University of Zurich[UZH] | ETH zürich

Institute of Informatics – Institute of Neuroinformatics

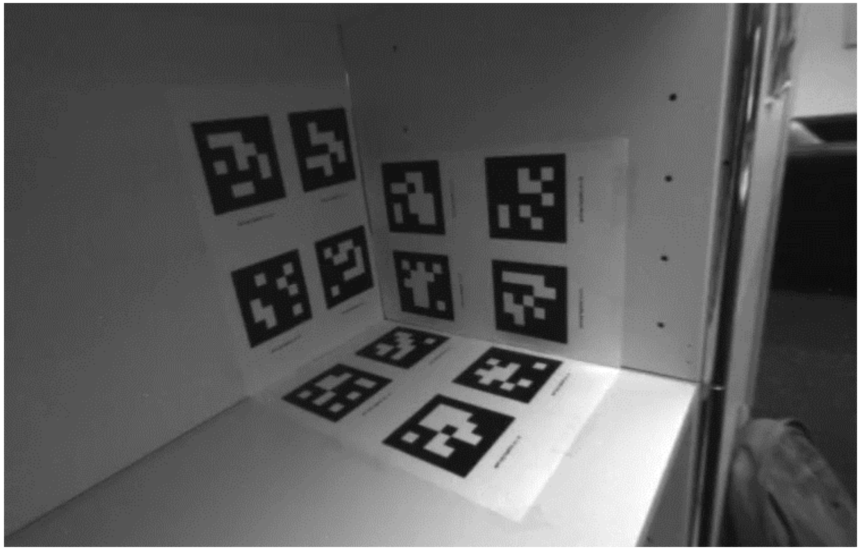
ROBOTICS & PERCEPTION GROUP

Lecture 04
Image Filtering

Davide Scaramuzza

Lab Exercise 2 - Today afternoon

- Room ETH HG E 1.1 from 13:15 to 15:00
- Work description: your first camera motion estimator using DLT



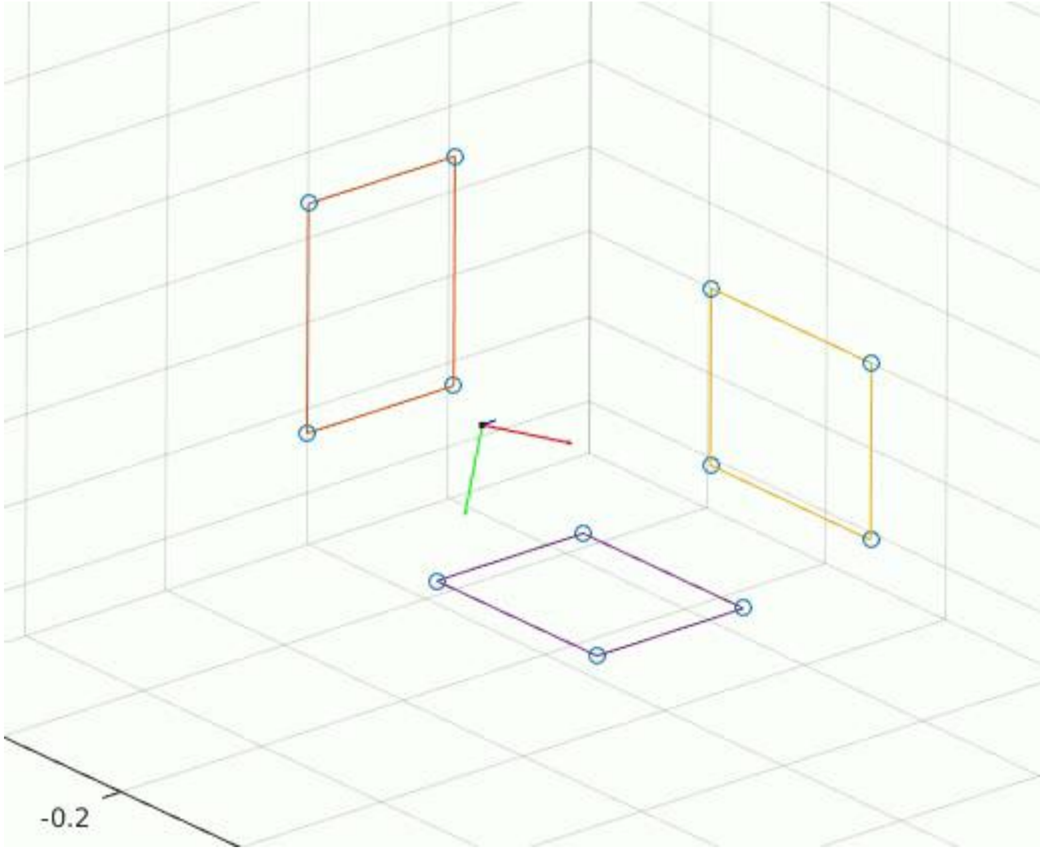
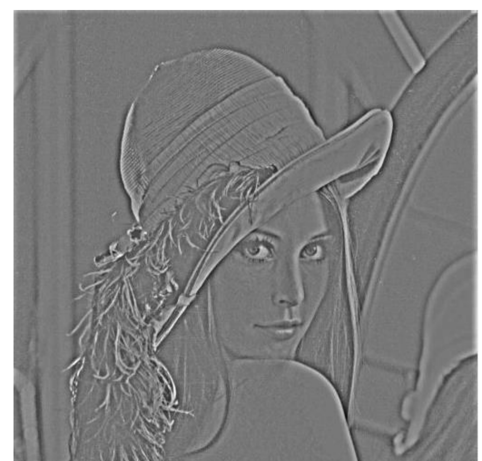


Image filtering

- The word *filter* comes from frequency-domain processing, where “filtering” refers to the process of accepting or rejecting certain frequency components
- We distinguish between low-pass and high-pass filtering
  - A **low-pass filter** smooths an image (retains low-frequency components)
  - A **high-pass filter** retains the contours (also called edges) of an image (high frequency)

Low-pass filtered image

High-pass filtered image



Low-pass filtering

Low-pass filtering
Motivation: noise reduction

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution


Original

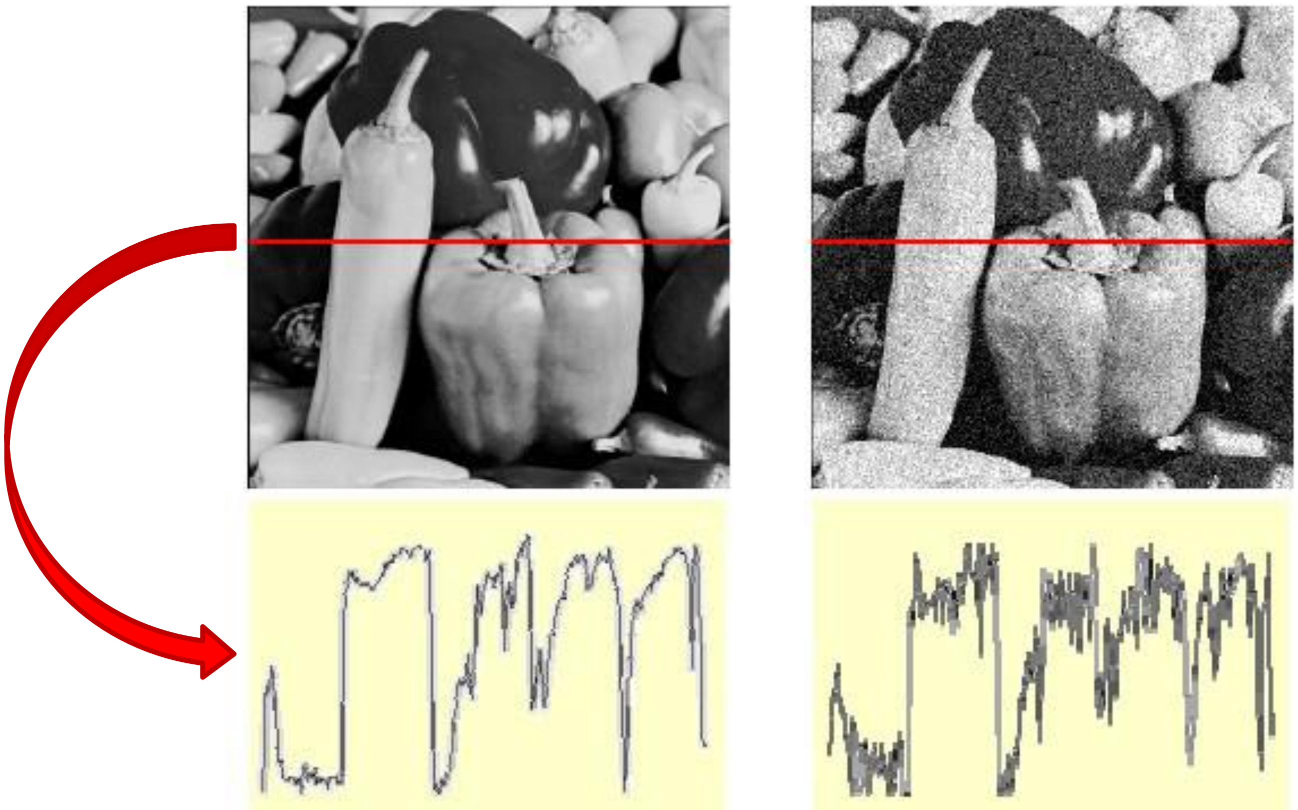

Salt and pepper noise


Impulse noise


Gaussian noise

Source: S. Seitz

Gaussian noise



$$f(x,y) = \overbrace{\tilde{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

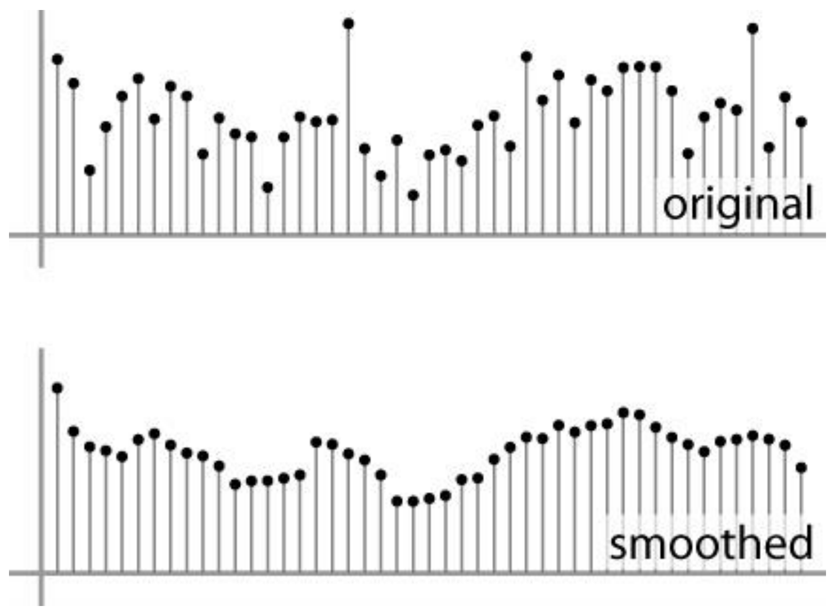
How could we reduce the noise to try to recover the "ideal image"?

