Edge Detection
CS-477 Computer Vision

Dr. Mohsin Kamal
Associate Professor
dr.mohsinkamal@seecs.edu.pk

School of Electrical Engineering and Computer Science (SEECS)
National University of Sciences and Technology (NUST), Pakistan

1 Basics

2 Edge detectors

Edge detection includes a variety of mathematical methods that aim at identifying edges, curves in a digital image at which the image brightness changes sharply or, more formally, has discontinuities.

1 Basics

2 Edge detectors

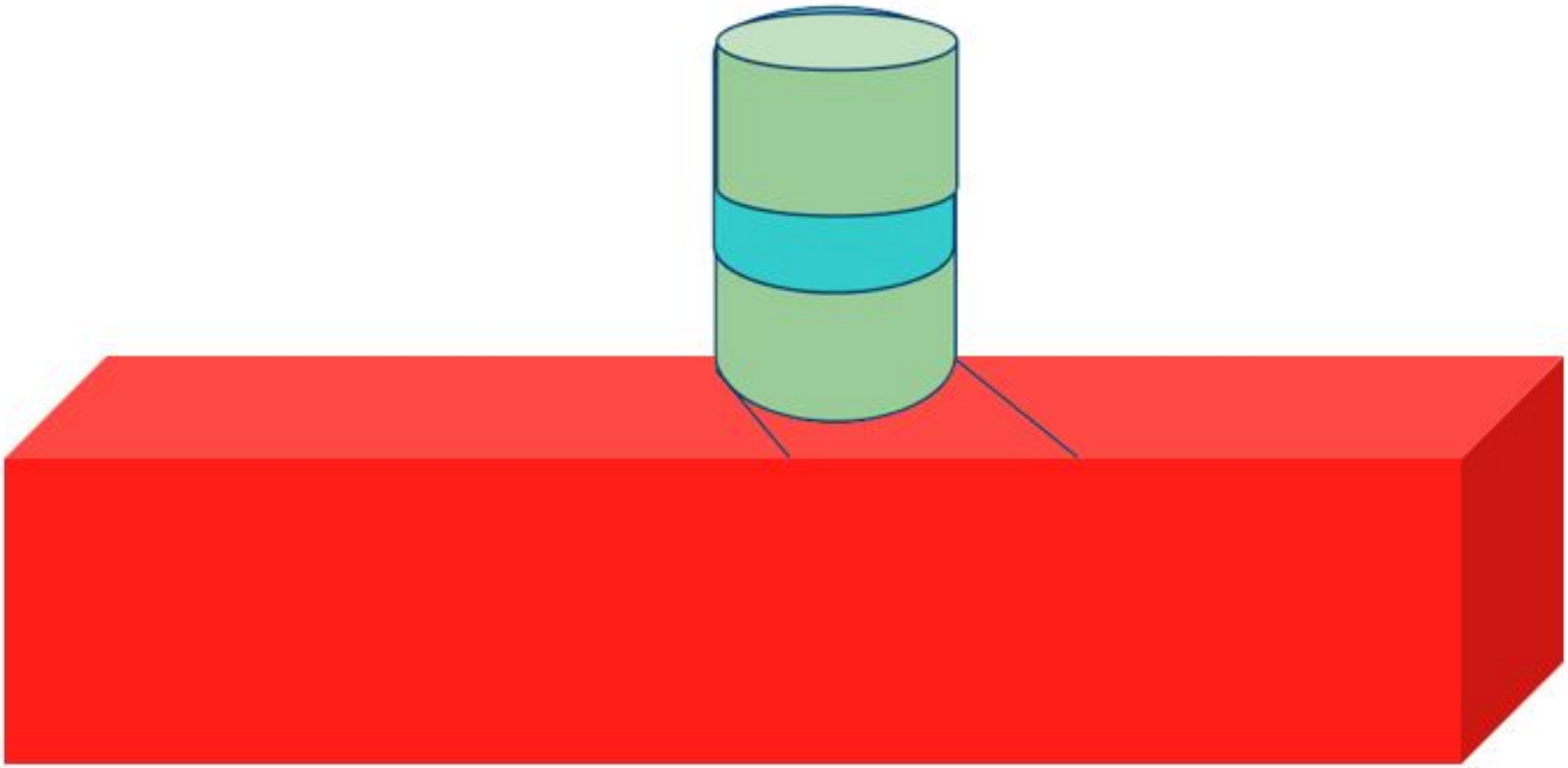
Examples

Application of edge

- Edge is the basic building block for analyzing an image
- The main aim of finding an edge is to reduce the information.
- There are many pixels in an image, among which many are irrelevant.

Edge detection in images

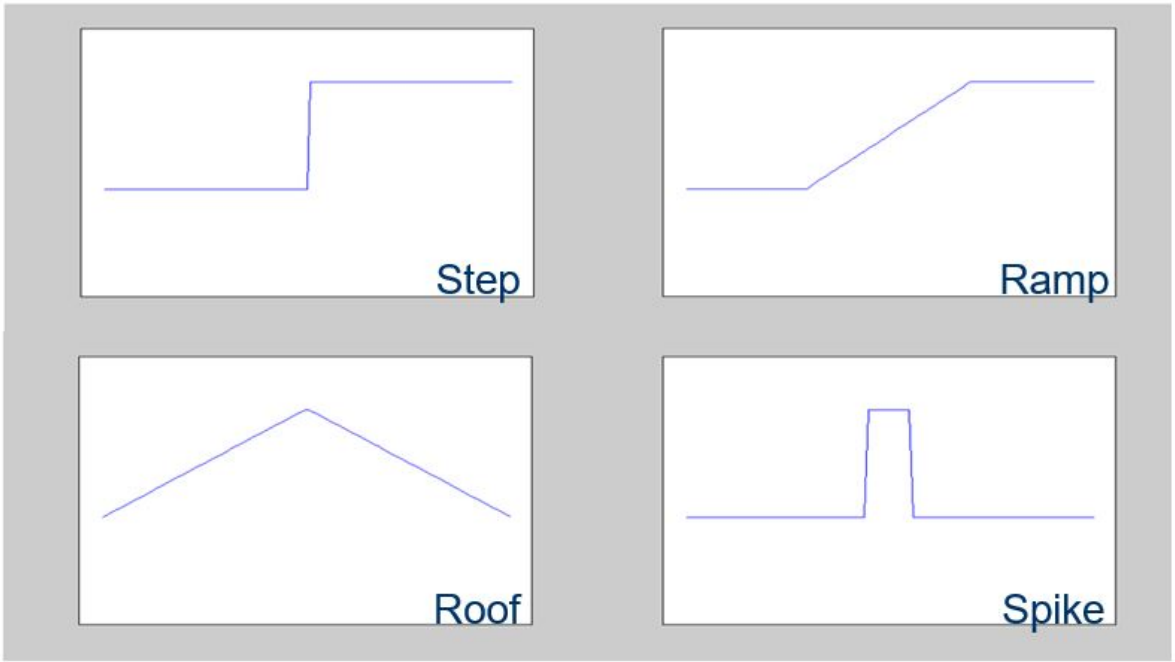
- At edges intensity or color changes
- In other words, discontinuity is observed



What is an edge?

Discontinuity of intensities in the image

- Edge models
  - Step
  - Roof
  - Ramp
  - Spike



Detecting Discontinuities

- Image derivatives

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x+\epsilon) - f(x)}{\epsilon} \right) \rightarrow \frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}) - f(x)}{\Delta x}$$

- Convolve image with derivative filters
  - Backward difference: $[-1 \quad 1]$
  - Forward difference: $[1 \quad -1]$
  - Central difference: $[-1 \quad 0 \quad 1]$

Derivatives in two-dimensions

- Definition

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

- Approximation

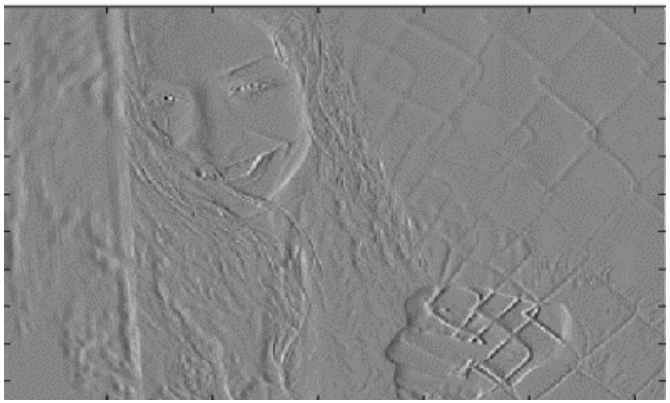
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x_{n+1},y_m) - f(x_n,y_m)}{\Delta x} \qquad \frac{\partial f(x,y)}{\partial y} \approx \frac{f(x_n,y_{m+1}) - f(x_n,y_m)}{\Delta y}$$

- Convolution kernels

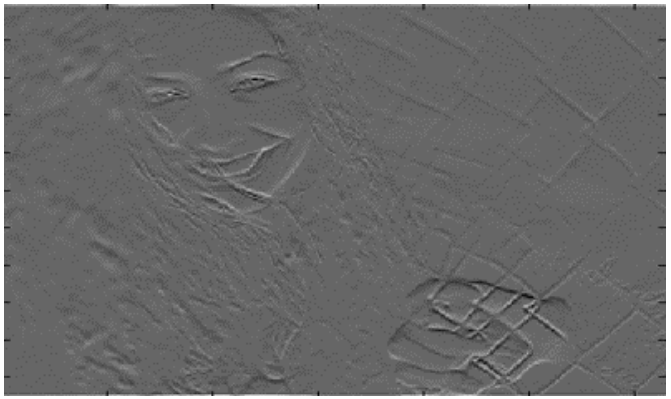
$$f_x = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad f_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Image derivatives

