ROBOTICS & PERCEPTION GROUP

Universität Zürich[UZH]

Lecture 04
Image Filtering

Prof. Dr. Davide Scaramuzza

sdavide@ifi.uzh.ch

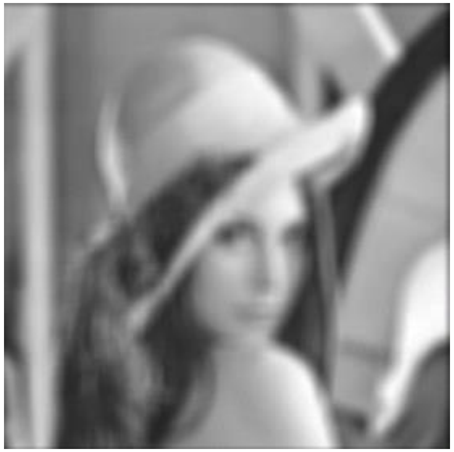
Exercise schedule update

Lab exercise sessions are shown in YELLOW.
The online program has been updated too.

Date	Time	Description of the lecture/exercise	Lecturer
15.10.2015	10:15 - 12:00 14:15 – 15:45	05 - Point Feature Detectors 1: Harris detector Lab Exercise 2: Harris detector	Scaramuzza Elias Mueggler/Zichao Zhang
22.10.2015	10:15 - 12:00	06 - Point Feature Detectors 2: SIFT, BRIEF, BRISK	Scaramuzza
29.10.2015	10:15 - 12:00 14:15 – 15:45	07 - Multiple-view geometry 1: Epipolar geometry and stereo Lab Exercise 3: Stereo vision	Scaramuzza Elias Mueggler/Zichao Zhang
05.11.2015	10:15 - 12:00	08 - Multiple-view geometry 2: Two-view Structure from Motion and RANSAC	Scaramuzza
12.11.2015	10:15 - 12:00 14:15 – 15:45	09 - Multiple-view geometry 3: N-view Structure-from-Motion and Bundle Adjustment Exercise 4: 8-point algorithm and RANSAC	Scaramuzza Elias Mueggler/Zichao Zhang
19.11.2015	10:15 - 12:00	10 - Dense 3D Reconstruction (Multi-view Stereo)	Scaramuzza
26.11.2015	10:15 - 12:00 14:15 – 15:45	11 - Optical Flow and Tracking (Lucas-Kanade) Exercise 5: Lucas-Kanade tracker	Scaramuzza Elias Mueggler/Zichao Zhang
03.12.2015	10:15 - 12:00 14:15 – 15:45	12 - Image Retrieval Exercise 6: Recognition with Bag of Words	Scaramuzza Elias Mueggler/Zichao Zhang

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** enhances the contours of an image (high frequency)



Low-pass filtered image

High-pass filtered image

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

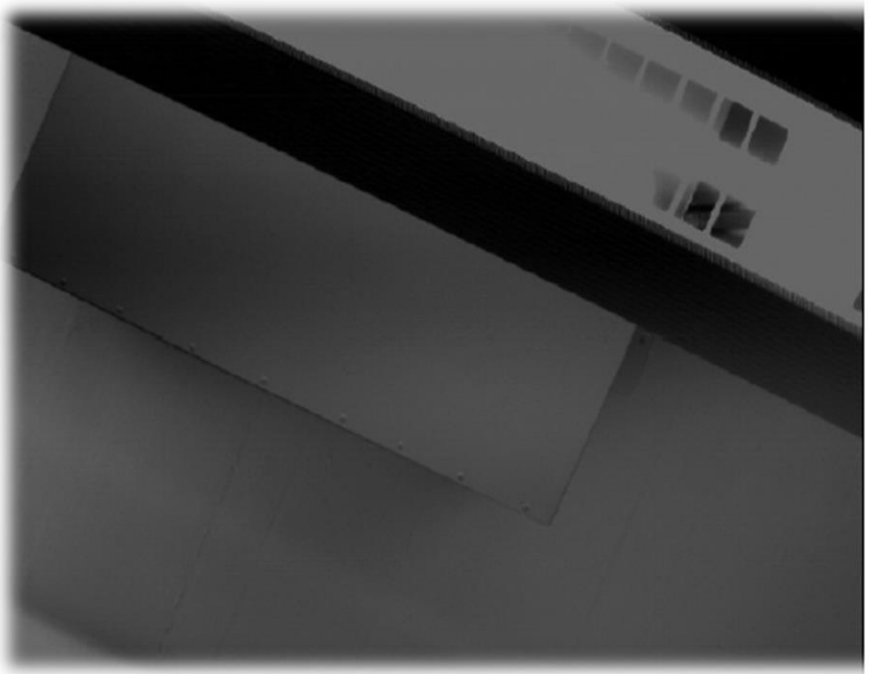
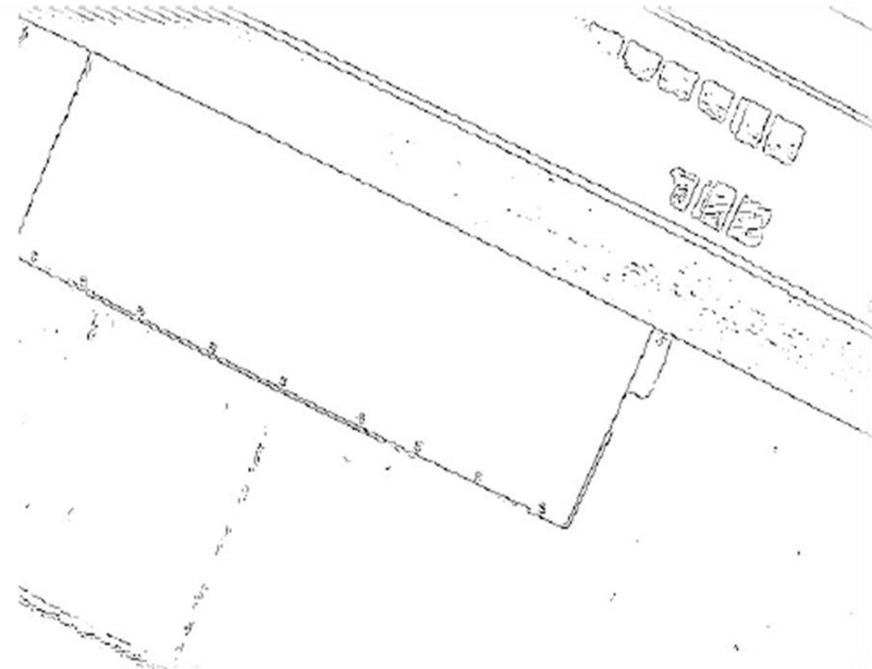
Salt and pepper noise

