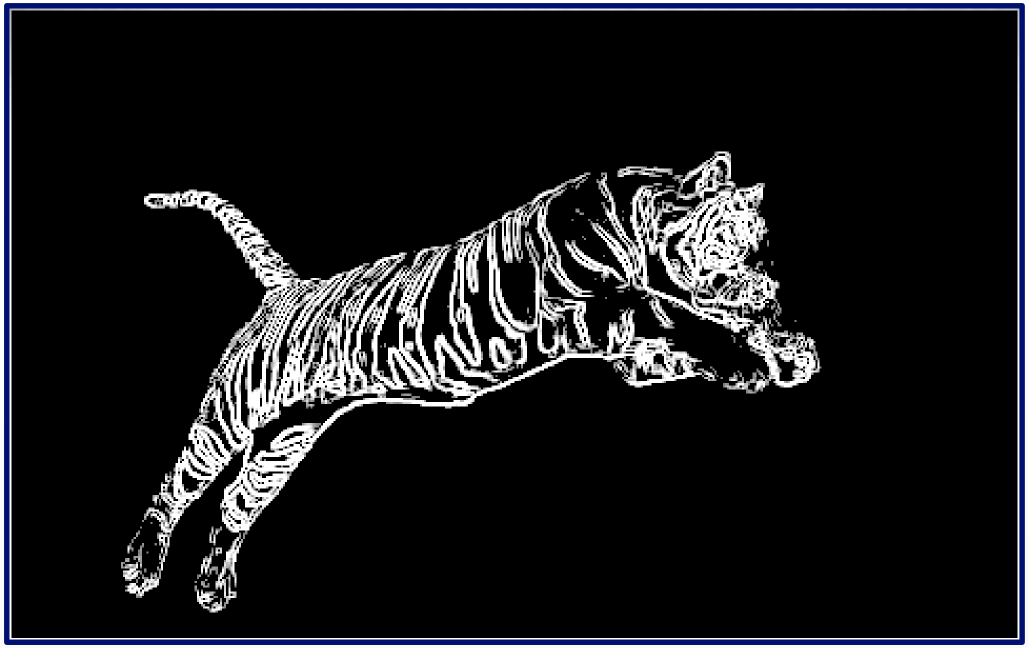
Basic Image Processing Algorithms

PPKE-ITK

Lecture 3.

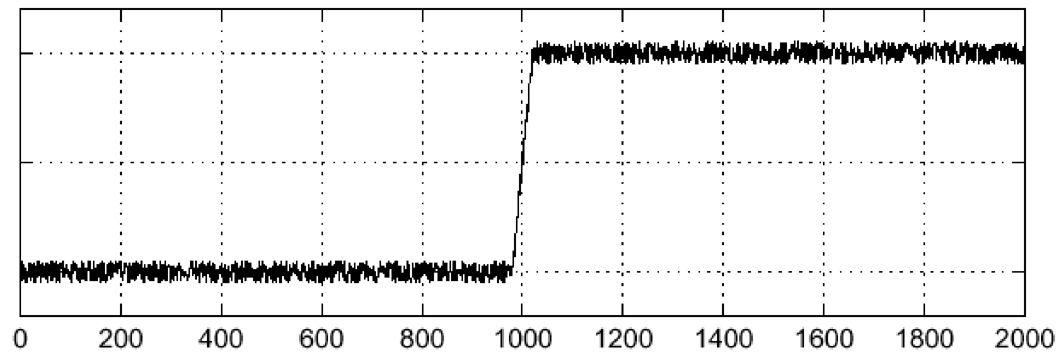
Recap: first/second order edge detection

- Noise filtering is required…

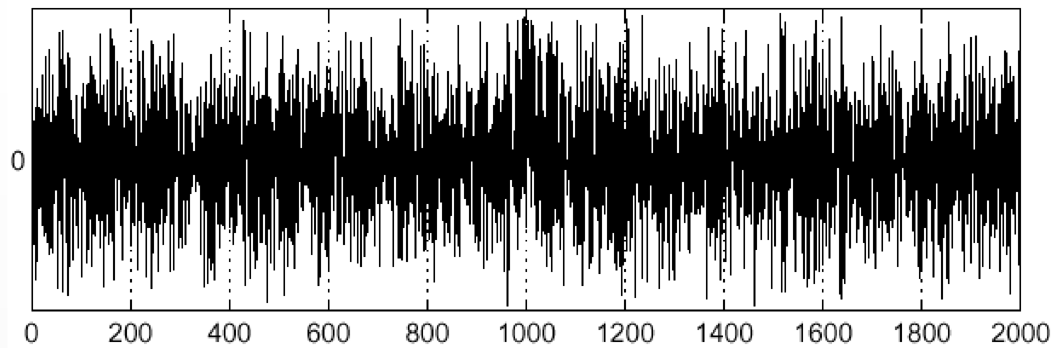


Noise filtering (1D demonstration)

$f(x)$



$\frac{d}{dx}f(x)$



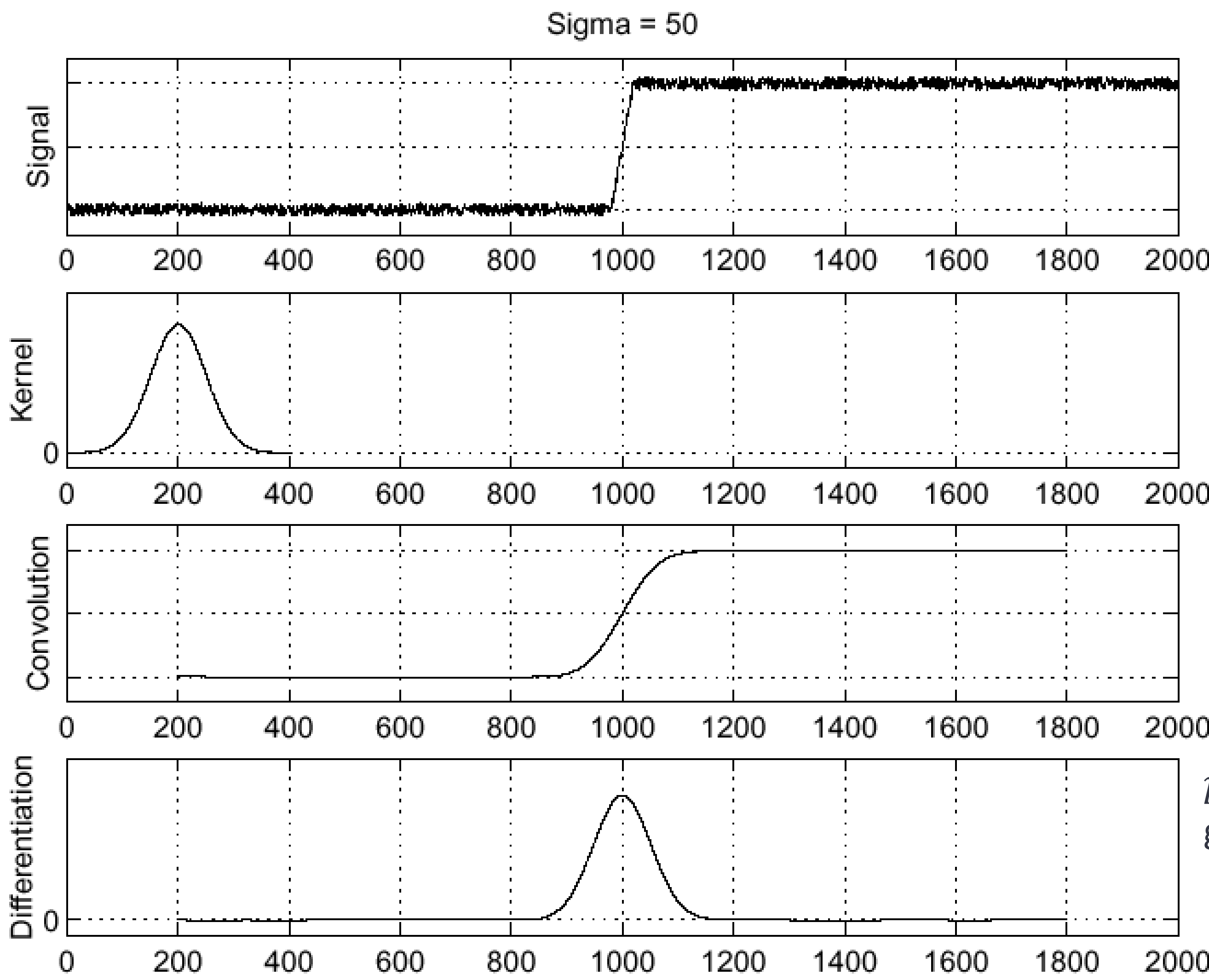
- Where is the edge?

f

h

$h * f$

$p * (h * f)$



f: original signal

h: Gaussian blur kernel

$h * f$: filtered signal

p: Prewitt (first order gradient kernel)

-1	0	1

Smoothing the signal with Gaussian kernel, followed by applying a first order Prewitt kernel

Associativity of convolution: $p * (h * f) = (p * h) * f$

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

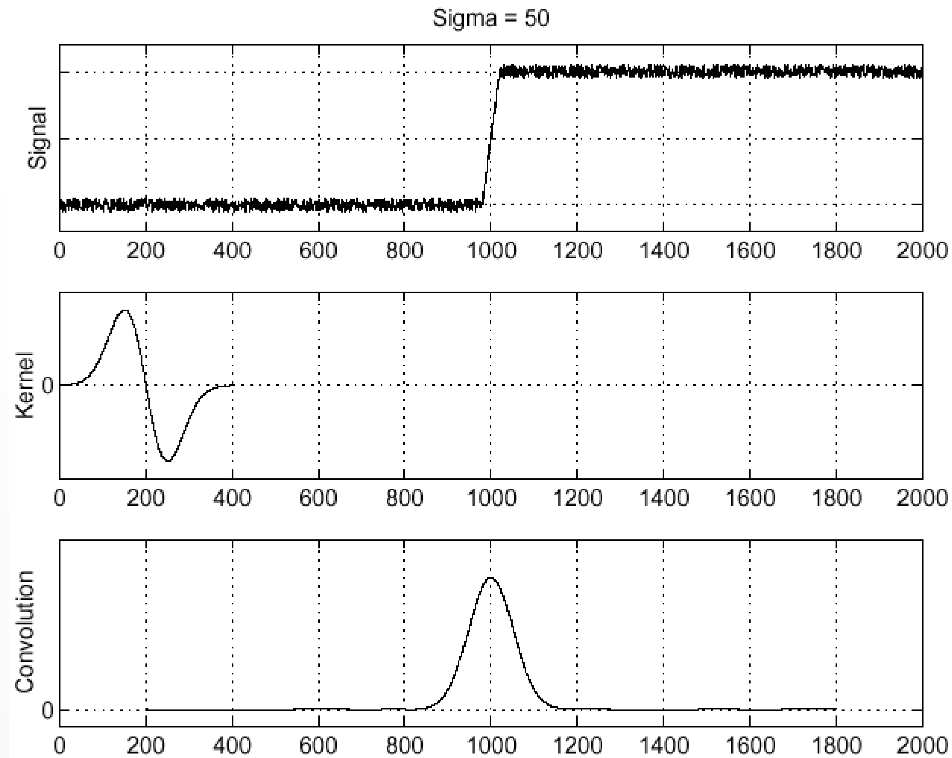
- No need for applying 2 convolutions, only one with the derivative of Gaussian operator (can also be approximated by a discrete kernel)

f

$\frac{\partial}{\partial x}h$ ≈ p*h

Derviative of the Gaussian kernel

$(\frac{\partial}{\partial x}h) \star f$



Second order case: Laplacian of Gaussian (LoG)