Image I



$$I_x = I * \begin{bmatrix} 1 & -1 \end{bmatrix}$$



$$I_y = I * \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Image derivatives

An edge is a place of rapid change in the image intensity function

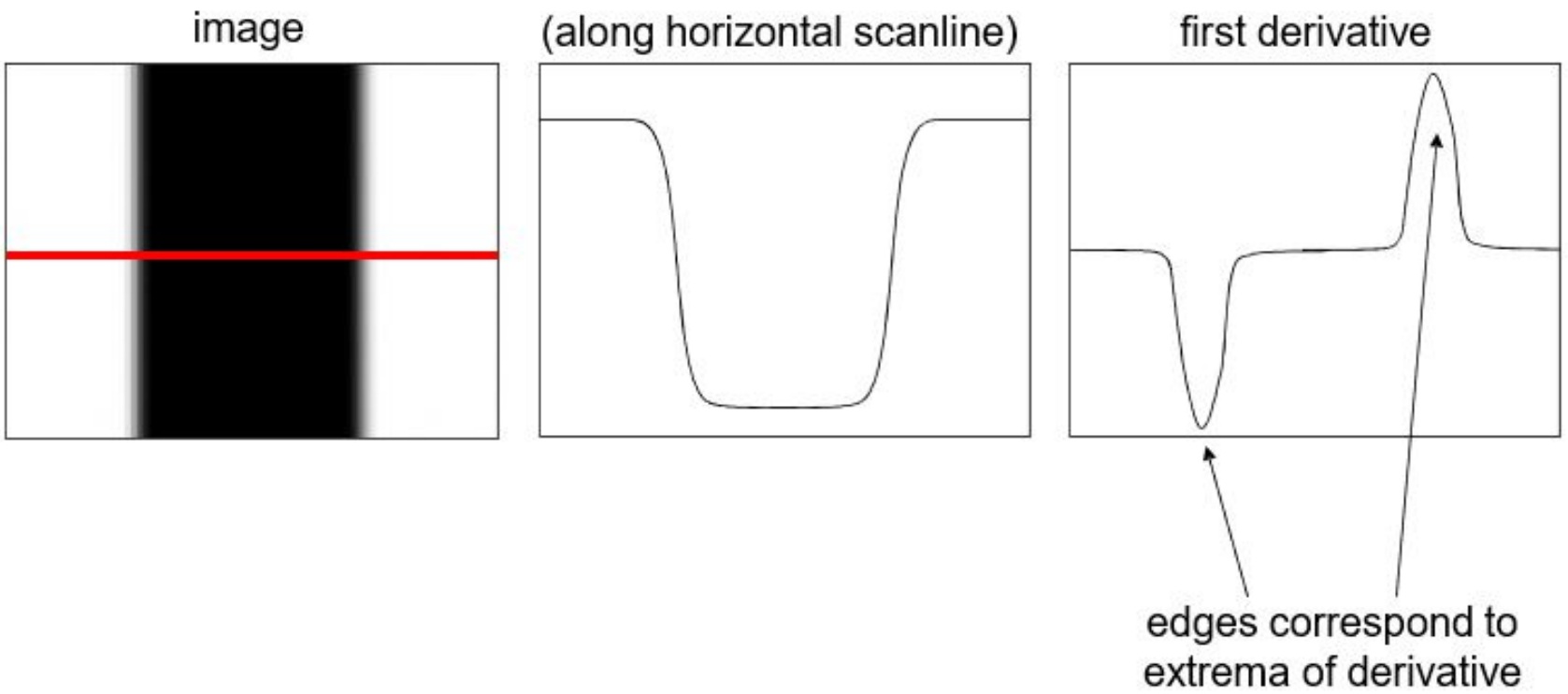
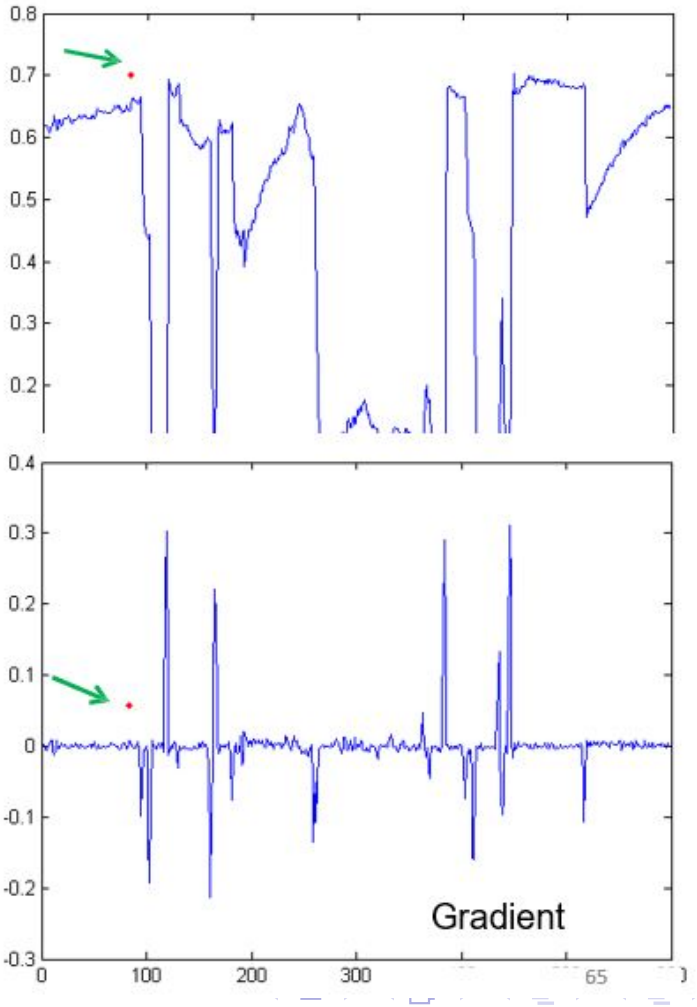


Image derivatives

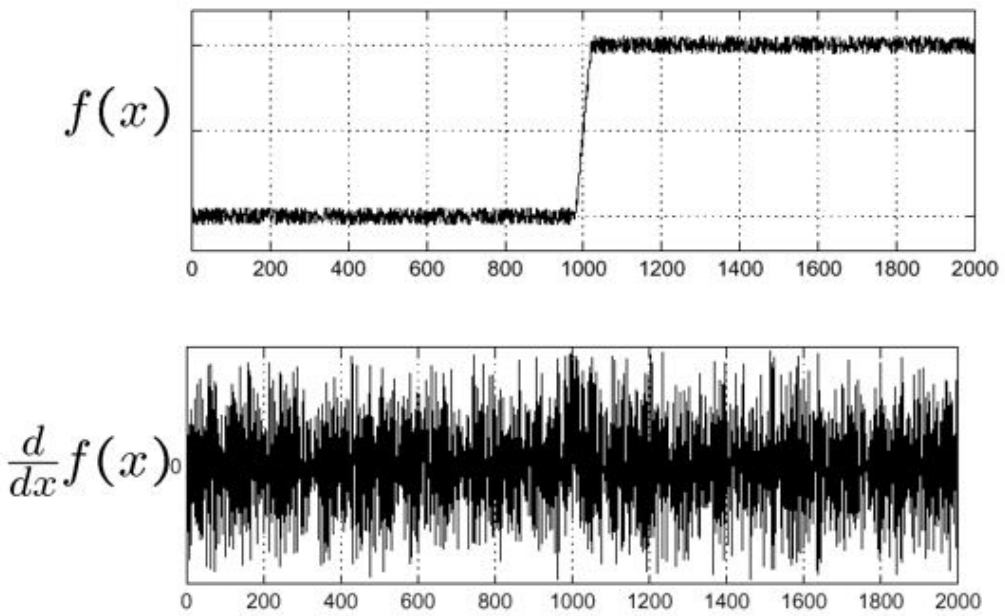


Derivatives and noise

Effects of noise:

Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal

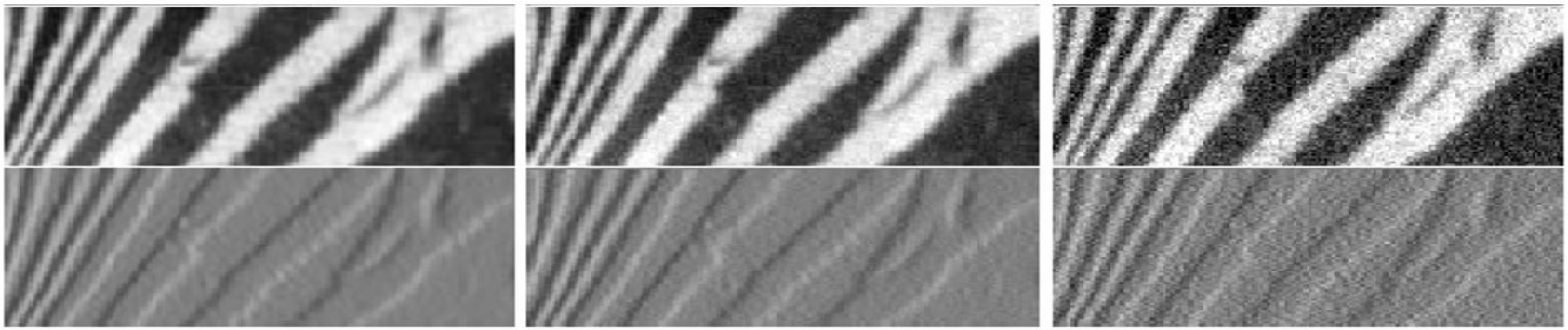


Derivatives and noise

- Strongly affected by noise
  - Reason: Image noise results in pixels that look very different from their neighbors
- The larger the noise, the stronger the response

- What needs to be done?
  - Neighbouring pixels look alike
  - Pixel along an edge look alike
  - Image smoothing should help
    - Force pixels different to their neighbors (possibly noise) to look like neighbors.

