 0; c < 3; c++ ) {
      new_image3.at<Vec3b>(y,x)[c] =
         saturate_cast<uchar>((contraste*image.at<Vec3b>(y,x)[c]));
    }
  }
}
```

```
image.convertTo(new_image4, -1, contraste, 0);
```

Contraste + Equalização





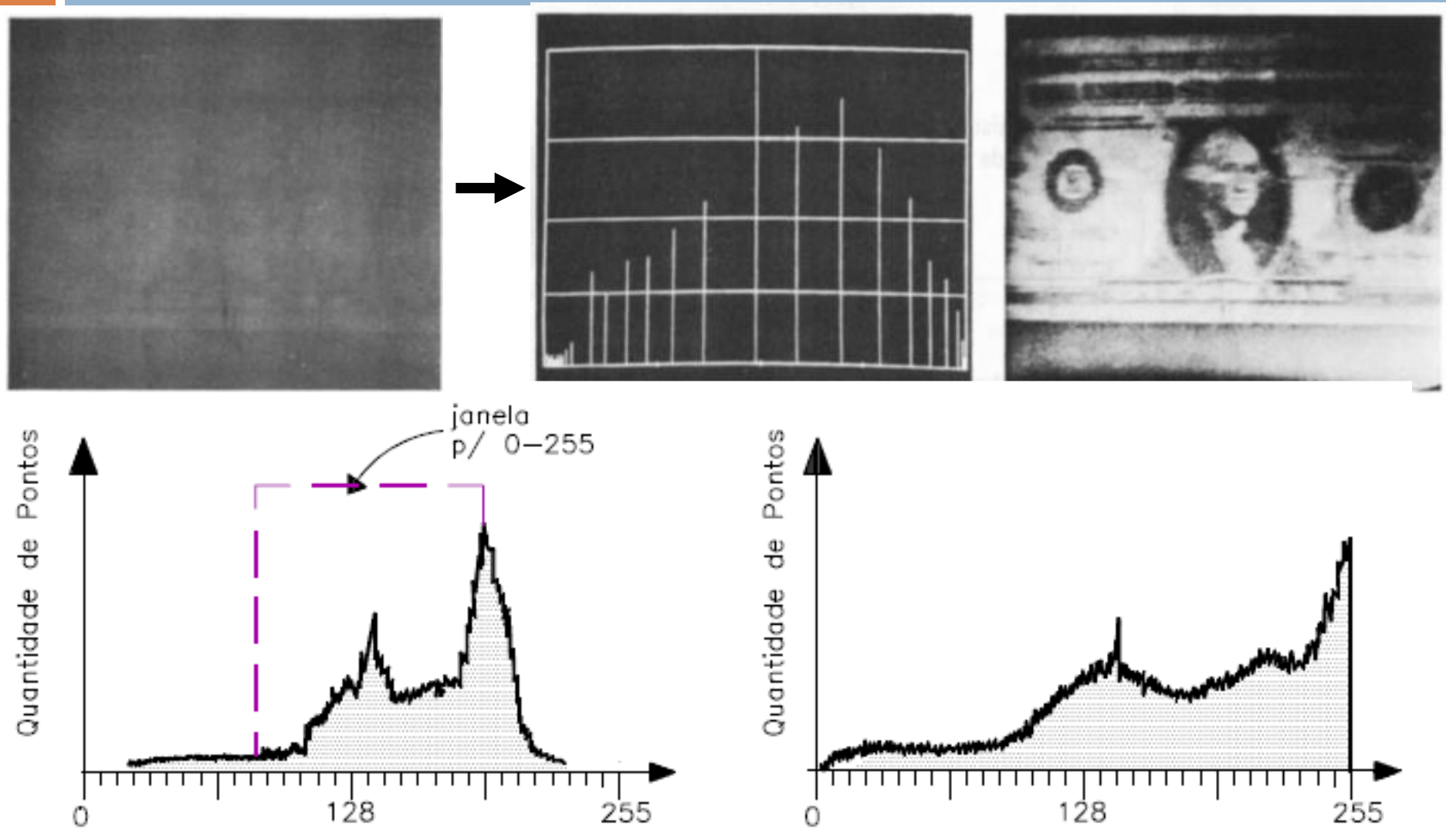


Contraste

- Supondo que o menor valor de interesse de uma imagem original seja ***min1*** e o maior ***max1*** e queira-se transformá-la em uma nova faixa (***min2***, ***max2***), todo ponto ***N***, para melhoria de contraste tem-se:

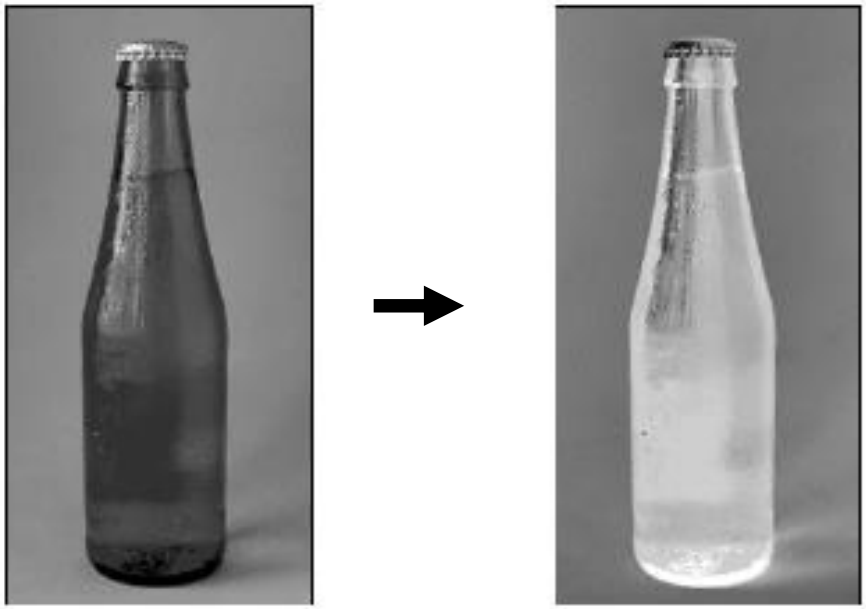
$$f(N) = \frac{(N - min1)\,(max2 - min2)}{(max1 - min1)} + min2$$

Contraste


janela
p/ 0–255
Quantidade de Pontos
0
128
255
Quantidade de Pontos
0
128
255

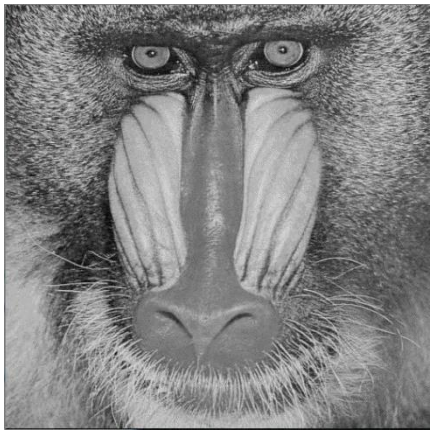
Negativo

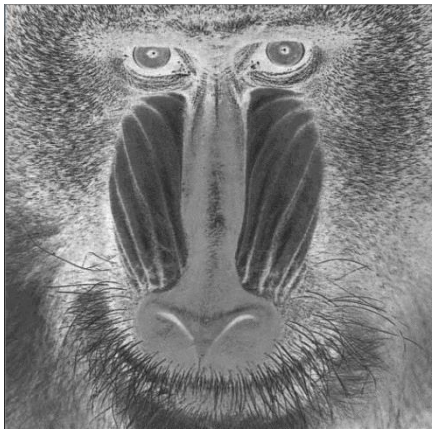
- Inverte a imagem, ou seja, pega a intensidade máxima do canal (255) e diminui do valor do pixel



Negativo - OpenCv


