Image Processing and Computer Vision

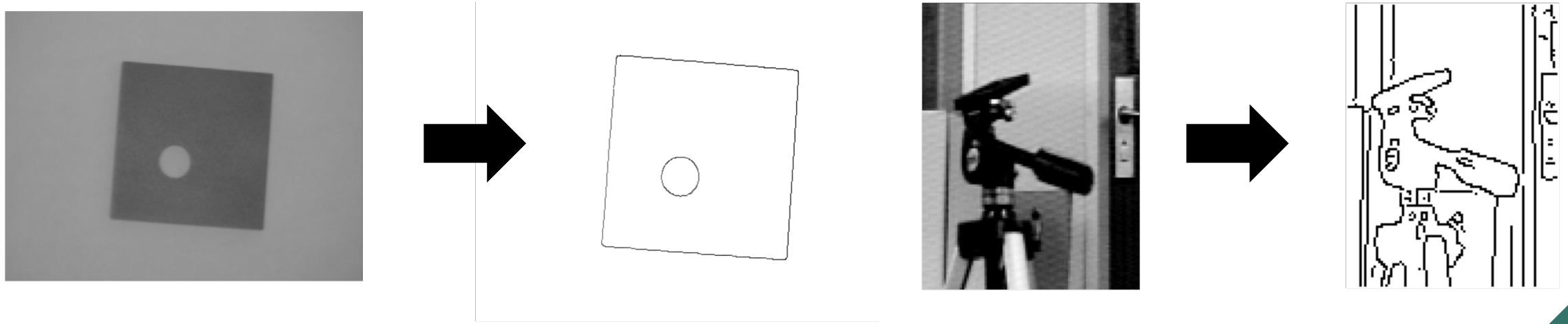
Prof. Giuseppe Lisanti
giuseppe.lisanti@unibo.it

Edge Detection

edges are the simplest feature we can extract from images

- 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

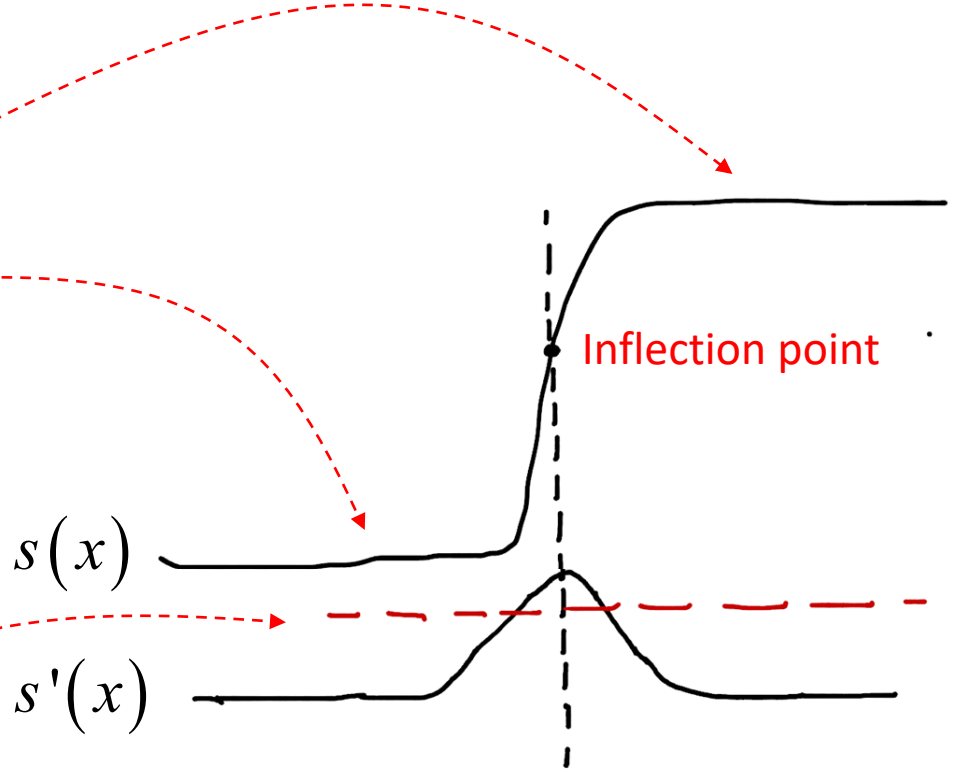
- Edges are pixels that can be thought of as lying exactly in between image regions of different intensity, or, in other words, to separate different image regions



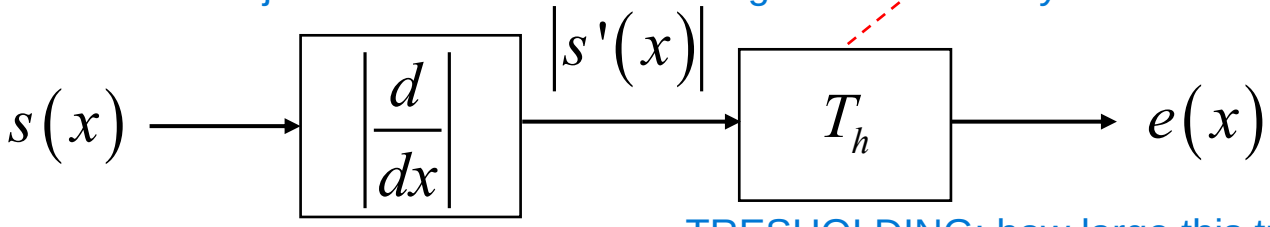
Edge Detection (1D Step-Edge)

how we do compute edges?

- 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



I take the absolute value because
I don't care about polarity (if I go from low to high or viceversa),
I just want to sense the change in the intensity



with the DERIVATIVE of the signal we can measure the steepness of the change in intensity

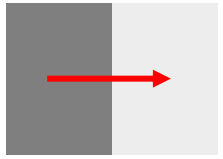
with derivatives I sense changes

TRESHOLDING: how large this transition/change has to be in order to be considered an edge?

