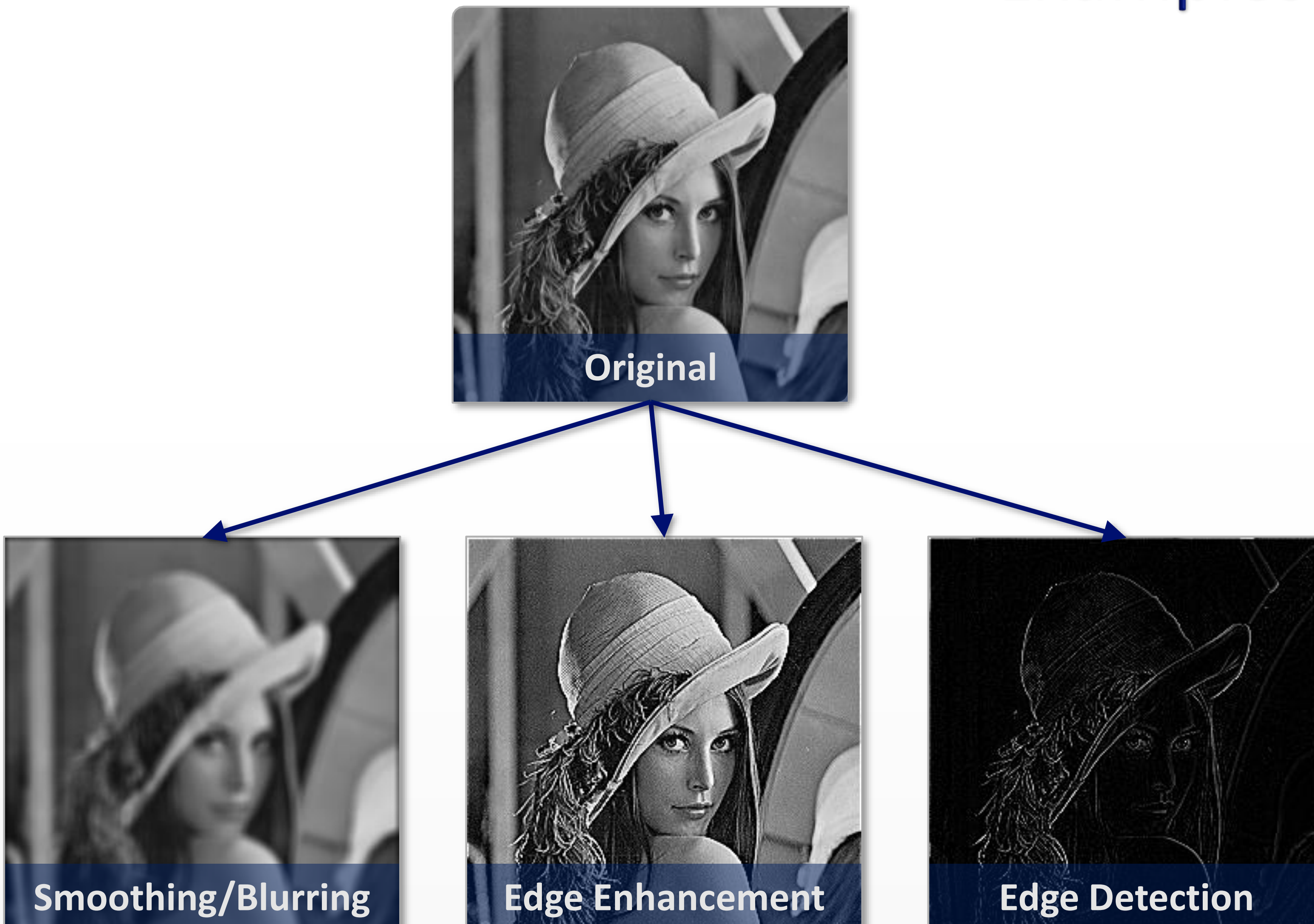
# Basic Image Processing

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: $f(x,y)$
  - $x$ and $y$ are the pixel coordinates
  - $f$ is a gray level from $[0,255]$

Image $f$

- We can define different transformations:
  - Intensity value inversion:

$$g(x,y) = 255 - f(x,y)$$

Image $g$

# Mathematical background

- We look at the image as a 2D function: $f(x, y)$
  - $x$ and $y$ are the pixel coordinates
  - $f$ is a gray level from $[0,255]$

Image $f$

- We can define different transformations:
  - Intensity shift with constant:
    $$g(x, y) = f(x, y) + 100$$

Image $g$

# Mathematical background

- We look at the image as a 2D function: $f(x, y)$
  - $x$ and $y$ are the pixel coordinates
  - $f$ is a gray level from $[0,255]$

Image $f$

- We can define different transforms:
  - Weighting :

$$g(x, y) = f(x, y) \cdot w(x, y)$$

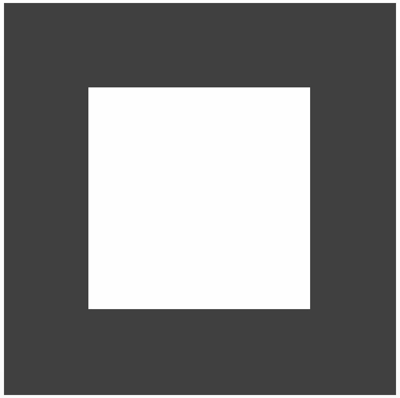


Image $w$

$w(x, y) \in [0.5,2]$

Image $g$

# Mathematical background

- We look at the image as a 2D function: $f(x, y)$
  - $x$ and $y$ are the pixel coordinates
  - $f$ is a gray level from $[0,255]$

Image $f$

- We can define different transformations:
  - Average on an $N$ neighborhood :

    $f(x, y) = \text{average } N(f(x, y))$

Image $g$

# Mathematical background

- We look at the image as a 2D function: $f(x,y)$
- We can define different transformations:
  - Intensity value inversion: $g(x,y) = 255 - f(x,y)$
  - Intensity shift with constant: $g(x,y) = f(x,y) + 100$
  - Weighting: $g(x,y) = f(x,y) \cdot w(x,y)$
  - Average on an *N* neighborhood: $g(x,y) = \text{average } N\big(f(x,y)\big)$
- *In this lecture*, there are two important properties of the transformations we want to use on images: ***linearity*** and ***shift invariance***