Derivatives and noise



Increasing noise →

Zero mean additive gaussian noise

Image smoothing

- Expect pixels to "**be like**" their neighbors
  - Relatively few reflectance changes
- Generally expect noise to be independent from pixel to pixel
  - Smoothing suppresses noise

Gaussian smoothing

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

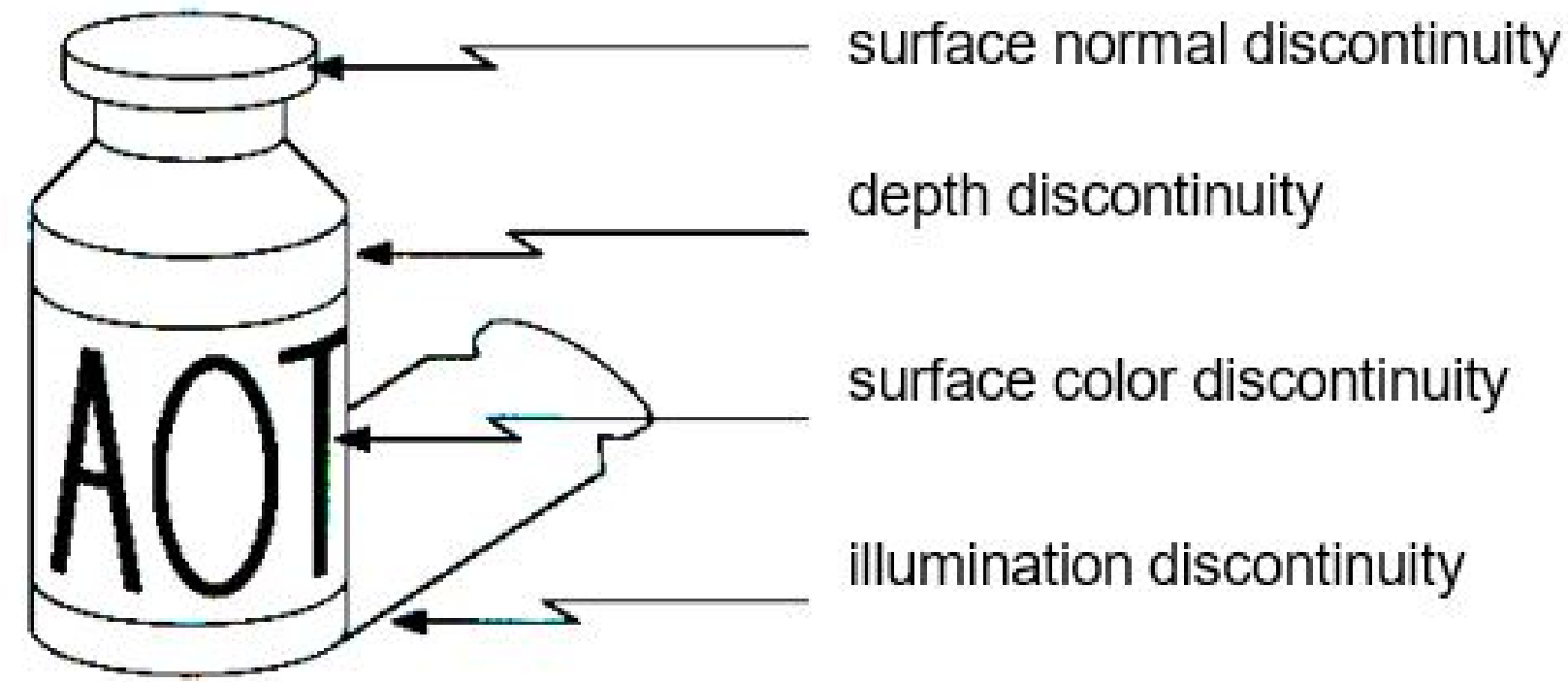
- Scale of Gaussian
  - As σ increases, more pixels are involved in average
  - As σ increases, image is more blurred
  - As σ increases, noise is more effectively suppressed

Examples of Gaussian smoothing

1 Basics

2 Edge detectors

- **Goal:** Identify visual changes (discontinuities) in an image
- Intuitively, semantic information is encoded in edges
- Edges are caused by variety of factors as provided in Figure below



- Gradient operators
  - Prewit
  - Sobel
- Laplacian of Gaussian (Marr-Hildreth)
- Gradient of Gaussian (Canny)

Prewitt and Sobel edge detector

- Compute derivatives
  - In x and y directions
- Find gradient magnitudes
- Threshold gradient magnitude

Prewitt edge detector

Vertical Edges:

$$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Horizontal Edges:

$$h_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Gradient:

$G_x = I * h_x$

$G_y = I * h_y$

Sobel edge detector

- The central row in h_x and h_y has more weights
- It is performing weighted averaging

Vertical Edges:

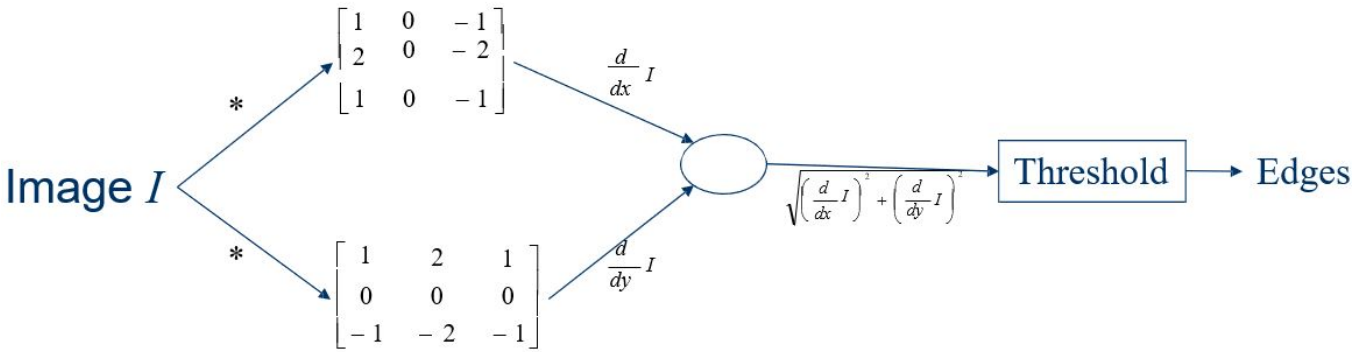
$$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Horizontal Edges:

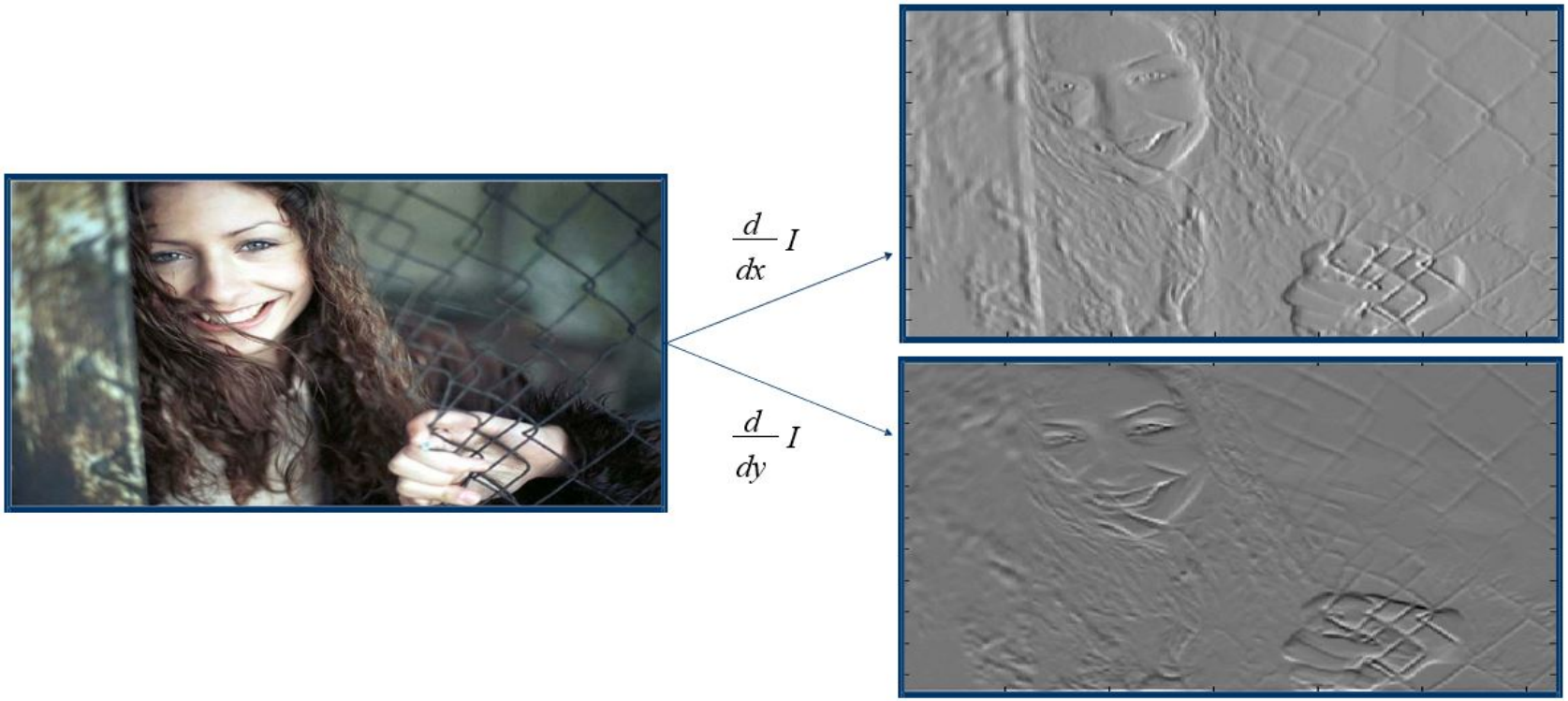
$$h_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Gradient:

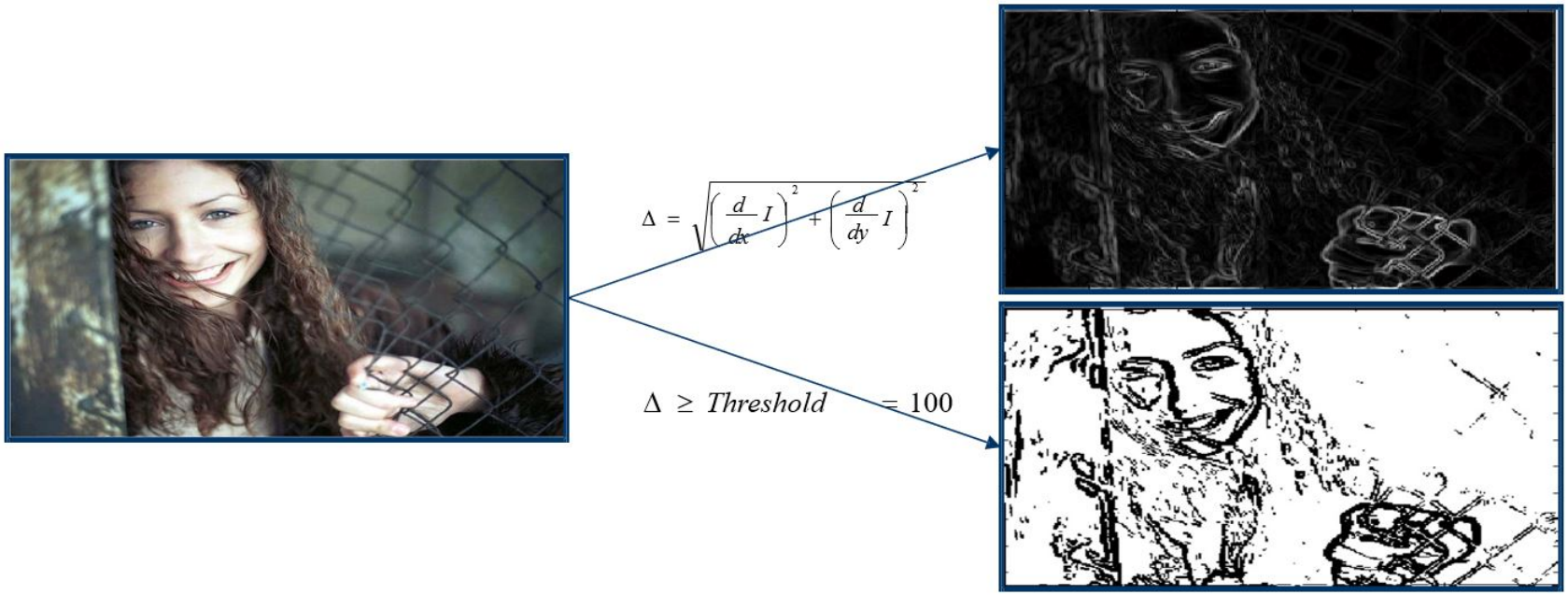
$$G_x = I * h_x$$
$$G_y = I * h_y$$



Sobel edge detector

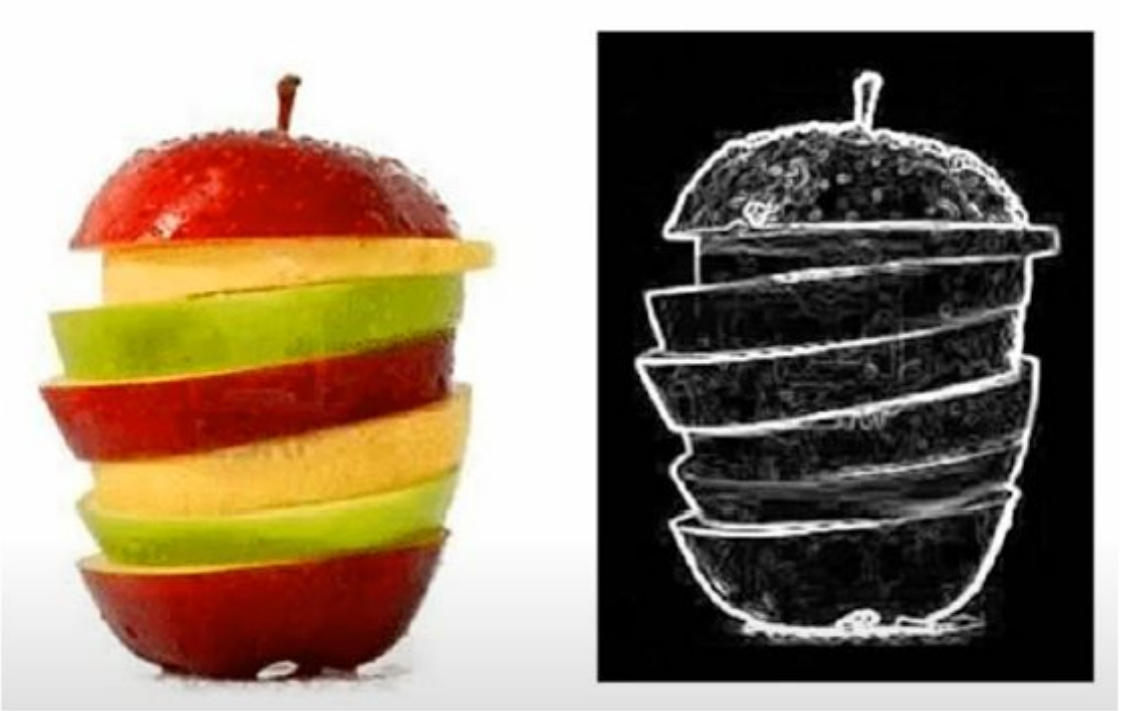


Sobel edge detector



Drawback

Edges are not clear



Marr Hildreth edge detector (Laplacian of Gaussian)

- Smooth image by Gaussian filter $\rightarrow S$
- Apply Laplacian to S
  - Used in mechanics, electromagnetics, wave theory, quantum mechanics and Laplace equation
- Find zero crossings
  - If there is no change in the intensity, the derivative will be zero.
  - Scan along each row, record an edge point at the location of zero-crossing.
  - Repeat above step along each column

Marr Hildreth edge detector (Laplacian of Gaussian)

Mathematically, Gaussian smoothing is represented as,

$$S = g * I \tag{1}$$

where,

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

Next step is to find the Laplacian of S. This is represented as,

$$\Delta^2 S = \frac{\partial^2}{\partial x^2} S + \frac{\partial^2}{\partial y^2} S \tag{3}$$

Marr Hildreth edge detector (Laplacian of Gaussian)

Derivation of the Laplacian of Gaussian (LoG)

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I \tag{4}$$

where $\Delta^2 g$ is represented as

$$\Delta^2 g = -\frac{1}{\sqrt{2\pi}\sigma^3}\left(2 - \frac{x^2+y^2}{\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5}$$

This means, rather than convolve the image with the gaussian and take the second order derivative of the result, you can actually take the 2nd derivative of the gaussian itself.

Marr Hildreth edge detector (Laplacian of Gaussian)

Gaussian

1-D Gaussian: $g(x) = e^{\frac{-x^2}{2\sigma^2}}$

x	-3	-2	-1	0	1	2	3
g(x)	.011	.13	.6	1	.6	.13	.011

2-D Gaussian: $g(x, y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}}$ for $\sigma = 2$

0	0	0	0	1	2	2	2	1	0	0	0	0
0	0	1	3	6	9	11	9	6	3	1	0	0
0	1	4	11	20	30	34	30	20	11	4	1	0
0	3	11	26	50	73	82	73	50	26	11	3	0
1	6	20	50	93	136	154	136	93	50	20	6	1
2	9	30	73	136	198	225	198	136	73	30	9	2
2	11	34	82	154	225	255	225	154	82	34	11	2
2	9	30	73	136	198	225	198	136	73	30	9	2
1	6	20	50	93	136	154	136	93	50	20	6	1
0	3	11	26	50	73	82	73	50	26	11	3	0
0	1	4	11	20	30	34	30	20	11	4	1	0
0	0	1	3	6	9	11	9	6	3	1	0	0
0	0	0	0	1	2	2	2	1	0	0	0	0

Here the mask is multiplied with 255 to have integer values from range 0-255.