Moving average

- Replaces each pixel with an average of all the values in its neighborhood
- Assumptions:
  - Expect pixels to be like their neighbors
  - Expect noise process to be independent from pixel to pixel

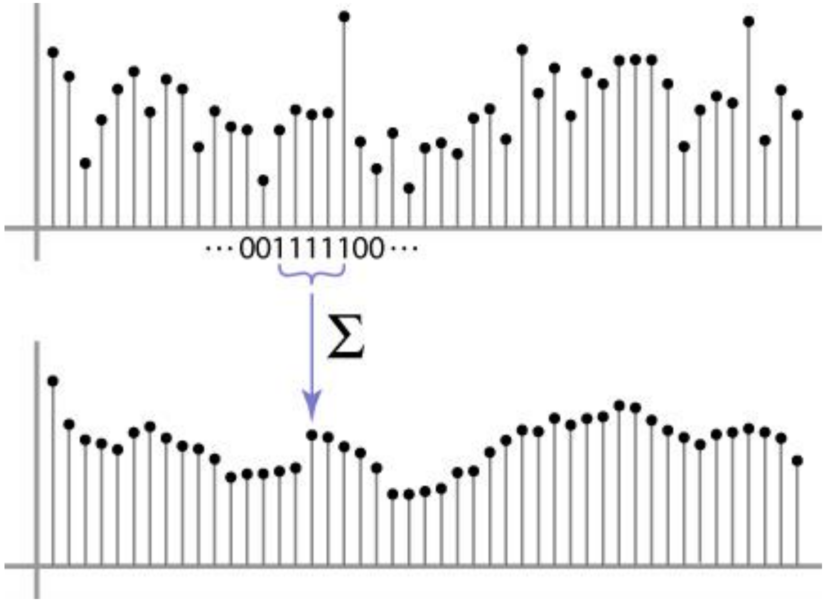
Moving average

- Replaces each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



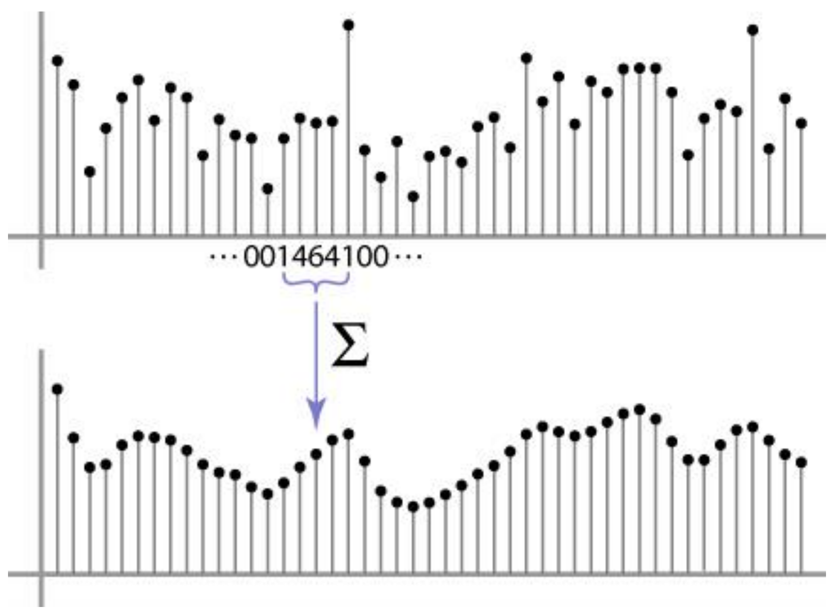
Weighted Moving Average

- Can add weights to our moving average
- *Weights* [1, 1, 1, 1, 1] / 5



Weighted Moving Average

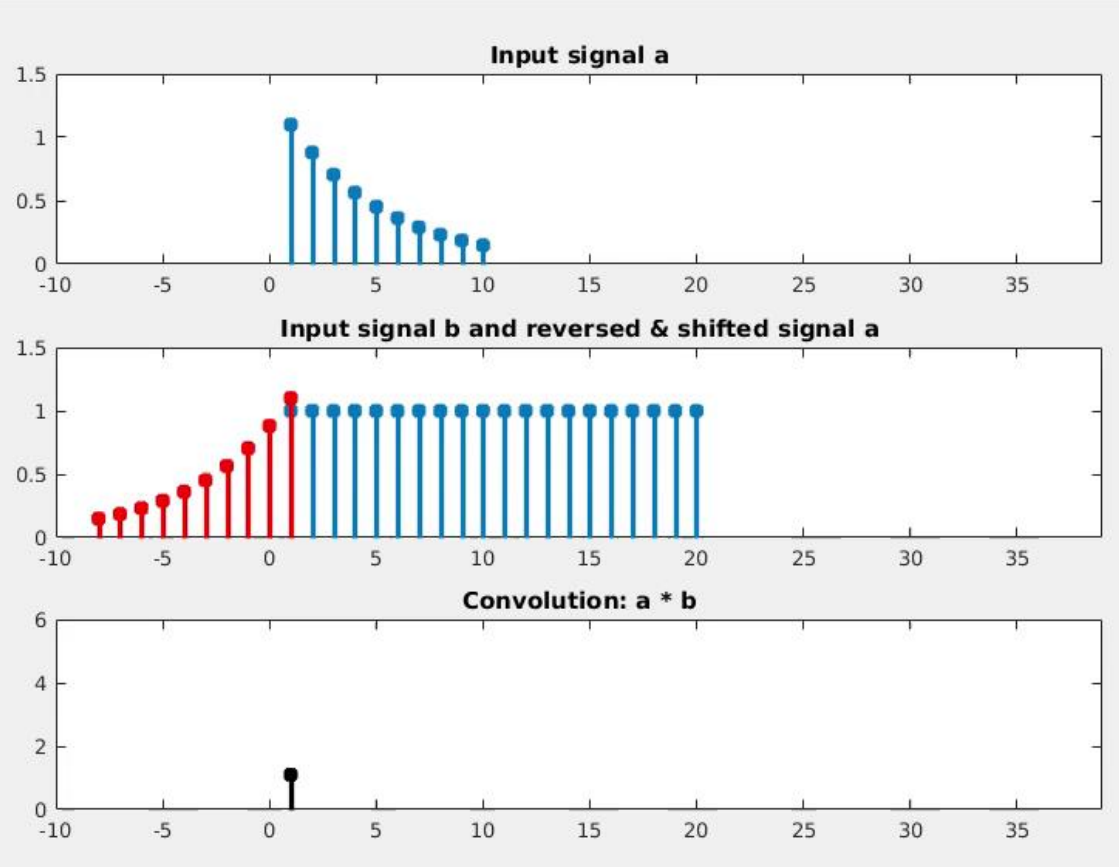
- Non-uniform weights [1, 4, 6, 4, 1] / 16



This operation is called *convolution*

Example of convolution of two sequences (or “signals”)

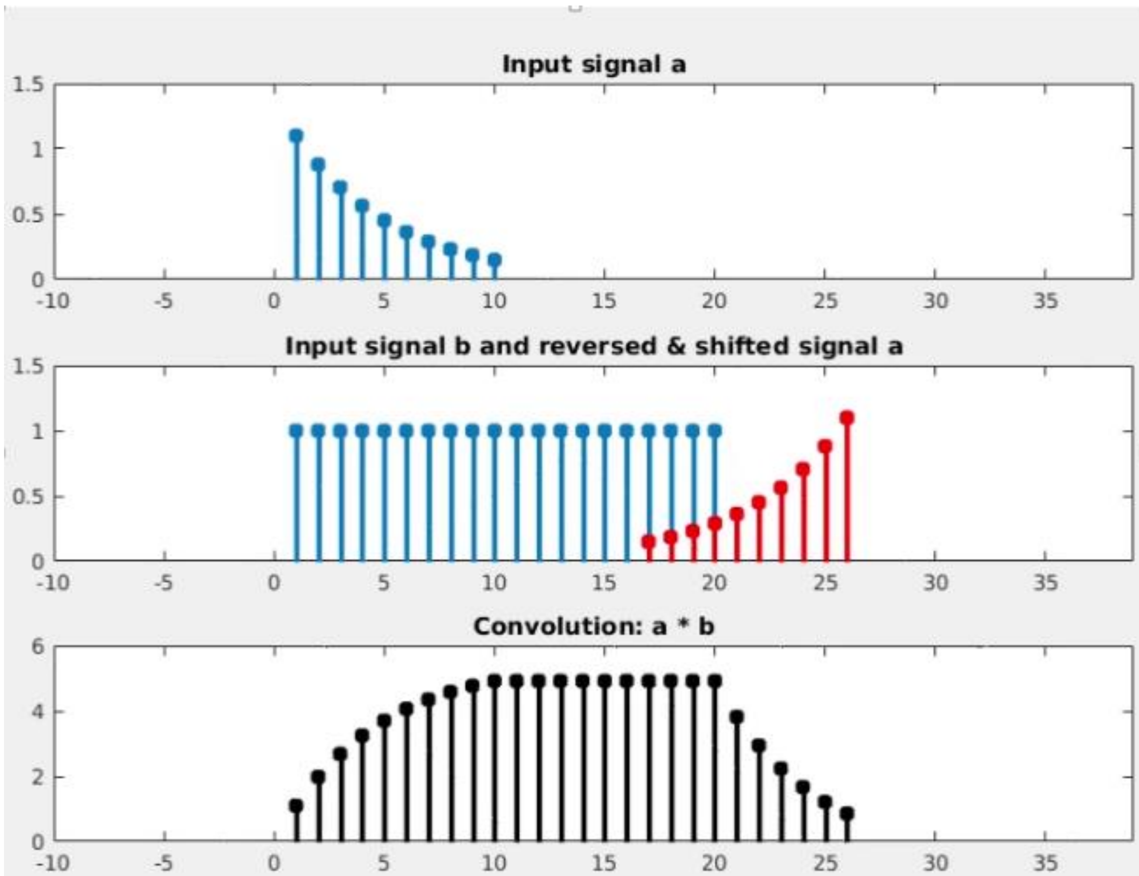
- One of the sequences is flipped (right to left) before sliding over the other
- Notation: $a \star b$
- Nice properties: linearity, associativity, commutativity, etc.



This operation is called *convolution*

Example of convolution of two sequences (or “signals”)

- One of the sequences is flipped (right to left) before sliding over the other
- Notation: $a \star b$
- Nice properties: linearity, associativity, commutativity, etc.

