Digital Image Filters

Giacomo Boracchi

CVPR USI, March 31 2020

Book: GW chapters 3, 9, 10

Outline

Local Image Transformations

- **Derivatives estimation**
- Nonlinear Filters

Edge Detection (Canny Edge Detector)

Derivatives Estimation

Differentiation and convolution

Recall the definition of derivative

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x+\epsilon) - f(x_n)}{\epsilon} \right)$$

Now this is linear and shift invariant.

Therefore, in discrete domain, it will be represented as a convolution

Differentiation and convolution

Recall the definition of derivative

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x+\epsilon) - f(x_n)}{\epsilon} \right)$$

We could approximate this as

$$\frac{\partial f}{\partial x}(x_n) \approx \frac{f(x_{n+1}) - f(x_n)}{\Delta x}$$

which is obviously a convolution against the Kernel [1 -1];

$I \mapsto I_x$

$I_x = I * [1, -1]$

Now this is linear and shift invariant.

Therefore, in discrete domain, it will be represented as a convolution

Finite Differences in 2D (discrete) domain

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0}\left(\frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon} \right)$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{\varepsilon \to 0}\left(\frac{f(x, y+\varepsilon) - f(x,y)}{\varepsilon} \right)$$

Horizontal

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

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x_{n+1}, y_m) - f(x_n, y_m)}{\Delta x}$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x_n, y_{m+1}) - f(x_n, y_m)}{\Delta x}$$

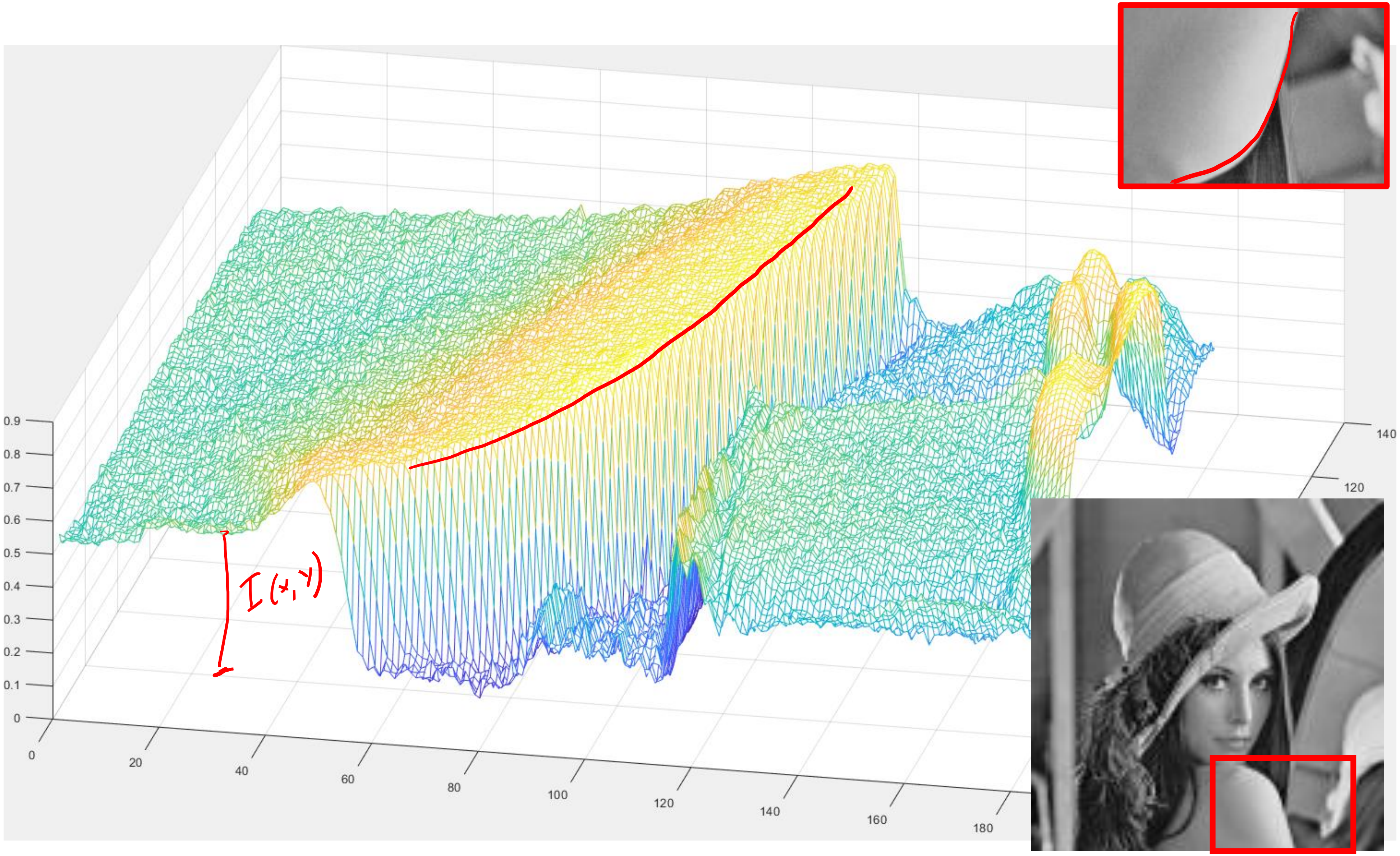
Vertical

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

Discrete Approximation

Convolution Kernels

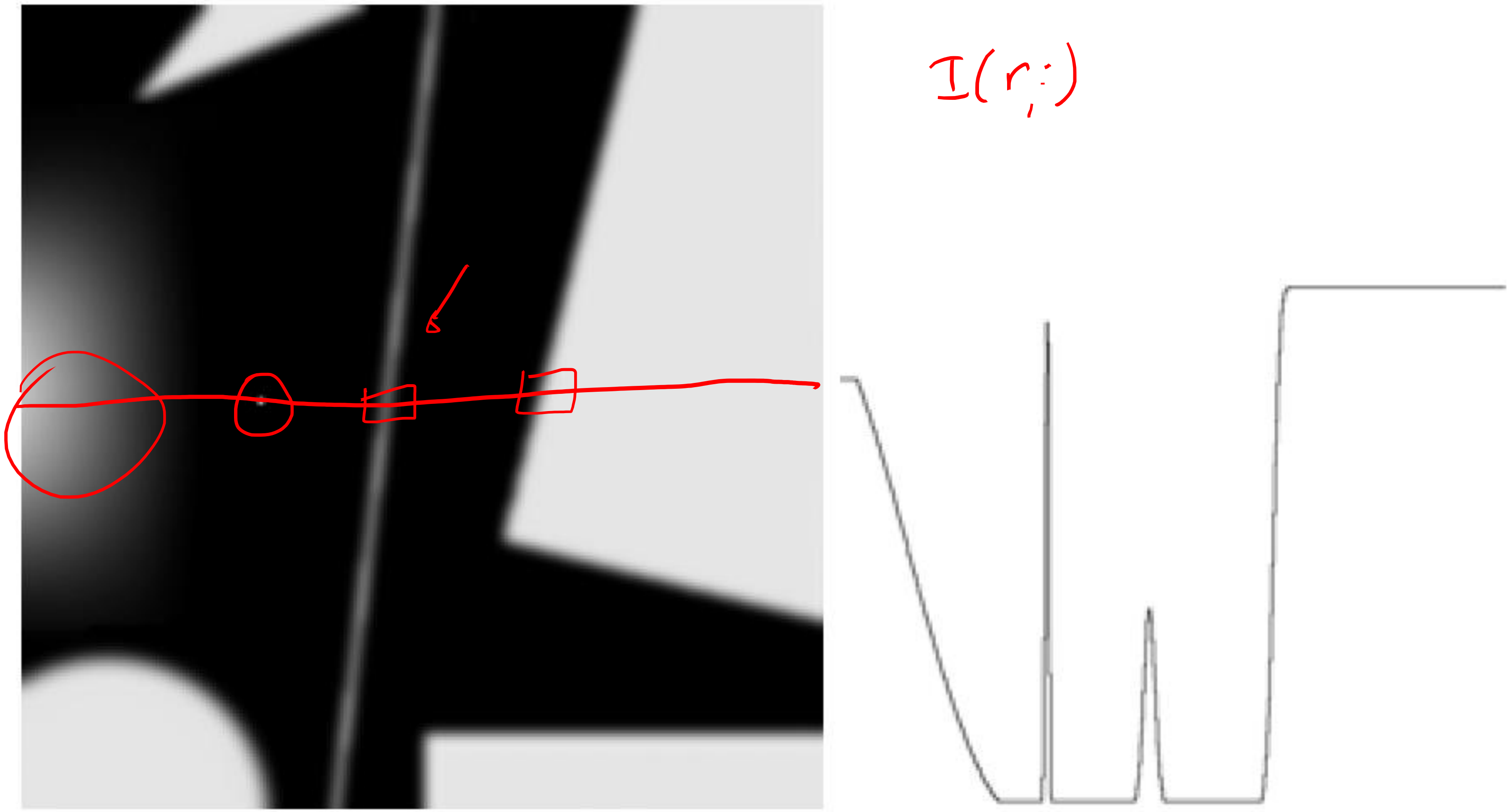
Think of an image as a 2d, real-valued function



Giacomo Boracchi

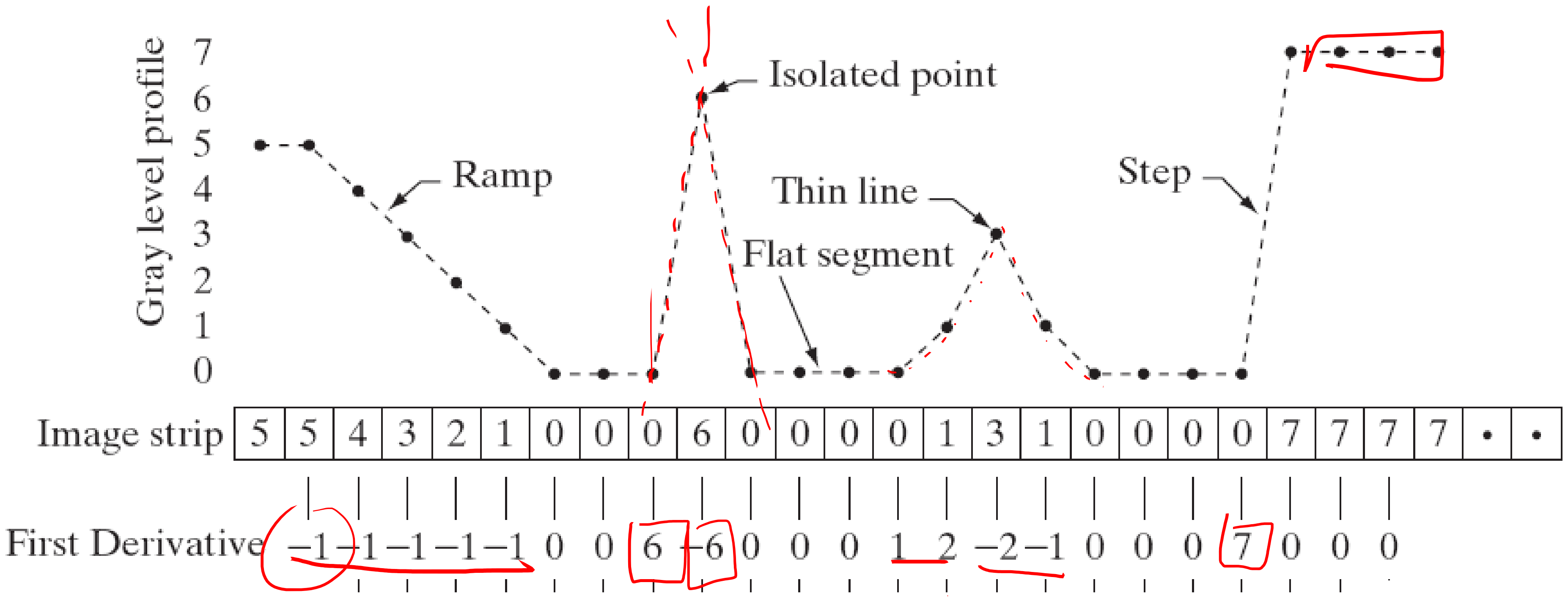
A 1D Example

Take a line on a grayscale image



A 1D Example (II)

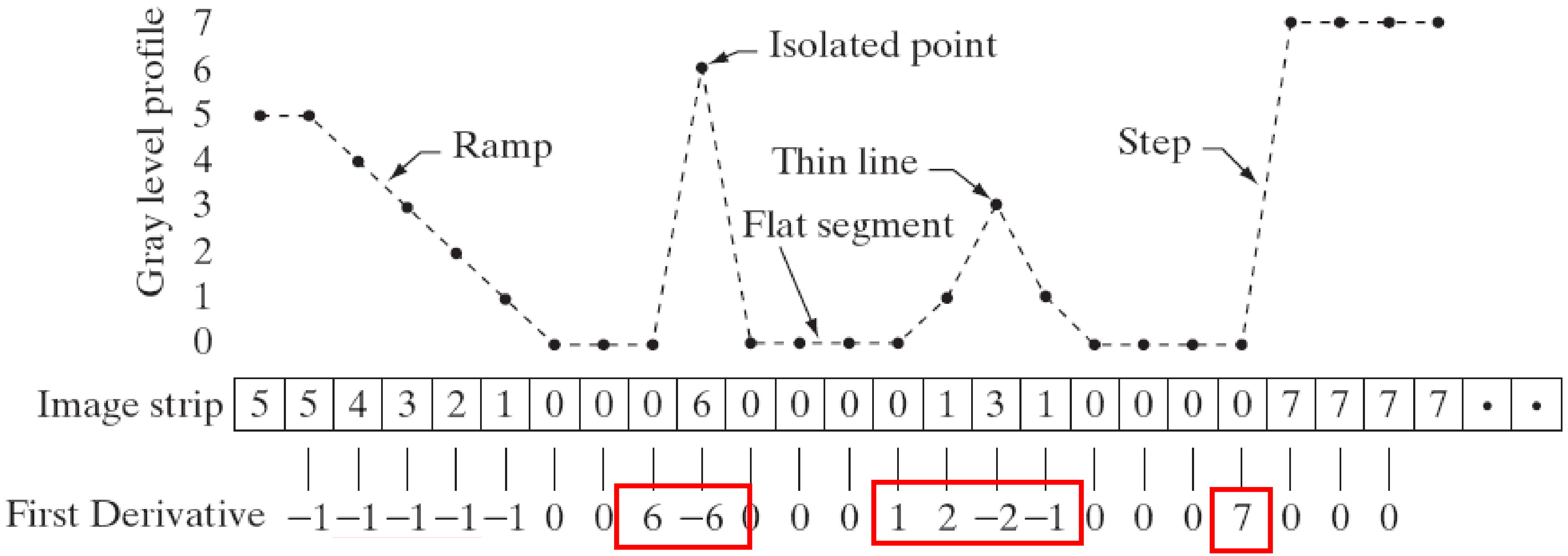
Filter the image values by a convolution against the filter [1 -1]



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Derivatives

Derivatives are used to **highlight intensity discontinuities** in an image and to deemphasize regions with slowly varying intensity levels



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Differentiating Filters

The derivatives can be also computed using centered filters:

$$f_x(x) = f(x-1) - f(x+1)$$

Such that the horizontal derivative is:

$$f_x = f \otimes \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline \end{array}$$

While the vertical derivative is:

$$f_y = f \otimes \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline \end{array}^{\,t}$$

Classical Operators: Prewitt

Horizontal derivative

$I * (s * dx)$

$I * h_x$

$$s = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad dx = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad h_x = s \circledast dx = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Smooth   Differentiate

Vertical derivative

$$s = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad dy = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad h_y = s \circledast dy = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Classical Operators: Sobel

$\text{sum}(h_x(:)) = 0$

Horizontal derivative

$[-1 \ \ +1]$

$$s = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 1 & 1 \end{bmatrix} \qquad dx = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad h_x = s \circledast dx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Smooth — Differentiate

conv2(s, dx)

Vertical derivative

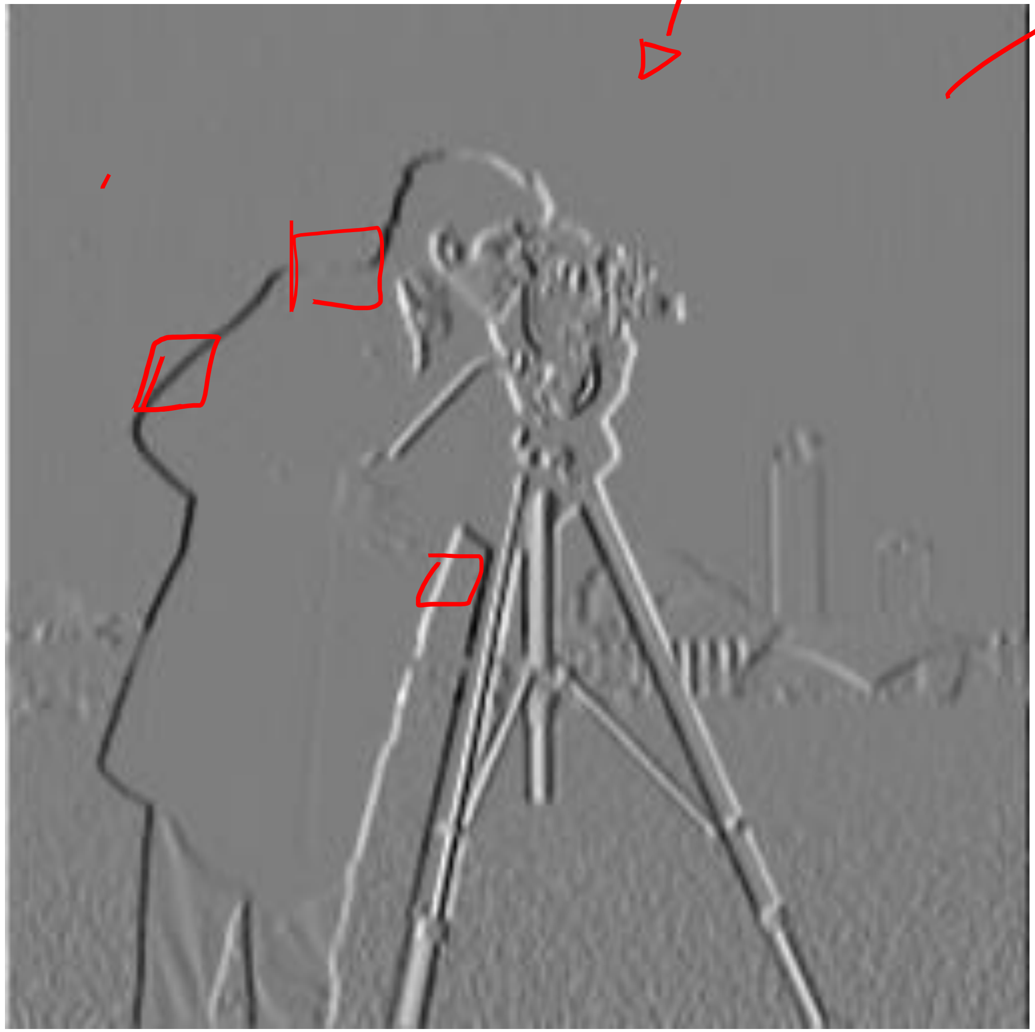
$$s = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 2 & 1 \end{bmatrix} \qquad dy = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad h_y = s \circledast dy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Another famous test image - cameraman

Horizontal Derivatives using Sobel

$$\nabla I_x = (I \circledast d_x)$$

$$\boldsymbol{\nabla} I(r,c) = \begin{bmatrix} \nabla I_x(r,c) \\ \nabla I_y\,(r,c) \end{bmatrix}$$

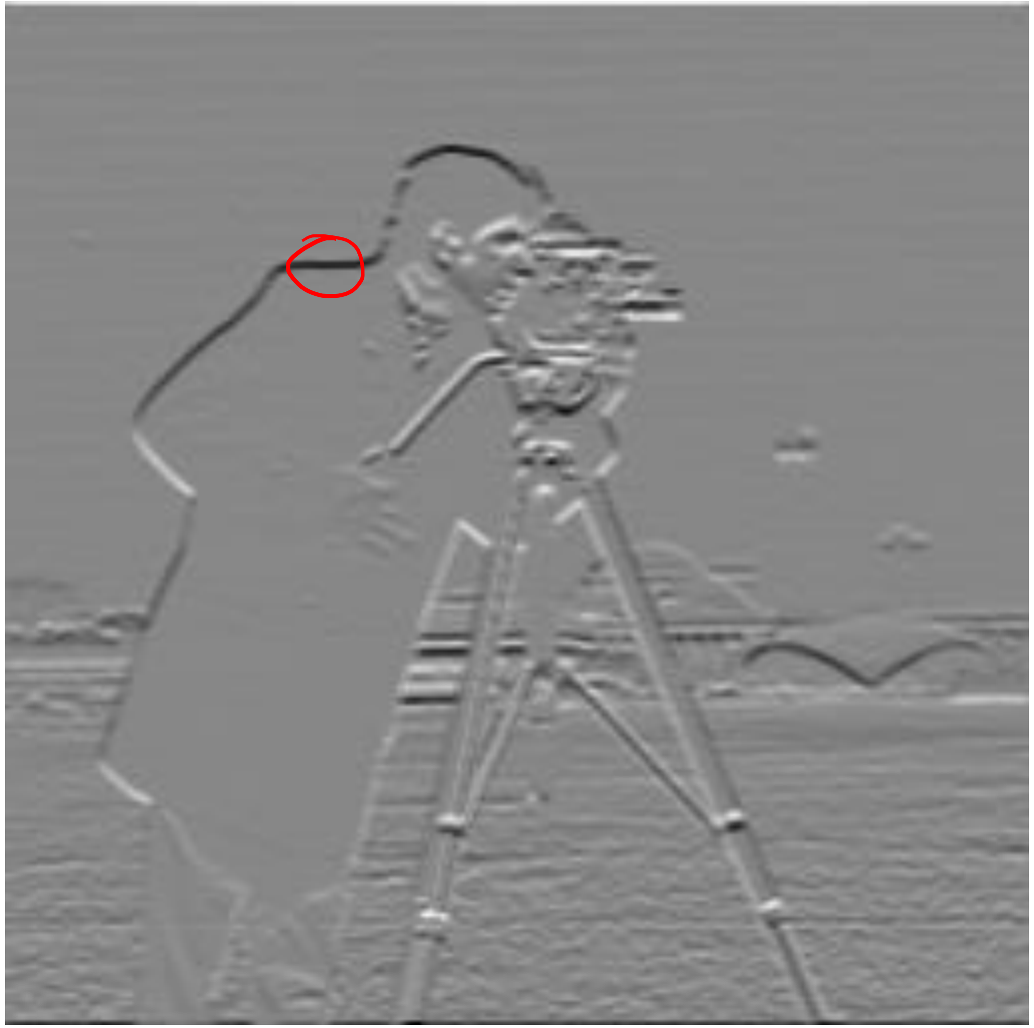


Vertical Derivatives using Sobel

$$\nabla I_y = \left(I \circledast d_y\right)$$

$$d_y = d_x{}'$$

$$\boldsymbol{\nabla} I(r,c) = \begin{bmatrix} \nabla I_x(r,c) \\ \nabla I_y\,(r,c) \end{bmatrix}$$



Giacomo Boracchi

Gradient Magnitude

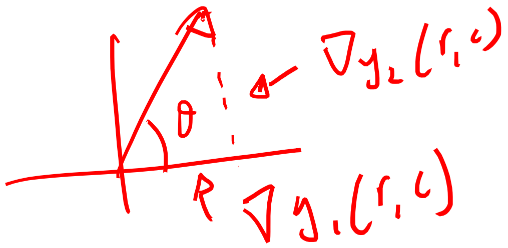
$$\|\nabla I\| = \sqrt{(I \circledast d_x)^2 + (I \circledast d_y)^2}$$

$$\boldsymbol{\nabla} I(r,c) = \begin{bmatrix} \nabla I_x(r,c) \\ \nabla I_y(r,c) \end{bmatrix}$$

The Gradient Orientation

Like for continuous function, the gradient in each pixel points at the **steepest growth/decrease direction.**

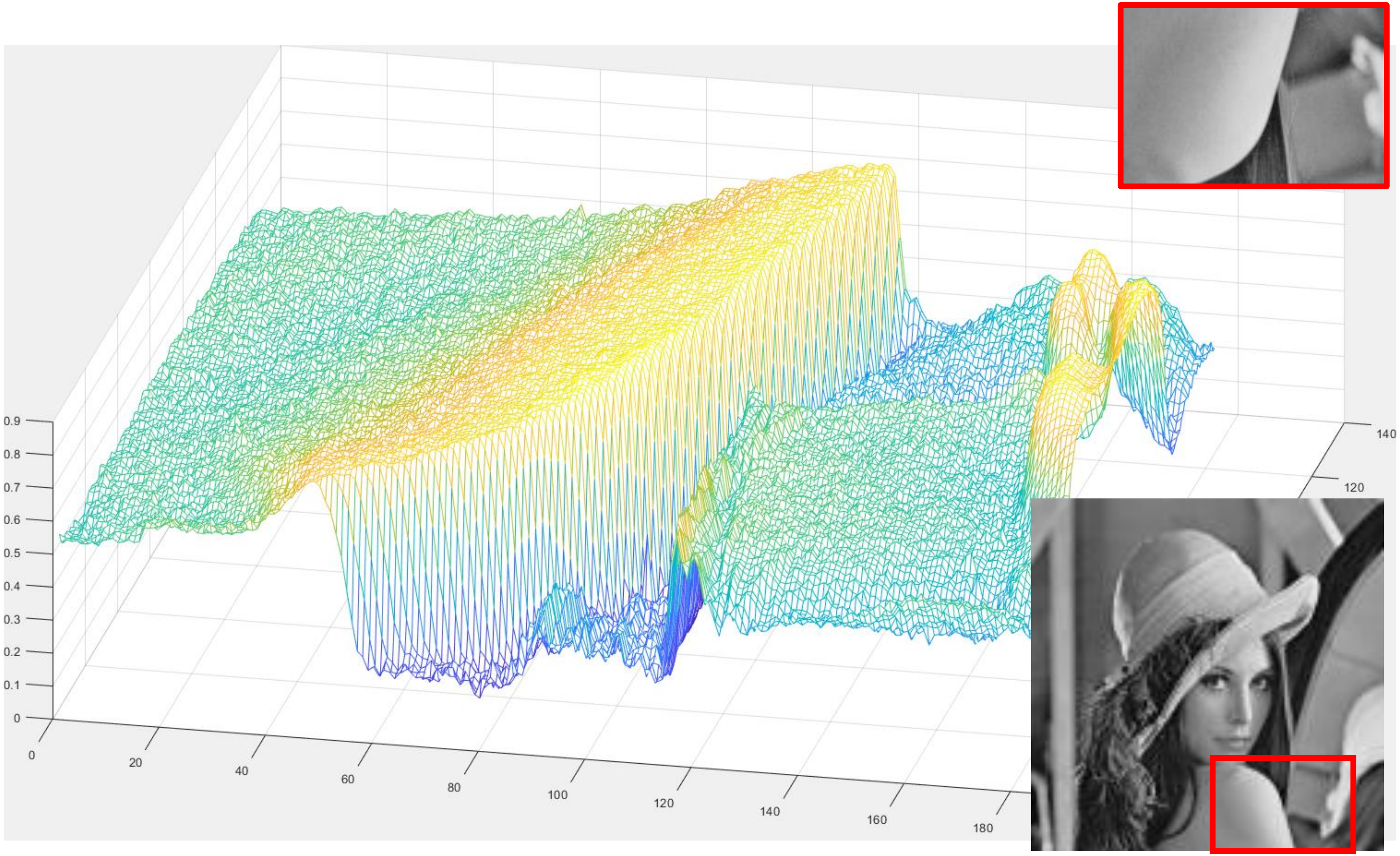
$$\angle \nabla I(r,c) = \text{atand}\left(\frac{\nabla I_2(r,c)}{\nabla I_1(r,c)}\right) = \text{atand}\left(\frac{(I \circledast d_x)(r,c)}{(I \circledast d_y)(r,c)}\right)$$



The gradient norm indicates the strength of the intensity variation

Let's switch to Matlab…..