# Mathematical background

- Linearity:

$$T[f_1(x,y)+f_2(x,y)] = T[f_1(x,y)] + T[f_2(x,y)]$$
$$T[\alpha \cdot f(x,y)] = \alpha \cdot T[f(x,y)]$$

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

- Spatial Invariance (SI): for any $[k,l]$ spatial shift vector,

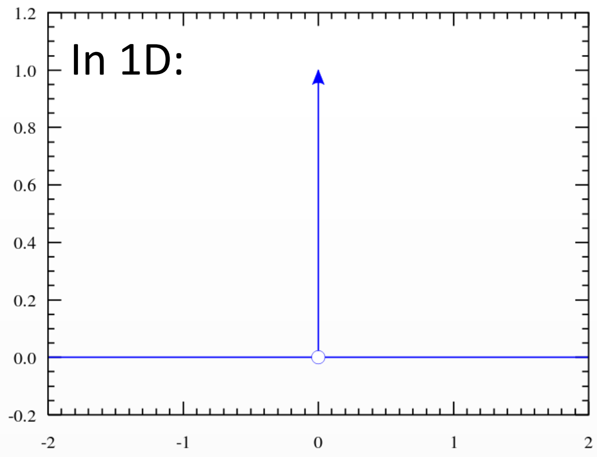
$$T[f(x,y)] = g(x,y)$$
$$T[f(x-k,y-l)] = g(x-k,y-l)$$

  - e.g.: weighting is not SI, intensity inversion is SI
  - e.g.: averaging on neighborhood is both linear and SI, we call it LSI

# Unit Impulse Function

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

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




- For any 2D function $f(x,y)$:

$$f(x,y)=\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}\delta(x-k,y-l)\cdot f(k,l)$$

# Convolution

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

  If $T$ is an LSI system:

  $$T[f(x,y)] = g(x,y)$$

  Then we can define convolution as follows:

  $$\begin{aligned} g(x,y) &= f(x,y) * h(x,y) = \\ &= h(x,y) * f(x,y) = \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(k,l) \cdot h(x-k, y-l) \end{aligned}$$

# Derivation of Convolution

$$g(x,y) = T\left[f(x,y)\right] =$$

$$= T\left[\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} f(k,l)\delta(x-k,y-l)\right] =$$

Linearity

$$= \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} f(k,l)\cdot T\left[\delta(x-k,y-l)\right] =$$

$$= \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} f(k,l)\cdot h(x-k,y-l)$$

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 for image processing

$$\underbrace{g}_{\text{output image}} = \overbrace{h}^{\text{convolution kernel}} * \underbrace{f}_{\text{input image}}$$

- In practice both the $h$ kernel and the $f$ image have finite size.
- Typically the size of $h$ is much smaller than the image size ($3 \times 3$, $5 \times 5$, $5 \times 7$ etc.)

# 2D convolution in Practice

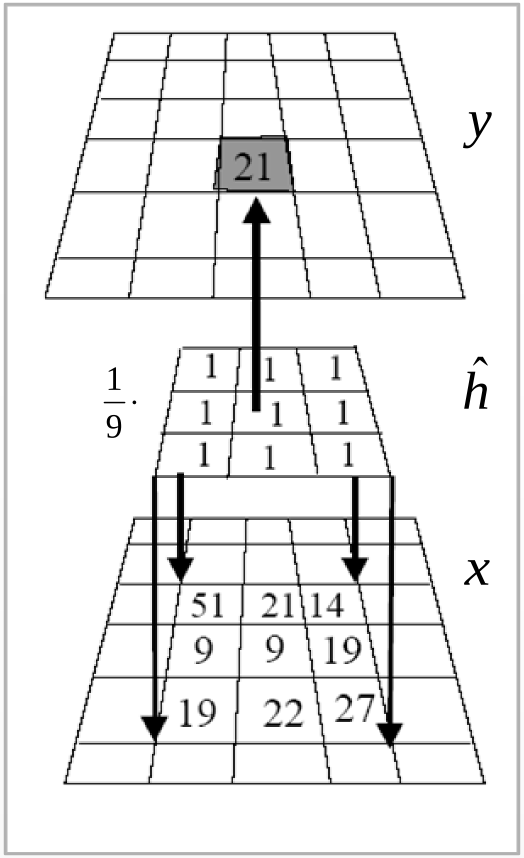
- Let $h$ and $\hat{h}$ be $(2r_1 + 1) \times (2r_2 + 1)$ sized kernels where $\hat{h}$ is the rotated version of $h$ with 180°

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

$$g(x,y) = \sum_{l=-r_1}^{r_1} \sum_{l=-r_2}^{r_2} f(k,l) \cdot h(x-k, y-l) =$$

$$= \sum_{k=-r_1}^{r_1} \sum_{l=-r_2}^{r_2} h(k,l) \cdot f(x-k, y-l) =$$

$$\boxed{= \sum_{k=-r_1}^{r_1} \sum_{l=-r_2}^{r_2} \hat{h}(k,l) \cdot f(x+k, y+l)}$$



# 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: $A \times B$

Size of the kernel: $C \times D$

Size of the output image: $(A + C - 1) \times (B + D - 1)$

# 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:

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

Original




$$h = \frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

# Bluring for noise filtering

Noisy image

Result of bluring

# Computational requirements

- For $k_s$ kernel size and $P$ image size (area, measured in pixels) approximately $\sim k_s P$ 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: $t(x,y) = \sum_{j=1}^{y} f(x,j)$
  - Calculation of image $t$:

$$t(x,1) \coloneqq f(x,1), x = 1 \dots w; \quad t(x,y) = t(x,y-1) + f(x,y)$$

  - Calculation of $I_f$ using image $t$:

$$I_f(1,y) \coloneqq t(1,y), y = 1 \dots h; \quad I_f(x,y) = I_f(x-1,y) + t(x,y)$$

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.