- Smoothing + Laplace = conv. with LoG operator

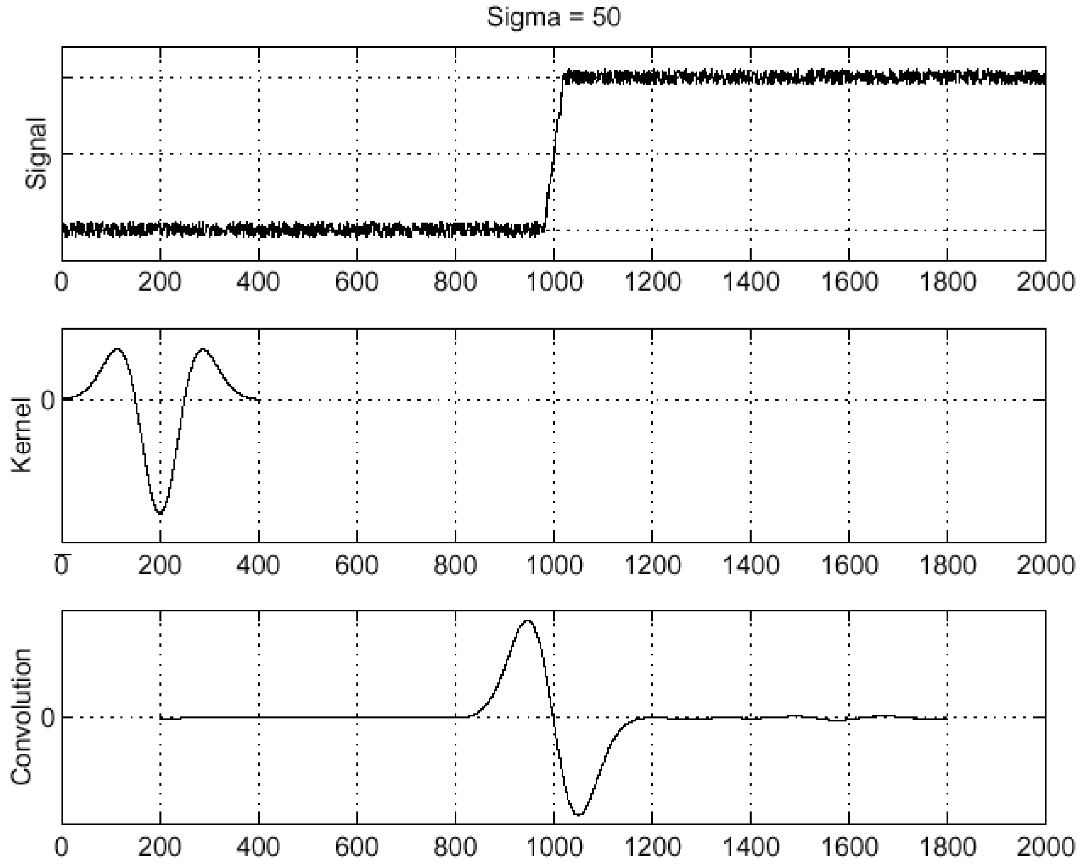
f

h: Gaussian smoothing kernel

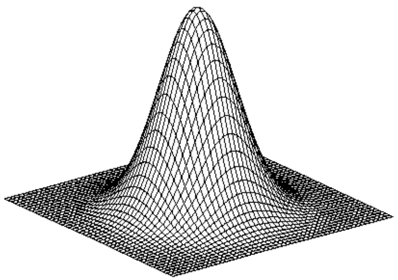
l: Laplace-kernel

$$\frac{\partial^2}{\partial x^2}h \approx l * h$$

$$\left(\frac{\partial^2}{\partial x^2}h\right) \star f$$

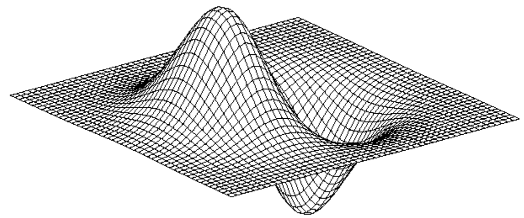


2D edge detection with filtering:



2D Gaussian smoothing kernel

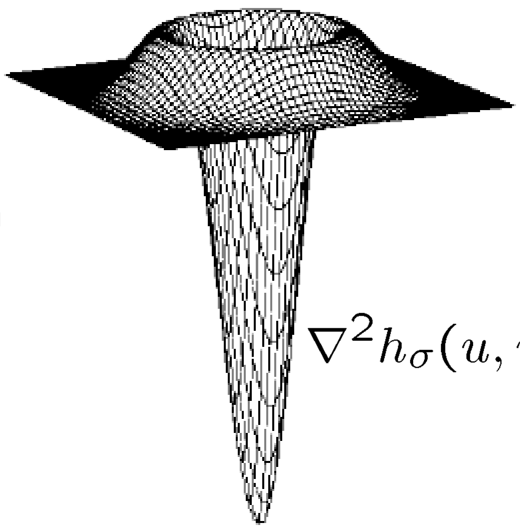
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



„Derivative” of the 2D Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

- ∇^2 henceforward the **Laplace** operator:

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

- Determining kernel coefficients with discrete approximation of the 2D function

Edge Enhancement

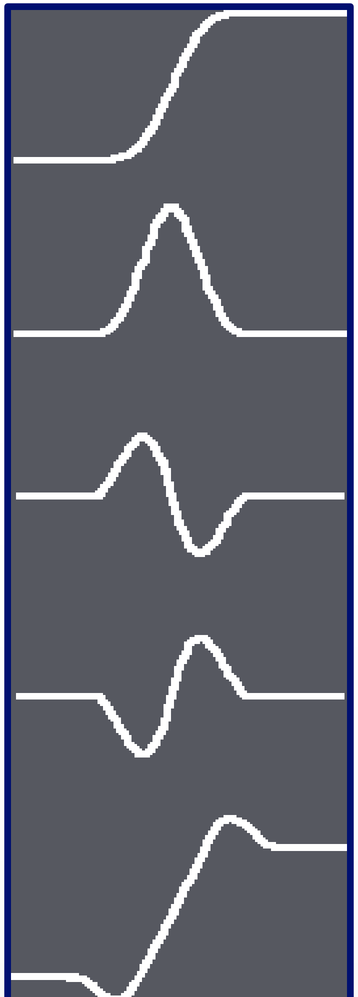
$f(x)$

$\frac{\partial f}{\partial x}$

$\frac{\partial^2 f}{\partial x^2}$

$-\frac{\partial^2 f}{\partial x^2}$

$f(x)-\frac{\partial^2 f}{\partial x^2}$



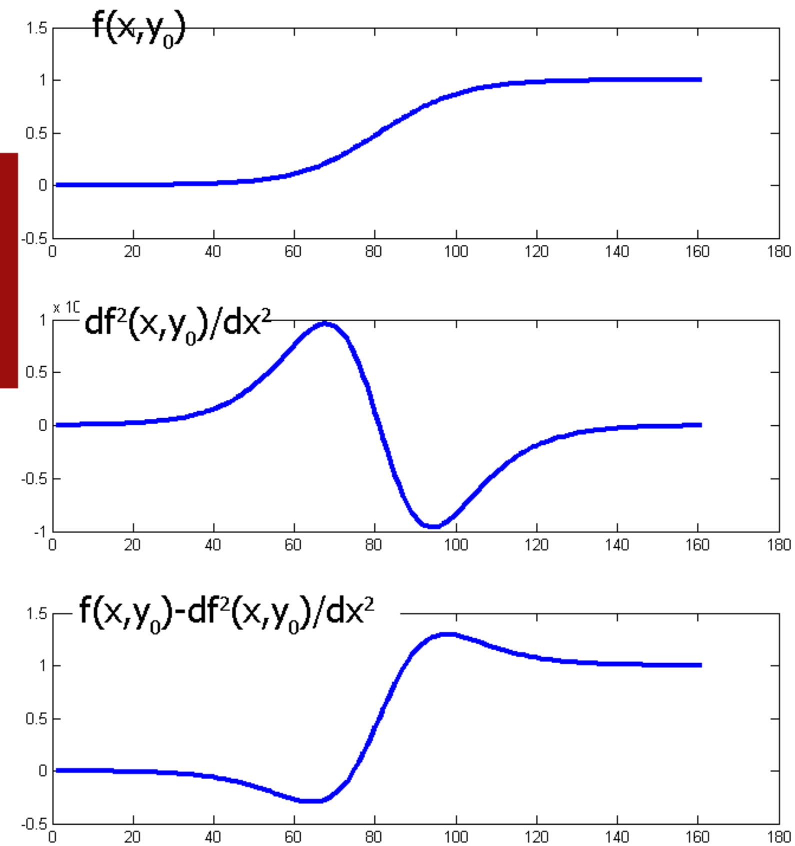
Kernel for edge enhancement with Laplace operator:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \boxed{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}}$$

Original image

Edge enhanced image

Edge crispening

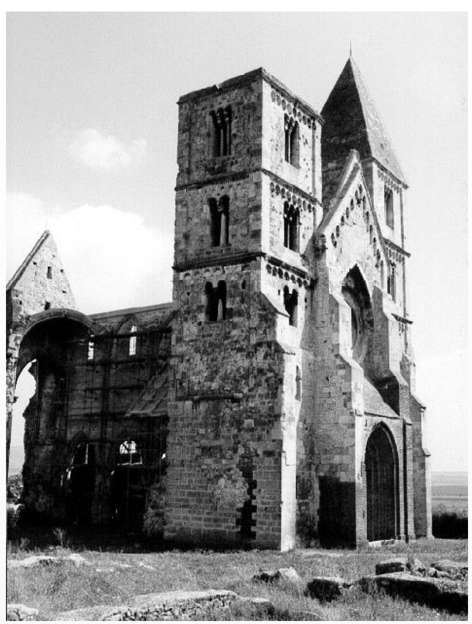


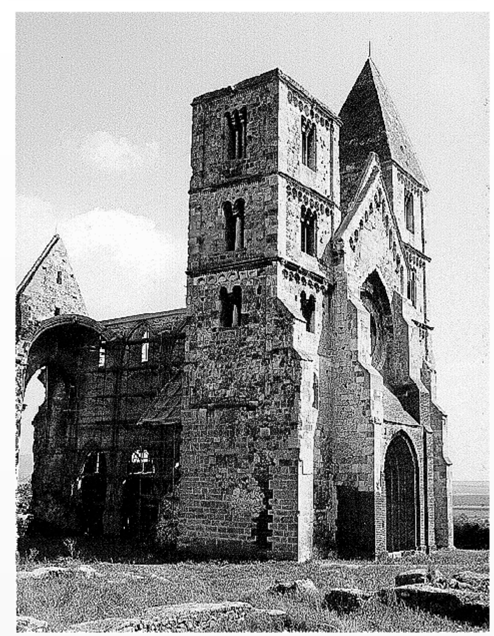
Convolution matrix:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$


Knauer
BRASSÓ


Knauer
BRASSÓ

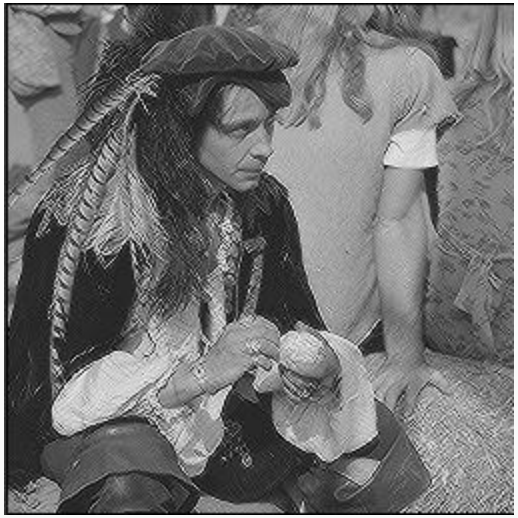




Edge crispening variants

- Often enhances the image qualiy

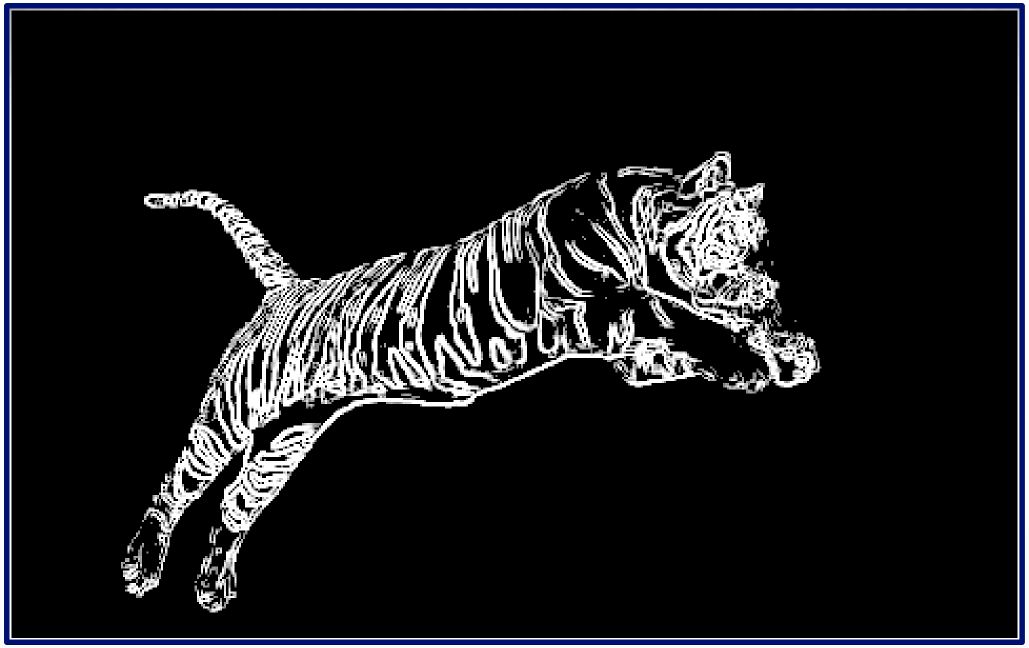
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

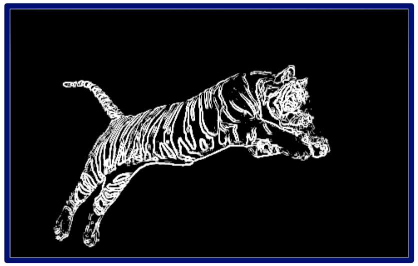


Canny edge detector

Evaluation of first/second order edge detection

- Prewitt kernel + threshold
  - is it a good edge detector?