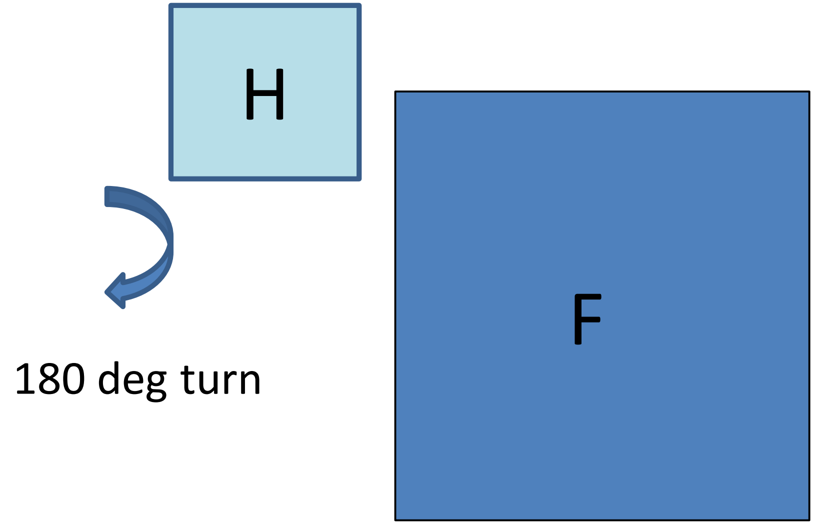


2D Filtering

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then slide the filter over the image

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

$$G = H \star F$$



Filtering an image: replace each pixel
with a linear combination of its neighbors.

The **filter** $\boldsymbol{H}$ is also called "**kernel**" or "**mask**".
It allows to have different weights depending on neighboring pixel's relative position.

Example: Moving Average In 2D

"box filter"

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Input image

$F[x, y]$

			0	0	0	0	0	0	0
			0	0	0	0	0	0	0
			90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filtered image

$G[x, y]$

	0								

Example: Moving Average In 2D

Input image

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filtered image

$$G[x, y]$$

	0	10							

Example: Moving Average In 2D

Input image

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filtered image

$$G[x, y]$$

	0	10	20						

Example: Moving Average In 2D

Input image

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filtered image

$$G[x, y]$$

	0	10	20	30					

Example: Moving Average In 2D

Input image

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filtered image

$$G[x, y]$$

	0	10	20	30	30				

Example: Moving Average In 2D

Input image

$$F[x, y]$$

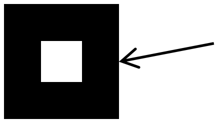
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filtered image

$$G[x, y]$$

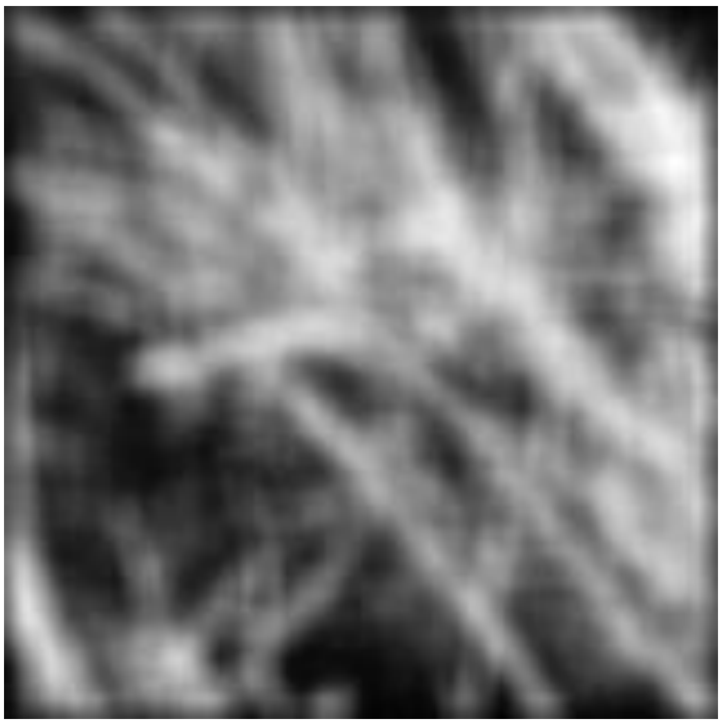
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Smoothing by averaging



Box filter:
white = high value, black = low value

original



filtered

Gaussian filter

- What if we want the closest pixels to have higher influence on the output?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

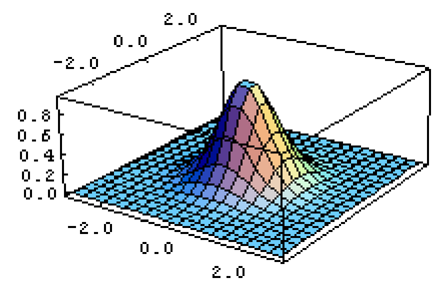
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

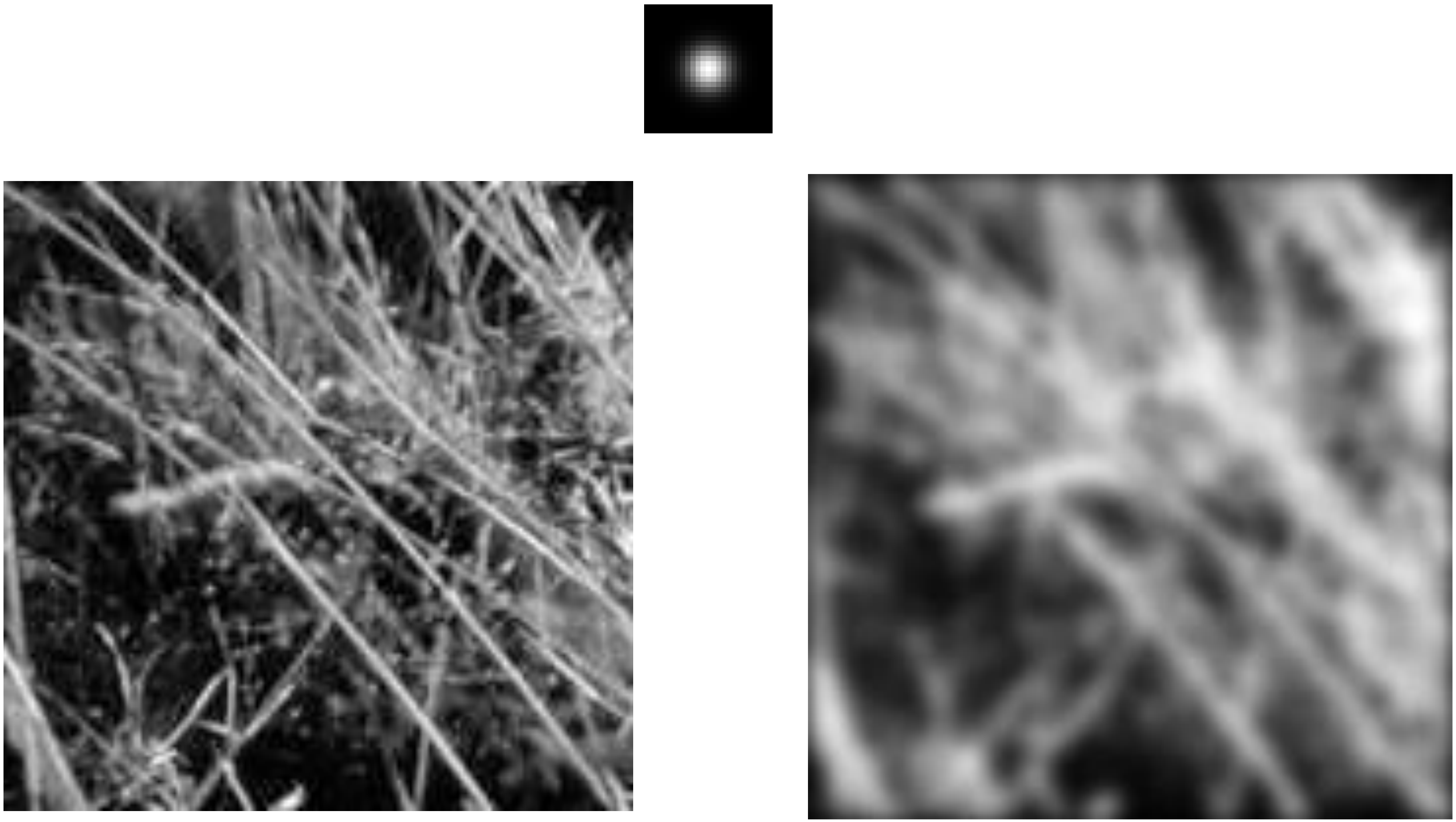
$H[u, v]$

This kernel is an approximation of a Gaussian function:

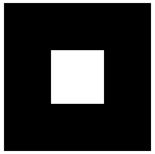
$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

