Filtros espaciais

Prof. Luiz Otavio Murta Jr.
Depto. De Computação e Matemática
(FFCLRP/USP)

Propriedades

- **Operadores de suavização**
  - os elementos da máscara são positivos e somam um, de modo a que a saída é igual à entrada em regiões de constante intensidade
  - A quantidade de suavização e remoção de ruído é proporcional à dimensão da máscara
  - Transições abruptas (step edges) são tanto mais espalhadas (blurred) quanto maior for a dimensão da máscara
- **Operadores diferenciais**
  - as coordenadas das máscaras tem sinais opostos para que se obtenha uma resposta máxima quando existem transições de intensidade (contraste)
  - A soma dos valores é zero para que a resposta seja zero quando a região é constante
  - As máscaras de primeira derivada produzem valores absolutos elevados em pontos de grande contraste
  - As máscaras de segunda derivada produzem cruzamentos por zero em pontos de grande contraste

Suavização de imagem

- **Suavização (filtragem passa-baixas) de imagem**

$$O(r,c) = \left(\sum_{i=-N}^{N} \sum_{j=-N}^{N} I(r+i, c+j) \right) \Big/ N^2$$

  - filtro de média (*box filter*)

$$O(r,c) = \sum_{i=-N}^{N} \sum_{j=-N}^{N} g(i,j) I(r+i, c+j)$$

  - filtro gaussiano

$$g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{d^2}{2\sigma^2}}$$

$$d = \sqrt{(x - x_c)^2 + (y - y_c)^2}$$

Filtros espaciais

- Suavuzação (borramento)

Filtros espaciais

- Suavuzação (borramento)

$$I'(u,v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

Filtros espaciais

- Suavuzação (borramento)

$$I'(u,v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

$$\begin{aligned} I'(u,v) \leftarrow \tfrac{1}{9} \cdot [\, & I(u-1,v-1) + I(u,v-1) + I(u+1,v-1) + \\ & I(u-1,v) + I(u,v) + I(u+1,v) + \\ & I(u-1,v+1) + I(u,v+1) + I(u+1,v+1) \,] \end{aligned}$$

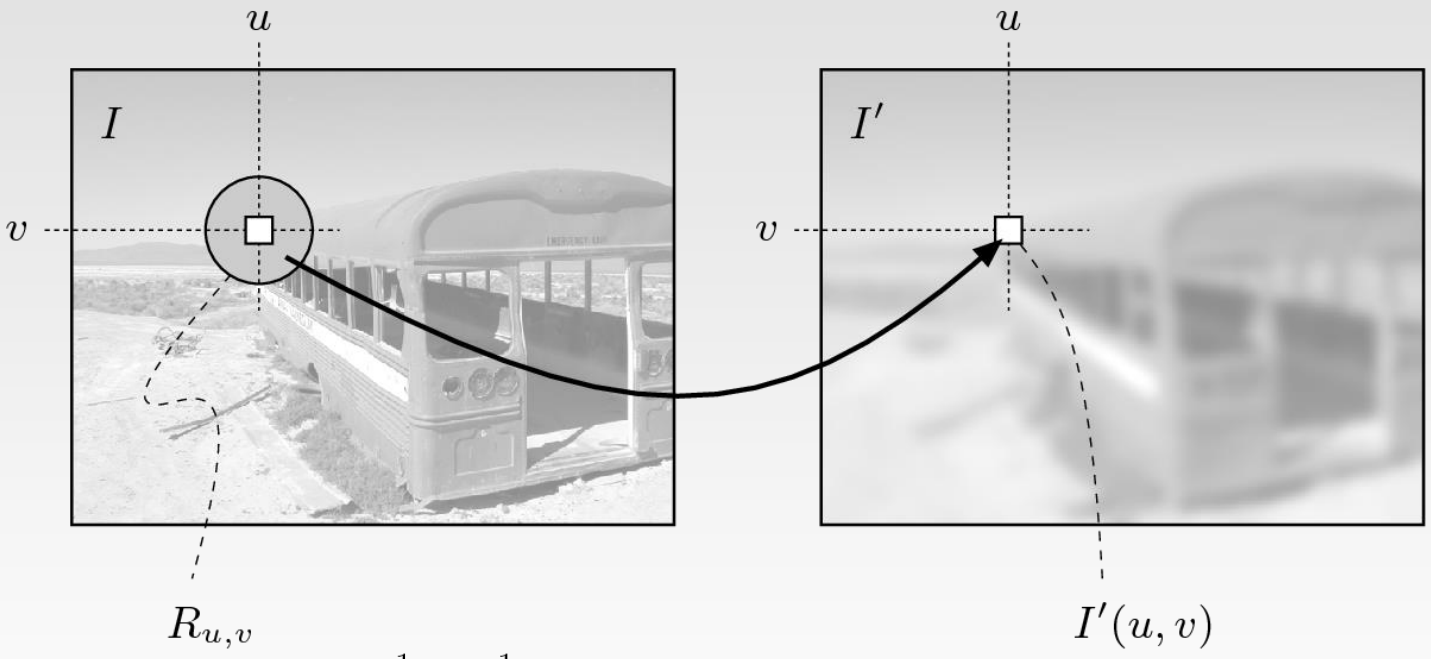
Filtros espaciais

- Suavuzação (borramento)

$$I'(u,v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(u+i, v+j)$$

Filtros espaciais

- Suavuzação (borramento)



$$I'(u,v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(u+i, v+j)$$

Filtros espaciais

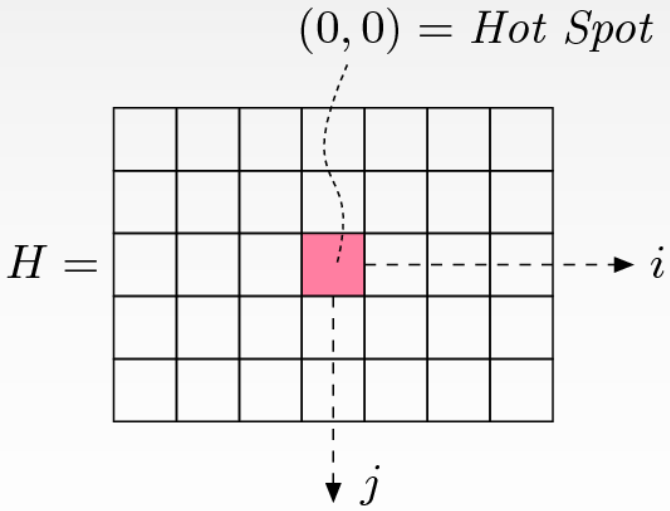
- Convolução 2D (bidimensional)

$$H(i,j) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$I'(u,v) \leftarrow \sum_{(i,j)\in R_H} I(u+i, v+j) \cdot H(i,j)$$

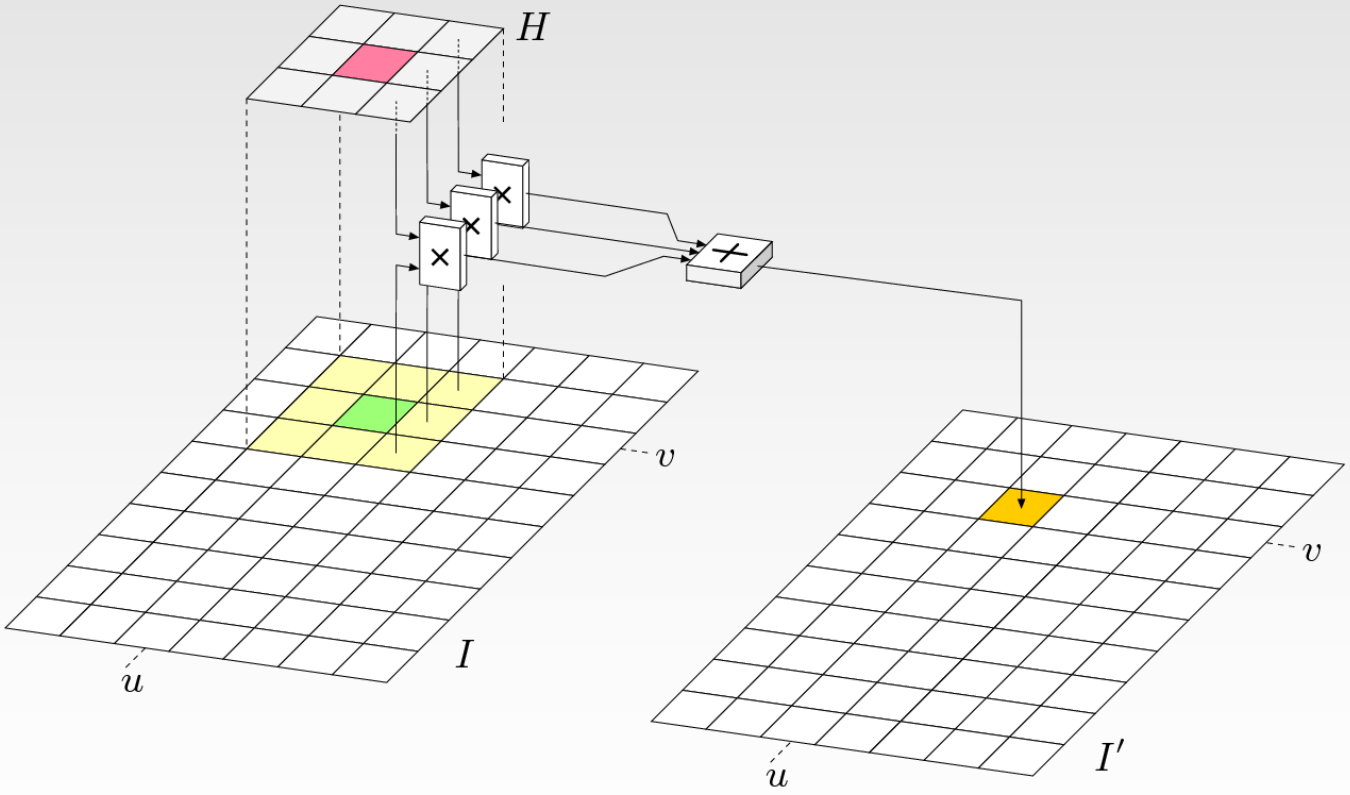
ou

$$I'(u,v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u+i, v+j) \cdot H(i,j)$$