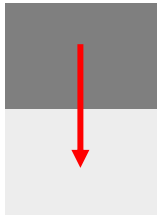
Edge Detection (2D Step-Edge)

in reality we deal with 2D signals (images)

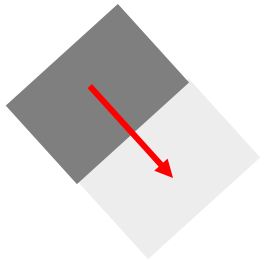
- A 2D Step-Edge is characterized not only by its strength but also its direction:



Vertical Edge (change sensed horizontally)



Horizontal Edge (change sensed vertically)



Diagonal Edge

we compute derivatives of 2D signals using GRADIENT

↳ it tells us the direction along which we sense the maximum variation of intensity for our 2D signal

- An operator which allow us to sense the edge whatever its direction is $\nabla I(x,y) = \frac{\partial I(x,y)}{\partial x}\mathbf{i} + \frac{\partial I(x,y)}{\partial y}\mathbf{j}$
- 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

Edge Detection by Gradient Thresholding

compute the two partial derivatives

$$I_x = \frac{\partial I(x,y)}{\partial x}$$

$$I_y = \frac{\partial I(x,y)}{\partial y}$$

Magnitude

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

tresholding

$$T_h$$

we can also extract another information, the orientation

we exploit arctan function

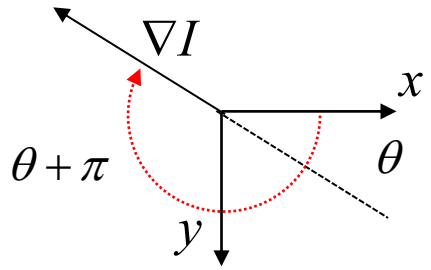
$$\begin{cases} \vartheta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] = \text{atan}\left(\frac{I_y}{I_x}\right) \\ \vartheta \in [0, 2\pi] = \text{atan2}(I_x, I_y) \end{cases}$$

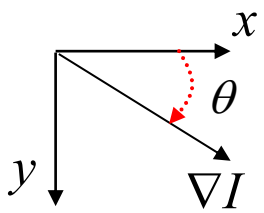
Direction

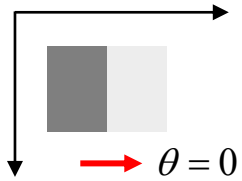
Direction and sign

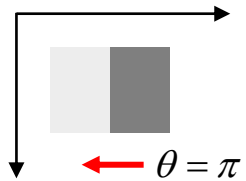
sign: polarity

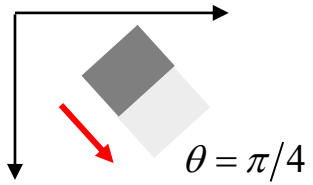
Same direction but different sign (i.e. contrast polarity)

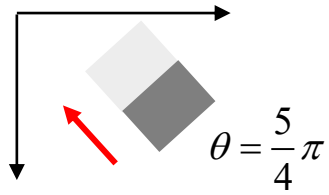




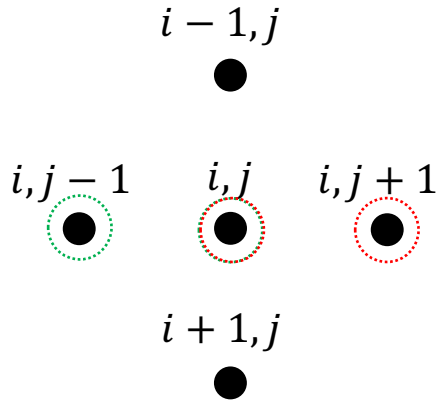








Discrete Approximation of the Gradient



I look at changes from my current position and the previous/next pixel

in prev/next row or column

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

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

backward

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

forward

look around my current position (previous/next position)

- Or central differences:

$$I_x(i,j) = I(i,j+1) - I(i,j-1), \qquad I_y(i,j) = I(i+1,j) - I(i-1,j)$$

the corresponding correlation kernels are:

vertical derivative $[-1 \quad 0 \quad 1]$

horizontal $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

we can approximate the computation of the derivative using this two filters

Discrete Approximation of the Gradient

I cannot choose a treshold for every direction (horizontal, vertical, diagonal)

- We can estimate the magnitude using different approximations:

remark: we use abs value because I dont care about polarity

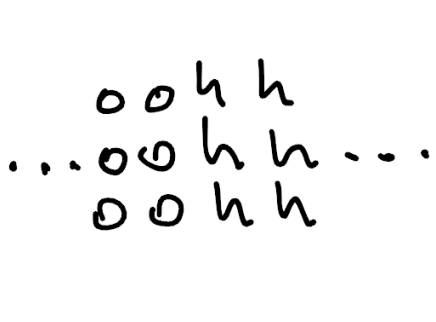
L2

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

L1

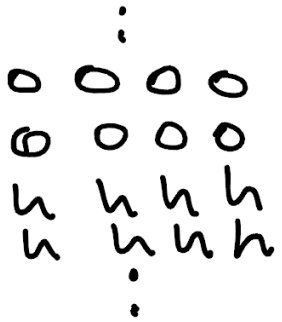
$$|\nabla I|_+ = |I_x| + |I_y|$$

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



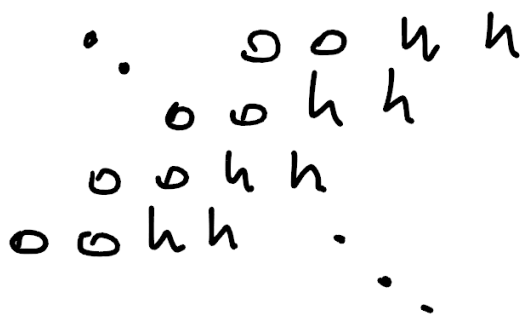
vertical

E_v



horizontal

E_h



diagonal

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

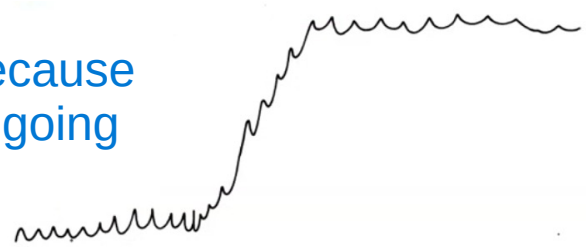
More isotropic

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

we want to have the same response from the function that computes magnitude

Edges and noise

we cannot use the solutions proveided before because the previous gradient approximation kernels are going to amplify the noise even more



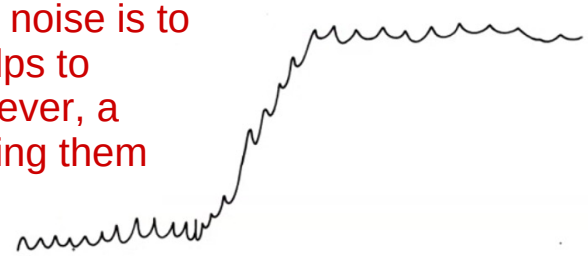
- In real images an edge will not look as smooth as we have seen (due to noise)
  - Taking derivatives of noisy signals is an ill posed problem…the solution is not robust wrt to input variations => *Derivatives amplify noise*
  - To work with real images, an edge detector should therefore be robust to 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?

filters? no, because they smooth edges (linear ones even more)

Edges and noise

Noise can cause fluctuations in pixel values that may obscure or distort the true edges in the image.

* One common approach to make edge detection more robust to noise is to smooth the image before computing derivatives. Smoothing helps to suppress noise, making it easier to detect genuine edges. However, a side-effect of smoothing is that it can also blur true edges, making them more difficult to detect and localize.



