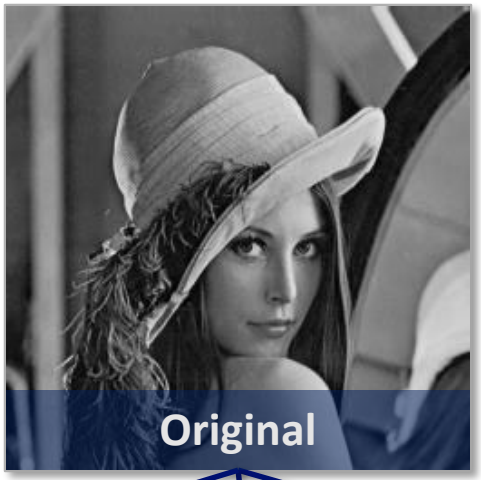
Basic Image Processing Algorithms

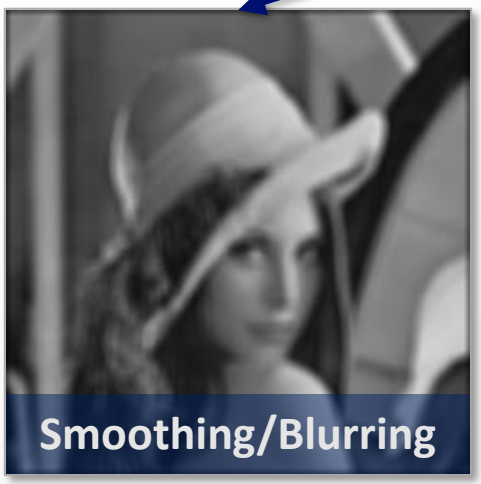
PPKE-ITK

Lecture 2.

2D Convolution

Examples


Original


Smoothing/Blurring


Edge Enhancement


Edge Detection

Mathematical background

- We look at the image as a 2D function: $x(n_1, n_2)$
- We can define different transformations:
  - Intensity value inversion:
$$y(n_1, n_2) = 255 - x(n_1, n_2)$$
  - Intensity shift with constant:
$$y(n_1, n_2) = x(n_1, n_2) + 100$$
  - Weighting:
$$y(n_1, n_2) = x(n_1, n_2) w(n_1, n_2)$$
  - Average on an *N* neighborhood:
$$y(n_1, n_2) = average\ \left(N\left(x(n_1, n_2)\right)\right)$$
- 2 important properties of the transformations we want to use on images: ***linearity*** and ***shift invariance***

Mathematical background

- Linearity:

$$T[x_1(n_1,n_2)+x_2(n_1,n_2)]=T[x_1(n_1,n_2)]+T[x_2(n_1,n_2)]$$

$$T[\alpha x(n_1,n_2)]=\alpha T[x(n_1,n_2)]$$

  - e.g.: weighting is linear, intensity inversion is non-linear

- Spatial Invariance:

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

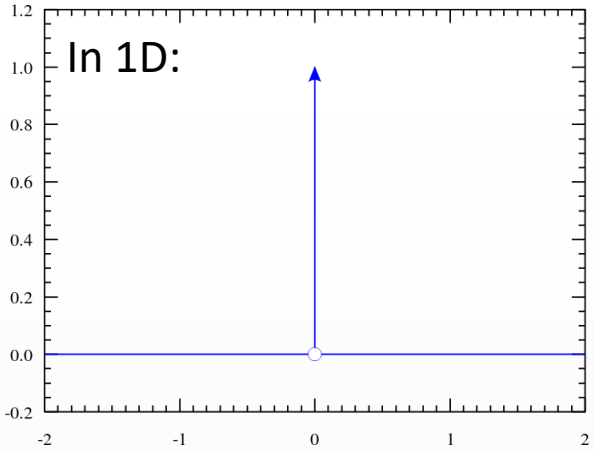
$$T[x(n_1-k_1,n_2-k_2)]=y(n_1-k_1,n_2-k_2)$$

  - e.g.: weighting is not SI, intensity inversion is SI
  - e.g.: averaging on neighborhood is LSI

Unit Impulse Function

- 2D Unit Impulse function (Delta function) on $\mathbb{Z}$ as follows:

$$\delta(n_1, n_2) = \begin{cases} 1 & \text{when } n_1 = 0 \text{ and } n_2 = 0 \\ 0 & \text{otherwise} \end{cases}$$



- For any 2D function $x(n_1,n_2)$:

$$x(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \delta(n_1 - k_1, n_2 - k_2) x(k_1, k_2)$$

Convolution

- ***Impulse response*** is the output of an LSI transformation if the input was the Delta function: $\delta(n_1, n_2) \rightarrow T \rightarrow h(n_1, n_2)$

  If *T* is an LSI system:

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

  Then we can define convolution as follows:

$$y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2) =$$

$$= h(n_1, n_2) * x(n_1, n_2) =$$

$$= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) h(n_1 - k_1, n_2 - k_2)$$

Derivation of Convolution

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

$$= T\left[\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} x(k_1, k_2)\delta(n_1 - k_1, n_2 - k_2)\right] =$$

Linearity

$$= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} x(k_1, k_2)T\left[\delta(n_1 - k_1, n_2 - k_2)\right] =$$

$$= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} x(k_1, k_2)h(n_1 - k_1, n_2 - k_2)$$

Spatial Invariance

The Properties of Convolution

- Commutative:

$$f * g = g * f$$

- Associative:

$$f * (g * h) = (f * g) * h$$

- Distributive:

$$f * (g + h) = f * g + f * h$$

- Associative with scalar multiplication:

$$\alpha(f * g) = (\alpha f) * g$$

2D convolution in Practice

- In practice both the kernel and the image have finite size.

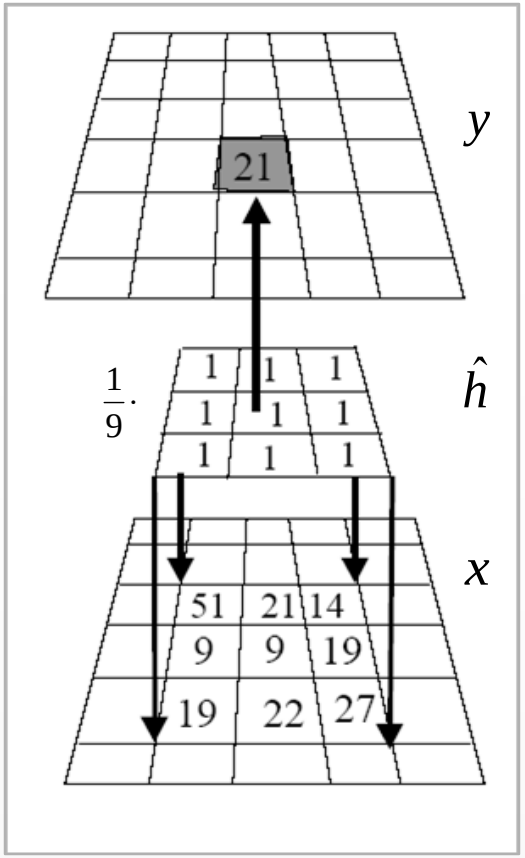
Let h and $\hat{h}$ be $(2r_1+1)\times(2r_2+1)$ sized kernels, where $\hat{h}$ is the $180°$ rotated version of h:

$$h=\begin{bmatrix} a_{-r_1,-r_2} & \cdots & a_{-r_1,r_2} \\ \vdots & \ddots & \vdots \\ a_{r_1,-r_2} & \cdots & a_{r_1,r_2} \end{bmatrix} \text{ and } \hat{h}=\begin{bmatrix} a_{r_1,r_2} & \cdots & a_{r_1,-r_2} \\ \vdots & \ddots & \vdots \\ a_{-r_1,r_2} & \cdots & a_{-r_1,-r_2} \end{bmatrix}$$

$$y(n_1,n_2)=\sum_{k_1=-r_1}^{r_1}\sum_{k_2=-r_2}^{r_2}x(k_1,k_2)h(n_1-k_1,n_2-k_2)=$$

$$=\sum_{k_1=-r_1}^{r_1}\sum_{k_2=-r_2}^{r_2}h(k_1,k_2)x(n_1-k_1,n_2-k_2)=$$

$$=\sum_{k_1=-r_1}^{r_1}\sum_{k_2=-r_2}^{r_2}\hat{h}(k_1,k_2)x(n_1+k_1,n_2+k_2)$$



Size of the Convolved Image

Convolutional Kernel

$$* \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

5x5

In general:

Size of the input image: ***(A x B)***

Size of the kernel: ***(C x D)***

Size of the output image: ***(A+C-1) x (B+D-1)***

Boundary Effects

- What happens at the border of the image?

Original image with
the problematic area

Zero padding

Mirroring

Circular padding

Repeating border

Applications

- Possible application of convolution:
  - Smoothing/Noise reduction
  - Edge detection
  - Edge enhancement
- Depending on the task the *sum of the elements of the kernel matrix* can be different:
  - 1: smoothing, edge enhancement

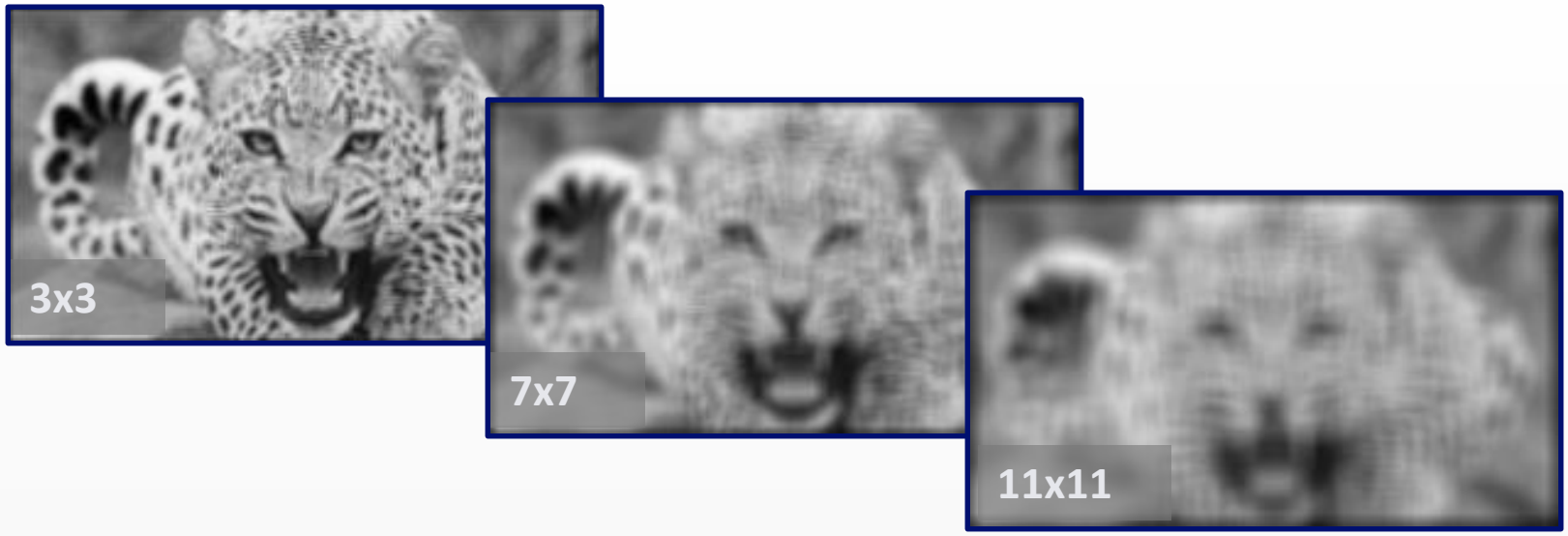
E.g.: $\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ $\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$