Think of an image as a 2d, real-valued function



The Image Gradient

Image Gradient is the gradient of a real-valued 2D function

$$\nabla I(r,c) = \begin{bmatrix} I \circledast d_x \\ I \circledast d_y \end{bmatrix}(r,c)$$

where principal derivatives are computed through convolution against the derivative filters (e.g. Prewitt)

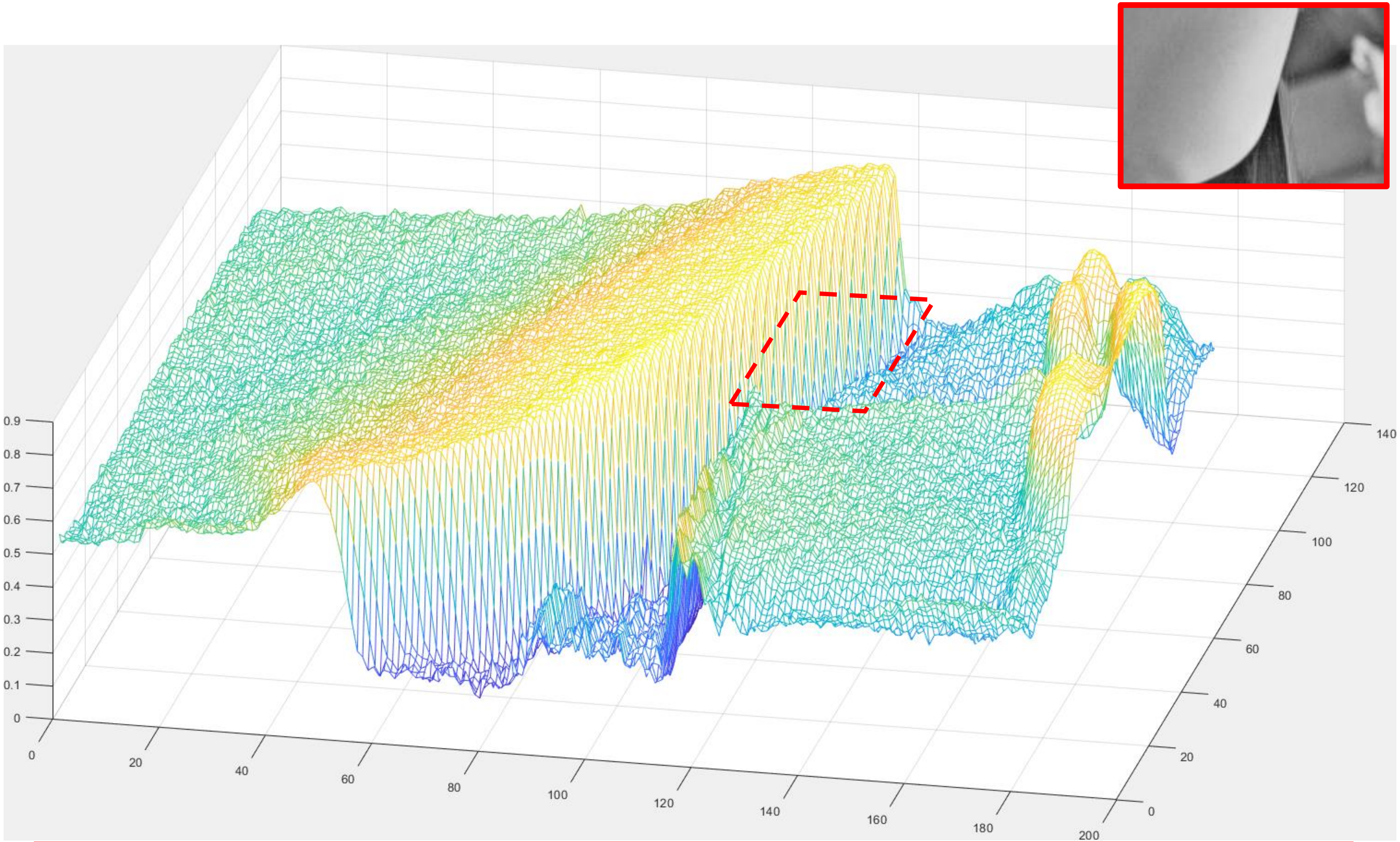
$$dx = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \qquad dy = dx'$$

Image gradient behaves like the gradient of a function:

$|\nabla I(r,c)|$ is large where there are large variations

$\angle \nabla I(r,c)$ is the direction of the steepest variation

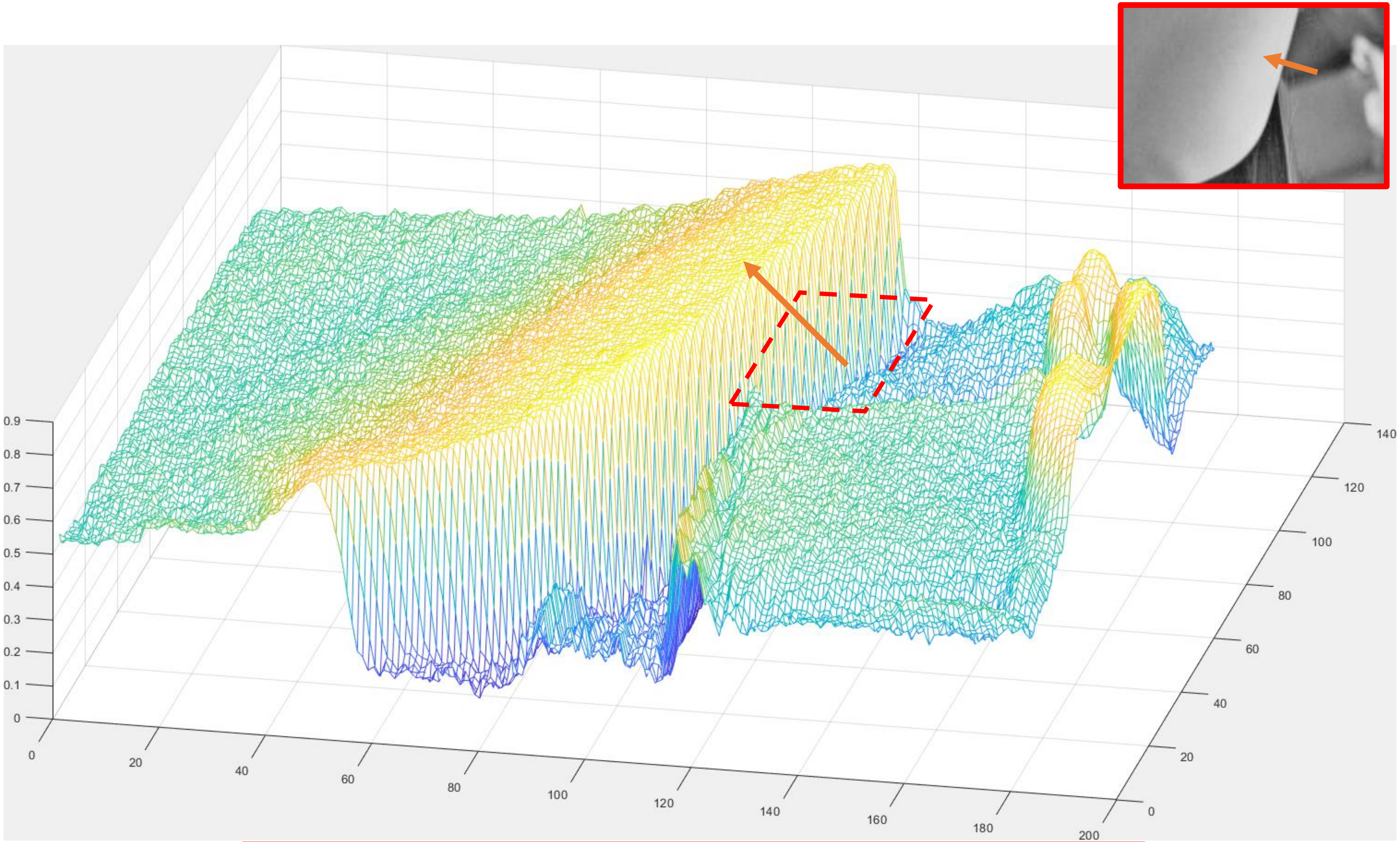
Think of an image as a 2d, real-valued function



Local spatial transformations are defined over neighborhood like this

Giacomo Boracchi

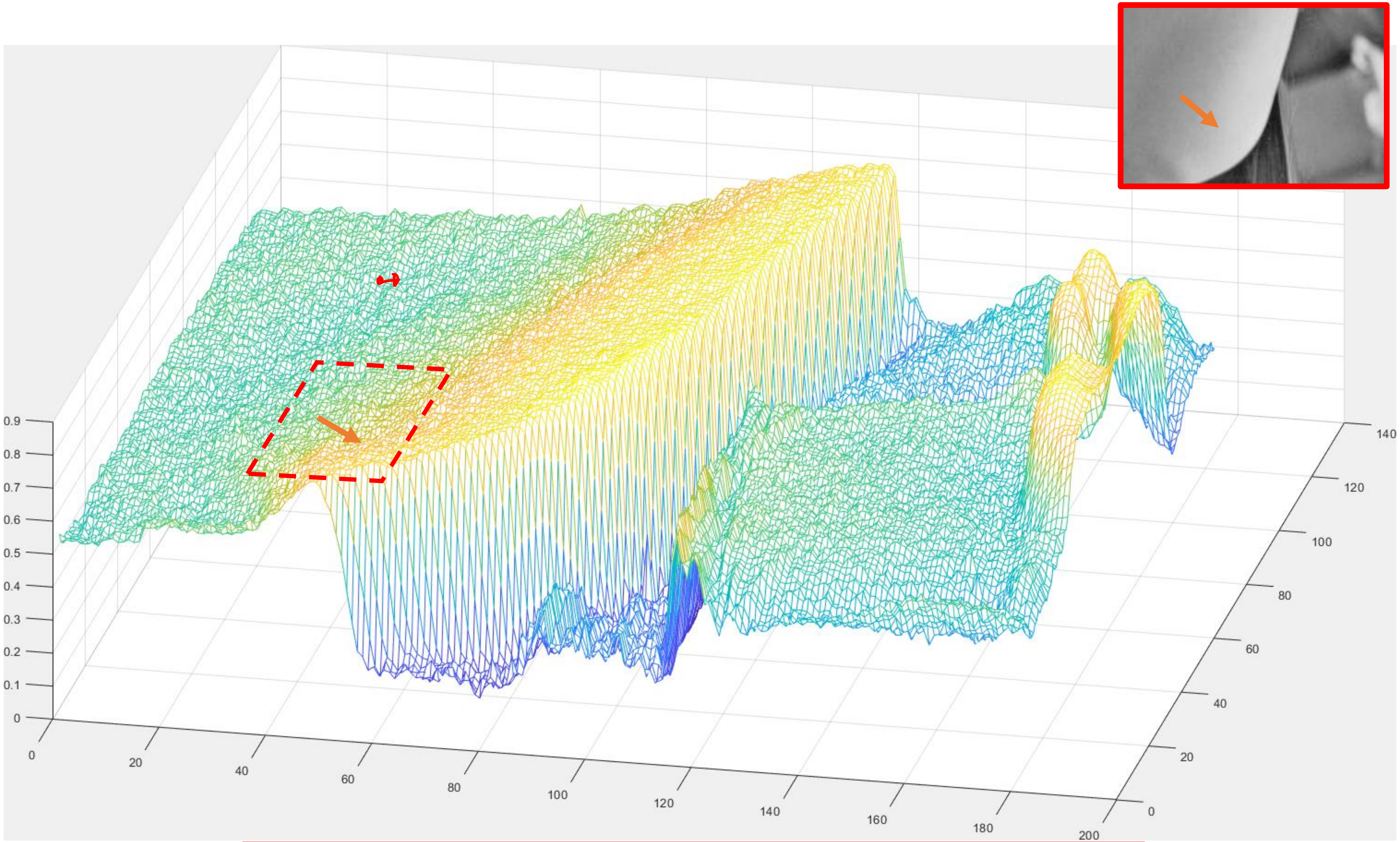
Think of an image as a 2d, real-valued function



What about the gradient in this neighborhood?

Giacomo Boracchi

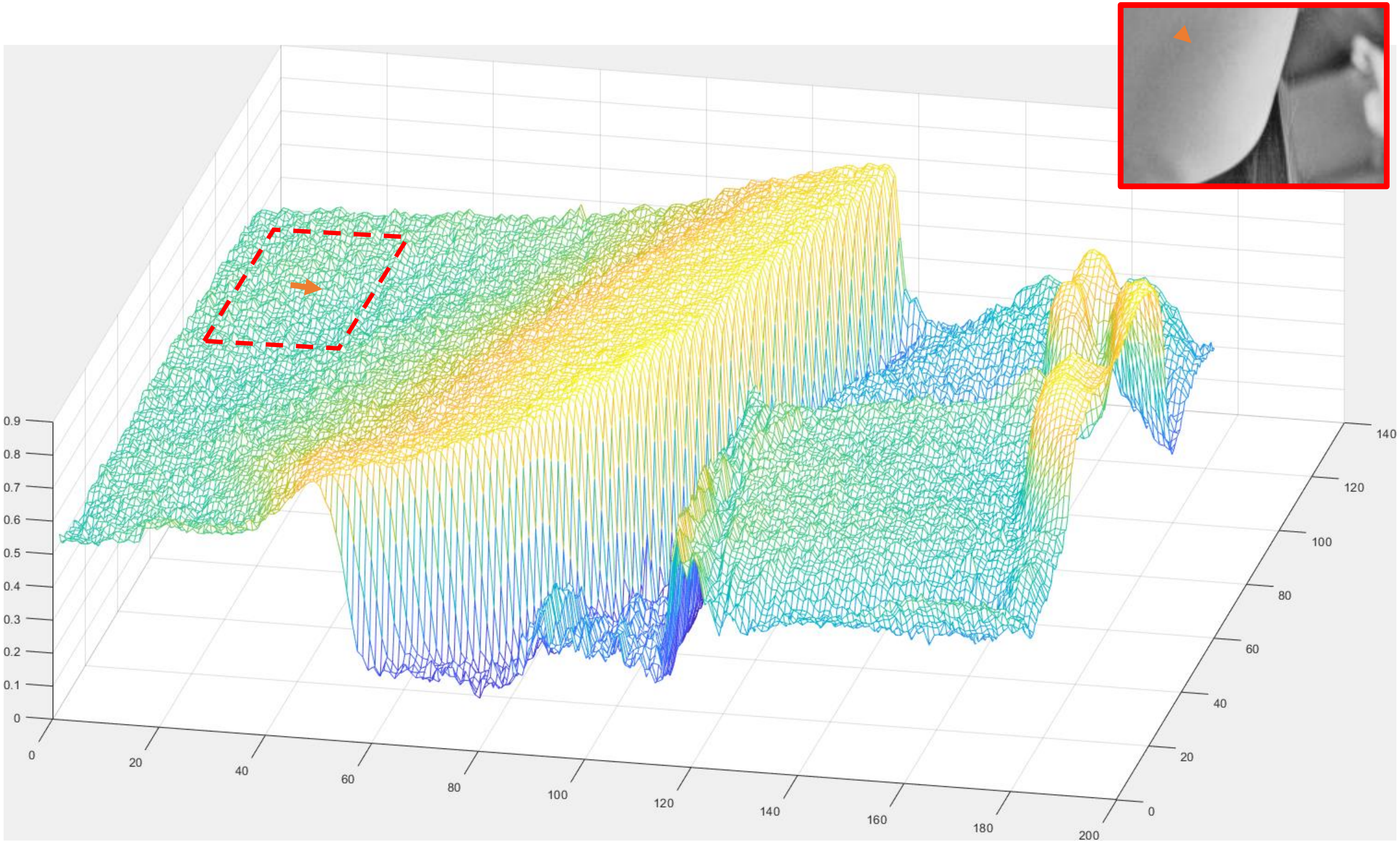
Think of an image as a 2d, real-valued function



What about the gradient in this neighborhood?

Giacomo Boracchi

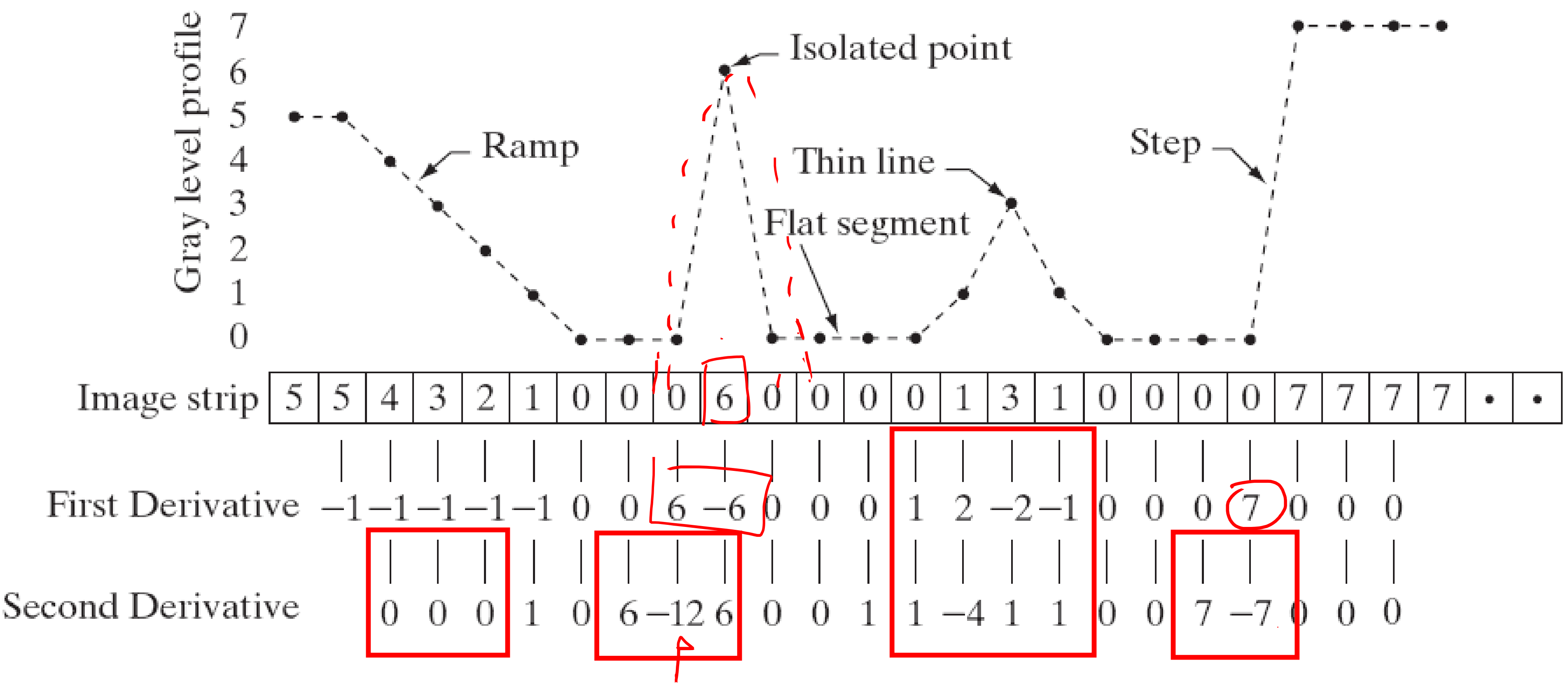
Think of an image as a 2d, real-valued function



Higher Order Derivatives

Derivatives

Derivatives are used to highlight intensity discontinuities in an image and to deemphasize regions with slowly varying intensity levels



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Second Order Derivatives

The Laplacian of the second order derivative is defined as

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

where

$$\frac{\partial^2 I}{\partial x^2} = I(x+1, y) + I(x-1, y) - 2I(x, y)$$

$$\frac{\partial^2 I}{\partial y^2} = I(x, y-1) + I(x, y+1) - 2I(x, y), \text{ thus}$$

$$\nabla^2 I = I(x+1, y) + I(x-1, y) + I(x, y-1) + I(x, y+1) - 4I(x, y)$$

It's a linear operator -> it can be implemented as a convolution

TODO: prove that the second order derivatve is like this

Filter for Digital Laplacian

The Laplacian of the second order derivative is defined as

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

0	1	0
1	-4	1
0	1	0

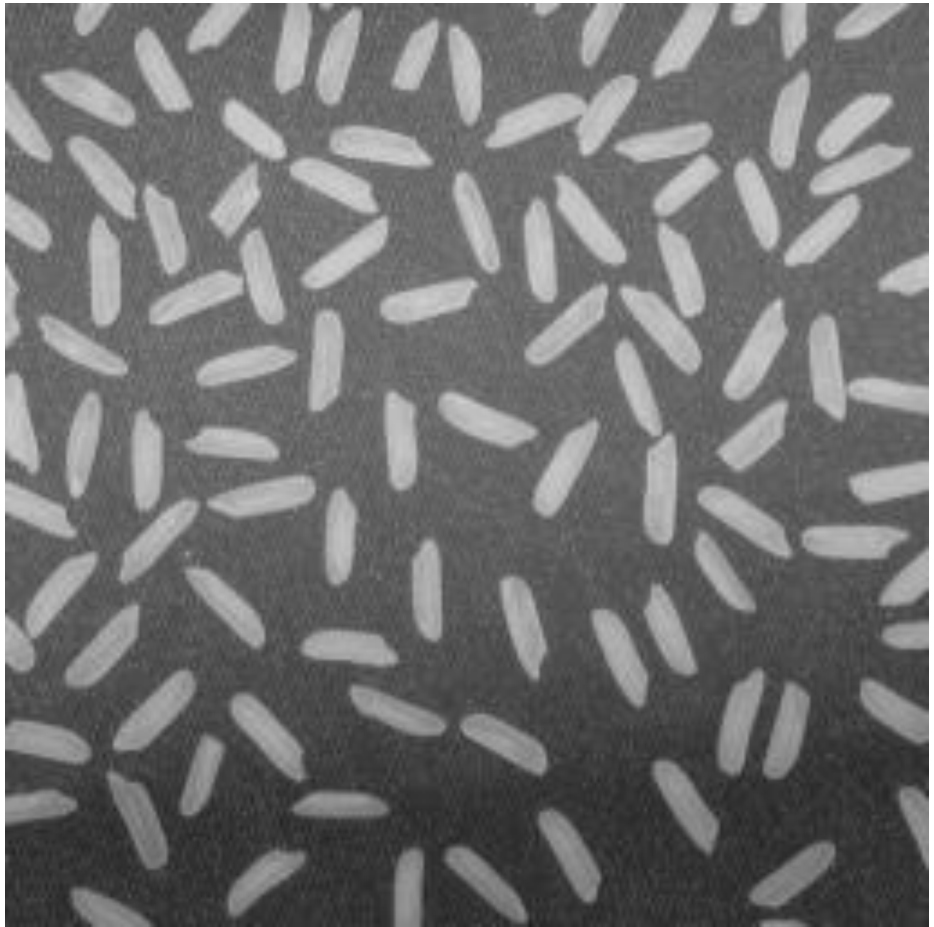
Standard definition, inviariant to 90° rotation

1	1	1
1	-8	1
1	1	1

Alternative definition, inviariant to 45° rotation

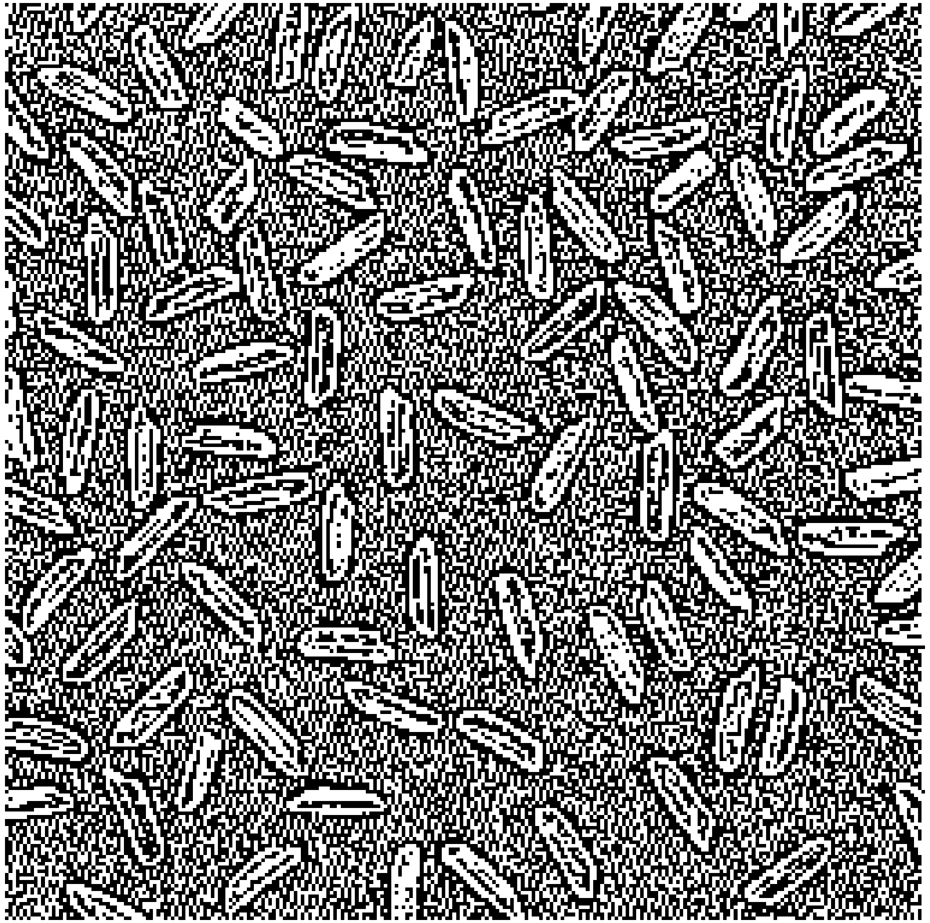
The Laplacian: Image Sharpening

The Laplacian of an image have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background.