- 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

solution

- Smoothing and differentiation can be carried out jointly within a single step

smoothing reduces noise, differentiation detects edges

  - This is achieved by computing *differences of averages* (rather than averaging the image and then computing differences)

differences of averages involves breaking down the image into smaller regions or neighborhoods and then computing the difference in average pixel values between neighboring regions.

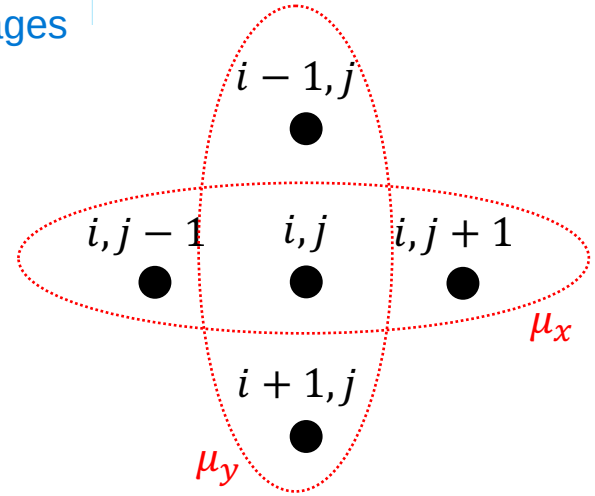
  - To try avoiding smoothing across edges the two operations are carried out along orthogonal directions

The process above helps to preserve edge information while reducing noise.

Edges and noise

we no longer take differences on a single pixel, like before, but we average the image taking difference of averages

we use FORWARD differences



in this case we divide by 3 because we consider 3 pixels in each direction

$$\mu_x(i,j) = \frac{1}{3}[I(i,j-1) + I(i,j) + I(i,j+1)]$$ take the average along the horizontal axis

averages

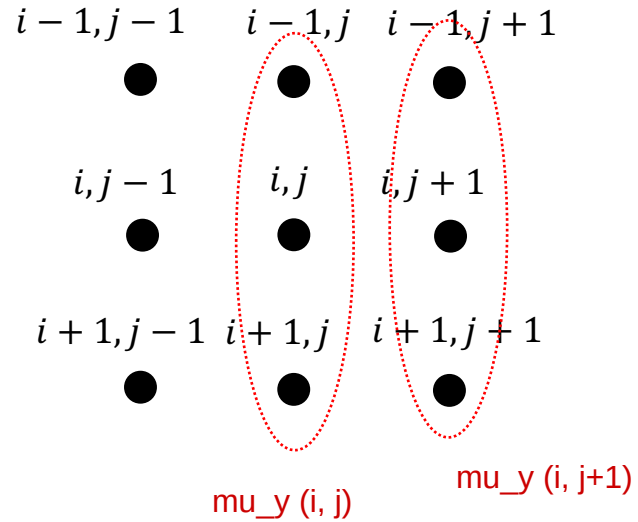
$$\mu_y(i,j) = \frac{1}{3}[I(i-1,j) + I(i,j) + I(i+1,j)]$$ take the average along the vertical axis

difference of averages along x-axis

$$\tilde{I}_x(i,j) = \underbrace{\mu_y(i,j+1)}_{\text{average}} - \underbrace{\mu_y(i,j)}_{\text{average}}$$ difference

then we choose as ideal noiseless value along the horizontal axis the difference between the average along the vertical axis of the next position and the current one

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



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

Explanation of the process:

1) The image is divided into smaller regions or windows. These windows can be of fixed size or variable size depending on the specific implementation.

2) Within each window, the average pixel value is computed. This average represents the "smoothed" version of the image within that window.

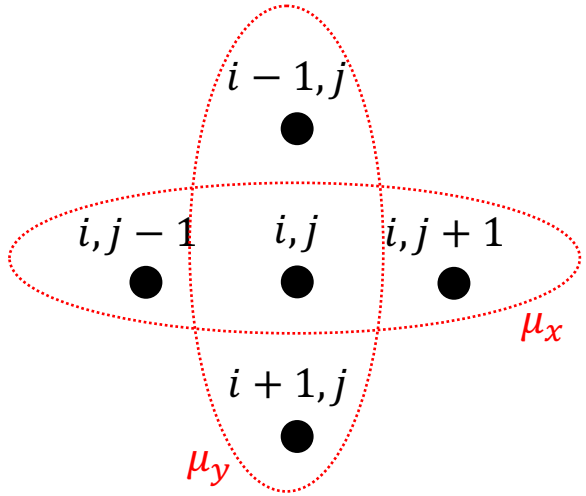
3) The differences between neighboring averages are calculated. Specifically, for each pair of neighboring windows, the absolute difference between their average pixel values is computed. These differences highlight regions where there are significant changes in intensity, potentially indicating edges.

Edges and noise

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

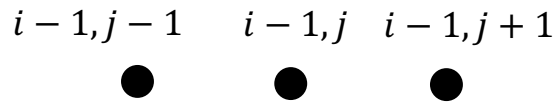
$$\mu_y(i,j) = \frac{1}{3}[I(i-1,j) + I(i,j) + I(i+1,j)]$$

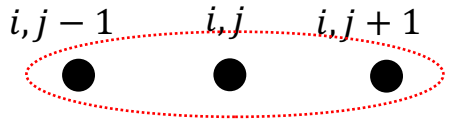
averages



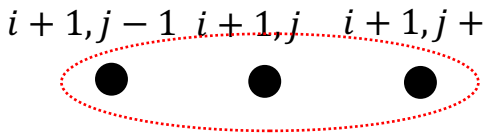
$$\tilde{I}_x(i,j) = \mu_y(i,j+1) - \mu_y(i,j)$$

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





mu_x(i, j)



mu_x(i+1, j)

difference of averages along y-axis

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

like before, to compute the ideal noiseless value along the vertical axis, we take the difference between the averages along the horizontal axis

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

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

Prewitt and Sobel

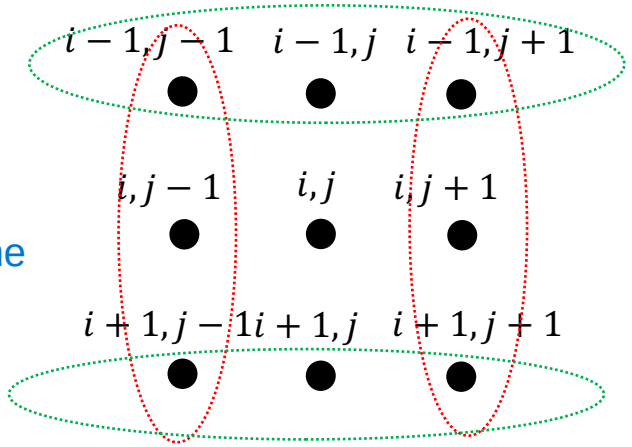
they are two of the most famous kernel / filters for edge detection

unlike before (where forward differences were used) Prewitt uses central differences to approx partial derivatives

