

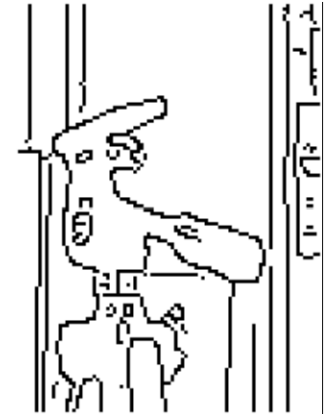
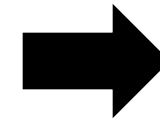
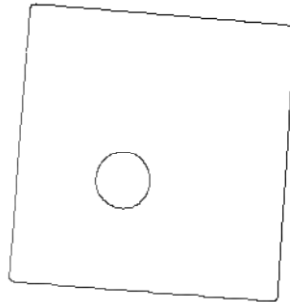
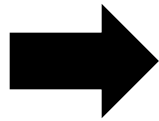
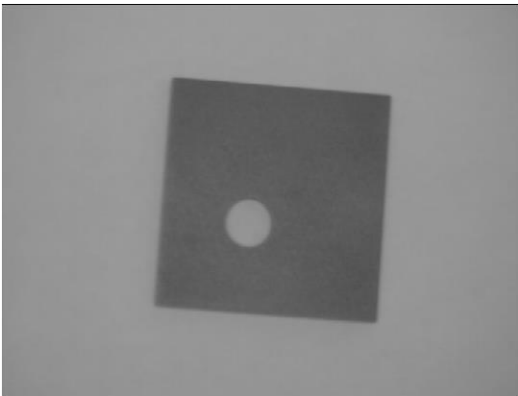
# Edge Detection

Prof. Giuseppe Lisanti

[giuseppe.lisanti@unibo.it](mailto:giuseppe.lisanti@unibo.it)

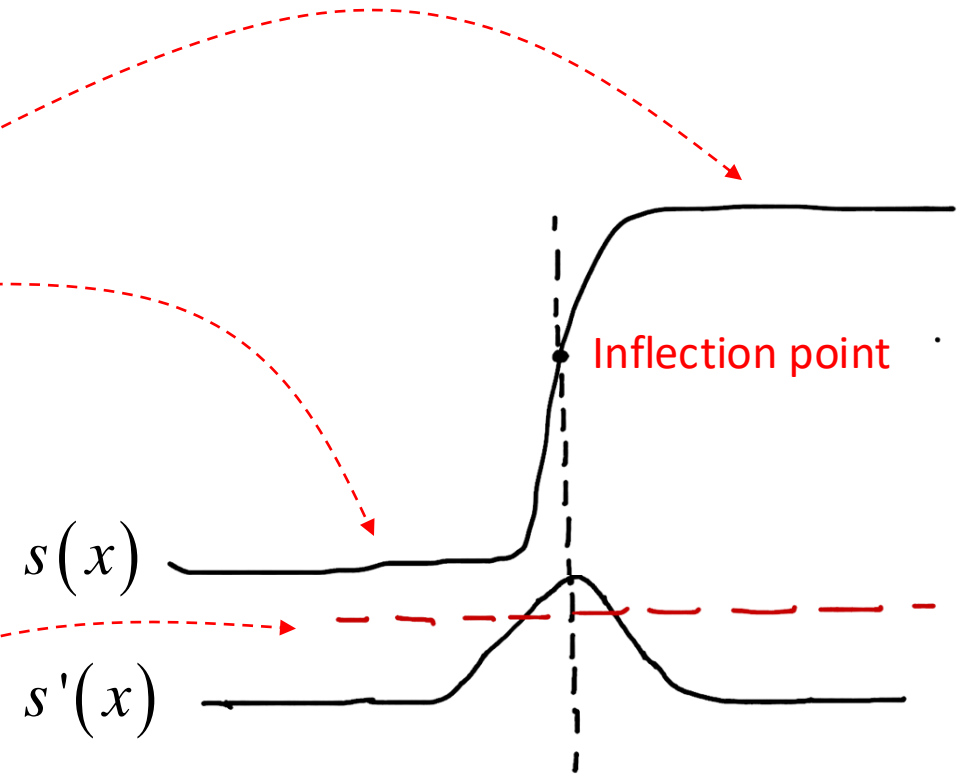
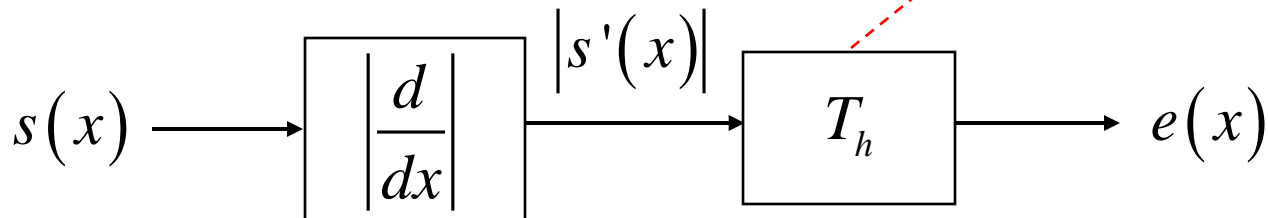
# Edge Detection

- Edge or contour points are local features of the image that capture important information related to its semantic content
  - Edges are exploited in countless image analysis tasks (e.g. foreground-background segmentation, template matching, stereo matching, visual tracking)
  - In machine vision, edge detection is key to measurement tools
- Edges are pixels that can be thought of as lying exactly in between image regions of different intensity, or, in other words, to separate different image regions



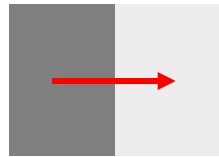
# Edge Detection (1D Step-Edge)

- Sharp change of a 1D signal (1D step-edge)
  - In the transition region between the two constant levels the absolute value of the first derivative gets high (and reaches an extremum at the inflection point)
  - The simplest edge-detection operator relies on thresholding the absolute value of the derivative of the signal

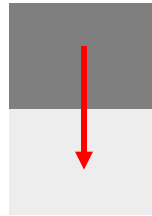


# Edge Detection (2D Step-Edge)

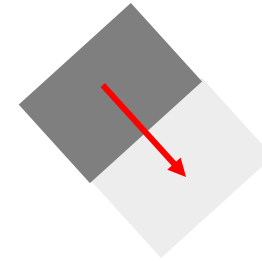
- A 2D Step-Edge is characterized not only by its strength but also its direction:



Vertical Edge  
(change sensed  
horizontally)



Horizontal Edge  
(change sensed  
vertically)