The Laplacian: Image Sharpening

The Laplacian of an image have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background.



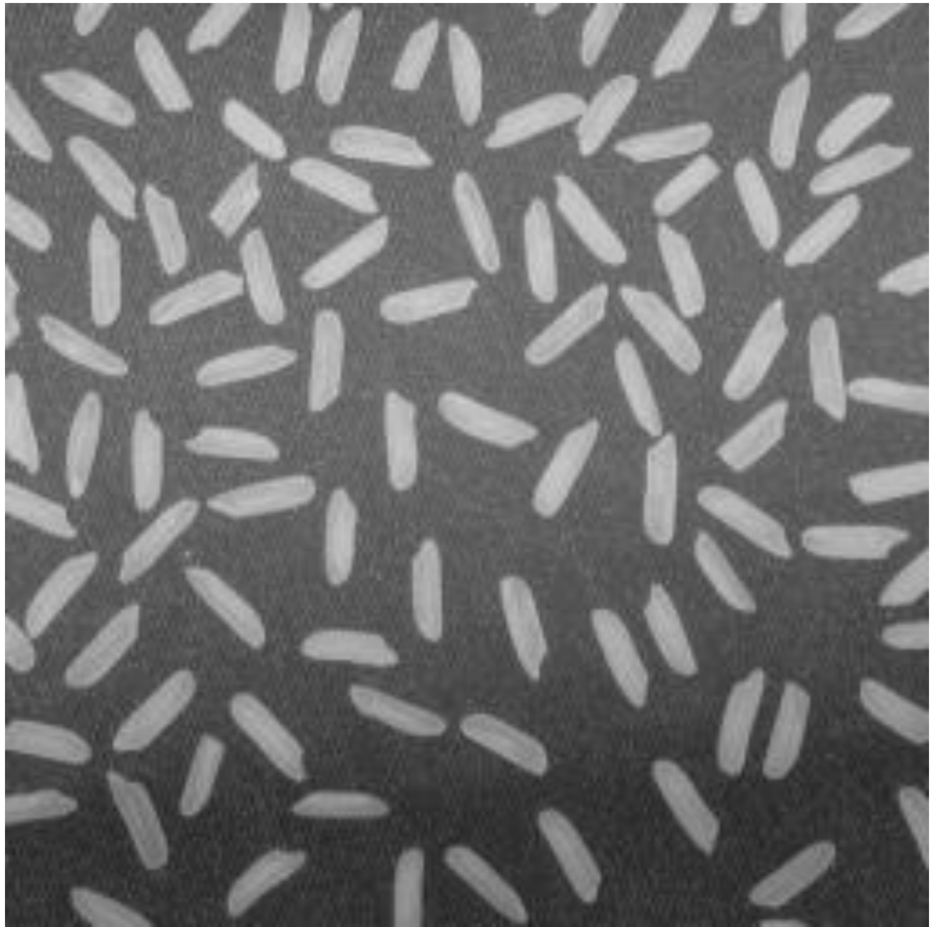
$I * \ell$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The Laplacian: Image Sharpening

Background features can be “recovered” simply by adding the Laplacian image to the original (provided suitable rescaling)

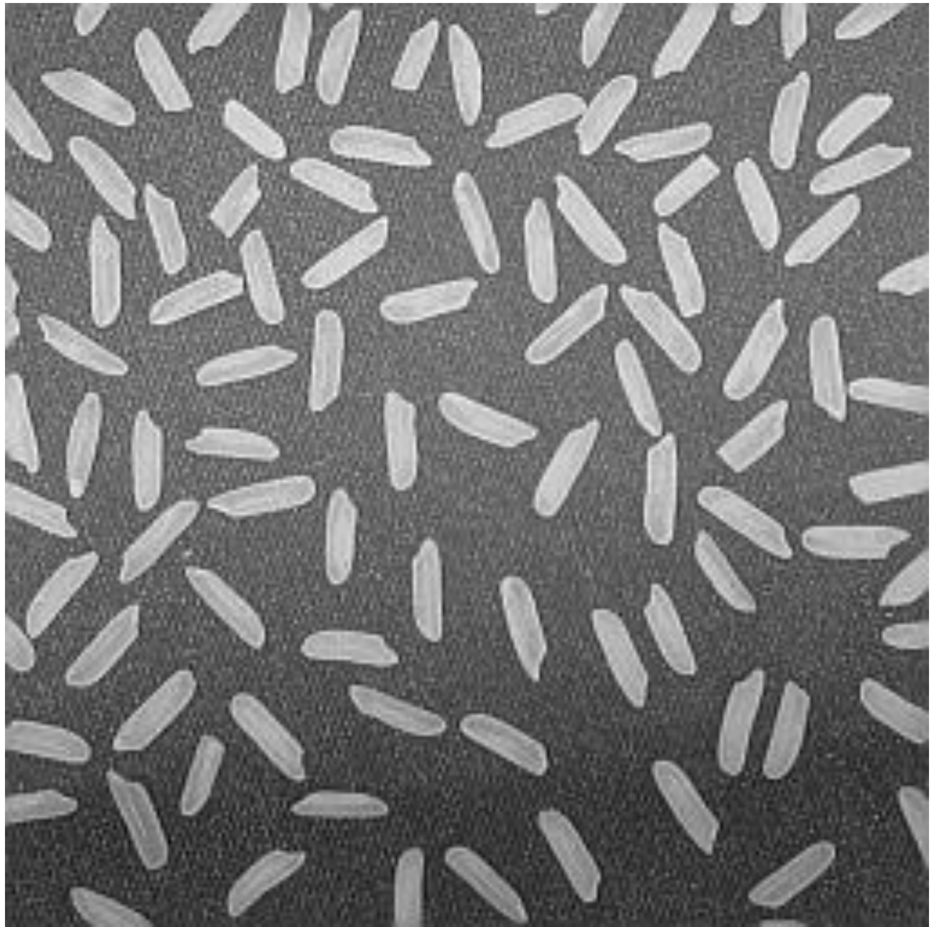
$$G(r,c) = I(r,c) + k[\nabla^2 I(r,c)]$$



The Laplacian: Image Sharpening

Background features can be “recovered” simply by adding the Laplacian image to the original (provided suitable rescaling)

$$G(r,c) = I(r,c) + k[\nabla^2 I(r,c)]$$

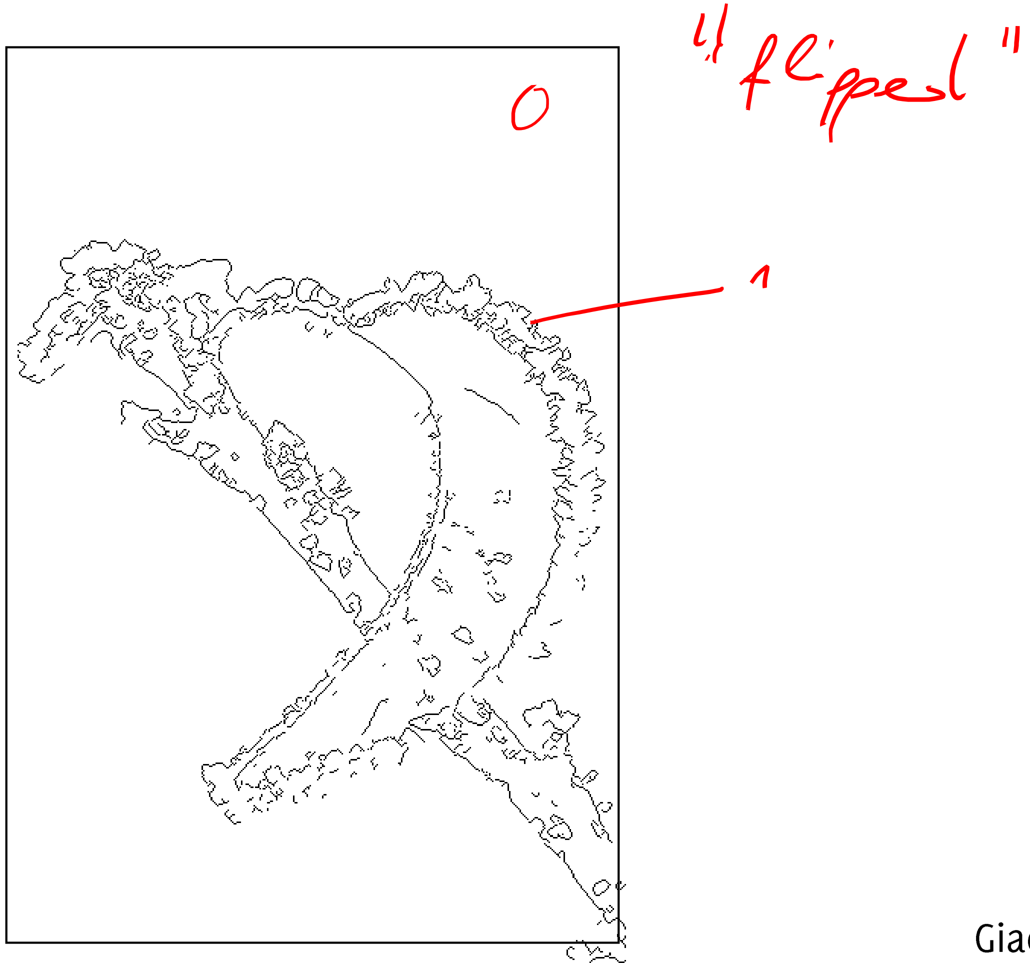


Edges in Images

Edge Detection in Images

Goal: **Automatically** find the contour of objects in a scene.

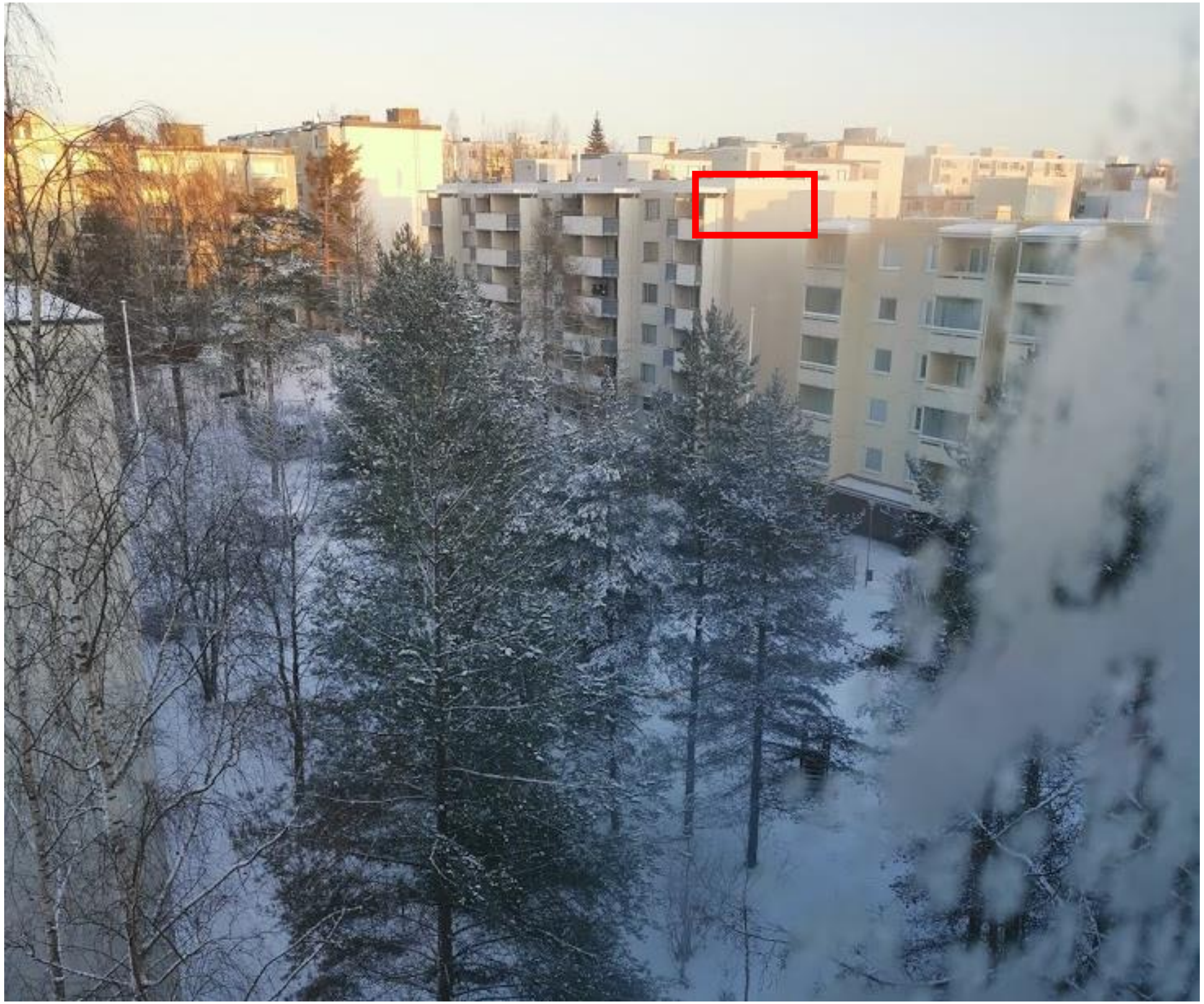
What For: Edges are significant for scene understanding, enhancement compression…



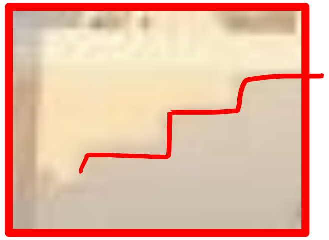
Edges in Images

Depth discontinuities

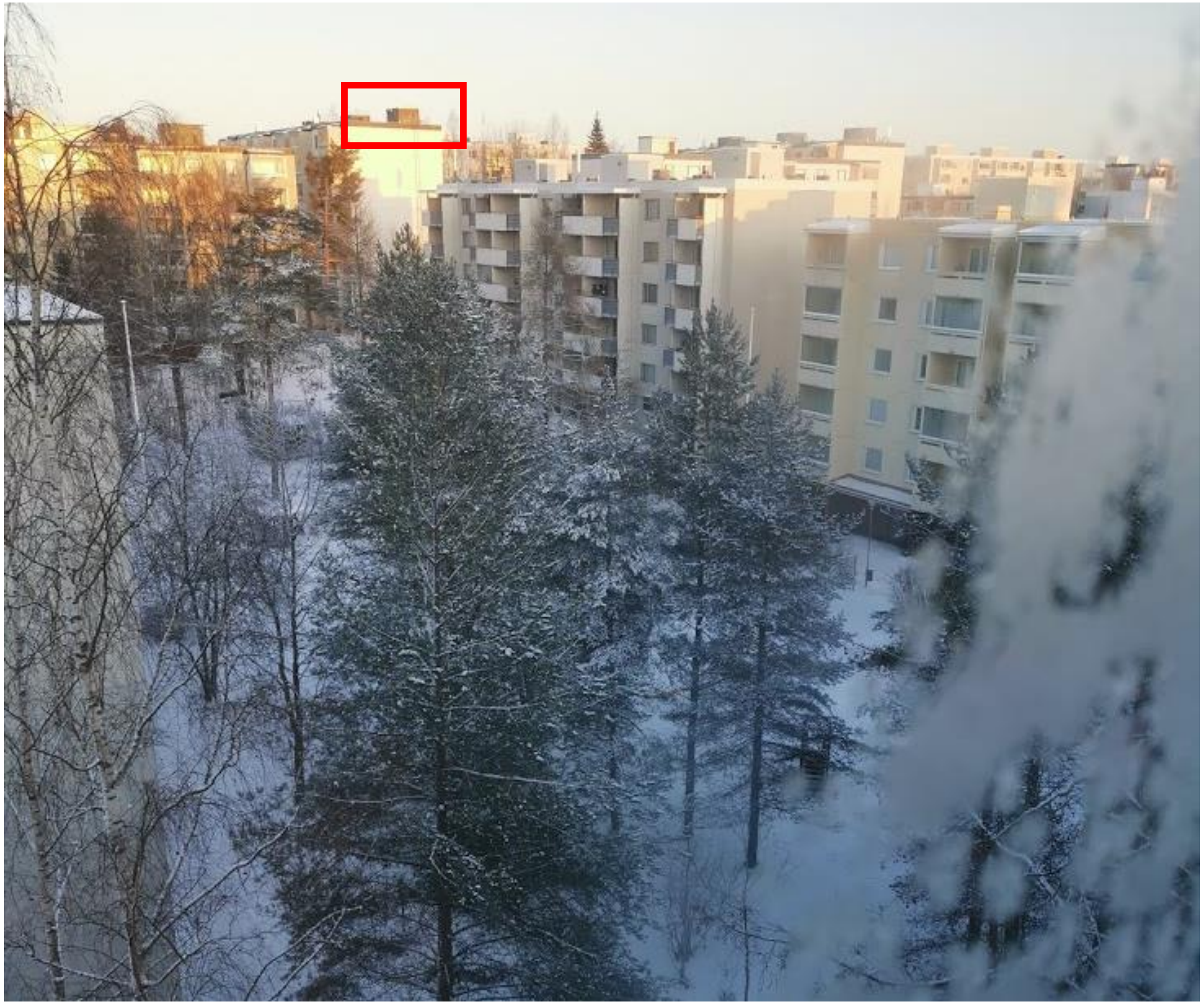
Edges in Images



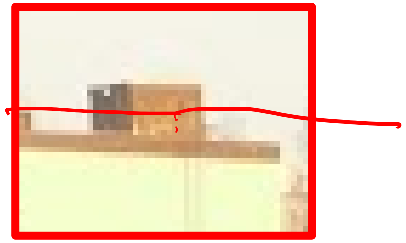
Shadows

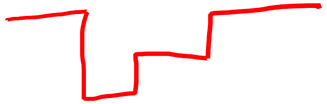


Edges in Images



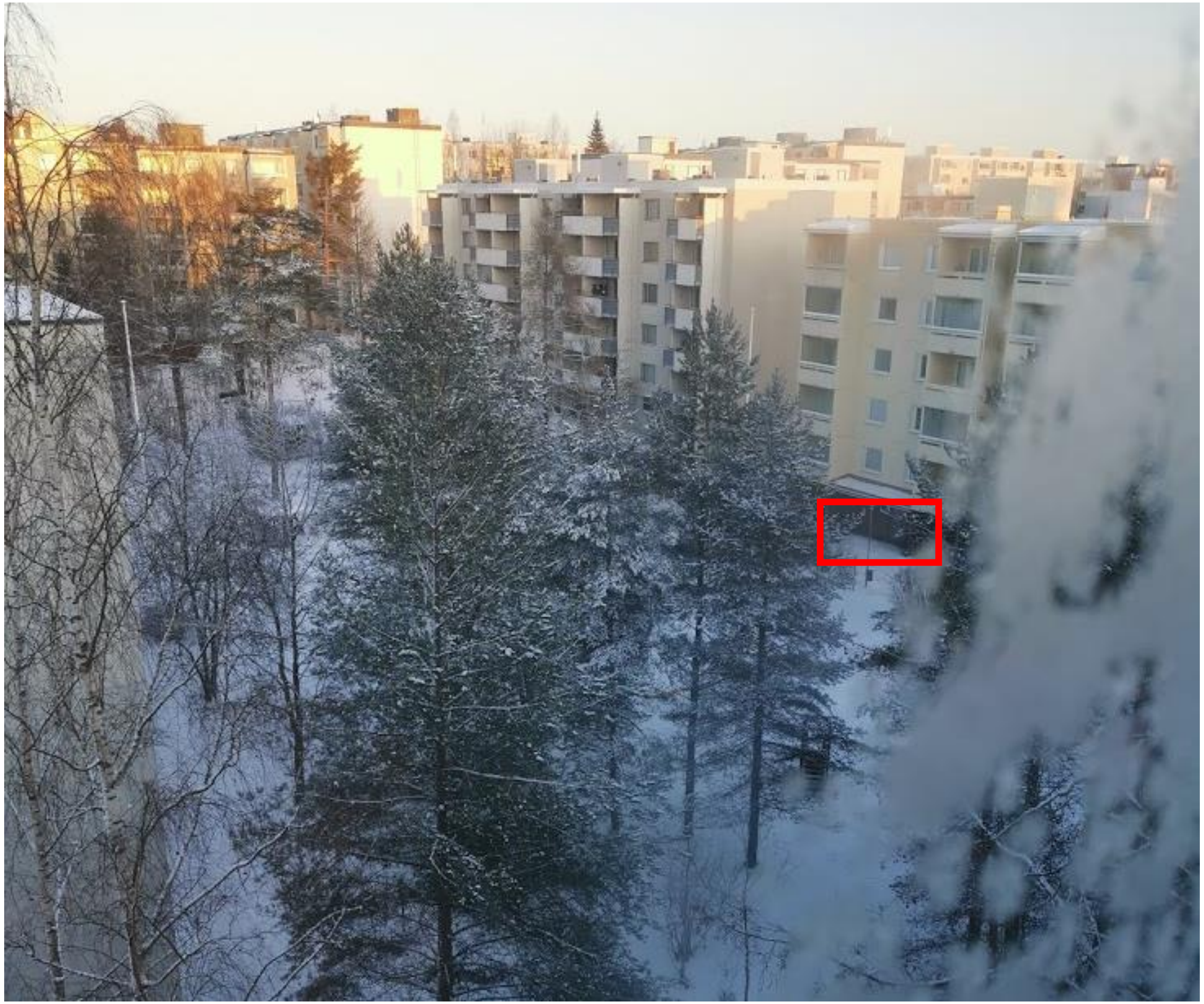
Discontinuities in the surface color, Color changes





Giacomo Boracchi

Edges in Images



Discontinuities in the surface normal

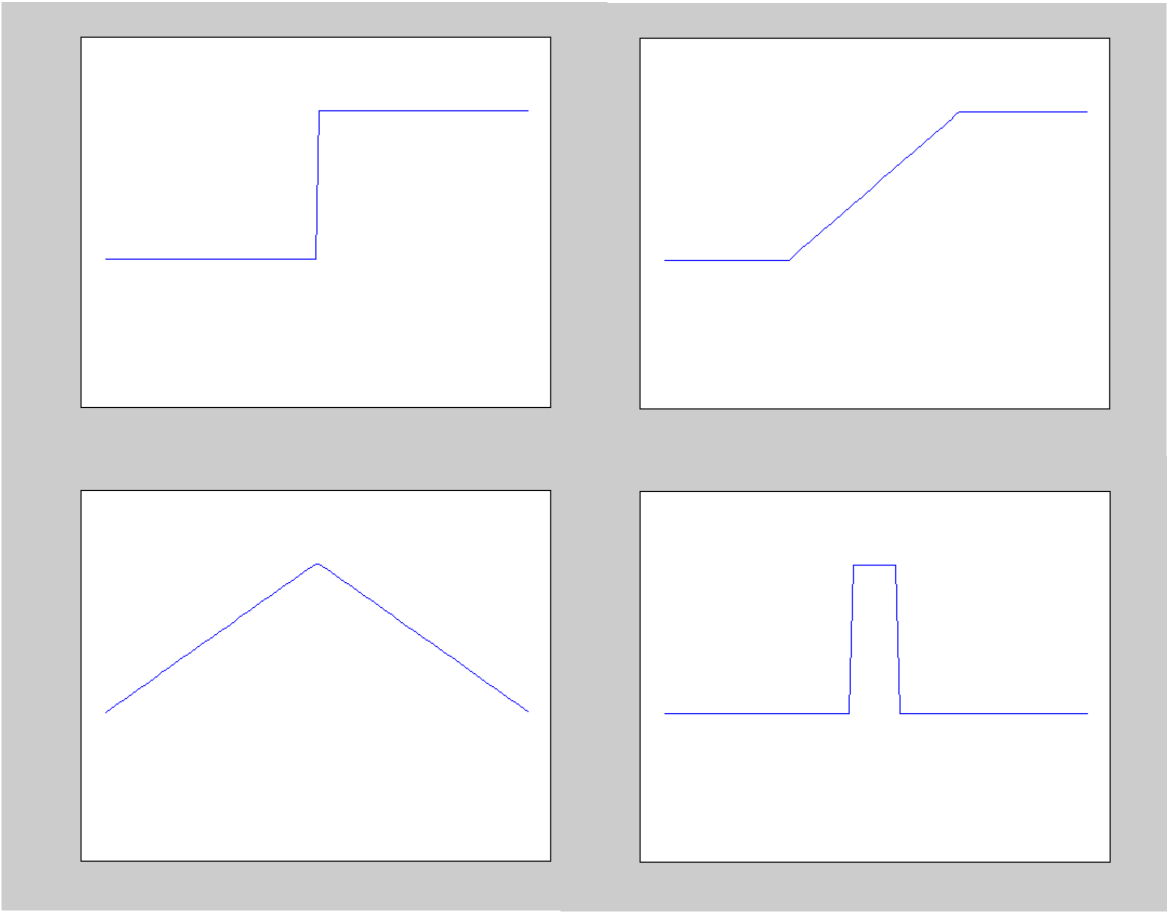
Giacomo Boracchi

What is an Edge

Lets define an edge to be a **discontinuity** in image intensity function.