Prewitt: is it a good edge detector?

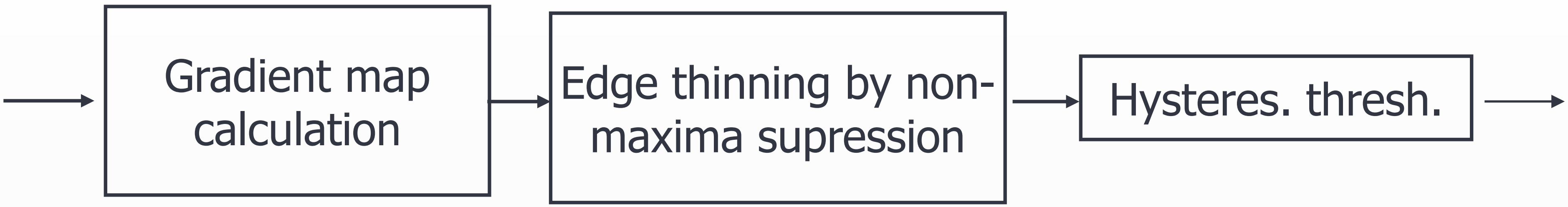
Canny edge detector

- Remember: properties of a good edge detector:
  - Good detection:
    - detects as many real edges as possible
    - does not create false edges
  - Good localization:
    - the detected edges should be as close to the real edges as possible
  - Isotropic:
    - all edges are detected regardless of their direction

- John F. ***Canny*** has developed an edge detector in 1986 to meet these requirements.

Canny edge detector

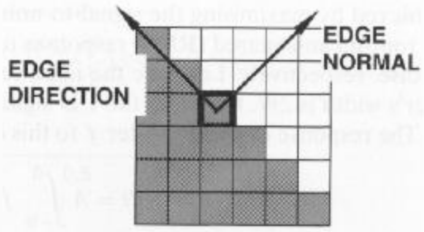
- Goal: extracting a **connected**, ***one-pixel-thick*** edge network
- Filtering Gaussian noise
- Three main steps:



Canny - 1st step: gradient map

- ***Noise reduction:***
  - The original image is convolved with a Gaussian kernel to reduce image noise.
- ***Gradient intensity and direction calculation:***
  - The horizontal and vertical derivative image is calculated (e.g. with Prewitt kernel)
  - At each pixel (i,j) calculate:
    - $d(i,j)$ gradient magnitude (how sharp is the edge – proportional to the gradient magnitude)

$$\|\nabla f\|_{ij} \propto d(i,j) = \sqrt{[d^x(i,j)]^2 + [d^y(i,j)]^2}$$

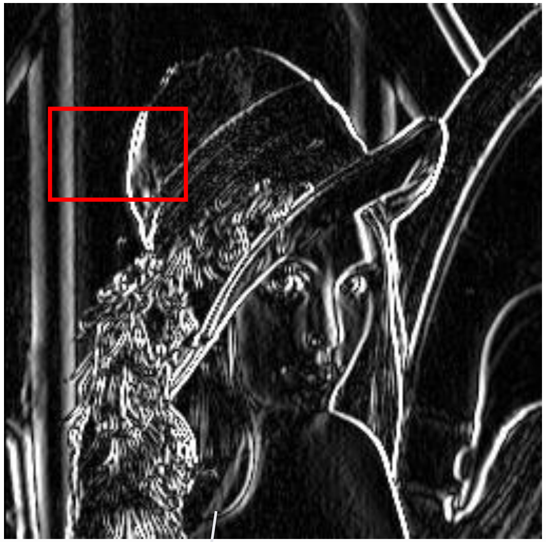


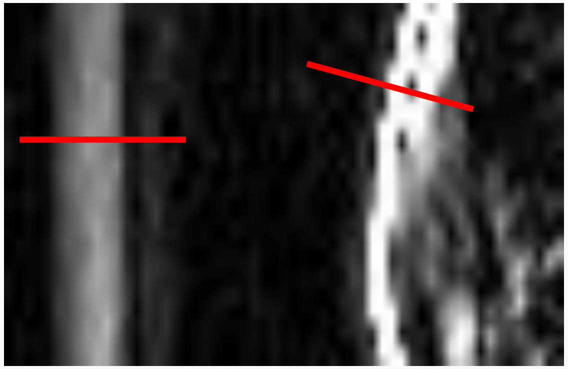
    - $n(i,j)$ edge normal (perpendicular to the direction)

$$n(i,j) = \arctan\left(\frac{d^x(i,j)}{d^y(i,j)}\right)$$

Canny – 2nd step: Non-max Suppression

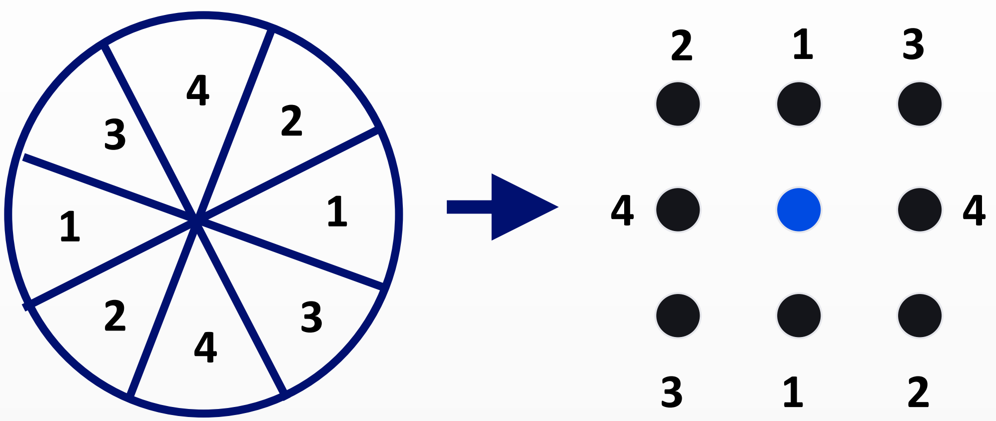
- Goal: thinning the edges
- The gradient map may contain „thick” regions with large gradient values. Earlier methods may classify all of these points as edges.
- Along the highlighted line segments perpendicular to the edges (marked with red) we should only mark a single point as edge point, the one which is **<u>locally the brightest</u>**





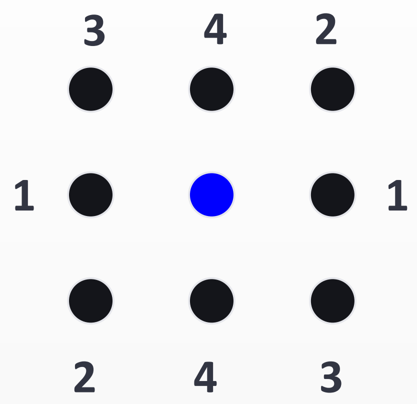
Canny Edge detector

3. ***Non-Maximum Suppression step for edge thinning:***
   - Each edge is categorized into one of *4 main edge directions* (0°, 45°, 90°, 135°), based on the gradient direction image (θ).
   - At every pixel, it suppresses the edge, by setting its value to 0, if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction:



Canny – 2nd step: Non-max Suppression

1. Each edge is categorized into one of 4 main edge directions (0°, 45°, 90°, 135°), based on the gradient direction image
   - to each pixel (i, j) we assign the principal direction $a(i, j)$, which one is the closest to local edge normal $n(i, j)$
2. At every pixel, it suppresses the edge, if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction:
   - If local edge magnitude $d(i, j)$ is smaller than in any neighboring pixel in the $a(i, j)$ direction set $G(i, j) := 0$. Otherwise (local max) : $G(i, j) := d(i, j)$
3. Result: G image obtained from the *d* gradient-magnitude map, where the edge-candidate regions become thin



Canny – 3rd step: Thresholding

- Naive solution: thresholding the G map with a threshold t
  - If t is too small, we obtain many false edge points. If t is too large: valid edges disappear.
  - If the gradient magnitude of the edges fluctuates around the threshold, many disruptions (broken edge segments) may appear
- Improved solution: hysteresis thresholding
  - Using 2 thresholds t_1 and t_2 ($t_1 < t_2$):
    - If the value of $G(x, y)$ is larger than t_2, (x, y) is certainly edge point
    - If the value of $G(x, y)$ is smaller than t_1, (x, y) is certainly *not an* edge point
    - If the value of $G(x, y)$ is between the threshold, we mark it as edge point if and only if it has a neighboring pixel already classified as edge in the direction perpendicular to the **edge normal**

Canny edge detector - result

Input image

Canny edge detector - result

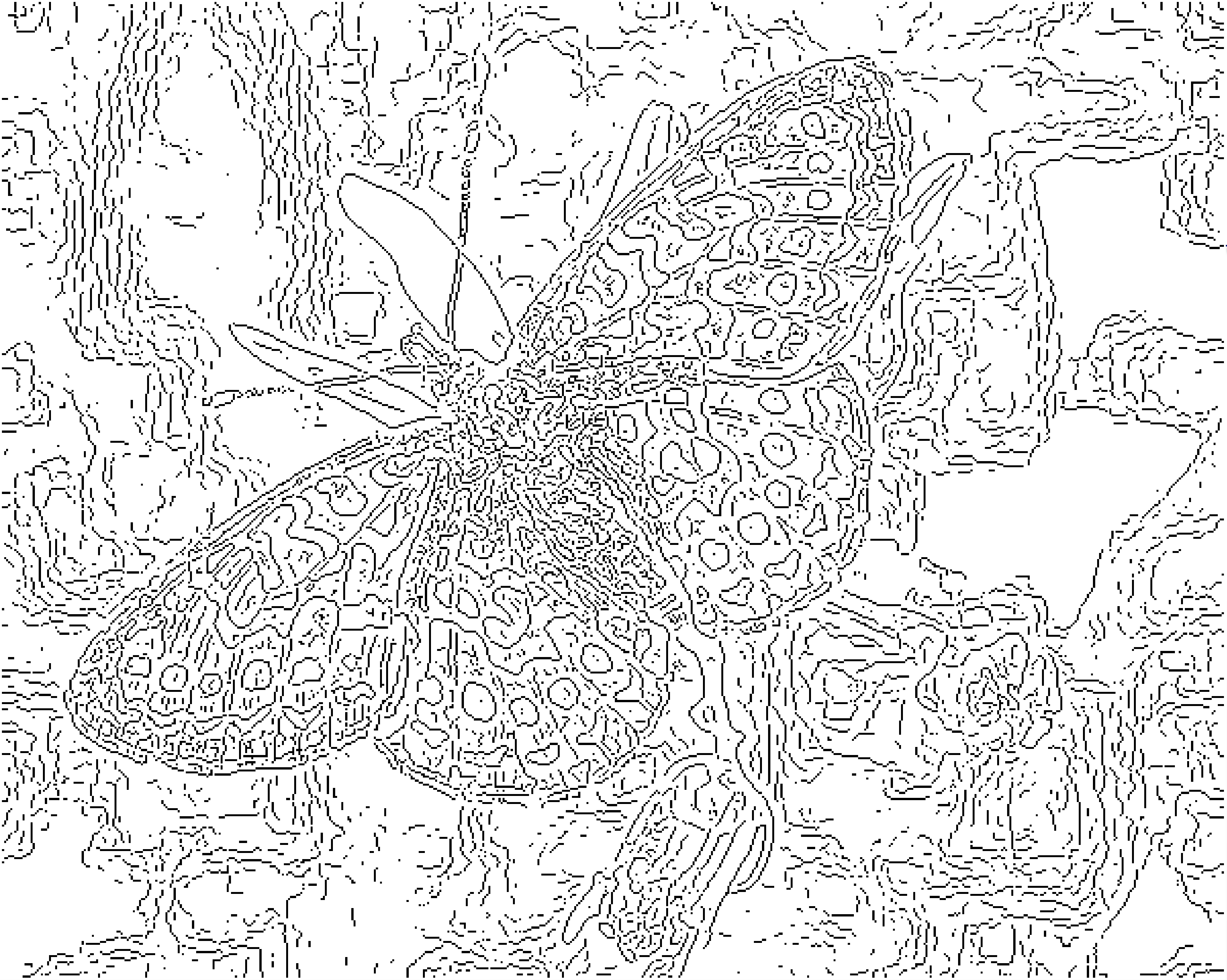
Norm of gradient: „d”