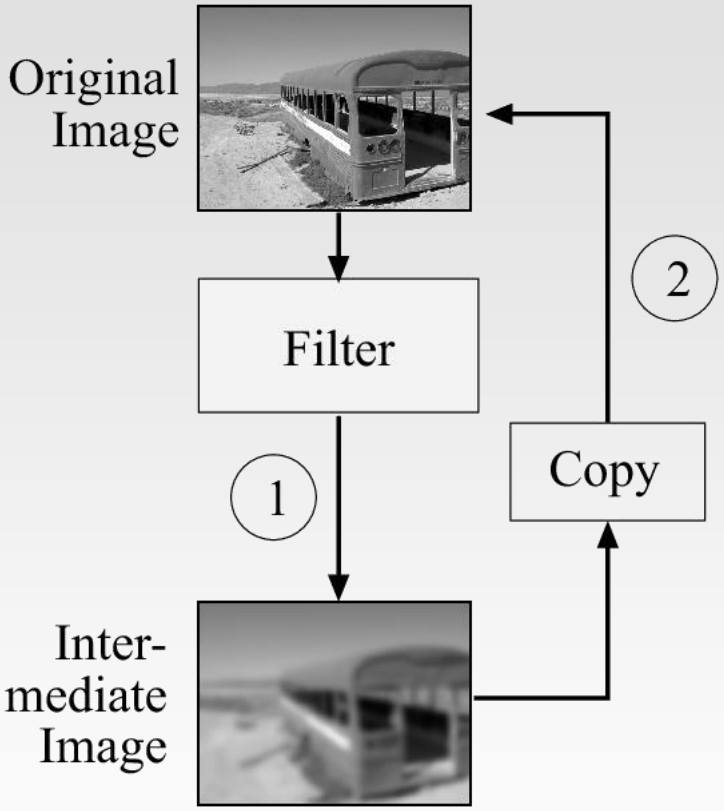


Filtros espaciais

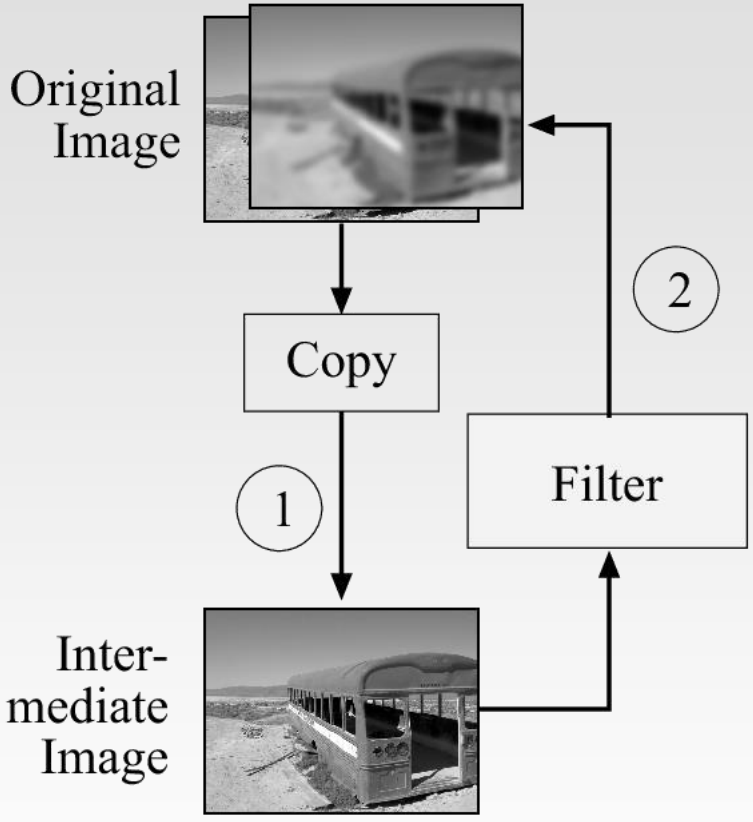
- Convolução 2D (bidimensional)



Filtros espaciais



Version A



Version B

Filtros espaciais

- Convolução 2D (bidimensional)

$$H(i,j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \mathbf{0.2} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

$$H(i,j) = s \cdot H'(i,j) \qquad \text{sendo} \qquad s = \frac{1}{\sum_{i,j} H'(i,j)}$$

Por exemplo:

$$H(i,j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \underline{0.200} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix} = \frac{1}{40} \begin{bmatrix} 3 & 5 & 3 \\ 5 & \underline{8} & 5 \\ 3 & 5 & 3 \end{bmatrix}$$

Filtros espaciais

```
1  import ij.*;
2  import ij.plugin.filter.PlugInFilter;
3  import ij.process.*;
4
5  public class Filter_Average3x3 implements PlugInFilter {
6      ...
7      public void run(ImageProcessor orig) {
8          int w = orig.getWidth();
9          int h = orig.getHeight();
10         ImageProcessor copy = orig.duplicate();
11
12         for (int v = 1; v <= h-2; v++) {
13             for (int u = 1; u <= w-2; u++) {
14                 //compute filter result for position (u,v)
15                 int sum = 0;
16                 for (int j = -1; j <= 1; j++) {
17                     for (int i = -1; i <= 1; i++) {
18                         int p = copy.getPixel(u+i, v+j);
19                         sum = sum + p;
20                     }
21                 }
22                 int q = (int) Math.round(sum/9.0);
23                 orig.putPixel(u, v, q);
24             }
25         }
26     }
27 } // end of class Filter_Average3x3
```

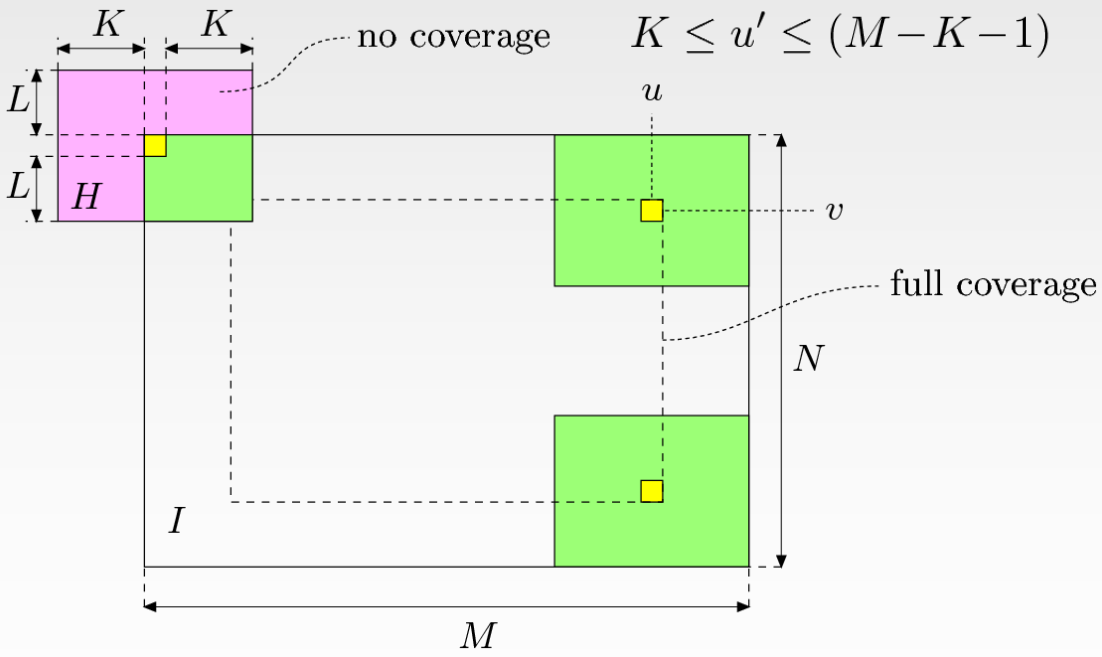
Filtros espaciais

Tratamento de fronteiras

$$I'(u,v) \leftarrow \text{Offset} + \frac{1}{\text{Scale}} \sum_{j=-2}^{j=2} \sum_{i=-2}^{i=2} I(u+i, v+j) \cdot H(i,j)$$, caso geral:

$$I(u,v) \quad \text{with} \quad 0 \le u < M \quad \text{and} \quad 0 \le v < N$$

$$K \le u' \le (M-K-1) \qquad \text{and} \qquad L \le v' \le (N-L-1)$$



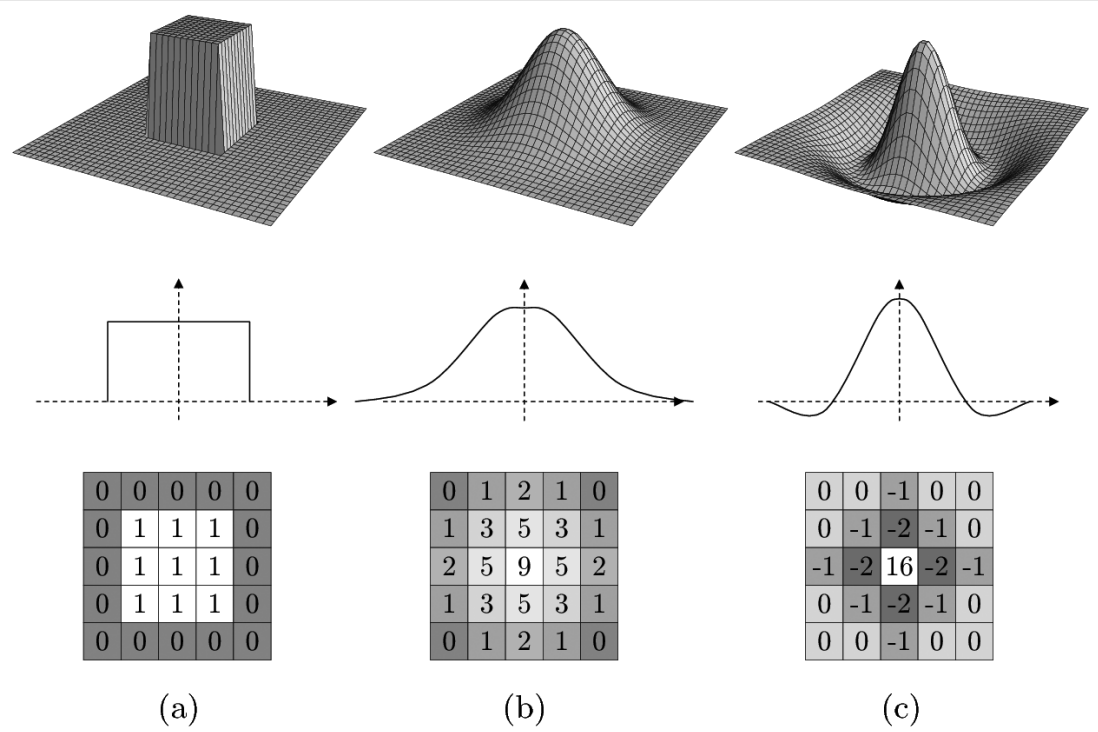
Filtros espaciais

```
1   public void run(ImageProcessor orig) {
2       int w = orig.getWidth();
3       int h = orig.getHeight();
4       // 3 × 3 filter matrix
5       double[][] filter = {
6           {0.075, 0.125, 0.075},
7           {0.125, 0.200, 0.125},
8           {0.075, 0.125, 0.075}
9       };
10      ImageProcessor copy = orig.duplicate();
11
12      for (int v = 1; v <= h-2; v++) {
13          for (int u = 1; u <= w-2; u++) {
14              // compute filter result for position (u, v)
15              double sum = 0;
16              for (int j = -1; j <= 1; j++) {
17                  for (int i = -1; i <= 1; i++) {
18                      int p = copy.getPixel(u+i, v+j);
19                      // get the corresponding filter coefficient:
20                      double c = filter[j+1][i+1];
21                      sum = sum + c * p;
22                  }
23              }
24              int q = (int) Math.round(sum);
25              orig.putPixel(u, v, q);
26          }
27      }
28  }
```

Filtros espaciais

Mascaras (H):

a) quadrada; b) gaussiana; c) LoG



0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

(a)