Impulse noise

Gaussian noise

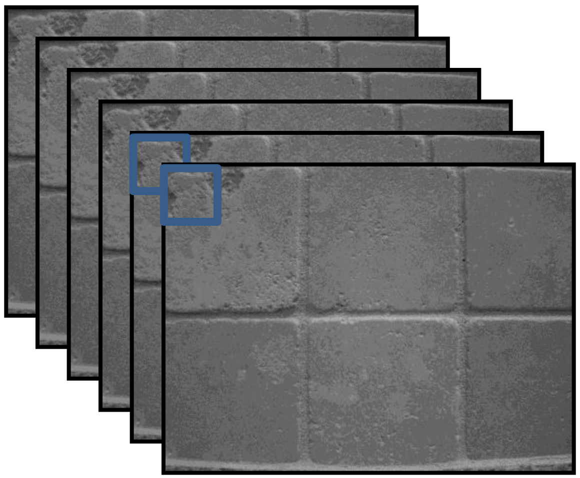
Source: S. Seitz

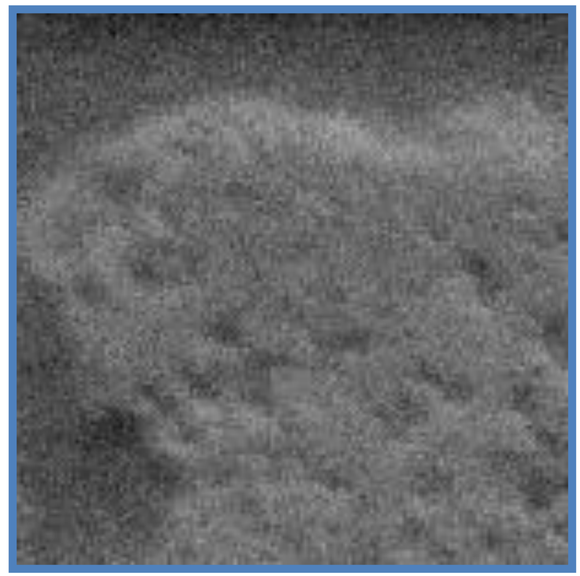
High-pass filtering
Motivation: edge detection

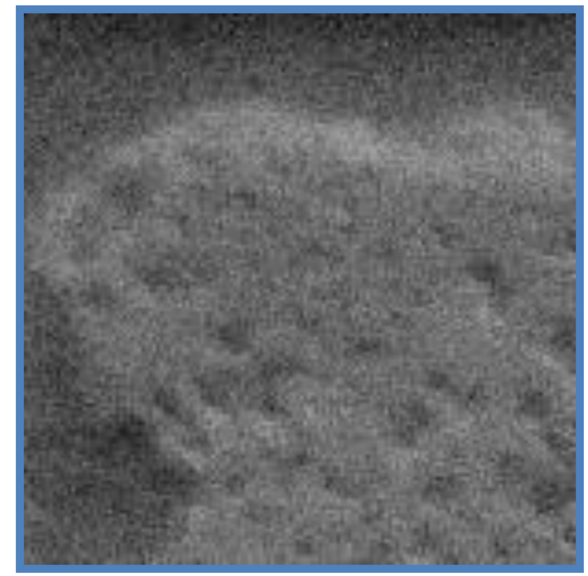
Low-pass filtering

A simple noise reduction algorithm





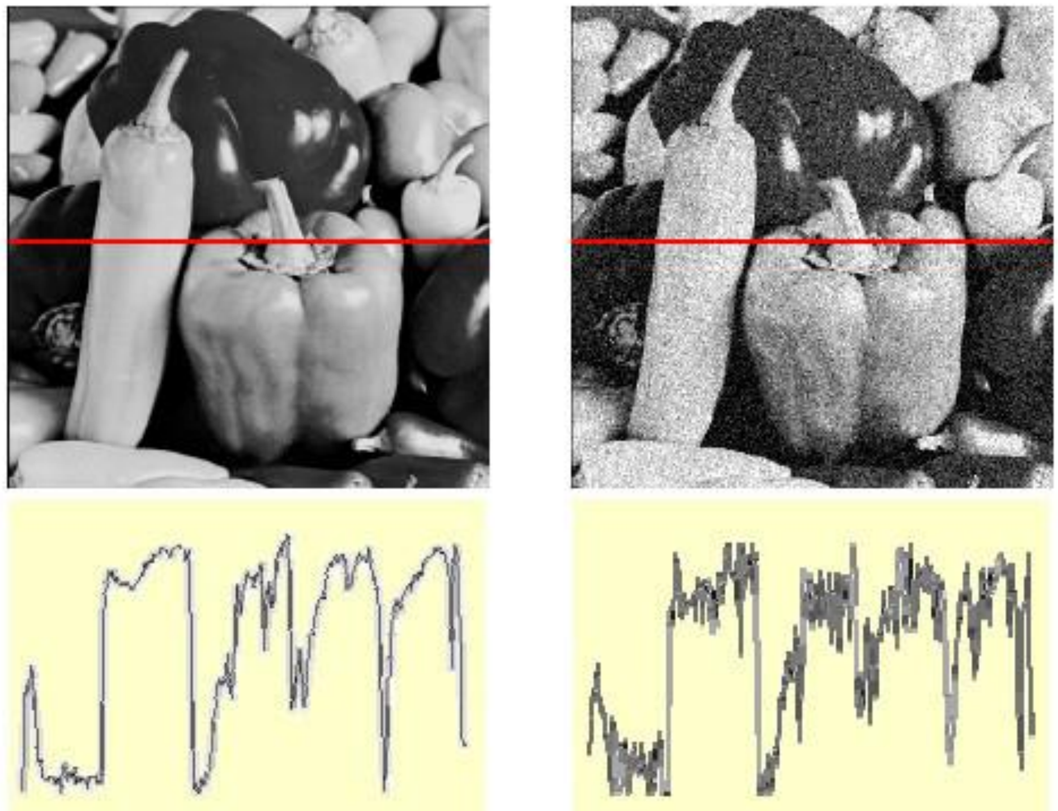
Time t



Time $t + 1$

- We can measure **noise** in multiple images of the same static scene.
- How could we reduce the noise?
- What if there is only one image?