Several Models

- Step Edge
- Ramp Edge
- Roof Edge
- Spike Edge



They can be thus detected as **discontinuities** of image **Derivatives**

Edge Detection

Gradient Magnitude and edge detectors

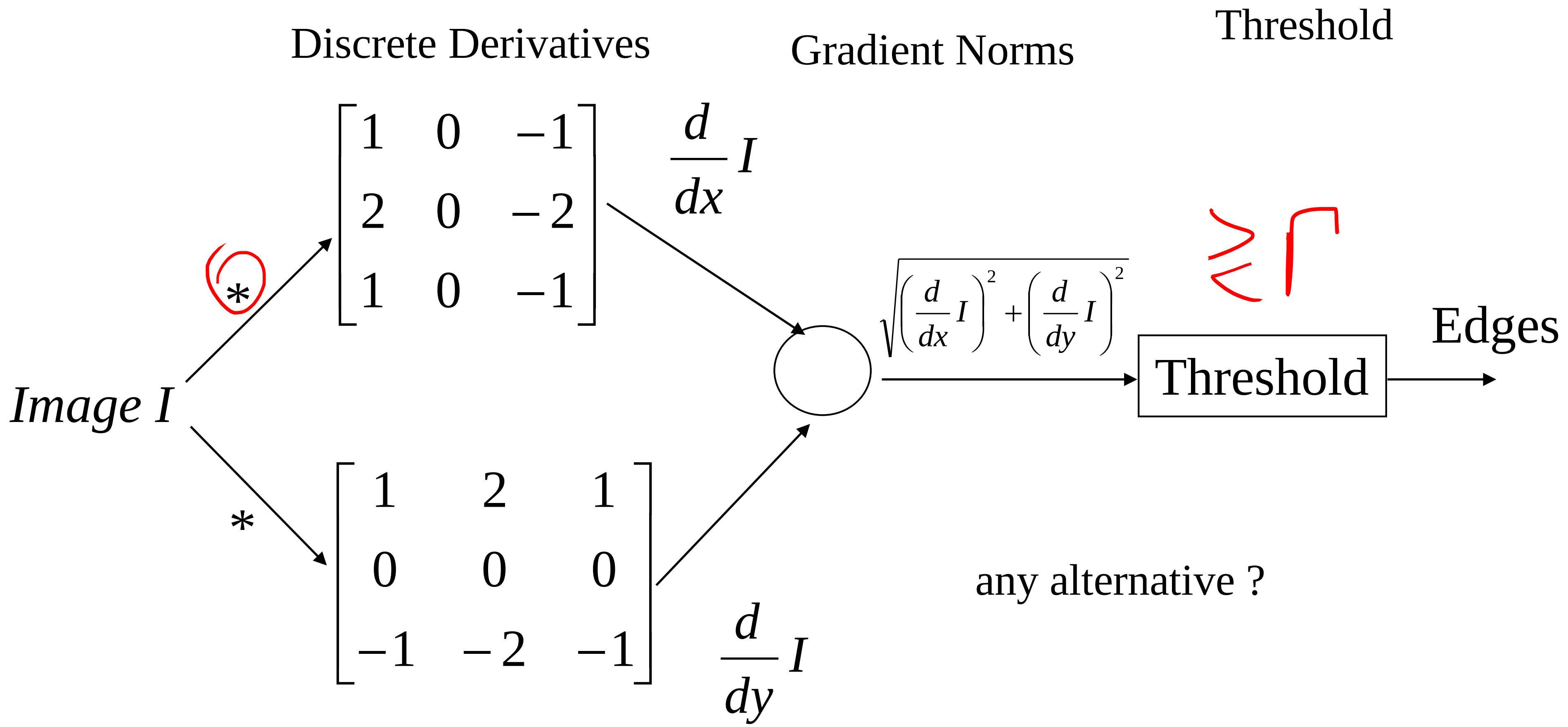
Gradient Magnitute is not a binary image

We can see edges but we cannot identify them, yet

$$\|\nabla I\| = \sqrt{(I \circledast d_x)^2 + \left(I \circledast d_y\right)^2}$$

Detecting Edges in Image

Sobel Edge Detector

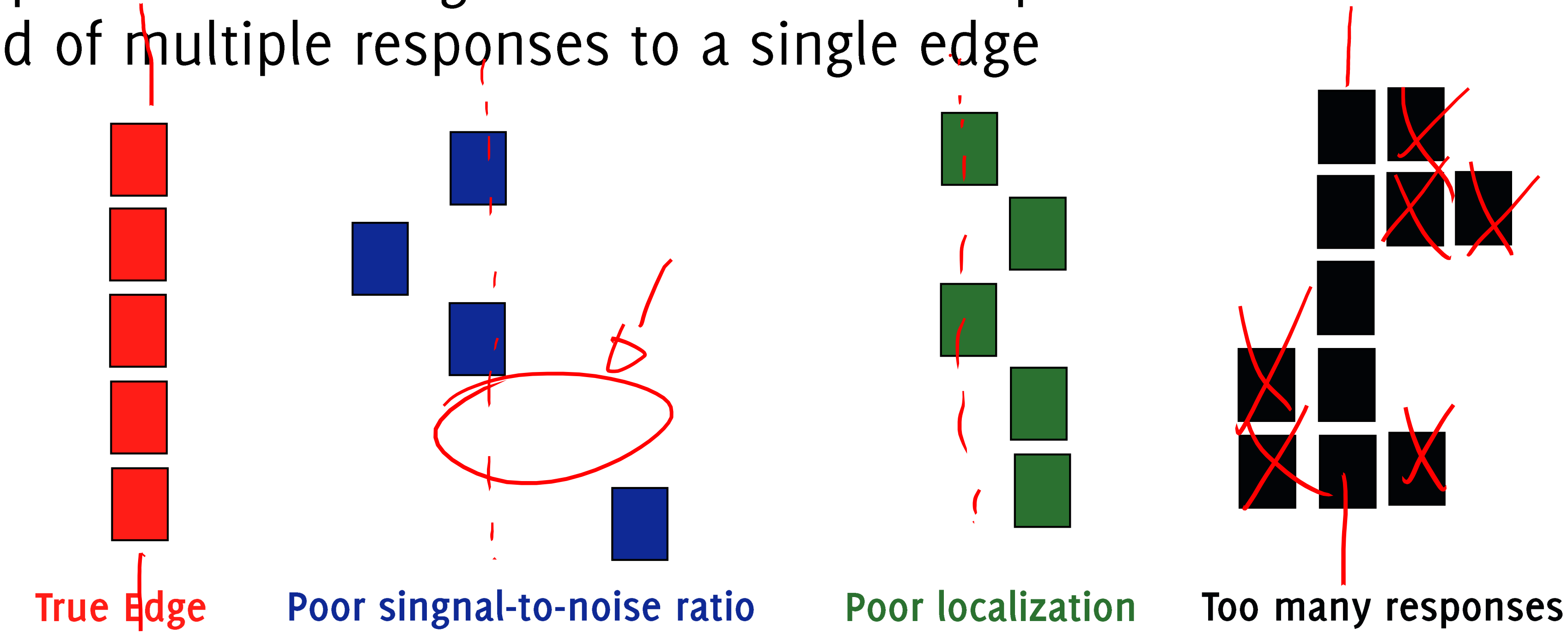


Canny Edge Detector Criteria

Good Detection: The optimal detector must minimize the probability of false positives as well as false negatives.

Good Localization: The edges detected must be as close as possible to the true edges.

Single Response Constraint: The detector must return one point only for each edge point. similar to good detection but requires an ad-hoc formulation to get rid of multiple responses to a single edge



Canny Edge Detector

It is characterized by 3 important steps

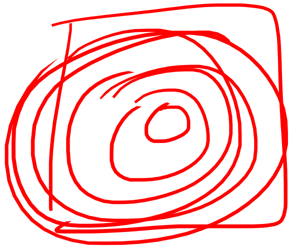
- Convolution with smoothing Gaussian filter before computing image derivatives
- Non-maximum Suppression
- Hysteresis Thresholding

J. Canny "A Computational Approach to Edge Detection" IEEE PAMI vol 8, no. 6, Nov. 1986
http://perso.limsi.fr/Individu/vezien/PAPIERS_ACS/canny1986.pdf

Canny Edge Detector

Smooth by Gaussian

$$S = G_\sigma * I \qquad G_\sigma = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Compute x and y derivatives

$$\Delta S = \left[\frac{\partial}{\partial x} S \quad \frac{\partial}{\partial y} S \right]^T = \left[S_x \quad S_y \right]^T$$

$I \mapsto S \rightarrow |\nabla S|, \angle \nabla S$

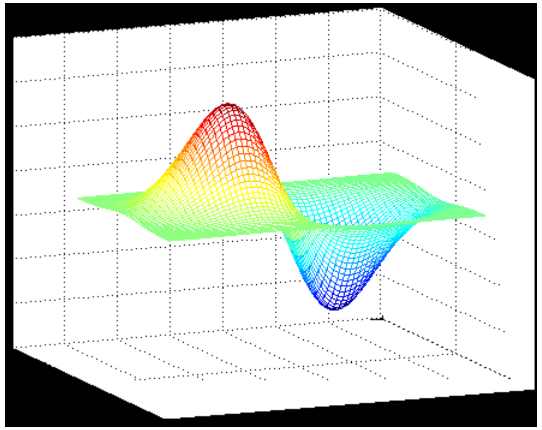
Compute gradient magnitude and orientation

$$|\Delta S| = \sqrt{S_x^2 + S_y^2} \qquad \theta = \tan^{-1} \frac{S_y}{S_x}$$

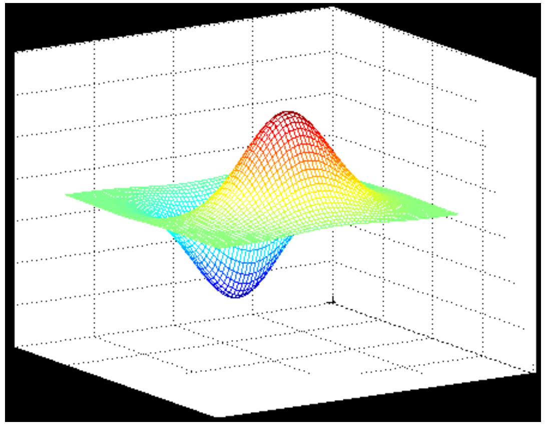
Canny Edge Operator

$$\Delta S = \Delta\left(G_\sigma * I\right) = \Delta G_\sigma * I$$

$$\Delta G_\sigma = \begin{bmatrix} \dfrac{\partial G_\sigma}{\partial x} & \dfrac{\partial G_\sigma}{\partial y} \end{bmatrix}^T$$

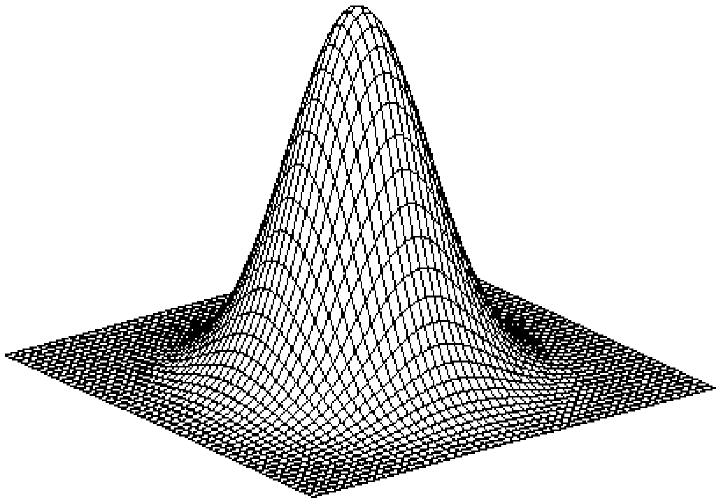


$$\Delta S = \begin{bmatrix} \dfrac{\partial G_\sigma}{\partial x} * I & \dfrac{\partial G_\sigma}{\partial y} * I \end{bmatrix}^T$$

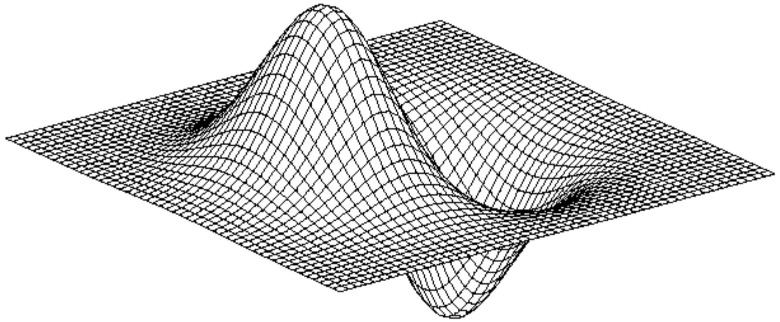


Convolution is associative

$I * (g * d_x)$



$* \quad [1 \quad 0 \quad -1] =$



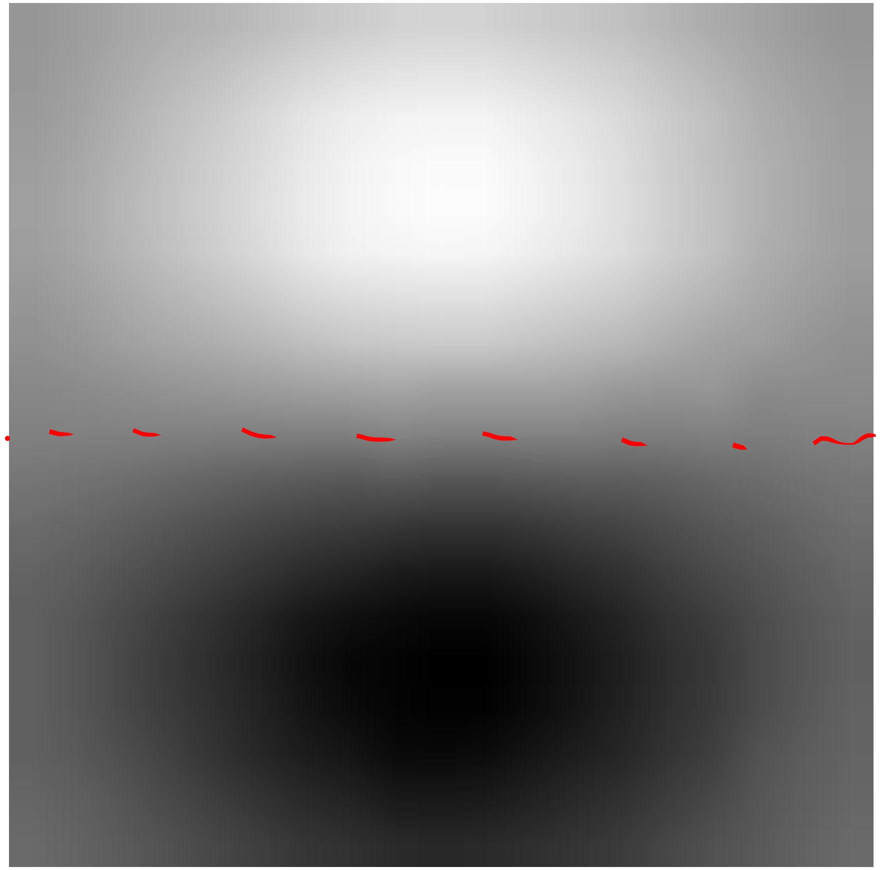
2D-gaussian

x - derivative

Gaussian Derivative Filters

x-direction

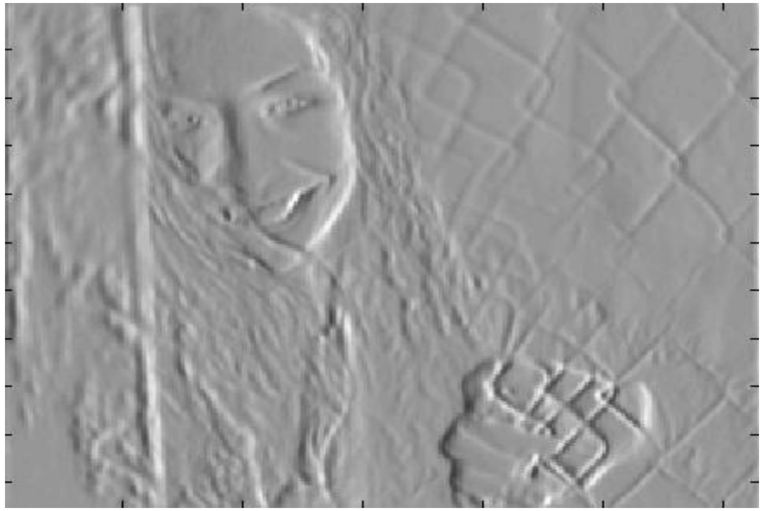
y-direction



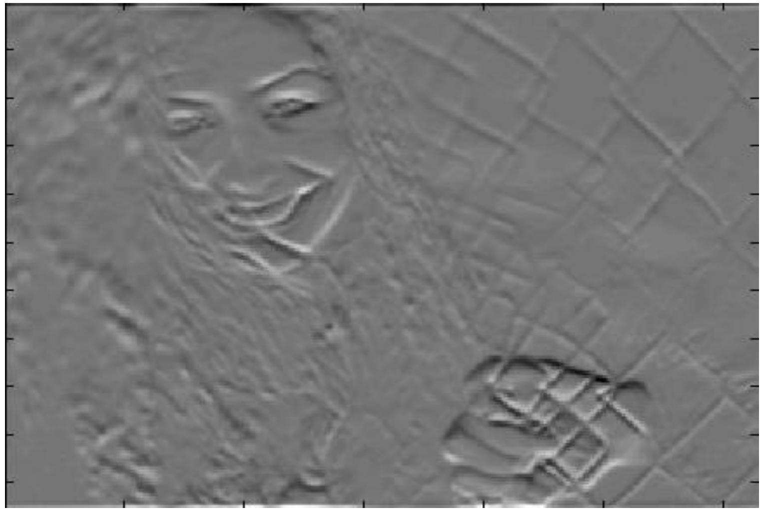
Canny Edge Detector

I

S_x



S_y



Canny Edge Detector

$$|\Delta S| = \sqrt{S_x^2 + S_y^2}$$

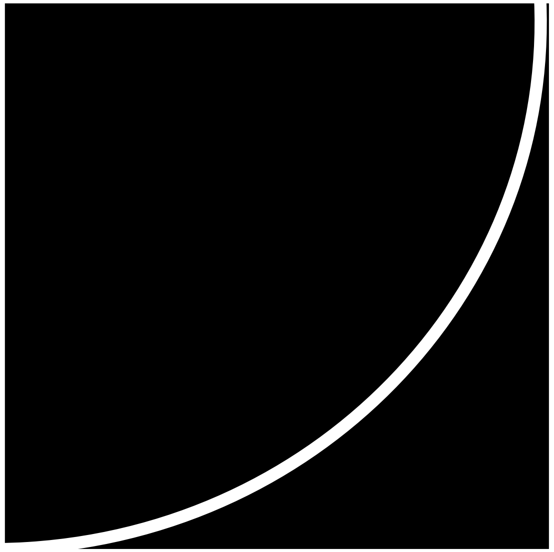
I

$$|\Delta S| \geq Threshold = 25$$

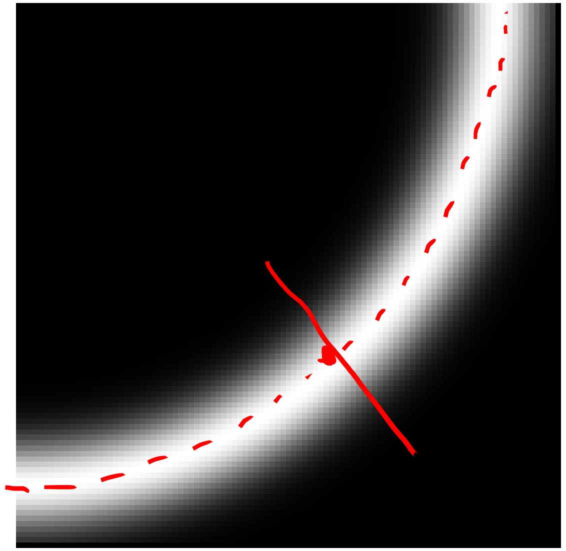
Non-Maximum Suppression: The Idea

We wish to determine the points along the curve where the gradient magnitude is largest.

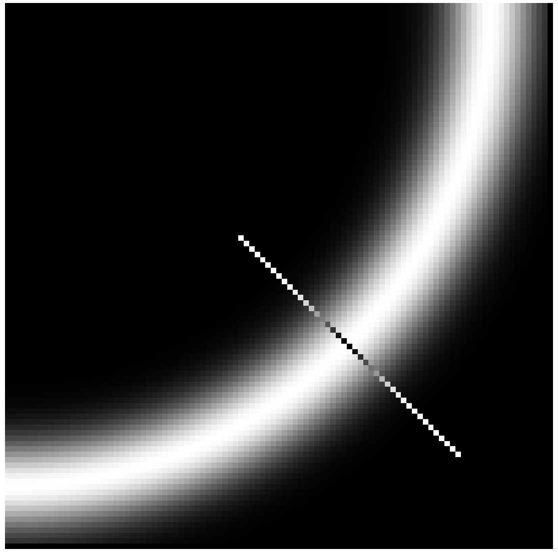
Non-maximum suppression: we look for a maximum along a slice orthogonal to the curve. These points form a 1D signal.