0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

(b)

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

(c)

Filtros espaciais

Mascaras (H):

$$I'(u,v) = \sum_{(i,j)\in R_H^+} I(u+i, v+j)\cdot |H(i,j)| - \sum_{(i,j)\in R_H^-} I(u+i, v+j)\cdot |H(i,j)|$$

De fato, a equação pode assumir a forma geral:

$$I'(u,v) = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} I(u-i, v-j)\cdot H(i,j) \quad ; \quad I' = I * H$$

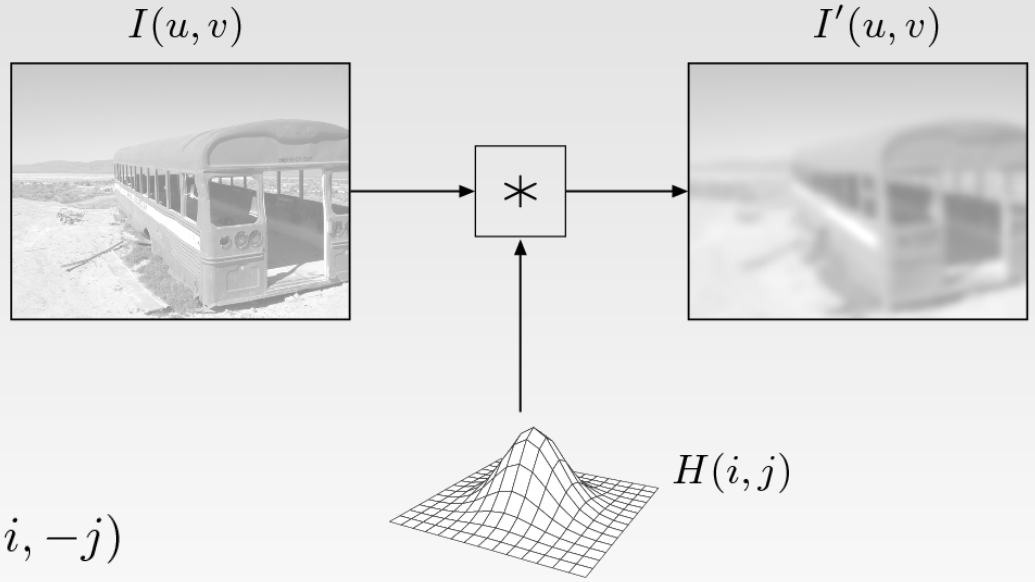
$$I'(u,v) = \sum_{(i,j)\in R_H} I(u-i, v-j)\cdot H(i,j)$$

$$= \sum_{(i,j)\in R_H} I(u+i, v+j)\cdot H(-i,-j)$$

$$= \sum_{(i,j)\in R_H} I(u+i, v+j)\cdot H^*(i,j).$$

Filtros espaciais

Propriedades da convolução 2D



$$H^*(i,j) = H(-i,-j)$$

$$I * H = H * I$$

$$(s \cdot I) * H \quad = \quad I * (s \cdot H) \quad = \quad s \cdot (I * H)$$

Filtros espaciais

Propriedades da convolução 2D

$$H^*(i,j) = H(-i,-j)$$

$$I * H = H * I$$

$$(s \cdot I) * H \quad = \quad I * (s \cdot H) \quad = \quad s \cdot (I * H)$$

$$(I_1 + I_2) * H \quad = \quad (I_1 * H) + (I_2 * H)$$

$$A * (B * C) = (A * B) * C$$

$$\begin{aligned} I * H &= I * (H_1 * H_2 * \ldots * H_n) \\ &= \big(\ldots((I * H_1) * H_2) * \ldots * H_n\big) \end{aligned}$$

Filtros espaciais

Separabilidade da convolução 2D

Sejam:

$$H_1 = [11111] \quad \text{e} \quad H_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

dada a propriedade:

$$H = H_1 * H_2 * \ldots * H_n$$

Podemos considerar:

$$I' \leftarrow (I * H_x) * H_y = I * \underbrace{(H_x * H_y)}_{H_{xy}}$$

Filtros espaciais

Separabilidade da convolução 2D

Sejam:

$$H_1 = [11111] \quad \text{e} \quad H_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

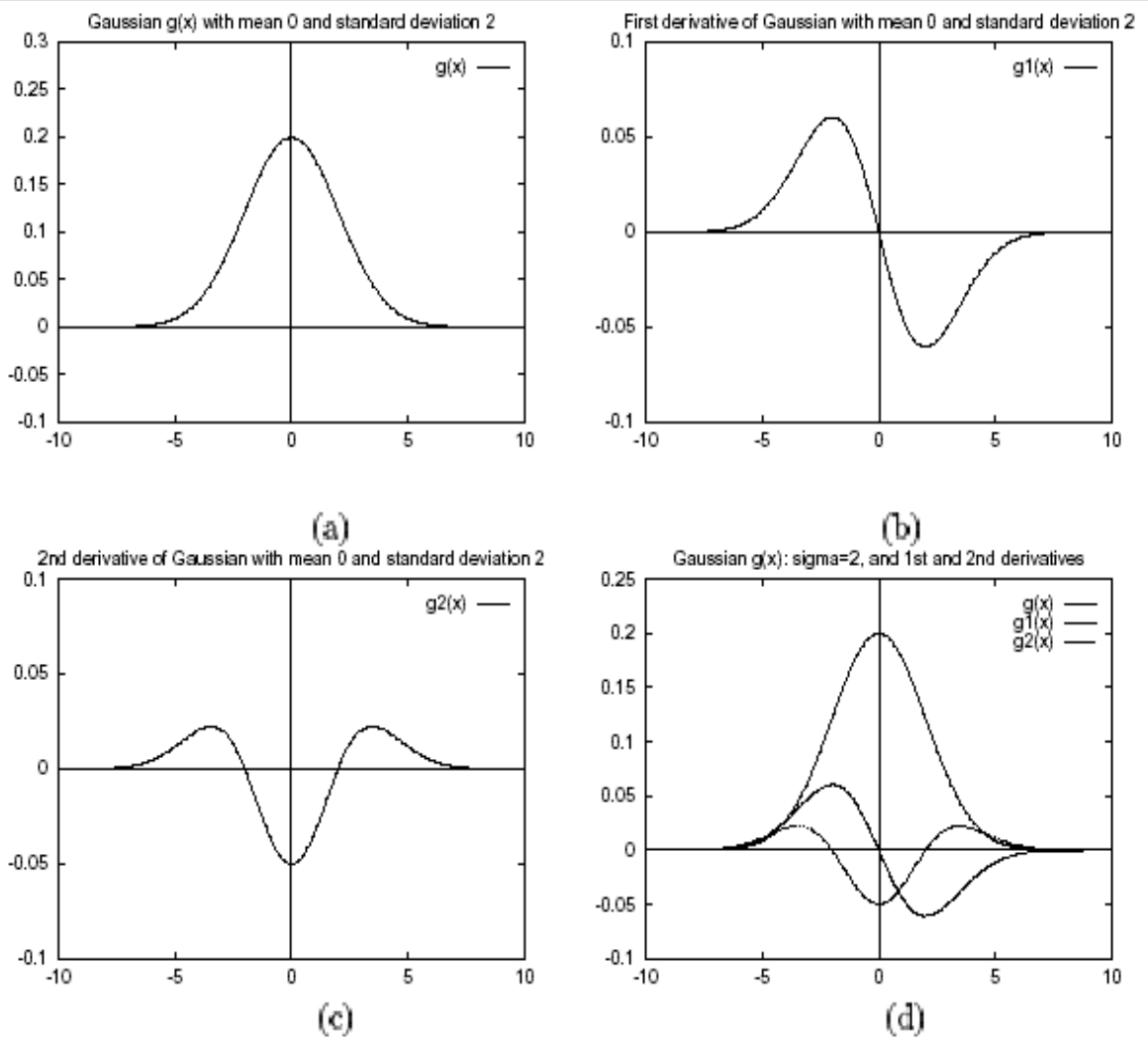
Podemos considerar:

$$I' \leftarrow (I * H_x) * H_y = I * \underbrace{(H_x * H_y)}_{H_{xy}}$$

Então:

$$H_{xy} = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtros gaussianos



Caso 1D:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

$$g'(x) = \frac{-x}{\sigma^2} g(x)$$

$$g''(x) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right) g(x)$$

Caso 2D:

$$h(x,y) = g(r)$$

$$r = \sqrt{x^2 + y^2}$$

Filtros espaciais

Separabilidade da convolução 2D **(H Gaussiano)**

Seja:

$$H_{x,y}(i,j) = (H_x \otimes H_y)(i,j) = H_x(i) \cdot H_y(j)$$

Podemos considerar:

$$G_\sigma(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} = e^{-\frac{x^2}{2\sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}} = g_\sigma(x) \cdot g_\sigma(y)$$

Então:

$$I' \leftarrow I * H^{G,\sigma} = I * H_x^{G,\sigma} * H_y^{G,\sigma}$$

Filtros espaciais

Kernel Gaussiano **(H Gaussiano)**

```
1   float[] makeGaussKernel1d(double sigma) {
2
3     // create the kernel
4     int center = (int) (3.0*sigma);
5     float[] kernel = new float[2*center+1]; // odd size
6
7     // fill the kernel
8     double sigma2 = sigma * sigma;          // σ^2
9     for (int i=0; i<kernel.length; i++) {
10      double r = center - i;
11      kernel[i] = (float) Math.exp(-0.5 * (r*r) / sigma2);
12    }
13
14    return kernel;
15  }
```

Filtros espaciais

Convolução 2D **(Função Delta)**

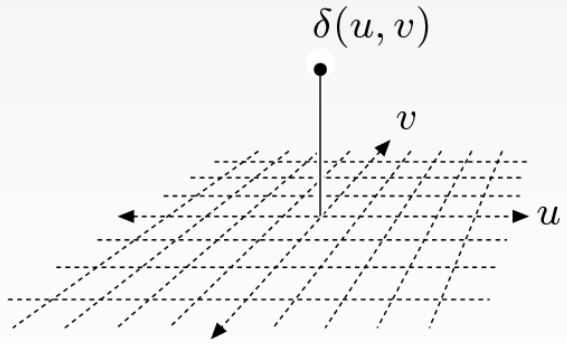
Seja:

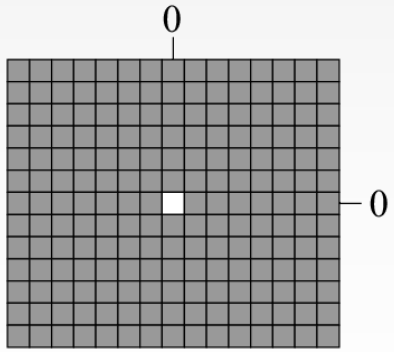
$$\delta(u,v) = \begin{cases} 1 & \text{for } u = v = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Sabemos que:

$$I * \delta = I$$

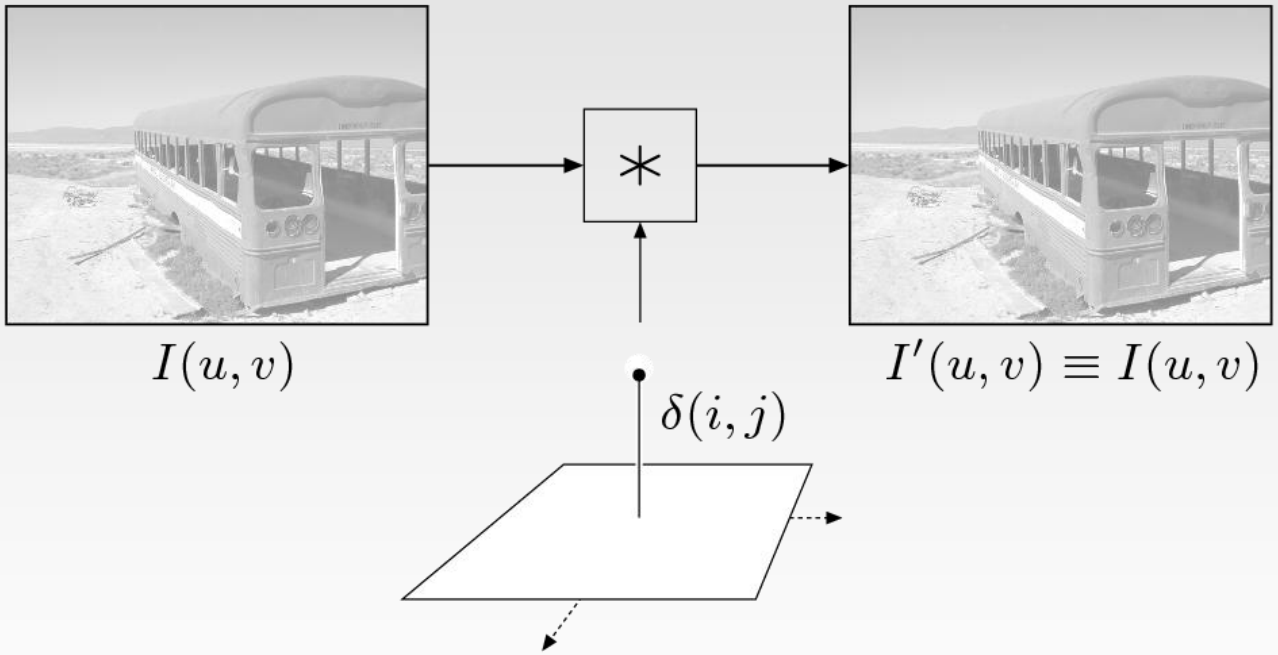
$$H * \delta = \delta * H = H$$





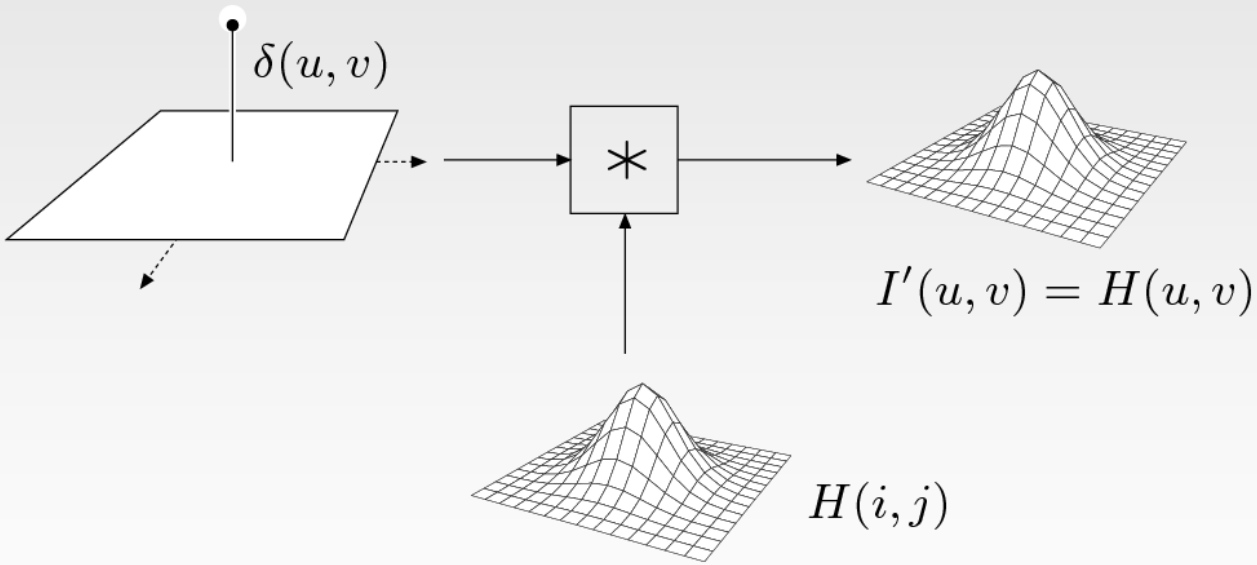
Filtros espaciais

Convolução 2D **(Função Delta)**



Filtros espaciais

Convolução 2D **(Função Delta e Kernel Gaussiano)**



Filtros espaciais

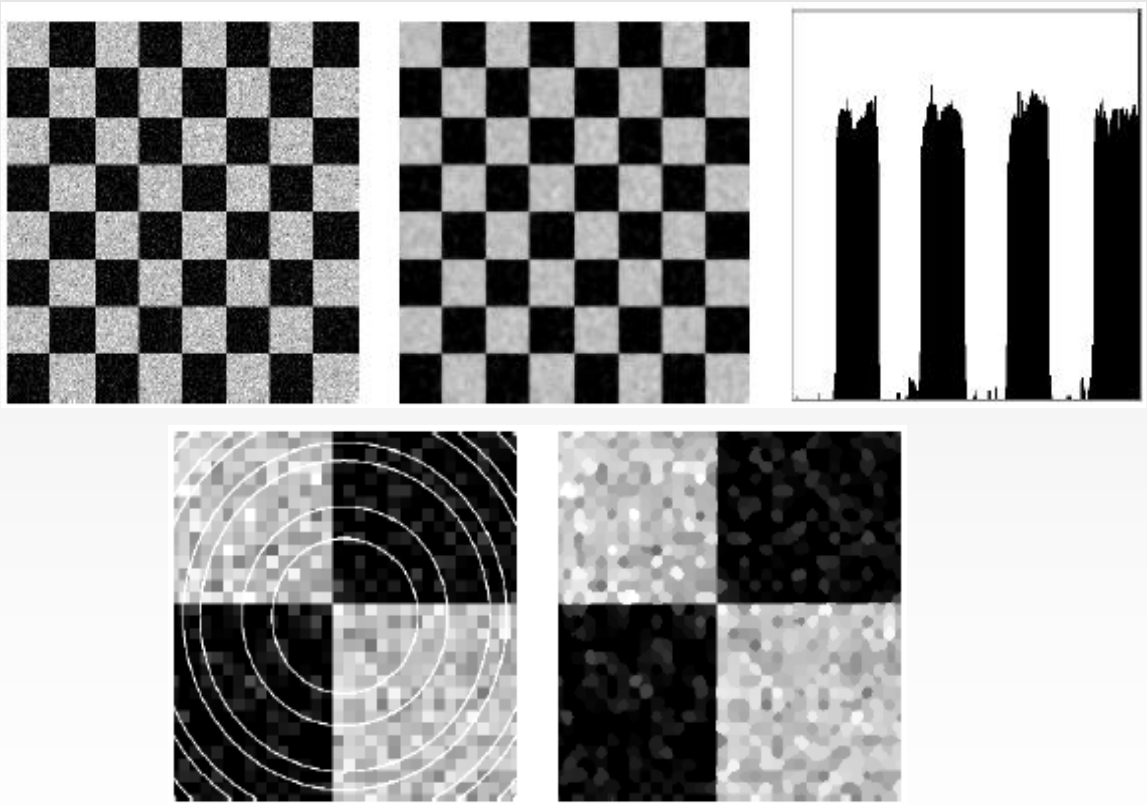
Filtro de mediana

(a) (b) (c)

Filtragem de mediana

- **Seja $A[i]_{i=0,\cdots,n-1}$ uma lista ordenada de números reais. A *mediana* do conjunto A é o valor A[(n-1)/2]**

  - Exemplos



Filtros espaciais

Filtro de mediana

$$\text{median}\,(p_0, p_1, \ldots, p_K, \ldots, p_{2K}) \triangleq p_K$$

$$\text{median}\,(p_0, \ldots, p_{K-1}, p_K, \ldots, p_{2K-1}) \triangleq (p_{K-1} + p_K)\,/\,2$$

$$I'(u,v) \leftarrow \text{median}\,\{I(u+i, v+j) \mid (i,j) \in R\}$$

Filtros espaciais

Filtro de mediana

$$\text{median}\,(p_0, p_1, \ldots, p_K, \ldots, p_{2K}) \triangleq p_K$$

$$\text{median}\,(p_0, \ldots, p_{K-1}, p_K, \ldots, p_{2K-1}) \triangleq (p_{K-1} + p_K)\,/\,2$$

$$I'(u,v) \leftarrow \text{median}\,\{I(u+i, v+j) \mid (i,j) \in R\}$$

Filtros espaciais

Filtro de mediana

(a) (b) (c)

Filtragem temporal com filtro de mediana

Filtragem de Mediana da Sequência

Filtros espaciais

Filtro de mediana

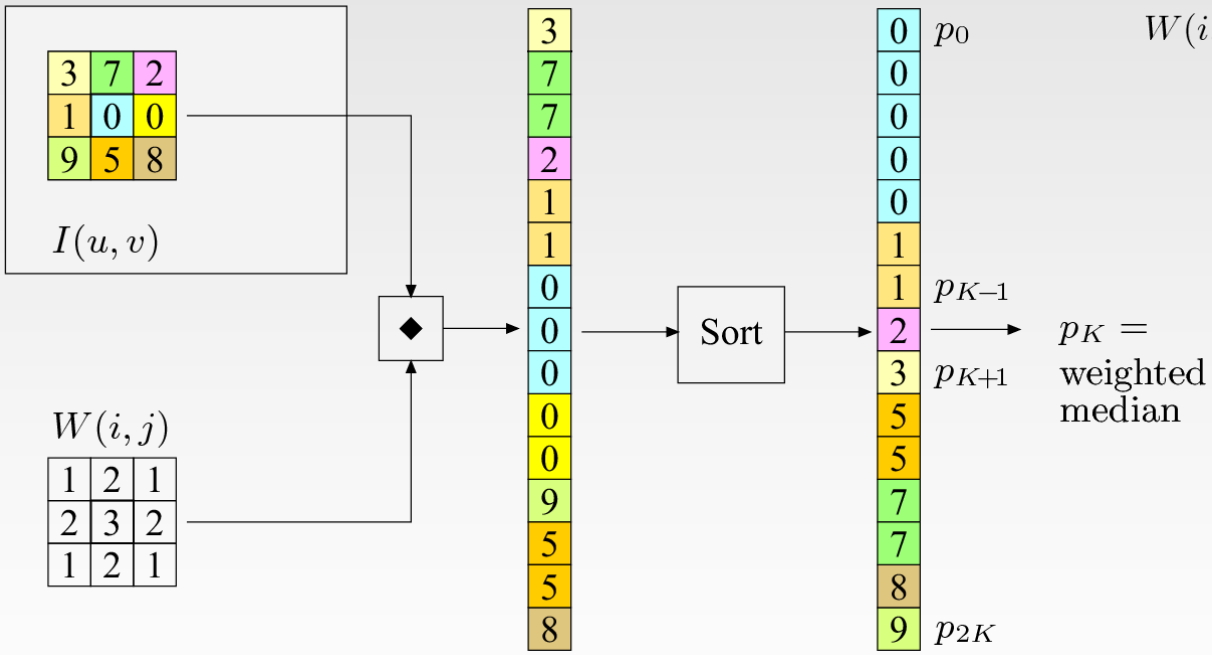
```
1  import ij.*;
2  import ij.plugin.filter.PlugInFilter;
3  import ij.process.*;
4  import java.util.Arrays;
5
6  public class Filter_Median3x3 implements PlugInFilter {
7      final int K = 4; // filter size
8
9      public void run(ImageProcessor orig) {
10         int w = orig.getWidth();
11         int h = orig.getHeight();
12         ImageProcessor copy = orig.duplicate();
13
14         // vector to hold pixels from 3×3 neighborhood
15         int[] P = new int[2*K+1];
16
17         for (int v = 1; v <= h-2; v++) {
18             for (int u = 1; u <= w-2; u++) {
19                 // fill the pixel vector P for filter position u, v
20                 int k = 0;
21                 for (int j = -1; j <= 1; j++) {
22                     for (int i = -1; i <= 1; i++) {
23                         P[k] = copy.getPixel(u+i, v+j);
24                         k++;
25                     }
26                 }
27                 // sort pixel vector and take the center element
28                 Arrays.sort(P);
29                 orig.putPixel(u, v, P[K]);
30             }
31         }
32     }
33
34 } // end of class Filter_Median3x3
```