```
Mat image = imread( "lena.bmp");
Mat invertida = Mat::zeros( image.size(), image.type() );
Mat aux = Mat::ones(image.size(), image.type())*255;
subtract(aux, image, invertida);
```



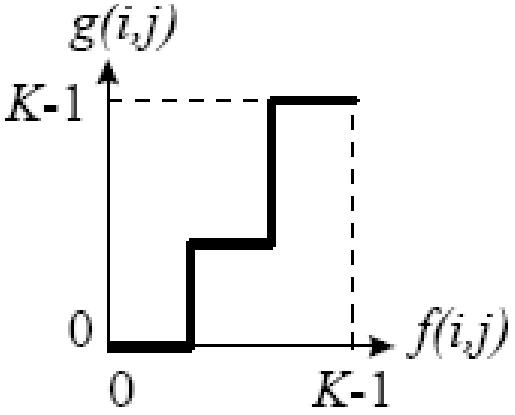


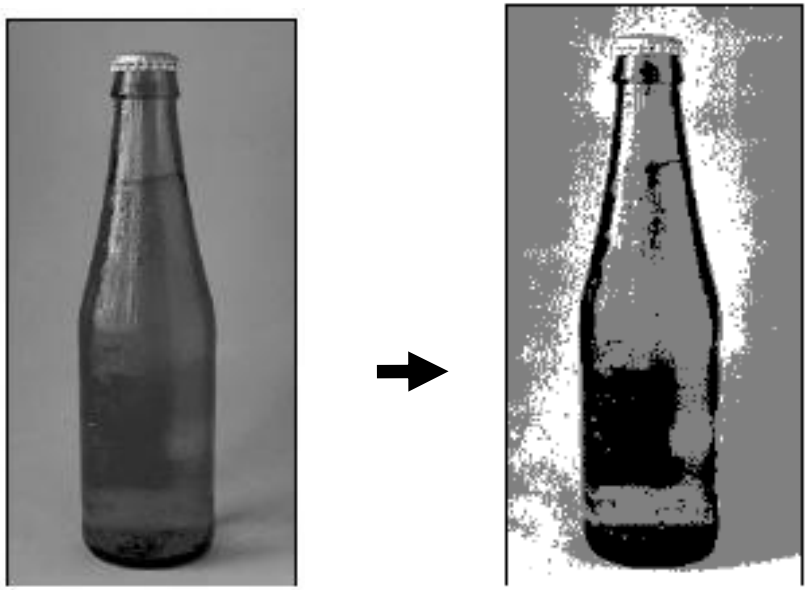
Limiarização – Threshold

- Com base no histograma de uma imagem define-se um limiar e, com isso, transforma-a em uma imagem binária (preto e branco)

Limiarização – Threshold Binível

- Similar ao threshold normal, porem usa dois limiares, e mantém a imagem em tons de cinza na faixa escolhida





Threshold - OpenCv

- Threshold Binário (**THRESH_BINARY**)
  - Se (pixel>limiar) valorMaximo senão valorMinimo
- Threshold Binário invertido (**THRESH_BINARY_INV**)
  - Se (pixel>limiar) valorMinimo senão valorMaximo
- Truncado (**THRESH_TRUNC**)
  - Se (pixel>limiar) **Threshold** senão mantém valor do pixel
- Threshold até zero (**THRESH_TOZERO**)
  - Se (pixel>limiar) mantém valor senão 0
- Threshold até zero invertido (**THRESH_TOZERO_INV**)
  - Se (pixel>limiar) 0 senão mantém valor
- OTSU → (**THRESH_OTSU**)
  - Calcula limiar através da técnica de OTSU

Threshold - OpenCv

- Aplica um limiar de threshold a cada pixel
- Sintaxe:
  - double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)
  - Thresh → limiar
  - MaxVal → valor máximo a ser usado no **THRESH_BINARY** e **THRESH_BINARY_INV**
  - Type →tipo do threshold a ser usado
- Para usar o OTSU, ele deve ser combinado com outro tipo e o limiar deve estar zerado

```
cv::threshold(im_gray, img_bw, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
```

Threshold - OpenCv


```
void Threshold_Demo( int, void* ){
  /* 0: Binary
     1: Binary Inverted
     2: Threshold Truncated
     3: Threshold to Zero
     4: Threshold to Zero Inverted
   */
  threshold( src_gray, dst, threshold_value, 255,threshold_type );
  imshow("Threshold Demo", dst );
}
```

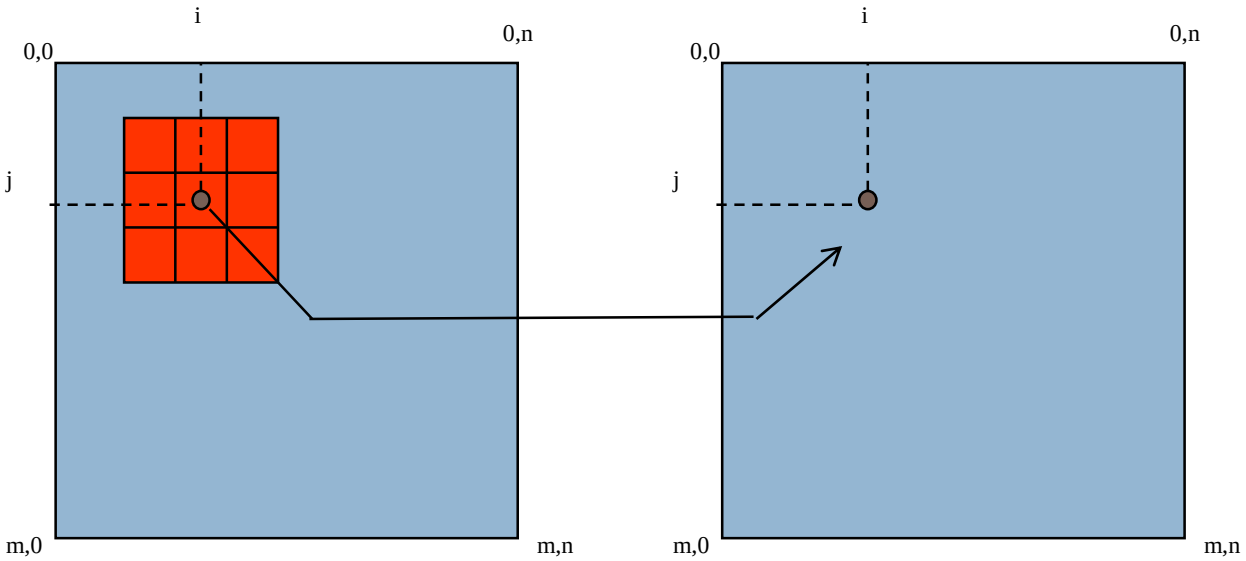
```
int main( ){
  int c;
  src = imread("baboon.bmp");
  cvtColor( src, src_gray, CV_RGB2GRAY );
  namedWindow("Threshold Demo", CV_WINDOW_AUTOSIZE );