Original Image

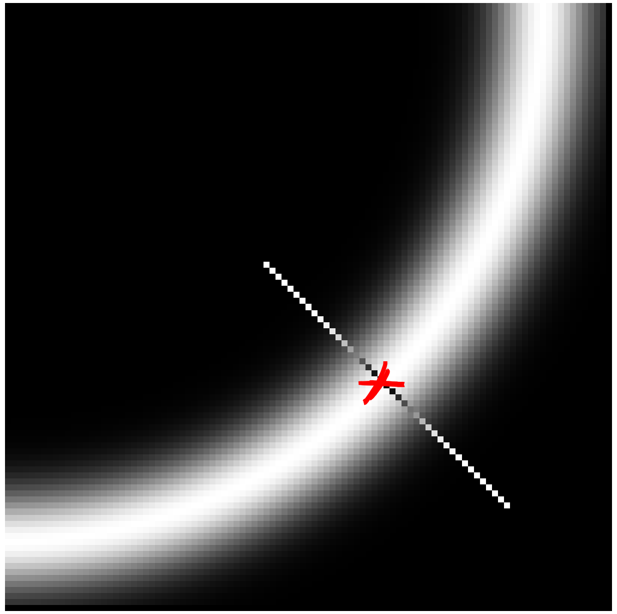


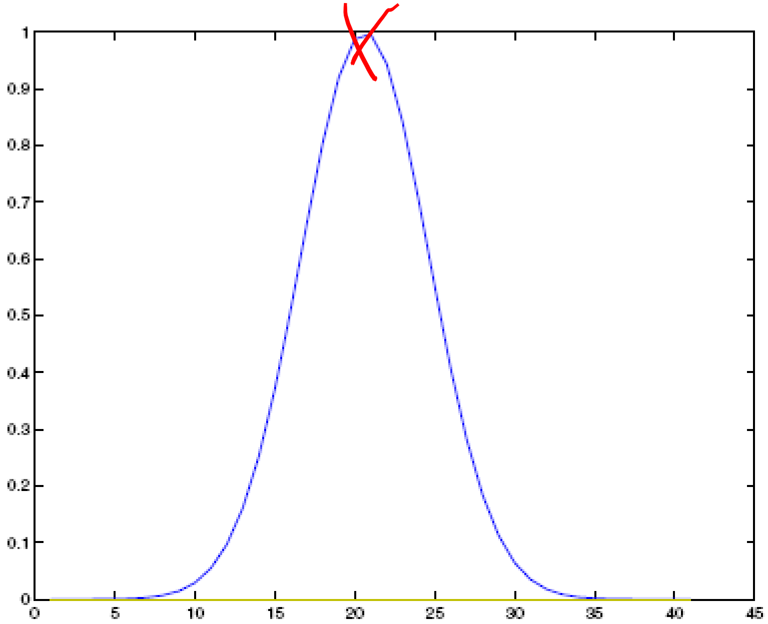
Gradient Magnitude



Segment orthogonal

Non-Maximum Suppression

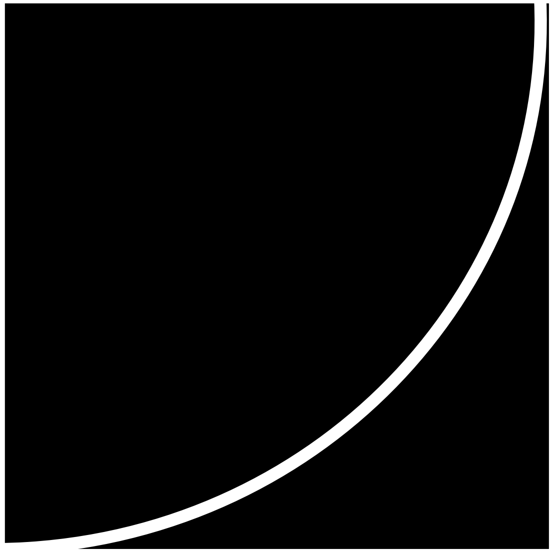




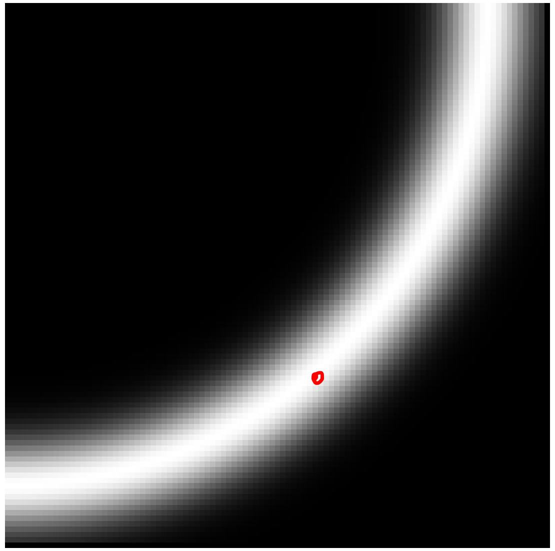
Non-Maximum Suppression: The Idea

There are two issues:

- **which slice to select to extract the maximum?**
- once an edge pixel has been found, **which pixel to test next?**

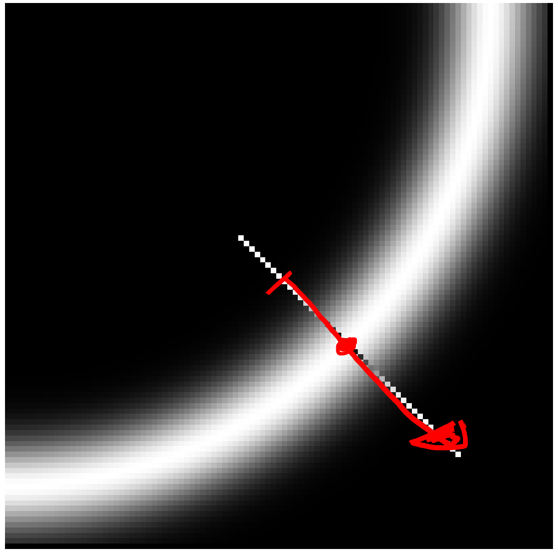


Original Image



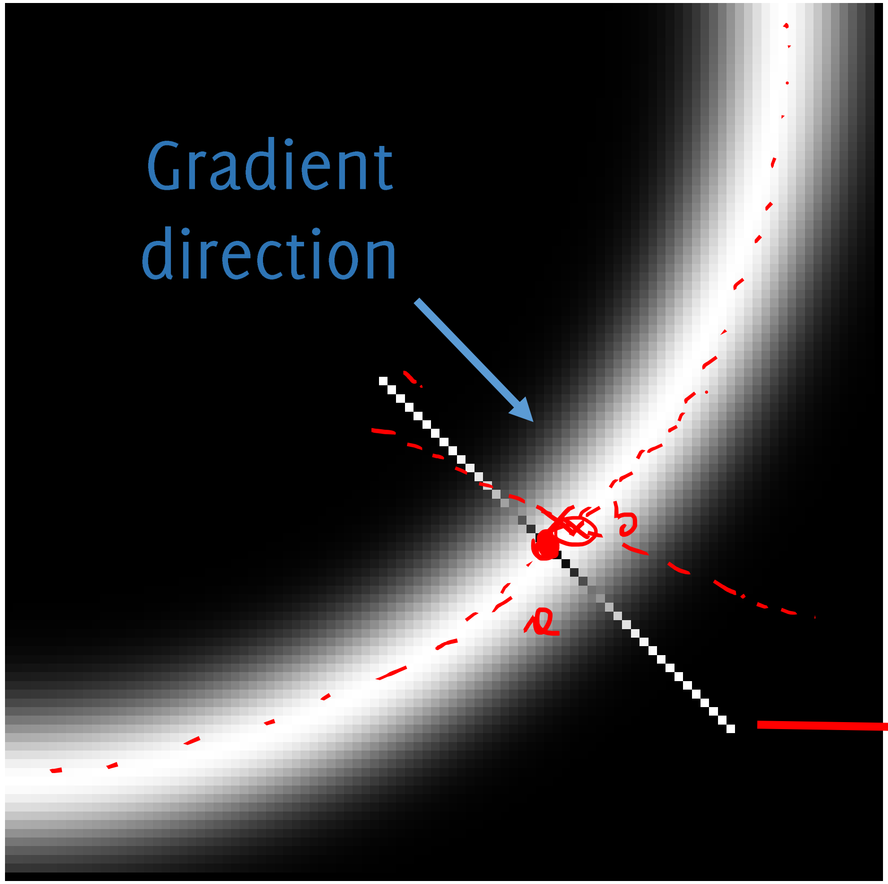
Gradient Magnitude

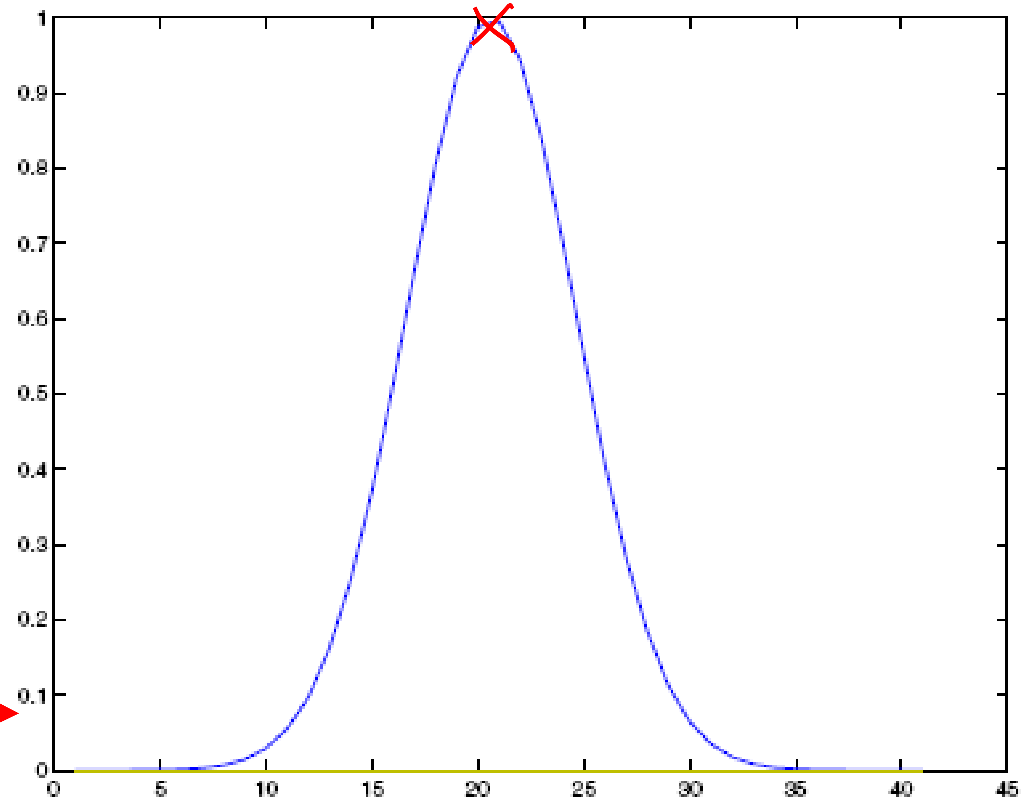
(Thresholded)



Segment orthogonal

Non-Maximum Suppression – Idea (II)





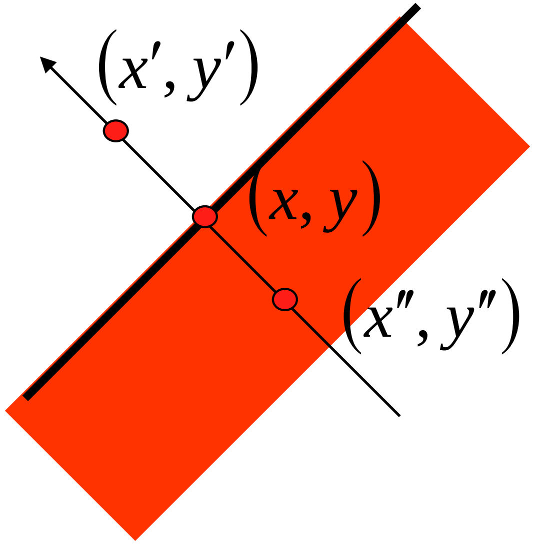
In each pixel, **the gradient indicates the direction of the steepest variation**: thus, the gradient is orthogonal to the edge direction (no variation along the edge). We have to consider pixels on a segment following the gradient direction

The intensity profile along the segment. We can easily identify the location of the maximum.

Non-Maximum Suppression - Threshold

Suppress the pixels in 'Gradient Magnitude Image' which are not local maximum

$$M(x,y)=\begin{cases} |\Delta S|(x,y) & \text{if } |\Delta S|(x,y)>|\Delta S|(x',y') \\ & \& \, |\Delta S|(x,y)>|\Delta S|(x'',y'') \\ 0 & \text{otherwise} \end{cases}$$



(x',y') and (x'',y'') are the neighbors of (x,y) in $|\Delta S|$

These have to be taken on a line along the gradient direction in (x,y)

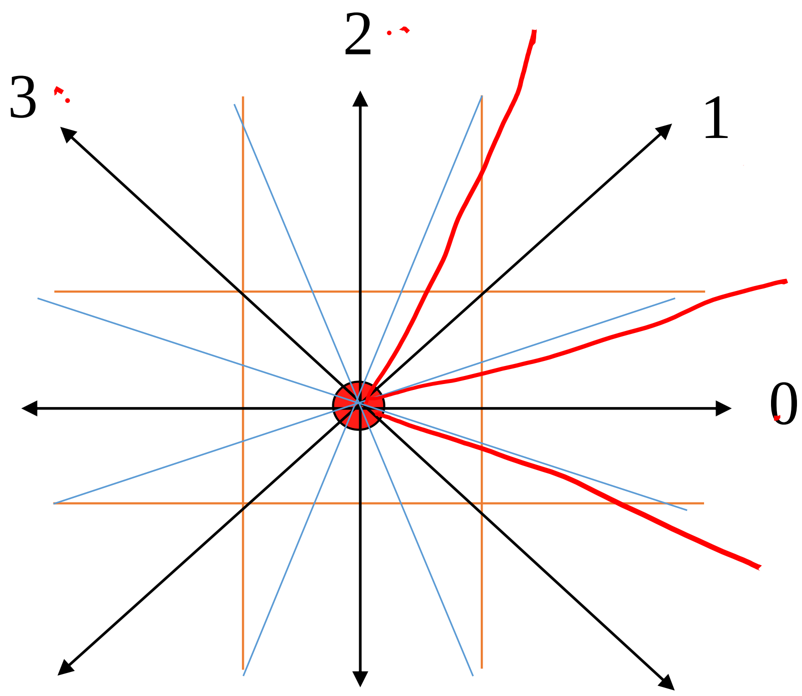
Non-Maximum Suppression: Quantize Gradient Directions

In practice the gradient directions are quantized according to 4 main directions, each covering 45° (orientation is not considered)

- Thus, only diagonal, horizontal, vertical line segments are considered

We consider 4 quantized directions 0,1,2, 3

$$\theta(\boldsymbol{x}_0) = \operatorname{atan}\left(\frac{\partial/_{\partial y}\, I(\boldsymbol{x}_0)}{\partial/_{\partial x}\, I(\boldsymbol{x}_0)}\right)$$



Orientation is irrelevant since this is meant for segment extraction

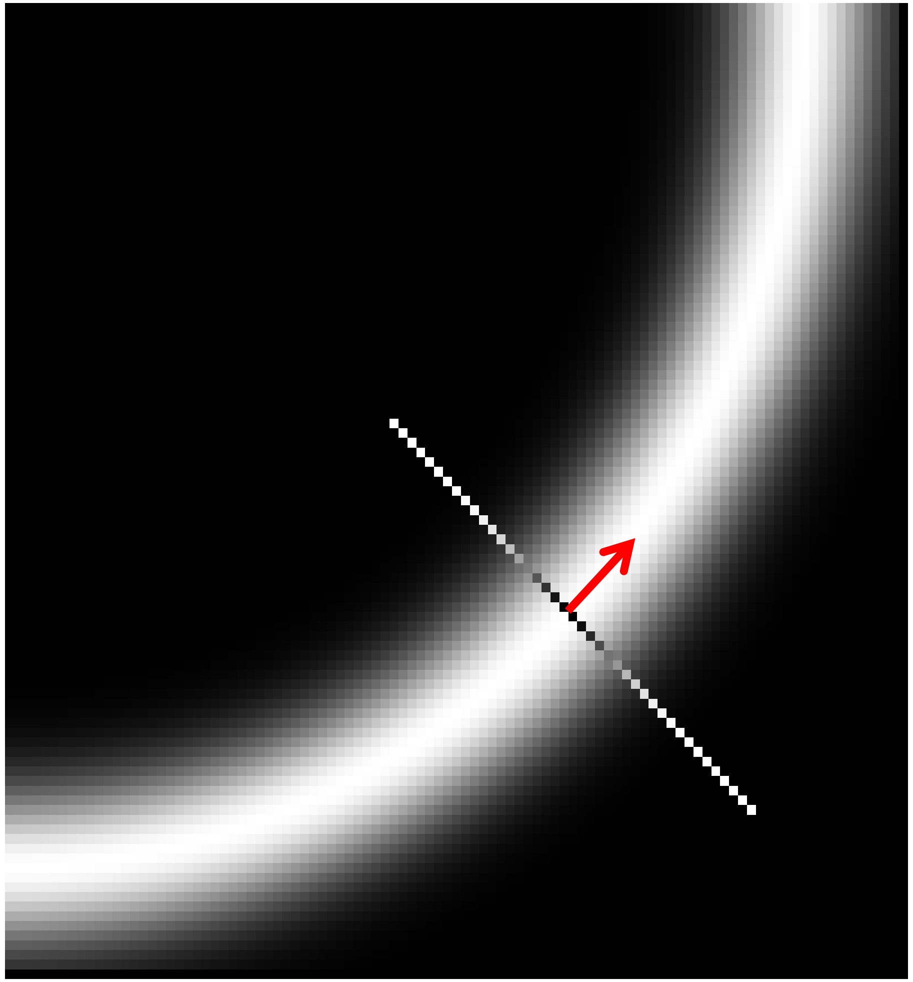
Tracking the edge direction

The direction orthogonal to the gradient follows the edge

Once a local maxima is found, **we consider the direction orthogonal to the gradient in that pixel,**

The direction is quantized as for extracting the 1D segment for nonmaximum suppression

We move one step in the quantized direction to determine another point where to extract 1D segments



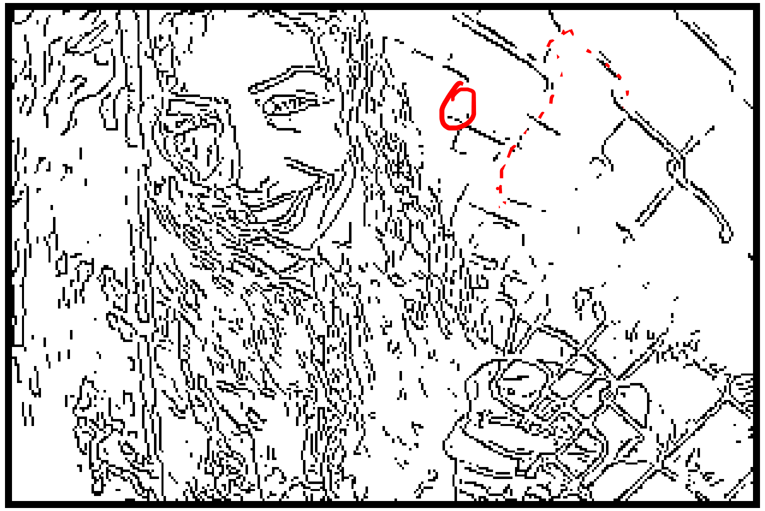
Non-Maximum Suppression

$$|\Delta S| = \sqrt{S_x^2 + S_y^2}$$

M

Non max suppr.

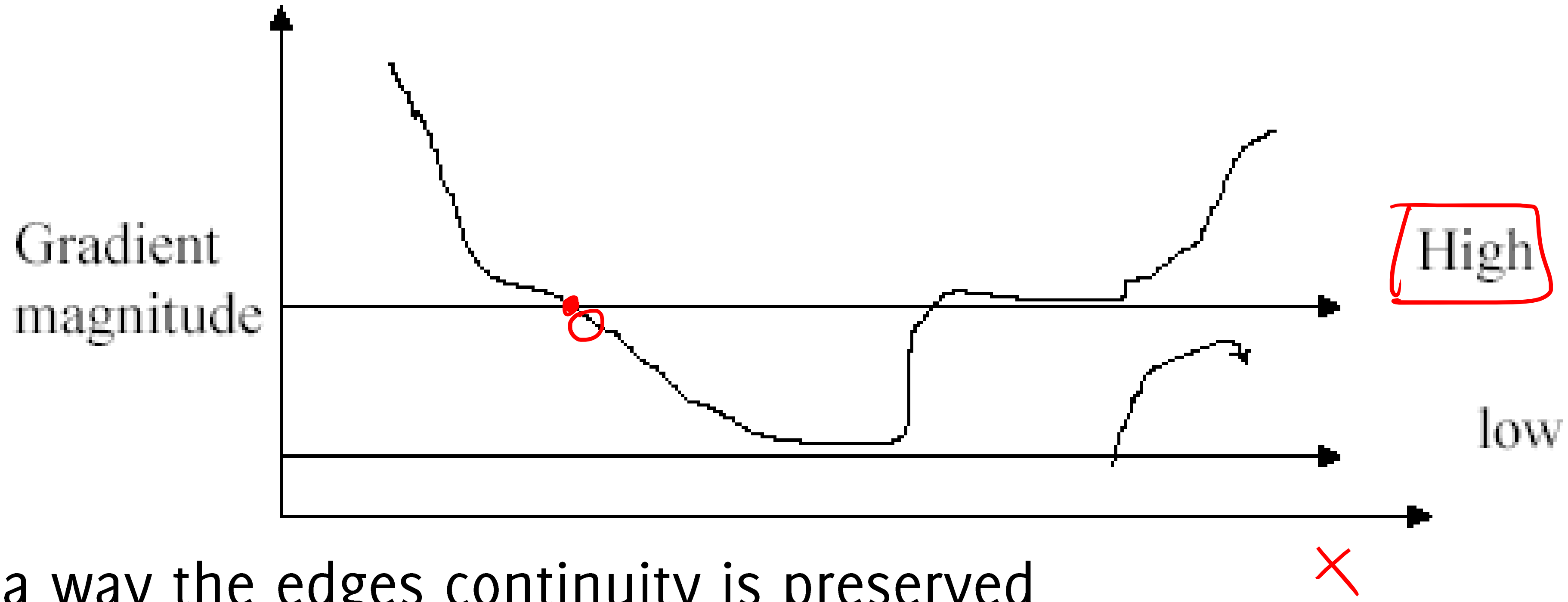
$$M \geq Threshold = 25$$



Hysteresis Thresholding

Use of two different threshold High and Low for

- For new edge starting point
- For continuing edges



In such a way the edges continuity is preserved

Hysteresis Thresholding

If the gradient at a pixel is **above 'High' threshold,**

- declare it an **'edge pixel'**.

If the gradient at a pixel is **below 'Low' threshold**

- declare it a **'non-edge-pixel'**.

If the gradient at a pixel is **between 'Low' and 'High' thresholds**

- then declare it an **'edge pixel'** if and only if can be directly **connected** to an 'edge pixel' or connected via pixels between 'Low' and ' High'.