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

the kernel is still positioned at i,j:

we want to compute the ideal noiseless value at i,j

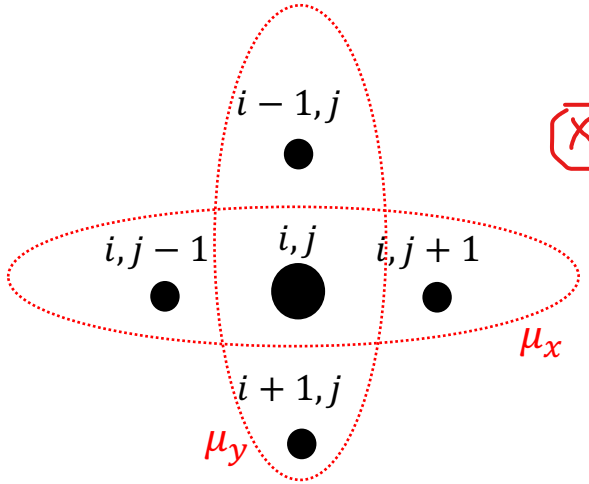


$$\tilde{I}_x(i,j) = \mu_y(i,j+1) - \mu_y(i,j-1) \quad \Rightarrow \quad \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

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

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

Sobel extends the idea above weighting more the pixels closer to the center



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

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

$$\Rightarrow \quad \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

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

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

$$\Rightarrow \quad \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

I divide by 4 because I count two times the central pixel in the average

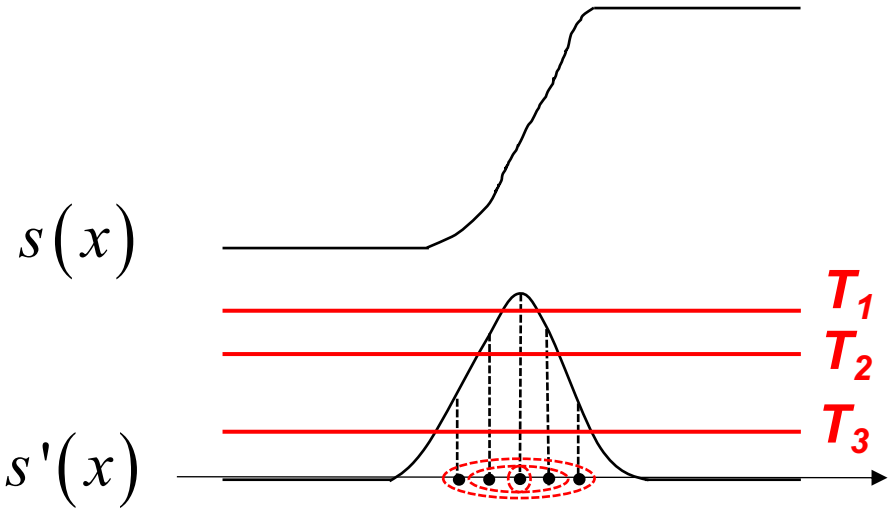
Edges Detection

LOCAL MAXIMA APPROACH:

Instead of relying solely on gradient thresholding, let's find the local maxima of the absolute value of the derivative of the image signal. In simpler terms, it means identifying points where the rate of change of intensity is maximized, which often corresponds to the presence of edges.

- 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”)

      many types of edges, some are more noticeable (stronger), some aren't that much (weaker)
  - Trying to detect weak edges implies poor localization of stronger ones



If I choose a lower treshold I can detect 3 pixels for a stronger edge: how can I solve this? there is no way of getting one pixel for one edge?

- A better approach to detect edges may consist in finding the *local maxima* of the absolute value of the derivative of the signal

select one representative pixel for only one "local" region

How many pixels we want to find for that edge?
one

A local maximum is a point where the derivative magnitude is greater than its neighboring points. This step involves scanning through the derivative image and identifying points where the magnitude is greater than both its immediate neighbors in all directions.

with this algorithm I can find local maxima suppressing the rest of the responses

Non-Maxima Suppresion (NMS)

it is a post-processing technique commonly used in edge detection algorithms to refine the detected edges and reduce them to thin lines along the true edges in the image. This helps to eliminate redundant or noisy edge responses

how it works:

1) in order to detect edges, the algorithm computes, in each pixel location, the gradient MAGNITUDE, which represents the strength of the edge at that point;
2) the algorithm calculates the gradient DIRECTION at each pixel, typically represented as an ANGLE;
3) for each pixel location where an edge is detected, the algorithm compares the gradient magnitude of that pixel with the magnitudes of its neighboring pixels in the direction of the gradient.
4) If the gradient magnitude of the current pixel is greater than the magnitudes of its neighbors in the direction of the gradient, it is retained as a potential edge point. Otherwise, if it is not greater than both of its neighbors along the gradient direction, it is SUPPRESSED (set to zero).

- 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)

gradient tells us the direction of the maximum variation of the function

$$\begin{matrix} 9 & 40 & 8 \\ 11 & 42 & 12 \\ 10 & 40 & 10 \\ 8 & 44 & 10 \end{matrix}$$

$$|\nabla I| = |I_x|$$

gradient magnitude

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{matrix}$$

NMS along x (correct)

$$\begin{matrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

e.g. 11 > 9, 42 > 40, 12 > 8

44 > 40

NMS along y (wrong !)

5) The result of non-maximum suppression is a binary image where only local maxima in the gradient direction are retained, while non-maxima are suppressed. This helps to thin out the detected edges and keep only the strongest responses along the true edges in the image.

- 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)

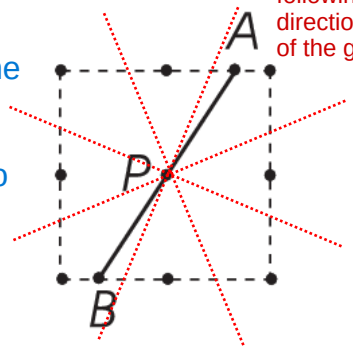
we will use gradient direction to decide along which direction we compute the local maxima through NMS

TWO METHODS: 1] quantization / discretization
2] interpolation / linear approximation

Non-Maxima Suppresion (NMS)

1] we quantize because we need discrete values, not continuous, given by the magnitude of the gradient

given the direction and the quantized angle, the other point involved in this local maxima computation is going to be the closest point in that quantized angle



we find A and B following the direction of the gradient

magnitude of the gradient computed in P, the point of interest at (i,j)

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

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

we approximate GA to the magnitude of the gradient of the top-right pixel

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

we approximate GB to the magnitude of the gradient of the bottom-left pixel

- for each pixel in the neighborhood, I draw a red line passing throuh P, the current pixel at pos i,j in which we're interested
- for each resulting region, I have a neighboring pixel
- then, I follow the direction given by the gradient, that gives the points A and B
- since the magnitude of the gradient is a cont. value, I discretize it approximating it to the nearest point in the neighborhood
- if the magnitude of the gradient in P is higher than the one fro A and B, then it means that following the direction of the gradient I do not reach any local maximum, because P is already the one in that neighborhood