Canny edge detector - result

After thinning (E) (non-maximum suppression)

Canny edge detector - result

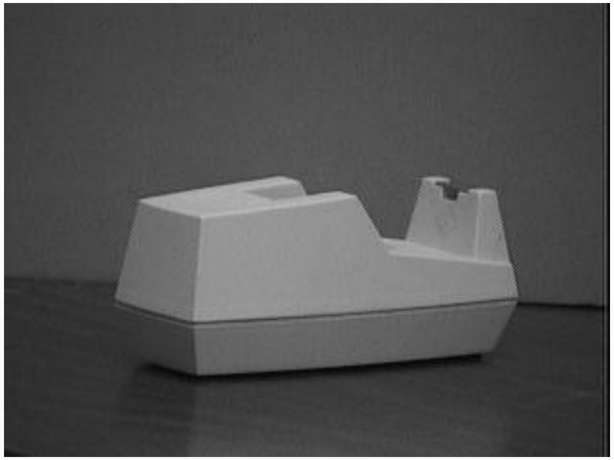
hysteresis thresholding

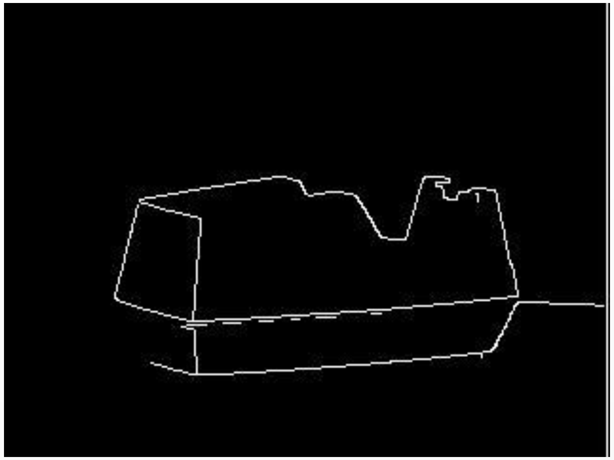


fine scale
high
threshold

coarse scale, high threshold

Canny edge detector - results





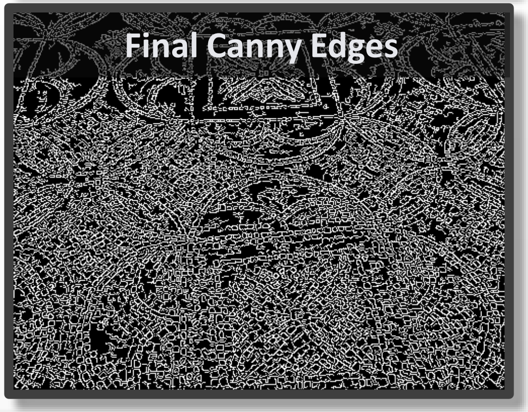

Original Image

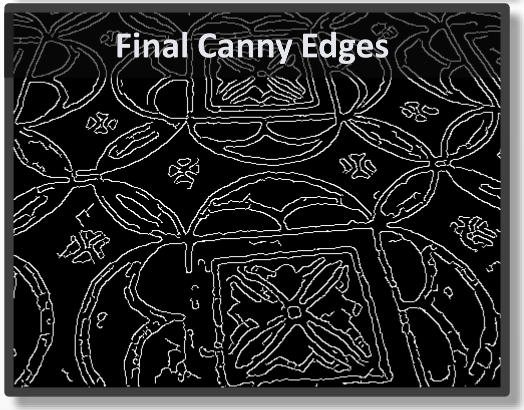

Weakly Blurred Image


Moderately Blurred Image


Gradient Magnitude


Gradient Magnitude

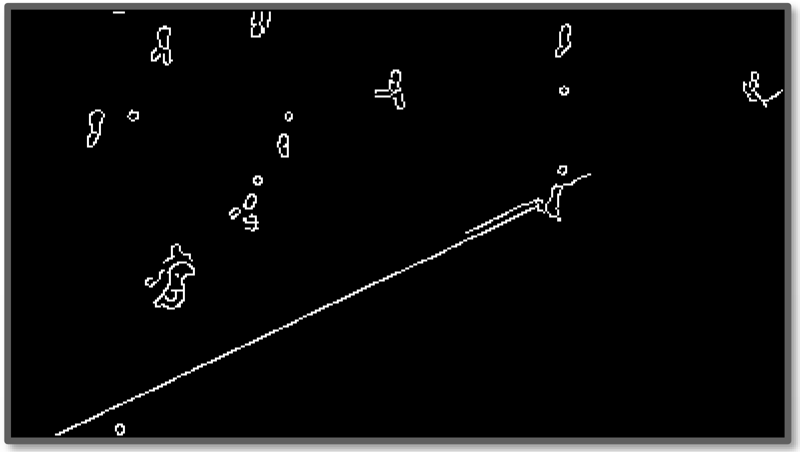

Final Canny Edges


Final Canny Edges

Line detection with Hough Transfrom

Hough Transformation

- An example of Canny edge detector…



- …where straight lines are not detected perfectly.
- The objective of the Hough transformation is to find the lines on a binary image, from fragments/points of the line.

Finding lines in an image

- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?
- Option 2:
  - Use a voting scheme: Hough transform

Finding lines in an image

- The basic idea:
  - A line can be written in the following form:

$$y = mx + b$$

where ***m*** is the slope of the line and ***b*** is the y-intercept.

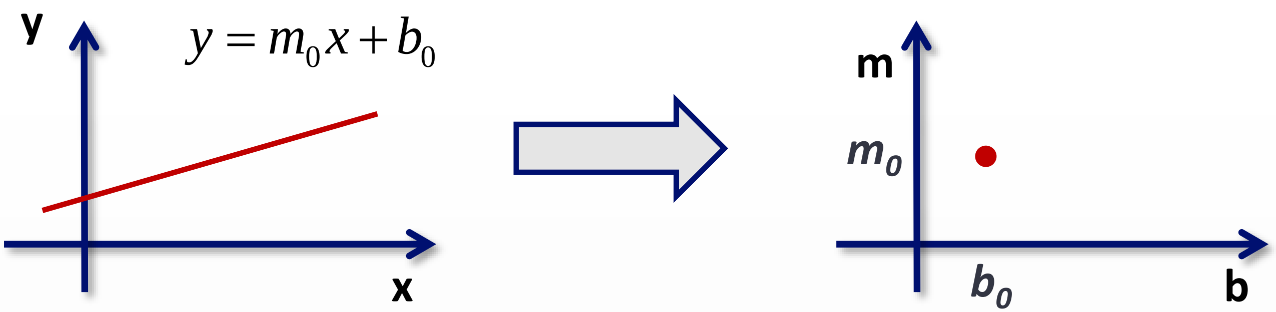
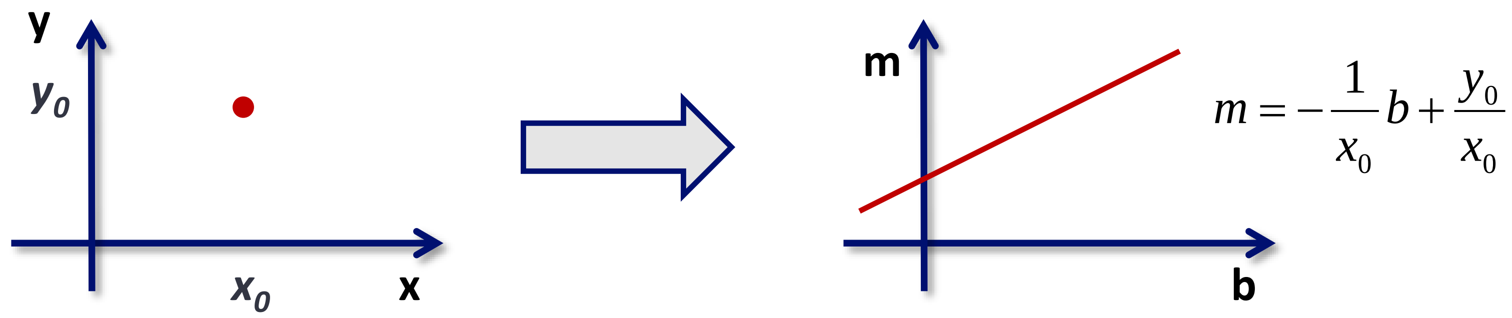


image space

m-b space of lines

- Connection between image ***(x,y)*** and the ***(m,b)*** spaces
  - A line in the image corresponds to a point in "m-b" space
  - To go from image space to (m-b) space:
    - given a set of points ***(x,y)***, find all ***(m,b)*** such that ***y = mx + b***

Finding lines in an image

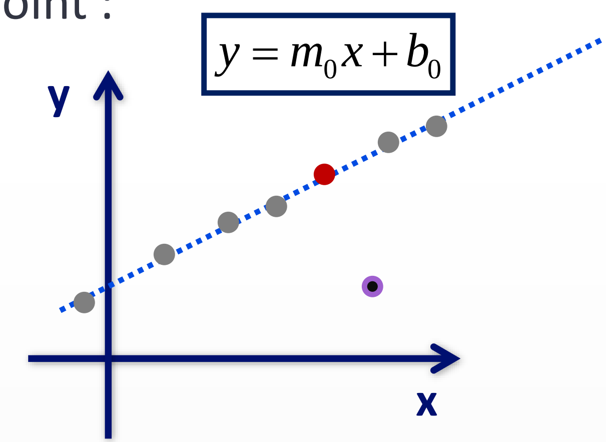


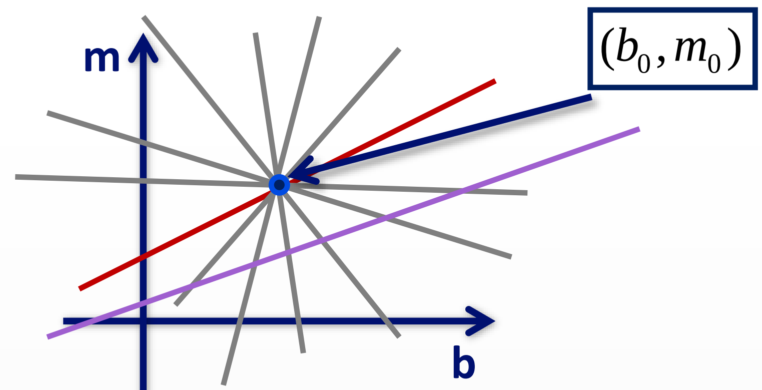
- Connection between image ***(x,y)*** and ***(m,b)*** spaces
  - What does a point ***(x_0, y_0)*** in the image space map to?
    - For a fixed ***$y = y_0$, $x = x_0$*** point in the image space, we get a line in the (m, b) space with a slope ***$-1/x_0$*** and an m-intercept: ***y_0/x_0***:

$$m = -\frac{1}{x_0} b + \frac{y_0}{x_0}$$

Hough Transformation

- The basic idea:
  - For the points that lie on the same line in the Euclidian space, their corresponding line in the parameter space will cross each other in one point :

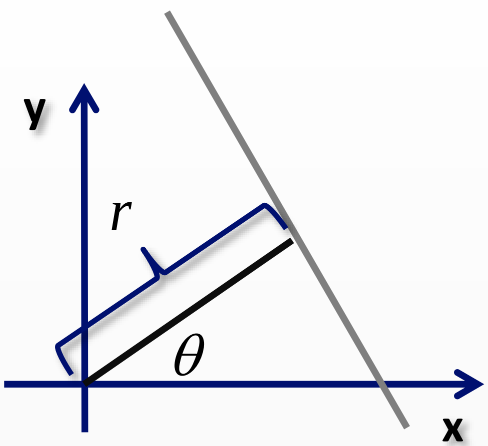




  - This point will be ***m=m₀*** and ***b=b₀***, the slope and intercept of the line in the image space. ⇨ We have the equation of the line!
- But, there is a problem with this equation of the line: vertical lines cannot be described (their slope would be infinite).