Marr Hildreth edge detector (Laplacian of Gaussian)

LOG Filter

$$\Delta^2 g = -\frac{1}{\sqrt{2\pi}\sigma^3}\left(2 - \frac{x^2+y^2}{\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

			Y			
0.0008	0.0066	0.0215	0.031	0.0215	0.0066	0.0008
0.0066	0.0438	0.0982	0.108	0.0982	0.0438	0.0066
0.0215	0.0982	0	-0.242	0	0.0982	0.0215
0.031	0.108	-0.242	-0.7979	-0.242	0.108	0.031 X
0.0215	0.0982	0	-0.242	0	0.0982	0.0215
0.0066	0.0438	0.0982	0.108	0.0982	0.0438	0.0066
0.0008	0.0066	0.0215	0.031	0.0215	0.0066	0.0008

Finding zero crossings (1/2)

- When the first derivative is at a maxima or minima the second derivative is 0
- Pixels where a zero crossing occurs are marked as edges (if the slope of the crossing exceeds a threshold)
- Four cases of zero-crossings
  - Zero crossings occur in the image wherever a positive value is followed by a negative value, or vice versa. Or there may even be a 0 in between a positive and negative value.

Finding zero crossings (2/2)

- The absolute difference between the negative and positive values gives you the slope of the crossing, which is a measure of the strength of the edge. Slope of zero-crossing {a, -b} is |a+b|.
- To mark an edge
  - Compute slope of zero-crossing
  - Apply a threshold to slope

On the separability of LoG

Similar to separability of Gaussian filter

- Two-dimensional Gaussian can be separated into 2 one-dimensional Gaussians

$$h(x,y) = I(x,y) * g(x,y) \quad n^2 \text{ mults}$$
$$h(x,y) = (I(x,y) * g_1(x)) * g_2(y) \quad 2n \text{ mults}$$

where,

$g(x) = e^{-\left(\frac{x^2}{2\sigma^2}\right)}$ and n represents the mask

$$g_1 = g(x) = \begin{bmatrix} .011 & .13 & .6 & 1 & .6 & .13 & .011 \end{bmatrix} \quad g_2 = g(y) = \begin{bmatrix} .011 \\ .13 \\ .6 \\ 1 \\ .6 \\ .13 \\ .011 \end{bmatrix}$$

On the separability of LoG

$$\Delta^2 S = \Delta^2(g * I) = (\Delta^2 g) * I = I * (\Delta^2 g)$$

Requires n^2 multiplications

$$\Delta^2 S = (I * g_{xx}(x)) * g(y) + (I * g_{yy}(y)) * g(x)$$

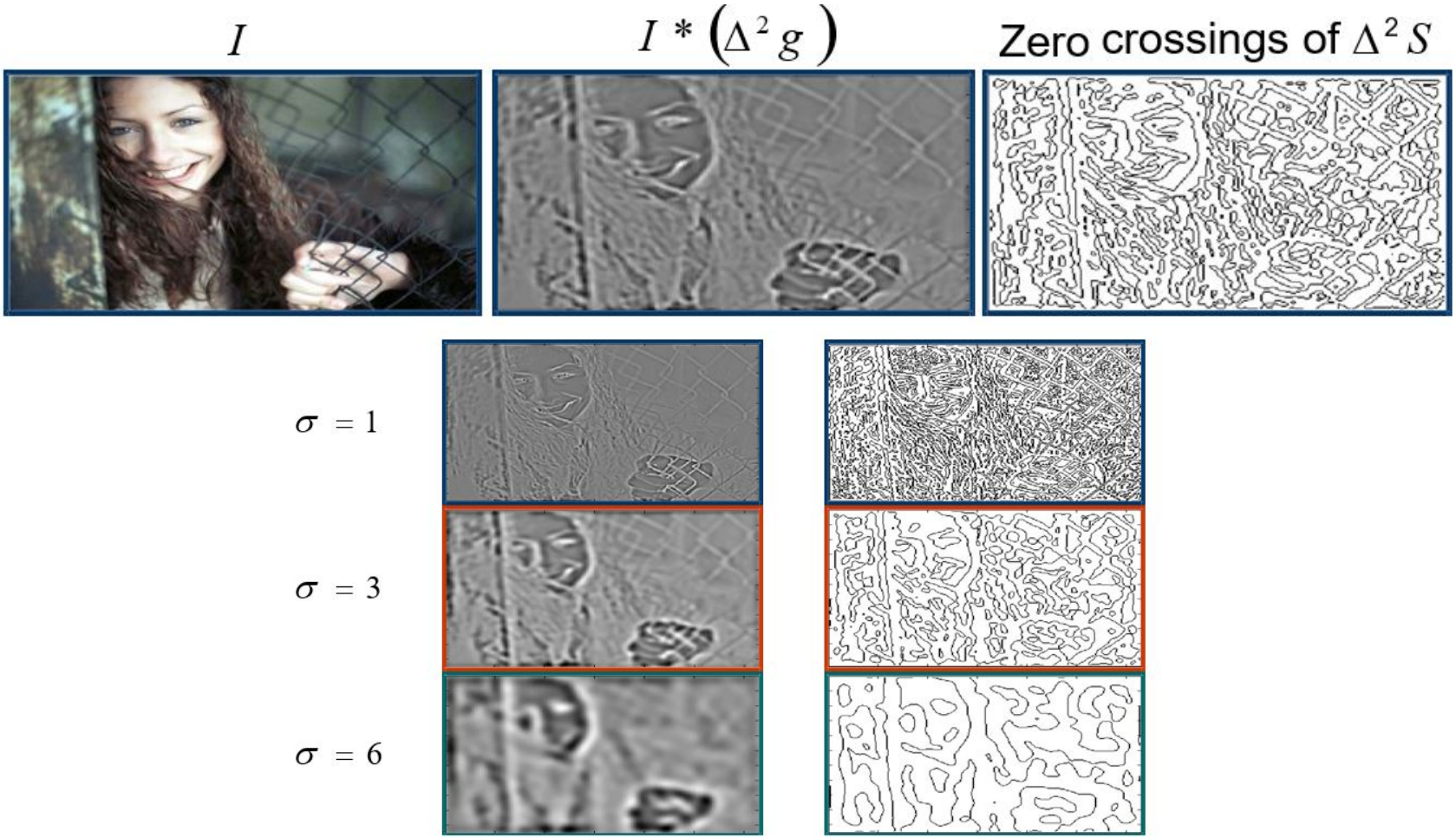
Requires $4n$ multiplications

Marr Hildreth edge detector (Laplacian of Gaussian)

Separability

Marr Hildreth edge detector (Laplacian of Gaussian)

Example



Algorithm

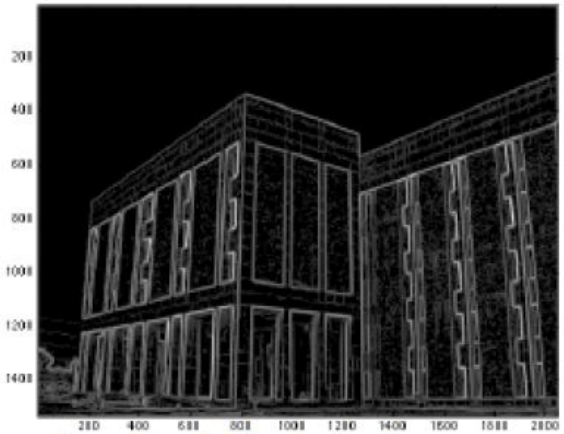
- Compute LoG
  - Use 2D filter i.e., $\Delta^2 g(x, y)$, OR
  - Use 4 1D filters i.e., $g(x), g_{xx}(x), g(y), g_{yy}(y)$
- Find zero-crossings from each row,
- Find slope of zero-crossings
- Apply threshold to slope and mark edges

Quality of an Edge

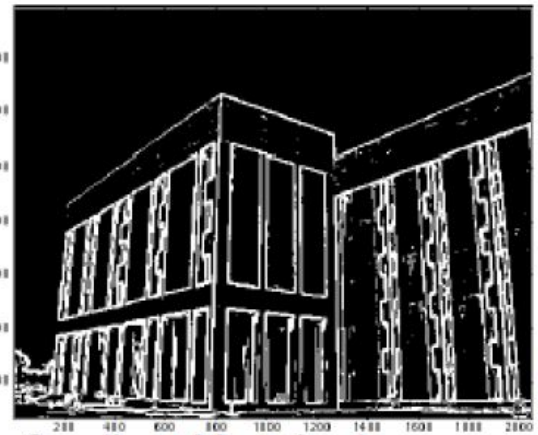
- Robust to noise
- Localization
- Too many or too less responses