Gaussian noise



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

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

```
>> noise = randn(size(im)).*sigma;

>> output = im + noise;
```

Gaussian noise

Effect of sigma on Gaussian noise. This image shows the noise values added to the raw intensities of an image.

Sigma = 1

Gaussian noise

Effect of sigma on Gaussian noise. This image shows the noise values added to the raw intensities of an image.

Sigma = 16

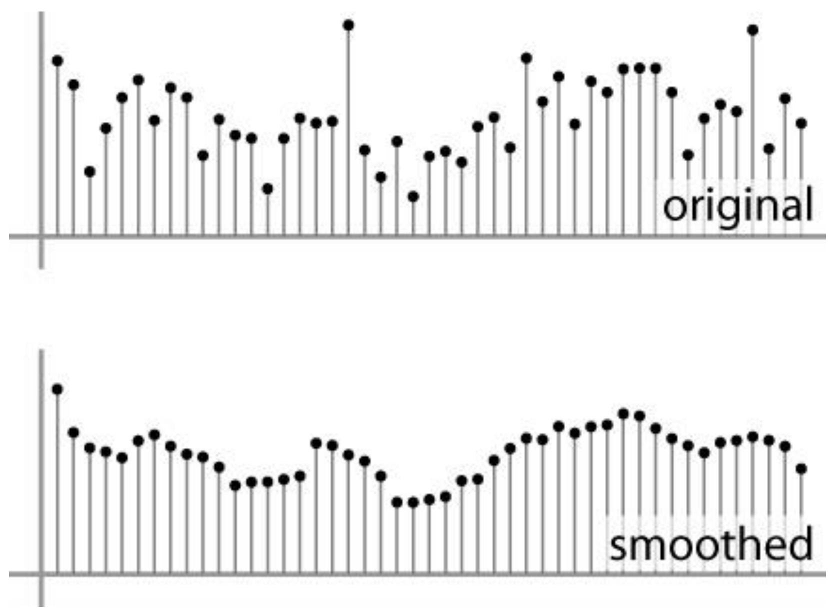
How can we reduce the noise?

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 processes to be independent from pixel to pixel

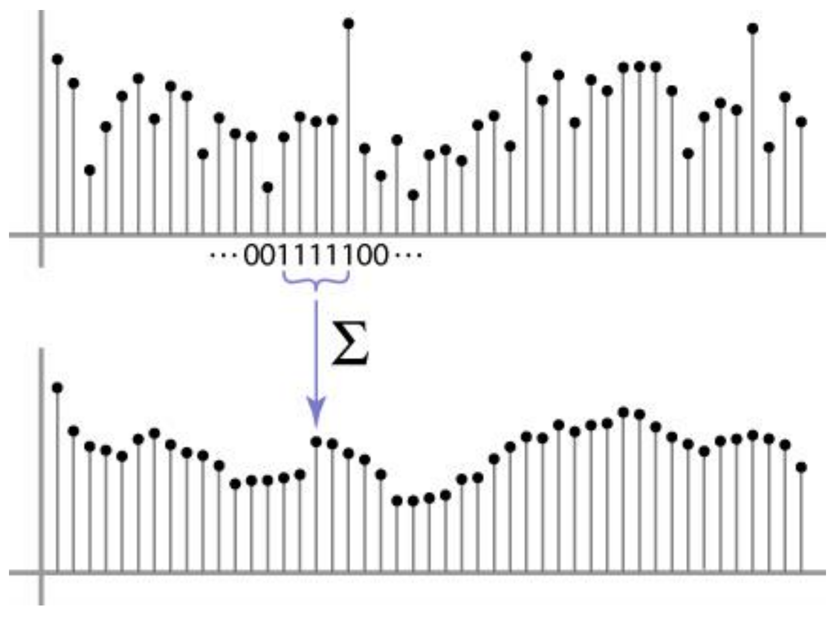
Moving average

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



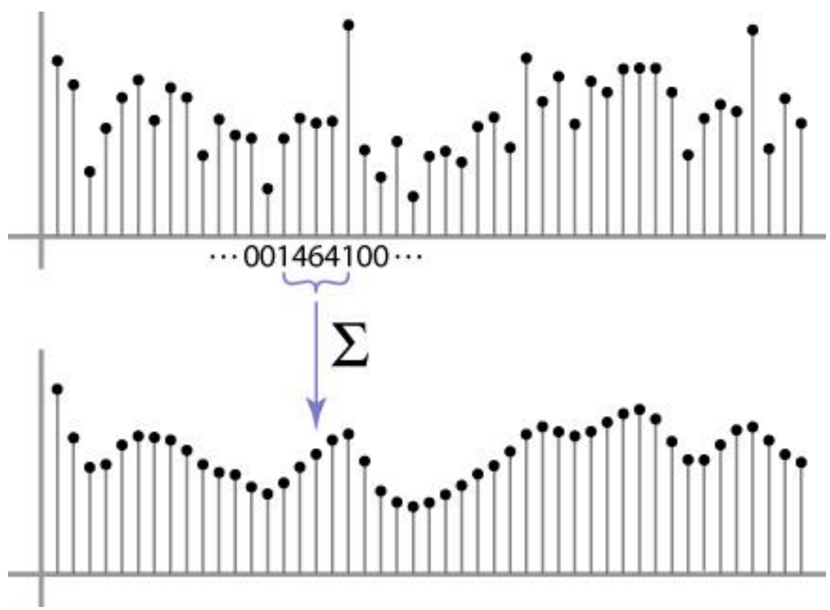
Weighted Moving Average

- 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



Moving Average In 2D

$F[x,y]$

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

$G[x,y]$

	0								

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10							

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20						

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20	30					

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30				

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	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	