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

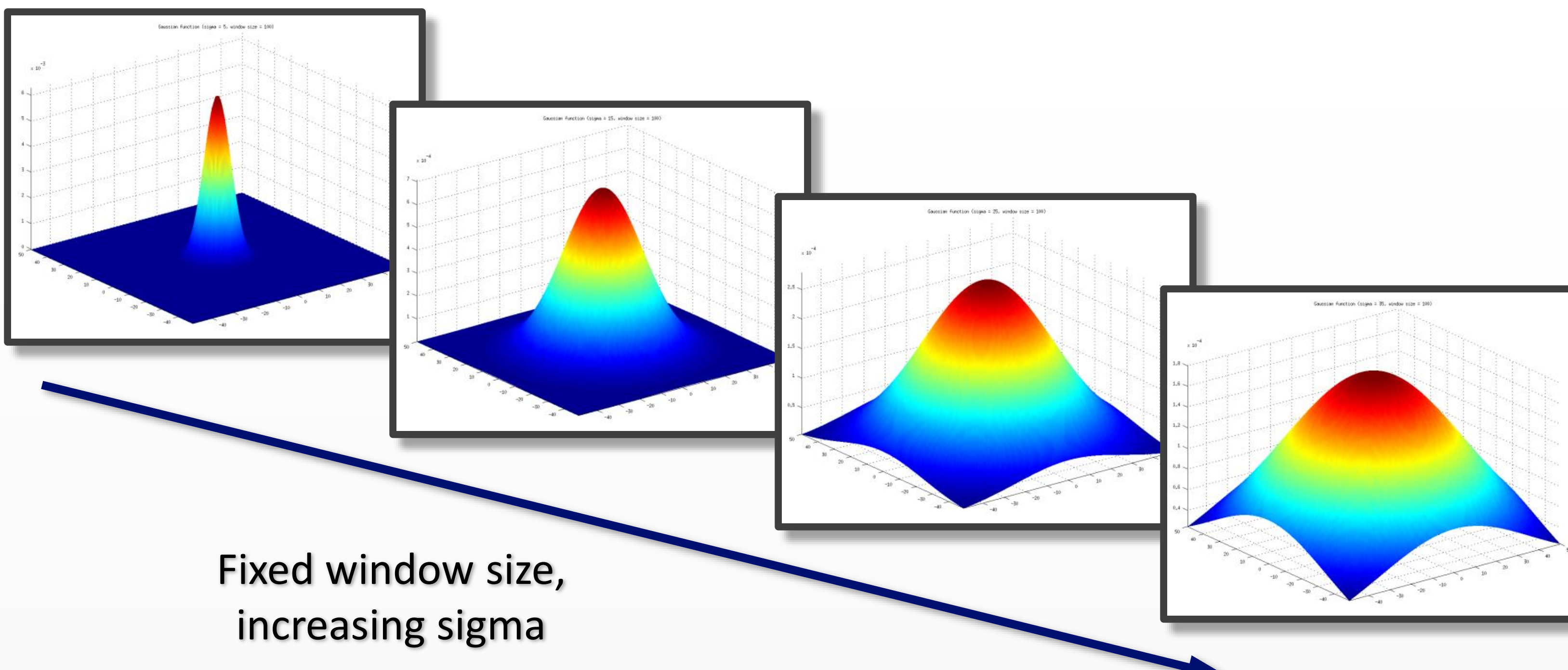
where: $$\tilde{f}(x,y) = \frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r} f(x+i,y+j)$$

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



Fixed window size, increasing sigma

# 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;

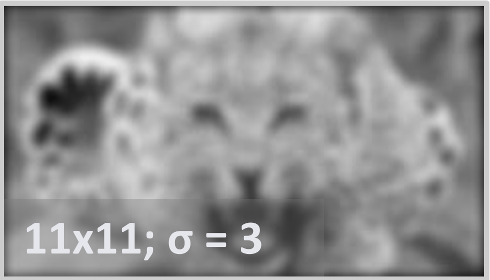
$$\begin{bmatrix} 0.0947 & 0.1183 & 0.0947 \\ 0.1183 & 0.1478 & 0.1183 \\ 0.0947 & 0.1183 & 0.0947 \end{bmatrix}$$

# Smoothing/Blurring

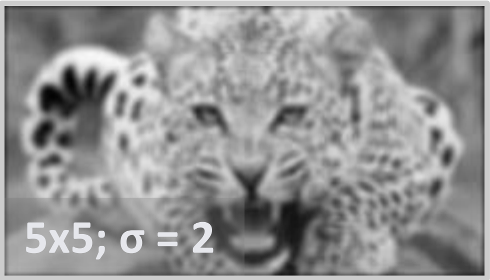
- Gaussian blur:



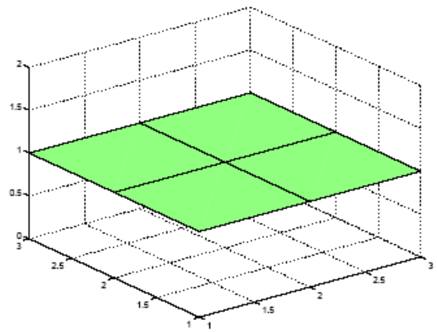


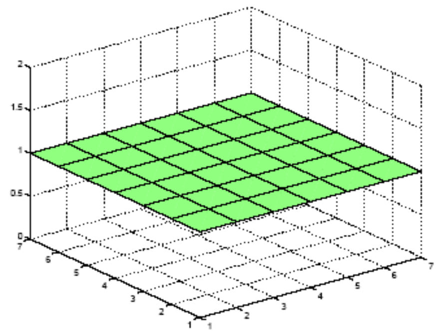








# Convolution examples – averaging blur

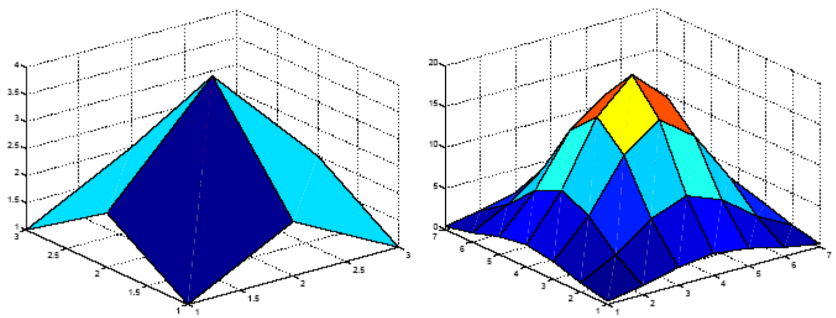




Input Image

Average blur

# Convolution examples– Gaussian blur



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

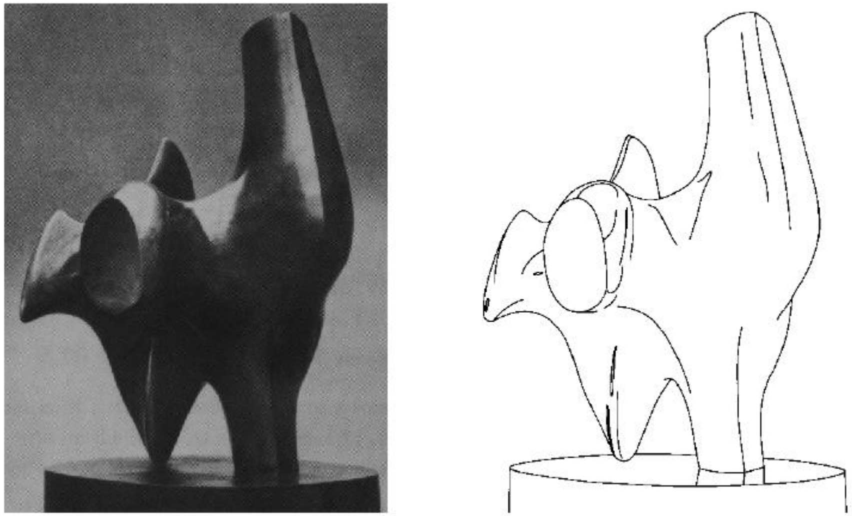
Input Image

Gaussian blur

# Edge detection

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

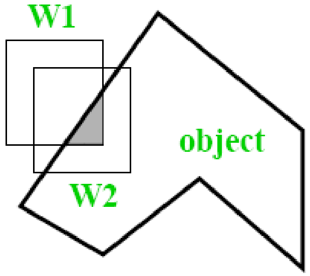
# Goals of edge detection



- 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...