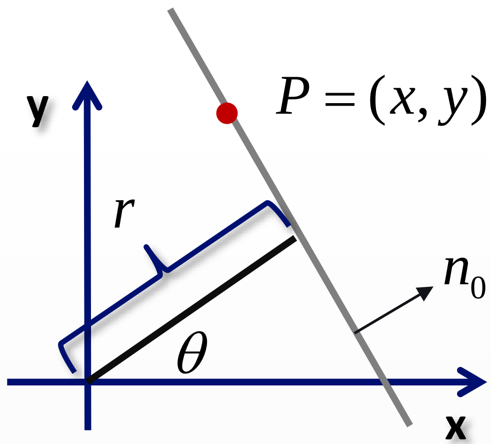
Hough Transformation

- To be able to describe all possible lines with two scalar parameters, we will use a **polar representation** of the line
- Each line is described by (r, θ) instead of (m, b), where
  - r is the perpendicular distance from the line to the origin
  - θ is the angle this perpendicular makes with the x axis



Hough Transformation

- Mathematical basis for using the **polar equation** of the line is the Hesse normal form*: $0 = P \cdot n_0 - r$



$P = (x, y)$ arbitrary point of the line

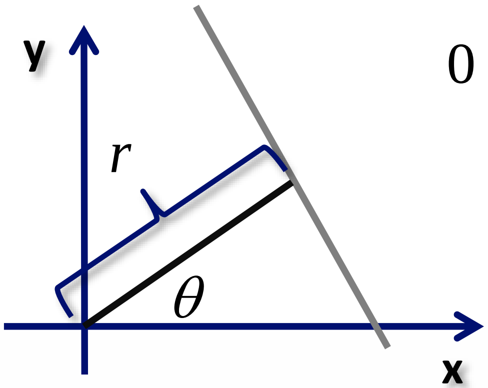
$n_0 = (\cos\theta, \sin\theta)$ normal vector of the line

- *r:* perpendicular distance from the line to the origin
- θ: the angle this perpendicular makes with the x axis

* https://en.wikipedia.org/wiki/Hesse_normal_form

Hough Transformation

- Hesse normal form based polar equation of the line:



$$0 = P \cdot n_0 - r = (x, y) \cdot (\cos\theta, \sin\theta) - r$$

$$\Downarrow$$

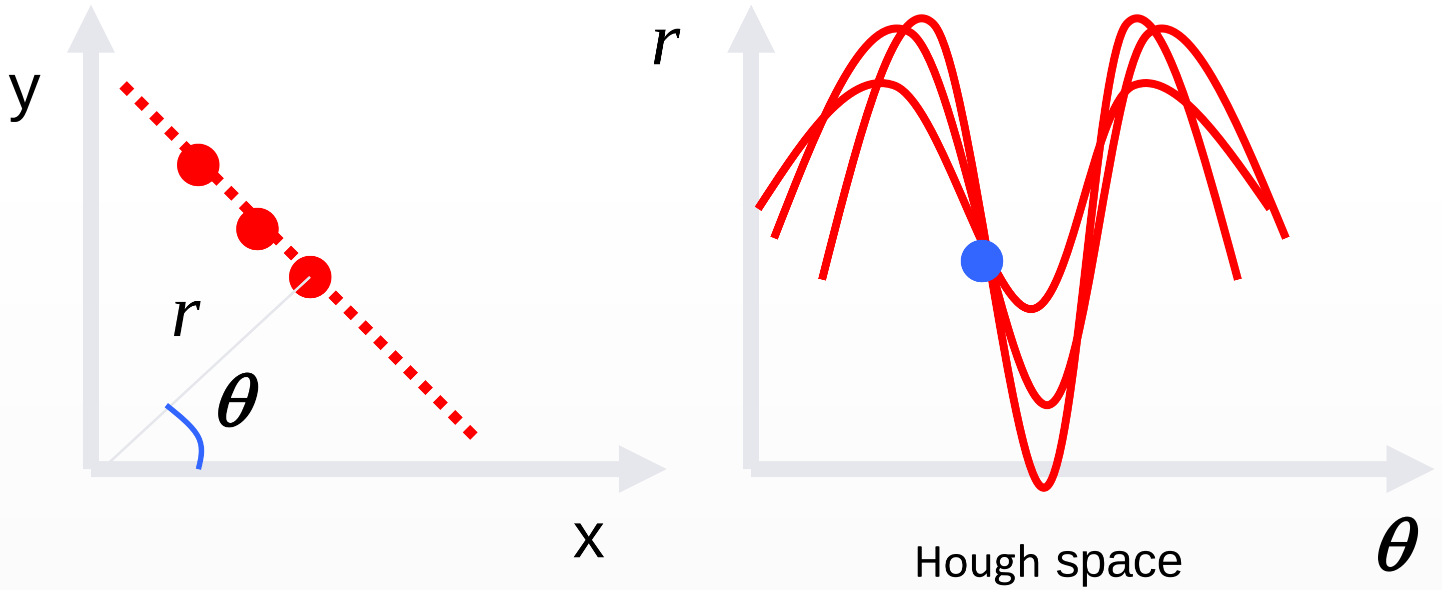
$$r = x \cdot \cos\theta + y \cdot \sin\theta$$

- The (r, θ) parameter space is called Hough space.
- A point in the Euclidian space is a sinusoid in the Hough space, described by the following equation:

$$r(\theta) = x \cdot \cos\theta + y \cdot \sin\theta$$

Hough Transformation

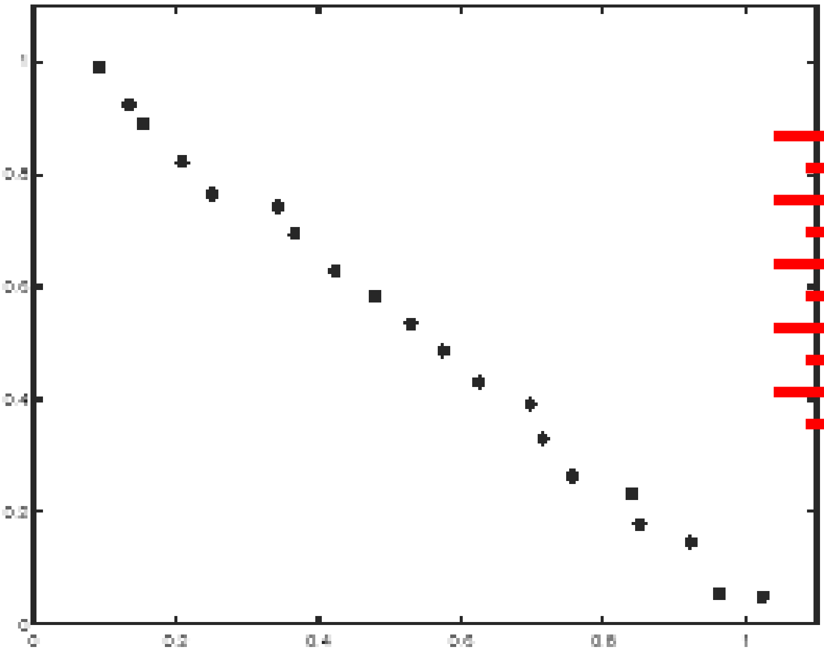
- All the sinusoid curves of the points in one line in the Euclidian space, cross each other in one point in the Hough space.



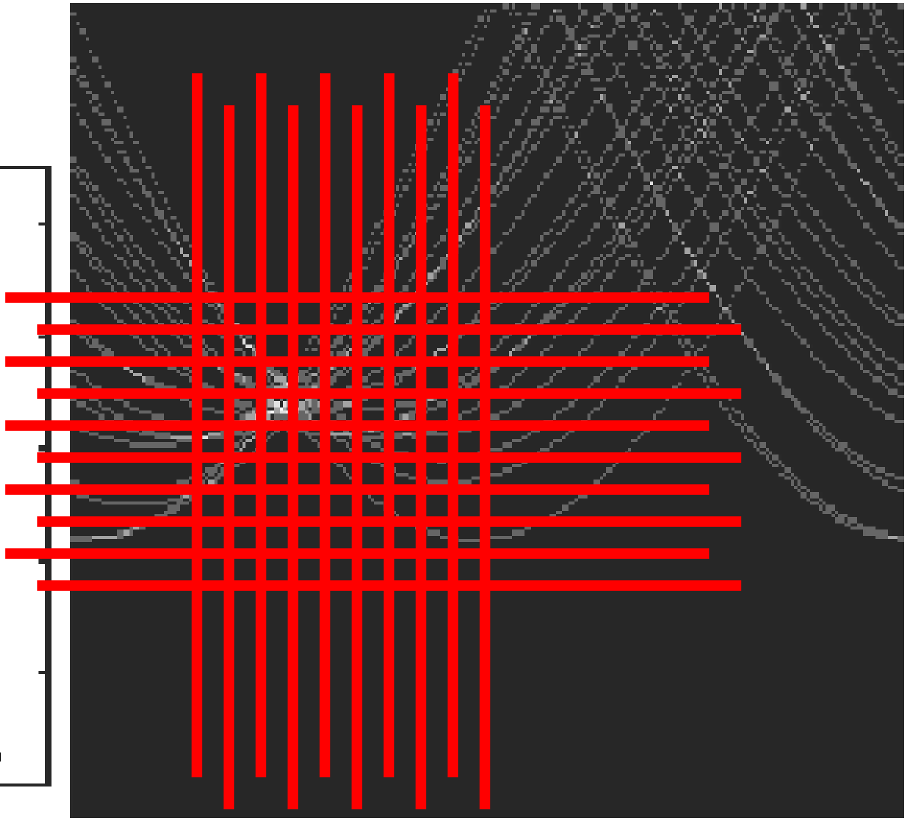
$$r(\theta) = x \cdot \cos\theta + y \cdot \sin\theta$$

Hough Transformation

Noisy data



features



votes

Issue: Grid size needs to be adjusted…

Hough transform algorithm

- Basic Hough transform algorithm
  1. for all r, θ: initialize H[r, θ]=0
  2. for each edge point I[x,y] in the image

     for θ = 0 to 180

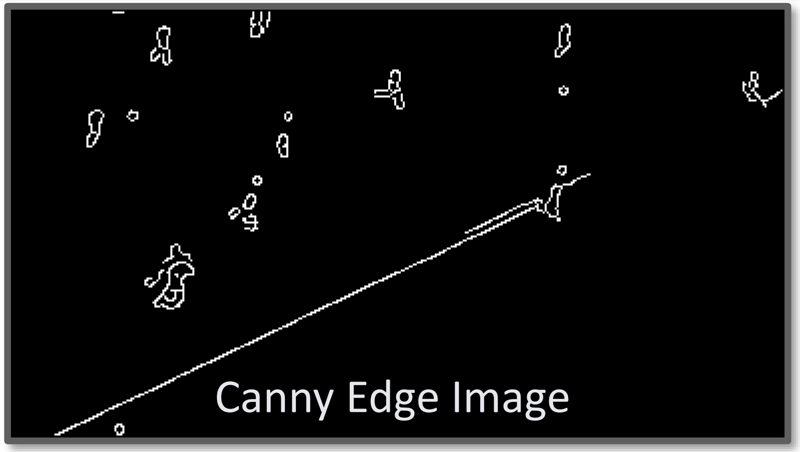
     $$r = x \cdot \cos\theta + y \cdot \sin\theta$$

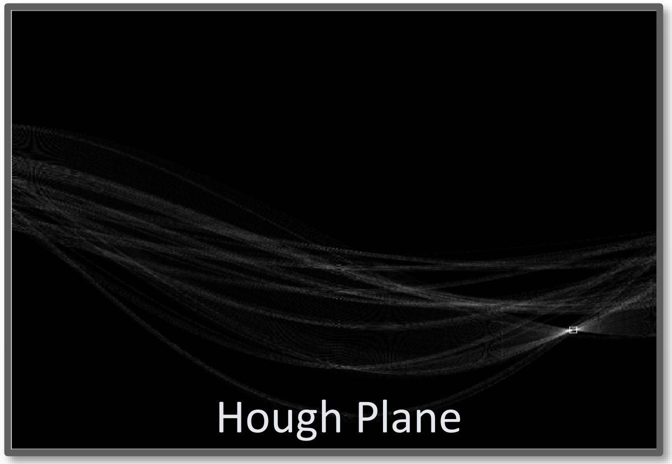
     H[r, θ] += 1
  3. Find the value(s) of (r, θ) where H[r, θ] is maximum
  4. The detected line in the image is given by

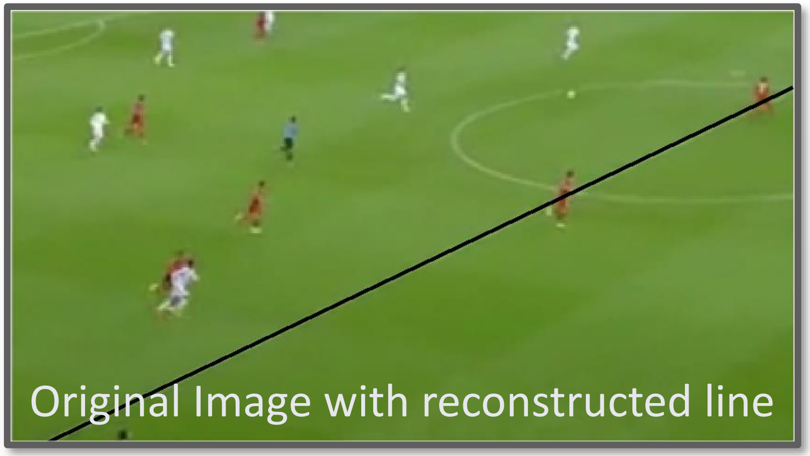
     $$r = x \cdot \cos\theta + y \cdot \sin\theta$$

- What's the running time (measured in # votes)?