  - 0: edge detection

E.g.: $\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$ $\begin{bmatrix}0&-1&0\\-1&4&-1\\0&-1&0\end{bmatrix}$

Smoothing/Blurring

- Simple average:



Computational requirements

- For k_s kernel size and P image size (area, measured in pixels) approximately ~ $k_s P$ operations are needed.
- For large kernel size the execution may be slow

Decreasing the computational need for a simple (averaging) blur operation

- Integral image: $f \rightarrow I_f$ auxilliary representation

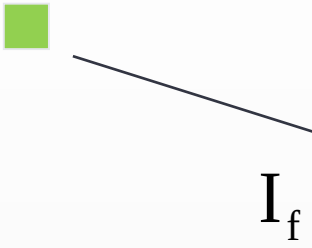
$$I_f(x,y) = \sum_{i=1}^{x}\sum_{j=1}^{y} f(i,j)$$

- **E.g.:** $I_f(3,3)$ **= sum of the values of pixels:11**

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4

f

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4



I_f

1	1	3	4
3	3	6	7
6	7	11	12
7	8	13	18

Calculation of I_f with dynamic programing in ~P time:

- Auxiliary-auxiliary image: $t(x,y)=\sum_{j=1}^{y} f(x,j)$

$$t(x,1) := f(x,1),\ x = 1...w \qquad t(x,y) = t(x,y-1) + f(x,y)$$

$$I_f(1,y) := t(1,y),\quad y = 1...h$$

$$I_f(x,y) = I_f(x-1,y) + t(x,y)$$

4+7=11

f :

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4

t :

1	0	2	1
3	0	3	1
6	1	4	1
7	1	5	5

I_f

1	1	3	4
3	3	6	7
6	7	11	12
7	8	13	18

Utilization of the integral image

- Sum of pixel values in an **arbitrary sized** sub-rectangle can be calculated by applying **3 additive operations** using the integral image:

$$\sum_{i=a}^{c}\sum_{j=b}^{d} f(i,j) = I_f(c,d) - I_f(a-1,d) - I_f(c,b-1) + I_f(a-1,b-1)$$

- Example (a=1, b=1, c=2, d=2): 11-6-3+1=3

f

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4

I_f

1	1	3	4
3	3	6	7
6	7	11	12
7	8	13	18

Using integral image for quick blurring (simple averaging kernel)

$$\tilde{f}(x,y) = \frac{1}{(2r+1)^2} \sum_{i=-r}^{r} \sum_{j=-r}^{r} f(x+i, y+j)$$

$(2r+1)^2$ addition
+ 1 division operations

Example: r=5 → For the whole image ~122P operations

$$\tilde{f}(x,y) = \frac{1}{(2r+1)^2} (I_f(x+r, y+r) - I_f(x-r-1, y+r) -$$
$$- I_f(x+r, y-r-1) + I_f(x-r-1, x-r-1))$$

3 addition
+1 division

Example: r=5 → For the whole image ~ 2P+4P=6P operations

calc. integral image calc. bluring

Optional homework (a bit more than a convolution)

- Construct an efficient contrast calculating algorithm using the integral image! Contrast is calculated as the standard deviation of pixel values of the $(2r+1)^2$ size neighborhood of each pixel.

$$\sigma^2(x,y) = \frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r}\left[f(x+i,y+j) - \tilde{f}(x,y)\right]^2$$

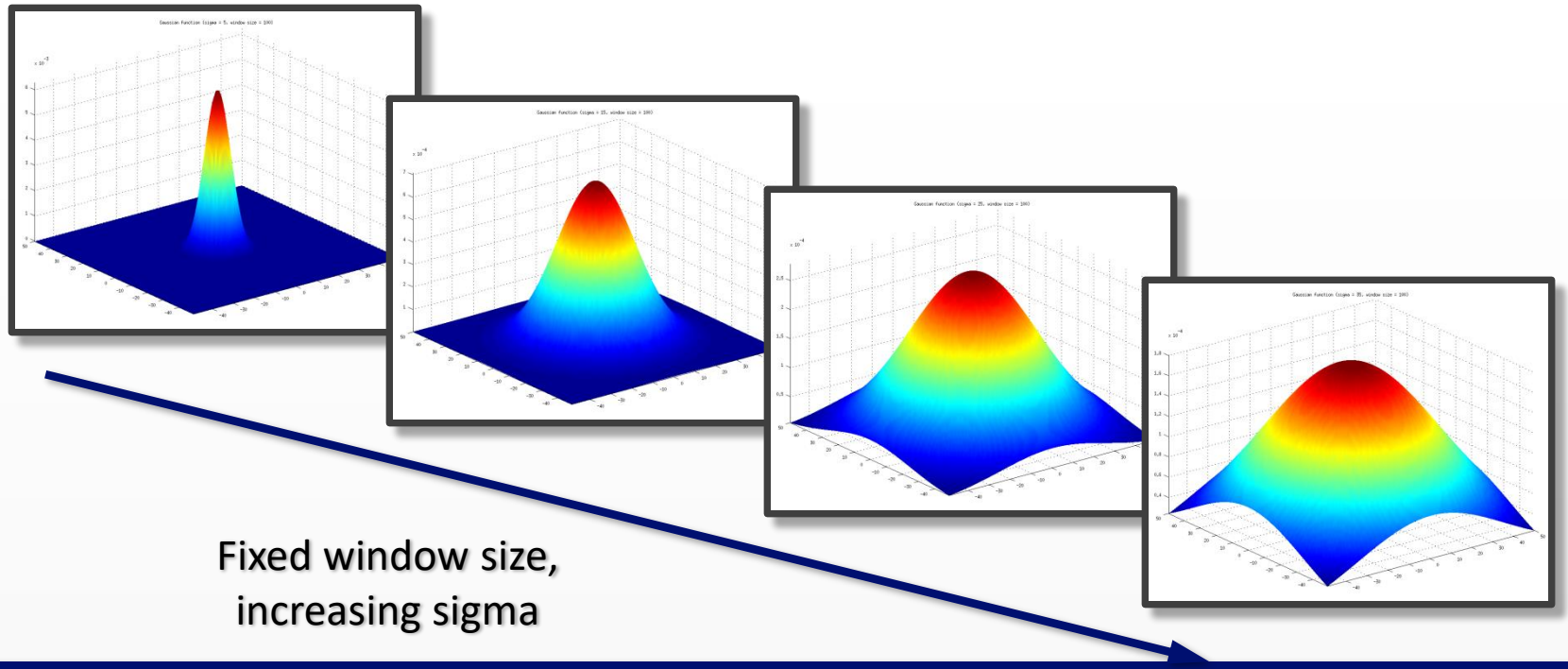
where: $$\tilde{f}(x,y) = \frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r} f(x+i,y+j)$$

Hint:

$$\sigma^2(x,y) = \left\{\frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r}\left[f(x+i,y+j)\right]^2\right\} - \left[\tilde{f}(x,y)\right]^2$$

Smoothing/Blurring

- Gaussian blur:
  - Weights are defined by a 2D Gaussian function
  - 2 parameters: **size of the window** and the **standard deviation** of the Gaussian



Smoothing/Blurring

- Gaussian blur:
  - Weights are defined by a 2D Gaussian function
  - 2 parameters: window size and the width of the Gaussian
  - E.g. kernel size = 5x5; σ = 1.5;

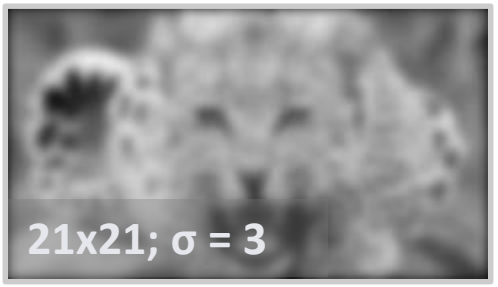
$$\begin{bmatrix} 0.0144 & 0.0281 & 0.0351 & 0.0281 & 0.0144 \\ 0.0281 & 0.0547 & 0.0683 & 0.0547 & 0.0281 \\ 0.0351 & 0.0683 & 0.0853 & 0.0683 & 0.0351 \\ 0.0281 & 0.0547 & 0.0683 & 0.0547 & 0.0281 \\ 0.0144 & 0.0281 & 0.0351 & 0.0281 & 0.0144 \end{bmatrix}$$

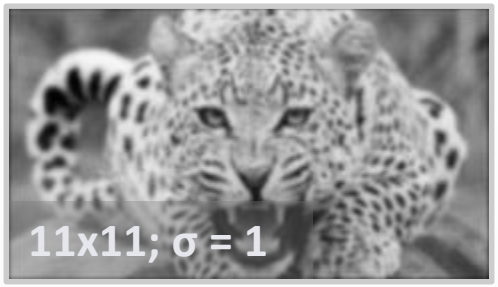
  - E.g. kernel size = 3x3; σ = 1.5;

$$\begin{bmatrix} 0.0947 & 0.1183 & 0.0947 \\ 0.1183 & 0.1478 & 0.1183 \\ 0.0947 & 0.1183 & 0.0947 \end{bmatrix}$$

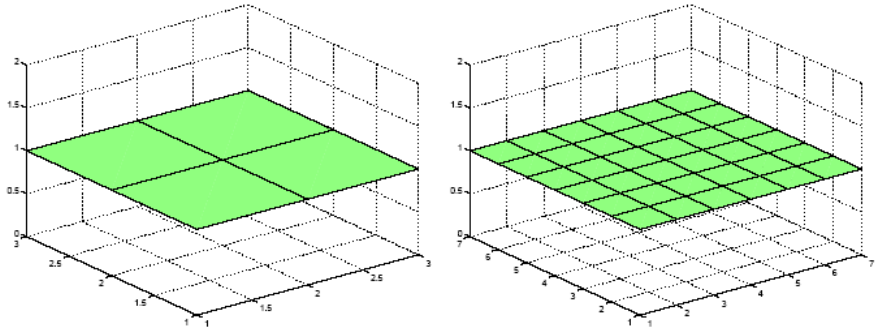
Smoothing/Blurring

- Gaussian blur:





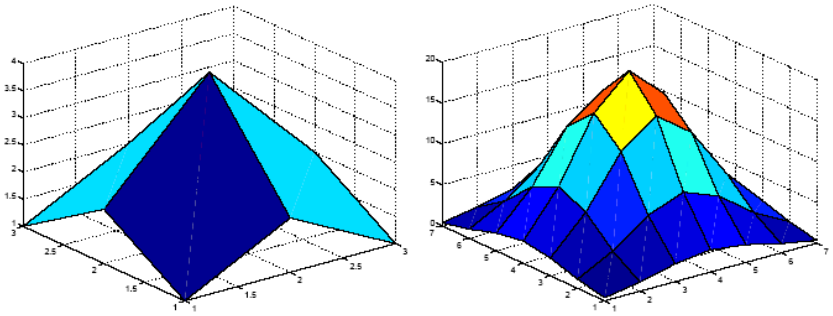
Convolution examples – averaging blur



Input Image

Average blur

Convolution examples– Gaussian blur



$$K = \frac{1}{123} * \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 7 & 11 & 7 & 2 \\ 3 & 11 & 17 & 11 & 3 \\ 2 & 7 & 11 & 7 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