Filtering by Correlation

If the averaging window size is 2k+1 x 2k+1:

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel } F[i,j]}$$

Attribute uniform weight to each pixel

Loop over all pixels in neighborhood around image pixel $F[i,j]$

Now generalize to allow different weights depending on neighboring pixel's relative position:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

Non-uniform weights

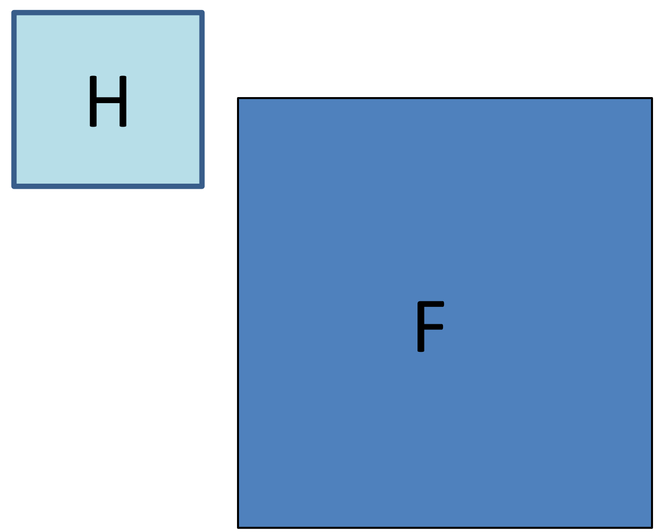
Filtering by Correlation

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

This is called cross-correlation, denoted $G = H \otimes F$

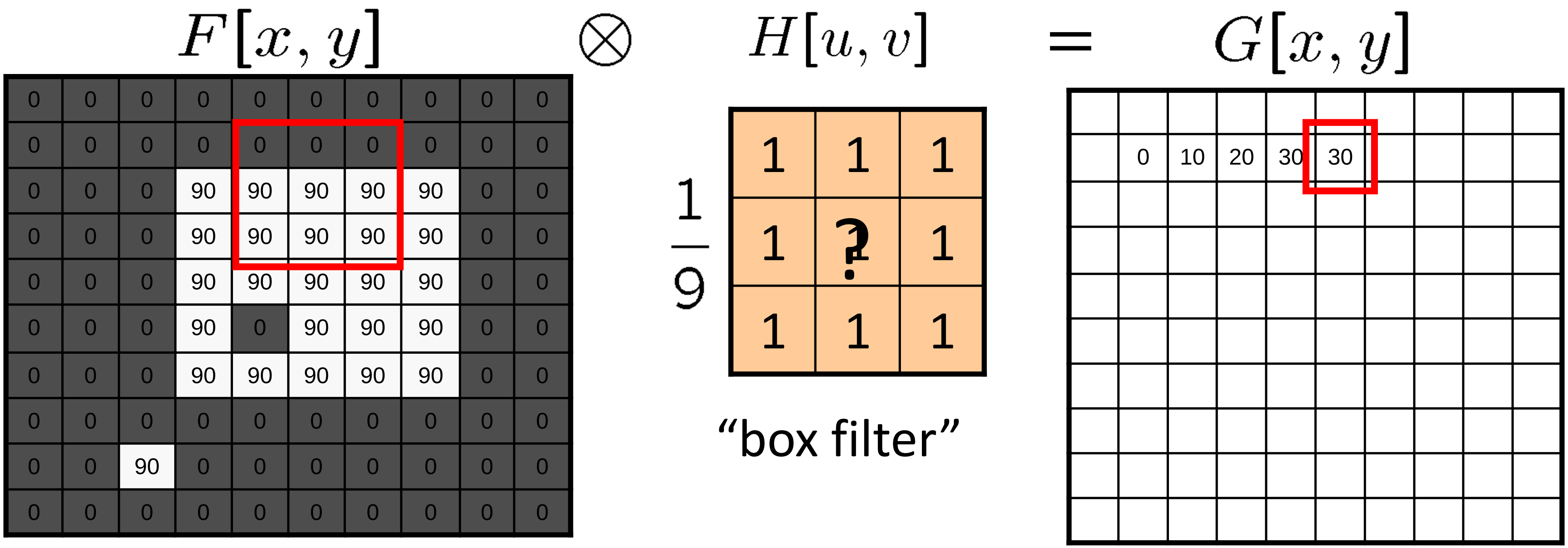
Filtering an image: replace each pixel with a linear combination of its neighbors.

The **filter** $\boldsymbol{H}$ is also called "**kernel**" or "**mask**"



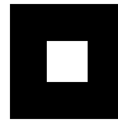
Averaging filter

- What values belong in the kernel *H* for the moving average example?