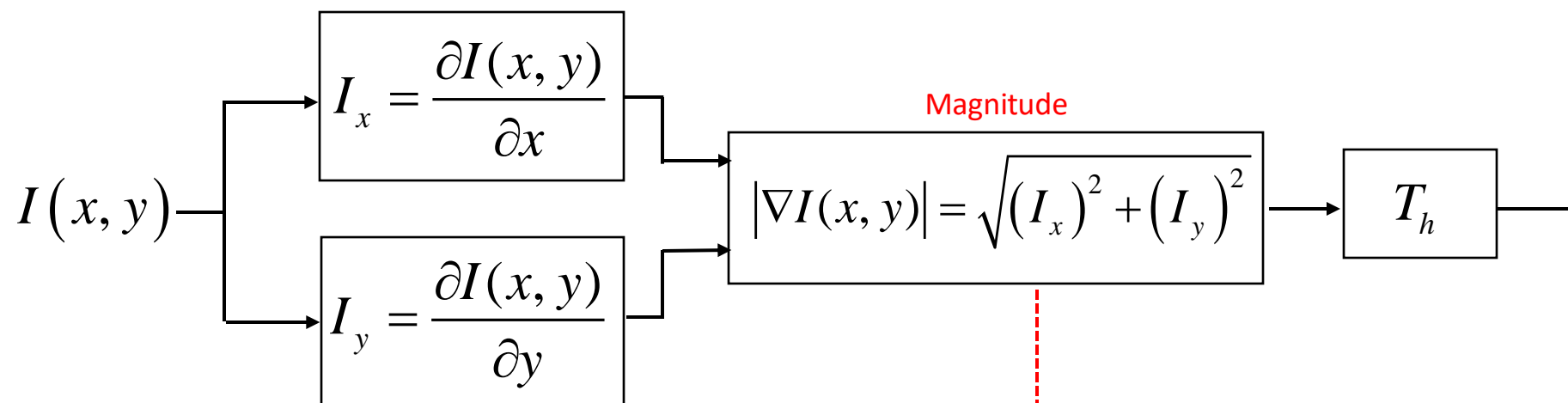


Diagonal Edge

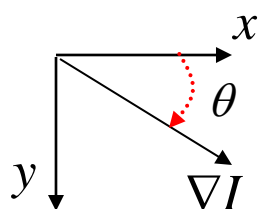
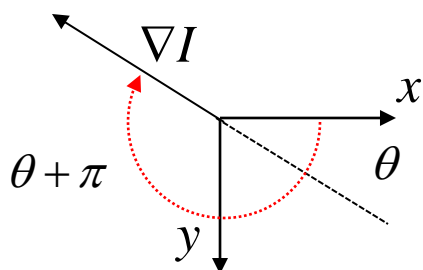
- An operator which allow us to sense the edge whatever its direction is
- The gradient is a vector composed by the derivatives along horizontal and vertical axis
- Gradient's direction gives the direction along which the function exhibits its maximum variation
- A generic directional derivative can be computed by the dot product between the gradient and the unit vector along the direction

$$\nabla I(x, y) = \frac{\partial I(x, y)}{\partial x} \mathbf{i} + \frac{\partial I(x, y)}{\partial y} \mathbf{j}$$

# Edge Detection by Gradient Thresholding



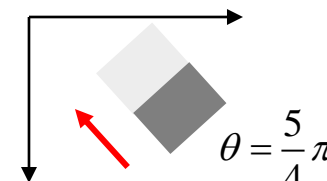
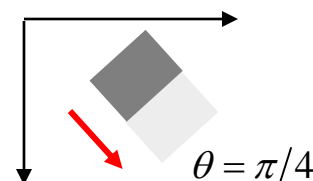
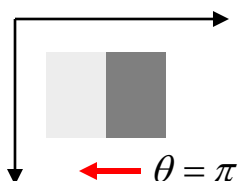
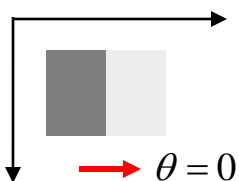
Same direction but  
different sign (i.e.  
contrast polarity)



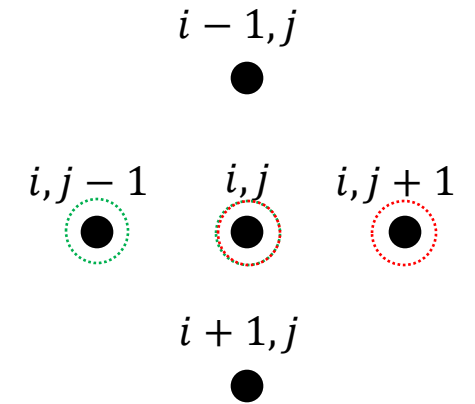
$$\left\{ \begin{array}{l} \mathcal{G} \in \left[ -\frac{\pi}{2}, \frac{\pi}{2} \right] = \text{atan} \left( \frac{I_y}{I_x} \right) \\ \mathcal{G} \in [0, 2\pi] = \text{atan2} (I_x, I_y) \end{array} \right.$$

Direction

Direction and sign



# Discrete Approximation of the Gradient



- We can use either *backward* or *forward* differences:

$$\frac{\partial I(x, y)}{\partial x} \cong I_x(i, j) = I(i, j) - I(i, j-1), \quad \frac{\partial I(x, y)}{\partial y} \cong I_y(i, j) = I(i, j) - I(i-1, j) \quad \text{backward}$$

$$\frac{\partial I(x, y)}{\partial x} \cong I_x(i, j) = I(i, j+1) - I(i, j), \quad \frac{\partial I(x, y)}{\partial y} \cong I_y(i, j) = I(i+1, j) - I(i, j) \quad \text{forward}$$

$$\begin{bmatrix} -1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

- Or central differences:  $I_x(i, j) = I(i, j+1) - I(i, j-1), \quad I_y(i, j) = I(i+1, j) - I(i-1, j)$

the corresponding correlation kernels are:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

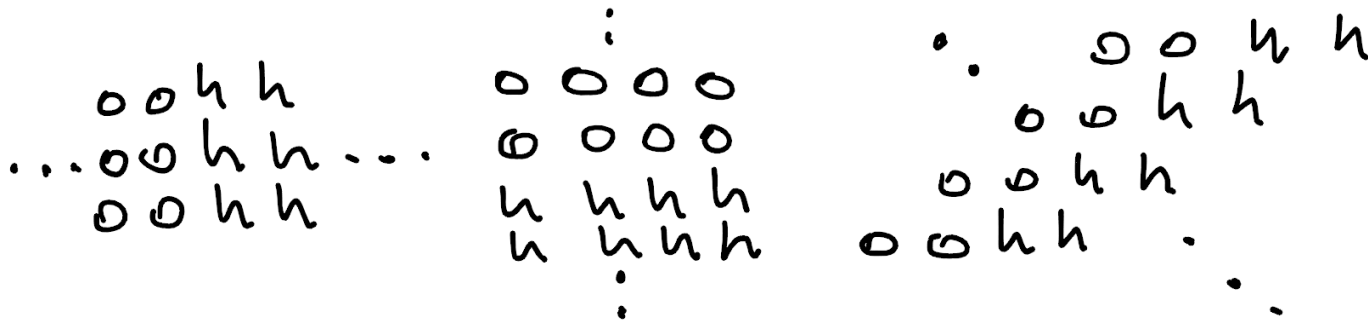
# Discrete Approximation of the Gradient

- We can estimate the magnitude using different approximations:

$$|\nabla I| = \sqrt{(I_x)^2 + (I_y)^2}$$

$$|\nabla I|_+ = |I_x| + |I_y|$$

$$|\nabla I|_{max} = \max(|I_x|, |I_y|)$$



$E_v$

$E_h$

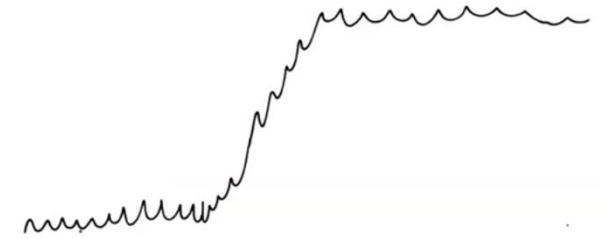
$E_d$

	$ \nabla I $	$ \nabla I _+$	$ \nabla I _{max}$
$E_h$	$h$	$h$	$h$
$E_v$	$h$	$h$	$h$
$E_d$	$h\sqrt{2}$	$2h$	$h$

More isotropic

- The third approximation is faster and more invariant with respect to edge direction

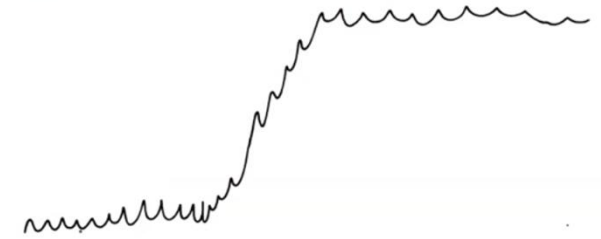
# Edges and noise



- In real images an edge will not look as smooth as we have seen (due to noise)
  - Taking derivatives of noisy signals is an ill posed problem...the solution is not robust wrt to input variations =>  
***Derivatives amplify noise***
  - To work with real images, an edge detector should therefore be robust to noise, so as to highlight the meaningful edges only and filter out effectively the spurious transitions caused by noise
  - How do we get rid of noise?