  /// Cria Trackbar para escolha do threshold
  createTrackbar("Tipo:","Threshold Demo",&threshold_type,4,Threshold_Demo);
  createTrackbar("Valor:","Threshold Demo",&threshold_value,255,Threshold_Demo);

  Threshold_Demo( 0, 0 );
  do {
    c = waitKey( 20 );
  }while((char)c != 27);
}
```

Transformações Locais

- Através da vizinhança produz novas intensidades de pixel
- Os filtros locais são técnicas baseadas na convolução de máscaras (*templates, kernel*)

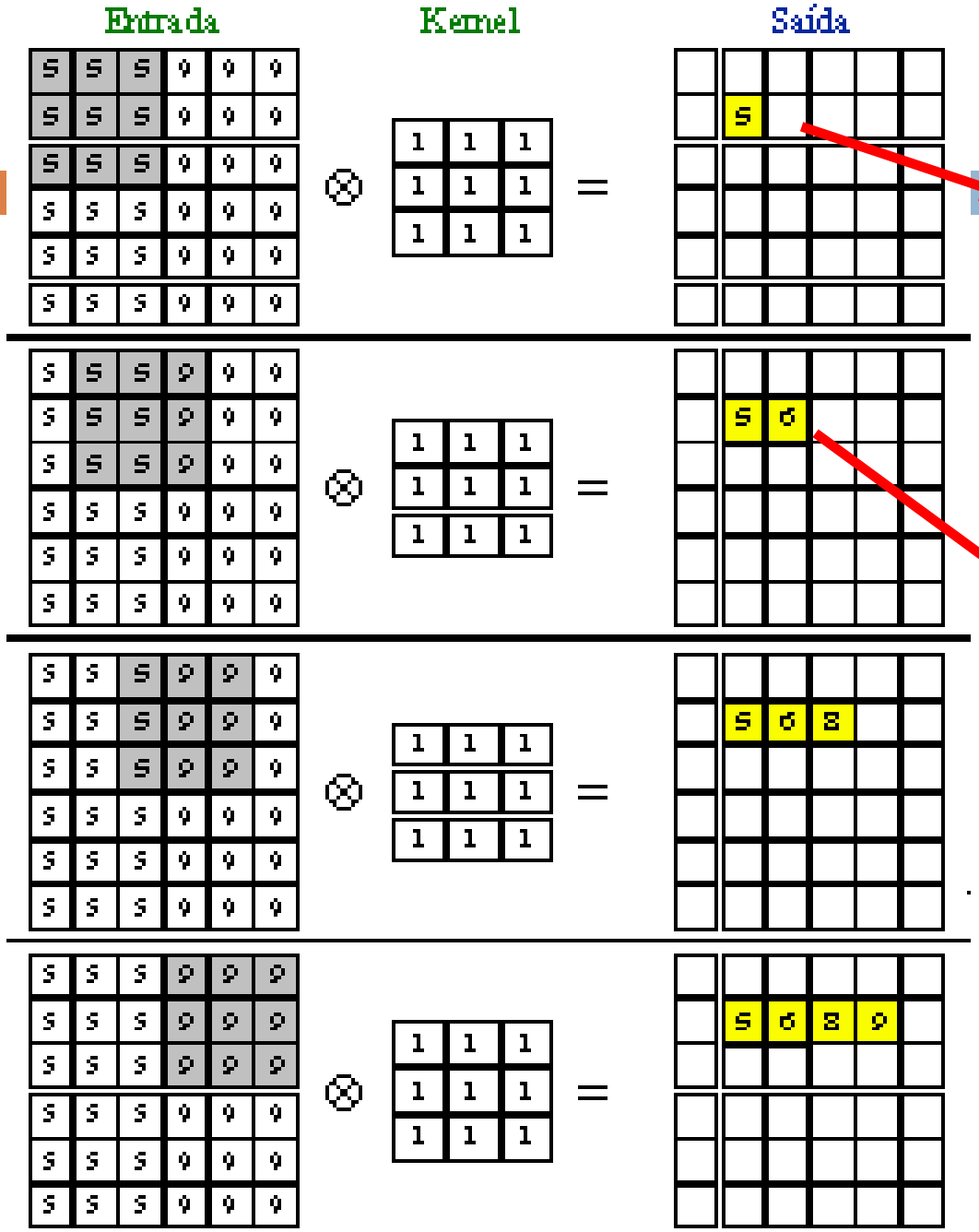


Convolução

- Seja T(x,y) uma máscara (mxn) e I(x,y) uma imagem (MxN)
- A convolução de T ⊗ I é dada por:

$$T \otimes I(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} T(i,j) I(x+i, y+i)$$

Filtro Passa-baixa



Soma-se:
5+5+5+5+5+5+5+5+5 = 45
Divide-se:
45 / 9 = 5
Atribui-se:
5 ao pixel em questão

Soma-se:
5+5+5+5+5+5+9+9+9 = 57
Divide-se:
57 / 9 = 6,333
Atribui-se:
6 ao pixel em questão

Convolução - OpenCv

- Faz a convolução de uma imagem com um kernel
- Sintaxe:

void filter2D(InputArray src, OutputArray dst, int ddepth, InputArray kernel, Point anchor=Point(-1,-1), double delta=0, int borderType=BORDER_DEFAULT)

  - Ddepth →profundidade da saida (-1 = src.depth())
  - Kernel → matriz de convolução a ser usada
  - Anchor → ponto base na convolução (centro = (-1, -1))
  - borderType → como tratar as bordas
    - BORDER_CONSTANT → considera zeros fora da faixa
    - BORDER_REPLICATE → replica valor mais proximo

Convolução - OpenCv


```
Mat_<float> teste = (Mat_<float>(6,6) <<  5, 5, 5, 9, 9, 9,
                                         5, 5, 5, 9, 9, 9,
                                         5, 5, 5, 9, 9, 9,
                                         5, 5, 5, 9, 9, 9,
                                         5, 5, 5, 9, 9, 9,
                                         5, 5, 5, 9, 9, 9);
Mat_<float> passaBaixa = 1.0/9.0 * (Mat_<float>(3,3) <<  1, 1, 1,
                                                         1, 1, 1,
                                                         1, 1, 1);
Mat_<float> sai;

filter2D(teste,sai,-1,passaBaixa,Point(-1,-1),0,BORDER_REPLICATE);
cout << sai;
```