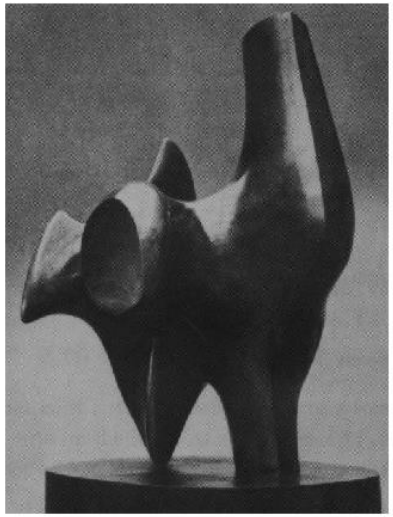
Input Image

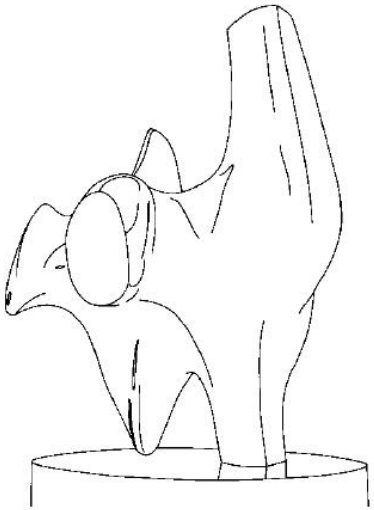
Gaussian blur

Edge detection

- Goal: extracting the object contours
- Edge points: brightness changes sharply

Goals of edge detection





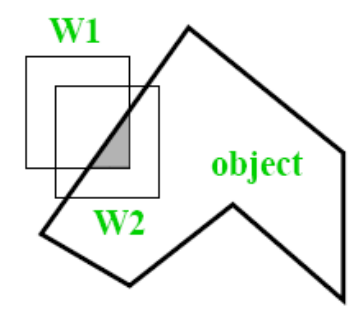
- Goal: extracting curves from 2D images
  - More compact content representation then pixel
  - Segmentation, recognition, scratch filtering

Goals of edge detection

- Extracting image information, structures
  - Corners, lines, borders
- Not always simple...

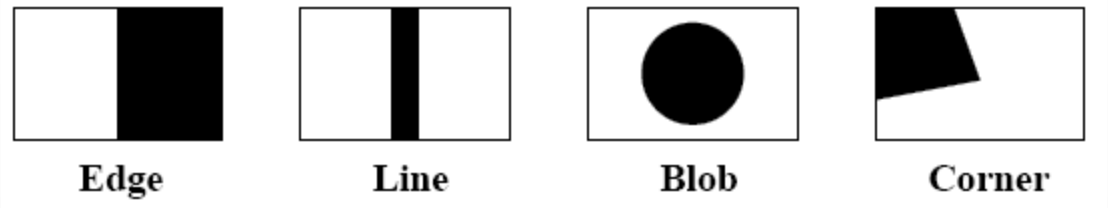
Edge detection

- Properties of a good edge filter:
  - (Near) zero output in homogeneous regions (constant intensity)
  - Good detection :
    - detects as many real edges as possible
    - does not create false edges (because of e.g. image noise)
  - Good localization: detected edges should be as close as possible to the real edges
  - Isotropic: filter response independent on edge directions
    - all edges are detected regardless of their direction



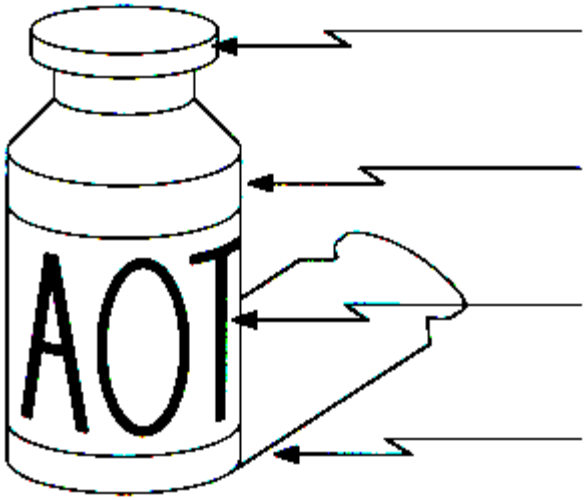
Basic structures

- **Edge**: sharp intensity change (steep or continuous)
- **Line**: thin, long region with approx. uniform width and intensity level
- **Blob**: closed region with homogeneous intensity
- **Corner**: breaking or direction change of a contour or edge



Origin of edges

- Various effects may cause edges



Sharp change in surface normals

Continuos change in surface depth

Change in surface color

Changes cased by illumination/shadows

Parameters of an edge

- Edge normal: vector, perpendicular to the edge, pointing toward the steepest intensity change
- Edge direction: vector point towards the direction of the line
  - Position, center point
- Strength: intensity ratio w.r.t. neighborhood

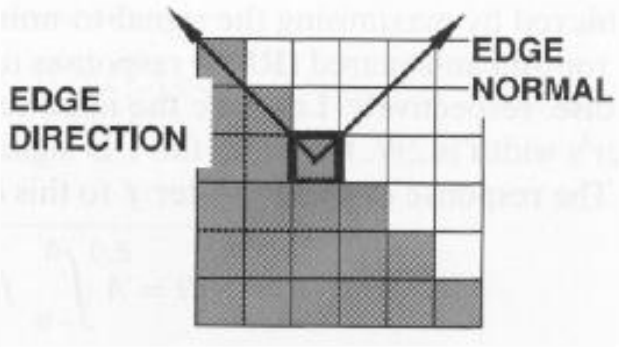
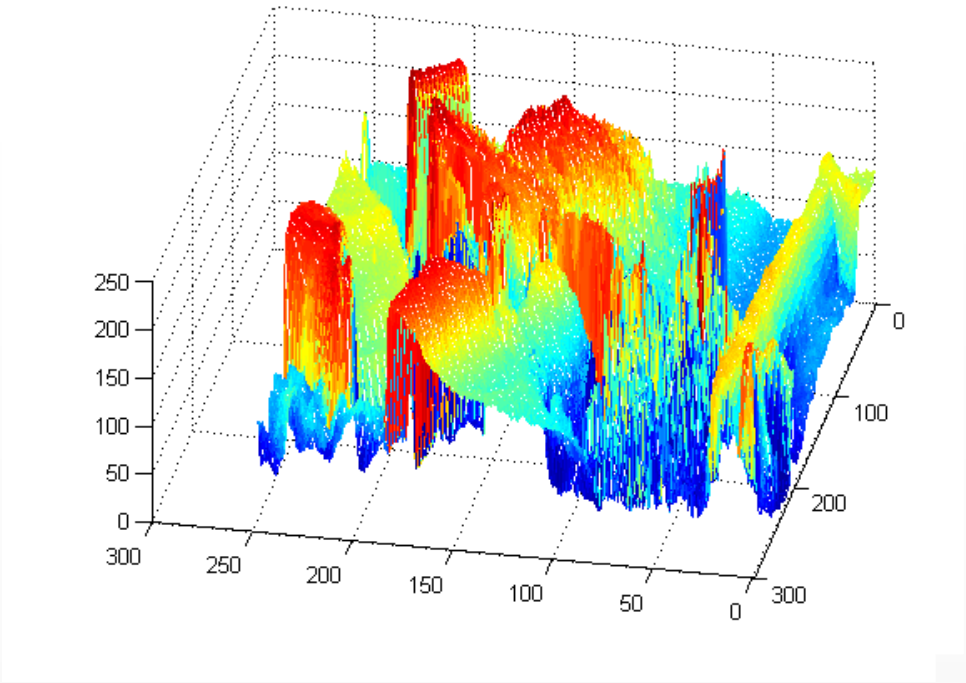


Image representation

- Image: gray value is function of the *x* and *y* coordinates (intensity function): $f(x, y)$



Edge Detection

- ***Edges types:*** locations on the image where the ***intensity changes sharply*** (usually at the contour of objects)

- We are searching for places where the gradient of the 2D function (the image) is high.
- Main types of edge detection:
  - First order derivative
  - Second order derivative
  - Others:
    - Complex methods e.g. Canny method
    - Phase Congruancy

Edge Detection

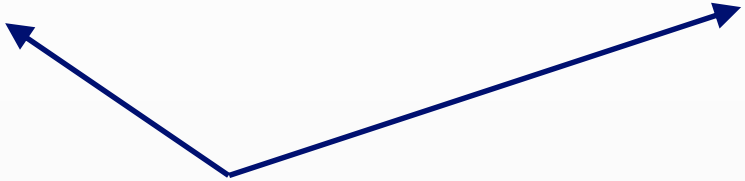
- **Edge detection with first order derivative:**
  - Using the gradient vector:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^T$$

  - The approximation of the partial derivatives:

$$\frac{\partial f}{\partial x} = \lim \frac{f(x+dx, y) - f(x, y)}{dx} \qquad \frac{\partial f}{\partial y} = \lim \frac{f(x, y+dy) - f(x, y)}{dy}$$

$$\approx f(x+1, y) - f(x, y) \qquad \approx f(x, y+1) - f(x, y)$$



Since the smallest meaningful discrete value is *dx*=1 and *dy* =1.