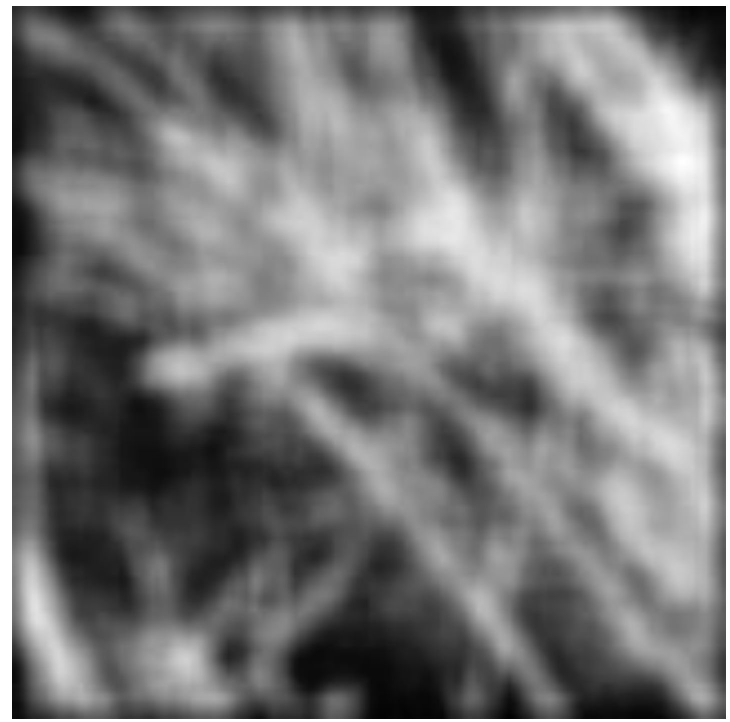
$$G = H \otimes F$$

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

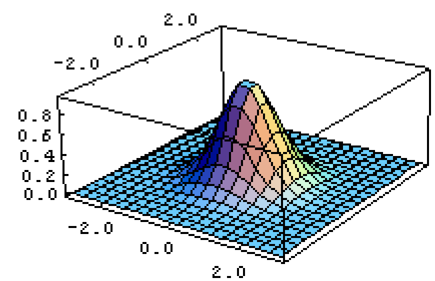
$F[x, y]$

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

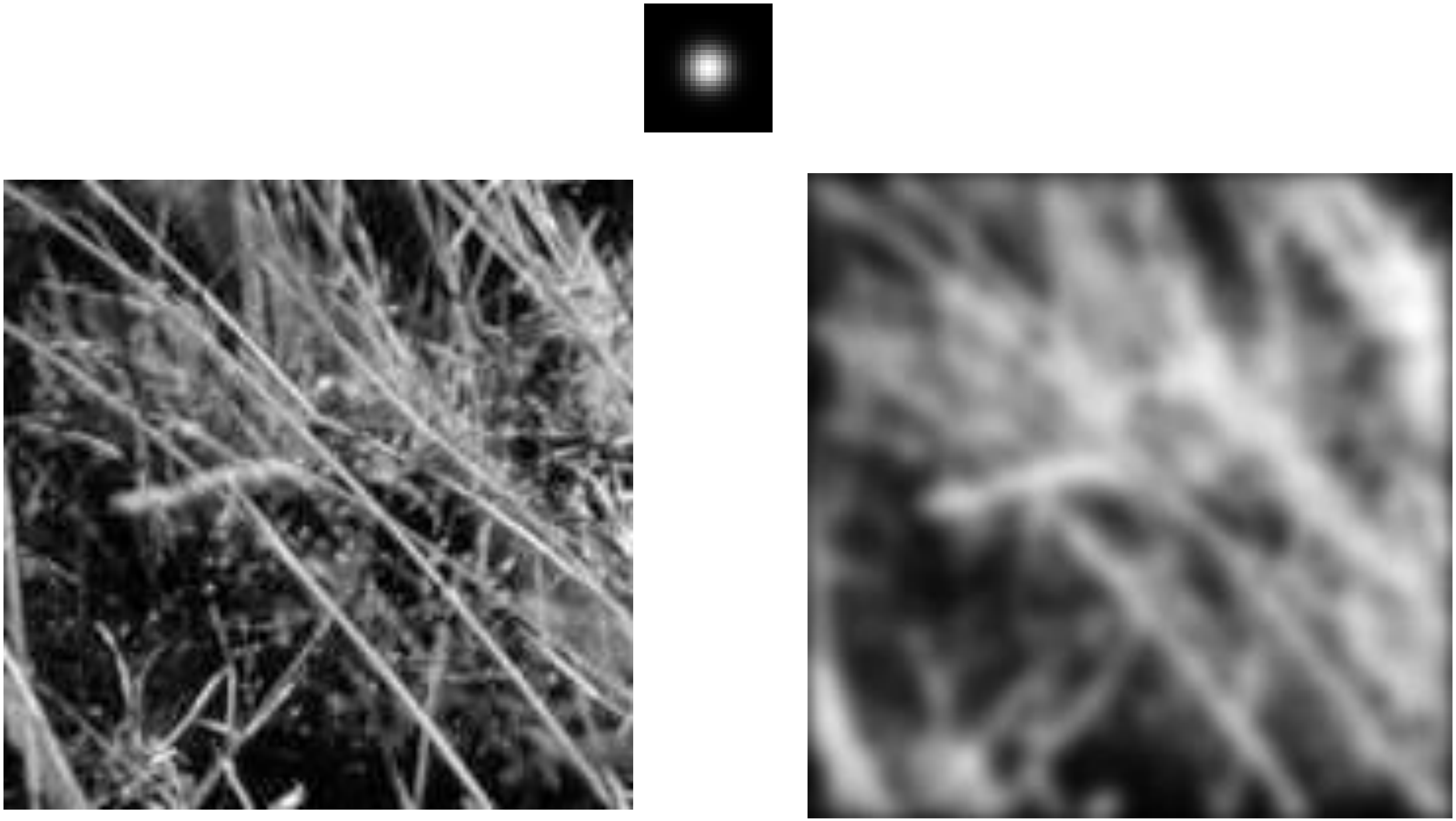
$H[u, v]$

This kernel is an approximation of a Gaussian function:

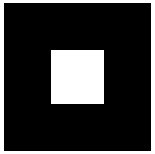
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