```
[5, 5, 6.333333, 7.666667, 9, 9;
  5, 5, 6.333333, 7.666667, 9, 9;
  5, 5, 6.333333, 7.666667, 9, 9;
  5, 5, 6.333333, 7.666667, 9, 9;
  5, 5, 6.333333, 7.666667, 9, 9;
  5, 5, 6.333333, 7.666667, 9, 9]
```

Filtros

- Podem ser divididos em:
  - Suavização
    - Passa Baixa e média espacial
    - Mediana
    - ...
  - Detecção de borda
    - Passa Alta
    - Passa Banda
    - Gradientes (roberts, sobel, ...)

Passa-baixa - Média

- Suavização ("Smoothing") da imagem
- Redução do efeito de ruído, pela média ponderada
- Quanto maior a máscara maior efeito de borramento

Original

3x3

5x5

7x7

Passa-baixa - Média

Original

Filtragem de média 3x3

Filtragem de média 5x5

Média - OpenCv

- Para fazer a média com OpenCv é possivel usar a função filter2d() ou a função **blur()**
- Sintaxe:

void blur(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int borderType = BORDER_DEFAULT)

  - Ksize → tamanho do kernel para o borramento
  - Anchor → ponto base na convolução (centro = (-1, -1))
  - borderType → como tratar as bordas
    - BORDER_CONSTANT → considera zeros fora da faixa
    - BORDER_REPLICATE → replica valor mais próximo

Média - OpenCv

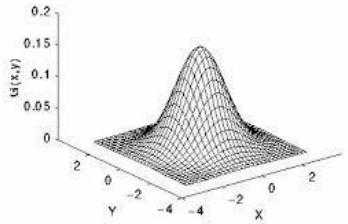

```
void testaBlur( int, void* ){
  blur( src, dst, Size( 3+tamBloco, 3+tamBloco ), Point(-1,-1));
  imshow("testaBlur", dst );
}
```

```
int main( ){
  int c;
  src = imread("retina.jpg");
  namedWindow("testaBlur", CV_WINDOW_AUTOSIZE );

  createTrackbar("Tamanho Bloco:","testaBlur",&tamBloco,50,testaBlur);

  testaBlur( 0, 0 );
  do {
    c = waitKey( 20 );
  }while((char)c != 27);
}
```

Filtro Gaussiano



- Um dos filtros mais úteis, nele o pixel localizado no meio do kernel tem um peso maior, e estes pesos vão sendo diminuídos gradativamente
- Sintaxe:

void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0, int borderType=BORDER_DEFAULT)

  - Ksize → tamanho kernel (3,5,7,...)
  - sigmaX → desvio padrão em X
  - sigmaY → desvio padrão em Y

1/256	4/256	6/256	4/245	1/256
4/256	16/256	24/256	16/256	4/256
6/256	24/256	36/256	24/256	6/256
4/256	16/256	24/256	16/256	4/256
1/256	4/256	6/256	4/245	1/256

Filtro Gaussiano


```
void testaGaussianBlur( int, void* ){
  GaussianBlur( src, dst, Size( 3+tamBloco*2, 3+tamBloco*2 ), 0, 0);
  imshow("testaBlur", dst );
}
```

```
int main( ){
  int c;
  src = imread("baboon.bmp");
  namedWindow("testaBlur", CV_WINDOW_AUTOSIZE );

  createTrackbar("Blur:","testaBlur",&tamBloco,50,testaBlur);
  createTrackbar("GaussianBlur:","testaBlur",&tamBloco,50,testaGaussianBlur);

  testaGaussianBlur(0,0);
  do {
    c = waitKey( 20 );
  }while((char)c != 27);
}
```

Filtro Bilateral

- Faz o efeito de blur preservando as bordas
- Sintaxe:
  - void bilateralFilter(InputArray src, OutputArray dst, int d, double sigmaColor, double sigmaSpace, int borderType=BORDER_DEFAULT)
  - D → diametro dos pixels da vizinhança
  - sigmaColor → faixas de cores a serem agrupadas
  - sigmaSpace → especifica o tamanho da vizinhança
- Para os 2 sigmas iguais (< 10) pouco efeito (>150) efeito de cartoon

http://eric-yuan.me/bilateral-filtering/

Filtro Bilateral