Edge Detection

- **Edge detection with first order derivative:**
  - The approximation of the partial derivatives:

$$\frac{\partial f}{\partial x} \approx f(x+1, y) - f(x, y) \qquad \frac{\partial f}{\partial y} \approx f(x, y+1) - f(x, y)$$

  - Kernels for the gradient calculation with convolution (Prewitt):

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

For better localization

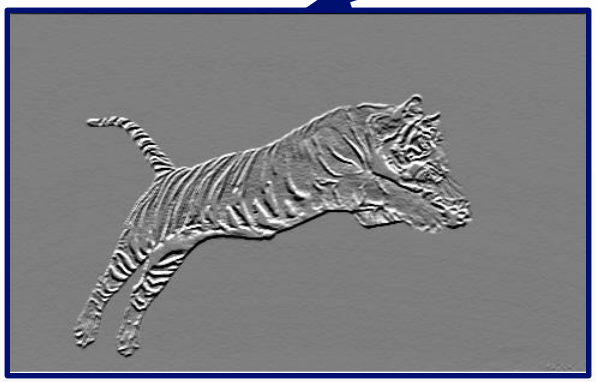
For noise reduction

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

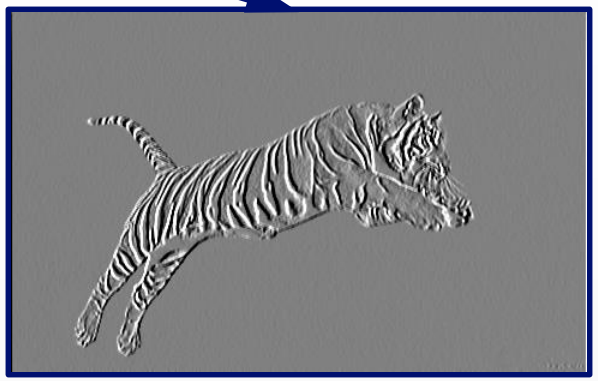
Edge Detection

- **Edge detection with first order derivative:**

Prewitt detector



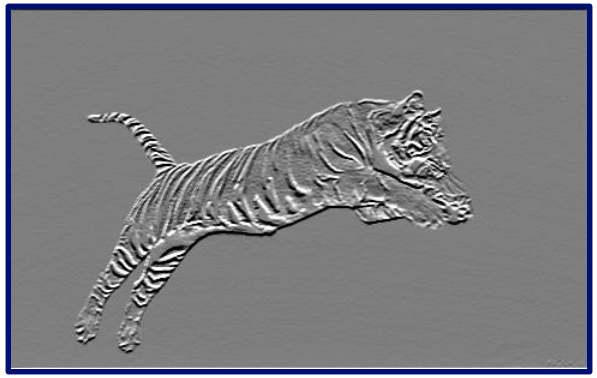
Horizontal gradient image



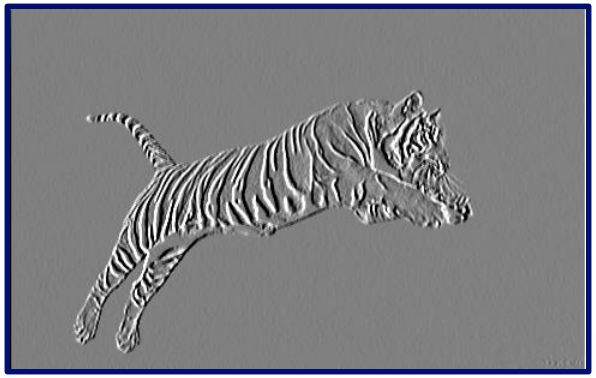
Vertical gradient image

Edge Detection

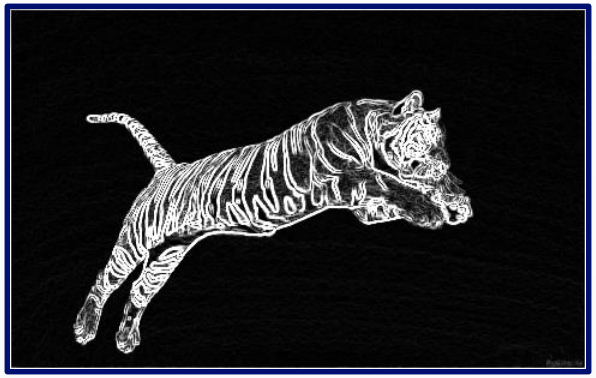
- **Edge detection with first order derivative:**



Horizontal gradient image



Vertical gradient image

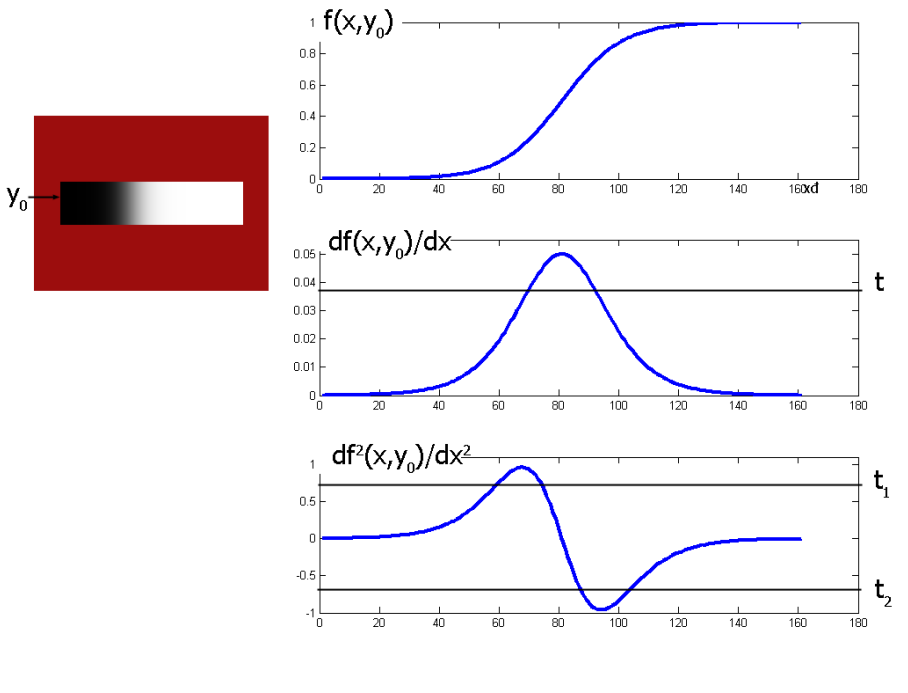


gradient image

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

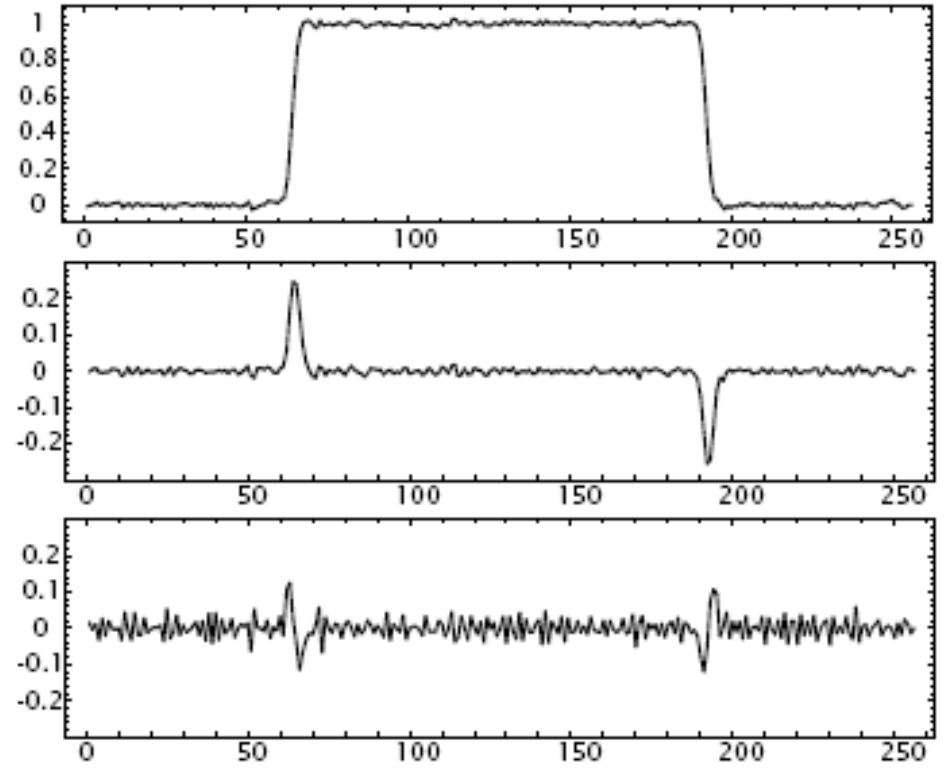
Second order edge detection: motivation

Horizontal edge detection in the following image:



Top: intensity function along a selected horizontal line
Center: x directional first derivative
Bottom: x directional second derivative

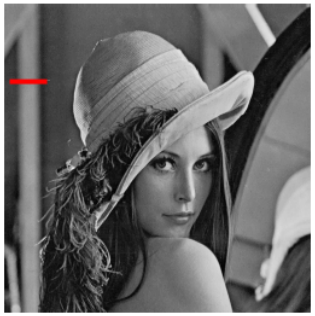
Second order case: instead of extreme values, search for zero crossing

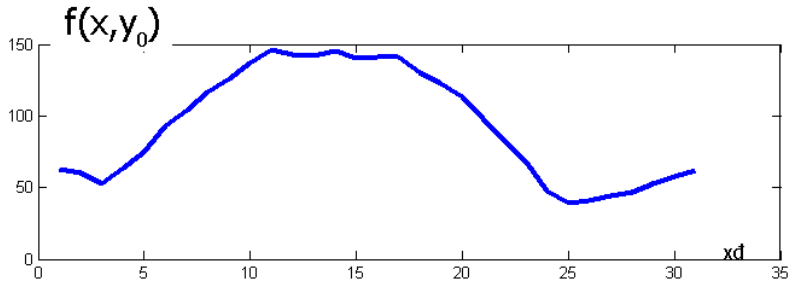


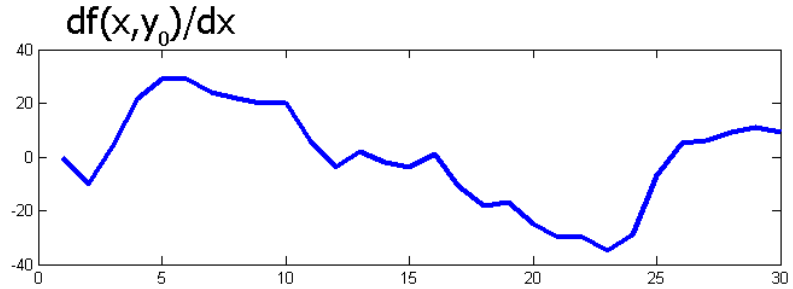
Extreme values (first order)

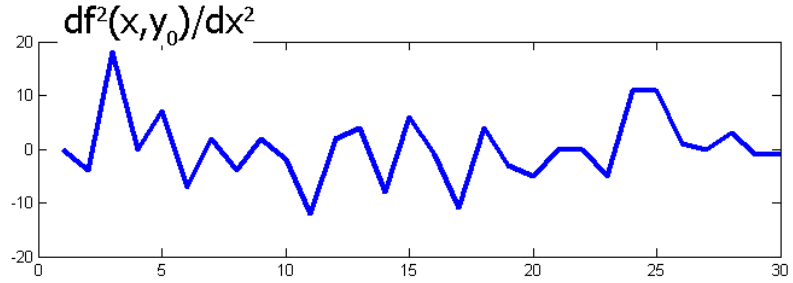
zero crossing (second order)

Real photo: intensity profile below the red line segment









Edge Detection

- **Edge detection with second order derivative:**
  - Approximation of the second order derivative for *x* direction:

$$\frac{\partial^2 f}{\partial x^2} = \lim_{d \to 0} \frac{\frac{\partial f}{\partial x}(x+d) - \frac{\partial f}{\partial x}(x)}{d} \approx \frac{\partial f}{\partial x}(x+1) - \frac{\partial f}{\partial x}(x) =$$

$$= f(x+2) - 2f(x+1) + f(x)$$

(*y* direction similarly)

  - Kernel for the second order gradient calculation with convolution:
    - Laplace operator:

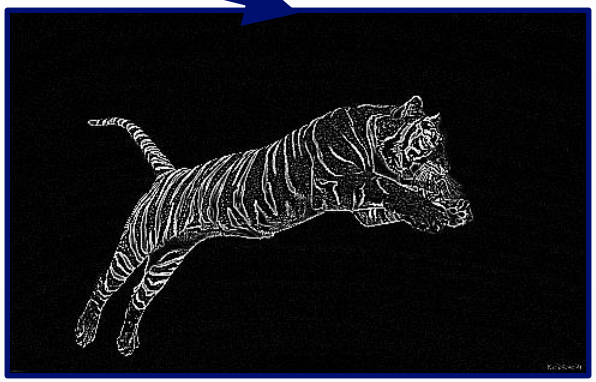
$$\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

    - There are other variations. (e.g. Second order Prewitt)

Edge Detection

- **Edge detection with second order derivative:**

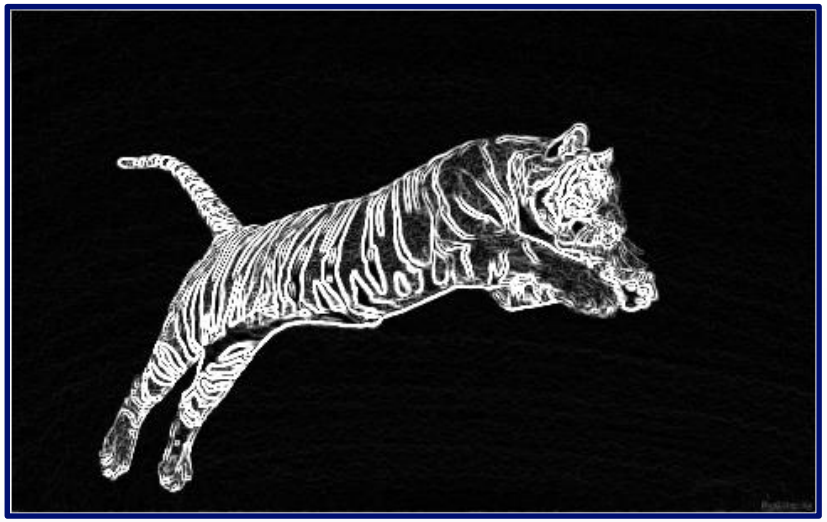
Laplace edge detector



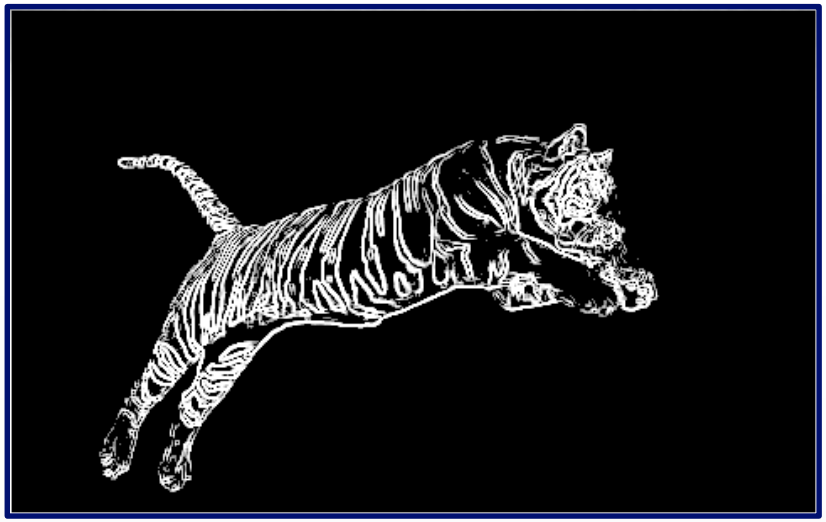
Prewitt 2nd order detector

Edge Detection

- **Thresholding**:
  - To eliminate weak edges, a threshold can be used on the gradient image:



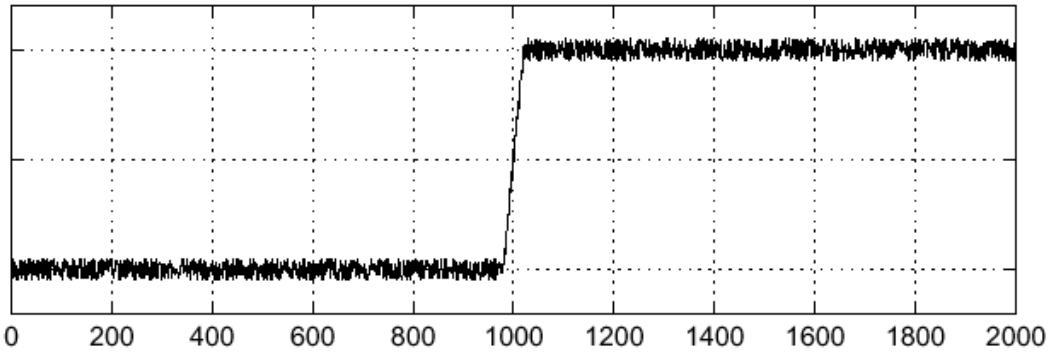
Prewitt first order gradient image



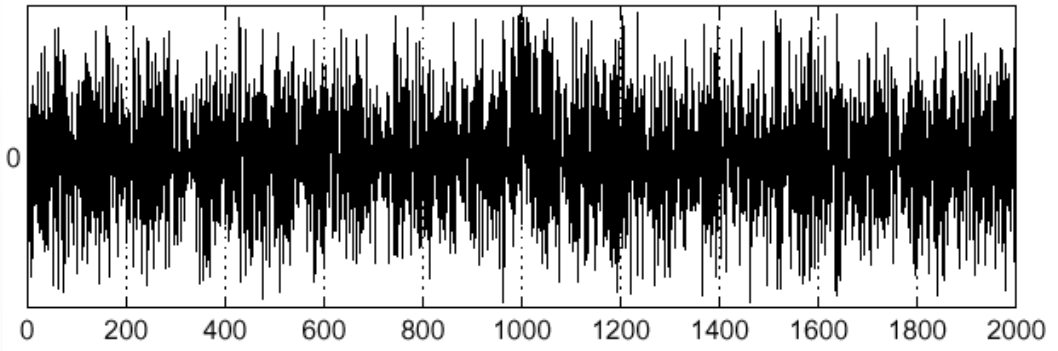
Prewitt first order gradient image with threshold = 120

Noise filtering (1D demonstration)

$f(x)$

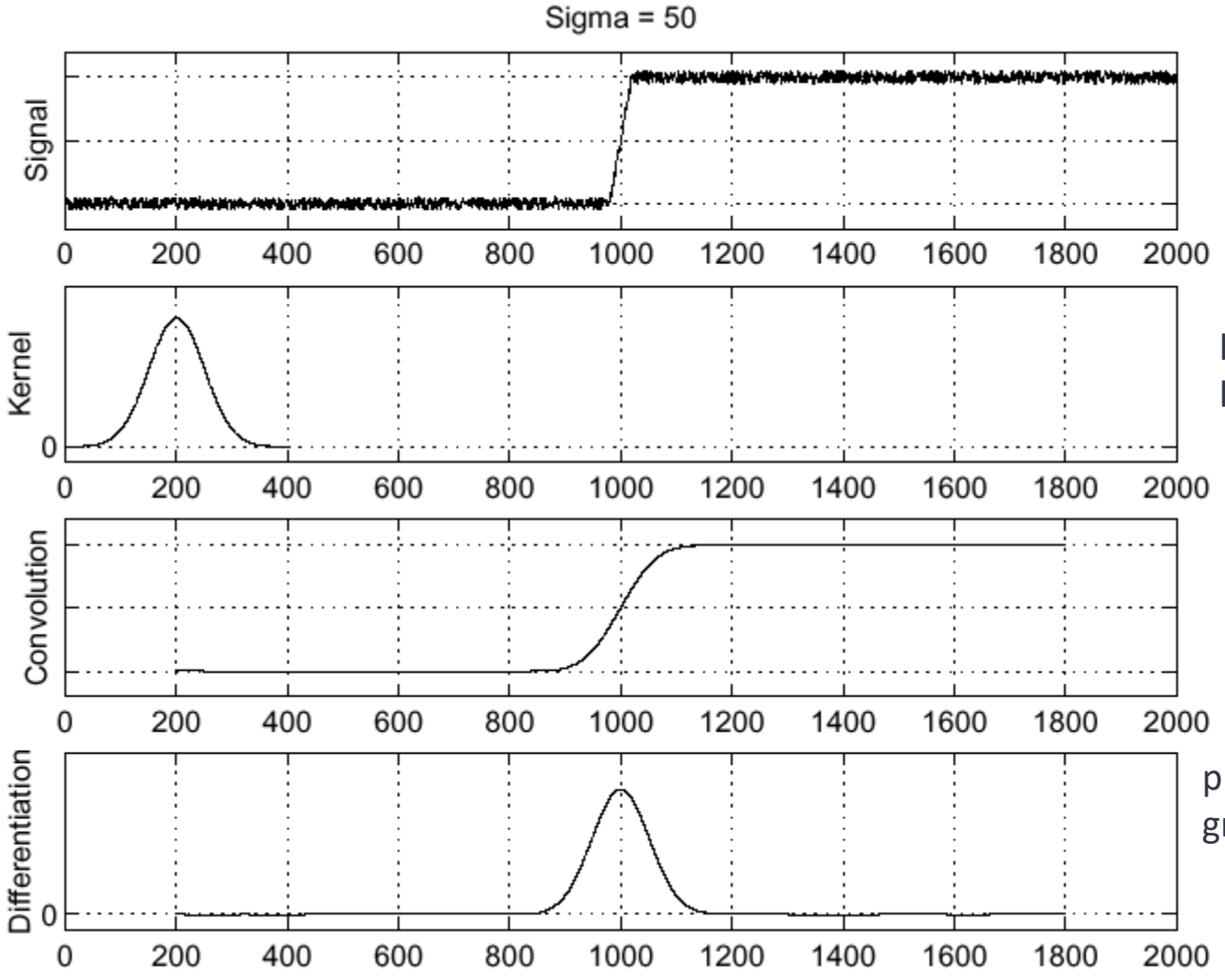


$\frac{d}{dx}f(x)$



- Where is the edge?

f



f: original signal

h

h: Gaussiam blur kernel

h*f

h*f: filtered signal

p*(h*f)