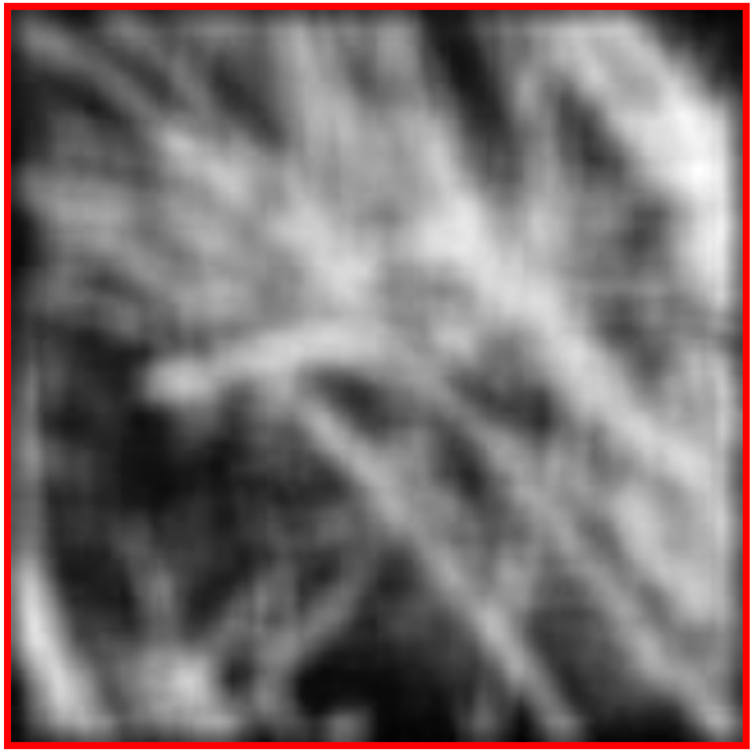


Smoothing with a Gaussian

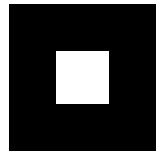


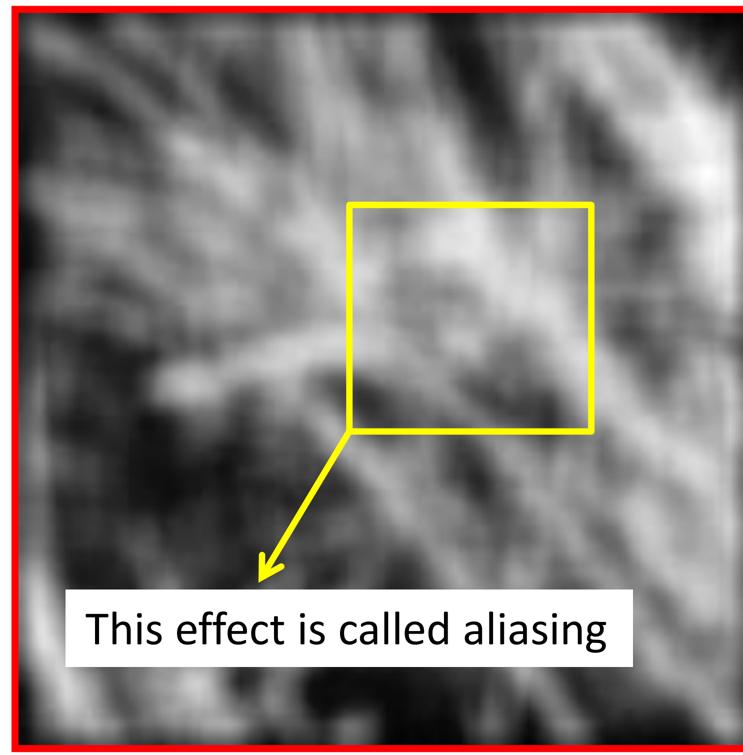
Compare the result with a box filter





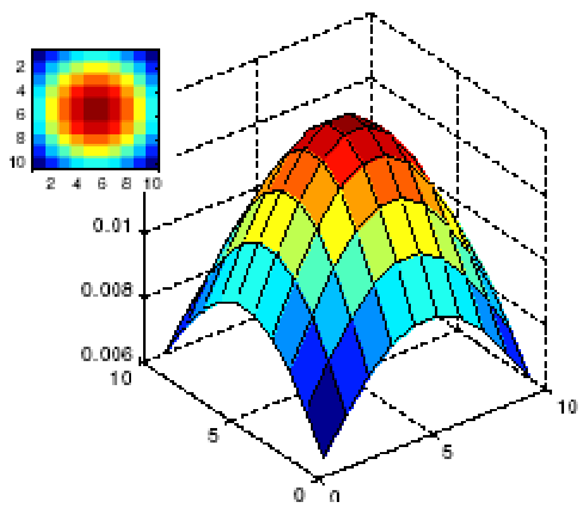
Compare the result with a box filter



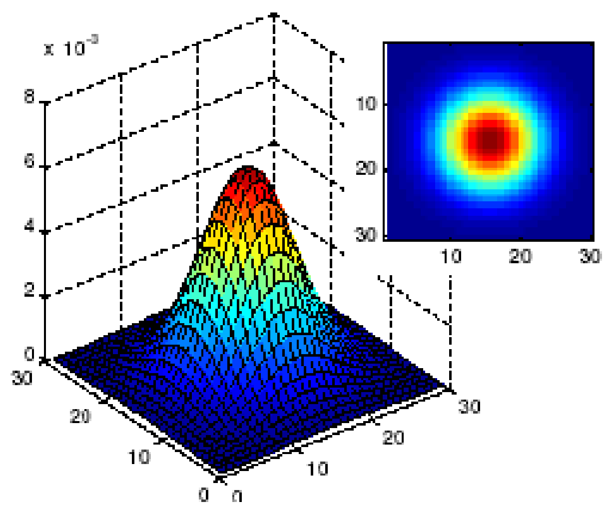


Gaussian filters

- What parameters matter?
- **Size** of the kernel
  - NB: a Gaussian function has **infinite support**, but discrete filters use finite kernels



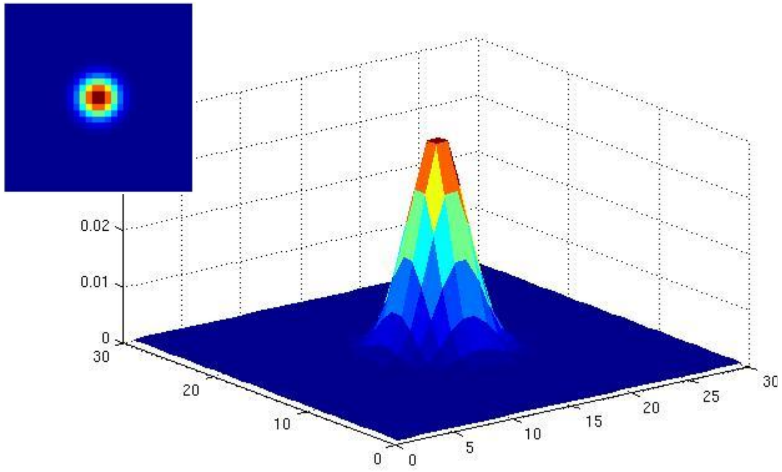
σ = 5 pixels
with 10 x 10 pixel kernel



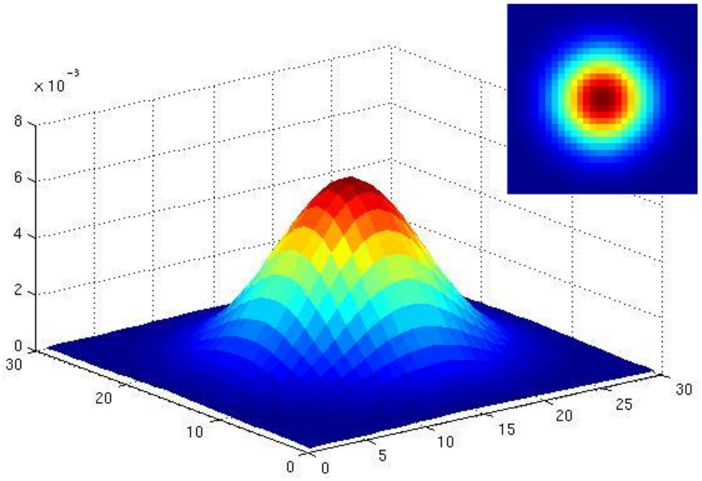
σ = 5 pixels
with 30 x 30 pixel kernel

Gaussian filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing



σ = 2 pixels
with 30 x 30 pixel kernel

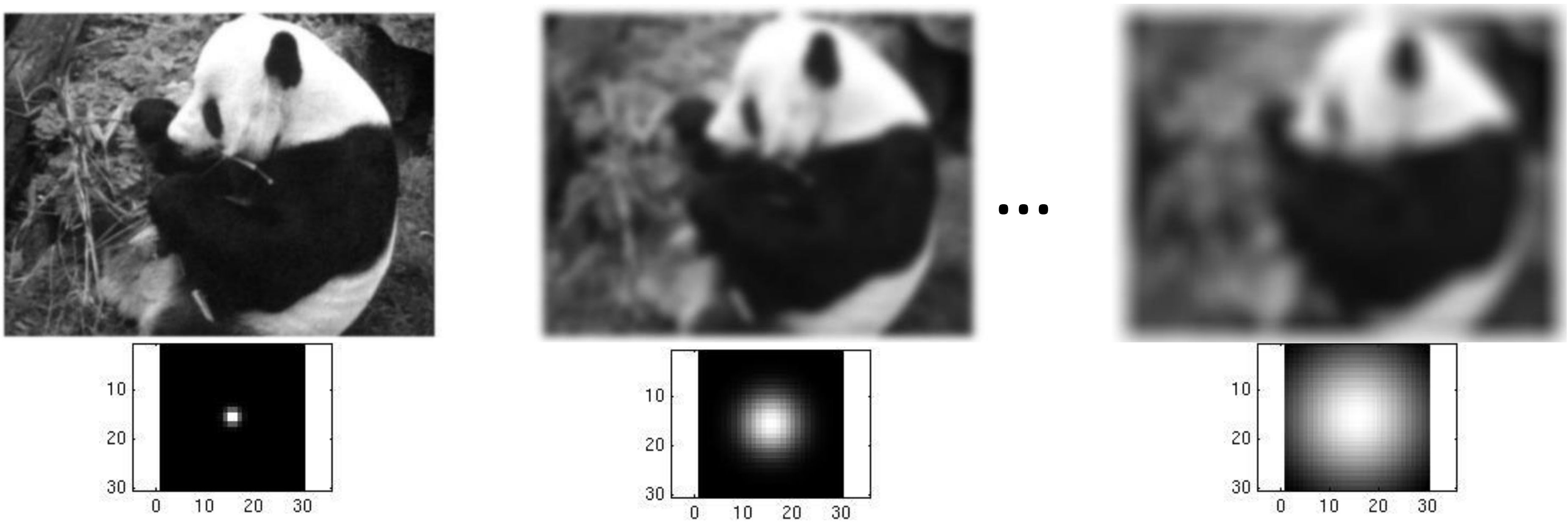


σ = 5 pixels
with 30 x 30 pixel kernel

Recall: standard deviation = σ [pixels], variance = σ^2 [pixels^2]

Smoothing with a Gaussian

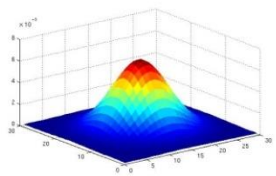
Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



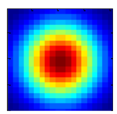
Sample Matlab code

```
>> hsize = 20;
>> sigma = 5;
>> h = fspecial('gaussian', hsize, sigma);
```

```
>> mesh(h);
```



```
>> imagesc(h);
```



```
>> im = imread('panda.jpg');
>> outim = imfilter(im, h);
>> imshow(outim);
```

outim

Boundary issues

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to pad the image borders
  - methods:
    - zero padding (black)
    - wrap around
    - copy edge
    - reflect across edge

Summary on (linear) smoothing filters

- <u>Smoothing filter</u>
  - has positive values (also called coefficients)
  - sums to 1 → preserve brightness of constant regions
  - removes “high-frequency” components; “low-pass” filter

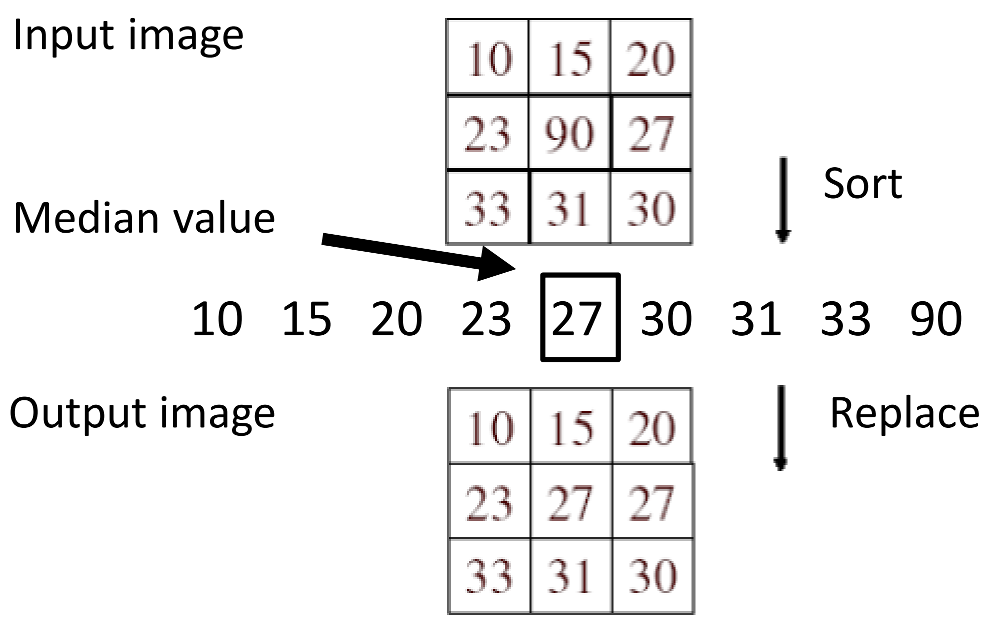
Non-linear filtering

Effect of smoothing filters

Linear smoothing filters do not alleviate salt and pepper noise!