Hough Transformation







Extensions

- Extension 1: Use the image gradient
  1. same
  2. for each edge point I[x,y] in the image

     compute unique (r, θ) based on local image gradient at (x,y)

     $H[r, \theta] \mathrel{+}= 1$
  3. same
  4. same

- Extension 2
  - give more votes for stronger edges

- Extension 3
  - change the sampling of (r, θ) to give more/less resolution

- Extension 4
  - The same procedure can be used with circles, squares, or any other shape

Hough demos

- Lines, circles and ellipses: http://dersmon.github.io/HoughTransformationDemo/
- Circle : http://www.markschulze.net/java/hough/

Image Enhancement

What is Image Enhancement?

- **Image enhancement** is the manipulation or transformation of the image to improve the visual appearance or to help further automatic processing steps.
- There is no general theory behind it, the result is highly application dependent and subjective.
  - e.g. in many cases the goal is to improve the quality for human viewing (Medical Imaging, Satellite Images)
- Enhancement is closely related to image recovery.
- Examples:
  - Contrast enhancement
  - Edge enhancement
  - Noise removal/smoothing

Types of Image Enhancement

- There are two main categories:
  - Spatial Domain Methods
  - Frequency Domain Methods
- In the Spatial Domain we are directly manipulating pixel values, through..
  - Point-wise Intensity Transformation
  - Histogram Transformations
  - Spatial Filtering
    - LSI (*Linear Shift-Invariant*)
    - Non-Linear
  - etc.

The Histogram of an Image

- **Histogram**:
  h(k) = the number of pixels on the image with value *k.*

Original Image*

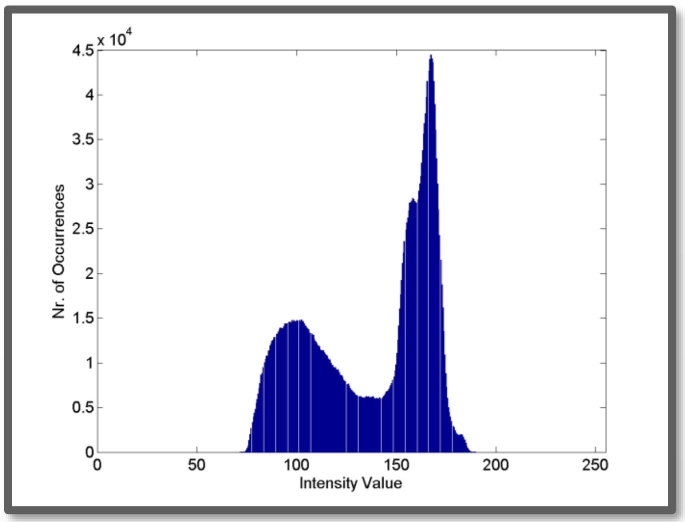


Image Histogram

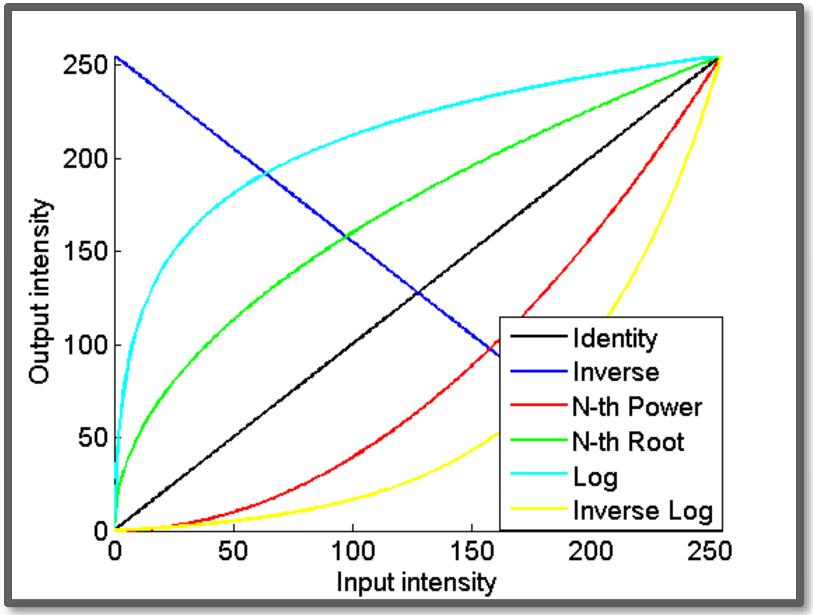
- The histogram normalized with the total number of pixels gives us the ***probability density function*** of the intensity values.

* Modified version of Riverscape with Ferry by Salomon van Ruysdael (1639)

Point-wise Intensity Transformation

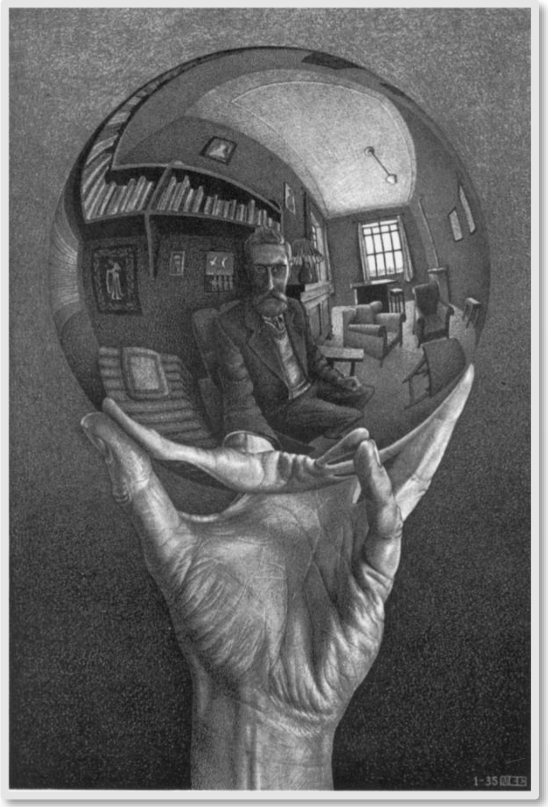
- Point wise transformations are operating directly on pixel values, independently of the values of its neighboring pixels.
- We can describe the transformation as follows:
    - Let *x* and *y* be two grayscale images, and let *T* be a point-wise image enhancement transformation that transforms *x* to *y*:

$$y(n_1, n_2) = T\left[x(n_1, n_2)\right]$$

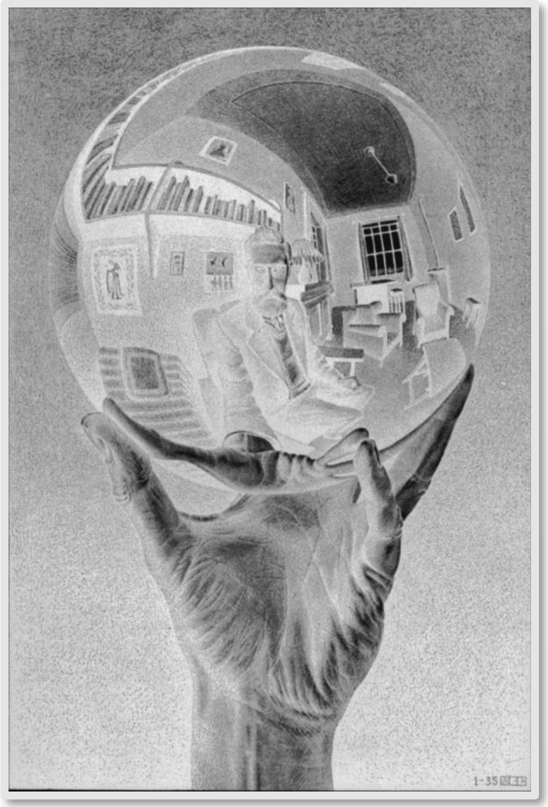


Point-wise Intensity Transformation

- **Inverse transformation**: $y(n_1, n_2) = 255 - x(n_1, n_2)$



Original Image*



Inverse Image

*Hand with Reflecting Sphere by M. S. Escher (1935)

Point-wise Intensity Transformation

- **Log transformation**: $y(n_1, n_2) = c \cdot \log\left(x(n_1, n_2) + 1\right)$
  - Expands low and compresses high pixel value range

Original Image*

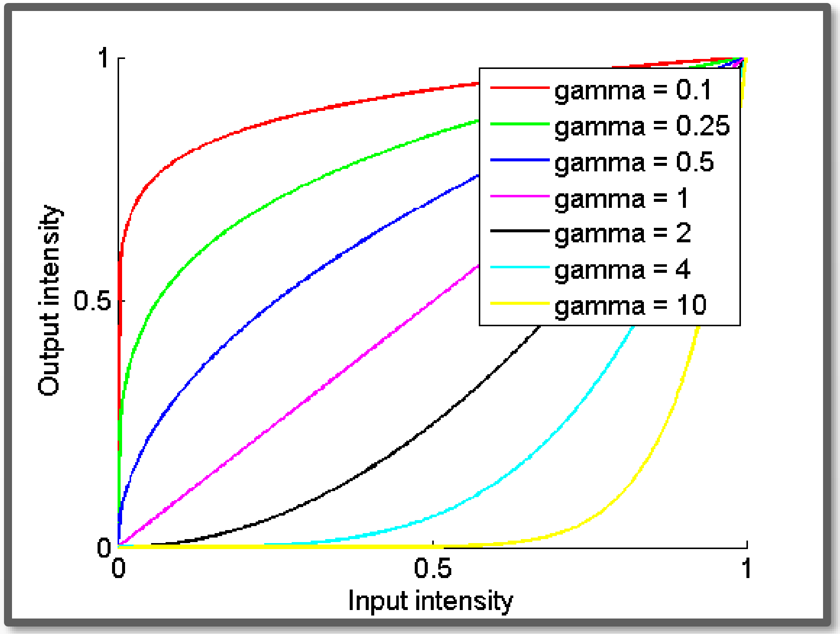
Log Image

Log Image
after histogram stretching

* Abbaye du Thoronet by Lucien Hervé (1951)

Point-wise Intensity Transformation

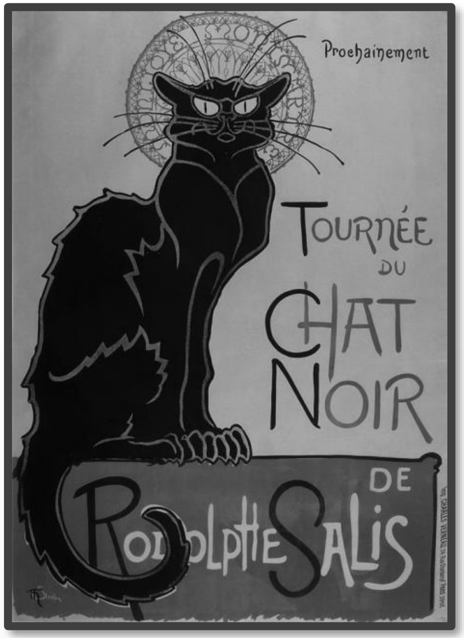
- **Power-law transformation**: $y(n_1, n_2) = c \cdot x(n_1, n_2)^{\gamma}$
  - Commonly referred to as ***gamma transformation***
  - Originally it was developed to compensate the input-output characteristics of CRT displays.
  - The expended/compressed region depends on γ:



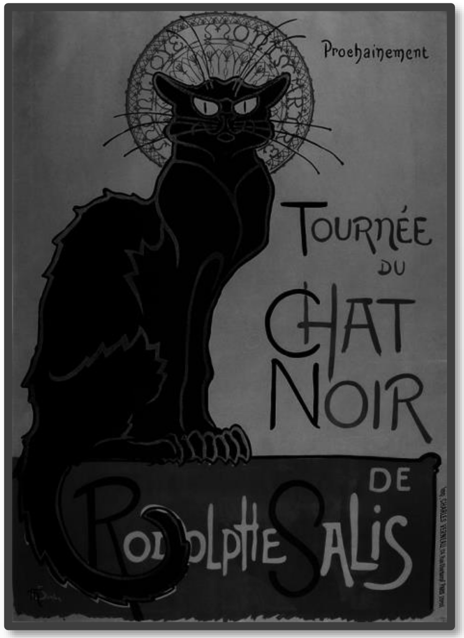
Here *c = 1*

Point-wise Intensity Transformation

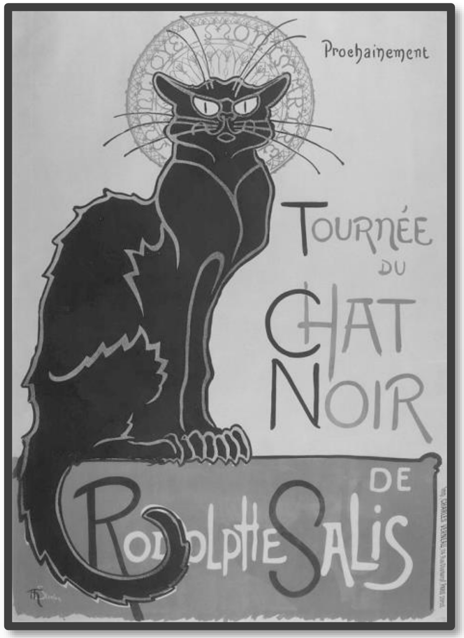
- **Power-law transformation**: $y(n_1, n_2) = c \cdot x(n_1, n_2)^{\gamma}$