Filtros espaciais

Filtro de mediana ponderada

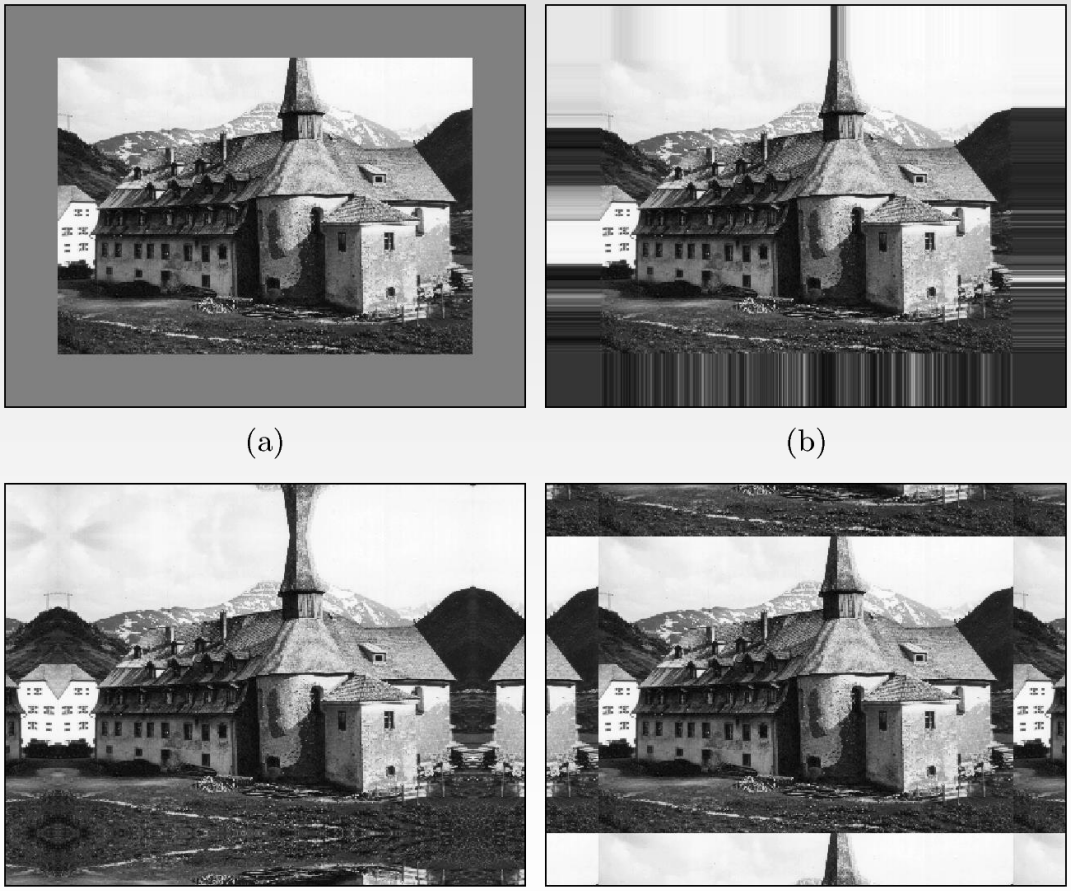
$$W^{+}(i,j) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$W(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 2 & \mathbf{3} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Filtros espaciais

Tratamento de fronteiras



(a) (b)

(c) (d)

Filtros espaciais

Implementação dos filtros

```
1  import ij.plugin.filter.GaussianBlur;
2  ...
3  public void run(ImageProcessor ip) {
4    GaussianBlur gb = new GaussianBlur();
5    double radius = 2.5;
6    gb.blur(ip, radius);
7  }
```

```
1  import ij.plugin.filter.Convolver;
2  ...
3  public void run(ImageProcessor I) {
4    float[] H = {            // filter array is one-dimensional!
5        0.075f, 0.125f, 0.075f,
6        0.125f, 0.200f, 0.125f,
7        0.075f, 0.125f, 0.075f };
8    Convolver cv = new Convolver();
9    cv.setNormalize(false);   // do not use filter normalization
10   cv.convolve(I, H, 3, 3); // apply the filter H to I
11 }
```

$$H(i,j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \mathbf{0.2} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

Filtros espaciais

Tratamento de fronteiras

```
1  import ij.plugin.filter.GaussianBlur;
2  ...
3  public void run(ImageProcessor ip) {
4    GaussianBlur gb = new GaussianBlur();
5    double radius = 2.5;
6    gb.blur(ip, radius);
7  }
```

```
1  import ij.plugin.filter.Convolver;
2  ...
3  public void run(ImageProcessor I) {
4    float[] H = {              // filter array is one-dimensional!
5        0.075f, 0.125f, 0.075f,
6        0.125f, 0.200f, 0.125f,
7        0.075f, 0.125f, 0.075f };
8    Convolver cv = new Convolver();
9    cv.setNormalize(false);   // do not use filter normalization
10   cv.convolve(I, H, 3, 3); // apply the filter H to I
11 }
```

Propriedades

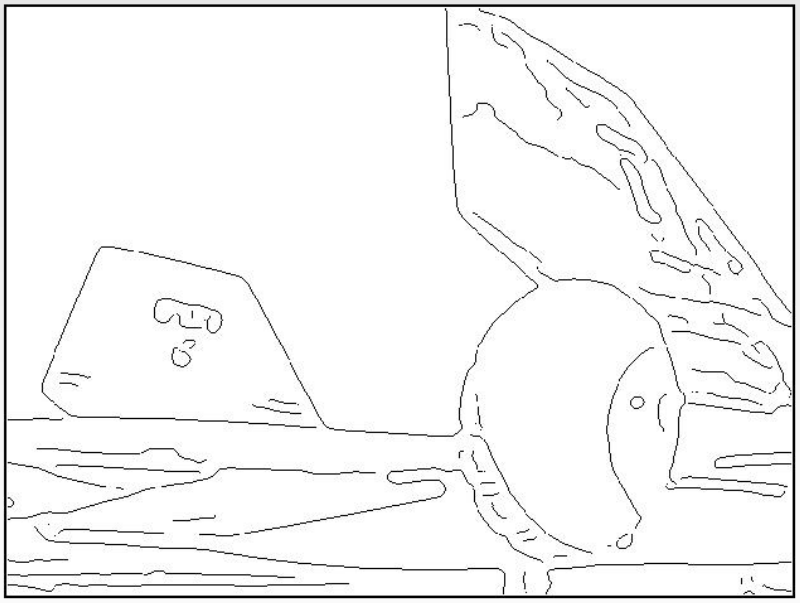
- **Operadores de suavização**
  - os elementos da máscara são positivos e somam um, de modo a que a saída é igual à entrada em regiões de constante intensidade
  - A quantidade de suavização e remoção de ruído é proporcional à dimensão da máscara
  - Transições abruptas (step edges) são tanto mais espalhadas (blurred) quanto maior for a dimensão da máscara
- **Operadores diferenciais**
  - as coordenadas das máscaras tem sinais opostos para que se obtenha uma resposta máxima quando existem transições de intensidade (contraste)
  - A soma dos valores é zero para que a resposta seja zero quando a região é constante
  - As máscaras de primeira derivada produzem valores absolutos elevados em pontos de grande contraste
  - As máscaras de segunda derivada produzem cruzamentos por zero em pontos de grande contraste

Operadores diferenciais 2D

Derivada em relação a **x**:

$$f'(x) = \frac{df}{dx}(x)$$

(a)



(b)

Máscaras 1D

mask $\mathbf{M} = [-1, 2, -1]$

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	0	0	0	0	-12	12	0	0	0	0

(a) S_1 is an upward step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	$\otimes$	M	0	0	0	0	12	-12	0	0	0	0

(b) S_2 is a downward step edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	$\otimes$	M	0	0	0	-3	0	0	0	3	0	0

(c) S_3 is an upward ramp

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	0	0	0	-12	24	-12	0	0	0	0

(d) S_4 is a bright impulse or "line"

box smoothing mask $\mathbf{M} = [1/3, 1/3, 1/3]$

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	12	12	12	12	16	20	24	24	24	24

(a) S_1 is an upward step edge

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	12	12	12	16	16	16	12	12	12	12

(d) S_4 is a bright impulse or "line"

Gaussian smoothing mask $\mathbf{M} = [1/4, 1/2, 1/4]$

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	12	12	12	12	15	21	24	24	24	24

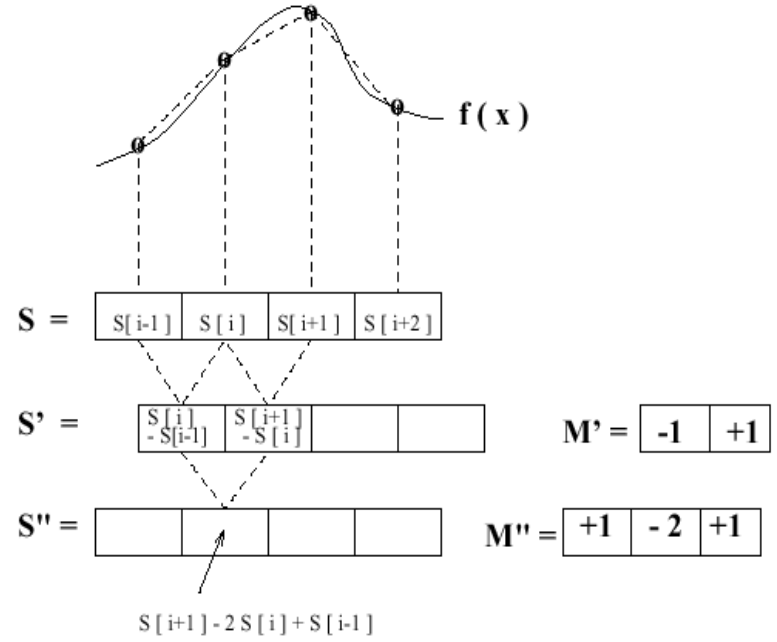
(a) S_1 is an upward step edge

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	12	12	12	15	18	15	12	12	12	12