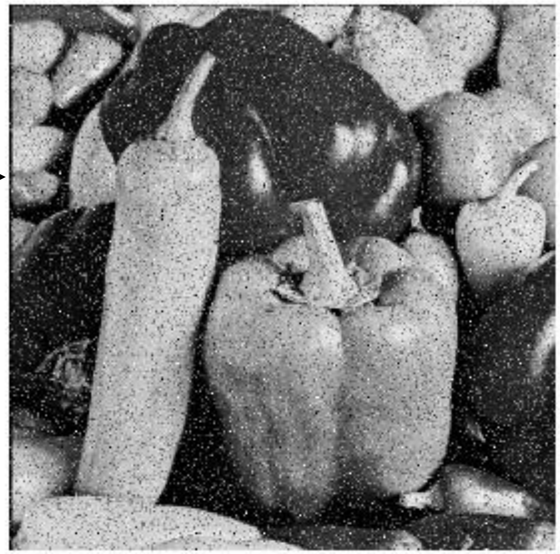
Median filter

- It is a non-linear filter
- Removes spikes: good for impulse, salt & pepper noise

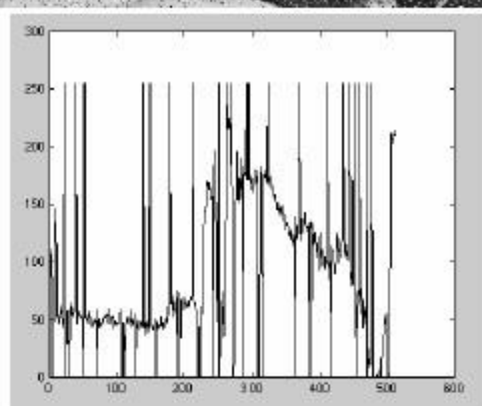


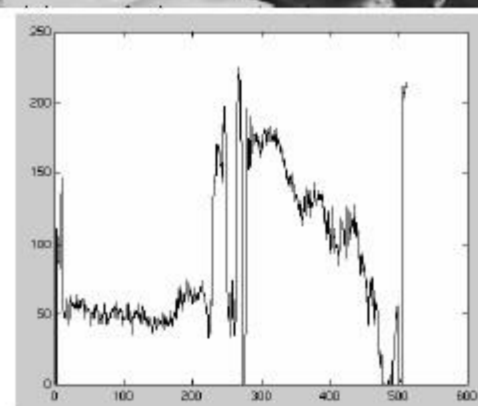
Median filter

Salt and pepper noise →



← Median filtered

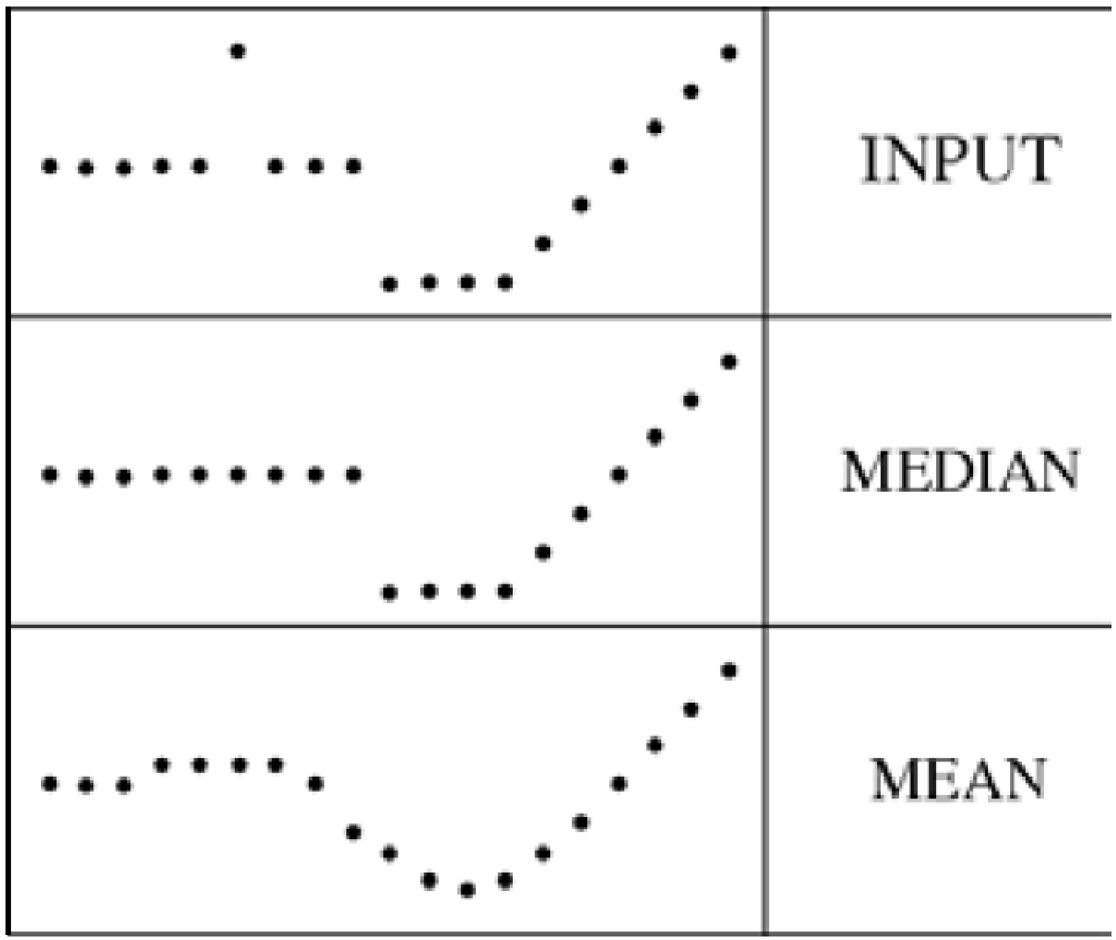




Plots of a row of the image

Median filter

- Median filter preserves sharp transitions (i.e., edges),

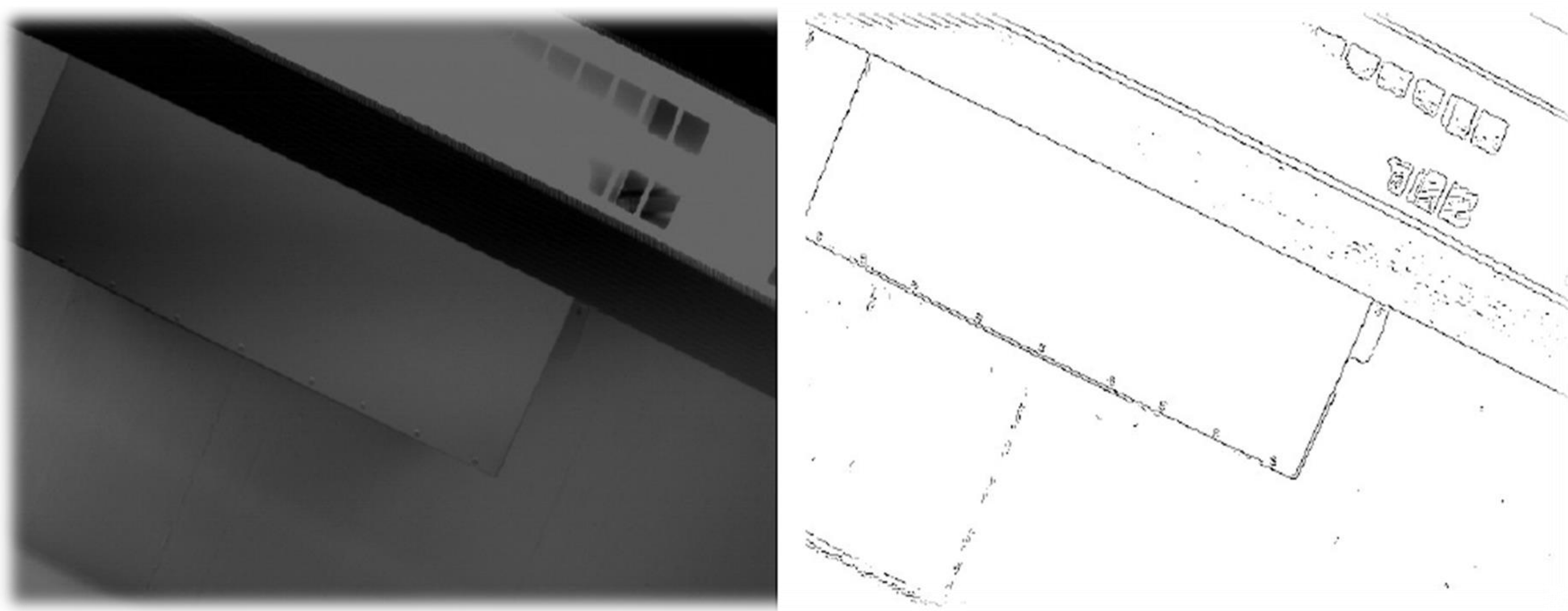


… but it removes small brightness variations.