# Edges and noise



- In real images an edge will not look as smooth as we have seen (due to noise)
  - Taking derivatives of noisy signals is an ill posed problem...the solution is not robust wrt to input variations => ***Derivatives amplify noise***
  - To work with real images, an edge detector should therefore be robust to noise, so as to highlight the meaningful edges only and filter out effectively the spurious transitions caused by noise
  - This is usually achieved by smoothing the signal before computing the derivatives required to highlight edges
    - smoothing side-effect: blurring true edges, making it more difficult to detect and localize them
- ***Pixel of different colors will look similar after filtering because of the blending***  
Smoothing and differentiation can be carried out jointly within a single step
  - This is achieved by computing ***differences of averages*** (rather than averaging the image and then computing differences)
  - To try avoiding smoothing across edges the two operations are carried out along orthogonal directions

# Edges and noise

$$\mu_x(i, j) = \frac{1}{3} [I(i, j-1) + I(i, j) + I(i, j+1)]$$

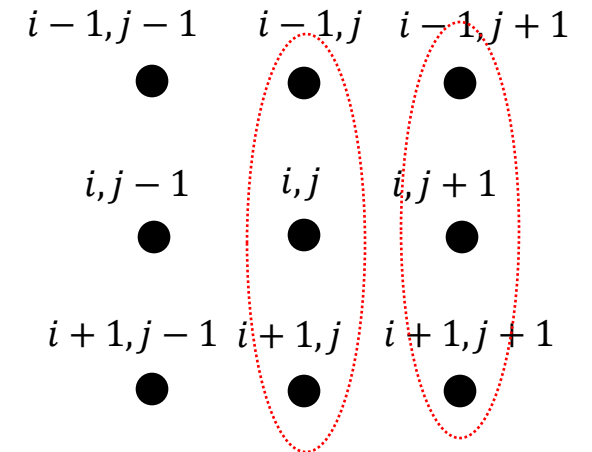
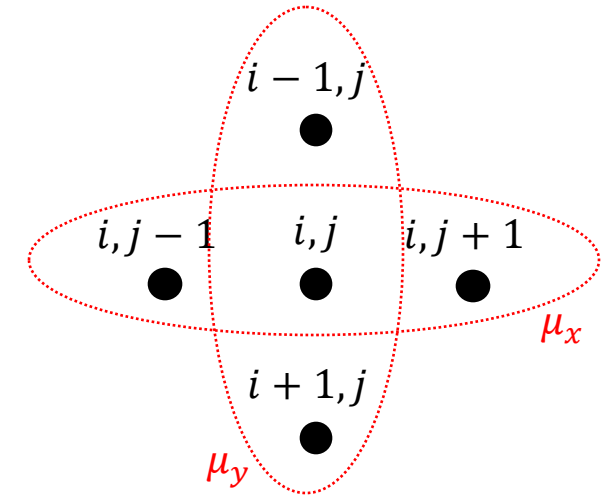
$$\mu_y(i, j) = \frac{1}{3} [I(i-1, j) + I(i, j) + I(i+1, j)]$$

averages

$$\tilde{I}_x(i, j) = \mu_y(i, j+1) - \mu_y(i, j)$$

$$= \frac{1}{3} [I(i-1, j+1) + I(i, j+1) + I(i+1, j+1) - I(i-1, j) - I(i, j) - I(i+1, j)]$$

$$\Rightarrow \frac{1}{3} \begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}$$



# Edges and noise

$$\mu_x(i, j) = \frac{1}{3} [I(i, j-1) + I(i, j) + I(i, j+1)]$$

$$\mu_y(i, j) = \frac{1}{3} [I(i-1, j) + I(i, j) + I(i+1, j)]$$

averages

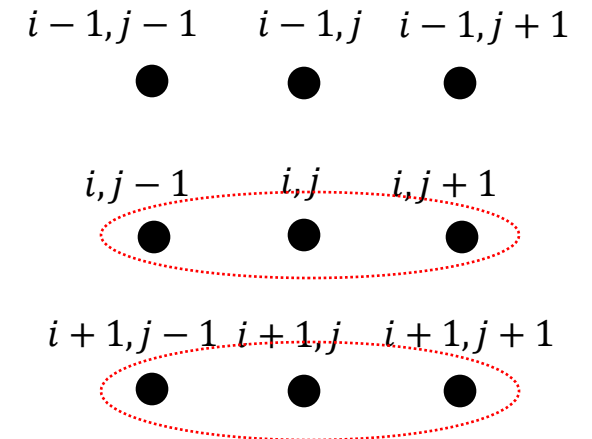
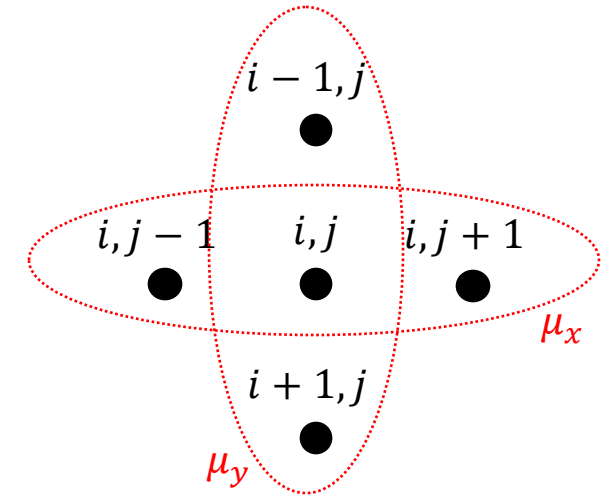
$$\tilde{I}_x(i, j) = \mu_y(i, j+1) - \mu_y(i, j)$$

$$= \frac{1}{3} [I(i-1, j+1) + I(i, j+1) + I(i+1, j+1) - I(i-1, j) - I(i, j) - I(i+1, j)]$$

$$\tilde{I}_y(i, j) = \mu_x(i+1, j) - \mu_x(i, j)$$

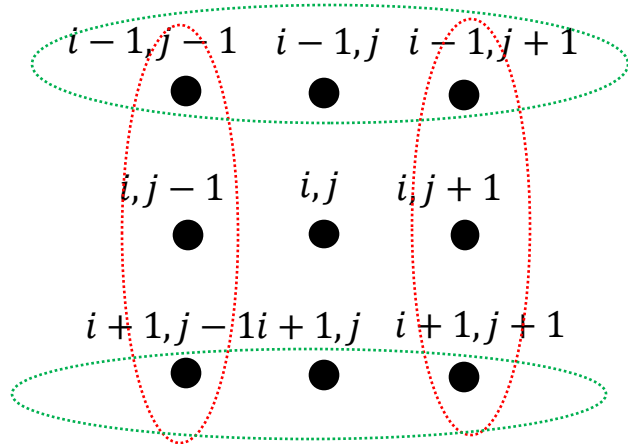
$$= \frac{1}{3} [I(i+1, j-1) + I(i+1, j) + I(i+1, j+1) - I(i, j-1) - I(i, j) - I(i, j+1)]$$

$$\Rightarrow \frac{1}{3} \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$