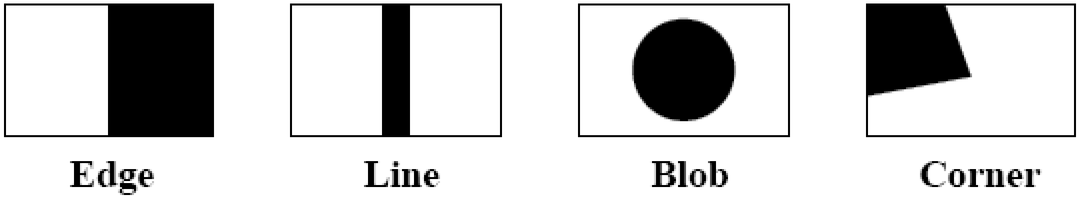



# Edge detection

- 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

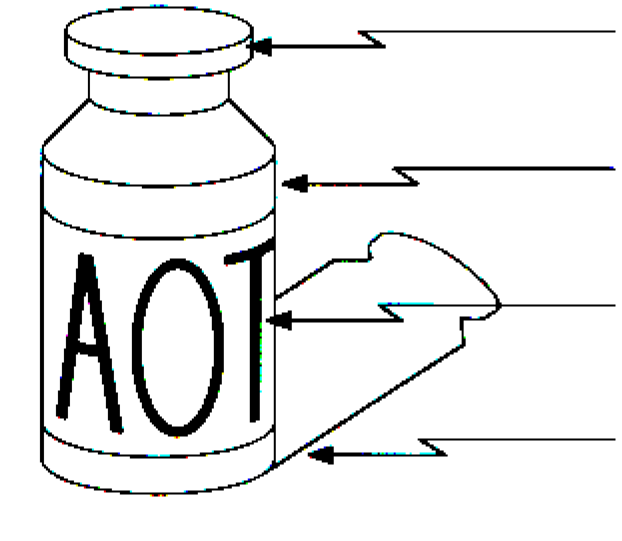
# Basic structures

- **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 and types of edges

- Various effects may cause edges




Sharp change in surface normals

Continuos change in surface depth

Change in surface color

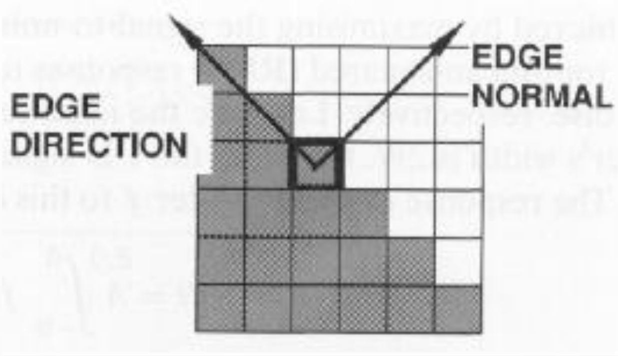
Changes cased by illumination/shadows

- Basic edge types

step ramp roof line

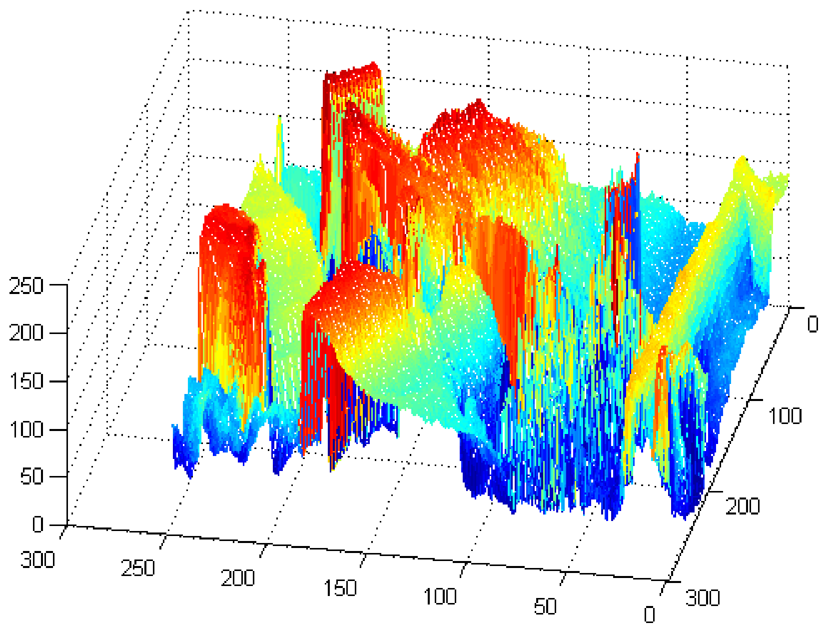
# Parameters of an edge

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



# Image representation

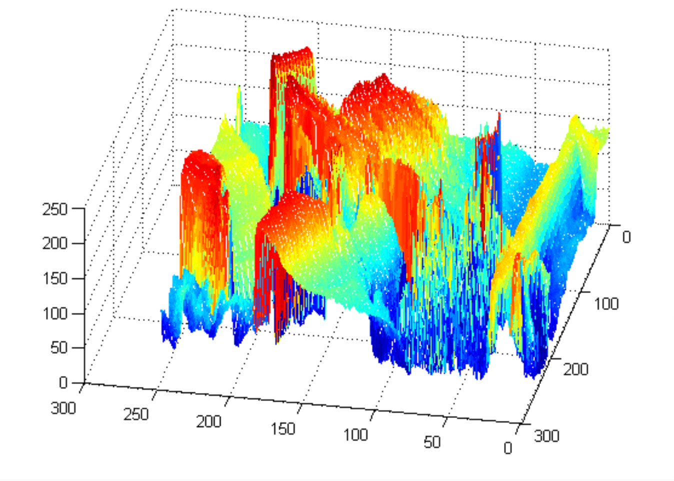
- Image: gray value is function of the *x* and *y* coordinates (intensity function): $f(x, y)$





# Edge Detection

- ***Edge:*** 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:

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

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

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

# Edge Detection

- **Approximation of the $x$ directional partial derivative:**
  - *For better localization*, use a symmetric formula around pixel $(x, y)$

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

    - Corresponding convolutional kernel:

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

  - *For noise reduction*, apply $y$ directional smoothing (i.e. do not blur a sharp vertical edge)

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

$x$ directional Prewitt operator: **vertical** edge detector

# Edge Detection

- **Approximation of the $y$ directional partial derivative:**
  - *For better localization*, use a symmetric formula around pixel $(x, y)$

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

    - Corresponding convolutional kernel: $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

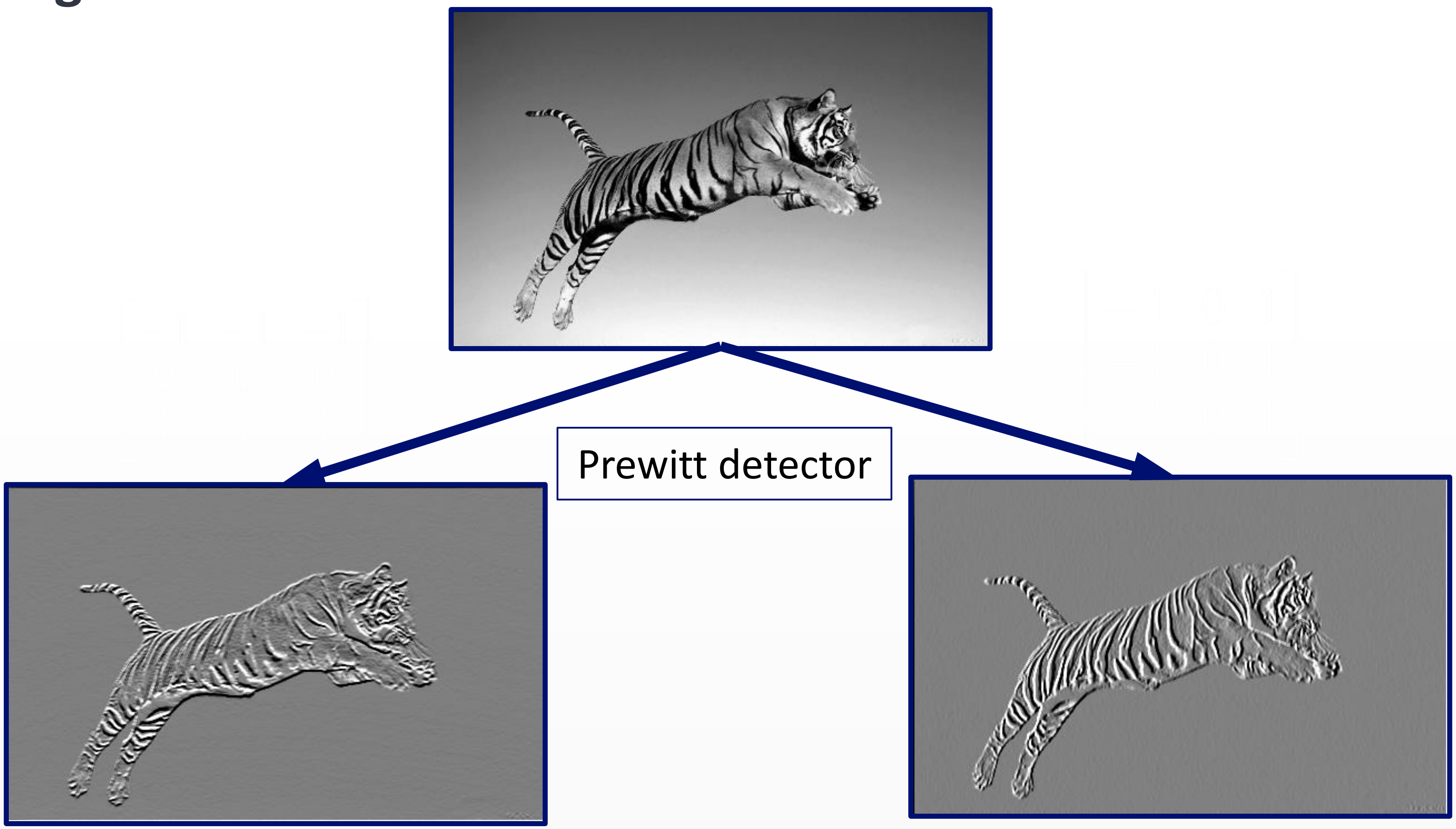
  - *For noise reduction*, apply $x$ directional smoothing (i.e. do not blur a sharp horizontal edge)