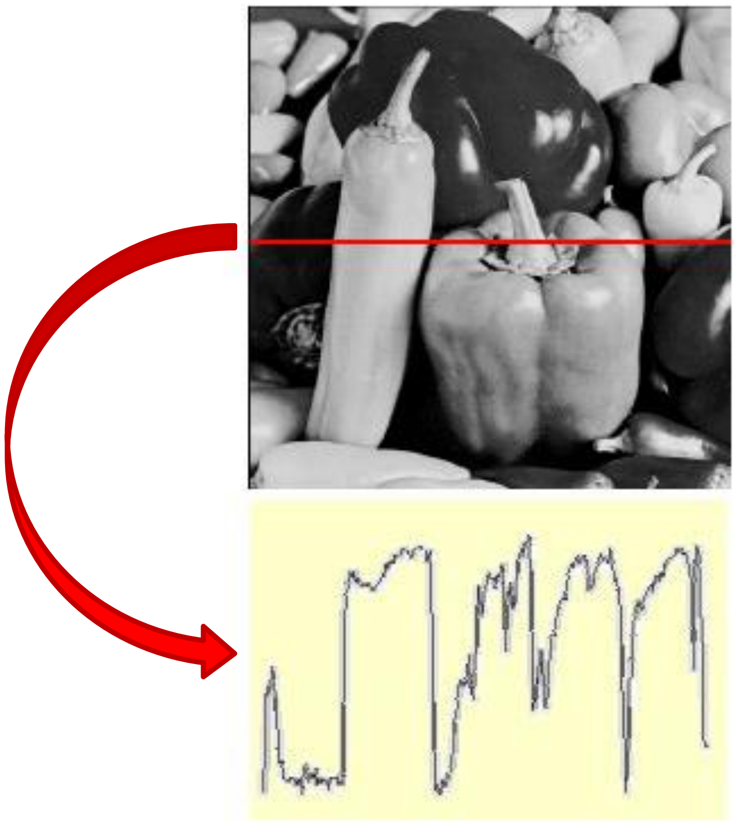
High-pass filtering
(edge detection)

Edge detection

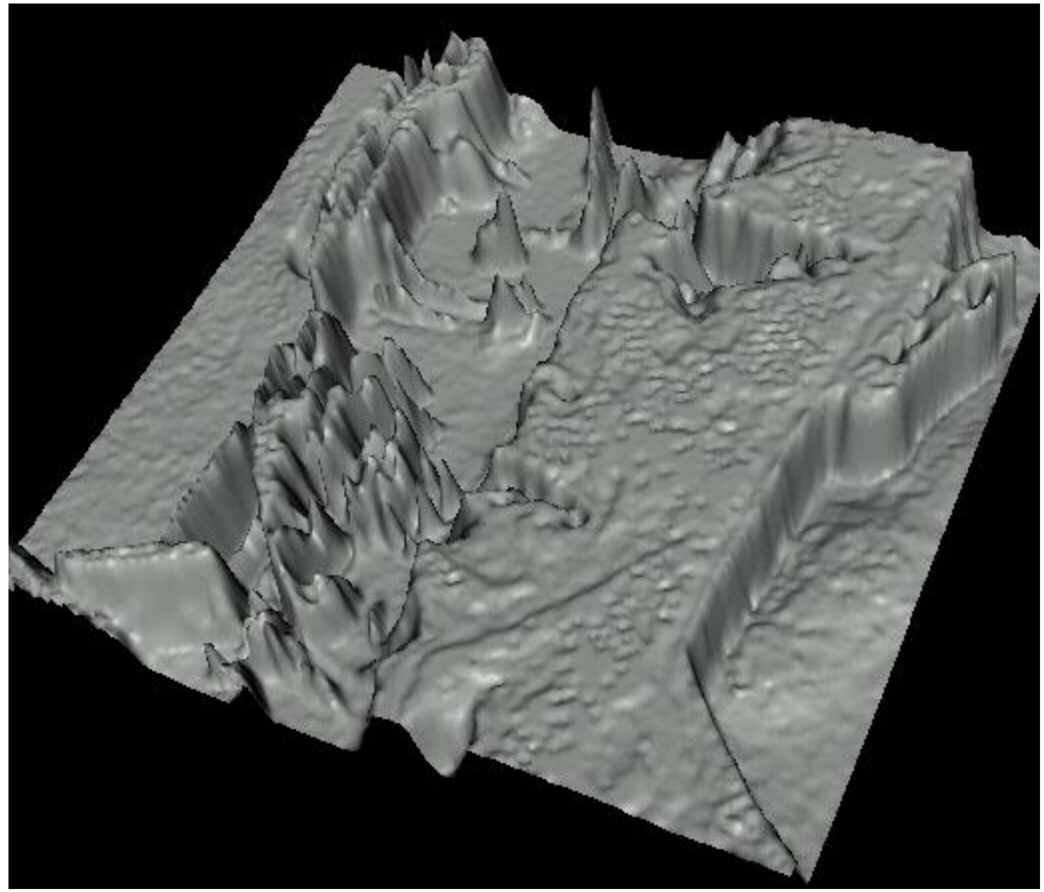
- Ultimate goal of edge detection: an idealized line drawing.
- Edge contours in the image correspond to important scene contours.



Edges are sharp intensity changes



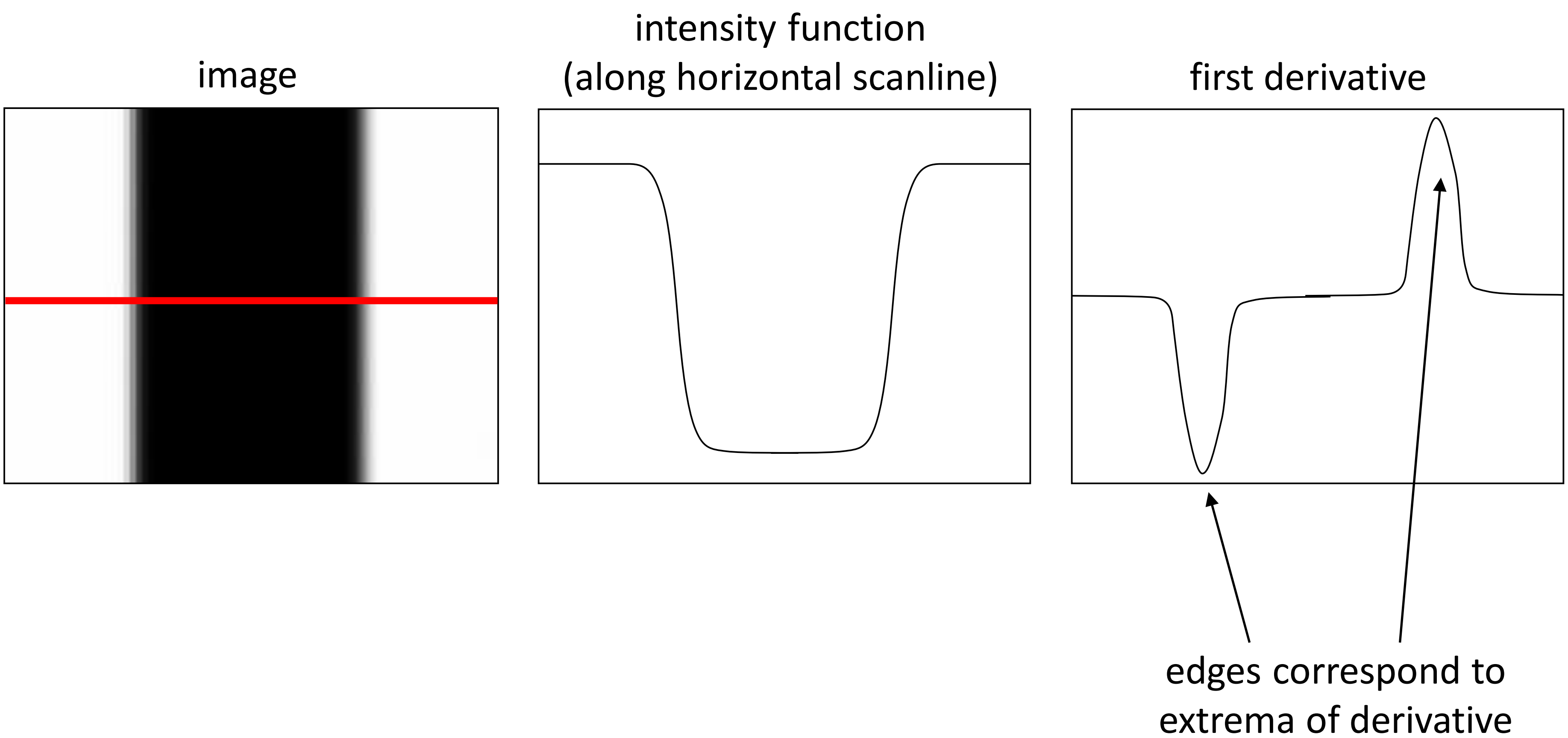
Images as functions $f(x, y)$



- Edges look like steep cliffs

Derivatives and edges

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



Differentiation and convolution

For 2D function, f(x,y), the partial derivative is:

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

For discrete data, we can approximate using finite differences:

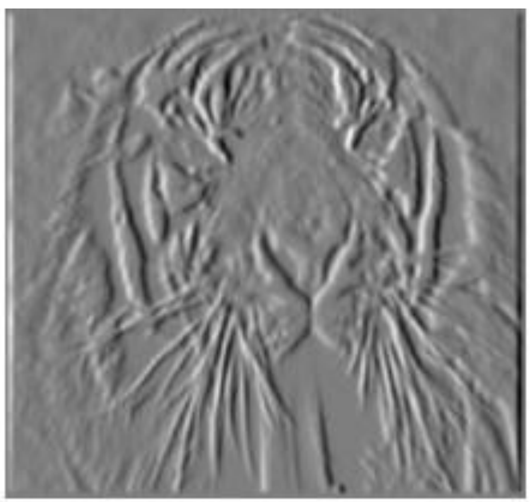
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

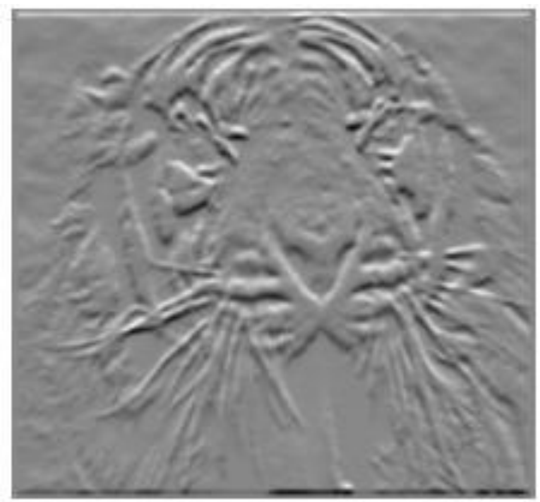
To implement the above as a convolution, what would be the associated filter?