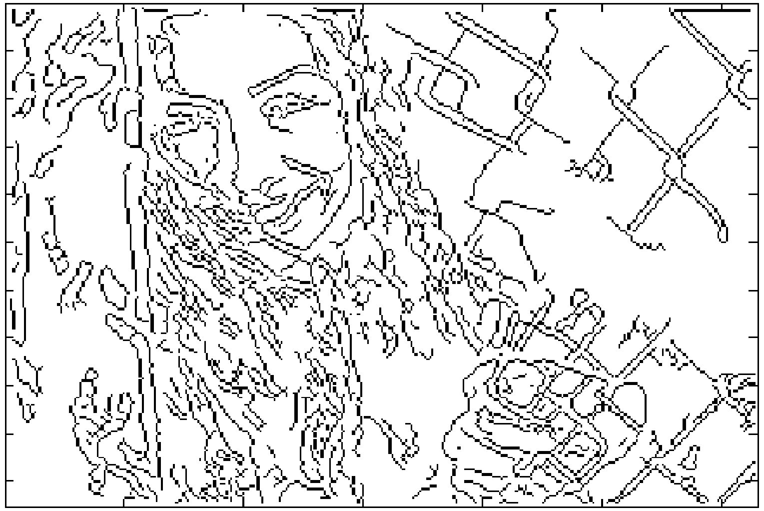
Hysteresis Thresholding

M

$M \geq Threshold = 25$

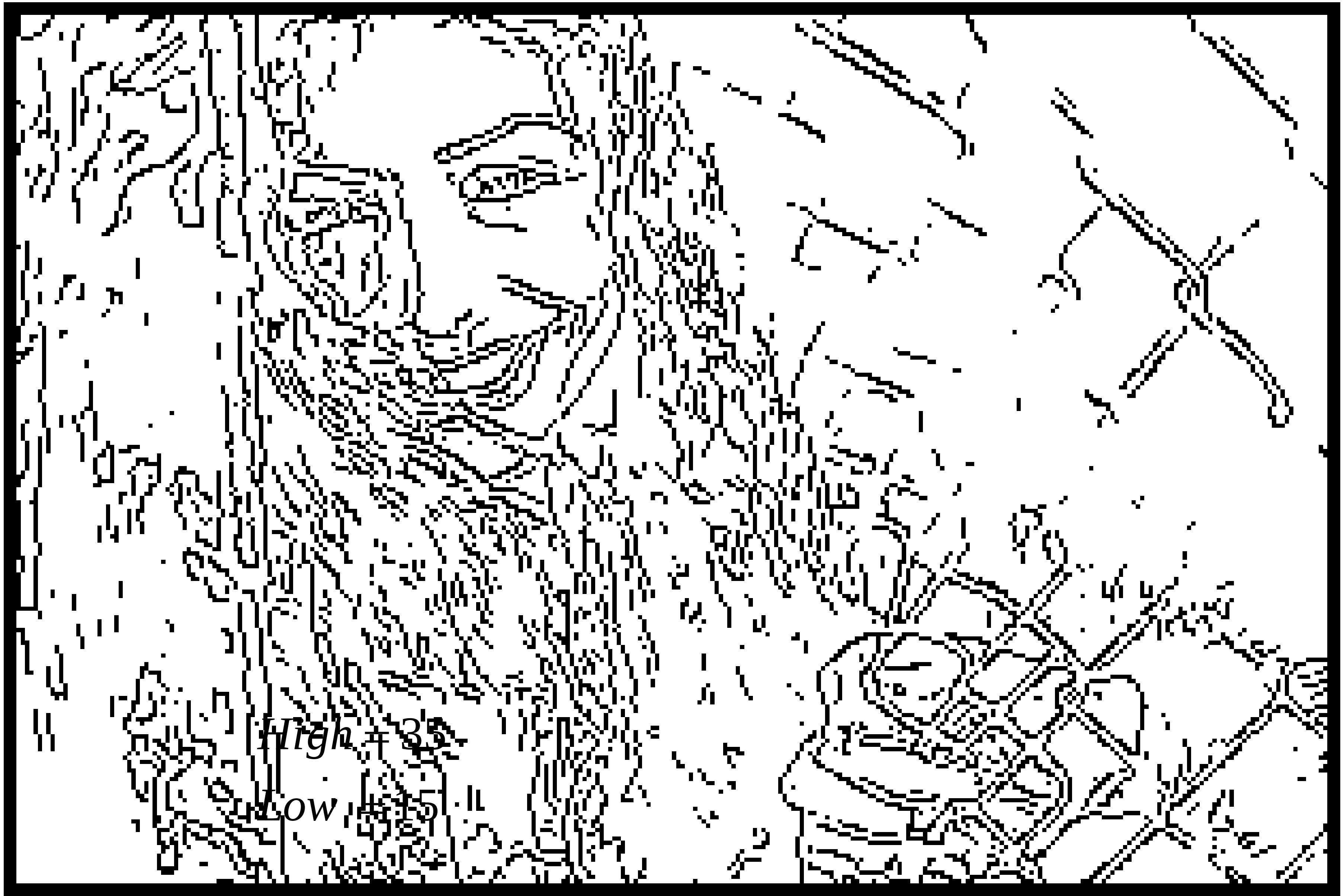
$High = 35$

$Low = 15$



Hysteresis Thresholding

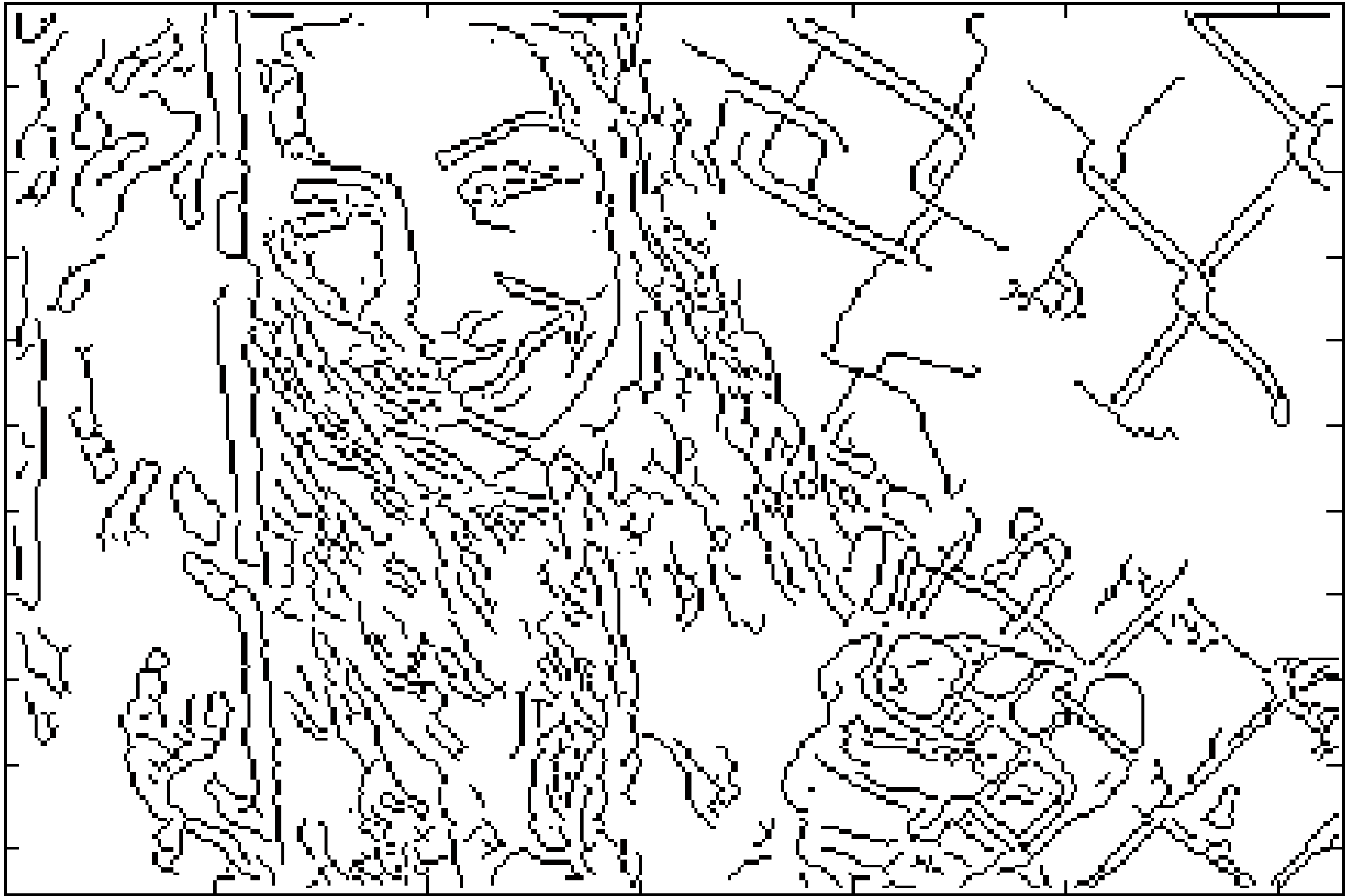
$M \geq Threshold = 25$



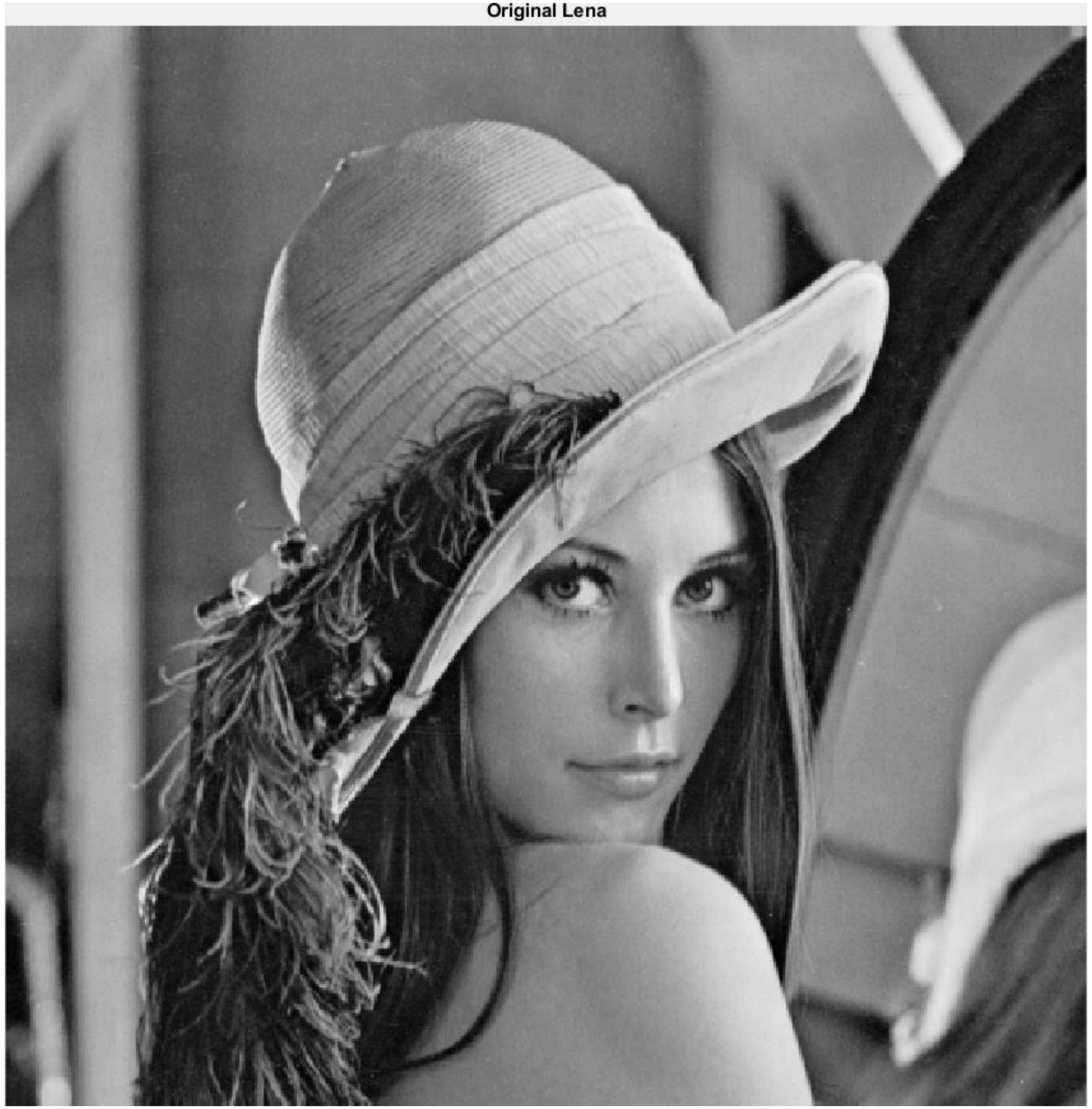
Hysteresis Thresholding

$High = 35$

$Low \ = 15$

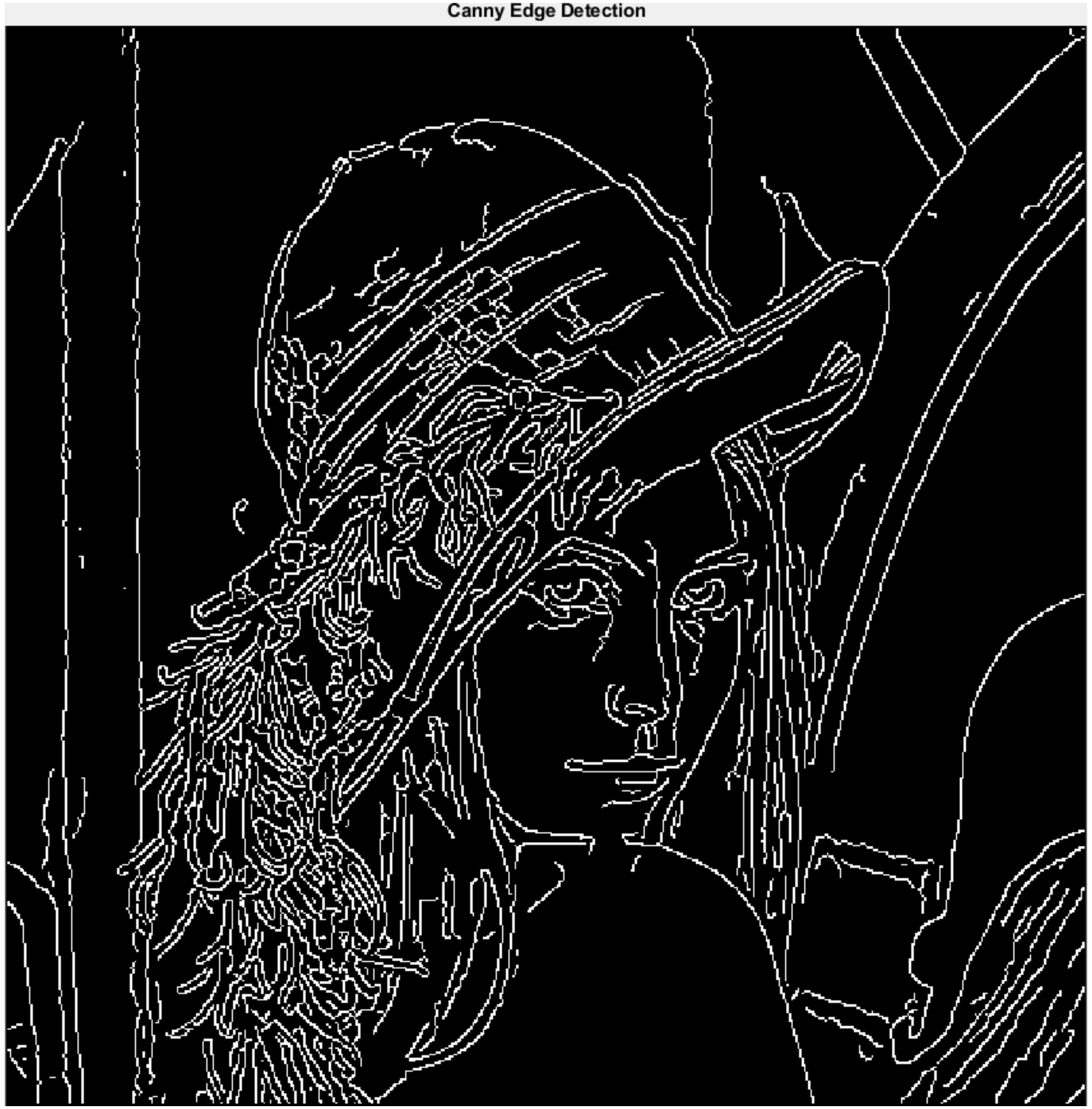


Canny Edge Detection



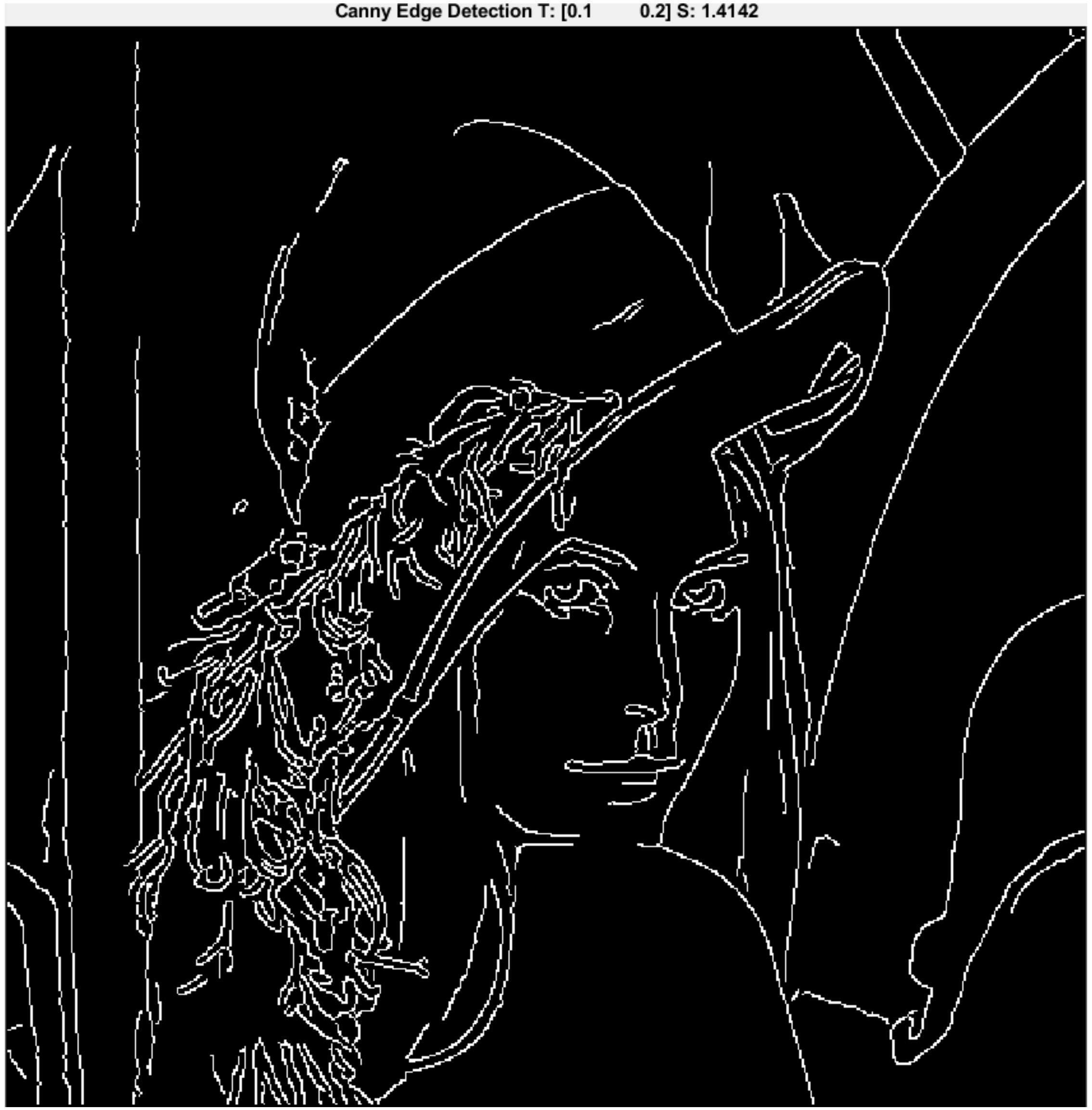
Canny Edge Detection

$\sigma = \sqrt{2}$

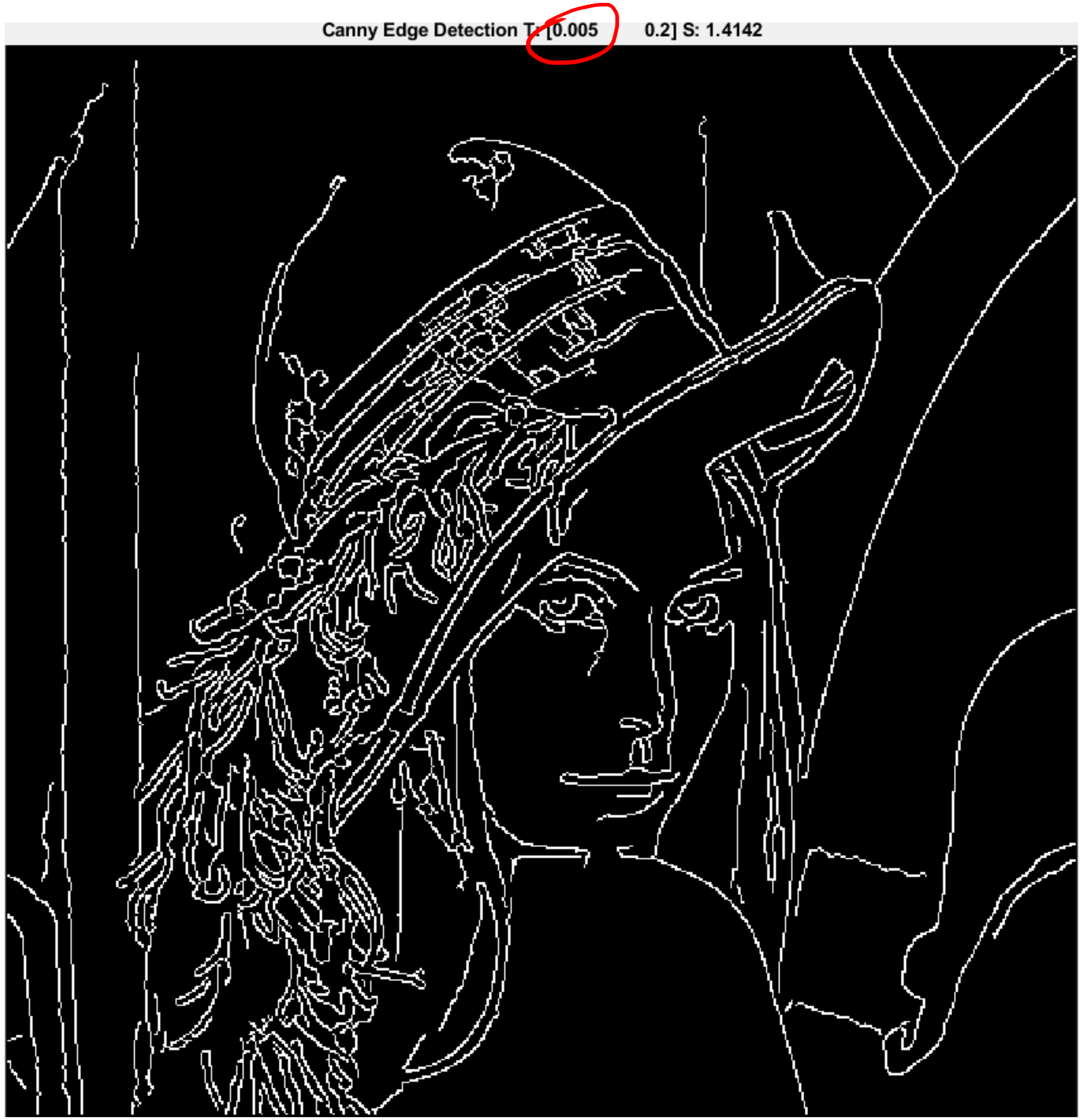


Canny Edge Detection – changing hysteresis thresholds

L U σ Gaussian smoothing

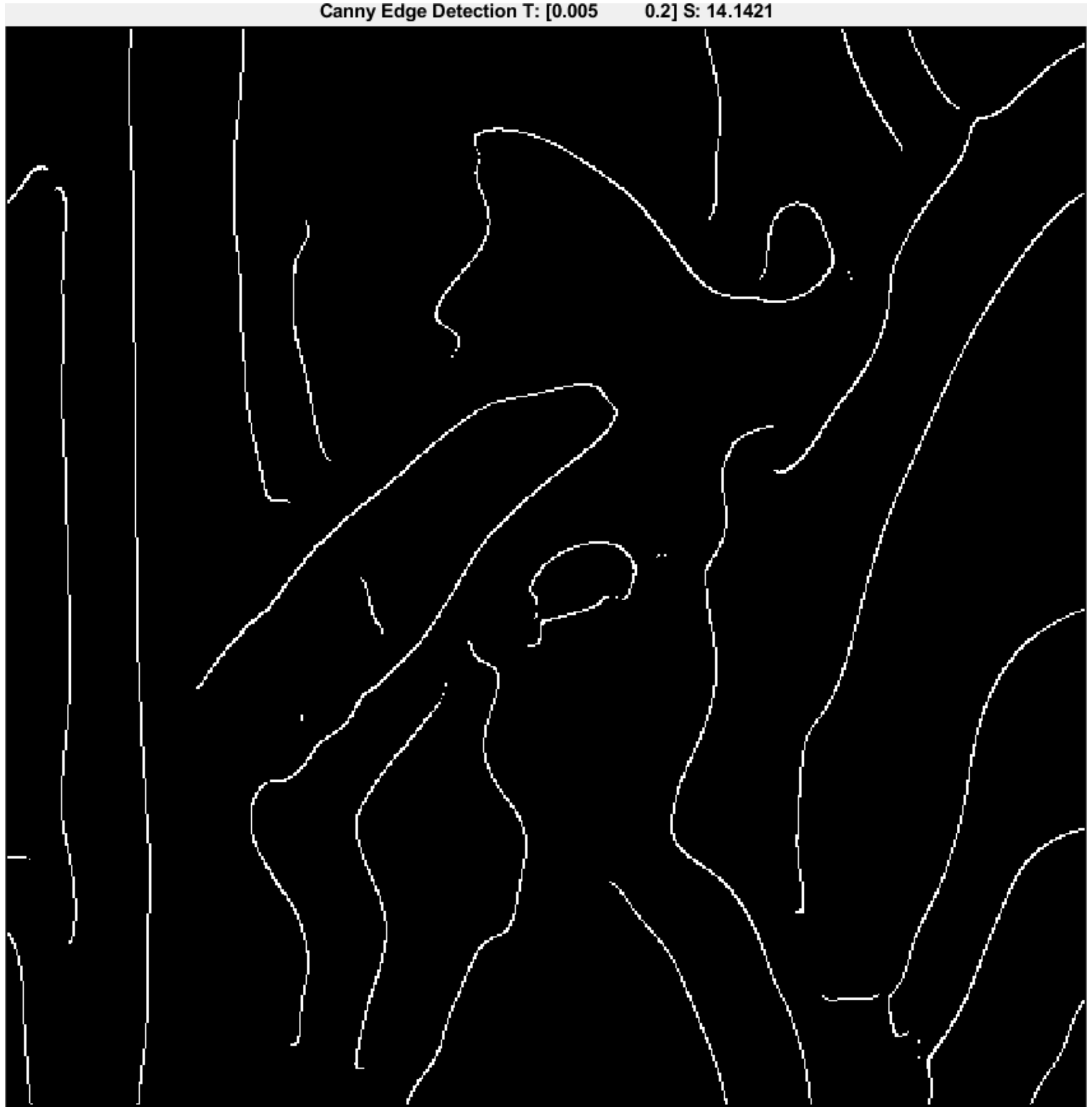


Canny Edge Detection – changing hysteresis thresholds



Canny Edge Detection – changing the smoothing

10 σ

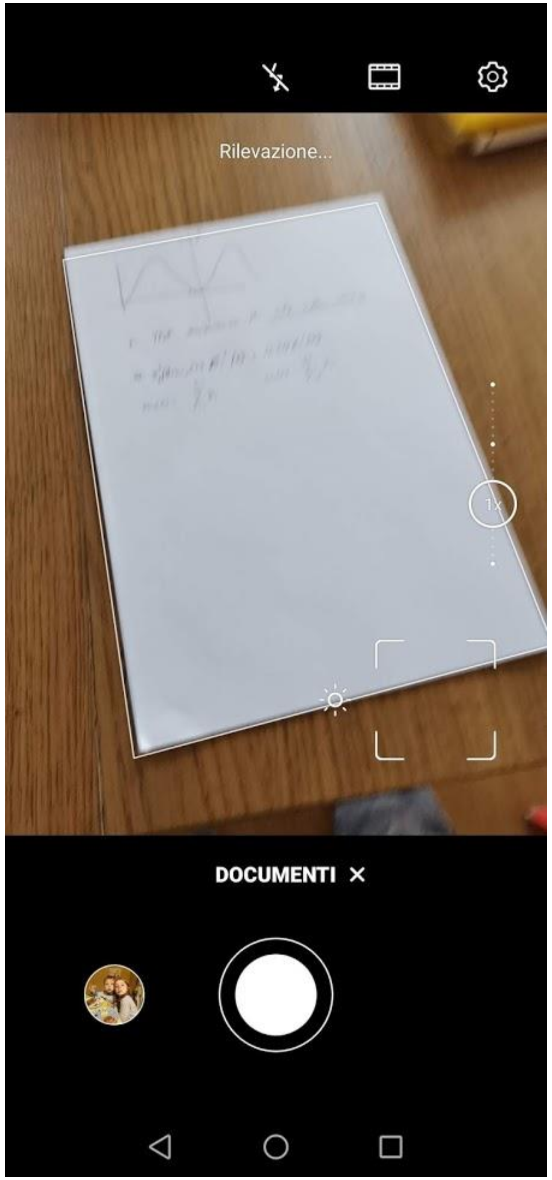


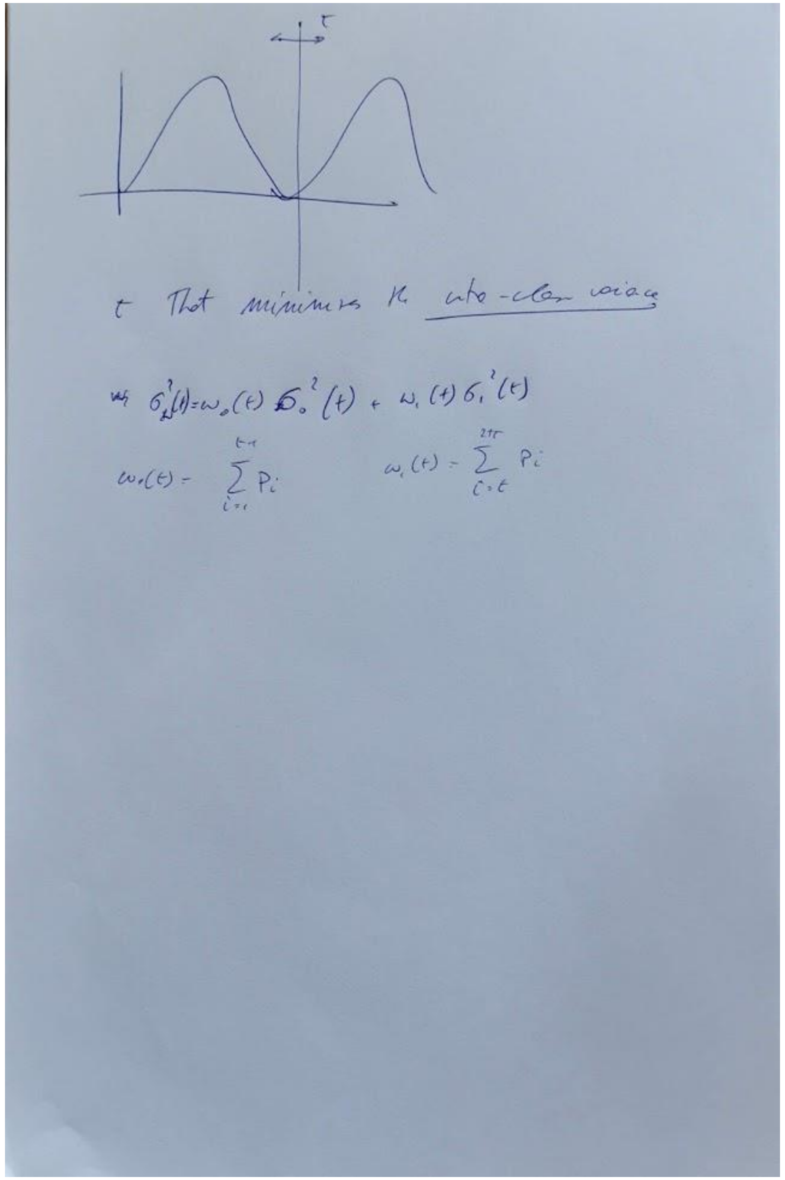
Line Detection: Hough Transform

Extracting Line Equations From Edges

Giacomo Boracchi

Line Detection is Important

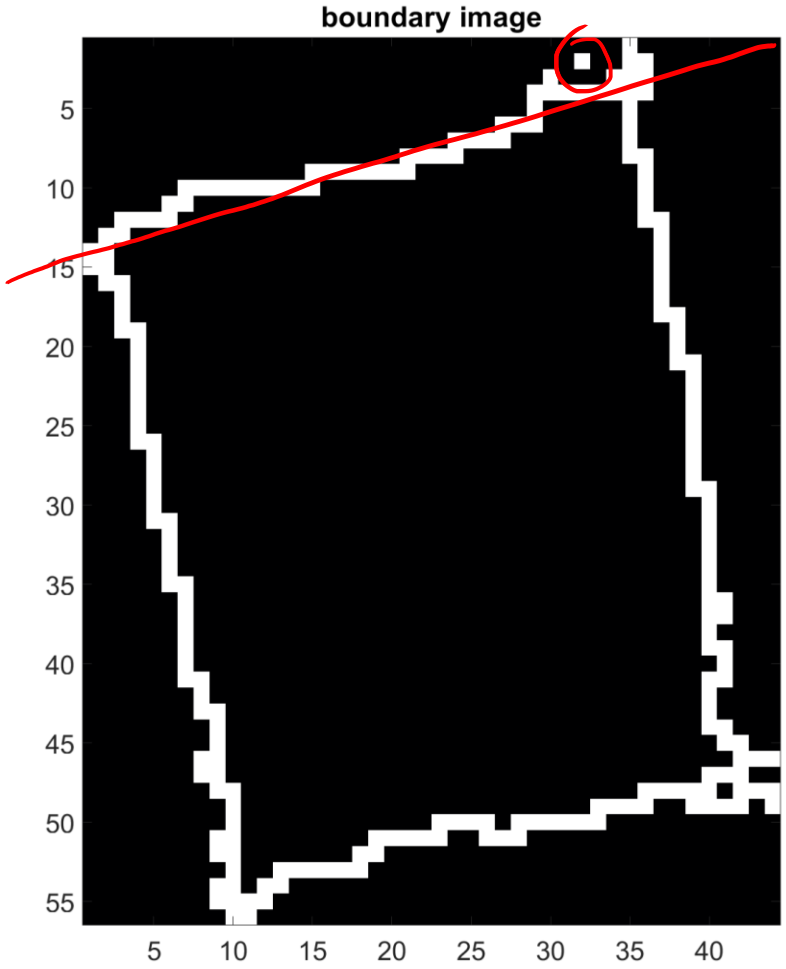




Giacomo Boracchi

Line Detection: The problem

Finding all the lines passing through points in (a binary) image

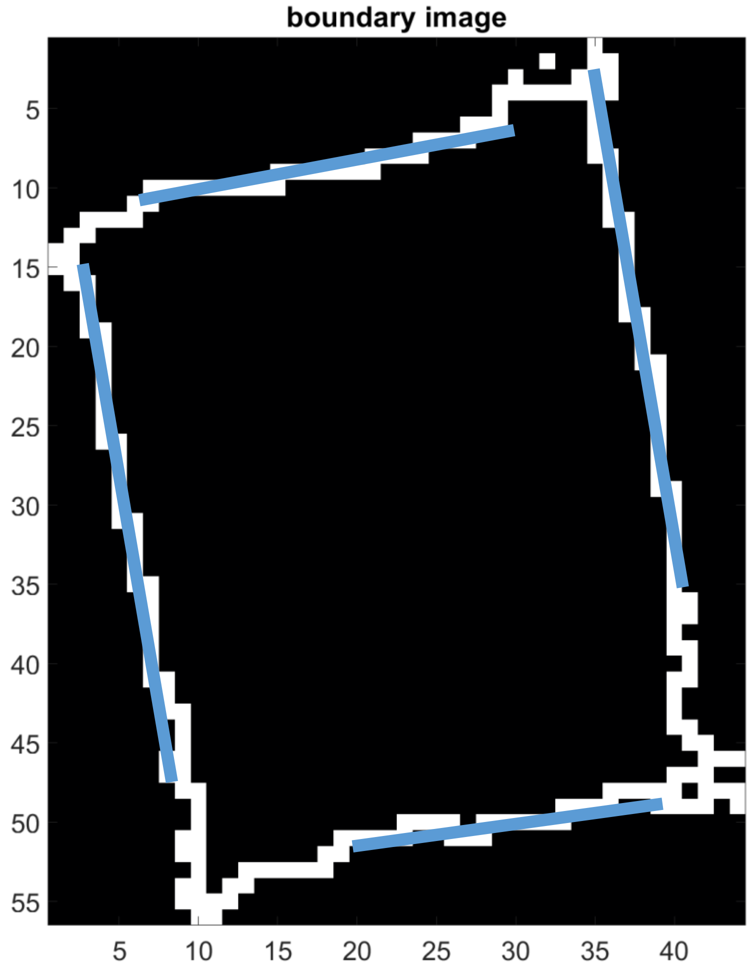


Line Detection: The problem

Finding all the lines passing through points in (a binary) image

Finding lines means

- Having an analytical expression for each line