Original Image*



γ = 2

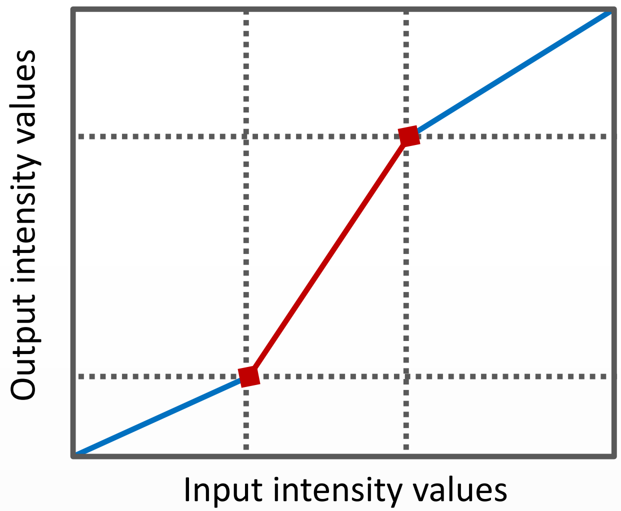


γ = 0.5

* Le chat Noir, Poster of Théophile Steinlen (1896)

Dynamic Range Expansion

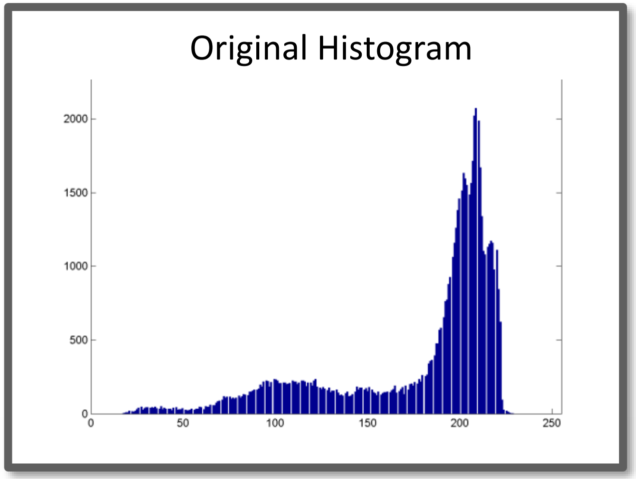
- Piecewise linear expansion/compression of predefined intensity ranges:



  - The red intensity range was expanded, while the blue ranges were compressed.

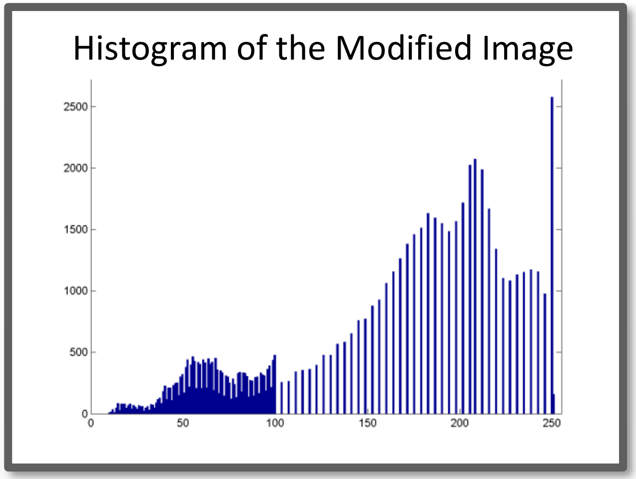
Dynamic Range Expansion

Original Image



Original Histogram

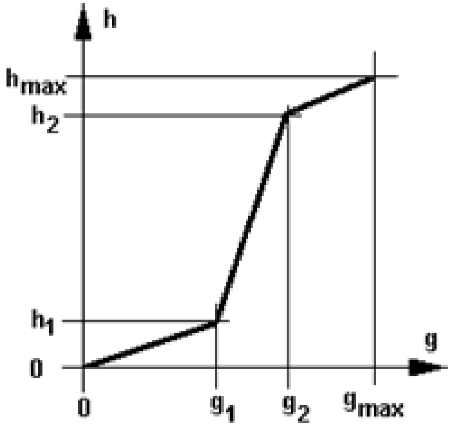
Modified Image



Histogram of the Modified Image

Dynamic Range Expansion

- Example: extracting the intensity values from the $[g_1, g_2]$ interval to a wider $[h_1, h_2]$ domain
  - Enhanced contrast in the selected region, details are better observable and distinguishable.
  - In the remaining image regions the contrast decreases



Histogram Transformations

- **Histogram Stretching**:
  - Based on the histogram we can see that the image does not use the whole range of possible intensities:
    - Minimum intensity level: 72
    - Maximum intensity level: 190
  - With the following transformation we can stretch the intensity values so they use the whole available range:

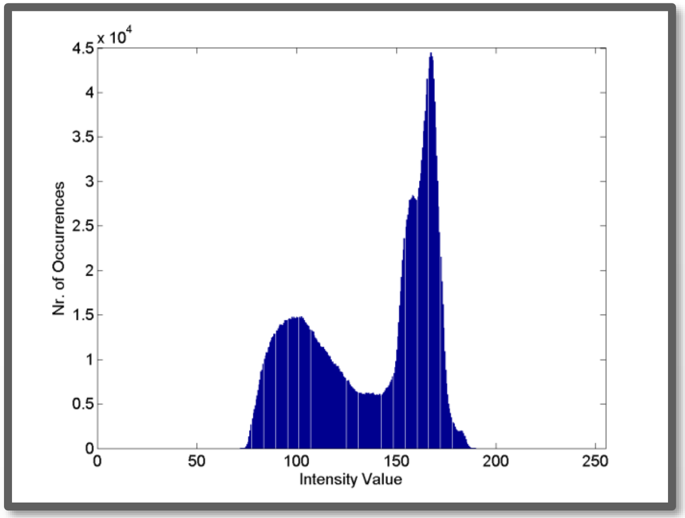


Image Histogram

$$y(n_1, n_2) = \frac{255}{x_{\max} - x_{\min}} \cdot (x(n_1, n_2) - x_{\min})$$

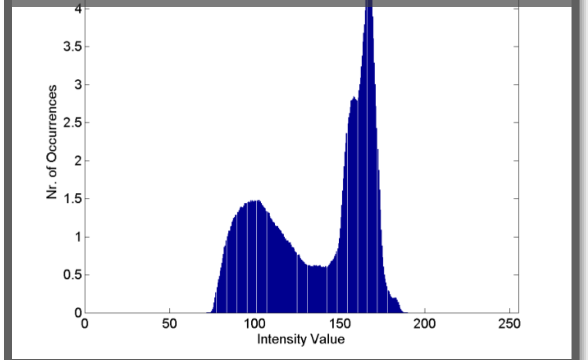
$$x_{\max} = \max_{n_1, n_2}(x(n_1, n_2)) \qquad x_{\min} = \min_{n_1, n_2}(x(n_1, n_2))$$

Histogram Transformations

- **Histogram Stretching:**

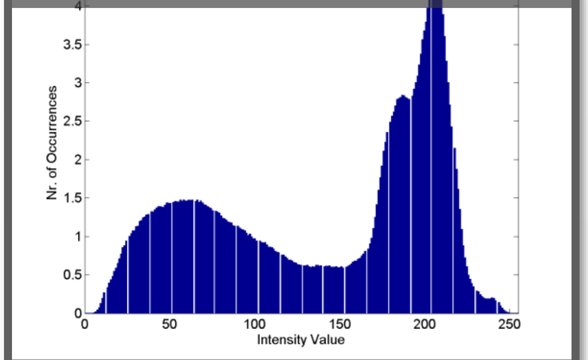
Original image

Original histogram



Histogram stretched image

Stretched histogram



Histogram Transformations

- Histogram stretching with various transfer functions:
  - Linear:

$$y(n_1,n_2)=\frac{255}{x_{\max}-x_{\min}}\cdot(x(n_1,n_2)-x_{\min})=255\cdot\frac{x(n_1,n_2)-x_{\min}}{x_{\max}-x_{\min}}$$

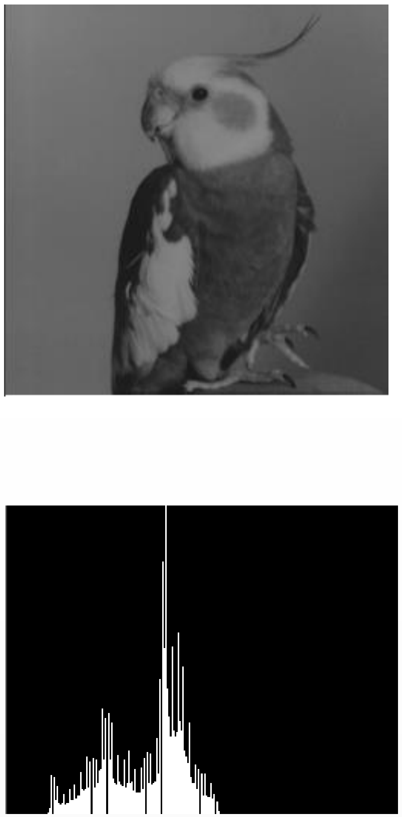
  - Quadratic:

$$y(n_1,n_2)=255\cdot\left(\frac{x(n_1,n_2)-x_{\min}}{x_{\max}-x_{\min}}\right)^2$$

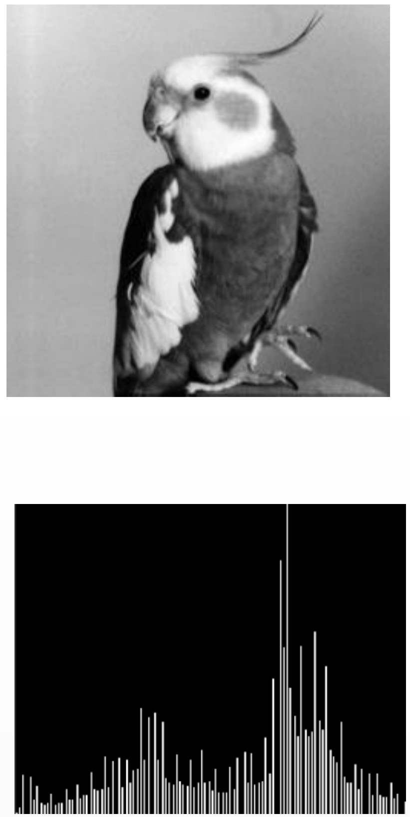
  - Square root

$$y(n_1,n_2)=255\cdot\sqrt{\frac{x(n_1,n_2)-x_{\min}}{x_{\max}-x_{\min}}}$$

Histogram stretching - results

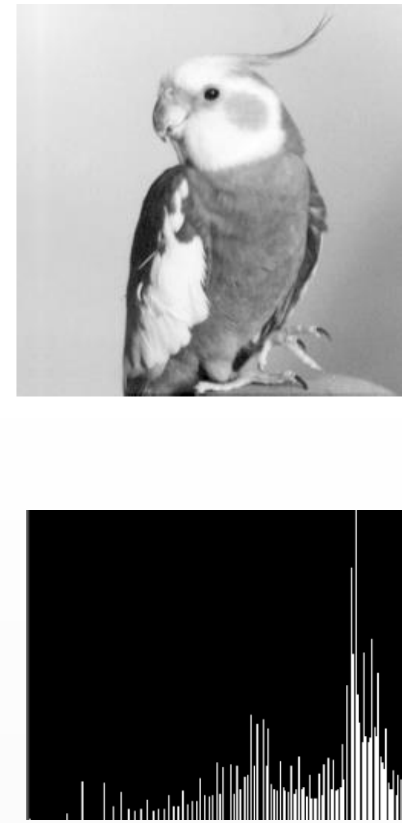


original



linear f()

quadratic



square root

Histogram stretching - results

original

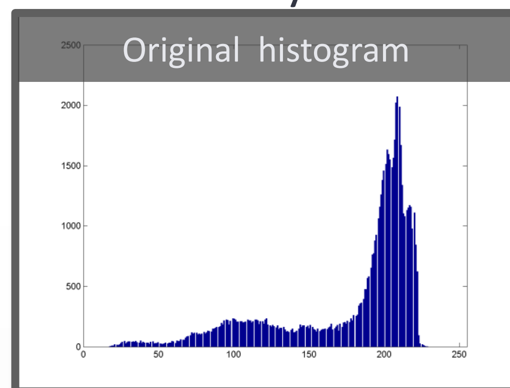
linear f()