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

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

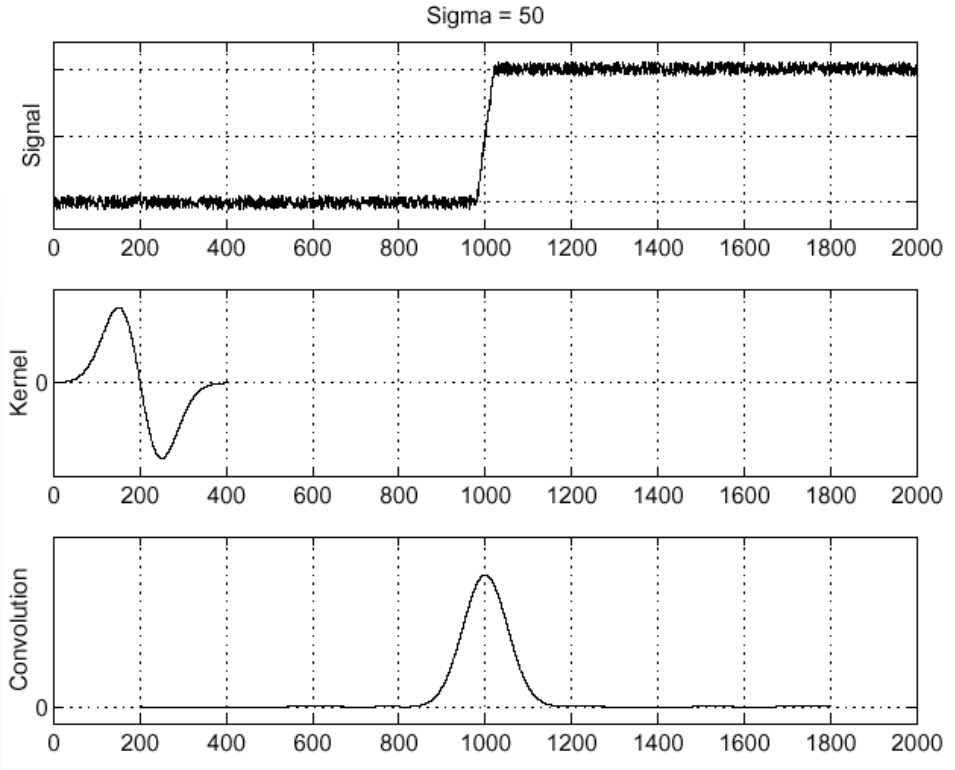
- No need for applying 2 convolutions, only one with the derivative of Gaissuan operator (can also be approximated by a discrete kernel)

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

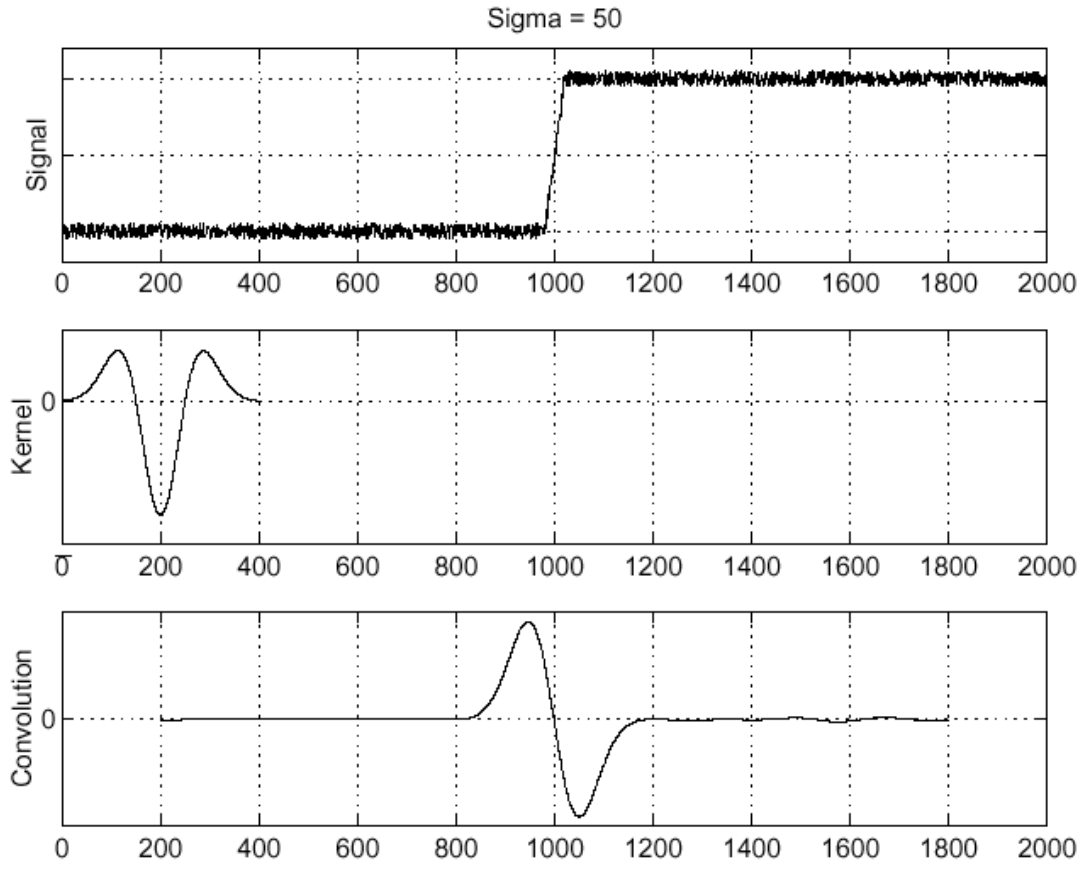
f

h: Gaussian smoothing kernel

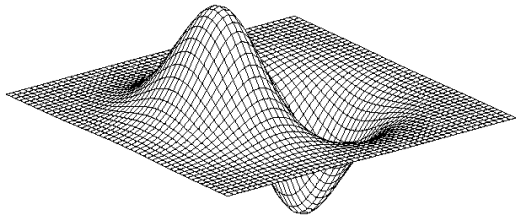
l: Laplace-kernel

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

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

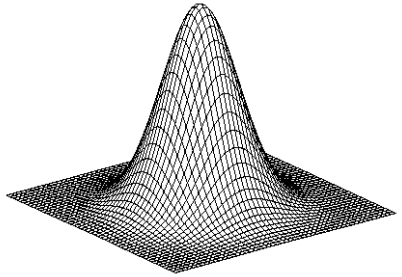


2D edge detection with filtering:



2D Gaussian smoothing kernel

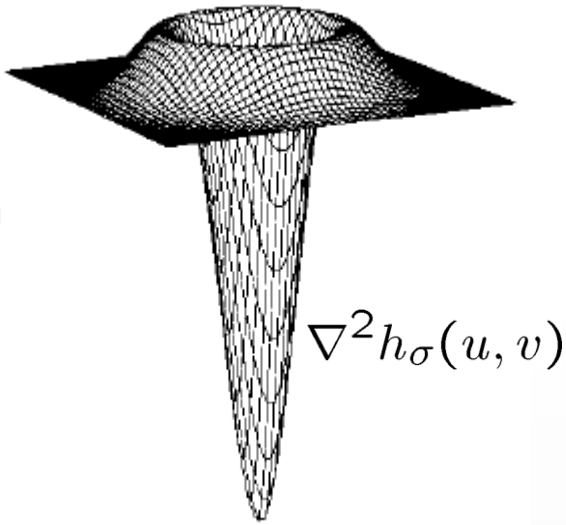
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



„Derivative" of the 2D Gaussian

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

Laplacian of Gaussian



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

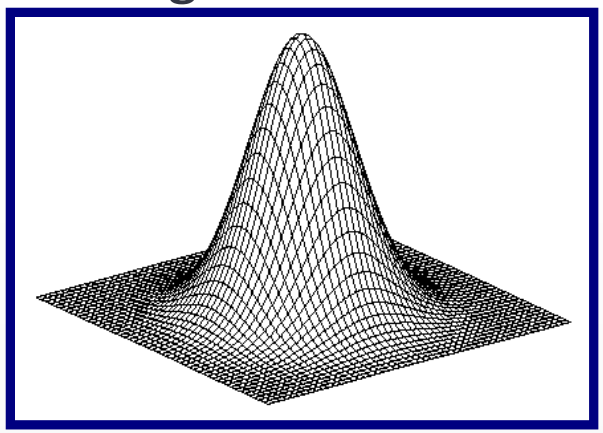
- ∇^2 henceforward the **Laplace** operator:

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

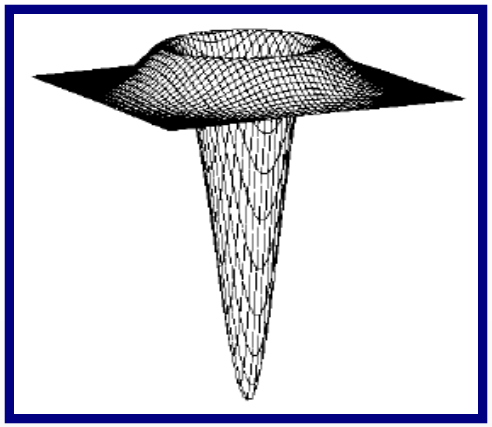
- Determining kernel coefficients with discrete approximation of the 2D function

Reducing the effect of noise on edge images

- Edge detection with noise reduction:
  - 1. step: Noise reduction by convolution with Gaussian filter
  - 2. step: Edge detection by convolution with Laplacian kernel
- Since convolution operation is associative we can convolve the Gaussian smoothing filter with the Laplacian filter first, and then convolve this hybrid filter (***Laplacian of Gaussian: LoG***) with the image.