```
void testaBilateralFilter( int, void* ){
  bilateralFilter ( src, dst, tamBloco, tamBloco*2, tamBloco/2 );
  imshow("testaBlur", dst );
}
```

Mediana

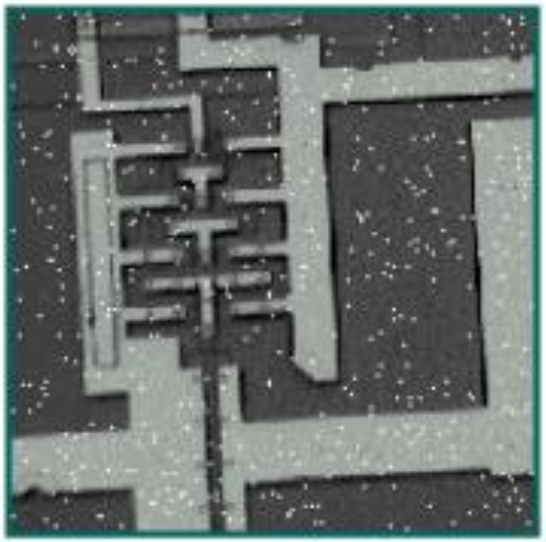
- Permite reduzir o ruído das imagens
- Com base nos vizinhos do pixel, ordena-os e pega-se o elemento do meio
- Este tipo de filtro é particularmente adaptado à remoção de ruído impulsivo que aparece em regiões limitadas da imagem.

Mediana

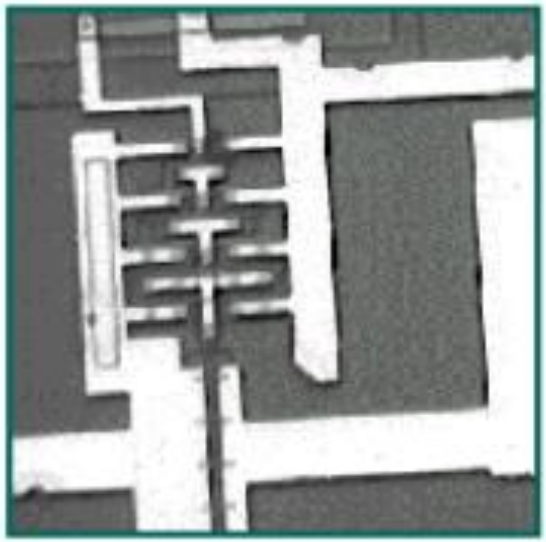
75	77	77
77	253	81
77	75	79

75 75 77 77 77 77 79 81 253

75	77	77
77	77	81
77	75	79



Original



Filtrado com Mediana 3x3

Mediana

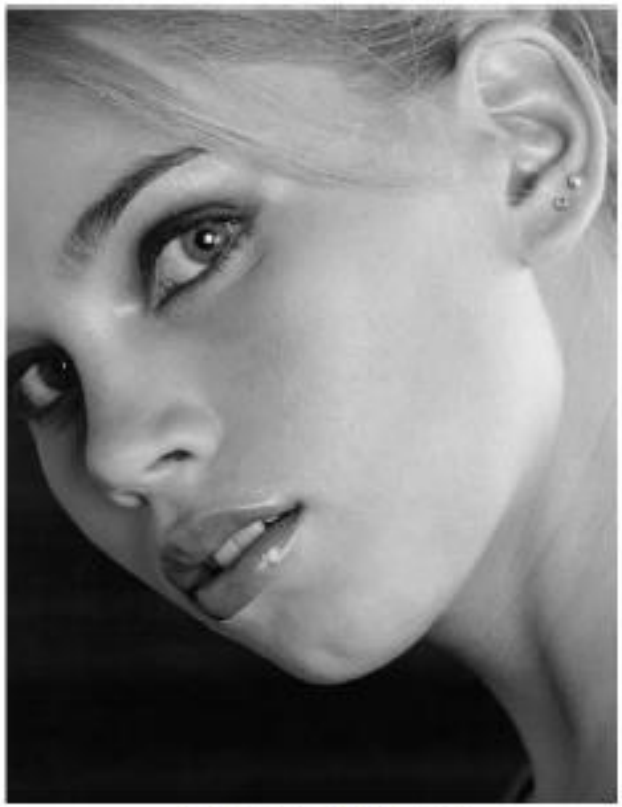


Imagem original

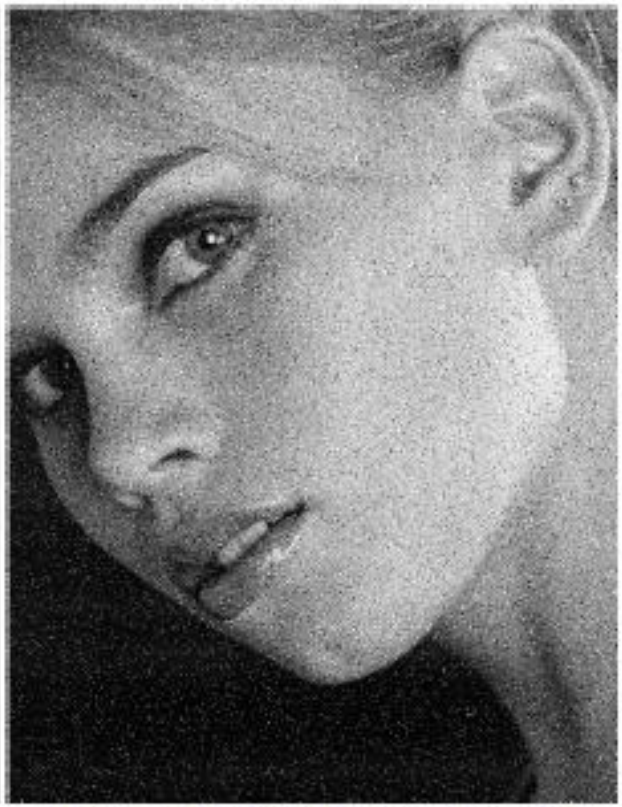


Imagem com ruído adicionado (20%)

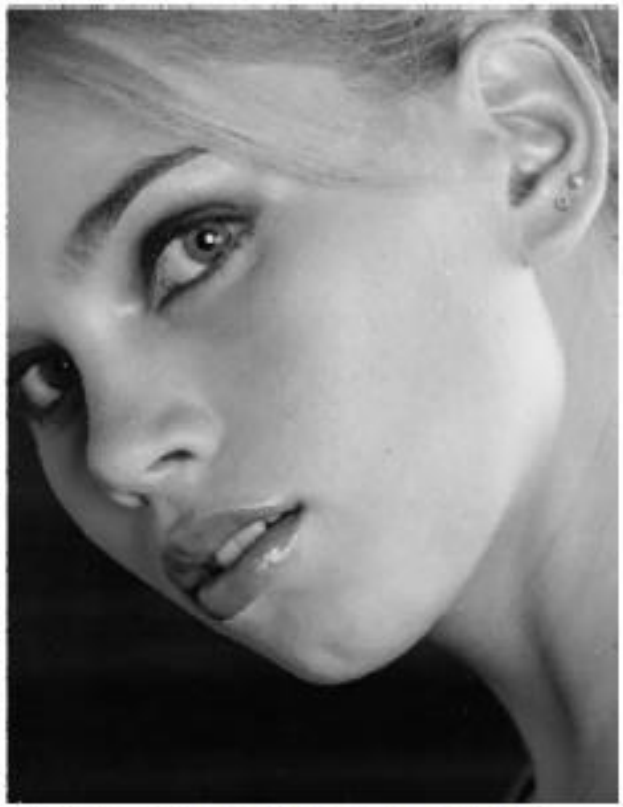


Imagem depois de filtrada com um filtro de mediana 3x3

Comparação Mediana/Média

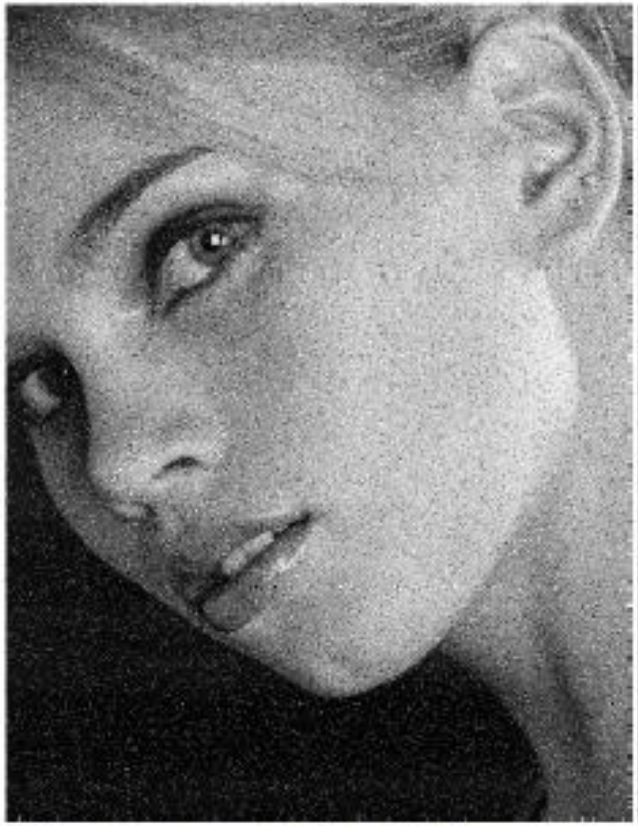


Imagem com ruído adicionado (20%)

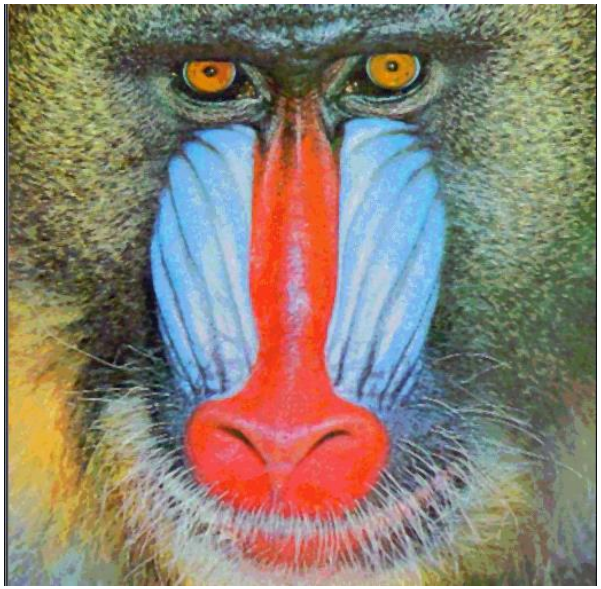
Imagem depois de filtrada com um filtro de mediana 3x3