# Prewitt and Sobel

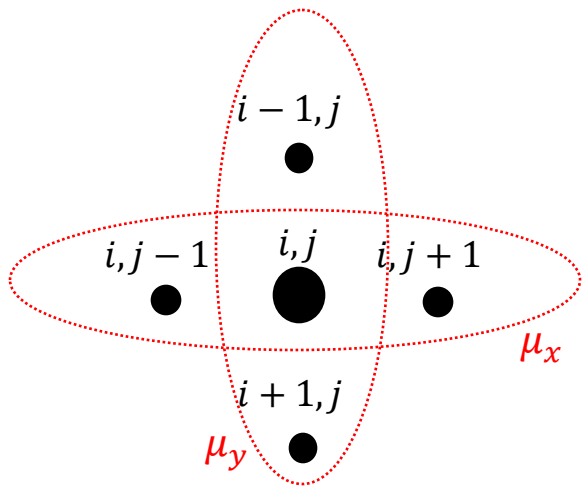
- Given the same smoothing, one might wish to approximate partial derivatives by central differences (Prewitt operator):



$$\tilde{I}_x(i, j) = \mu_y(i, j+1) - \mu_y(i, j-1) \Rightarrow \frac{1}{3} \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\tilde{I}_y(i, j) = \mu_x(i+1, j) - \mu_x(i-1, j) \Rightarrow \frac{1}{3} \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

- Likewise, the central pixel can be weighted more (Sobel operator):



$$\mu_x(i, j) = \frac{1}{4} [I(i, j-1) + 2I(i, j) + I(i, j+1)]$$

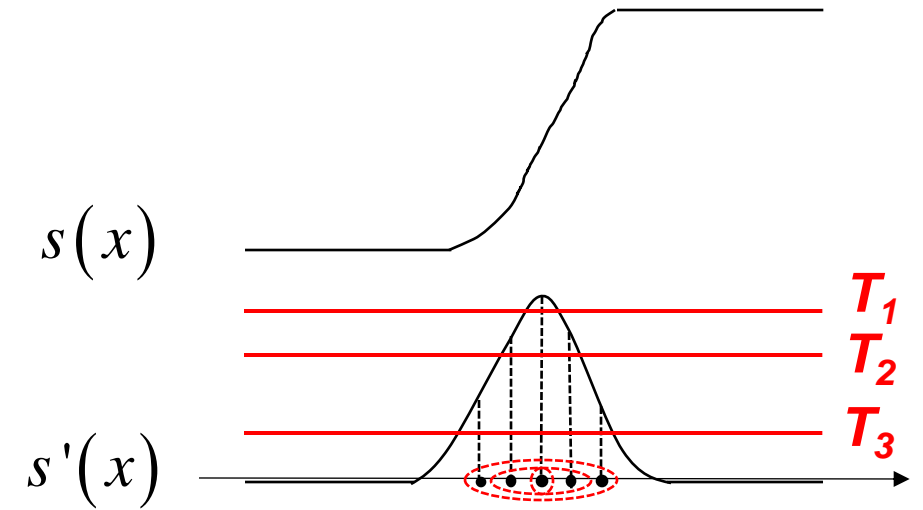
$$\tilde{I}_x(i, j) = \mu_y(i, j+1) - \mu_y(i, j-1) \Rightarrow \frac{1}{4} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\mu_y(i, j) = \frac{1}{4} [I(i-1, j) + 2I(i, j) + I(i+1, j)]$$

$$\tilde{I}_y(i, j) = \mu_x(i+1, j) - \mu_x(i-1, j) \Rightarrow \frac{1}{4} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

# Edges Detection

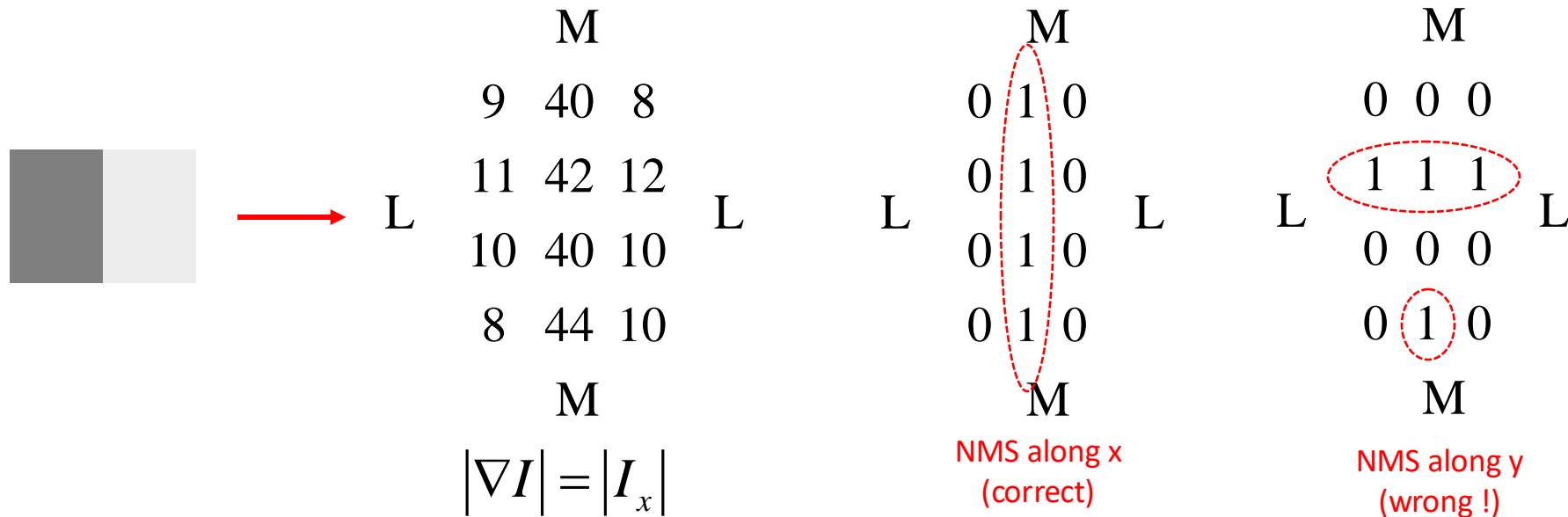
- Detecting edges by gradient thresholding is inherently inaccurate:
  - It is difficult to choose the right threshold
    - the image contains meaningful edges characterized by different contrast (i.e. “stronger” as well as “weaker”)
  - Trying to detect weak edges implies poor localization of stronger ones
- A better approach to detect edges may consist in finding the **local maxima** of the absolute value of the derivative of the signal



How many pixels we want to find for that edge?

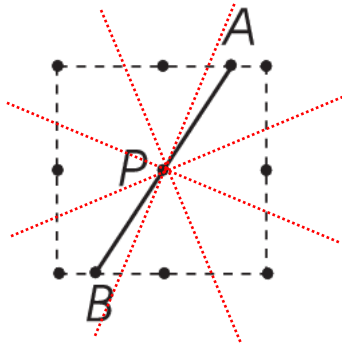
# Non-Maxima Suppression (NMS)

- When dealing with images (2D signals) one should look for:
  - maxima of the absolute value of the derivative (i.e. the gradient magnitude)
  - along the gradient direction (i.e. orthogonal to the edge direction)



- We don't know in advance the correct direction to carry out NMS
  - The direction has to be estimated locally (based on gradient's direction)