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

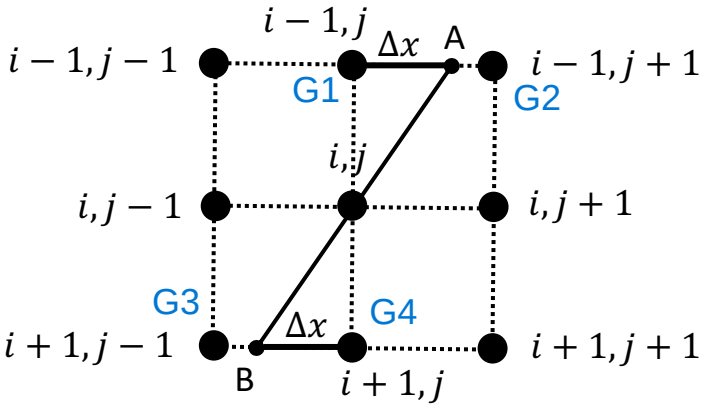
when G, the magnitude of the gradient at the current position i,j is going to be a local maxima? when it is higher than GA and GB

- The magnitude of the gradient has to be estimated at points which do not belong to the discrete pixel grid

we don't want this type of approximation given by 1], because the gradient in the approx points can be much different, then ...

- Such values can be estimated by linear interpolation of those computed at the closest points belonging to the grid

2] if we assume that the gradient between two points varies linearly (increase/decrease linearly), we can compute a linear INTERPOLATION between these two values by considering only pixels that I involve with the direction of the gradient



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

G2 is the most similar value to GA (the highest one, because the further I move away from P, the more the magn. of the gradient decreases)

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

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

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

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

weighted diff.

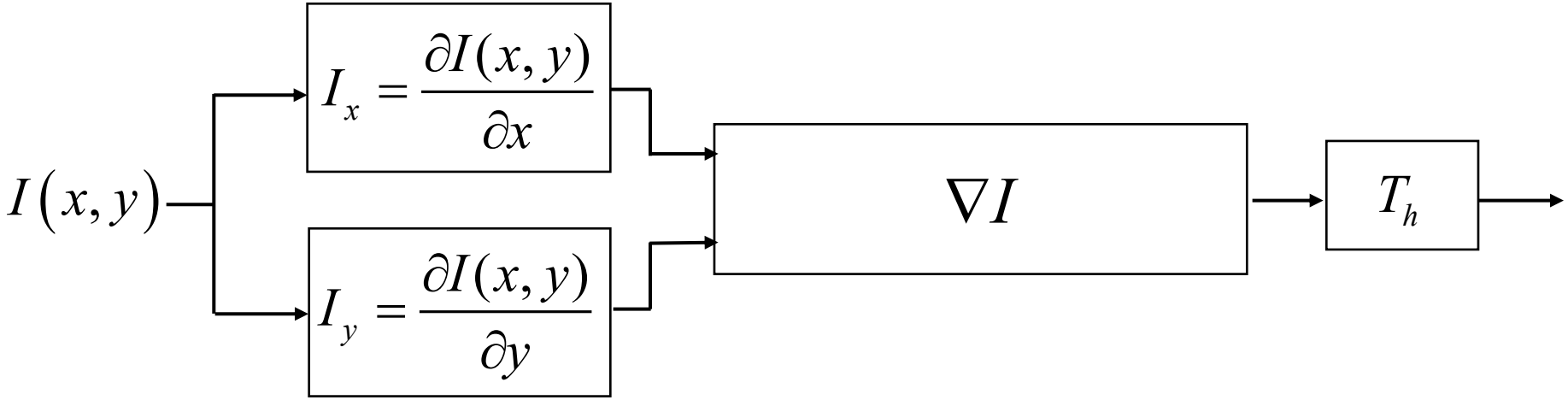
$$G_A \cong G_1 + (G_2 - G_1)\Delta x$$

we weight coz distance between (i-1,j) and (i-1, j+1) it's unitary

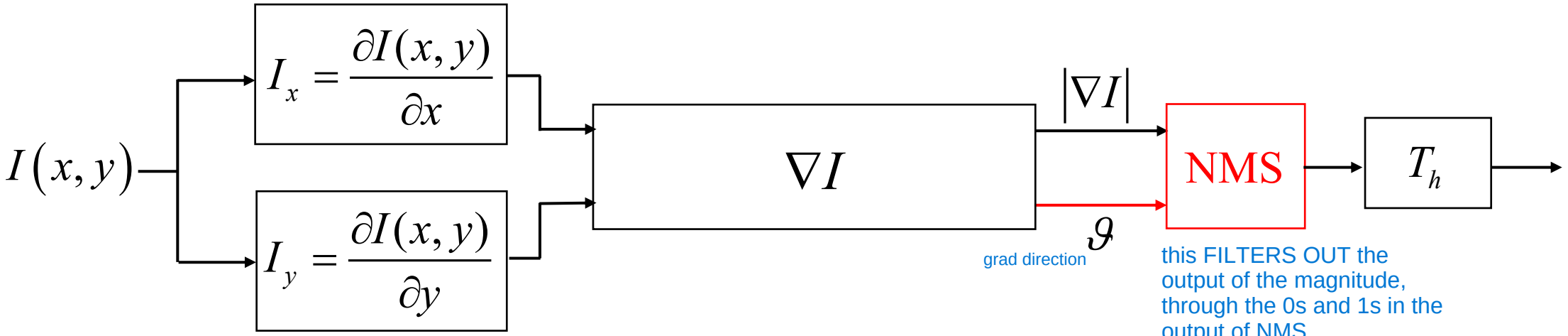
$$G_B \cong G_4 + (G_3 - G_4)\Delta x$$

compute a sort of linear combination

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

"found edge": the edge found by our algo

already said before

we don't want thick edges, only one pixel for one 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)).

minimum or maximum points

The first-order derivative provides information about how quickly the values of the Gaussian function are changing at different points.