Detecção de transições (*bordas*)

- **Operadores diferenciais de sinais 1D**

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$



Máscara 1D centrada

mask M = [−1, 0, 1]

S_1			12	12	12	12	12	24	24	24	24	24
S_1	⊗	M	0	0	0	0	12	12	0	0	0	0

(a) S_1 is an upward step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	⊗	M	0	0	0	0	-12	-12	0	0	0	0

(b) S_2 is a downward step edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	⊗	M	0	0	0	3	6	6	6	3	0	0

(c) S_3 is an upward ramp

S_4			12	12	12	12	24	12	12	12	12	12
S_4	⊗	M	0	0	0	12	0	-12	0	0	0	0

(d) S_4 is a bright impulse or "line"

Operadores diferenciais 2D

- **Gradiente duma função**

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

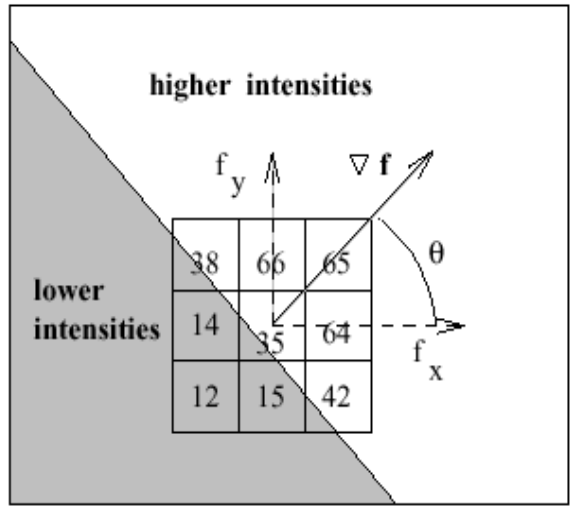
$$|\nabla f| \approx \sqrt{f_x^2 + f_y^2}$$

$$\theta \approx \tan^{-1}(f_y / f_x)$$

$$\frac{\partial f}{\partial x} \equiv f_x \approx \frac{1}{3}[(I(x+1,y) - I(x-1,y))/2 + (I(x+1,y-1) - I(x-1,y-1))/2 + (I(x+1,y+1) - I(x-1,y+1))/2]$$

$$\frac{\partial f}{\partial y} \equiv f_y \approx \frac{1}{3}[(I(x,y+1) - I(x,y-1))/2 + (I(x-1,y+1) - I(x-1,y-1))/2 + (I(x+1,y+1) - I(x+1,y-1))/2]$$

Exemplo:



f_y = ((38-12)/2 + (66-15)/2 + (65-42)/2) / 3
= (13 + 25 + 11) / 3 = 16

f_x = ((65-38)/2 + (64-14)/2 + (42-12)/2) / 3
= (13 + 25 + 15) / 3 = 18

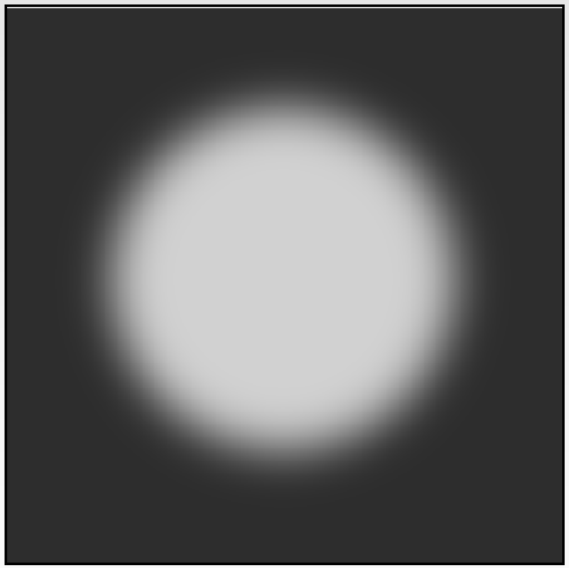
$\theta = \tan^{-1}(16 / 18) = 0.727$ rad
= 42 degrees

$|\nabla f| = (16^2 + 18^2)^{1/2} = 24$

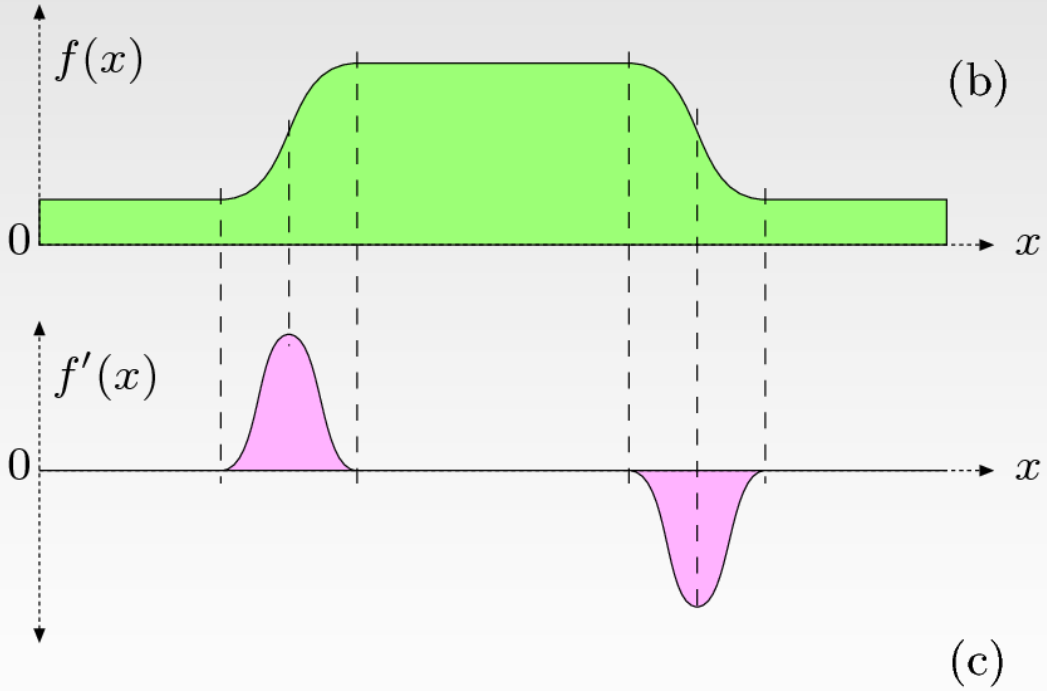
Operadores diferenciais 2D

Derivada em relação a **x**:

$$f'(x) = \frac{df}{dx}(x)$$



(a)



Operadores diferenciais 2D

Derivada em relação a **x**: $f'(x) = \frac{df}{dx}(x)$

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 \cdot \big(f(u+1) - f(u-1)\big)$$

Estendendo:

$\frac{\partial I}{\partial u}(u,v)$ e $\frac{\partial I}{\partial u}(u,v)$

Então o gradiente é:

$$\nabla I(u,v) = \begin{bmatrix} \frac{\partial I}{\partial u}(u,v) \\ \frac{\partial I}{\partial v}(u,v) \end{bmatrix}$$

Operadores diferenciais 2D

Gradiente:

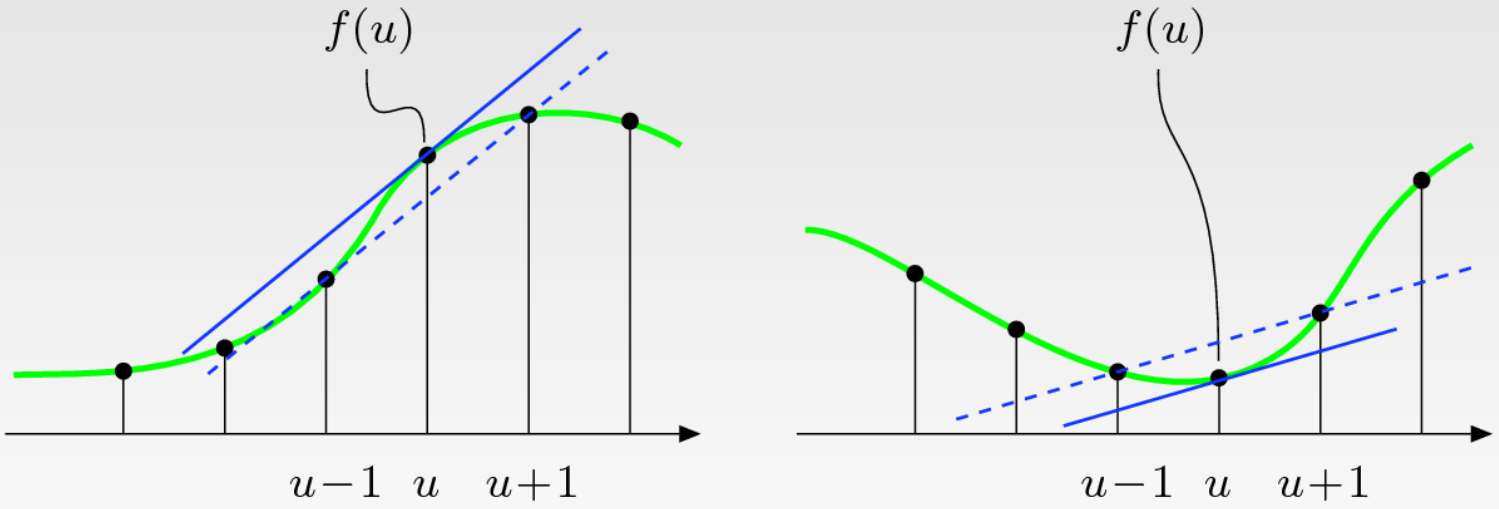
$$\nabla I(u,v) = \begin{bmatrix} \frac{\partial I}{\partial u}(u,v) \\ \frac{\partial I}{\partial v}(u,v) \end{bmatrix}$$

Módulo do gradiente:

$$|\nabla I|(u,v) = \sqrt{\left(\frac{\partial I}{\partial u}(u,v)\right)^2 + \left(\frac{\partial I}{\partial v}(u,v)\right)^2}$$

Operadores diferenciais 2D

Calculo do gradiente:



Operadores diferenciais 2D

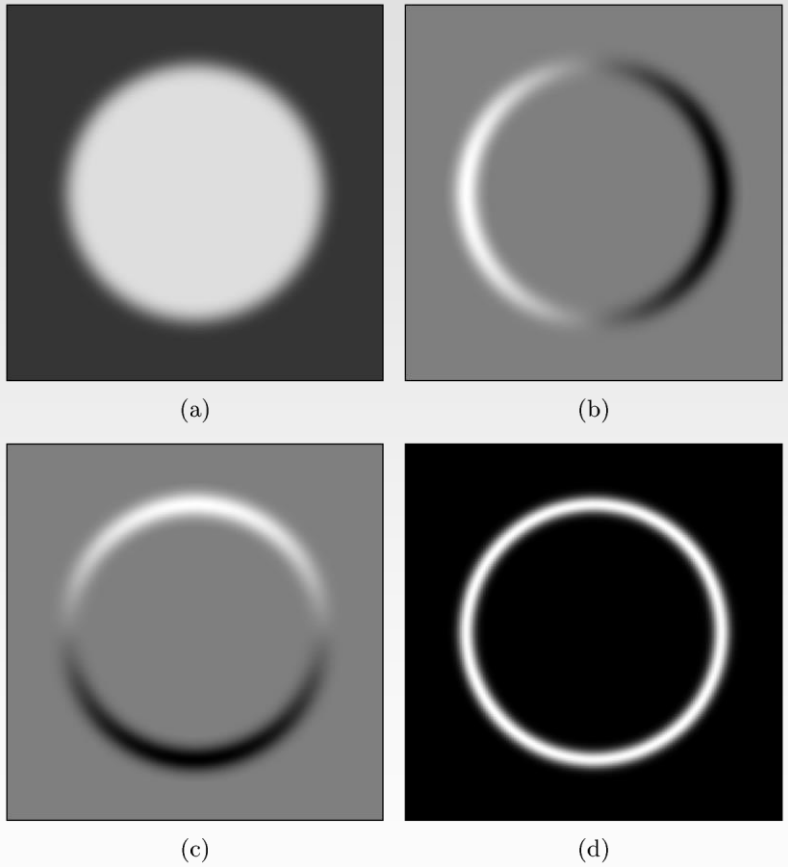
Calculo do gradiente:

•em x

$$H_x^D = \begin{bmatrix} -0.5 & \mathbf{0} & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & \mathbf{0} & 1 \end{bmatrix}$$

•em y

$$H_y^D = \begin{bmatrix} -0.5 \\ \mathbf{0} \\ 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ \mathbf{0} \\ 1 \end{bmatrix}$$



Operadores diferenciais 2D

Kernels diferenciais (*H*) **Prewitt**:

$$H_x^P = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \text{e} \quad H_x^P = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{e} \quad H_x^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Operadores diferenciais 2D

Kernels diferenciais (*H*) **Sobel**:

$$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{e} \quad H_x^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

O gradiente é:

$$\nabla I(u,v) \approx \frac{1}{8}\begin{bmatrix} (I * H_x^S)(u,v) \\ (I * H_y^S)(u,v) \end{bmatrix}$$

Lembrando que: $E(u,v) = \sqrt{\left(D_x(u,v)\right)^2 + \left(D_y(u,v)\right)^2}$

Operadores diferenciais 2D

O mesmo vale para **Prewitt**:

$$\nabla I(u,v) \approx \frac{1}{6} \cdot \begin{bmatrix} \left(I * H_x^P\right)(u,v) \\ \left(I * H_y^P\right)(u,v) \end{bmatrix}$$

e $$E(u,v) = \sqrt{\left(D_x(u,v)\right)^2 + \left(D_y(u,v)\right)^2} \quad ,$$

$$\Phi(u,v) = \tan^{-1}\left(\frac{D_y(u,v)}{D_x(u,v)}\right) = \mathrm{ArcTan}\left(D_x(u,v), D_y(u,v)\right)$$

Operadores diferenciais 2D

O mesmo vale ainda para **Roberts**:

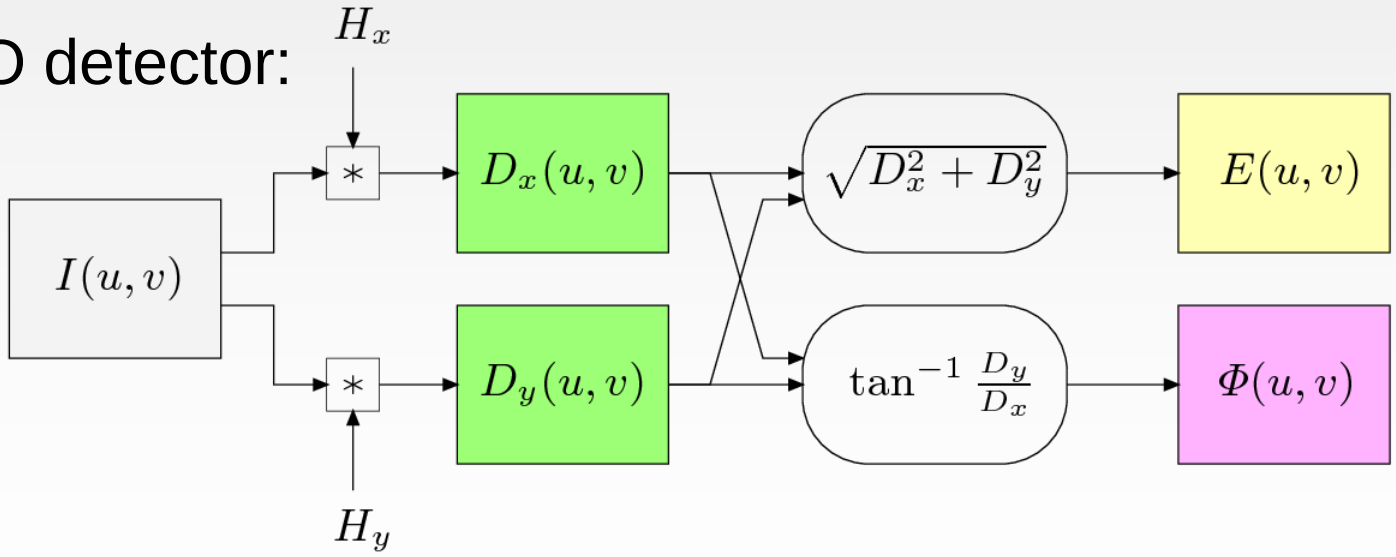
$$H_1^R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{e} \quad H_2^R = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Operadores diferenciais 2D

O mesmo vale ainda para **Roberts**:

$$H_1^R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{e} \quad H_2^R = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

O detector:



Detectores de bordas

Prewitt: $M_x =$

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

; $M_y =$

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

Sobel: $M_x =$

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

; $M_y =$

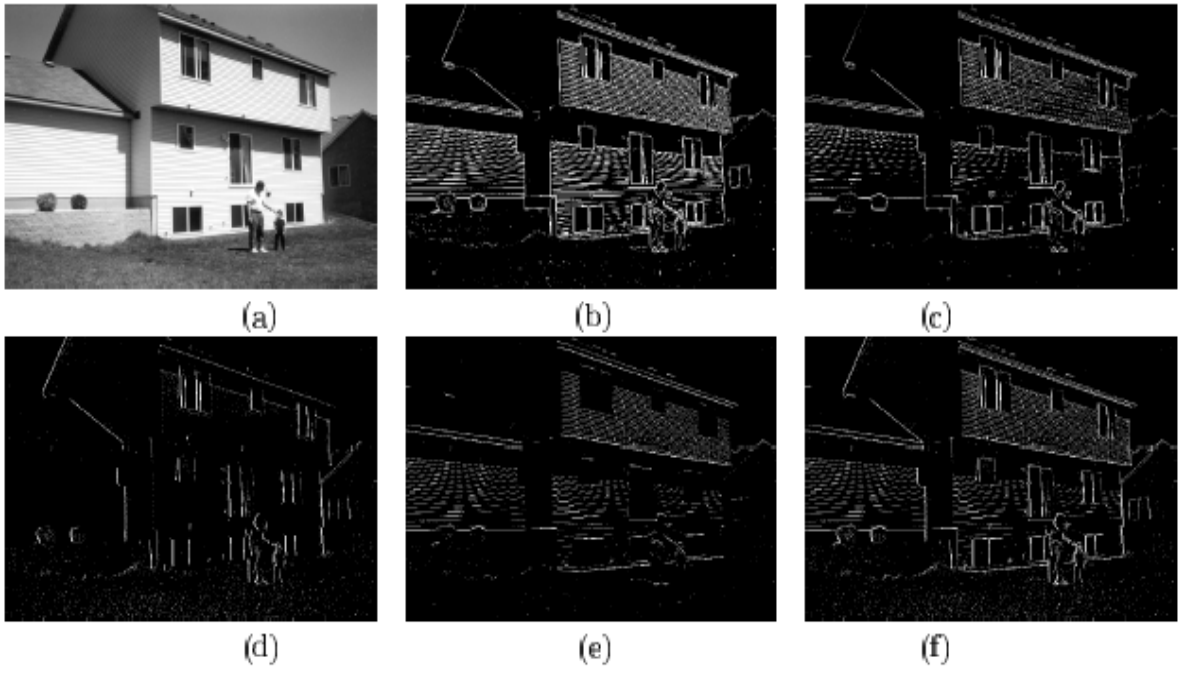
1	2	1
0	0	0
-1	-2	-1

Roberts: $M_x =$

0	1
-1	0

; $M_y =$

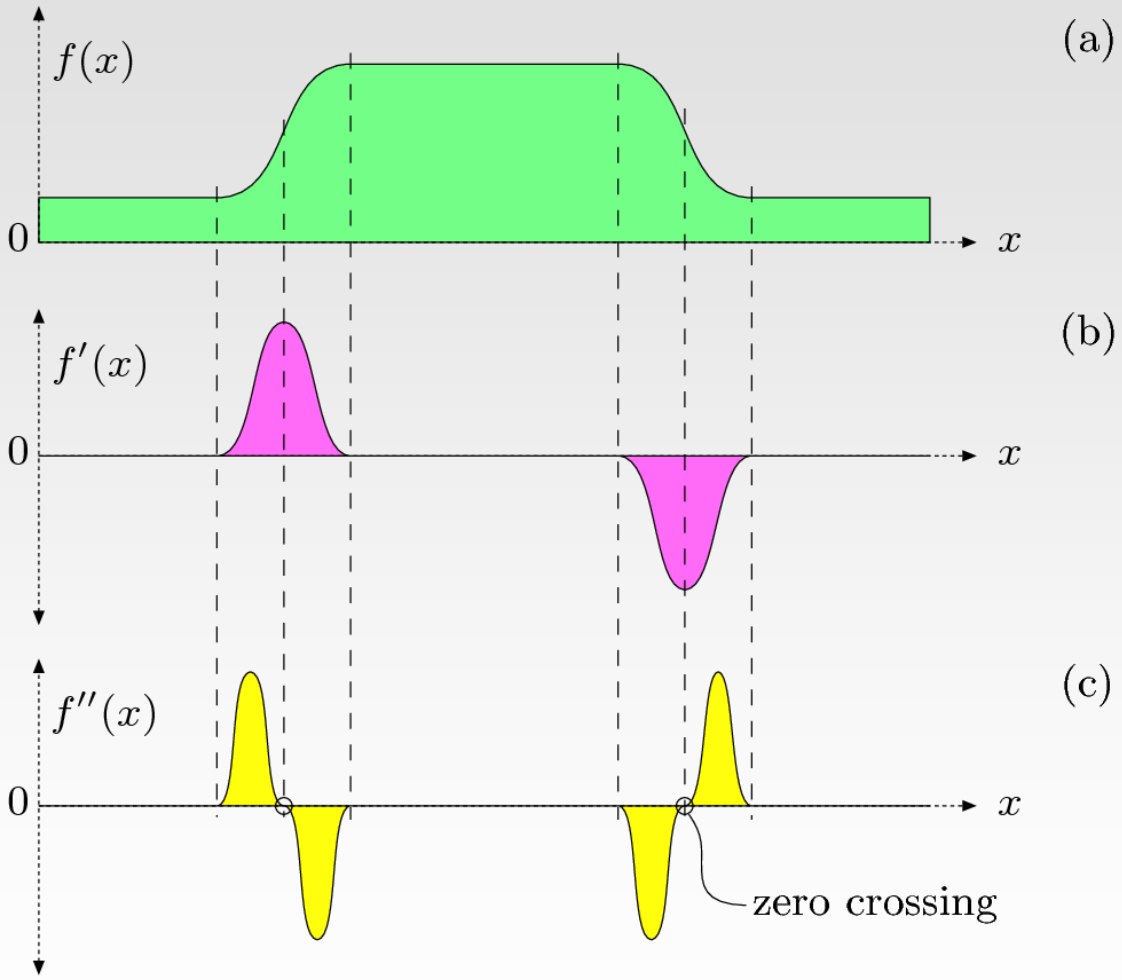
1	0
0	-1



(a) (b) (c)