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$

-1	1





$$\frac{\partial f(x,y)}{\partial y}$$

-1
1

Alternative Finite-difference filters

Prewitt filter $\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$ and $\mathbf{G_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$

Sobel filter $\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$ and $\mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$

```
Sample Matlab code
>> im = imread('lion.jpg');
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```

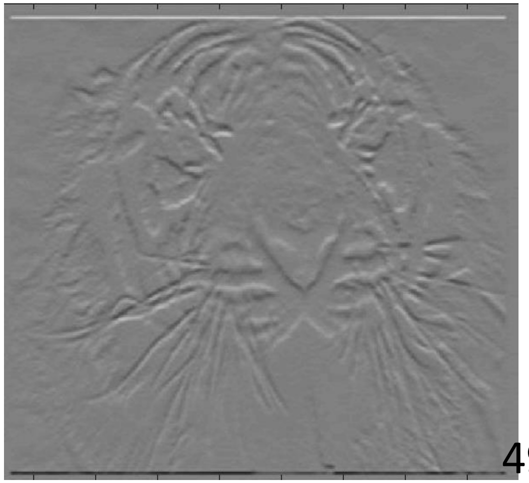
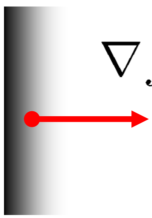


Image gradient

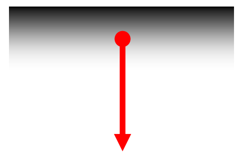
The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

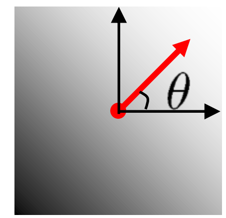
The gradient points in the direction of fastest intensity change



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient direction (orientation of edge normal) is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The *edge strength* is given by the gradient magnitude

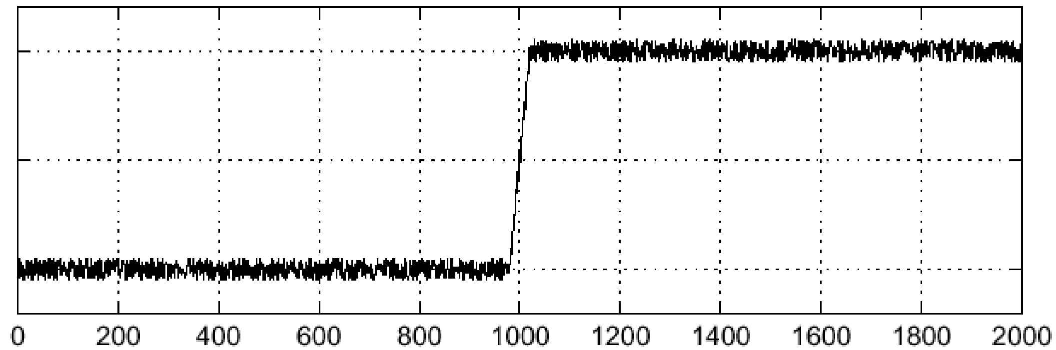
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Effects of noise

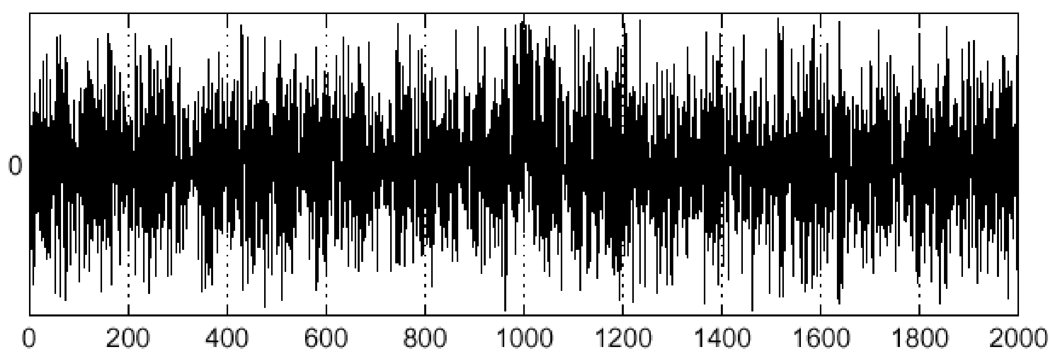
Consider a single row or column of the image

– Plotting intensity as a function of position gives a signal

$f(x)$



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



Where is the edge?

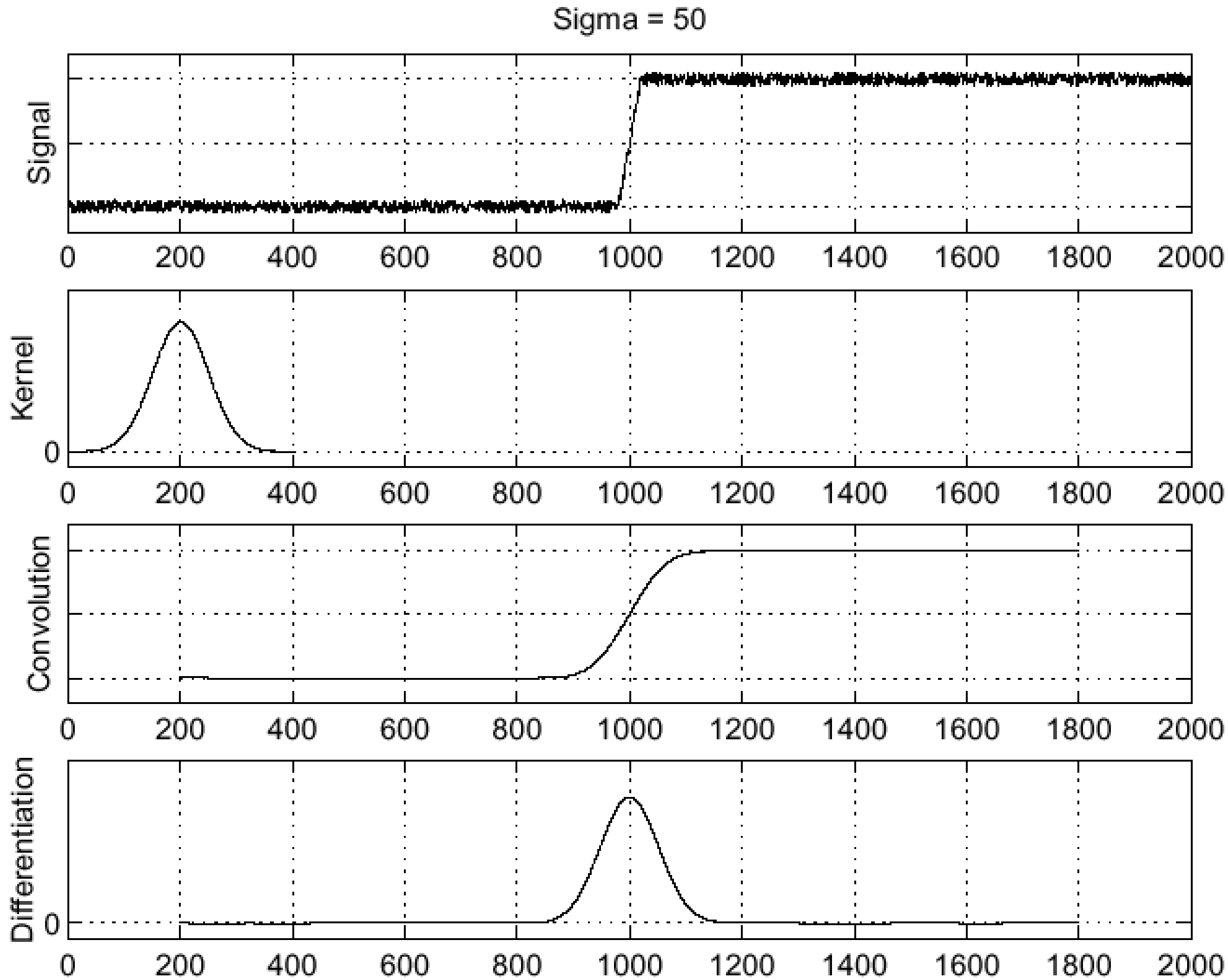
Solution: smooth first

f

h

$h \star f$

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



Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Alternative: combined derivative and smoothing filter

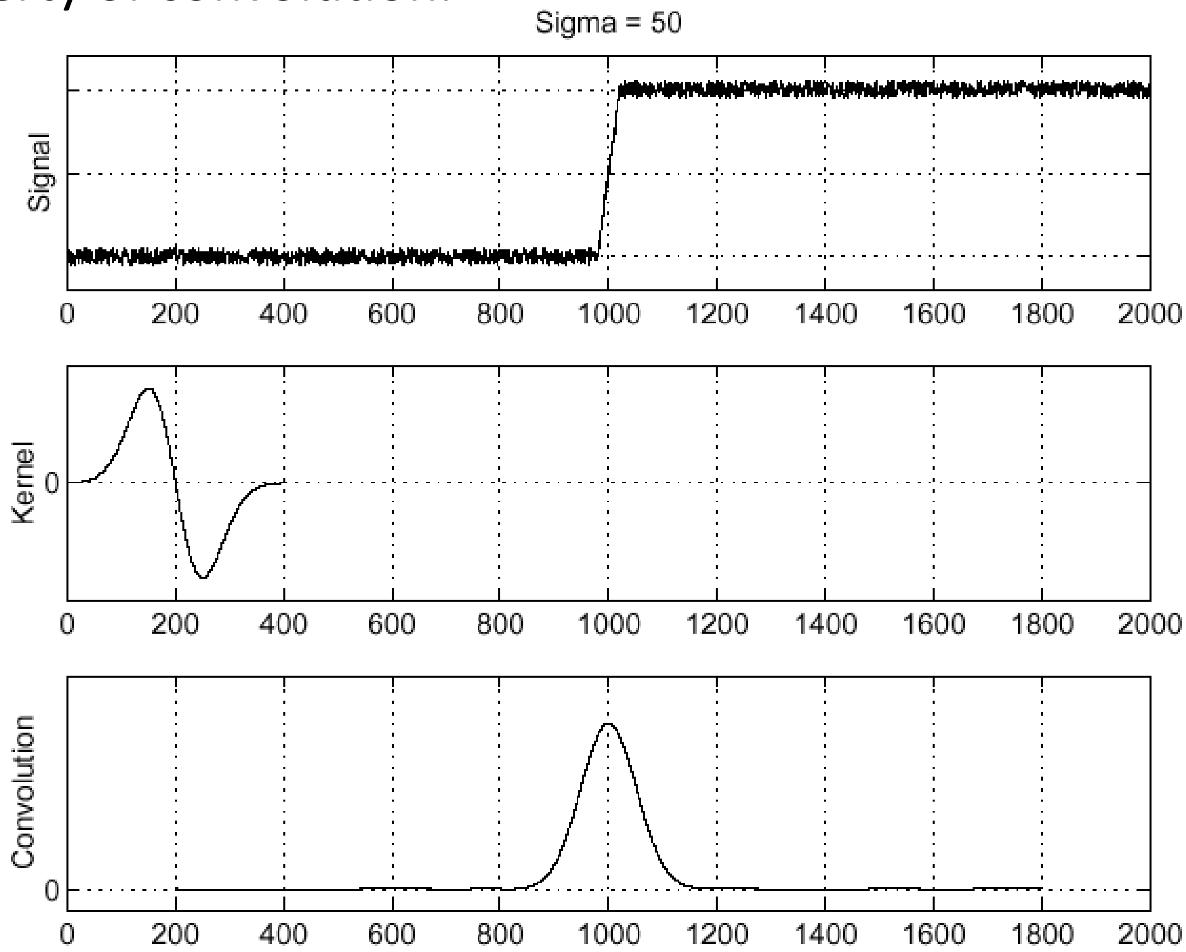
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Differentiation property of convolution.

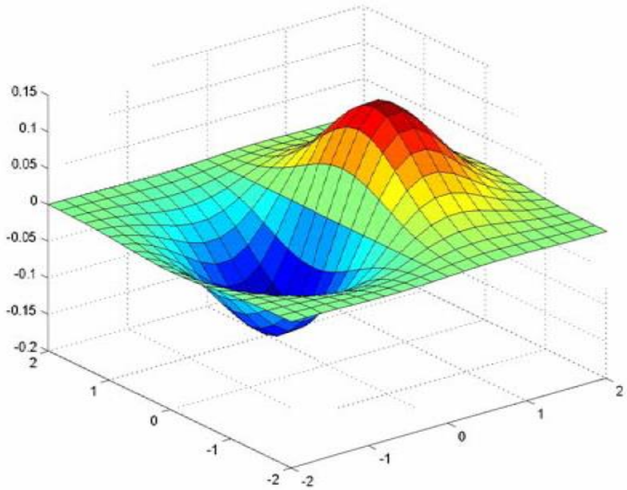
f

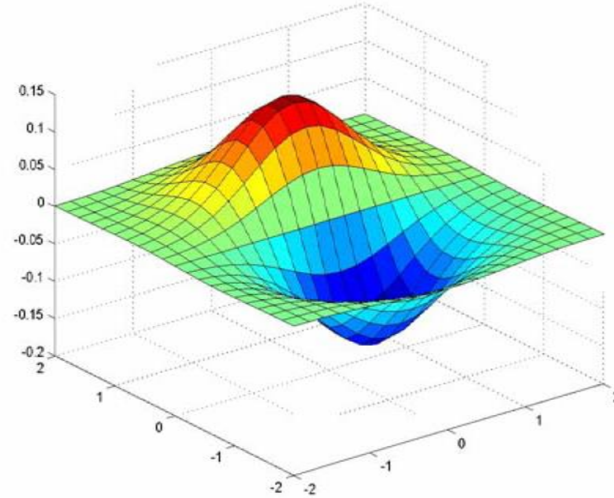
$\frac{\partial}{\partial x}h$

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



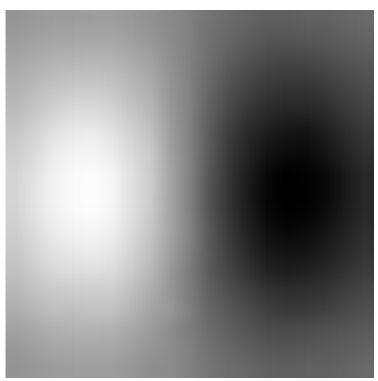
Derivative of Gaussian filters

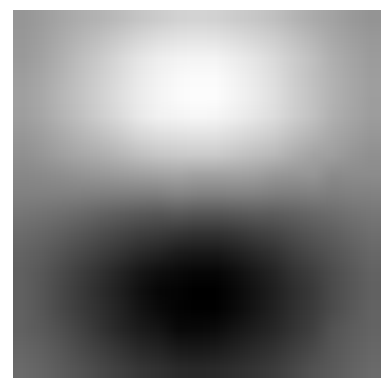




x-direction

y-direction





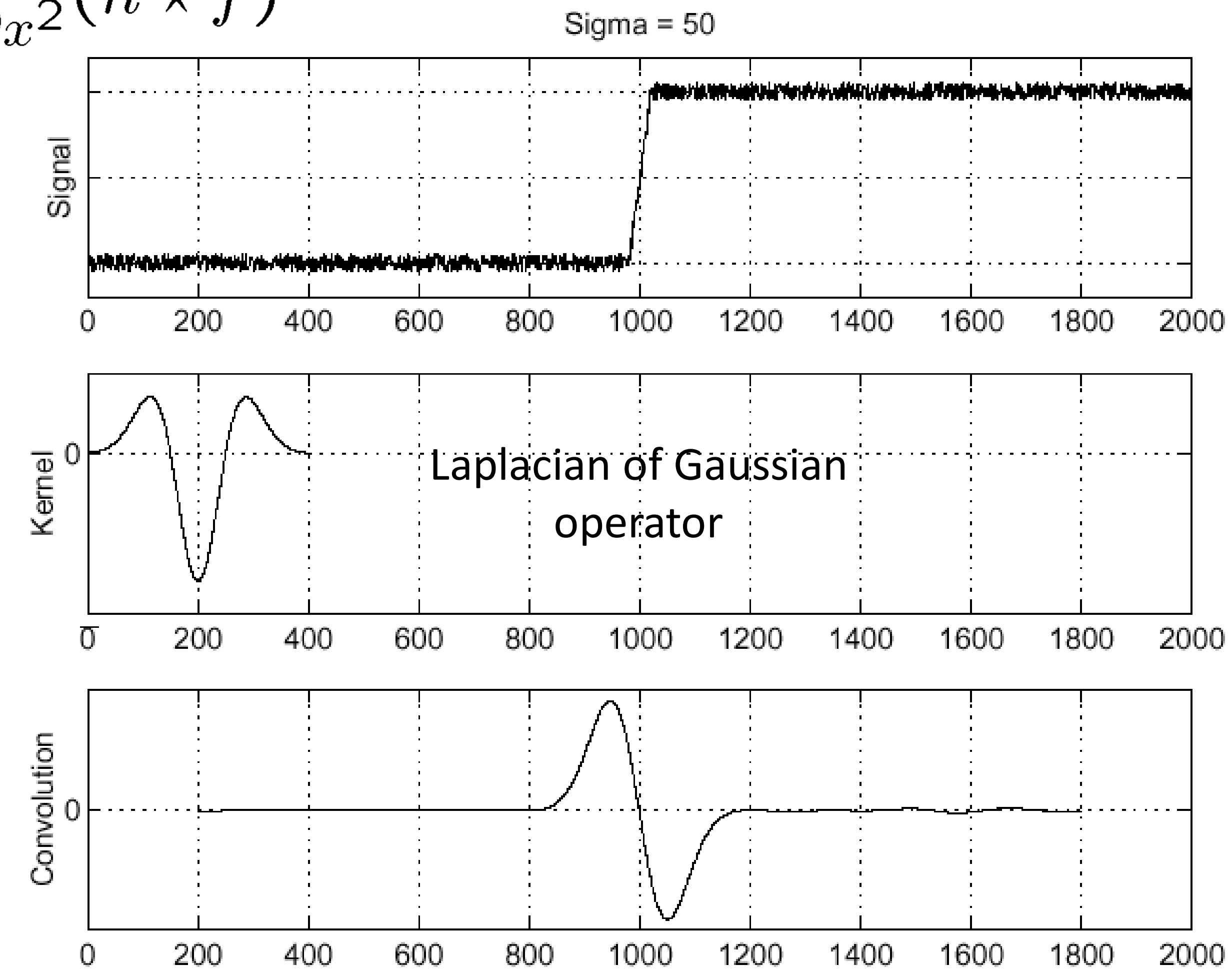
Laplacian of Gaussian

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

f

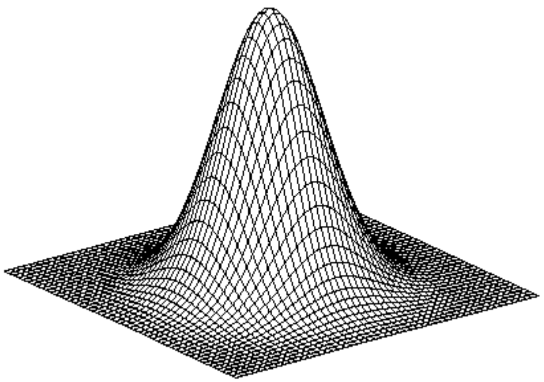
$\frac{\partial^2}{\partial x^2}h$

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



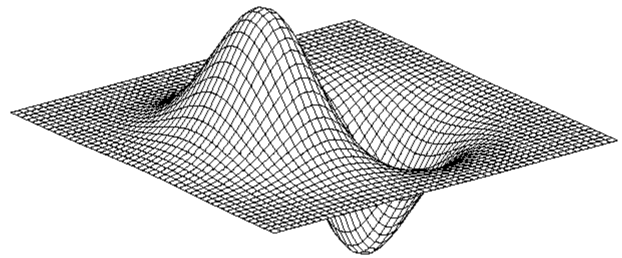
Where is the edge? Zero-crossings of bottom graph

2D edge detection filters