# Non-Maxima Suppression (NMS)



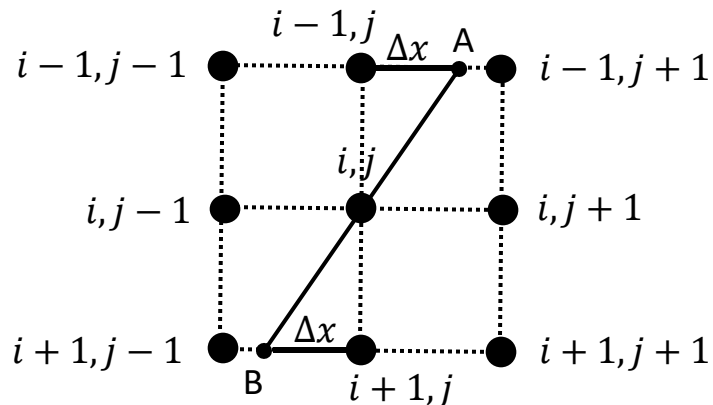
$$G = \|\nabla I(i, j)\|$$

$$G_A \cong \|\nabla I(i - 1, j + 1)\|$$

$$G_B \cong \|\nabla I(i + 1, j - 1)\|$$

$$NMS(i, j) = \begin{cases} 1: (G > G_A) \wedge (G > G_B) \\ 0: otherwise \end{cases}$$

- The magnitude of the gradient has to be estimated at points which do not belong to the discrete pixel grid
- Such values can be estimated by linear interpolation of those computed at the closest points belonging to the grid



$$G_1 = \|\nabla I(i - 1, j)\|$$

$$G_2 = \|\nabla I(i - 1, j + 1)\|$$

$$G_3 = \|\nabla I(i + 1, j - 1)\|$$

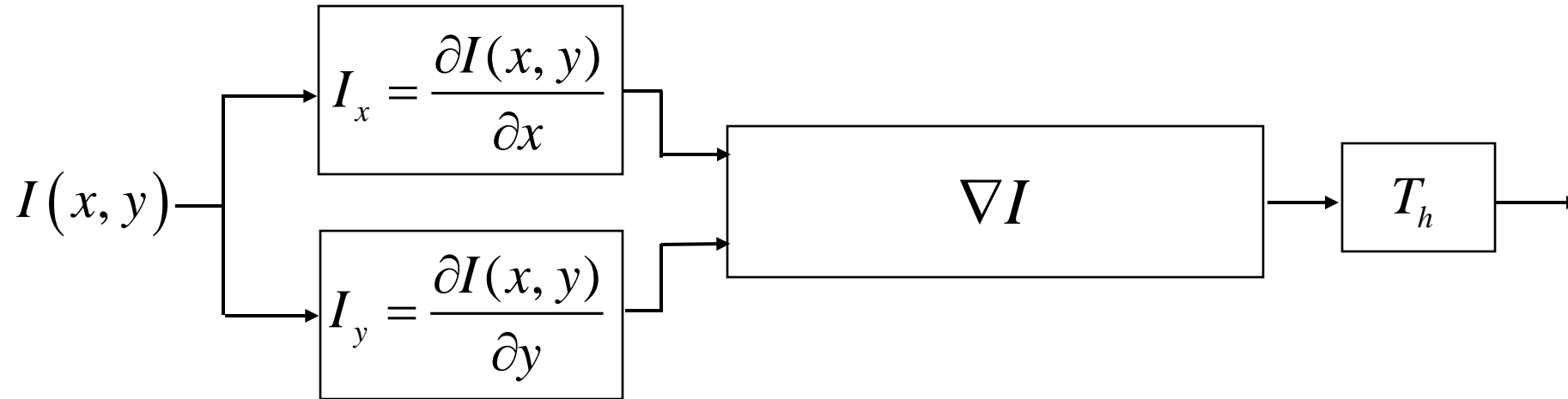
$$G_4 = \|\nabla I(i + 1, j)\|$$

$$G = \|\nabla I(i, j)\|$$

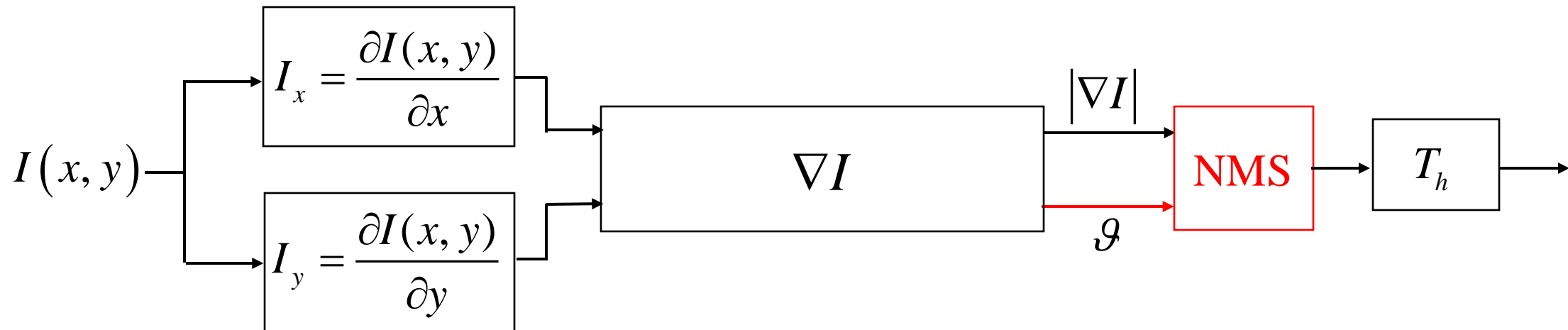
$$G_A \cong G_1 + (G_2 - G_1)\Delta x$$

$$G_B \cong G_4 + (G_3 - G_4)\Delta x$$

# Edge pipeline



- A final thresholding step on the magnitude of the gradient at the points selected by the NMS process typically helps pruning out unwanted edges due to either noise or less important details.





# Canny's Edge Detector

- Canny proposed to set forth quantitative criteria to measure the performance of an edge detector and then to find the optimal filter with respect to such criteria
- He proposed the following three criteria:
  - **Good Detection:** the filter should correctly extract edges in noisy images
  - **Good Localization:** the distance between the found edge and the “true” edge should be minimum
  - **One Response to One Edge:** the filter should detect one single edge pixel at each “true” edge
- Addressing the 1D case and modeling an edge as a noisy step, Canny shows that the optimal edge detection operation consists in finding **local extrema of the convolution of the signal by a first order Gaussian derivative** (i.e.  $G'(x)$ ).
- A straightforward Canny edge detector can be achieved by:
  - Gaussian smoothing
  - Gradient computation
  - NMS along the gradient direction

# Canny's Edge Detector

- 2D convolution by a Gaussian can be slow and we can leverage on separability of the Gaussian function to speed-up the calculation

$$\tilde{I}_x(x, y) = \frac{\partial}{\partial x} (I(x, y) * G(x, y)) = I(x, y) * \frac{\partial G(x, y)}{\partial x}$$

$$\tilde{I}_y(x, y) = \frac{\partial}{\partial y} (I(x, y) * G(x, y)) = I(x, y) * \frac{\partial G(x, y)}{\partial y}$$