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

$y$ directional Prewitt operator: **horizontal** edge detector

# Edge Detection

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

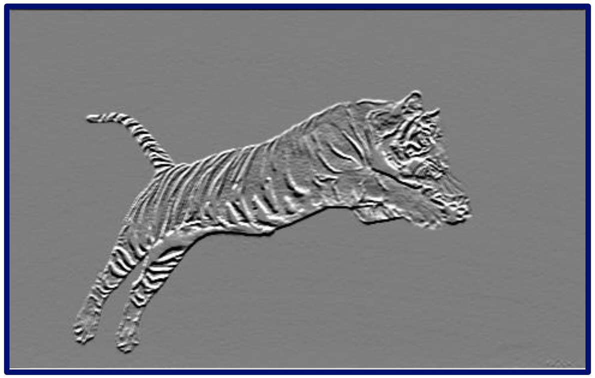



Horizontal gradient image

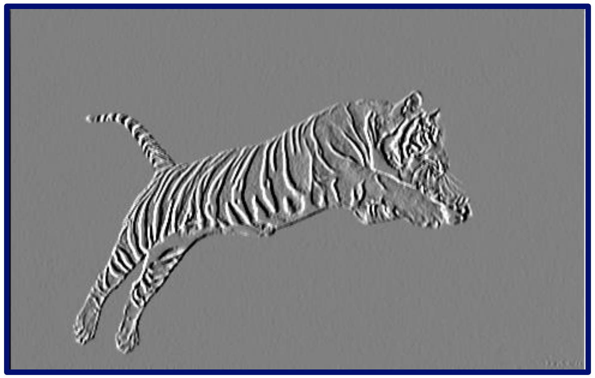
Vertical gradient image

# Edge Detection

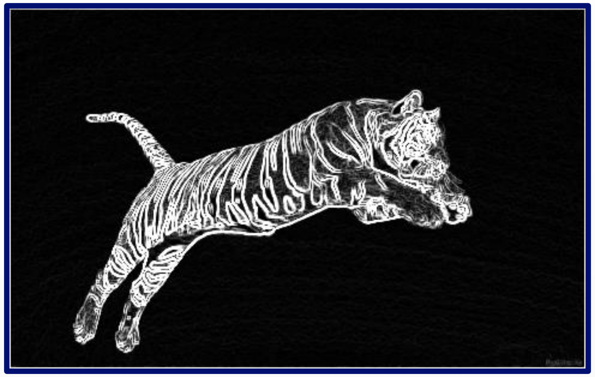
- **Edge detection with first order derivative:**



Horizontal gradient image



Vertical gradient image



gradient image

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

# Other first-order methods

- Sobel operator

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

$\partial/\partial x$

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

$\partial/\partial y$

- Roberts operator

| | |
|---|---|
| +1 | |
| | -1 |

$g_1$

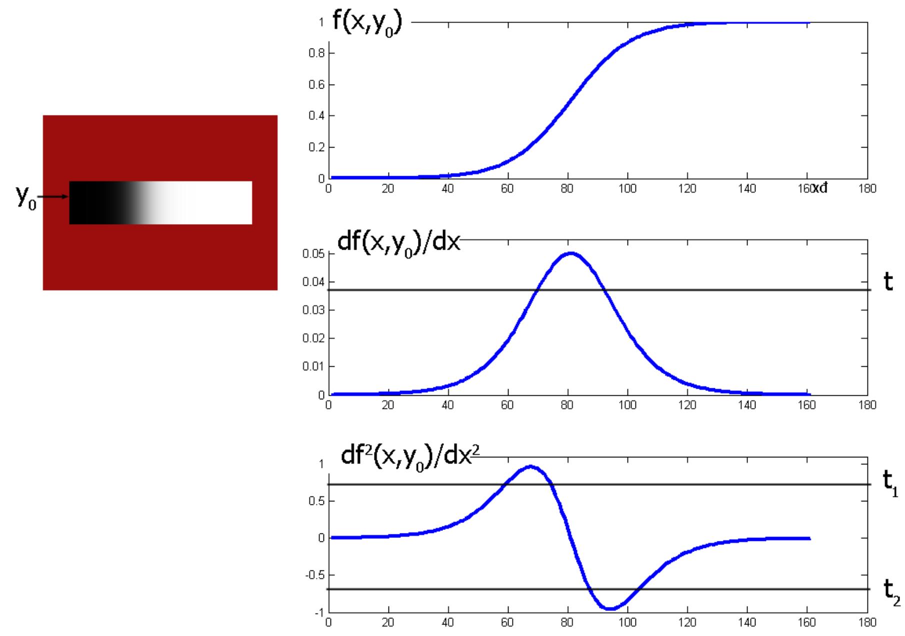
| | |
|---|---|
| | +1 |
| -1 | |

$g_2$

Emphasize edges with 45 degree slopes

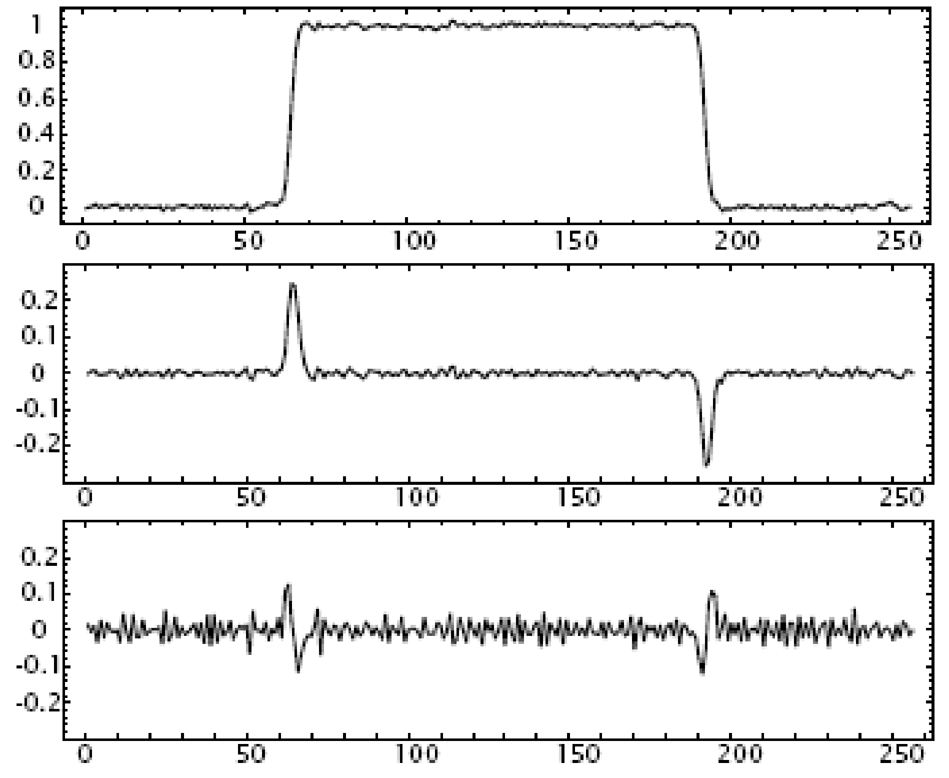
# Second order edge detection: motivation