Gaussian

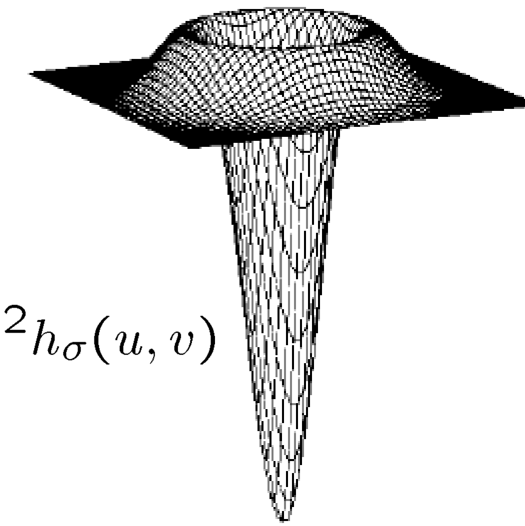
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



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

- ∇^2 is the Laplacian operator:

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

Summary on (linear) filters

- <u>Smoothing filter:</u>
  - **has positive values**
  - **sums to 1** → preserve brightness of constant regions
  - removes “high-frequency” components: “low-pass” filter
- <u>Derivative filter:</u>
  - **has opposite signs** used to get high response in regions of high contrast
  - **sums to 0** → no response in constant regions
  - **highlights “high-frequency”** components: “high-pass” filter

The Canny edge-detection algorithm (1986)

- Compute gradient of smoothed image in both directions
- Discard pixels whose gradient magnitude is below a certain threshold
- **Non-maximal suppression**: identify local maxima along gradient direction

The Canny edge-detection algorithm (1986)

Original image (Lenna image: https://en.wikipedia.org/wiki/Lenna)

Take a grayscale image. If not grayscale (i.g., RGB), convert it into a grayscale by replacing each pixel by the mean value of its R, G, B components.

The Canny edge-detection algorithm (1986)

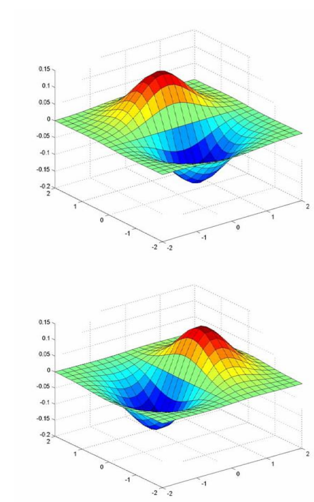
Take a grayscale image. If not grayscale (i.g., RGB), convert it into a grayscale by replacing each pixel by the mean value of its R, G, B components.

Original image (Lenna image: https://en.wikipedia.org/wiki/Lenna)

The Canny edge-detection algorithm (1986)

Convolve the image with x and y derivatives of Gaussian filter

$$\nabla f = \nabla\left(G_\sigma * I\right)$$



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

The Canny edge-detection algorithm (1986)

Thresholding $|\nabla f|$

Threshold it (i.e., set to 0 all pixels whose value is below a given threshold)

The Canny edge-detection algorithm (1986)

Take local maximum along gradient direction

Thinning: non-maxima suppression (local-maxima detection) along edge direction

Summary (things to remember)

- Image filtering (definition, motivation, applications)
- Moving average
- Linear filters and formulation: box filter, Gaussian filter
- Boundary issues
- Non-linear filters
  - Median filter and its applications
- Edge detection
  - Derivating filters (Prewitt, Sobel)
  - Combined derivative and smoothing filters (deriv. of Gaussian)
  - Laplacian of Gaussian
  - Canny edge detector
- Readings: Ch. 3.2, 4.2.1 of Szeliski book