Imagem depois de filtrada com um filtro de média

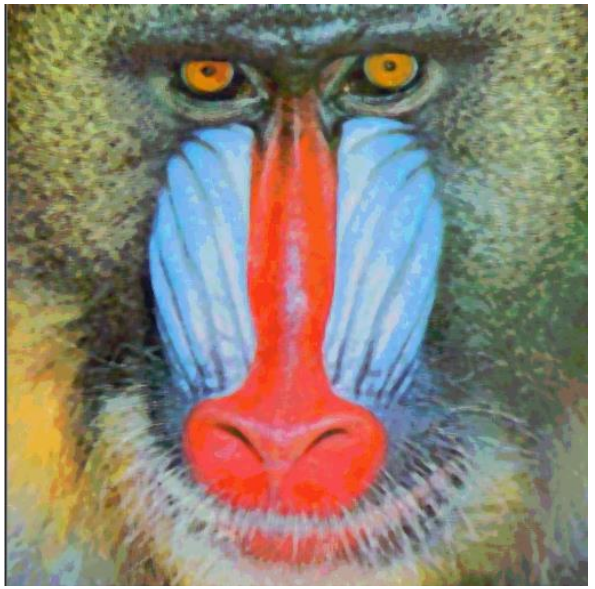
Mediana - OpenCv

- Sintaxe:
  - void medianBlur(InputArray src, OutputArray dst, int ksize)
  - Ksize → tamanho do kernel (3 ou 5)

Original



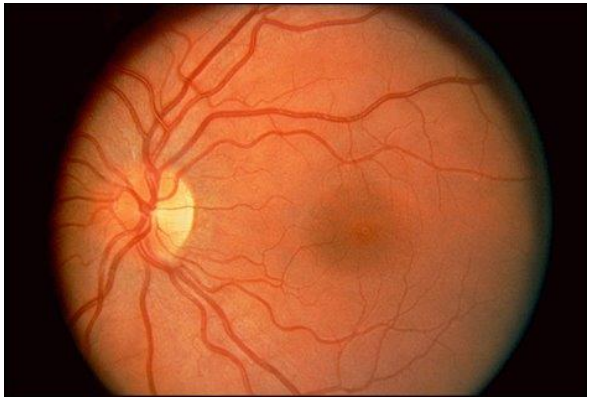
3x3

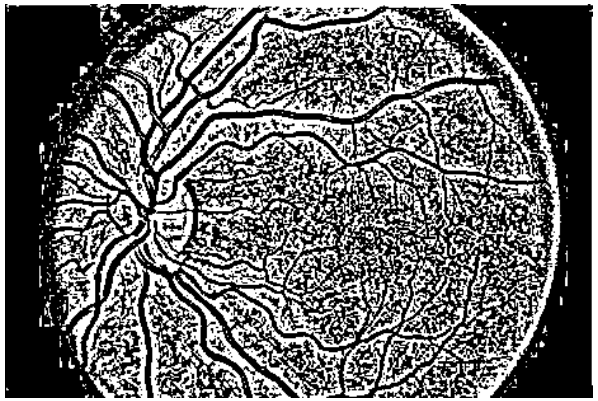


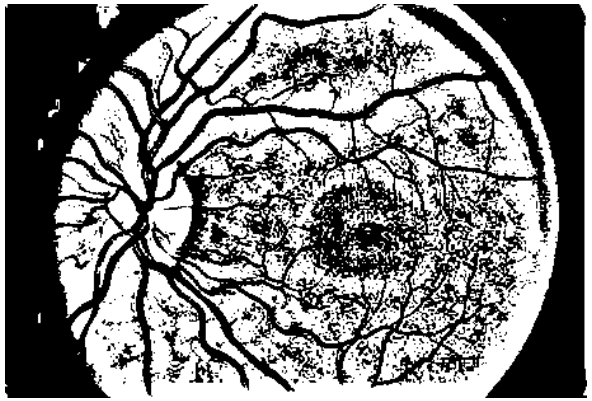
5x5

Adaptative Threshold

- Um threshold convencional usa o mesmo limiar por toda a imagem, já o adaptativo muda o limiar dinamicamente por toda a imagem







Adaptative Threshold - OpenCv

- Faz threshold usando um limiar local e não global
- Sintaxe:

void adaptiveThreshold(InputArray src, OutputArray dst, double maxValue, int adaptiveMethod, int thresholdType, int blockSize, double C)

  - maxValue → valor máximo a ser usado
  - adaptiveMethod → ADAPTIVE_THRESH_MEAN_C ou ADAPTIVE_THRESH_GAUSSIAN_C
  - thresholdType → THRESH_BINARY ou THRESH_BINARY_INV
  - blockSize → tamanho da vizinhança
  - C → Constante subtraída da média

Adaptative Threshold - OpenCv