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

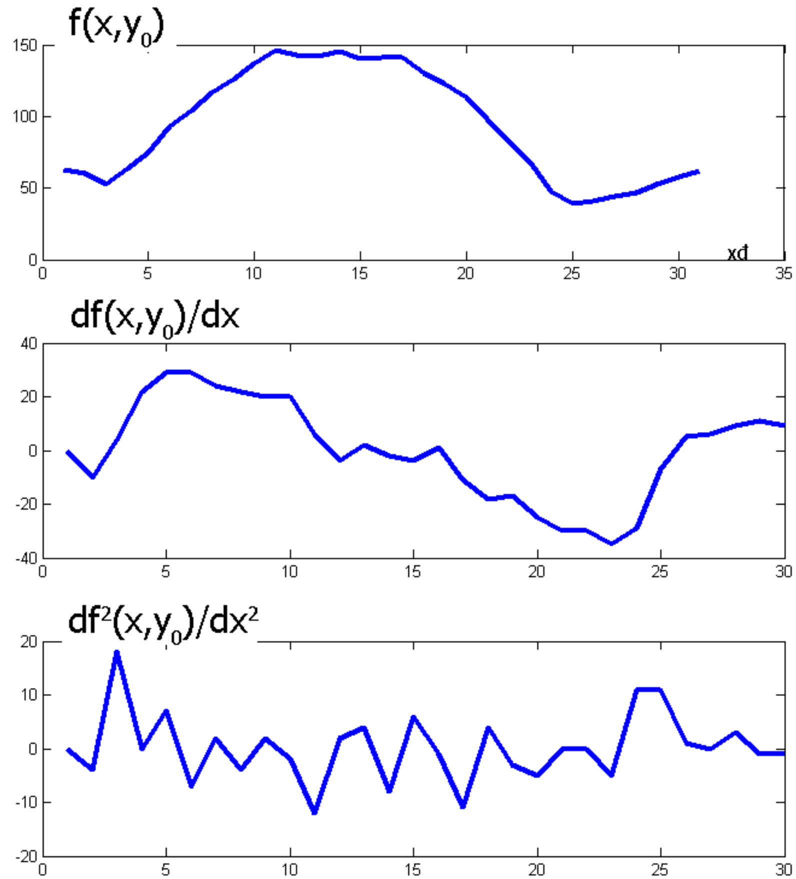
# Second order case: instead of extreme values, search for zero crossing




Extreme values (first order)

zero crossing (second order)

# Real photo: intensity profile below the red line segment



# Edge detection with second order derivative

- Calculating the divergence of the gradient vector

$$\nabla^2 f = \left(\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial y}\right) \cdot \nabla f = \frac{\partial^2}{\partial x^2} f + \frac{\partial^2}{\partial y^2} f$$

- Approximation for $x$ direction:

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

$$= \frac{1}{v^2} \cdot [f(x+1,y) - 2f(x,y) + f(x-1,y)]$$

just a constant – $v$: distance of neighboring pixel centers

# Edge detection with second order derivative

- Approximation of the second order derivatives for $x$ and $y$ directions

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

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

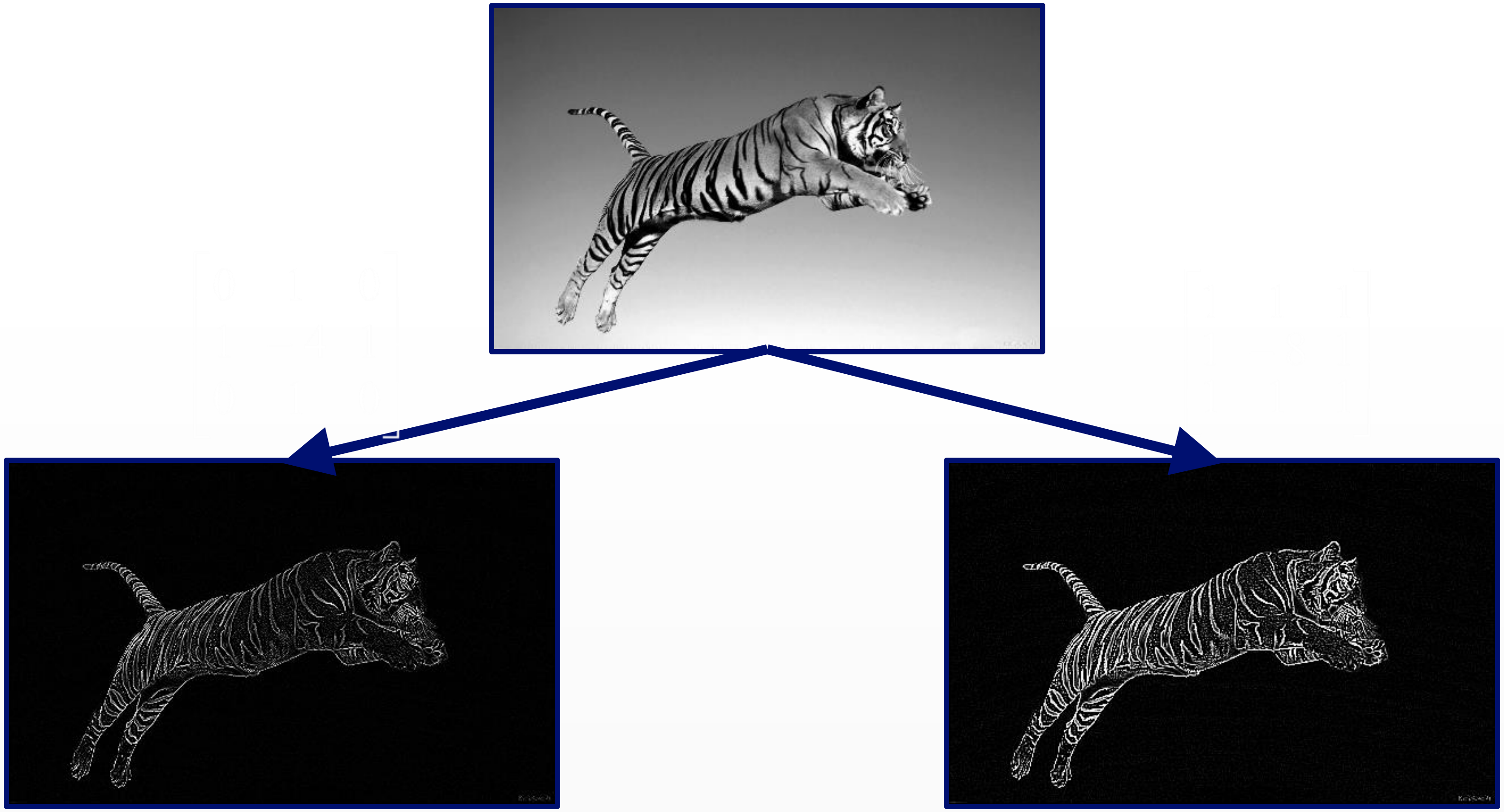
- Kernel for the second order gradient calculation with convolution:
  - Laplace operator:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \Rightarrow \qquad \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} = \boxed{\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}}$$