- Estimating its direction, length
- Thus, clustering points belonging to the same segment

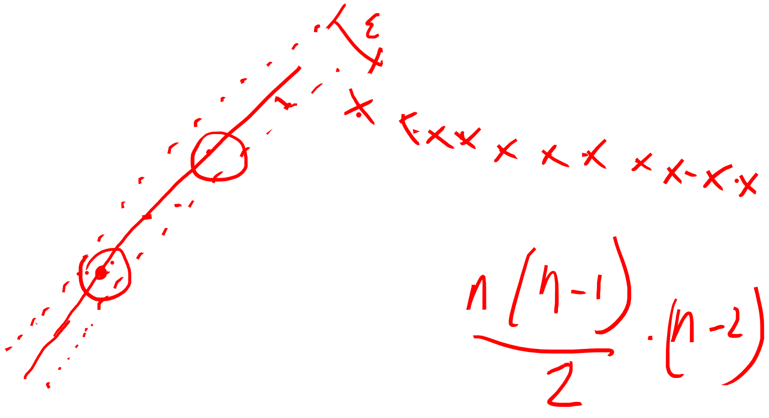


Line Detection: The problem

Brut-force attempt:

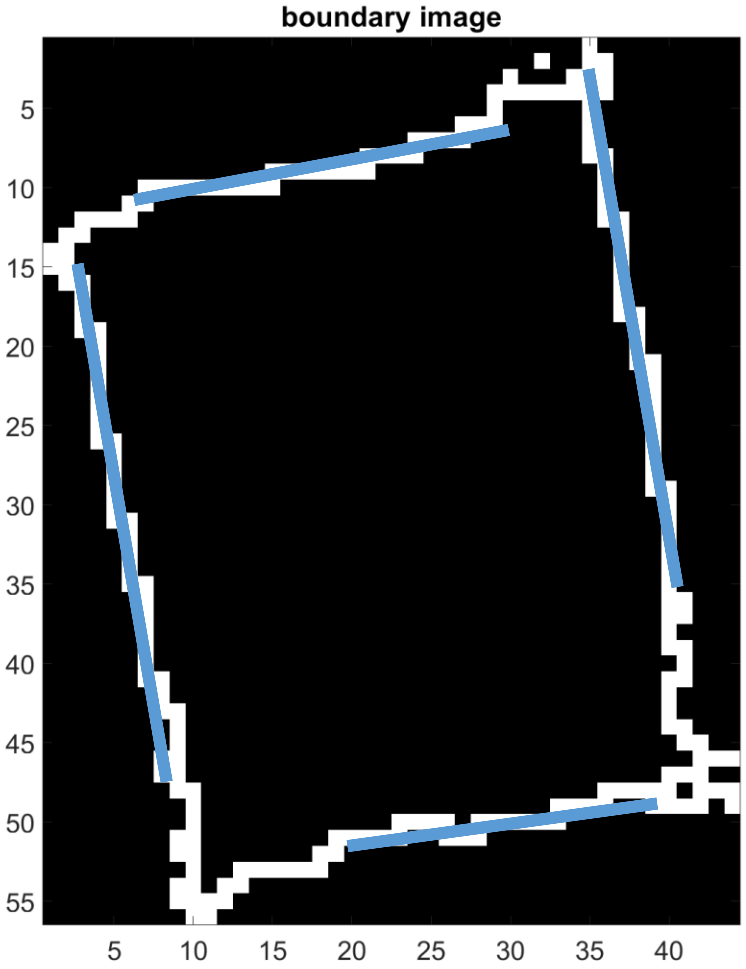
Given n points in a binary image, find subsets that lie on straight lines

- Compute all the lines passing through **any pair of points**
- Check **subsets of points** that belong / are close to these lines



n

$$\frac{n(n-1)}{2} \cdot (n-2) \sim n^3$$

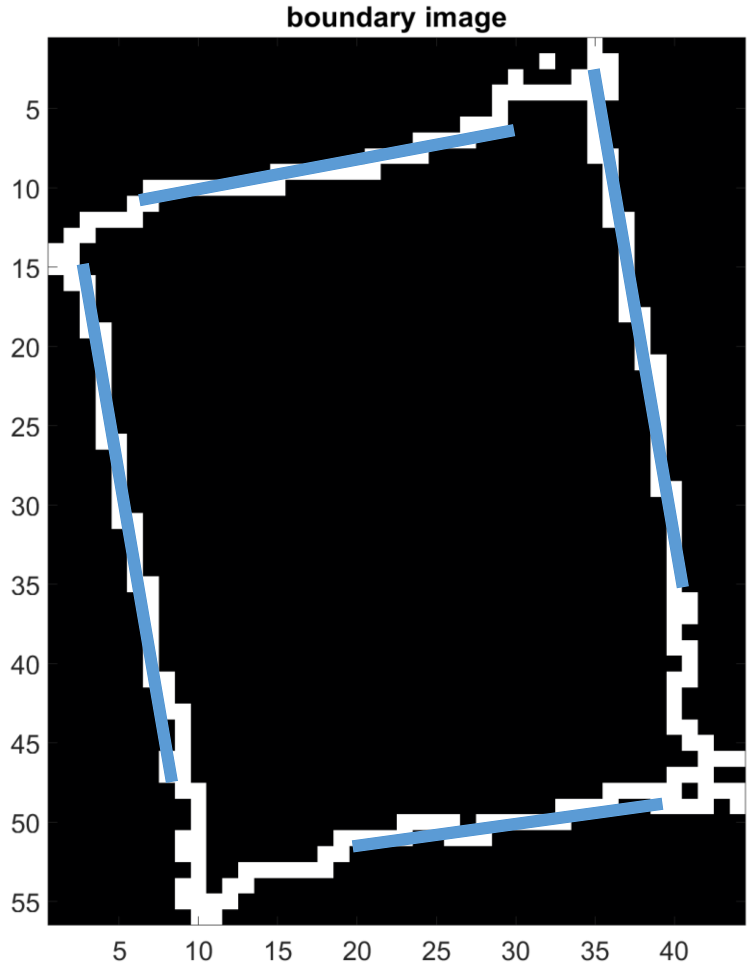


Line Detection: The problem

Brut-force attempt:

This requires computing

- $\frac{n(n-1)}{2}$ straight lines
- $n\left(\frac{n(n-1)}{2}\right)$ comparisons
- Computationally prohibitive task in all but the most trivial applications $\sim n^3$

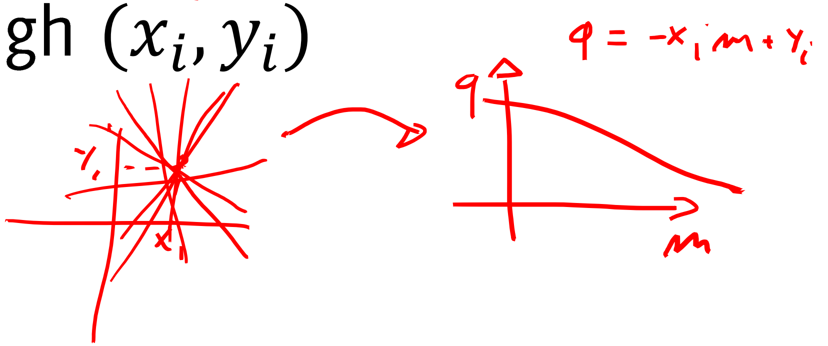


Hough Transform

Identify lines in the *"parameter space"* i.e. in the space of the parameters identifying lines (m, q). Let a straight line be:

$$y = mx + q$$

Now, for a given point (x_i, y_i), the equation $q = -x_i m + y_i$ in the variables m, q denotes the star of lines passing through (x_i, y_i)

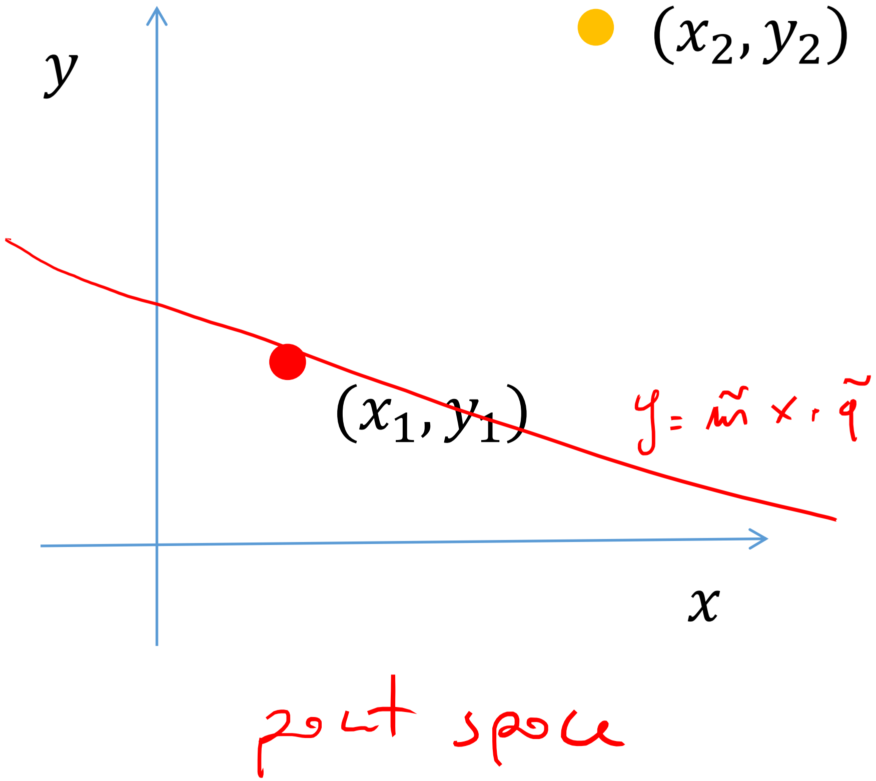


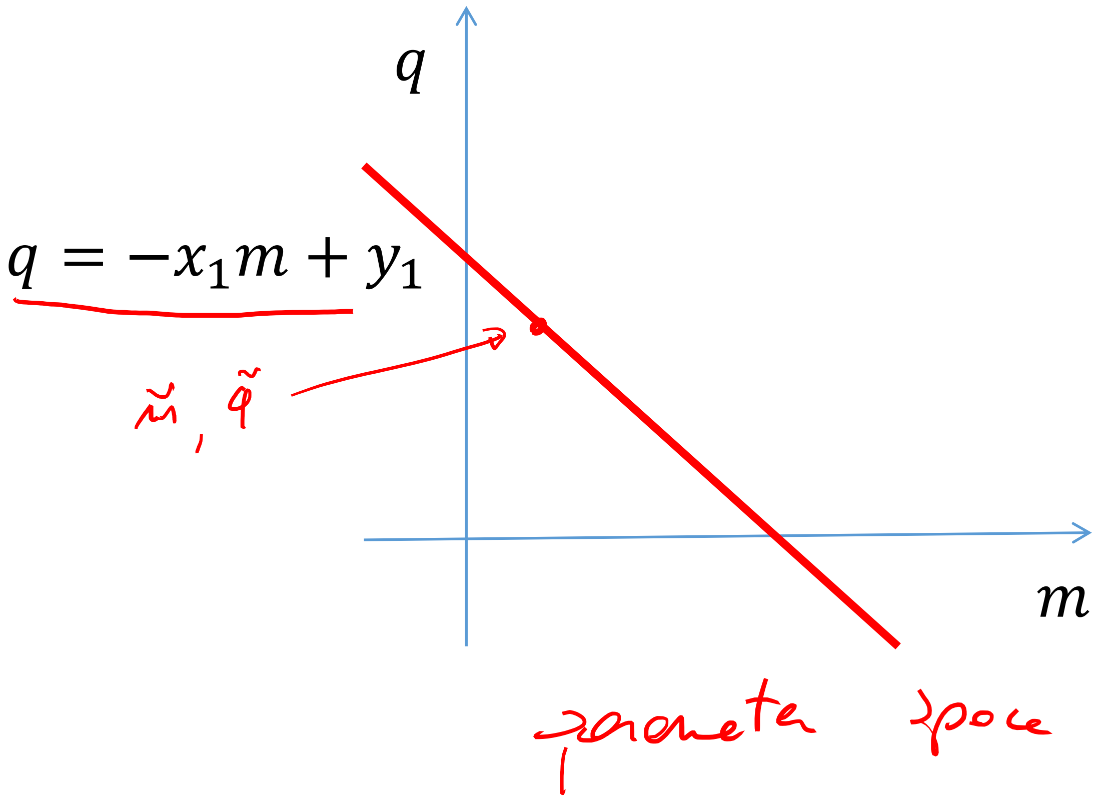
However,

$$q = -x_i m + y_i$$

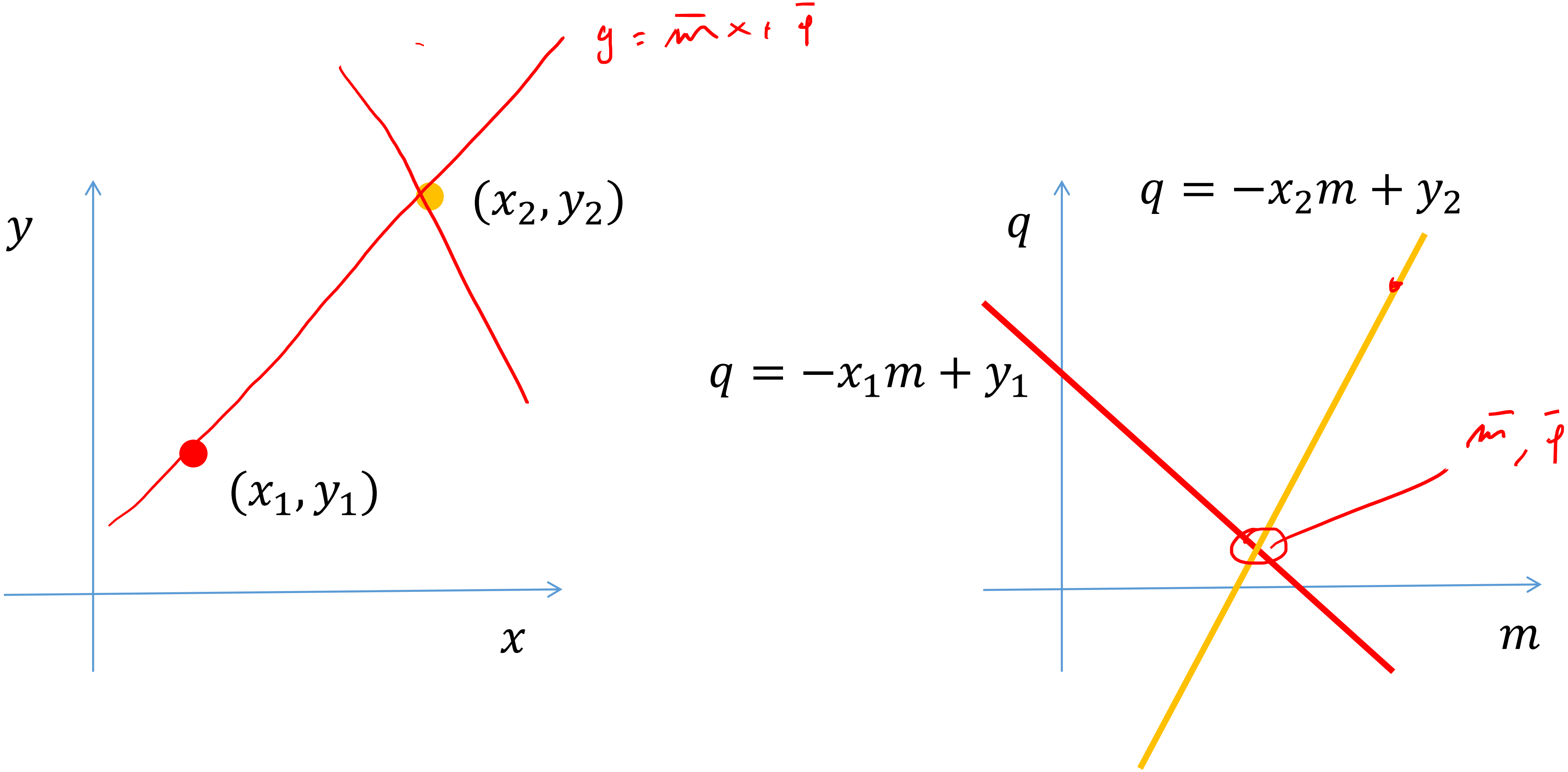
Can be also seen as the equation of a stragith line in m, q in the the parameter space