Line detection

We can use binary edge mask to detect lines in images



Gradient Magnitude

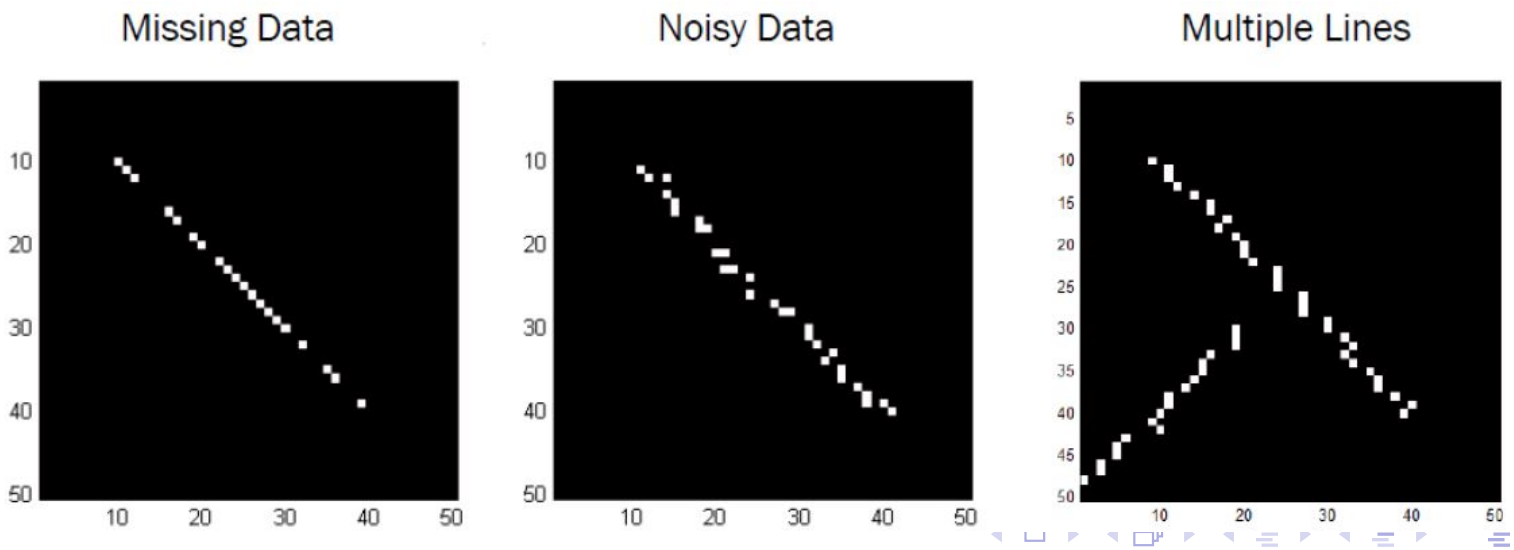


Gradient Magnitude Thresholded

Output of State of Art Line Detector

Quality of an Edge

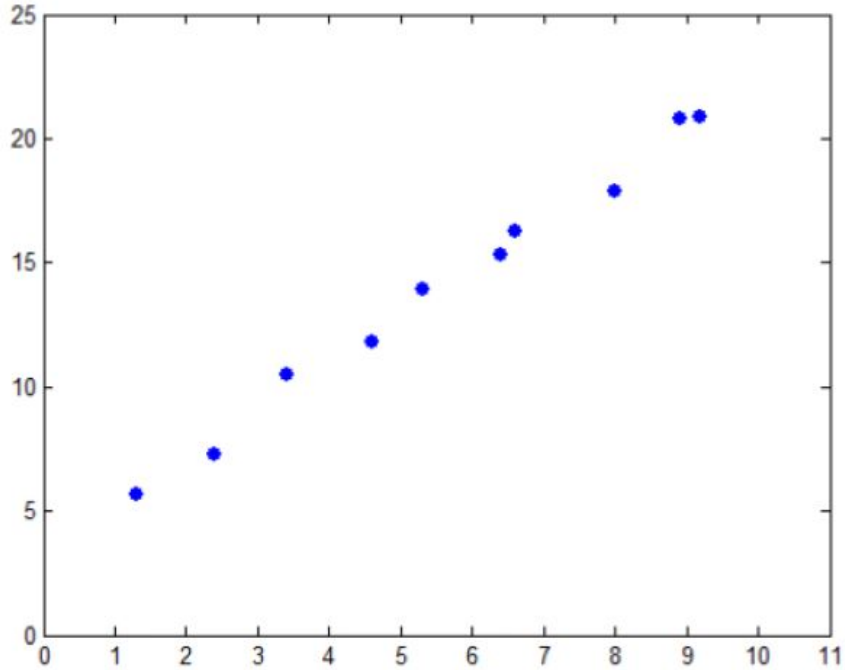
Line detection



Parameter optimization: Least Squared Error Solution

Fitting a line to a set of data points...

x	y
1.3	5.7
2.4	7.3
3.4	10.5
4.6	11.8
5.3	13.9
6.6	16.3
6.4	15.3
8.0	17.9
8.9	20.8
9.2	20.9



Can the line equation provide an estimated edge in this case?

Line fitting: least squared error solution

- Step 1: Identify the model
  - Equation of line: $y = mx + c$
- Step 2: Set up an error term which will give the goodness of every point with respect to the (unknown) model
  - Error induced by the i^{th} point:
    $$e_i = mx_i + c - y_i$$
  - Error for whole data: $E = \sum_i e_i^2$
    $$E = \sum_i (mx_i + c - y_i)^2$$
- Step 3: Differentiate Error w.r.t. parameters, put equal to zero and solve for minimum point

Line fitting: least squared error solution

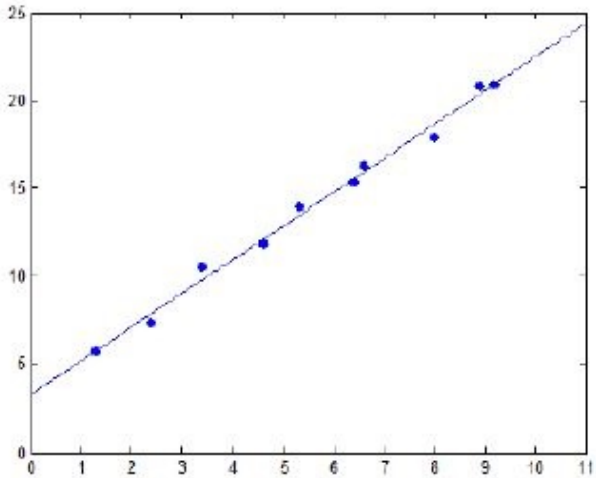
$$E = \sum_i (mx_i + c - y_i)^2$$

$$\frac{\partial E}{\partial m} = \sum_i (mx_i + c - y_i)x_i = 0$$

$$\frac{\partial E}{\partial c} = \sum_i (mx_i + c - y_i) = 0$$

$$\begin{bmatrix} \sum_i x_i^2 & \sum_i x_i \\ \sum_i x_i & \sum_i 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum_i x_i y_i \\ \sum_i y_i \end{bmatrix}$$

x	y
1.3	5.7
2.4	7.3
3.4	10.5
4.6	11.8
5.3	13.9
6.6	16.3
6.4	15.3
8.0	17.9
8.9	20.8
9.2	20.9

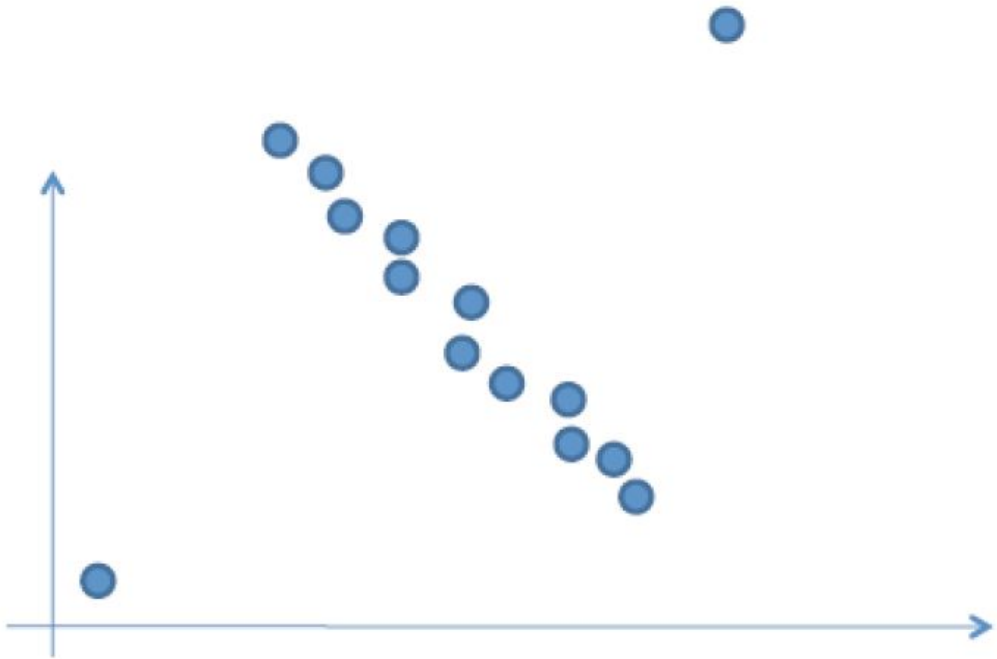


$$\begin{bmatrix} 380.63 & 56.1 \\ 56.1 & 10 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} 914.68 \\ 140.4 \end{bmatrix}$$

Solution: $m = 1.9274$ $c = 3.227$

Least squared error solution

- Disadvantages are:
  - Multiple lines
  - Not robust to noise

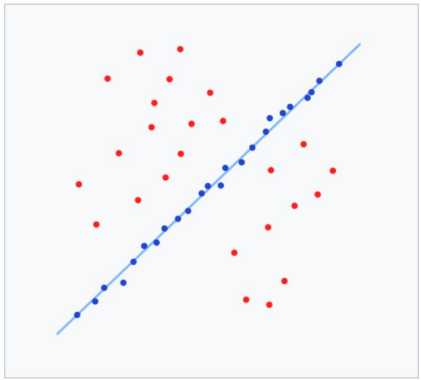


Another approach

RANSAC (RANdom SAmple Consensus[1])

- It is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates.
- Roboust against noise

A data set with many outliers for which a line has to be fitted.



Fitted line with RANSAC; outliers have no influence on the result.