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

# Edge Detection

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

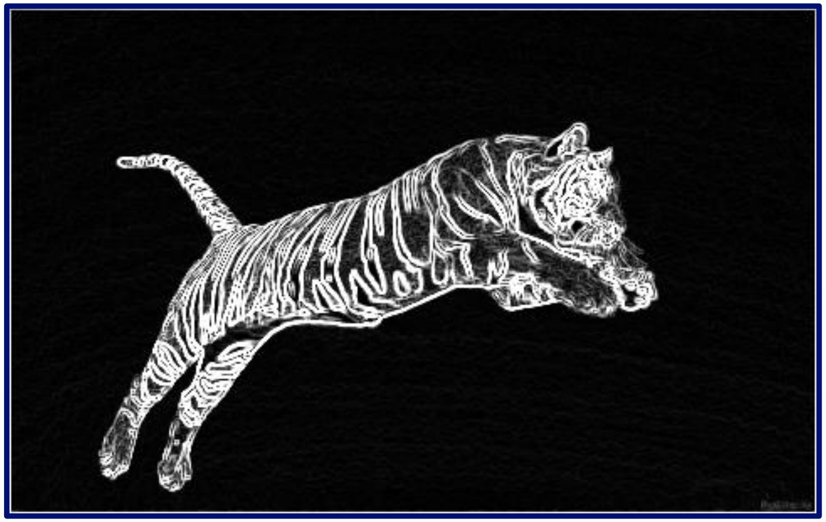


Laplace edge detector

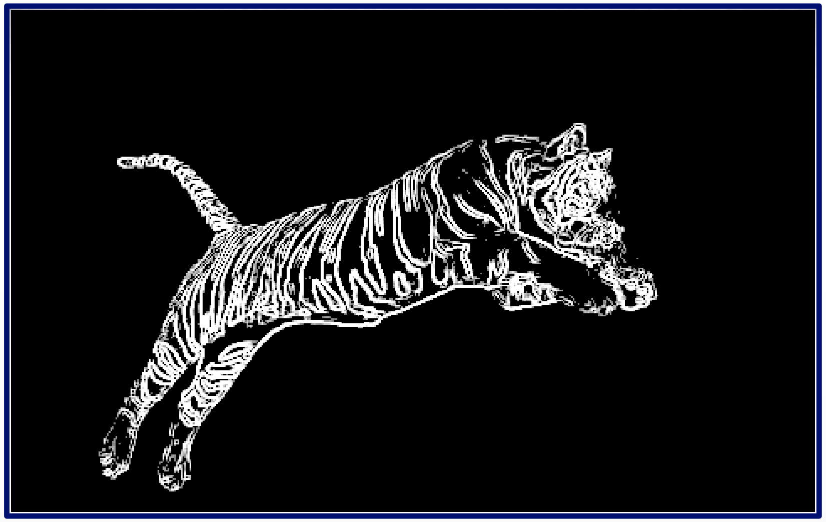
Prewitt 2nd order detector

# Edge Detection

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



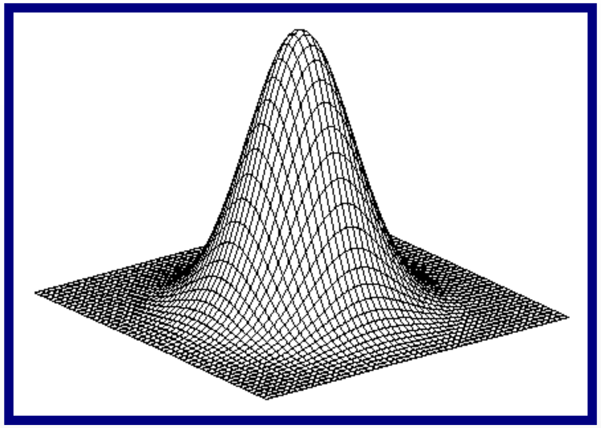
Prewitt first order gradient image



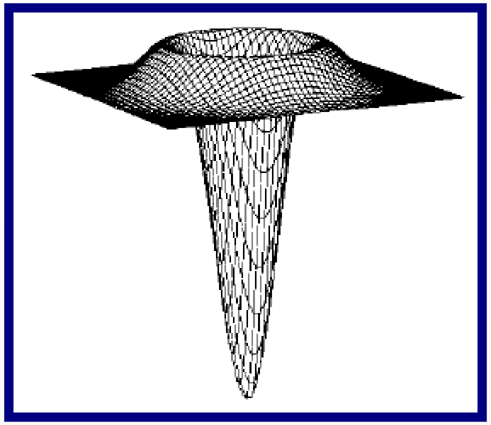
Prewitt first order gradient image
with threshold = 120

# Coming next week:
# 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