Intersections in the parameter space

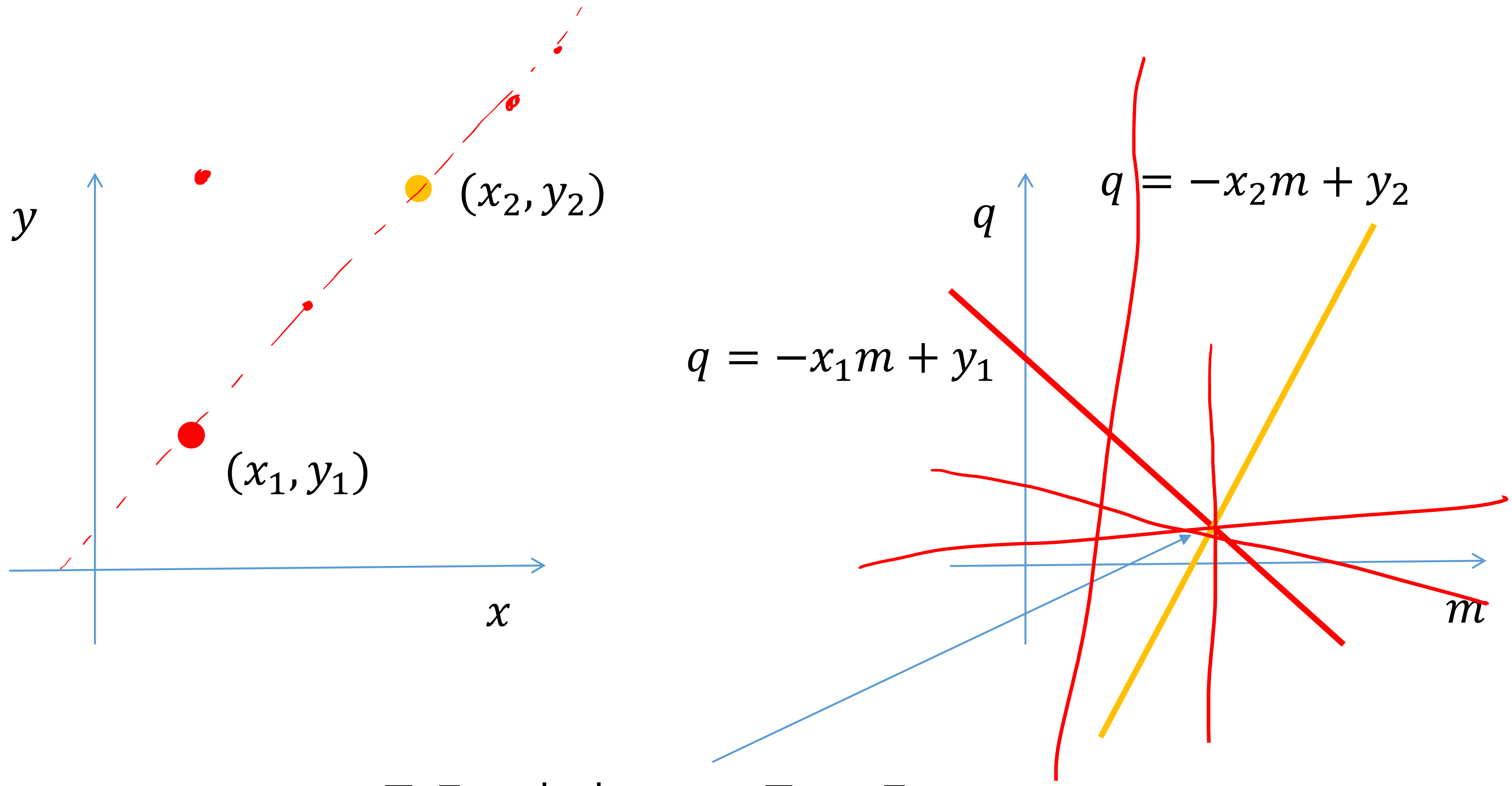




Intersections in the parameter space

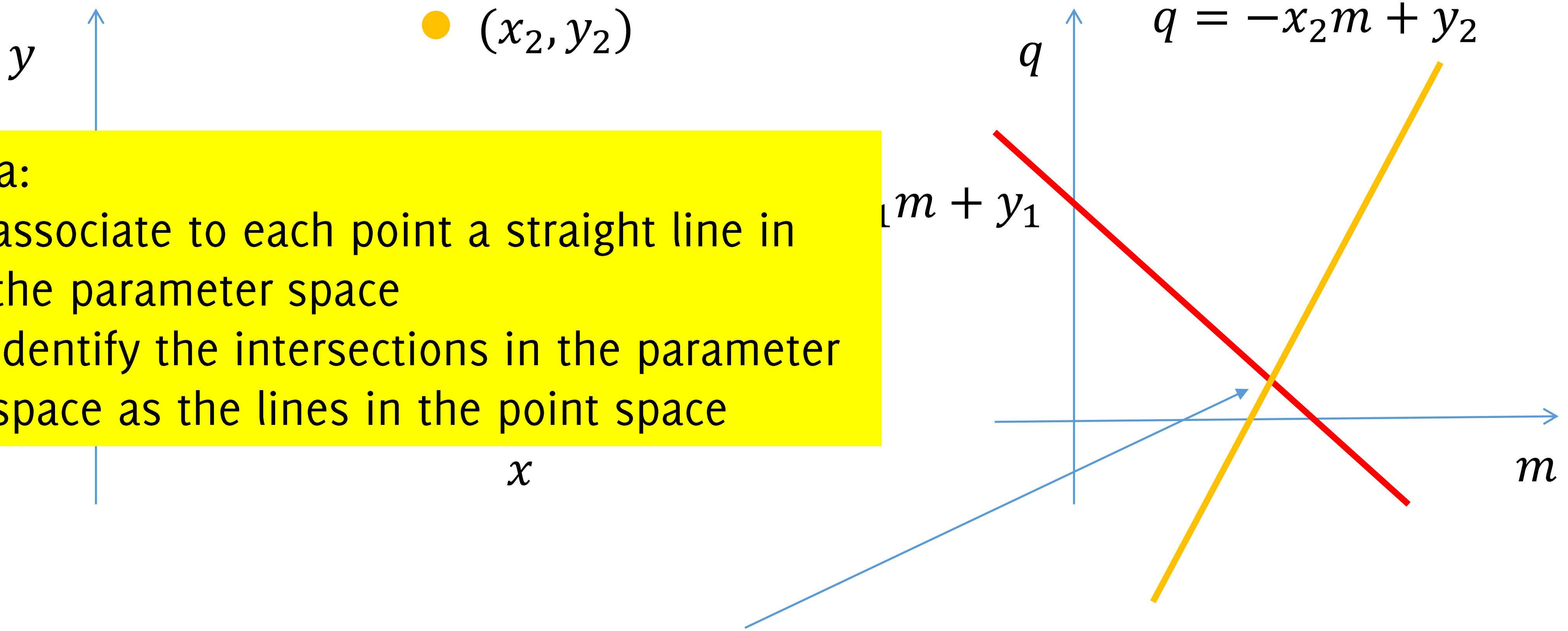


Intersections in the parameter space



$\bar{m}, \bar{q}$ such that $y = \bar{m}x + \bar{q}$ passes through both (x_1, y_1) and (x_2, y_2)

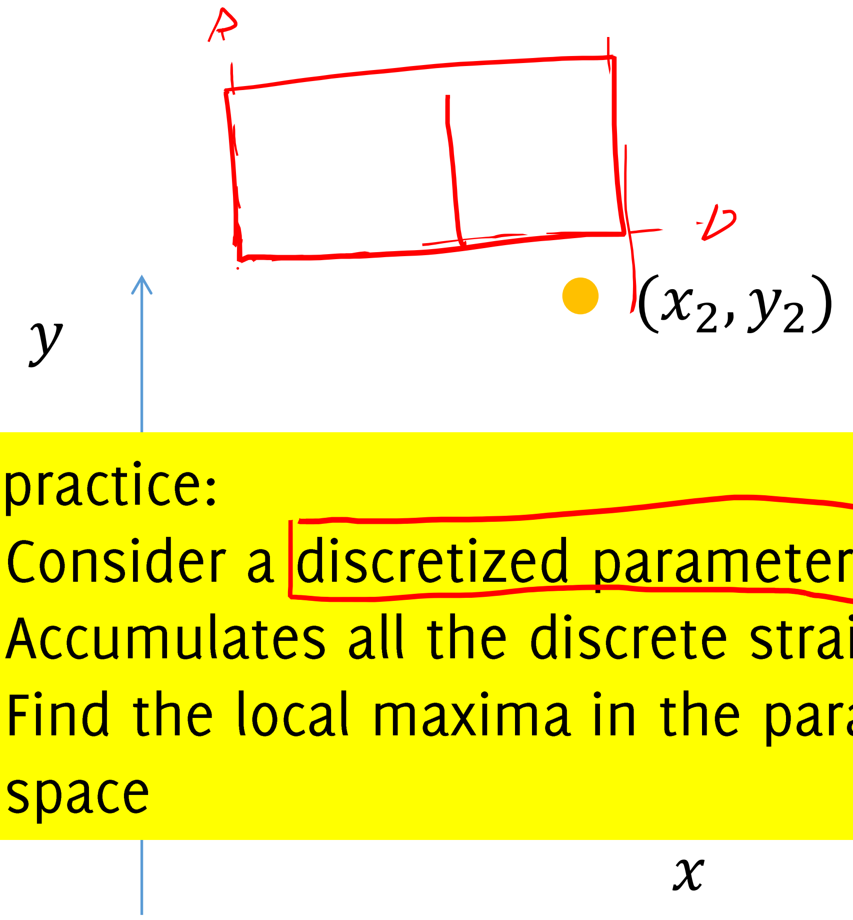
Intersections in the parameter space



Idea:
- associate to each point a straight line in the parameter space
- Identify the intersections in the parameter space as the lines in the point space

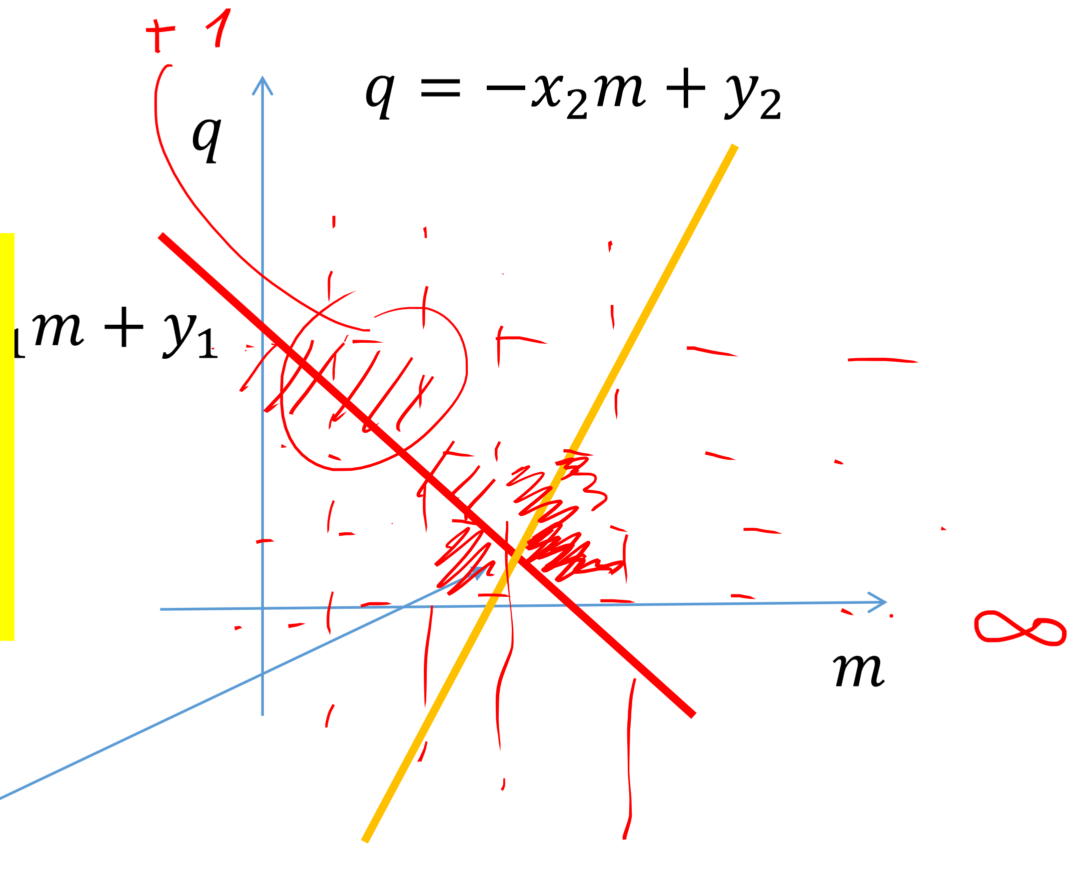
$\bar{m}, \bar{q}$ such that $y = \bar{m}x + \bar{q}$ passes through both (x_1, y_1) and (x_2, y_2)

Intersections in the parameter space



In practice:

- Consider a discretized parameter space
- Accumulates all the discrete straight lines
- Find the local maxima in the parameter space



$\bar{m}, \bar{q}$ such that $y = \bar{m}x + \bar{q}$ passes through both (x_1, y_1) and (x_2, y_2)

Hough Transform

Identify lines in the “parameter space” i.e. in the space of the parameters identifying lines.

$$q = -x_i m + y_i, \qquad \forall (x_i, y_i)$$

Core Idea:

- Discretize the parameter space where m, q live
- Accumulate the consensus in the parameter space by summing +1 at those bins where a straight line passess through
- Locate local maxima in the accumulator space

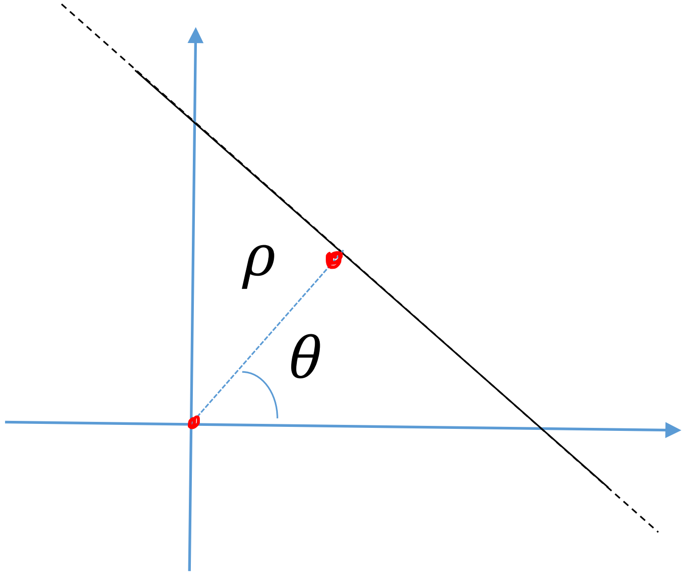
Major issue: m goes to infinity at vertical lines!

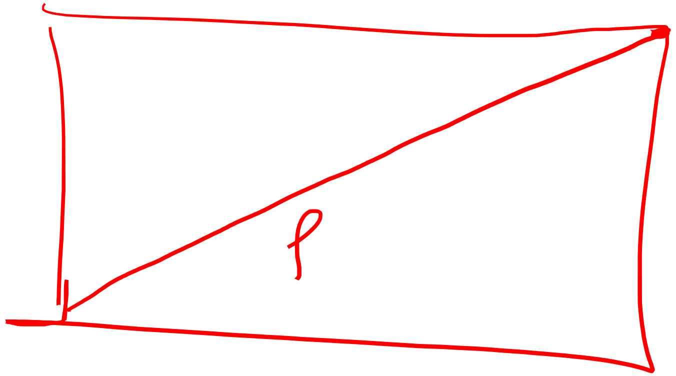
Hough Transform

There is a more convenient way of expressing a strainght line for this purpose:

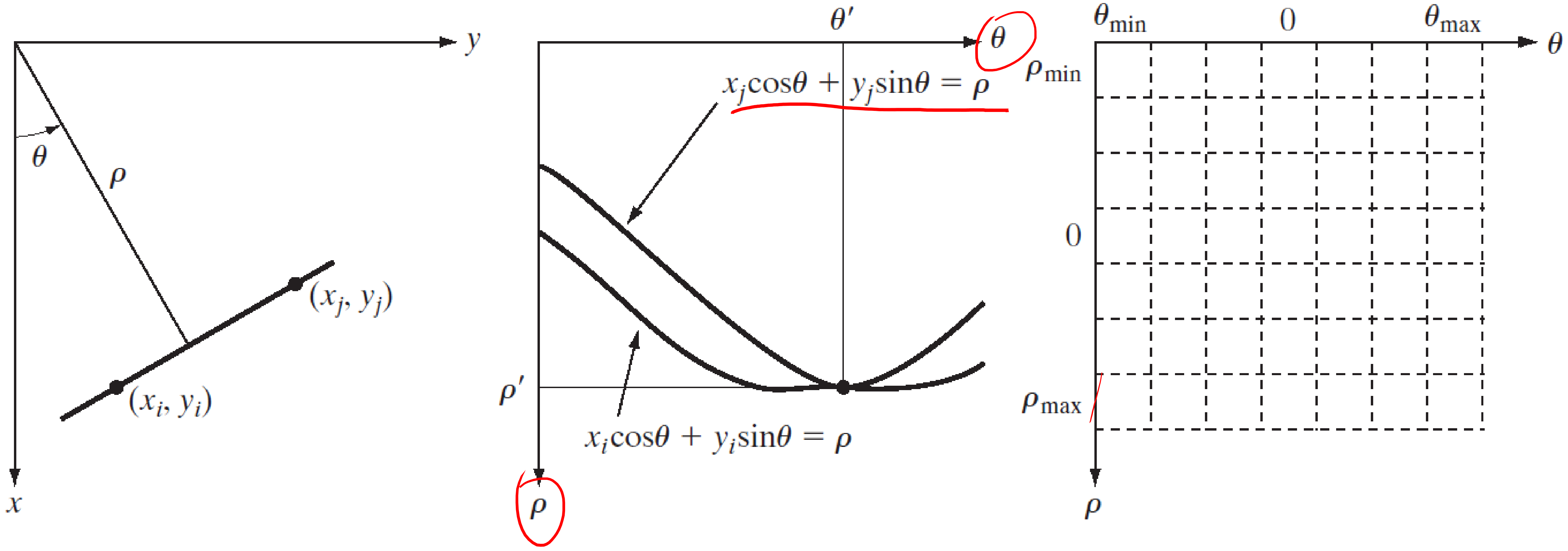
$$x\cos(\theta) + y\sin(\theta) = \rho$$

Where $\left\{(\rho, \theta),\ \rho \in [-L, L],\ \theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]\right\}$





New parametrization of straight lines



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

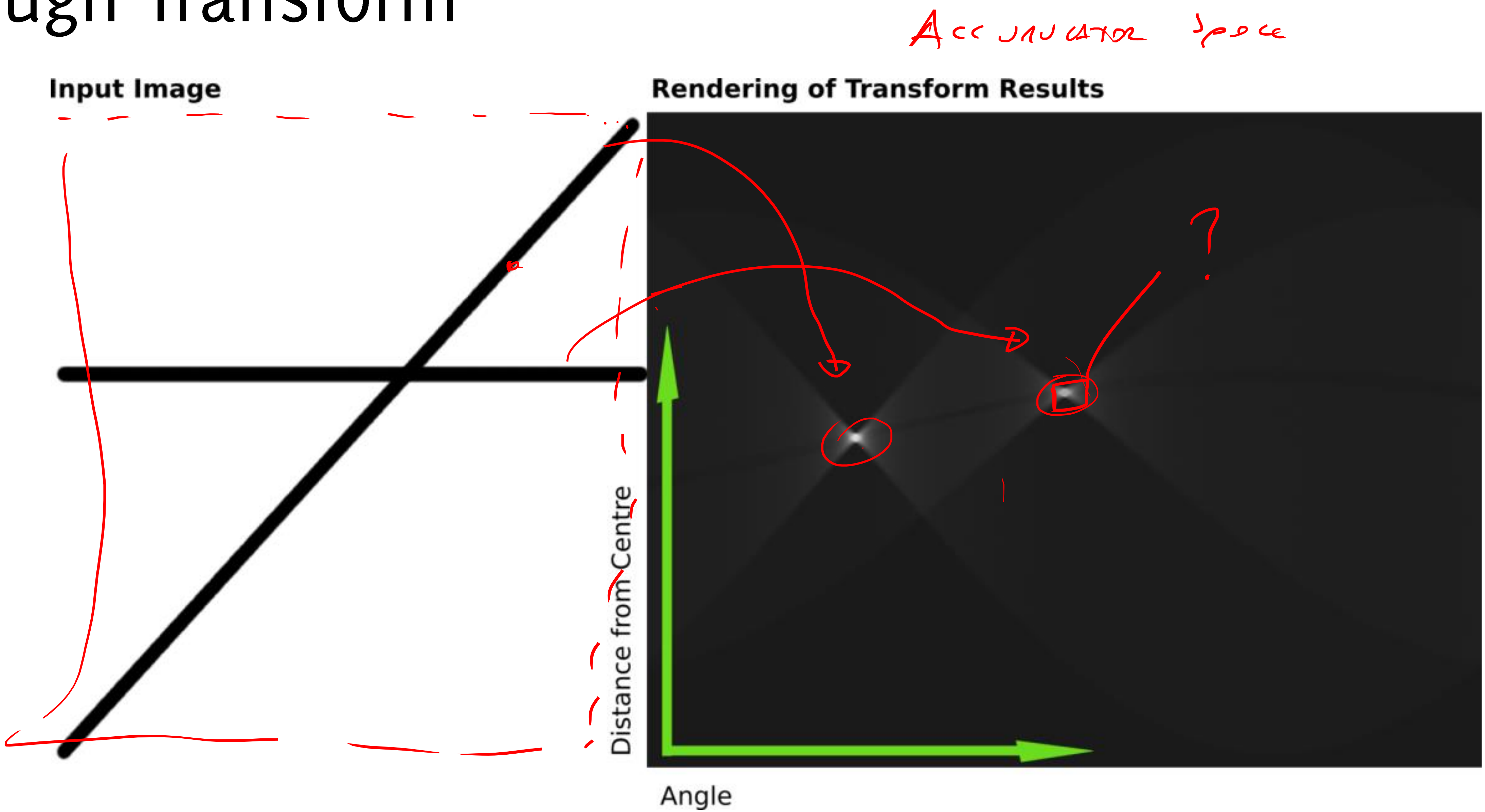
Hough Transform

The Hough transform identifies **through an optimized voting procedure** the most represented lines

The voting procedure is performed in the «accumulator space» which is a grid in (ρ, θ)-domain, for all the possible values.

From the Accumulator space we then extract local maxima, namely pairs (ρ, θ) corresponding to lines passing through most of points

Hough Transform



Hough Transform

The approach is not only limited to lines, but rather to any parametric model that we are able to fit

- Circles can be fit in a 3d accumulator space

It is quite robust to noise