$$G(x, y) = G(x)G(y)$$

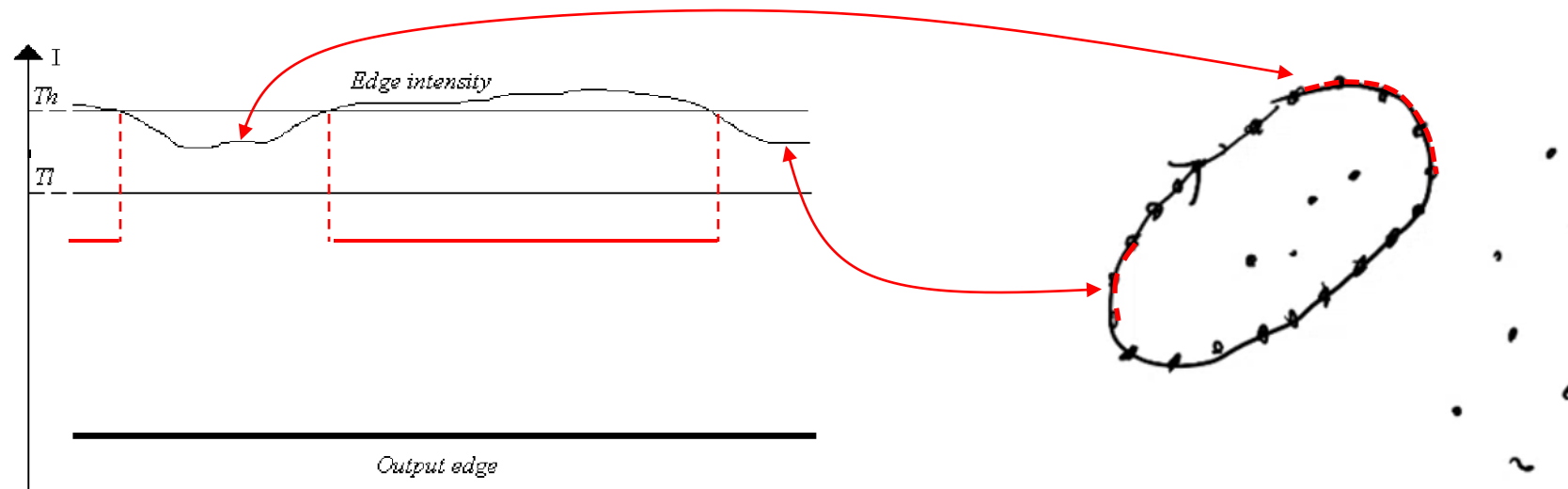


$$\tilde{I}_x(x, y) = I(x, y) * (G'(x)G(y)) = (I(x, y) * G'(x)) * G(y)$$

$$\tilde{I}_y(x, y) = I(x, y) * (G'(y)G(x)) = (I(x, y) * G'(y)) * G(x)$$

# Canny's Edge Detector

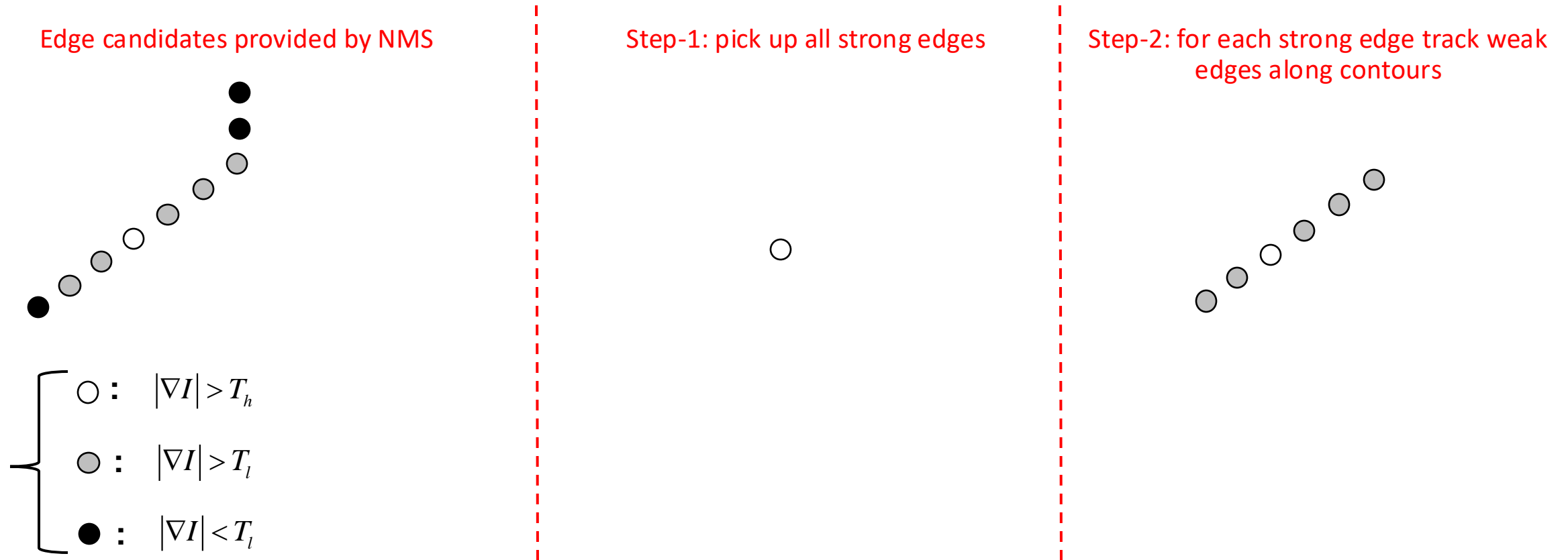
- NMS is often followed by thresholding of gradient magnitude to help distinguish between true “semantic” edges and unwanted ones



- Edge **streaking** may occur when magnitude varies along object contours
- Canny proposed a “**hysteresis**” thresholding approach relying on a higher ( $T_h$ ) and a lower ( $T_l$ ) threshold
  - A pixel is taken as an edge if either the gradient magnitude is higher than  $T_h$  **OR** higher than  $T_l$  **AND** the pixel is a neighbor of an already detected edge

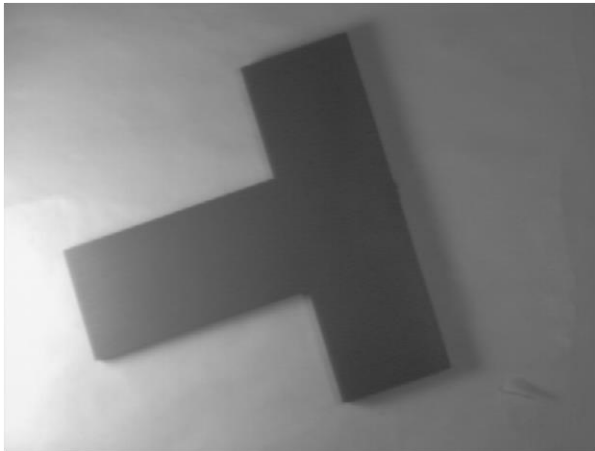
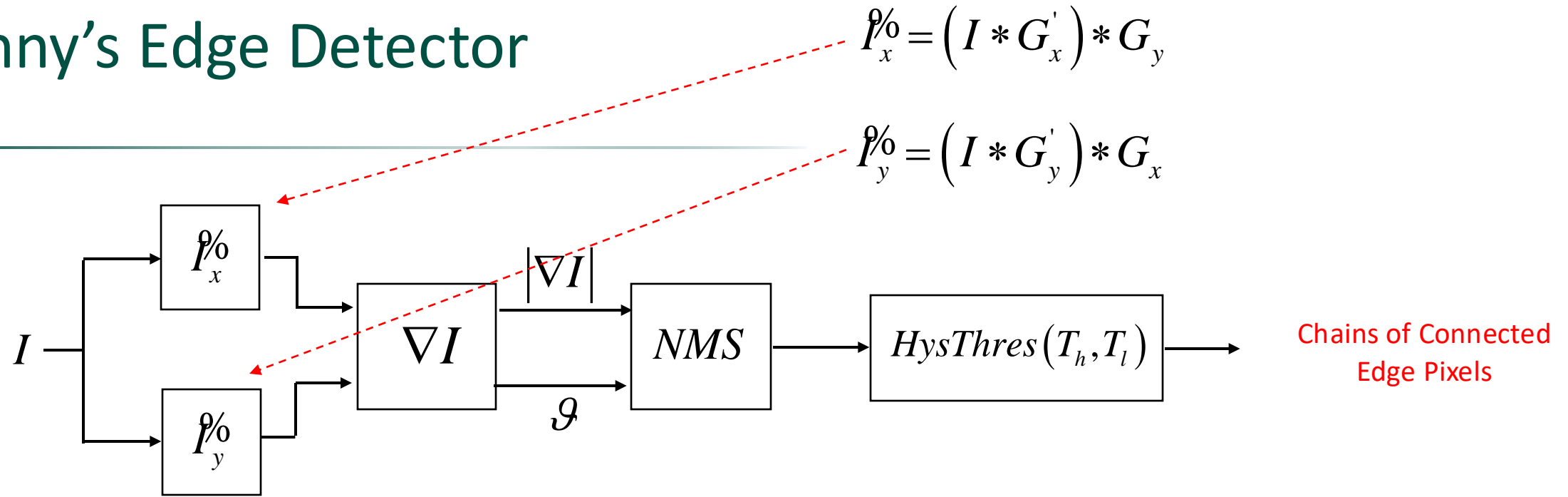
# Canny's Edge Detector