[1] https://en.wikipedia.org/wiki/Random_sample_consensus

RANSAC (RANdom SAmple Consensus)

Algorithm:

1. Sample (randomly) the number of points required to fit the model
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

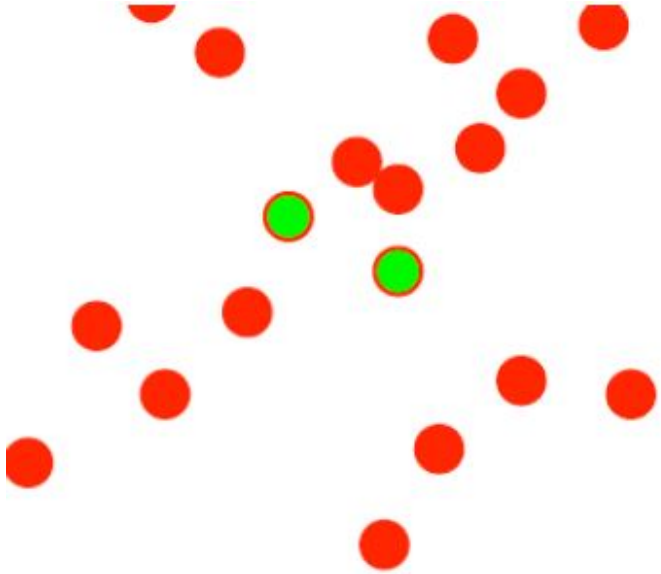
Repeat 1-3 until the best model is found with high confidence

RANSAC (RANdom SAmple Consensus)

Algorithm:

1. Sample (randomly) the number of points required to fit the model ($n = 2$)
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

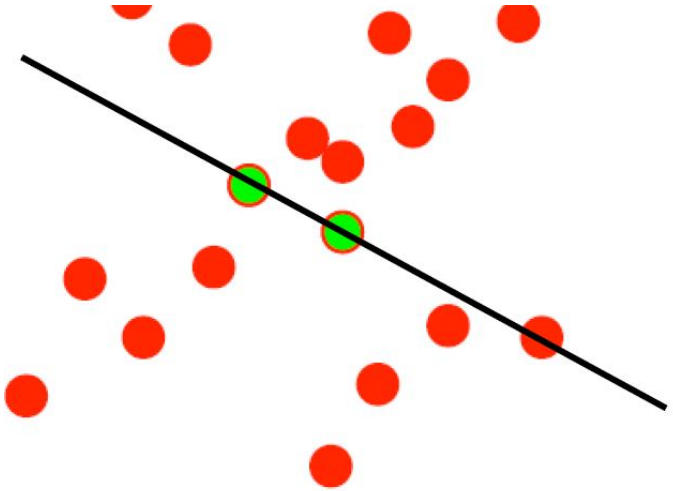


RANSAC (RANdom SAmple Consensus)

Algorithm:

1. Sample (randomly) the number of points required to fit the model ($n = 2$)
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

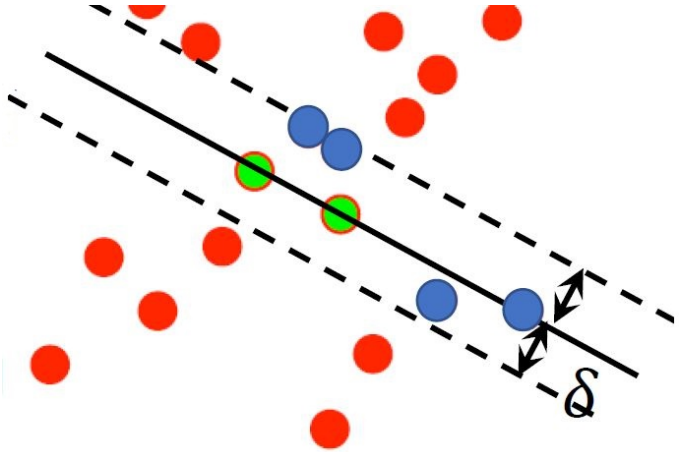


RANSAC (RANdom SAmple Consensus)

Algorithm:

1. Sample (randomly) the number of points required to fit the model ($n = 2$)
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model ($N_I = 6$)

Repeat 1-3 until the best model is found with high confidence

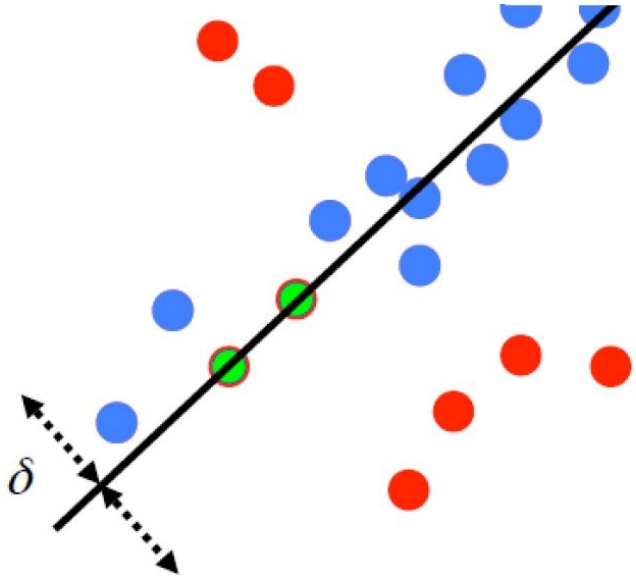


RANSAC (RANdom SAmple Consensus)

Algorithm:

1. Sample (randomly) the number of points required to fit the model ($n = 2$)
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model ($N_I = 6$)

Repeat 1-3 until the best model is found with high confidence



$N_I = 14$

RANSAC (RANdom SAmple Consensus)

Summary of RANSAC

- Select minimum number of random points from data needed to estimate the model
- Estimate the model from selected random points
- Check how many other points are consistent with the fitted model (Consistent Set)
  - If consistent set is large enough, estimate model from all points in the consistent set
  - If the consistent set is larger than the previous best model, make the current model as the best model
- Repeat a number of times

Quality of an Edge

Pros

- Robust to outliers
- Applicable for larger number of objective function parameters
- Optimization parameters are easier to choose than other approaches

Cons

- Computational time grows quickly with fraction of outliers and number of parameters
- Needs to be adapted further to allow multiple fits

Common applications

- Computing a homography(e.g., image stitching)
- Estimating fundamental matrix (relating two views)

Canny Edge Detector

- **Criterion 1:** Good Detection: The optimal detector must minimize the probability of false positives as well as false negatives.
- **Criterion 2:** 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.

Steps

1. Smooth image with Gaussian filter
2. Compute derivative of filtered image
3. Find magnitude and orientation of gradient
4. Apply "Non-maximum Suppression"
5. Apply "Hysteresis Threshold"

Step 1 and 2

1. **Smoothing**

$$s = I * g(x, y) = g(x, y) * I$$

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

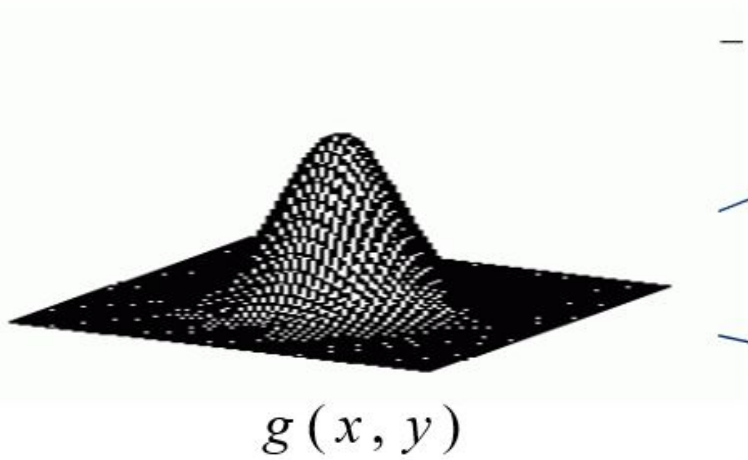
2. **Derivative**

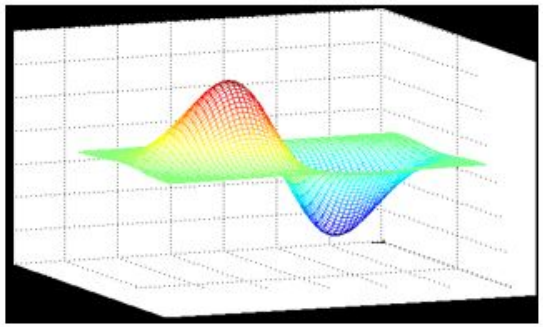
$$\nabla S = \nabla(g * I) = (\nabla g) * I$$

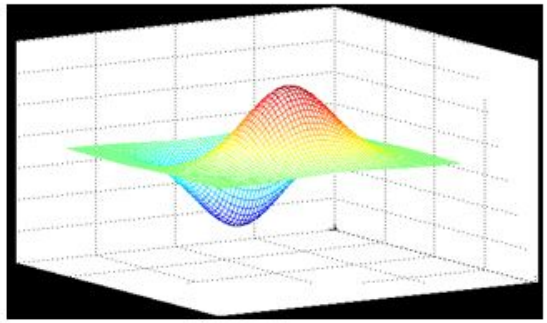
$$\text{where } \nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

$$\nabla S = \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix} = \begin{bmatrix} S_x \\ S_y \end{bmatrix}$$