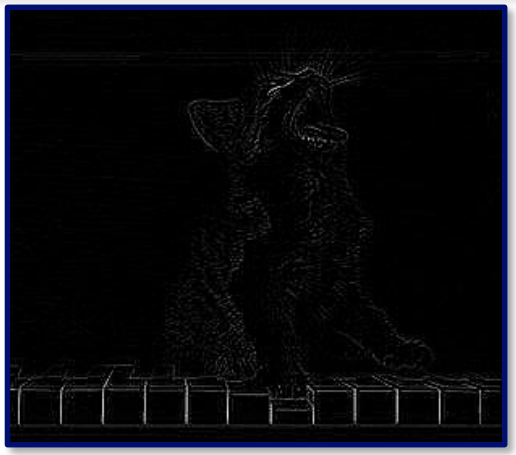
Gaussian function



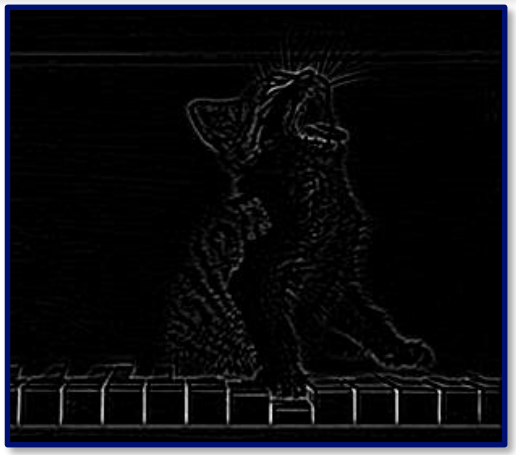
Laplacian of Gaussian

Laplacian of Gaussian

Original Image

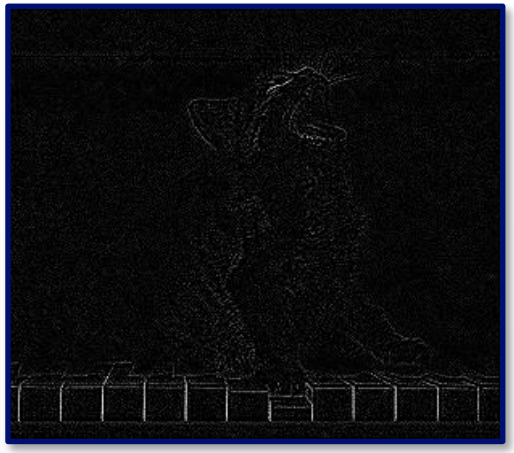


Laplacian

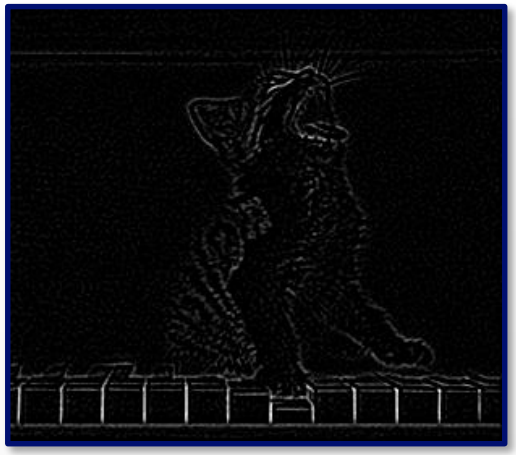


Laplacian of Gaussian

Noisy Image

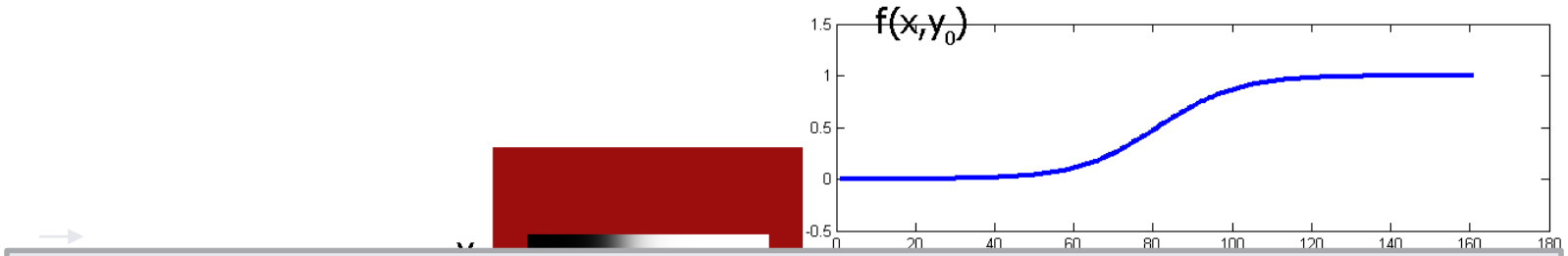


Laplacian



Laplacian of Gaussian

Edge crispening



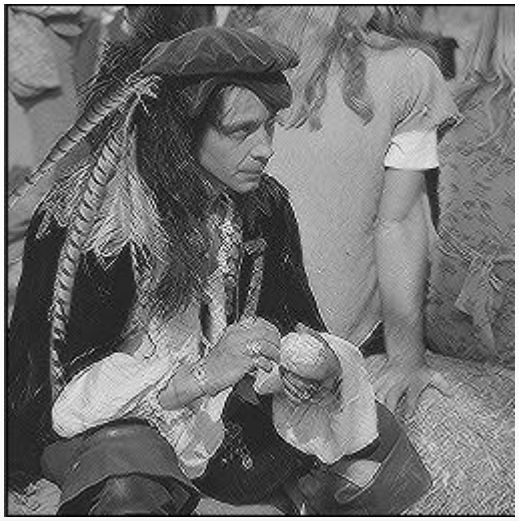
Convolution matrix:

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

Edge crispening

- Often enhances the image qualiy

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



Edge Enhancement

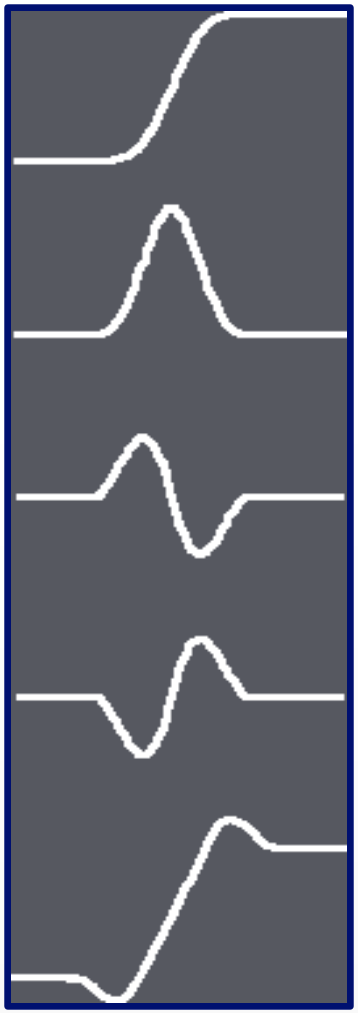
$f(x)$

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

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

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

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



Kernel for edge enhancement with Laplace operator:

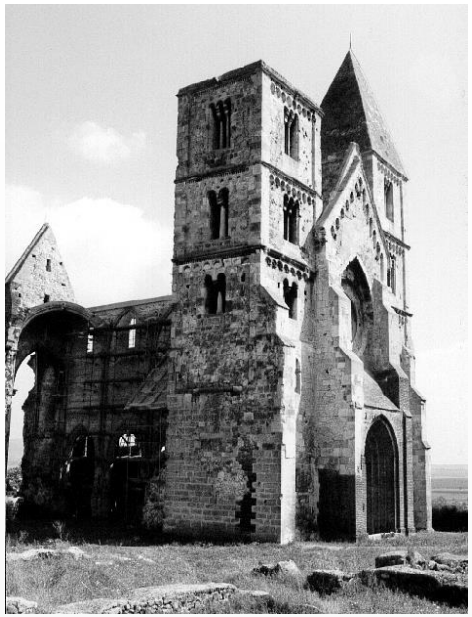
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Original image

Edge enhanced image


Knauer
BRASSÓ


Knauer
BRASSÓ



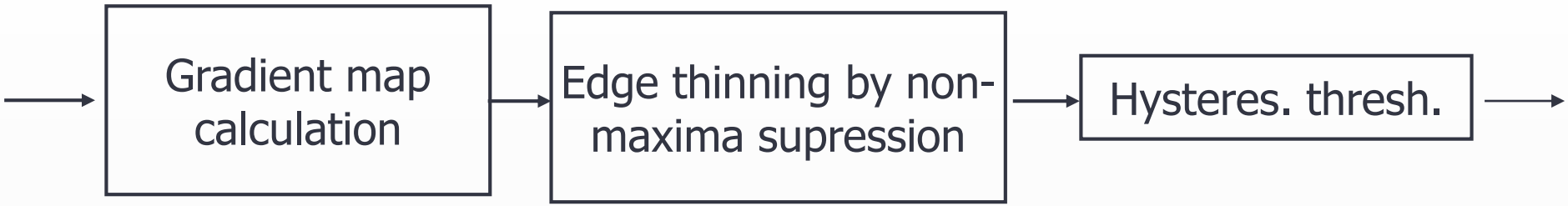
Canny edge detector

- Remember: properties of a good edge detector:
  - Good detection:
    - detects as many real edges as possible
    - does not create false edges (because of e.g. image noise)
  - 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 Edge detector

- Main steps of the algorithm:
  1. ***Noise reduction:***
     - The original image is convolved with a ***Gaussian kernel*** to reduce image noise.
  2. ***Gradient intensity and direction calculation:***
     - The horizontal and vertical derivative image is calculated (e.g. with Prewitt kernel)
     - Based on them the gradient intensity and direction can be calculated:

$$d^x = \left(\frac{\partial f}{\partial x}\right) \qquad d = \|\nabla f\| = \sqrt{\left(d^x\right)^2 + \left(d^y\right)^2}$$

$$d^y = \left(\frac{\partial f}{\partial y}\right) \qquad \Theta = \arctan\left(\frac{d^x}{d^y}\right)$$

Canny - 1st step: gradient map

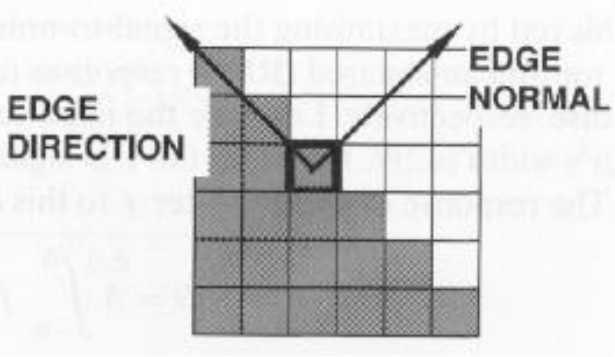
- Convolution (similarly as before)
  - Image smoothing with a Gaussian filter
  - x and y direction gradient estimation
- At each pixel (i,j) calculate gradient magnitude [d(i,j)] and direction [n(i,j) – edge normal]

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

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

Canny - 1st step: gradient map

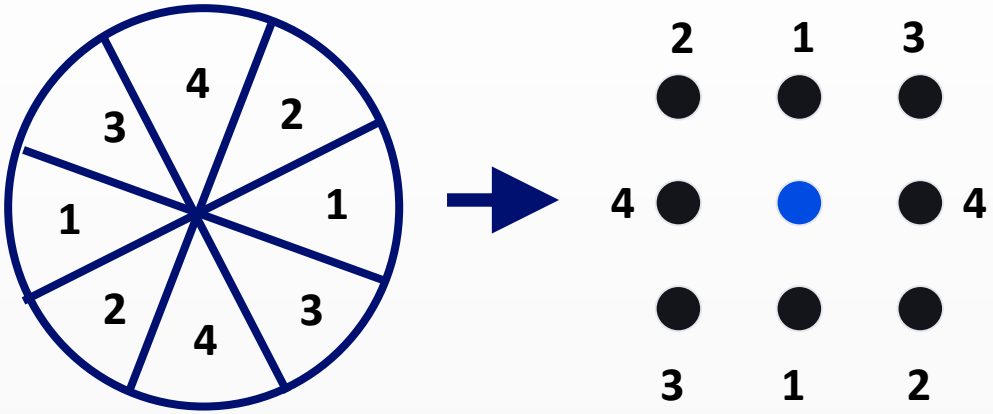
- d(i,j)= edge magnitude (how sharp is the edge – proportional to the gradient magnitude)
- n(i,j) = edge normal (which direction)



Canny Edge detector

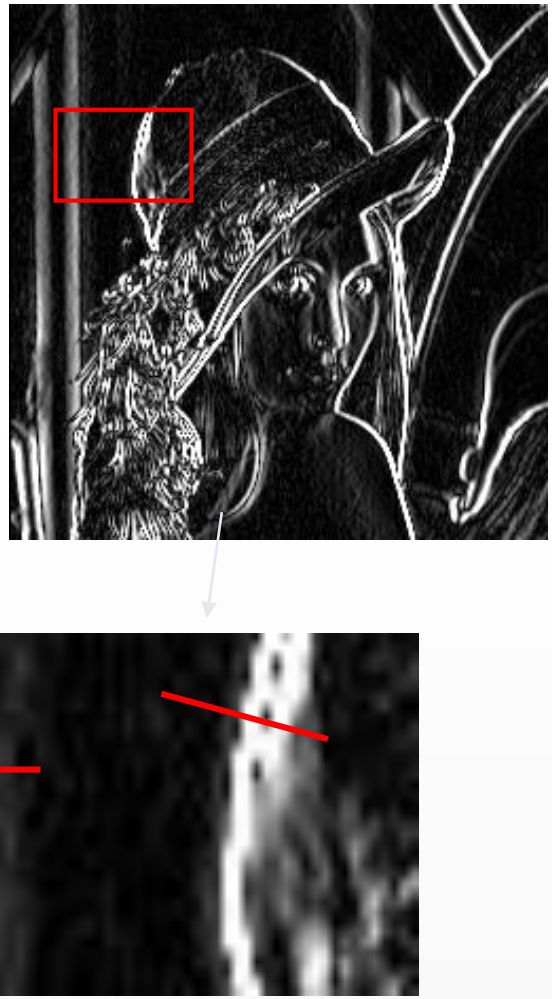
3. Non-Maximum Suppression:

- The goal is thinning the edges.
- 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:



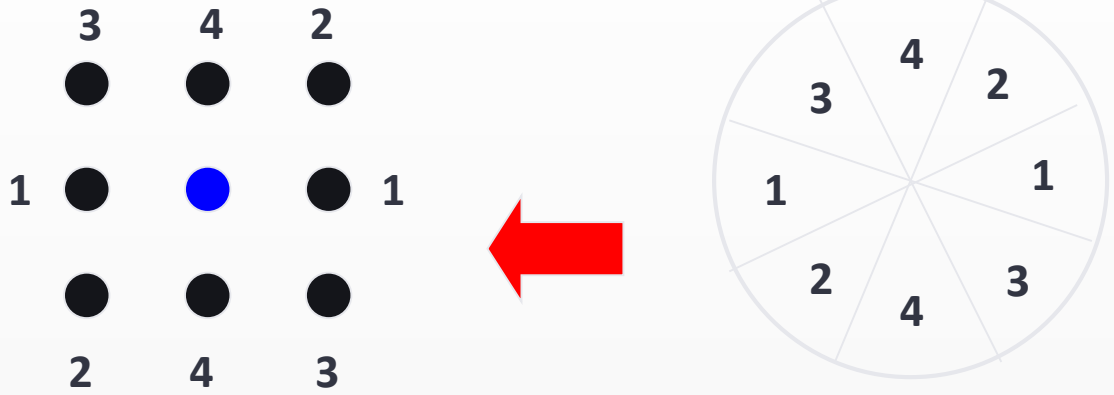
Canny – 2nd step: Non-max Suppression

- Goal: thinning the edges
- The gradient map may contain „thick” 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 **locally the brightest**



Canny – 2nd step: Non-max Suppression

1. Consider 4 principal directions: (0°, 45°, 90°, 135°)
2. To each pixel (i, j) we assign the principal direction $a(i, j)$, which one is the closest to local edge normal $n(i, j)$
3. If local edge magnitude $d(i, j)$ is smaller than in any neighboring pixel in the $a(i, j)$ set $G(i, j) := 0$. Otherwise (local max) : $G(i, j) := d(i, j)$
4. Result: G image obtained from the *d* gradient-magnitude map, where the edge-candidate regions become thin



Canny Edge detector

4. Hysteresis thresholding:

- Problem with simple thresholding:
  - if the threshold is low, many false edges will appear
  - if the threshold is high, true edges will disappear
- Solution: using two threshold instead of only one: t_1, t_2, where $t_1 > t_2$
  - if the edge magnitude at (i, j) point is higher than t_1 then it is an edge
  - if the edge magnitude at (i, j) point is lower than t_2 then it is not an edge
  - if the edge magnitude at (i, j) point is lower than t_1 but higher than $t_{2,}$ then it is an edge, only if one of its neighbors in the direction of $\theta(i, j)$ is an edge.

original image

gradient intensity

non-maximum suppression

hysteresis thresholding

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

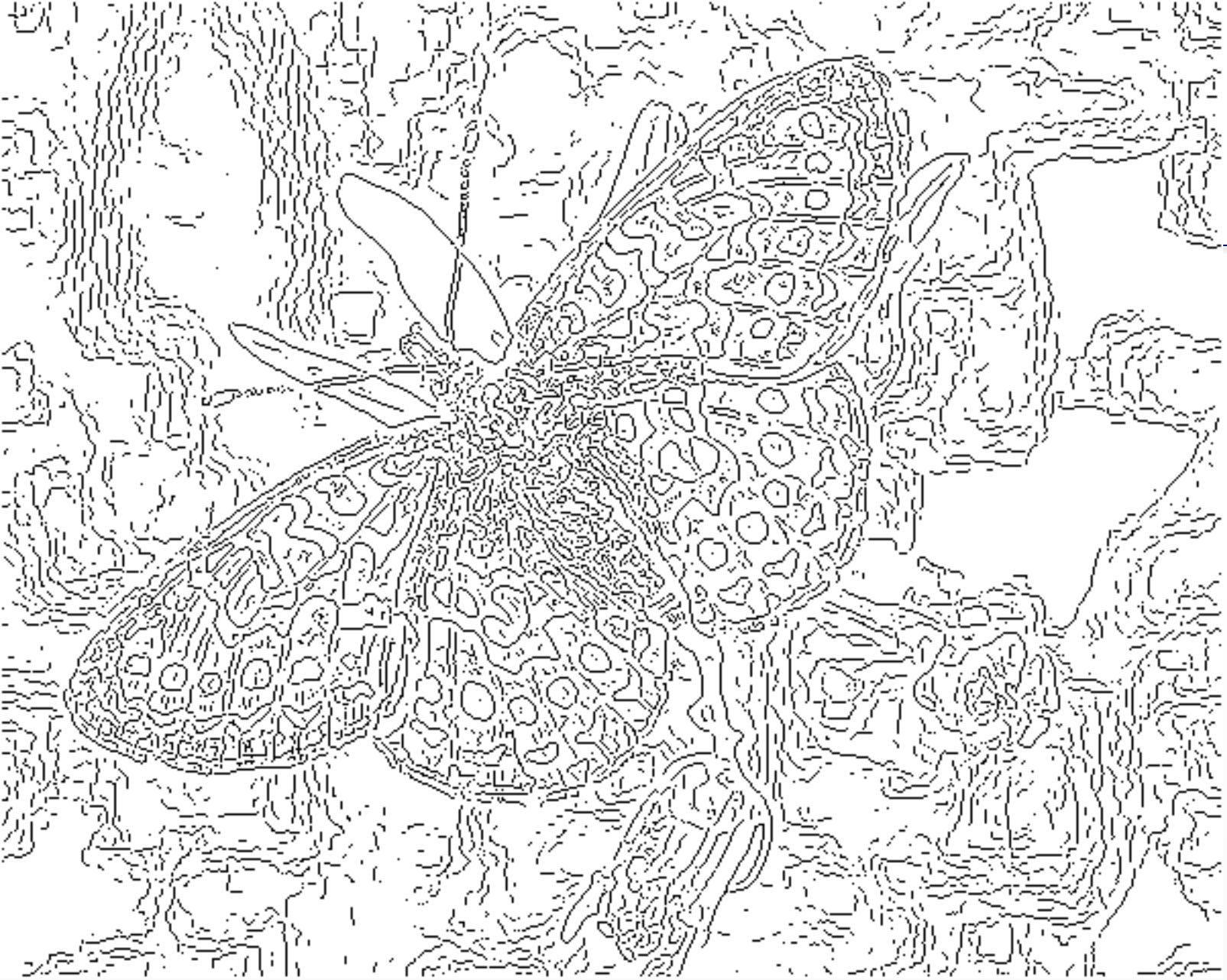
Norm of gradient: „d”

Canny edge detector - result

After thinning (E) (non-maximum suppression)

Canny edge detector - result

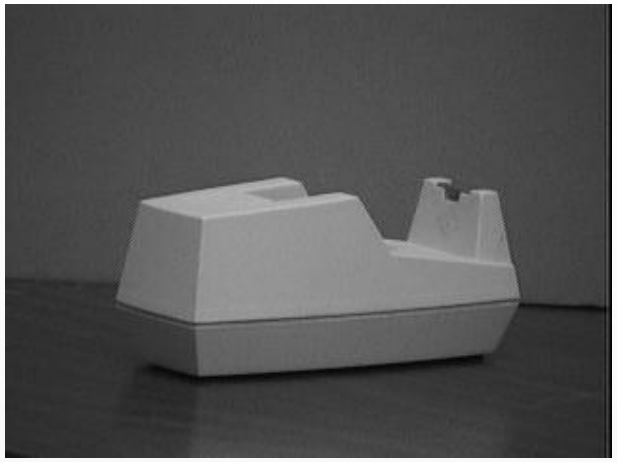
hysteresis thresholding

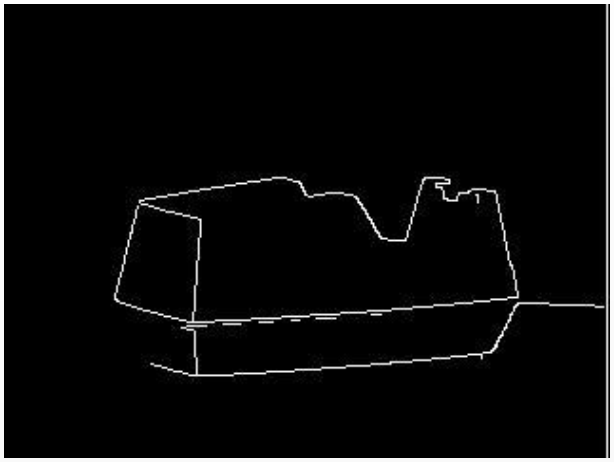


fine scale
high
threshold

coarse scale, high threshold

Canny edge detector - results





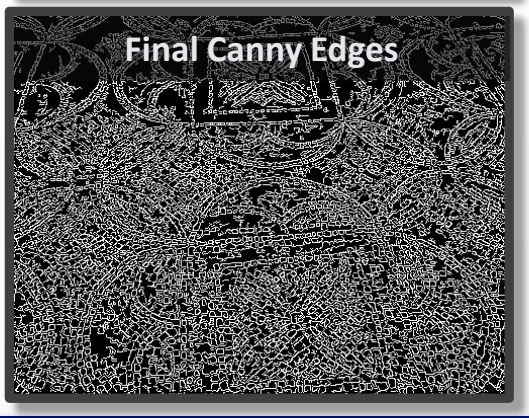

Original Image

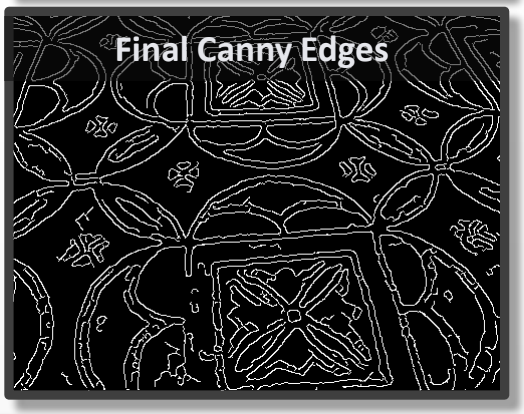

Weakly Blurred Image


Moderately Blurred Image


Gradient Magnitude


Gradient Magnitude


Final Canny Edges


Final Canny Edges