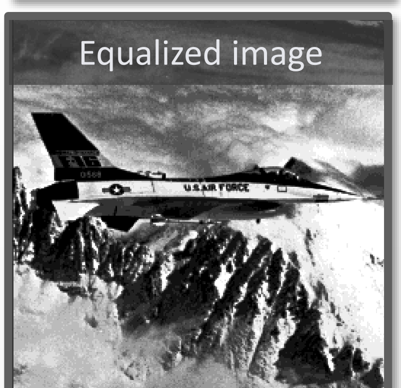
quadratic

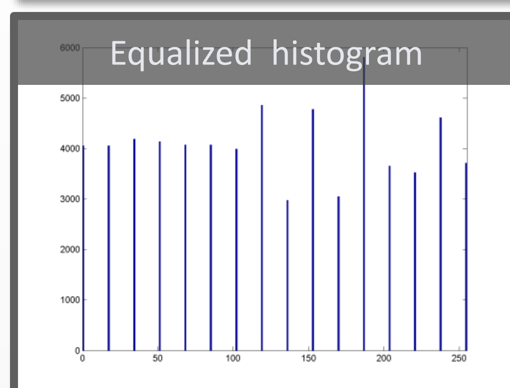
square
root

Histogram Transformations

- **Histogram Equalization**:
  - The goal is to increase the contrast, by distributing the occurrences of the intensity values evenly through the entire dynamic range.

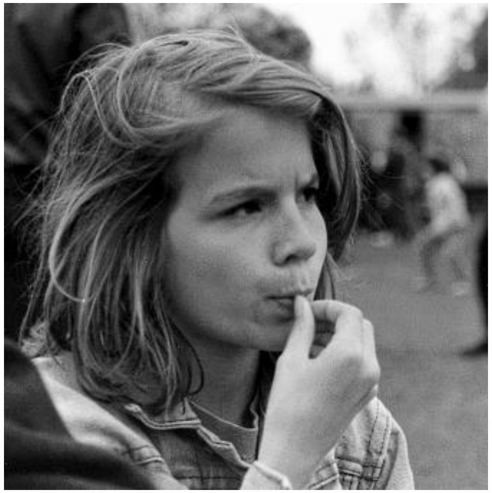


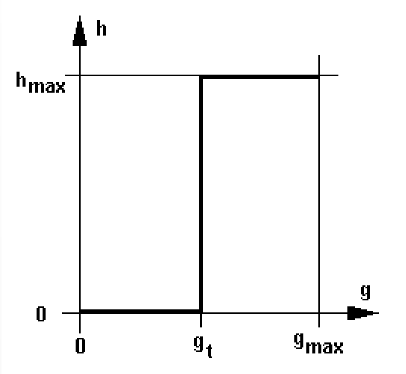




Histogram equalization background

- Simple thresholding





- For different g_t values

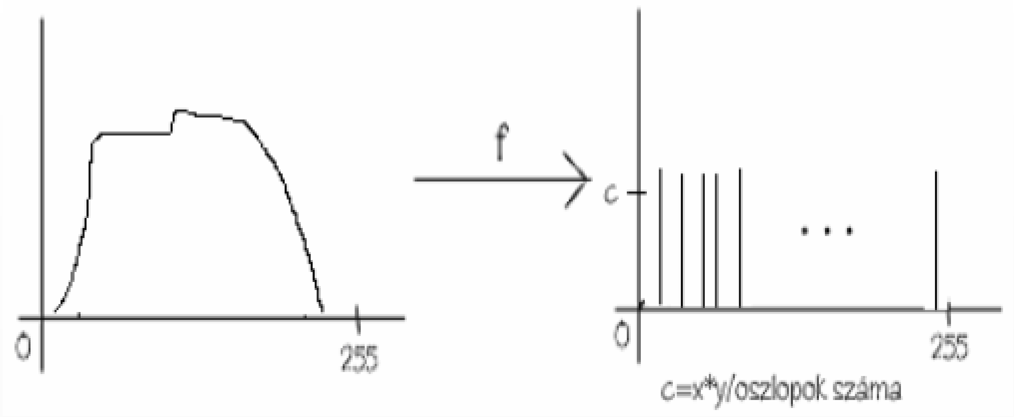
Optimal threshold value

- Task: converting a grayscale image to binary (black&white). What is the optimal threshold value?
  - A possible good solution is to prescribe that the number of black and white pixels should be approximately the same in the output image.
    - The g_t threshold value can be calculated from the histogram, (P is the total number of pixels):

$$\sum_{i=0}^{g_t} h[i] \approx \sum_{i=g_t+1}^{255} h[i] \approx \frac{P}{2}$$

Generalization: histogram equalization

- **Goal**: contrast enhancement
- Transform: step (staircase) function. The number of columns determines number of color (intensity) values appearing in the output, (e.g. number of columns=16, 32, 64, etc.).



Histogram equalization – c output values

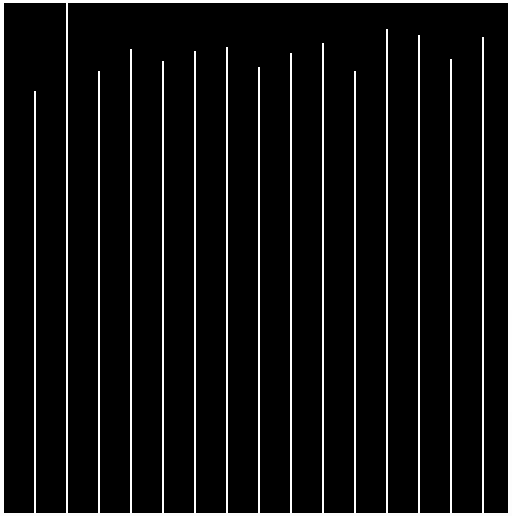
- Goal: determining the $t_0=0$, t_1, … t_{c-1},$t_c=255$ dividing points, where:

$$\sum_{i=0}^{t_j} h[i] \approx P \cdot \frac{j}{c} \qquad j \in \{1 \ldots c\}$$

  - c is the number of different gray levels in the output image (c=2 for thresholding, but it can also be 16, 32, … 256 as well)
  - P is the total number of pixels again.

Histogram equalization - result

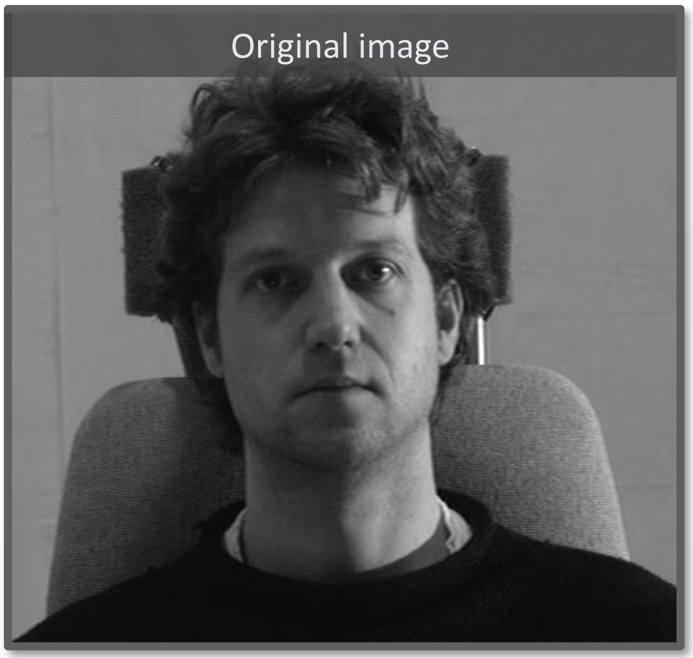
16 level ouptut



Histogram of the output image

Histogram Transformations

- Adaptive Histogram Equalization:
  - applies histogram equalization on parts of the image (called tiles) independently
  - Use post processing to reduce artifacts at the borders of the tiles.



[1] Zuiderveld, Karel. "Contrast Limited Adaptive Histograph Equalization." *Graphic Gems IV*. San Diego: Academic Press Professional, 1994. 474–485.

Spatial Filtering

- **Smoothing:**
  - Reduce the noise that may corrupt the image.
- A few noise types we will work with:
  - Impulse noise, (aka salt and pepper noise)
  - Additive Gaussian Noise

Additive Gaussian Noise

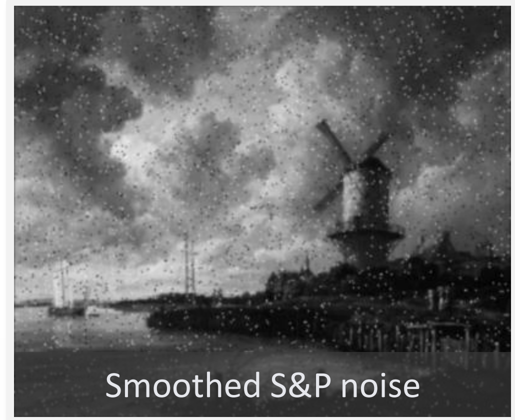
Impulse Noise

The windmill at Wijk bij Duurstede by Jacob van Ruisdael (1670)

Spatial Filtering

- **Gaussian Smoothing:**
  - With $\sigma=0.75$

Image with S&P noise



Smoothed S&P noise

Original image

Image with Gaussian noise

Smoothed Gaussian noise

Spatial Filtering

- **Gaussian Smoothing:**
  - With σ=*1.5*

Original image

Image with S&P noise

Smoothed S&P noise

Image with Gaussian noise

Smoothed Gaussian noise

Spatial Filtering

- **Spatially Adaptive Noise Smoothing:**
  - The smoothing takes into account the local characteristics of the image:

$$y(n_1, n_2) = \left(1 - \frac{\boxed{\sigma_n^2}}{\boxed{\sigma_l^2}}\right) \cdot x(n_1, n_2) + \frac{\boxed{\sigma_n^2}}{\boxed{\sigma_l^2}} \cdot \boxed{\bar{x}(n_1, n_2)}$$

$$\boxed{\sigma_l^2(n_1, n_2) = \sum_{(n_1, n_2) \in N} \sum \left(x(n_1, n_2) - \bar{x}(n_1, n_2)\right)^2}$$

Local variance of the image

$$\boxed{\bar{x}(n_1, n_2) = \frac{1}{|N|} \sum_{(n_1, n_2) \in N} \sum x(n_1, n_2)}$$

Local average of the image

Variance of the noise: either known a priori, or has to be measured

Spatial Filtering

- **Spatially Adaptive Noise Smoothing:**


Original image*

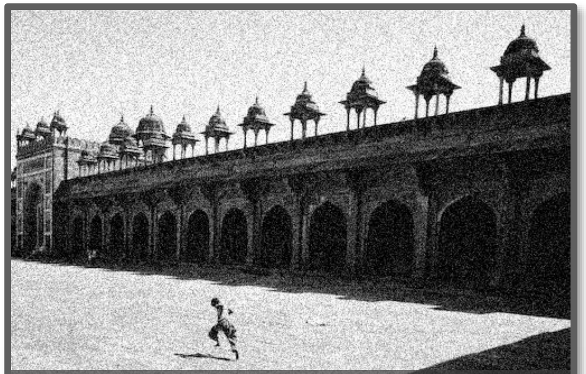
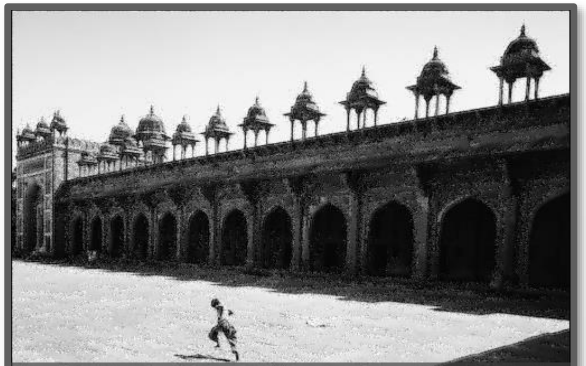


Image with Gaussian Noise


Gaussian Smoothing


Spatially Adaptive Smoothing

* Fatepuhr Sikri, Inde by Lucien Hervé (1955)