- Hysteresis thresholding is usually carried out by tracking edge pixels along contours

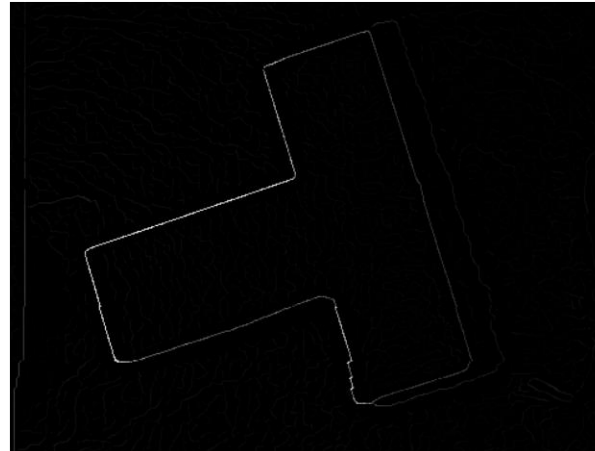
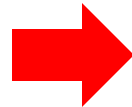


- You get a set of chains, each chain is a list of pixels belonging to that contour

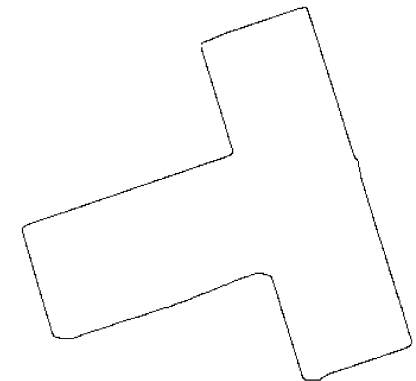
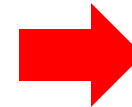
# Canny's Edge Detector



Input

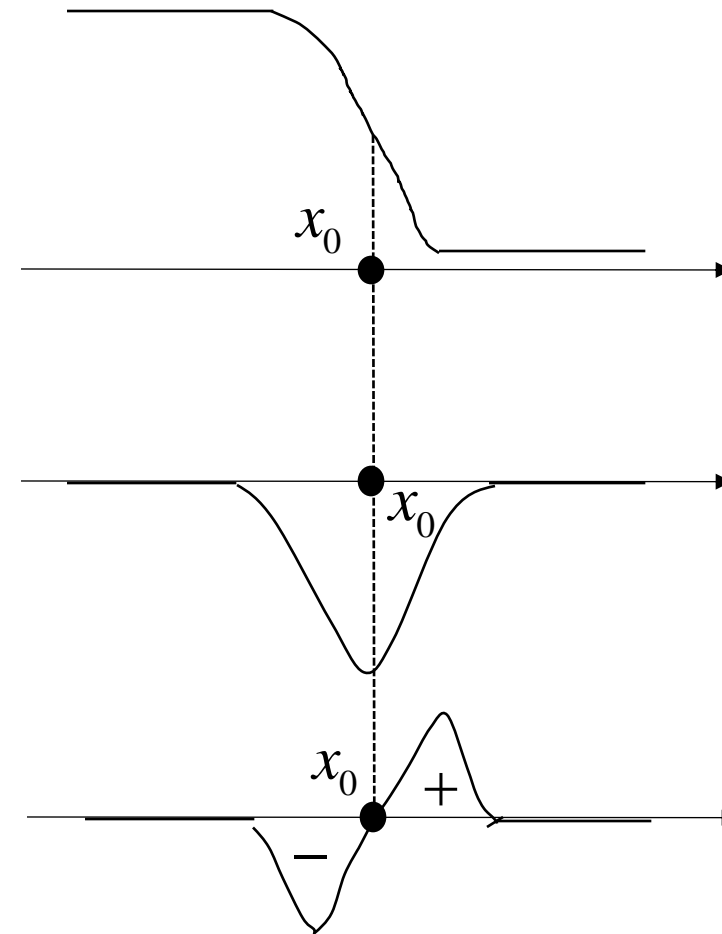
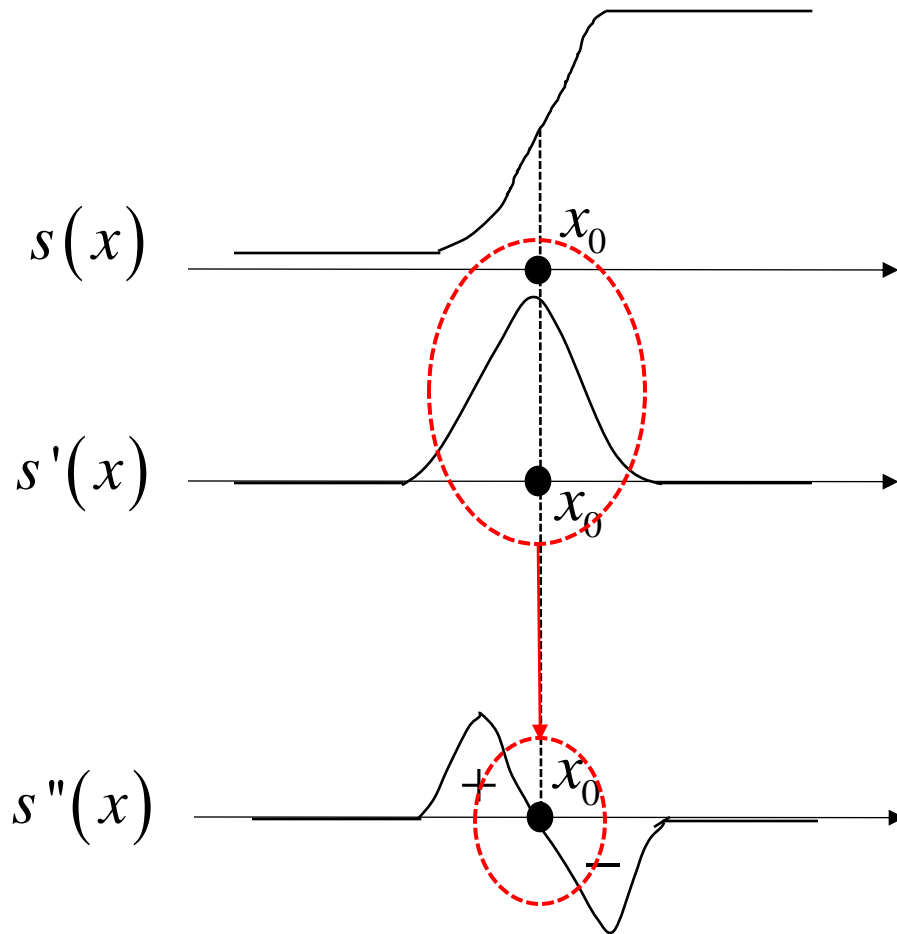


NMS output (with gradient magnitude represented by gray-scales)



Final Output

# Zero-crossing



- Look for zero-crossing of the second derivative of the signal to locate edges (instead of the peaks of the first derivative)
  - it crosses zero but before it is immediately positive (or negative)...
  - It requires significant computational effort