- A straightforward Canny edge detector can be achieved by:
  - Gaussian smoothing
  - Gradient computation
  - NMS along the gradient direction

problems RECAP

we compute gradient to detect edges -> gradient amplify noise ->
-> smoothing to eliminate noise -> smoothing blurs edges

Canny's Edge Detector

convolution commutes with differentiation (the order in which we perform convolution and differentiation doesn't matter)

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

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

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

in opencv before applying canny's edge detector apply gaussian smoothing because it only applies sobel

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

exploiting separability

we compute 1D convolution

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

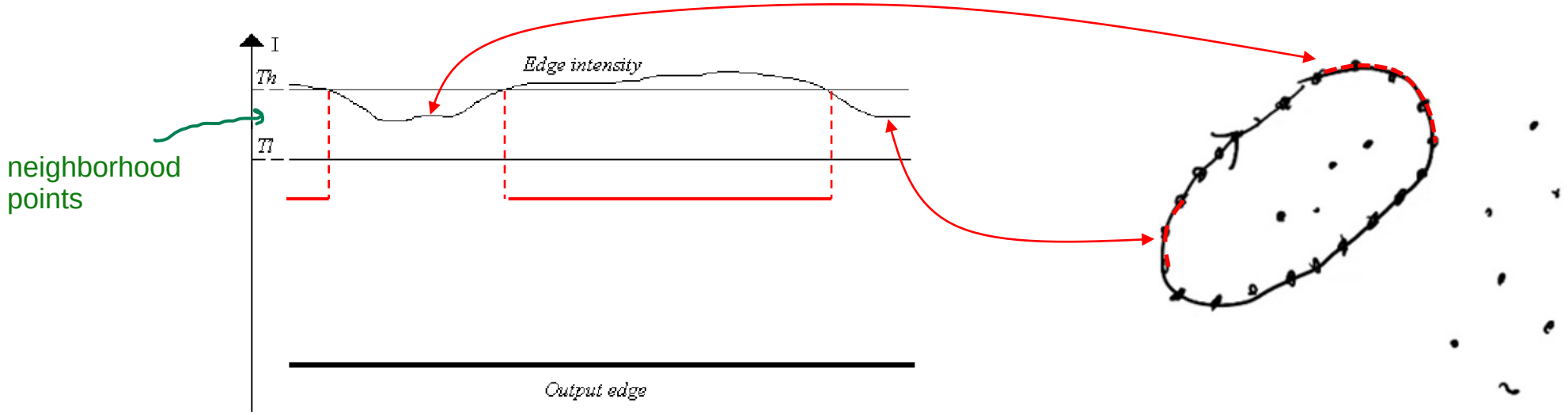
$$\tilde{I}_y(x,y) = I(x,y) * \left(G'(y)G(x)\right) = \left(I(x,y) * G'(y)\right) * G(x)$$

Canny's Edge Detector

sometimes edges along an object can appear with different intensities due to noise in the image or some changes of illumination during the acquisition

it can happen a phenomenon that decreases the intensity below the treshold and maybe increase later -> STREAKING

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

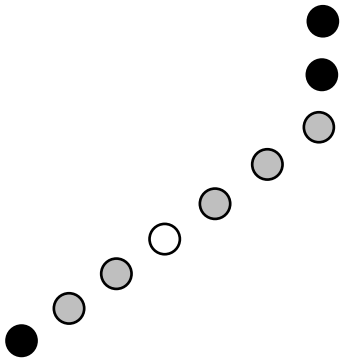


- Edge *streaking* may occur when magnitude varies along object contours
- Canny proposed a "*hysteresis*" thresholding approach relying on a higher (T_h) and a lower (T_l) threshold
  - A pixel is taken as an edge if either the gradient magnitude is higher than T_h OR higher than T_l AND the pixel is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

Canny's Edge Detector example

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

Edge candidates provided by NMS



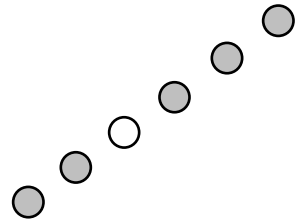
○ : $|\nabla I| > T_h$ strong edge point

● : $|\nabla I| > T_l$ neighbor of the edge points

● : $|\nabla I| < T_l$ no edge point, it has a gradient magnitude even below the neighboring treshold

Step-1: pick up all strong edges

Step-2: for each strong edge track weak edges along contours

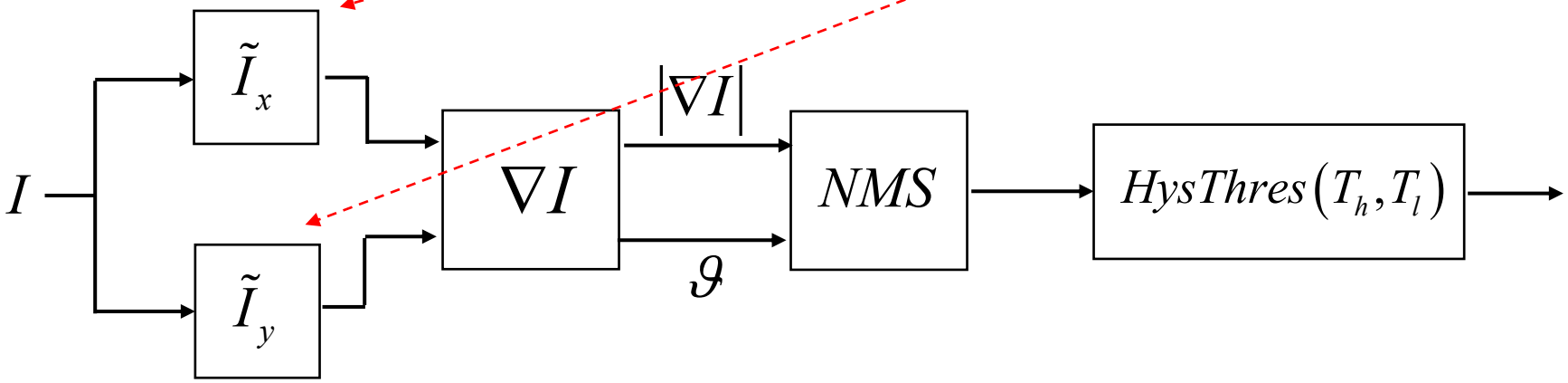


- You get a set of chains, each chain is a list of pixels belonging to that contour