(d) (e) (f)

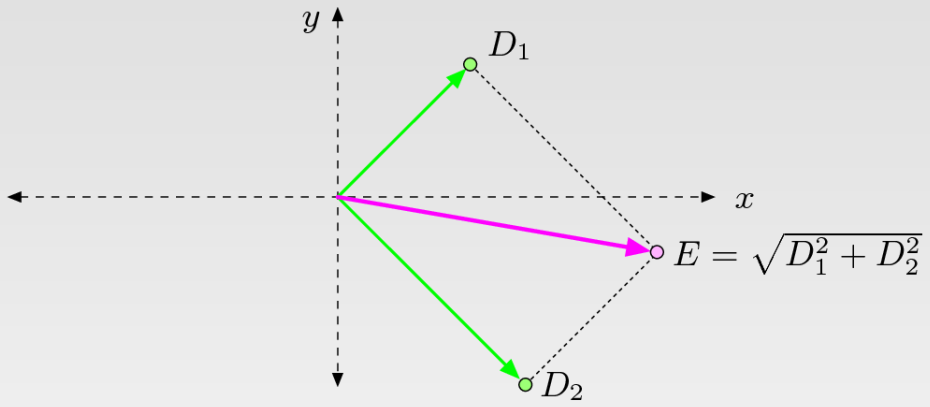
Detectores de bordas



Detectores de bordas

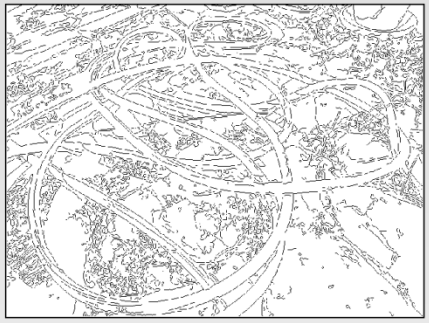
Algoritmo de **Canny**:

- Redução de ruídos
  - Filtro Gaussiano
- Detecção do gradiente
  - Magnitude e direção
- Supressão de não-máximos
  - Busca por máximos efetivos na direção do gradiente

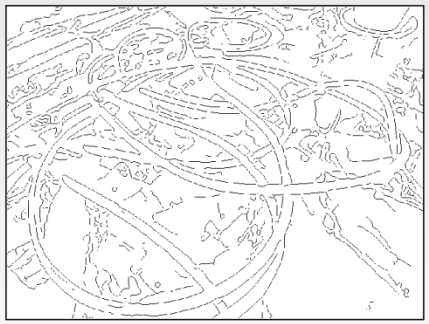


Detectores de bordas

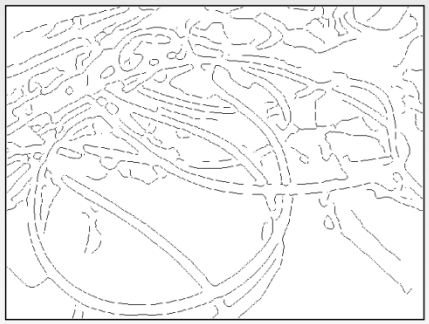
Original



$\sigma = 1.0$

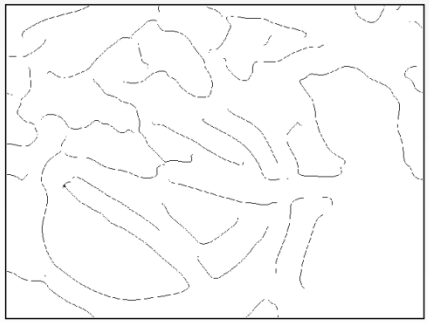


$\sigma = 2.0$



$\sigma = 4.0$

$\sigma = 8.0$

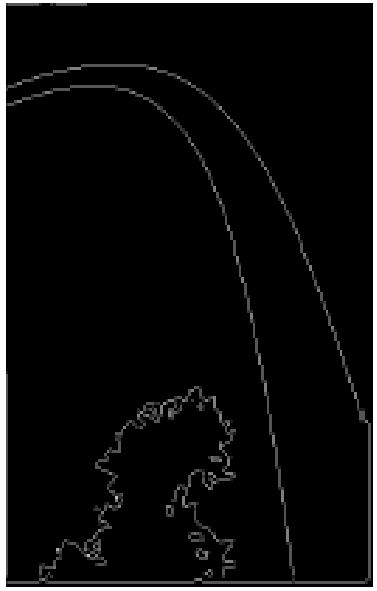


$\sigma = 16.0$

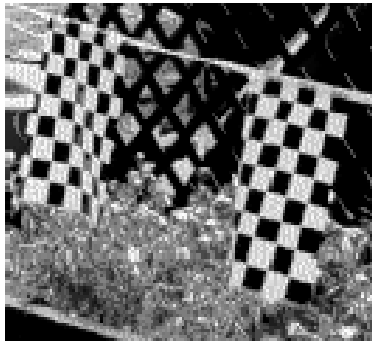
Detector de bordas de Canny

Original

Canny ($\sigma = 1$)



Canny ($\sigma = 4$)



Original

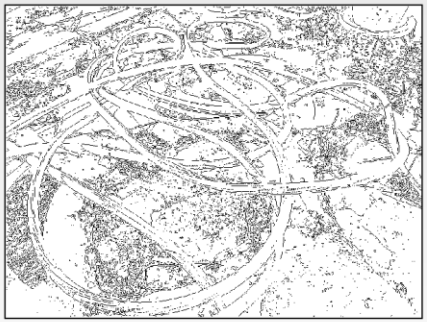
Canny ($\sigma = 1$)

Roberts (20%)

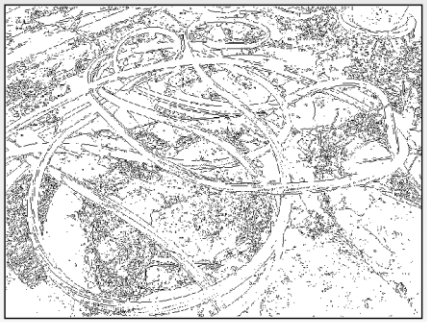
Detectores de bordas

Original

Roberts



Prewitt



Sobel

Laplacian of Gaussian

Canny ($\sigma = 1.0$)

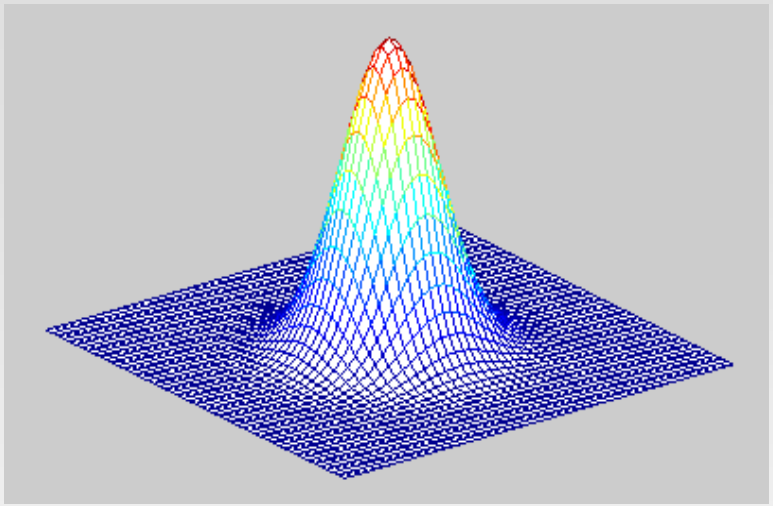
Filtros espaciais

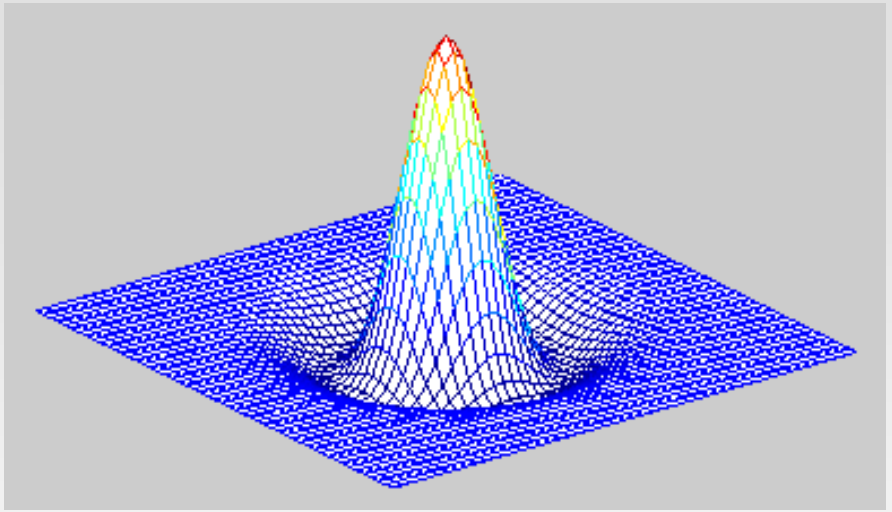
Filtro LoG:

$$\mathrm{LoG}_\sigma(x,y) = -\left(\frac{x^2+y^2-\sigma^2}{\sigma^4}\right)\cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Detector de bordas baseado na função Laplaciana – filtro LOG

$g(x, y)$





0	0	0	-1	-1	-2	-1	-1	0	0	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	-2	-7	-15	-22	-23	-22	15	-7	-2	0
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-2	-9	-23	-1	103	178	103	-1	-23	-9	-2
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
0	-2	-7	-15	-22	-23	-22	15	-7	-2	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	0	0	-1	-1	-2	-1	-1	0	0	0

$-L(x, y)$

⟶ Chapéu mexicano

⟵ Máscara 11x11 ($\sigma^2=2$)

$$L(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}$$