```
void Threshold_Demo( int, void* ){
  adaptiveThreshold(src_gray, dst, 255,
        ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY, 3 +tamBloco*2, 0);
  imshow("Adaptative Threshold", dst );
}
```

```
int main( ){
  int c;
  src = imread("retina.jpg");
  cvtColor( src, src_gray, CV_RGB2GRAY );
  namedWindow("Adaptative Threshold", CV_WINDOW_AUTOSIZE );

  createTrackbar("Tamanho Bloco:","Adaptative Threshold",&tamBloco,50,Threshold_Demo);

  Threshold_Demo( 0, 0 );
  do {
    c = waitKey( 20 );
  }while((char)c != 27);
}
```

Detectores de Bordas
Filtros passa alta

- Realça detalhes, torna a imagem mais aguda, onde as transições entre regiões diferentes tornam-se mais nítidas.
- Destacam informações de bordas, linhas, curvas ou manchas, porém, também enfatizam o ruído
- Realçam a cena, segundo direções preferenciais de interesse, definidas pelas máscaras.

Filtros passa alta – direcionais

	1	1	1			1	1	1
Norte	1	-2	1		Nordeste	-1	-2	1
	-1	-1	-1			-1	-1	1
	-1	1	1			-1	-1	1
Leste	-1	-2	1		Sudeste	-1	-2	1
	-1	1	1			1	1	1
	-1	-1	-1			1	-1	-1
Sul	1	-2	1		Sudoeste	1	-2	-1
	1	1	1			1	1	1
	1	1	-1			1	1	1
Oeste	1	-2	-1		Noroeste	1	-2	-1
	1	1	-1			1	-1	-1

Filtros passa alta – não direcionais

- Utilizado para realçar bordas, independentemente da direção.
- A máscara alta deixa bloqueia os baixos níveis de cinza, isto é, a imagem fica mais clara.
- A máscara baixa produz uma imagem mais escura.
- A máscara média apresenta resultados intermediário

Alta				Média				Baixa		
-1	-1	-1		0	-1	0		1	-2	1
-1	8	-1		-1	4	-1		-2	3	-2
-1	-1	-1		0	-1	0		1	-2	1

Operador de Roberts

1	0
0	-1

0	1
-1	0

Operador Sobel

Mudanças Verticais

G_x

-1	-2	-1
0	0	0
1	2	1

Gradiente

$$G = \sqrt{G_x^2 + G_y^2}$$

$$G = |G_x| + |G_y|$$

Mudanças Horizontais

G_y

-1	0	1
-2	0	2
-1	0	1

docs.opencv.org/modules/imgproc/doc/filtering.html?highlight=sobel#sobel

Operador Sobel - OpenCv

- Sintaxe:
  - void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize=3, double scale=1, double delta=0, int borderType=BORDER_DEFAULT)
  - Dx e Dy → ordem das derivadas de X e Y
- **convertScaleAbs** → escalona, calcula valor absoluto e converte o resultado para 8 bits

```
Sobel( image, grad_x, image.depth(), 1, 0, 3, 1, 0, BORDER_REPLICATE );
convertScaleAbs( grad_x, abs_grad_x );

Sobel( image, grad_y, image.depth(), 0, 1, 3, 1, 0, BORDER_REPLICATE );
convertScaleAbs( grad_y, abs_grad_y );

addWeighted( abs_grad_x, 0.5, abs_grad_y, 0.5, 0, gradFinal );
```

Operador Laplace

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

-1	2	-1
2	-4	2
-1	2	-1

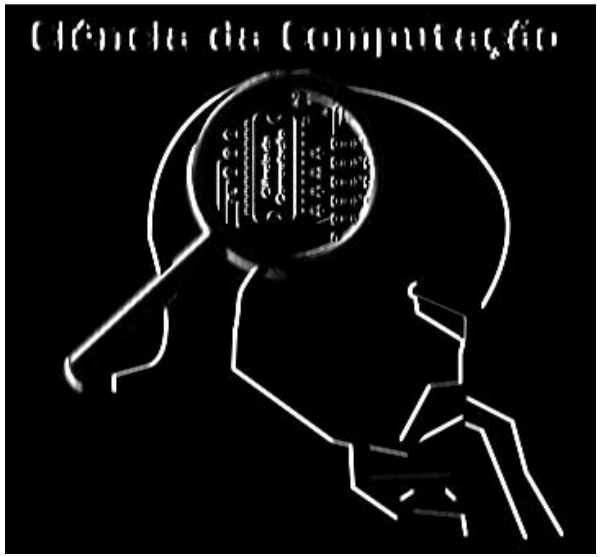
2	-1	2
-1	-4	-1
2	-1	2

```
Laplacian( image, dst, image.depth(), 3, 1, 0, BORDER_REPLICATE );
convertScaleAbs( dst, laplace );
```

Prewitt

Horizontal

1	1	1
0	0	0
-1	-1	-1



Vertical

-1	0	1
-1	0	1
-1	0	1

Diagonal

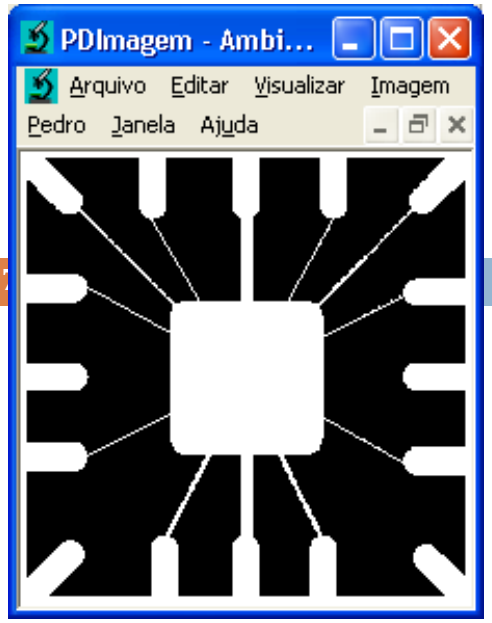
0	1	1
-1	0	1
-1	-1	0

Kirsch

$$\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

$$\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

Line Masks



a)

-1	-1	-1
2	2	2
-1	-1	-1

b)

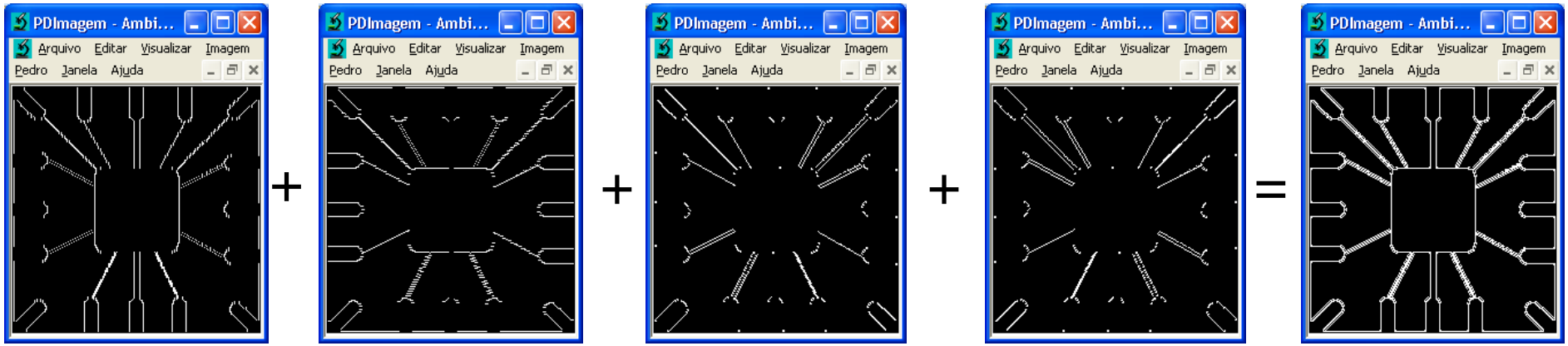
-1	2	-1
-1	2	-1
-1	2	-1

c)

-1	-1	2
-1	2	-1
2	-1	-1

d)

2	-1	-1
-1	2	-1
-1	-1	2



Canny

- Um dos melhores detectores de borda
- Visão geral do algoritmo Canny:
  - Filtro Gaussiano
  - Aplica Gradiente Gx e Gy (similar Sobel)
    - Busca a força e direção do gradiente
  - Elimina pixels não considerados borda
  - Aplicação de 2 thresholds
    - Se (pixel gradiente > upperThreshold) pixel é aceitável
    - Se (pixel gradiente < lowerThreshold) pixel é rejeitado
    - Se (pixel gradiente no intervalo [lower, upper]) pixel é aceitável se conectado a outro pixel passou no upper

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

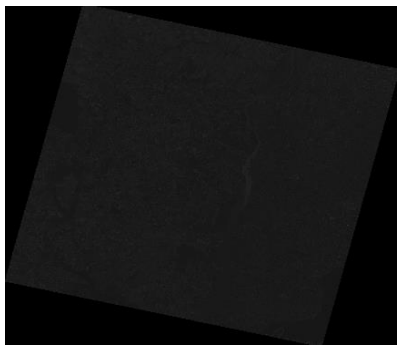
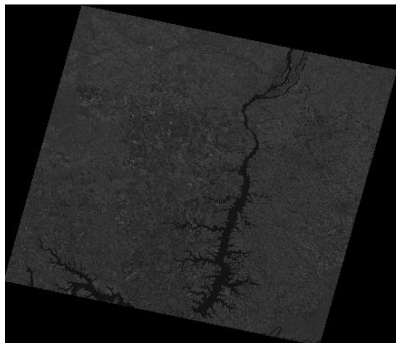
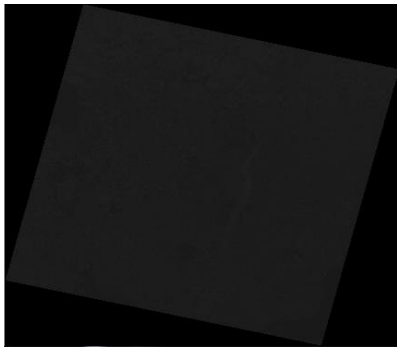
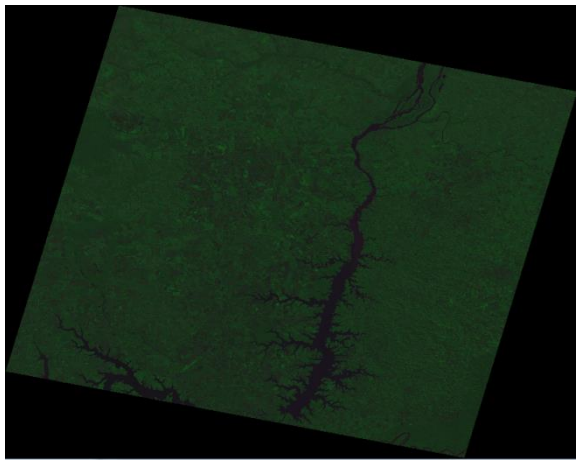
Canny

- Sintaxe:
  - void Canny(InputArray src, OutputArray dst, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false)
  - Threshold1 e Threshold2 → lower e upper Threshold
  - apertureSize → tamanho do Kernel do Sobel
  - L2gradient → true → $G = \sqrt{G_x^2 + G_y^2}$ false → $G = |G_x| + |G_y|$