# Second derivative along the gradient & Laplacian

- The second derivative along the gradient's direction can be obtained as  $\mathbf{n}^T \mathbf{H} \mathbf{n}$

$$\mathbf{n} = \frac{\nabla I(x, y)}{\|\nabla I(x, y)\|}$$

(unit vector along the gradient's direction)

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 I(x, y)}{\partial x^2} & \frac{\partial^2 I(x, y)}{\partial x \partial y} \\ \frac{\partial^2 I(x, y)}{\partial y \partial x} & \frac{\partial^2 I(x, y)}{\partial y^2} \end{bmatrix}$$

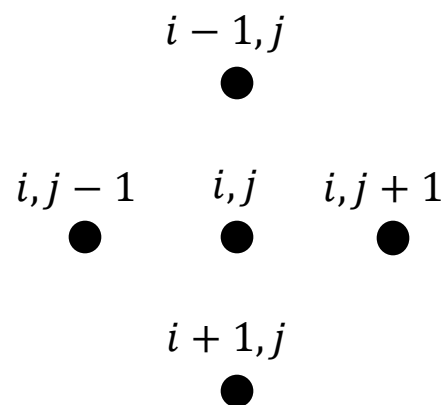
(Hessian matrix)

- Computing the second derivative along the gradient turns out very expensive. In their seminal work on edge detection, instead, Marr & Hildreth proposed to rely on the Laplacian as second order differential operator:

$$\nabla^2 I(x, y) = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2} = I_{xx} + I_{yy}$$

# Discrete Laplacian

- One can use **forward** and **backward** differences to approximate first and second order derivatives, respectively:


$$I_x(i, j) \approx \frac{I(i, j+1) - I(i, j-1)}{2}$$
$$I_{xx} \approx \frac{I(i, j+1) - I(i, j)}{\Delta x} - \frac{I(i, j) - I(i, j-1)}{\Delta x} = I(i, j+1) - 2I(i, j) + I(i, j-1)$$
$$I_y(i, j) \approx \frac{I(i, j+1) - I(i, j-1)}{2}$$
$$I_{yy} \approx \frac{I(i, j+1) - I(i, j)}{\Delta y} - \frac{I(i, j) - I(i, j-1)}{\Delta y} = I(i, j+1) - 2I(i, j) + I(i, j-1)$$

- It can be shown that the zero-crossing of the Laplacian typically lay close to those of the second derivative along the gradient. Yet, the Laplacian is much faster to compute, i.e. just a convolution by a 3x3 kernel

$$\nabla^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



# Laplacian of Gaussian (LOG)

- A robust edge detector should include a smoothing step to filter out noise (especially in case second rather than first order derivatives are deployed)
  - In their edge detector, Marr & Hildreth proposed to use a Gaussian filter as smoothing operator
- Edge detection by the LOG can be summarized conceptually as follows:
  - Gaussian smoothing:  $\tilde{I}(x, y) = I(x, y) * G(x, y)$
  - Second order differentiation by the Laplacian
  - Extraction of the zero-crossing of  $\nabla^2 \tilde{I}(x, y)$
- Practical implementations of the LOG may deploy the properties of convolutions to speed-up the computation

# Laplacian of Gaussian (LOG)

---

- Zero crossing (e.g., we may observe a Laplacian like  $[-10, 0, 20]$ , but this sometimes is impossible...)
- You can find sign changes from minus to plus (and vice versa), like  $[-10, 20]$ , between two consecutive pixels
- Once a sign change is found, the actual edge may be localized:
  - At the pixel where the LOG is positive (darker side of the edge)
  - At the pixel where the LOG is negative (brighter side of the edge)
  - At the pixel where the **absolute** value of the LOG is smaller (the best choice, as the edge turns out closer to the “true” zero-crossing)
- To help discarding spurious edges, a final thresholding step may be enforced (usually based on the slope of the LOG at the found zero-crossing)

# Laplacian of Gaussian (LOG)

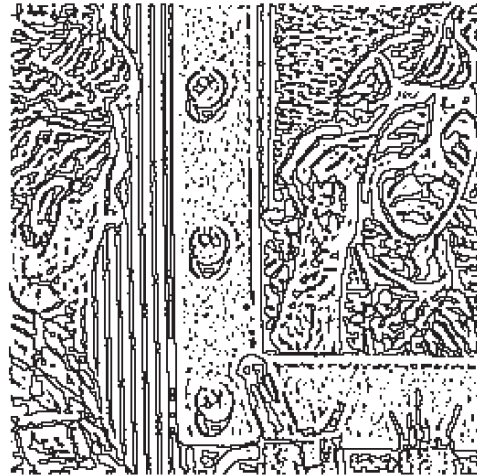
---

- Unlike those based on smooth derivatives, the LOG edge detector allows the degree of smoothing to be controlled (i.e. by changing the  $\sigma$  parameter of the Gaussian filter).
  - This, in turn, allows the edge detector to be tuned according to the degree of noise in the image (i.e. higher noise  $\rightarrow$  larger  $\sigma$ )
- Likewise,  $\sigma$  may be used to control the scale at which the image is analyzed, with larger  $\sigma$  typically chosen to extract the edges related to main scene structures and smaller  $\sigma$  to capture small size details
- Zero-crossing are usually sought for by scanning the image by both rows and columns to identify changes of the sign of the LOG

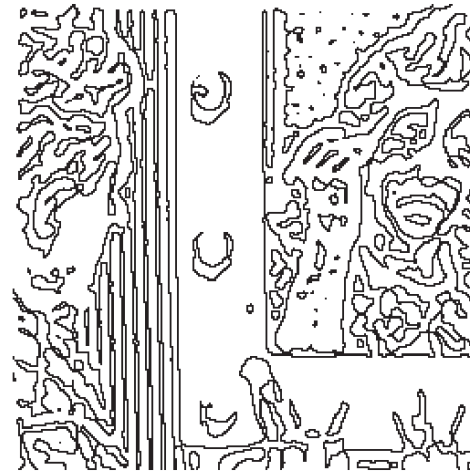
# LOG application examples



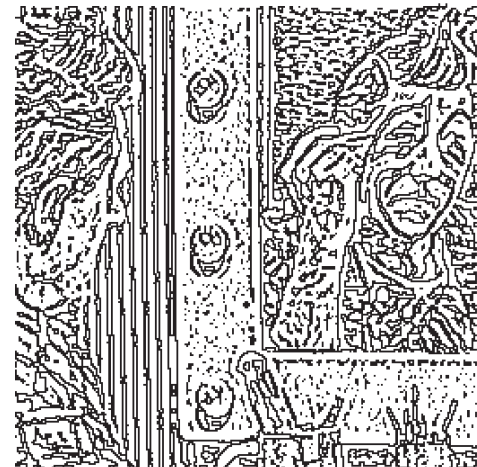
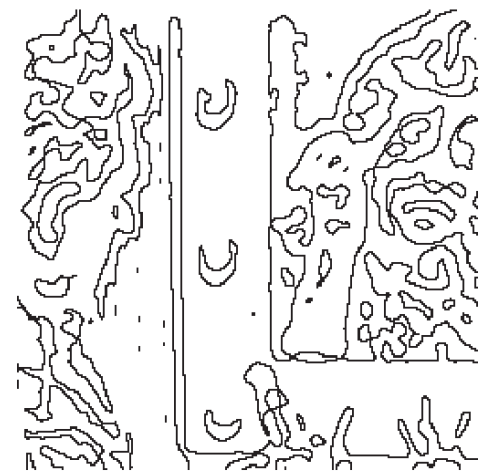
$\sigma = 1$



$\sigma = 2$



$\sigma = 3$



Use small sigma and then apply a threshold