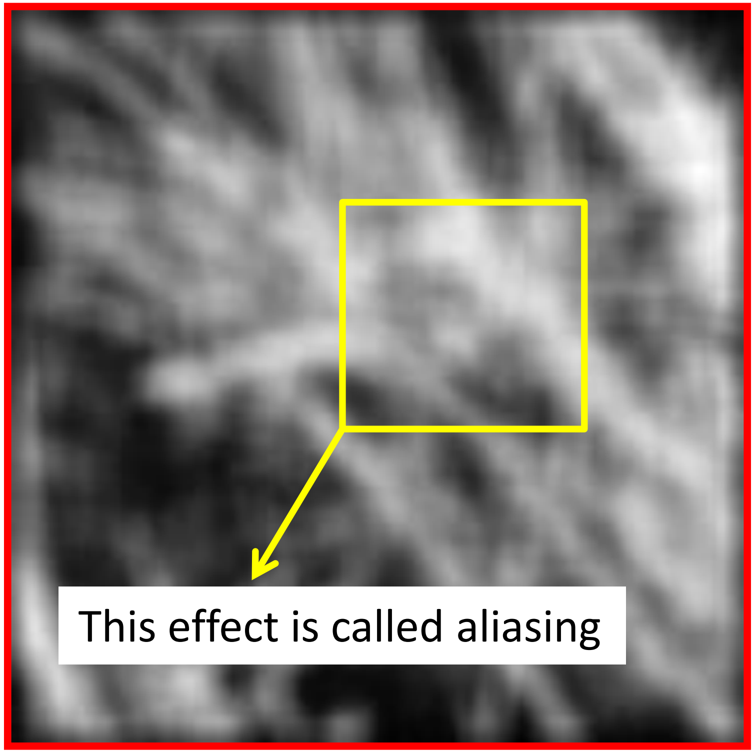


Smoothing with a Gaussian



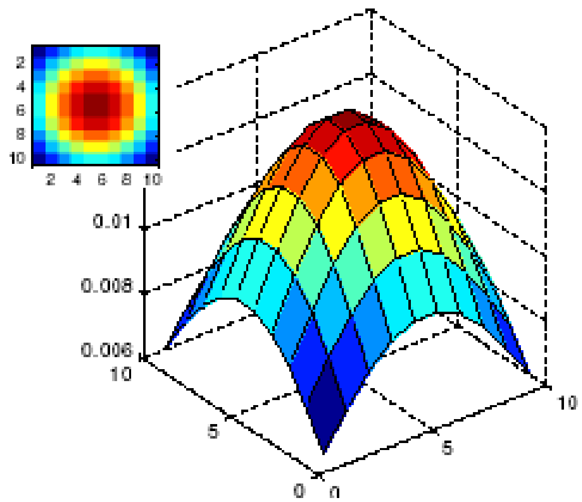
Compare the result with a box filter



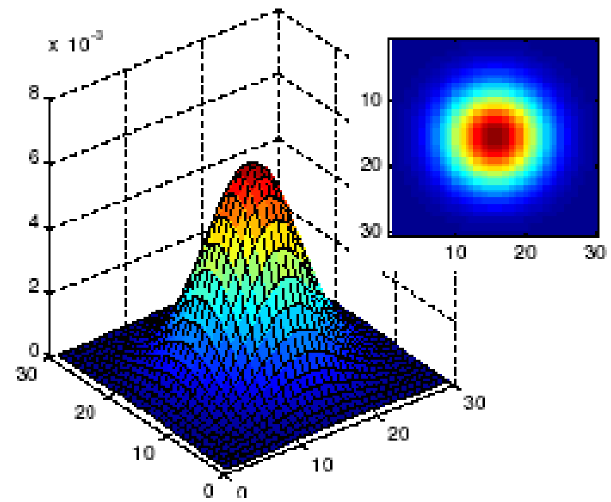


Gaussian filters

- What parameters matter?
- **Size** of kernel or mask
  - Note, Gaussian function has infinite support, but discrete filters use finite kernels