```
Canny( image, cann, 50, 150, 3 );
```

Merge

- Cria uma imagem multicanal (24/32 bits) a partir de imagens de 8 bits **(inverso do split)**
- Sintaxe:
  - void merge(const Mat* mv, size_t count, OutputArray dst)
  - void merge(InputArrayOfArrays mv, OutputArray dst)

Merge


```
vector <Mat> canais (3);
Mat junta;
canais[0] = imread( "C:/imagensSatelite/LC82240772014154LGN00_B3.tif",
                    CV_LOAD_IMAGE_GRAYSCALE);;
canais[1] = imread( "C:/imagensSatelite/LC82240772014154LGN00_B5.tif",
                    CV_LOAD_IMAGE_GRAYSCALE);
canais[2] = imread( "C:/imagensSatelite/LC82240772014154LGN00_B4.tif",
                    CV_LOAD_IMAGE_GRAYSCALE);
merge(canais, junta);
namedWindow("Satelite", 0 );
namedWindow("banda3", 0);
namedWindow("banda4", 0);
namedWindow("banda5", 0);
imshow("Satelite", junta );
imshow("banda3", canais[0]);
imshow("banda4", canais[1]);
imshow("banda5", canais[2]);
waitKey(0);
```

Região de Interesse (ROI - "Region Of Interest“)

- Limita o processamento a uma região.
- Para limitar essa área é necessário:
  - Cria uma ROI com um Rect:

```
Rect area = Rect(3388, 6041, 200, 200);
```

  - Para se aplicar o ROI, basta informar o Rect a imagem

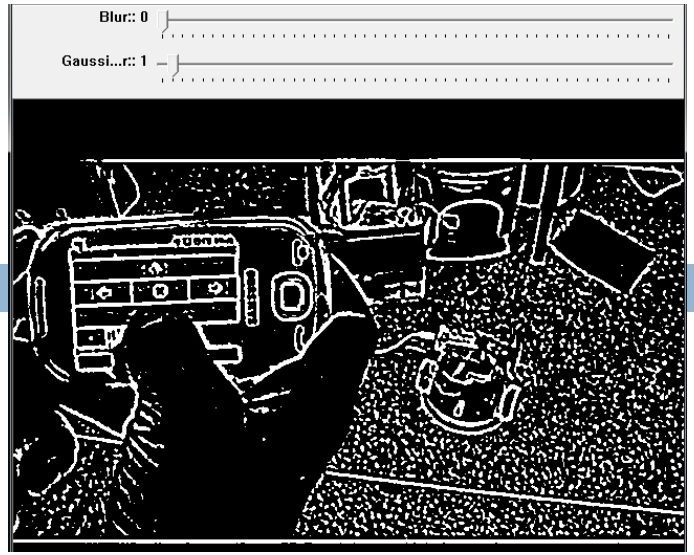
```
imshow("ROI", junta(area));
```

```
medianBlur( junta(area), aux, 3);
```

Referências

- CONCI, Aura; AZEVEDO, Eduardo; LETA, Fabiana R. **Computação Gráfica: Teoria e Prática – Volume 2**. Elsevier. Rio de Janeiro, 2008.
- FACON, Jacques. **Processamento e Análise de Imagens**. Material de Aula, Mestrado em Informática Aplicada – PUCPR – 2005.
- GONZALEZ, R. C.; WOODS, R. E. **Processamento de Imagens Digitais**. Edgard Blucher, 2000.
- PEDRINI, Hélio; SCHWARTZ, William R. **Análise de Imagens Digitais – Princípios, Algoritmos e Aplicações**. Thomson. São Paulo, 2008.
- http://www.dpi.inpe.br/spring/teoria/filtrage/filtragem.htm

Exercício



1. Abra um vídeo/câmera
    1. Capture a imagem
    2. Ao clicar o mouse na imagem original apresente a cor
    3. Crie 2 trackbars que disparam 2 filtros passa-baixa a escolha, em uma das bandas da imagem convertida
    4. Aplique um filtro passa-alta a escolha
    5. Binarize a imagem
    6. Apresente a imagem