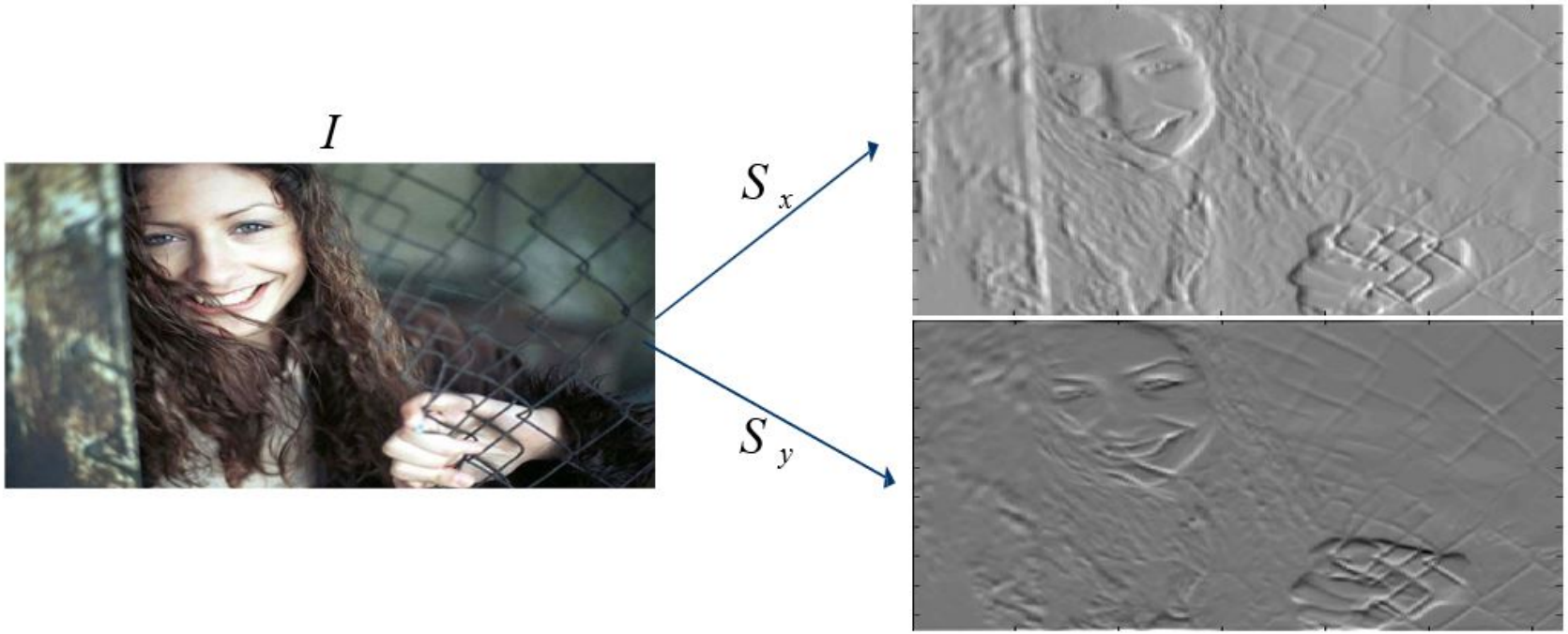
Canny Edge Detector Derivative of Gaussian







Canny Edge Detector



Step 3: Gradient magnitude and gradient direction

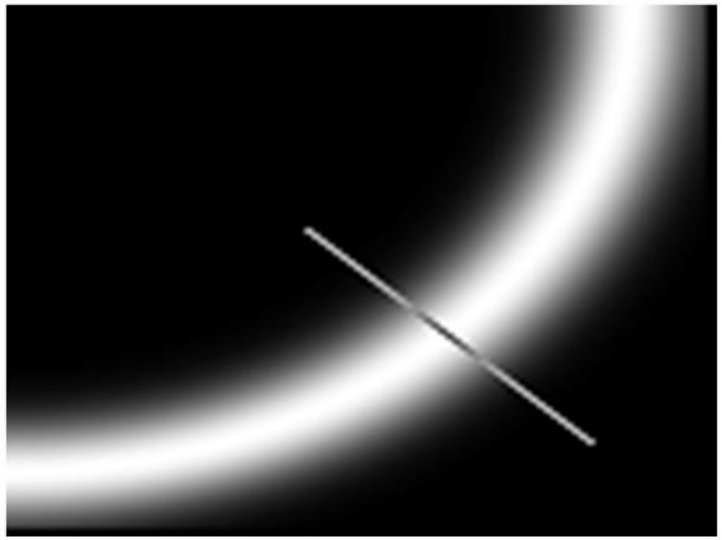
$$\text{Magnitude} = |\nabla S| = \sqrt{S_x^2 + S_y^2}$$

$$\text{Direction} = \theta = \tan^{-1} \frac{S_y}{S_x}$$

image

gradient magnitude

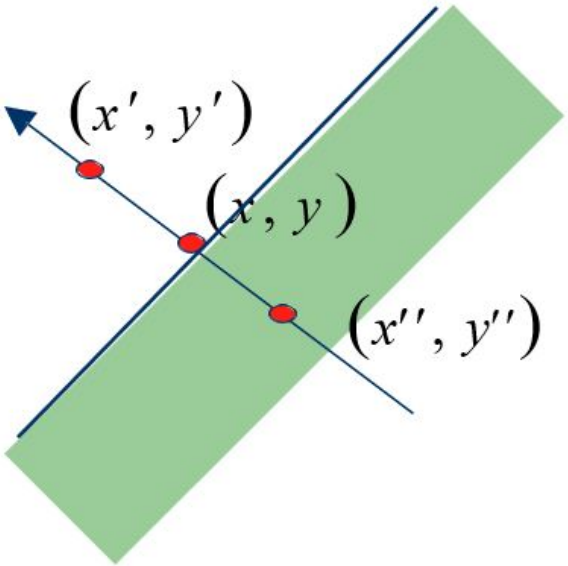
Step 4: Non maximum suppression



We wish to mark points along the curve where the magnitude is largest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

Canny Edge Detector

Suppress the pixels in $|\nabla S|$ which are not local maximum



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

x' and x'' are the neighbors of x along normal direction to an edge.

Canny Edge Detector

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

$$M(x, y)$$

For visualization

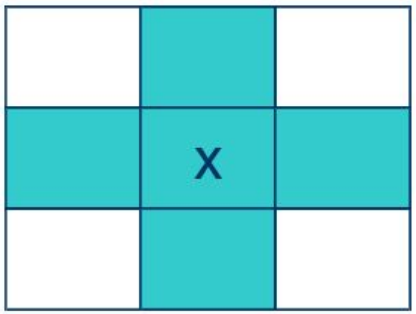
$$M \geq Threshold = 25$$

Step 5: Hysteresis Thresholding

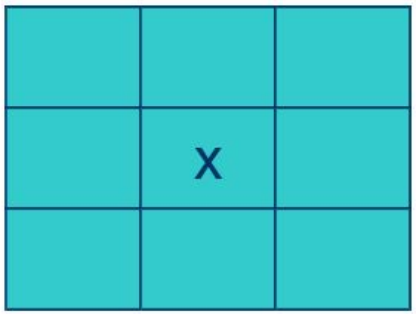
- If the gradient at a pixel is
  - above "**High**", declare it as an '**edge pixel**'
  - below "**Low**", declare it as a "**non-edge-pixel**"
  - **between** "Low" and "High"
    - Consider its neighbors iteratively then declare it an "edge pixel" if it is **connected** to an 'edge pixel' **directly** or via pixels **between** "Low" and "High".

Canny Edge Detector

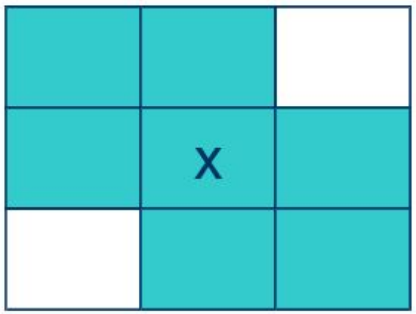
Connectedness:



4 connected

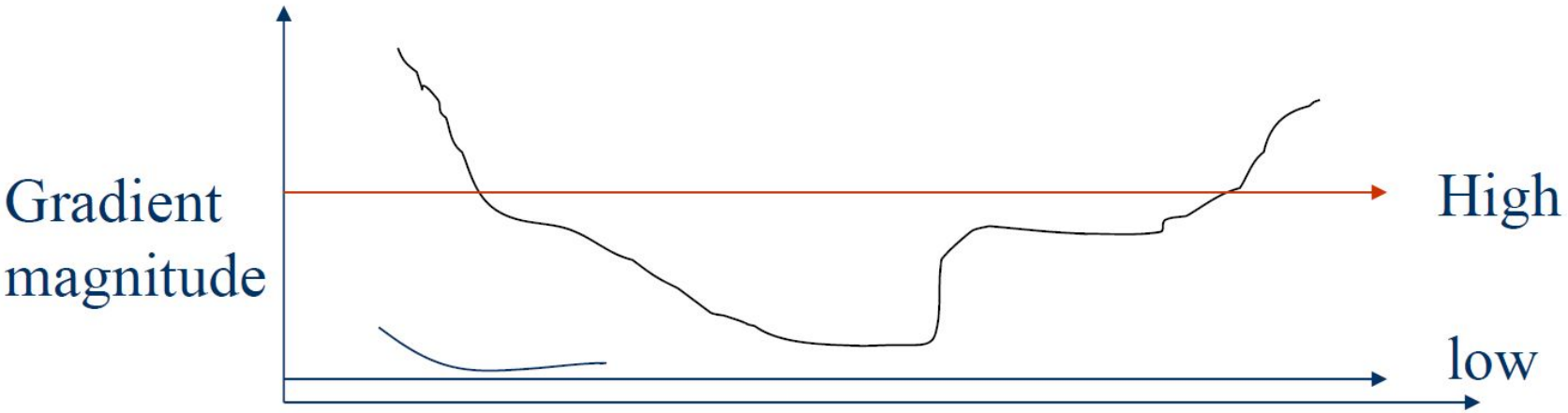


8 connected



6 connected

Canny Edge Detector



Canny Edge Detector

- Scan the image from left to right, top-bottom.
  - The gradient magnitude at a pixel is above a high threshold declare that as an edge point
  - Then recursively consider the neighbors of this pixel.
    - If the gradient magnitude is above the low threshold declare that as an edge pixel.

Canny Edge Detector

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

$$M(x, y)$$

$$M \geq 25$$

High = 35, Low = 15