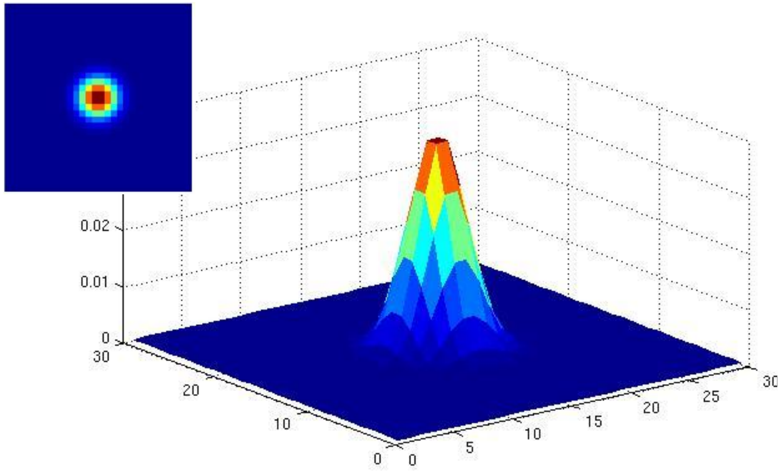
σ = 5 with 10 x 10 kernel



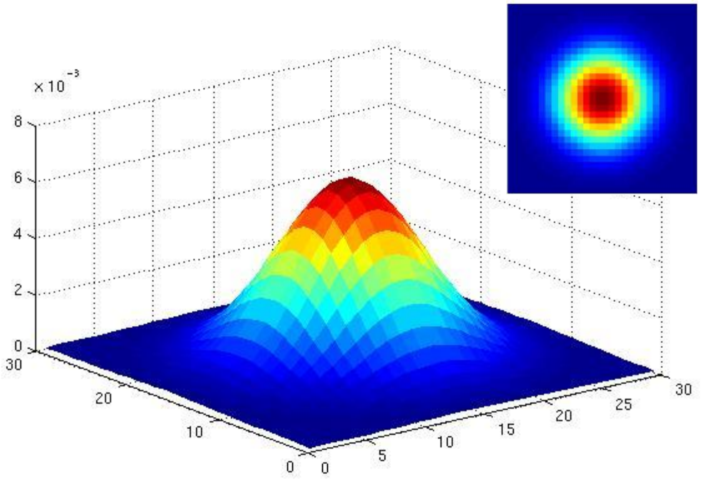
σ = 5 with 30 x 30 kernel

Gaussian filters

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



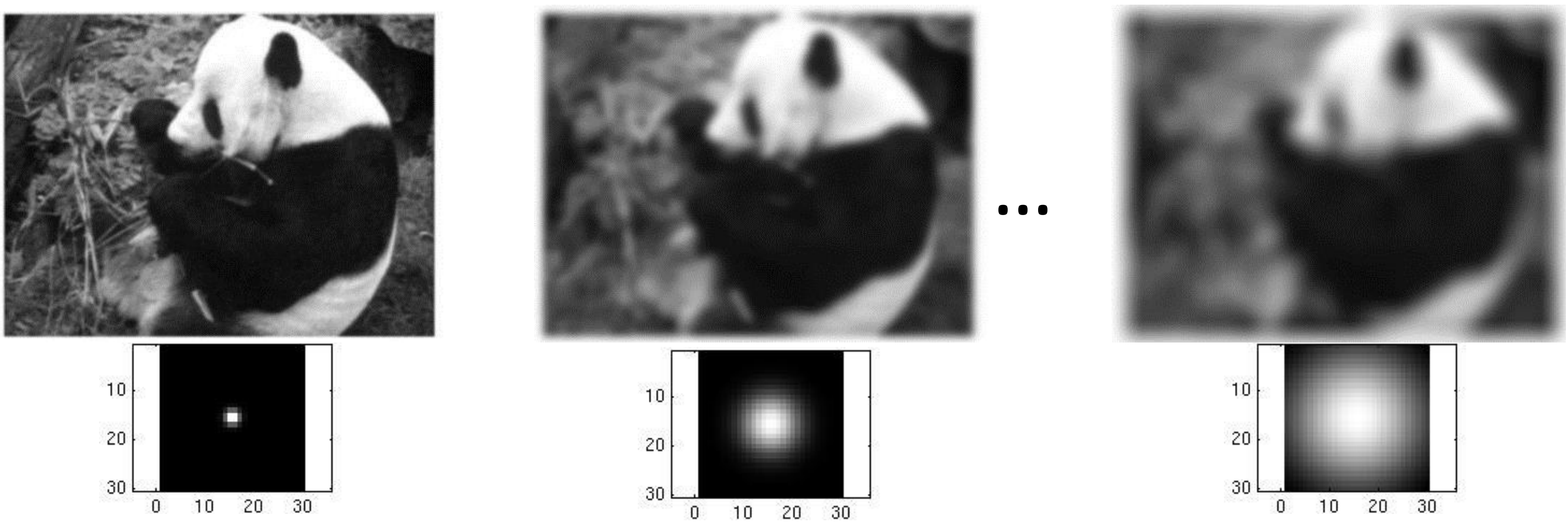
σ = 2 with 30 x 30 kernel



σ = 5 with 30 x 30 kernel

Smoothing with a Gaussian

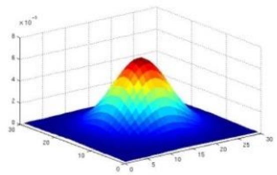
Parameter σ is the "scale" / "width" / "spread" of the Gaussian kernel, and controls the amount of smoothing.