Canny's Edge Detector

$$\tilde{I}_x = \left(I * G_x^{'}\right) * G_y$$

$$\tilde{I}_y = \left(I * G_y^{'}\right) * G_x$$

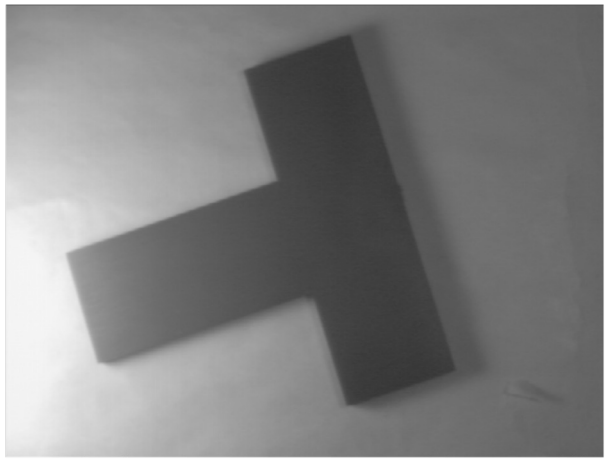


Chains of Connected Edge Pixels

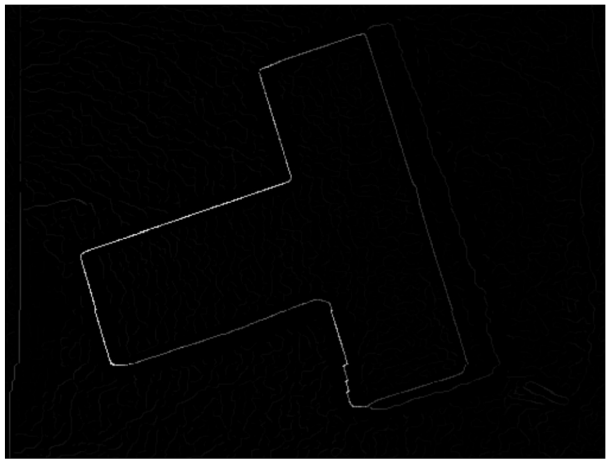
no longer an image as ouput

clear changes of intensities -> stronger and weaker edges

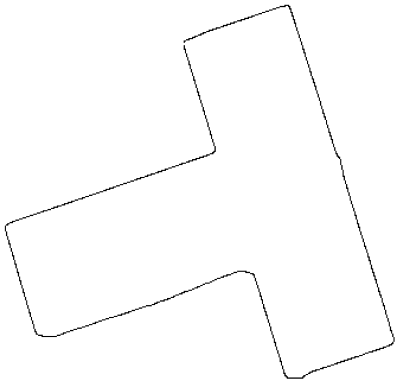
we are able to filter out shadows



Input



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



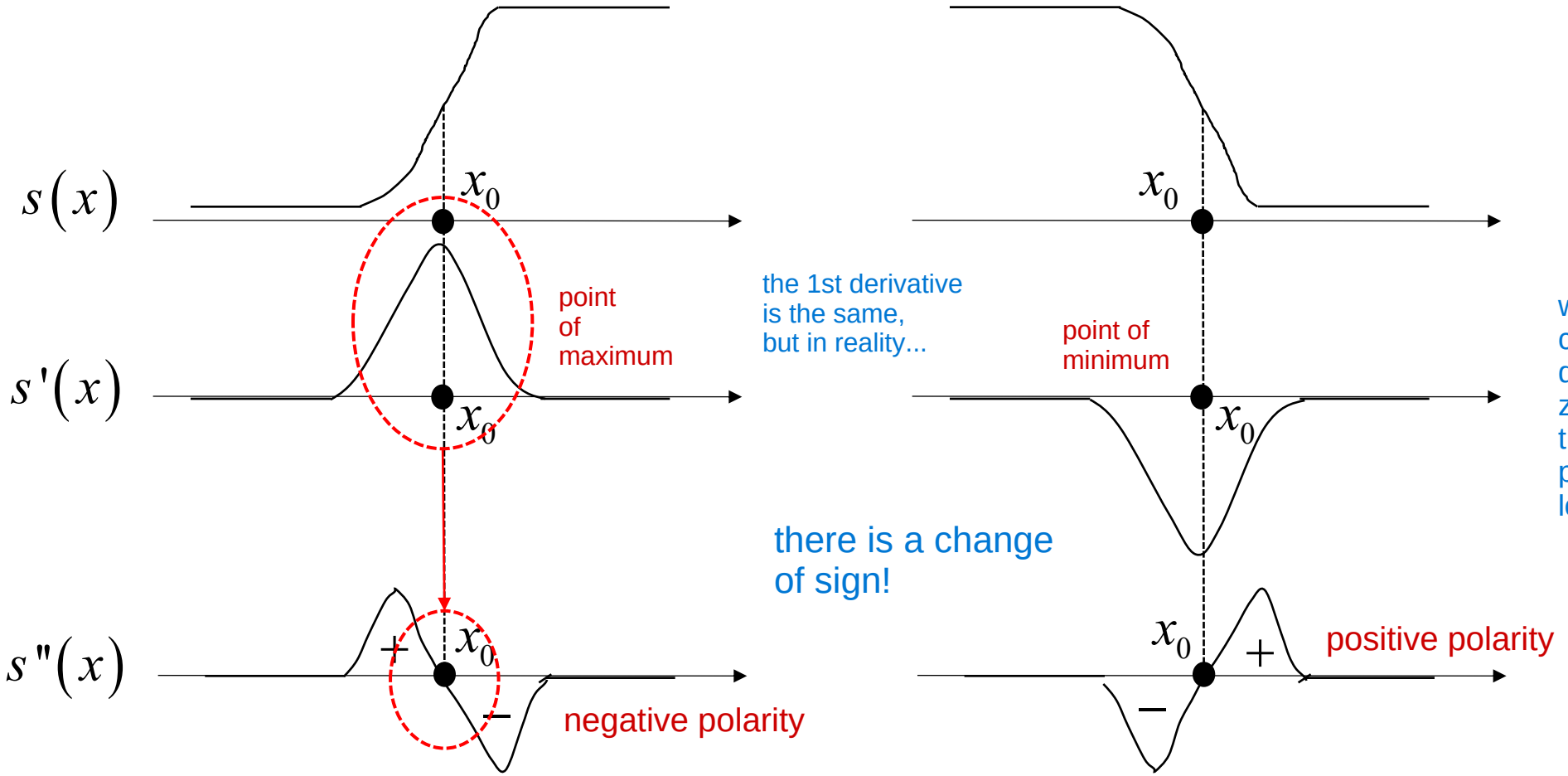
Final Output

remember: the main tool to compute edges is derivatives

Zero-crossing

we want to distinguish between changes of intensities: from low to high and viceversa

refers to a point in an image where the intensity changes sign as we traverse along a path



we look, when we compute the second derivative, at zero-crossing points: they indicates potential edge locations

- 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

we can avoid tresholding, because edges can be detected based on the natural variations in intensity, without the need for manual adjustment of threshold values.. but it comes with one drawback, heavy comput. effort

Second derivative along the gradient & Laplacian

The second derivative along a given direction provides crucial information about how the function behaves locally in that direction

- The second derivative along the gradient's direction can be obtained as $\mathbf{n}^T\mathbf{H}\mathbf{n}$ this results in a scalar
  it quantifies how the function's gradient changes as one moves along the direction of the gradient

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

(unit vector along the gradient's direction)

it represents the direction of the gradient
(of the steepest ascent/descent of the function)
with unitary magnitude

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

(Hessian matrix)

the Hessian provides information about the curvature of the function's surface in multiple dimensions (two in this case)

the process above is too expensive -> approximate using Laplacian

- 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:

the Laplacian measures the DIVERGENCE of the gradient of the function
(in this case the Intensity function)

- The divergence of a vector field measures the tendency of the vectors at a point to either converge towards or diverge away from that point, and it is calculated as the sum of the partial derivatives of its components

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

- If the divergence at a point is positive, the vector in the field (the gradient itself) is diverging away from that point. If it's negative, it is converging towards it. If it's zero, there's no net flow into or out of the point.

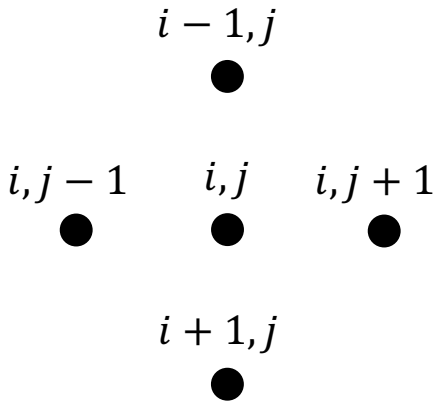
Geometrically, the Laplacian represents the rate at which the average value of f over a small sphere centered at a point changes with respect to the radius of the sphere. It measures the local curvature or second-order variation of the function.

Discrete Laplacian

Since edges in images correspond to regions of rapid intensity change, the Laplacian can effectively highlight these regions.

The Laplacian operator can be APPROXIMATED using discrete convolution with a Laplacian kernel.

- One can use forward and backward differences to approximate first and second order derivatives, respectively: using a combination of the two



$$I(i,j+1) - I(i,j) \qquad I(i,j) - I(i,j-1)$$

forward backward

$$I_{xx} \cong I_x(i,j) - I_x(i,j-1) = I(i,j-1) - 2I(i,j) + I(i,j+1)$$

backward

$$I_{yy} \cong I_y(i,j) - I_y(i-1,j) = I(i-1,j) - 2I(i,j) + I(i+1,j)$$

- 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

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

Now I can compute edges just convolving an image with a single kernel!

Laplacian of Gaussian (LOG)

but there is still a problem: as first-order derivatives, also second-derivatives amplify noise -> we still miss a smoothing step

- 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 the be summarized conceptually as follows:

1. Gaussian smoothing: $\tilde{I}(x,y) = I(x,y) * G(x,y)$ convolve the image with a gaussian kernel, to retrieve the "ideal noiseless image"
2. Second order differentiation by the Laplacian convolve the output of the smoothing, the noiseless image, with the previous laplacian kernel
3. Extraction of the zero-crossing of $\nabla^2 \tilde{I}(x,y)$ compute the zero-crossing, seeing where there is a change of sign

- 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…) there is an ISSUE: very difficult to see perfect zero crossing patterns in an image due to various factors (noise, etc.)
- You can find sign changes from minus to plus (and vice versa), like [-10,20], between two consecutive pixels

through the extraction of the zero crossing

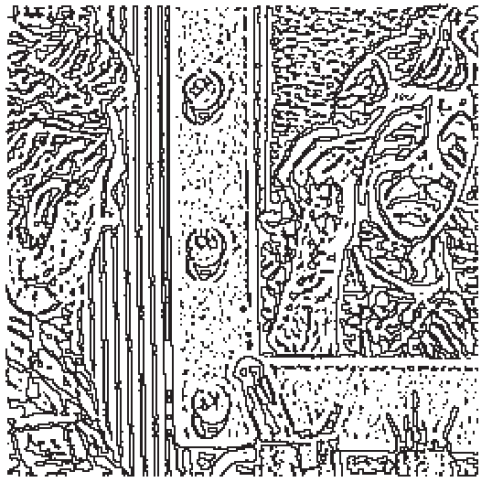
- Once a sign change is found, the actual edge may be localized:
  - At the pixel where the LOG is positive (darker side of the edge)
  - At the pixel where the LOG is negative (brighter side of the edge)
  - At the pixel where the **absolute** value of the LOG is smaller (the best choice, as the edge turns out closer to the "true" zero-crossing)

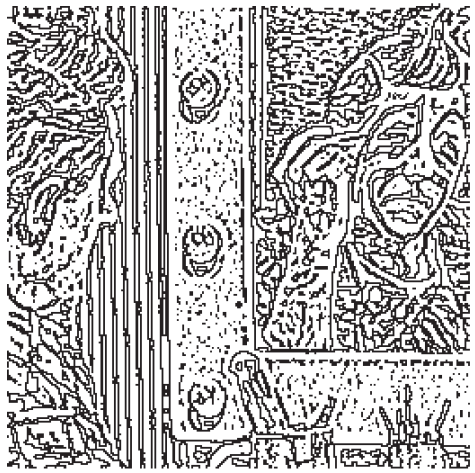
- To help discarding spurious edges, a final thresholding step may be enforced (usually based on the slope of the LOG at the found zero-crossing)

LOG application examples

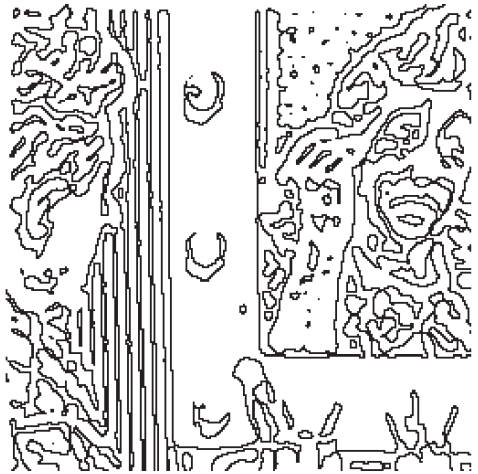
we can control the LEVEL of DETAIL of objects at which compute EDGES!!

$\sigma = 1$

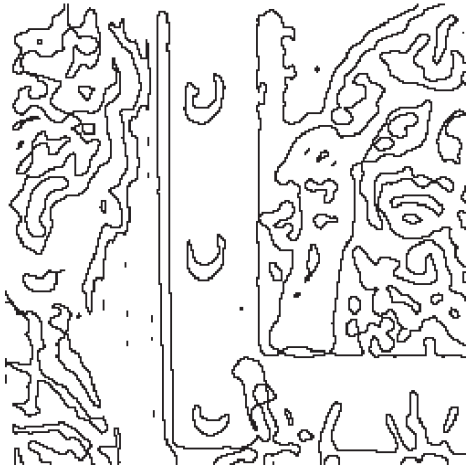




$\sigma = 2$



$\sigma = 3$



if we use larger sigma, we compute edges for larger objects, with a lower level of detail

Use small sigma and then apply a threshold

we can do that because we apply the gaussian smoothing before the edge detection process;
in fact, sigma determines the scale of the smoothing

obv, the more is the smoothing, the less is the level of detail

therefore, we will retrieve less edges

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 σ 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 -> larger σ)

- Likewise, σ may be used to control the scale at which the image is analyzed, with larger σ typically chosen to extract the edges related to main scene structures and smaller σ 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