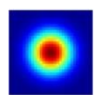
Sample Matlab code

```
>> hsize = 10;
>> 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

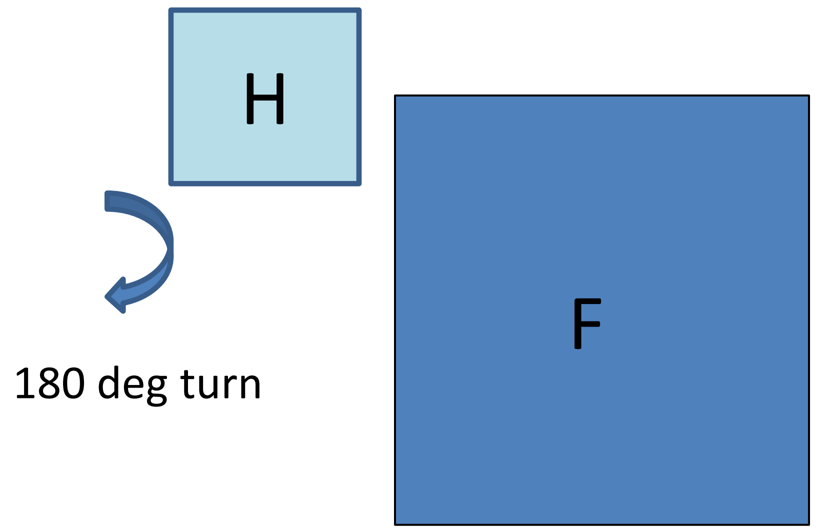
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

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

$$G = H \star F$$

Notation for convolution operator



Convolution vs. correlation

Convolution

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

$$G = H \star F$$

Cross-correlation

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

$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

Summary on filters

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter

Effect of smoothing filters

5x 5

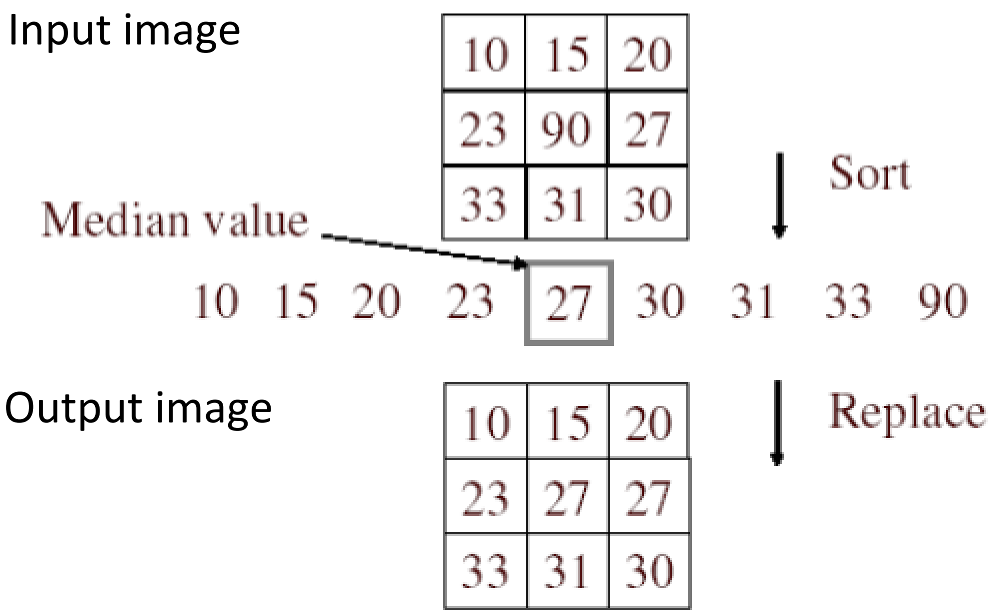
Additive Gaussian noise

Salt and pepper noise

Linear smoothing filters do not alleviate salt and pepper noise!

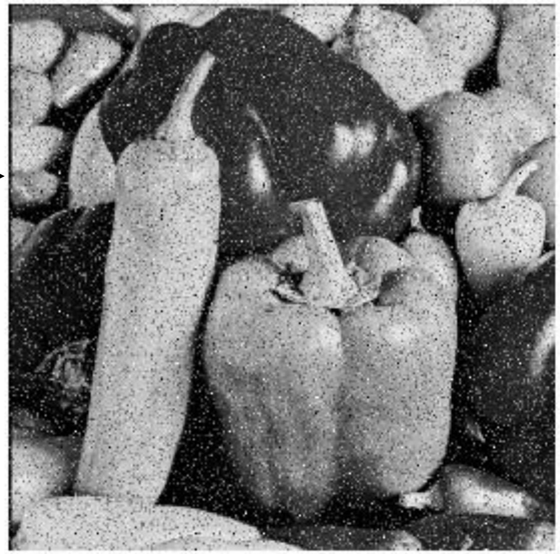
Median filter

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

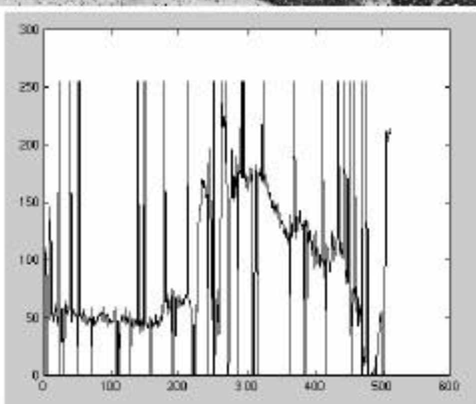


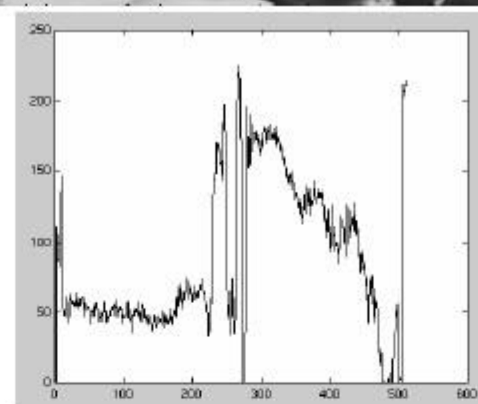
Median filter

Salt and pepper noise



Median filtered

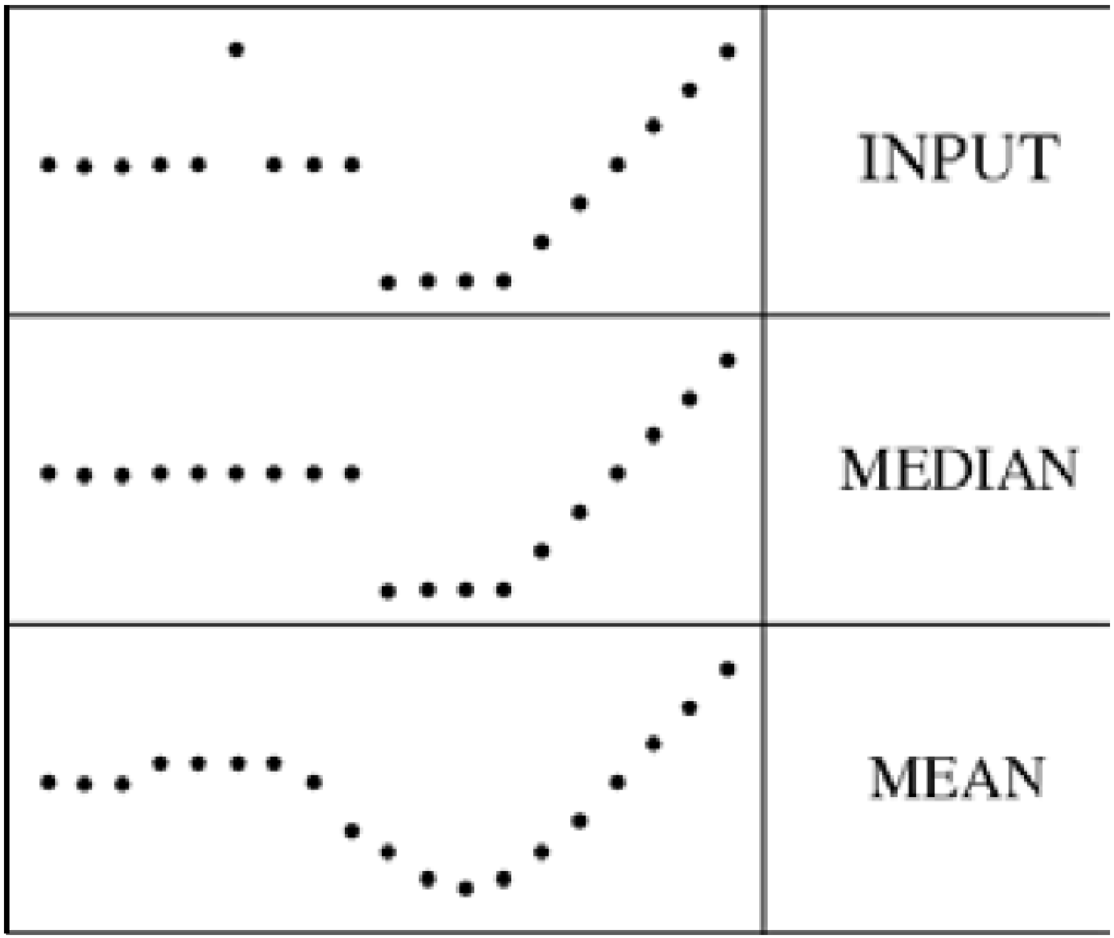




Plots of a row of the image

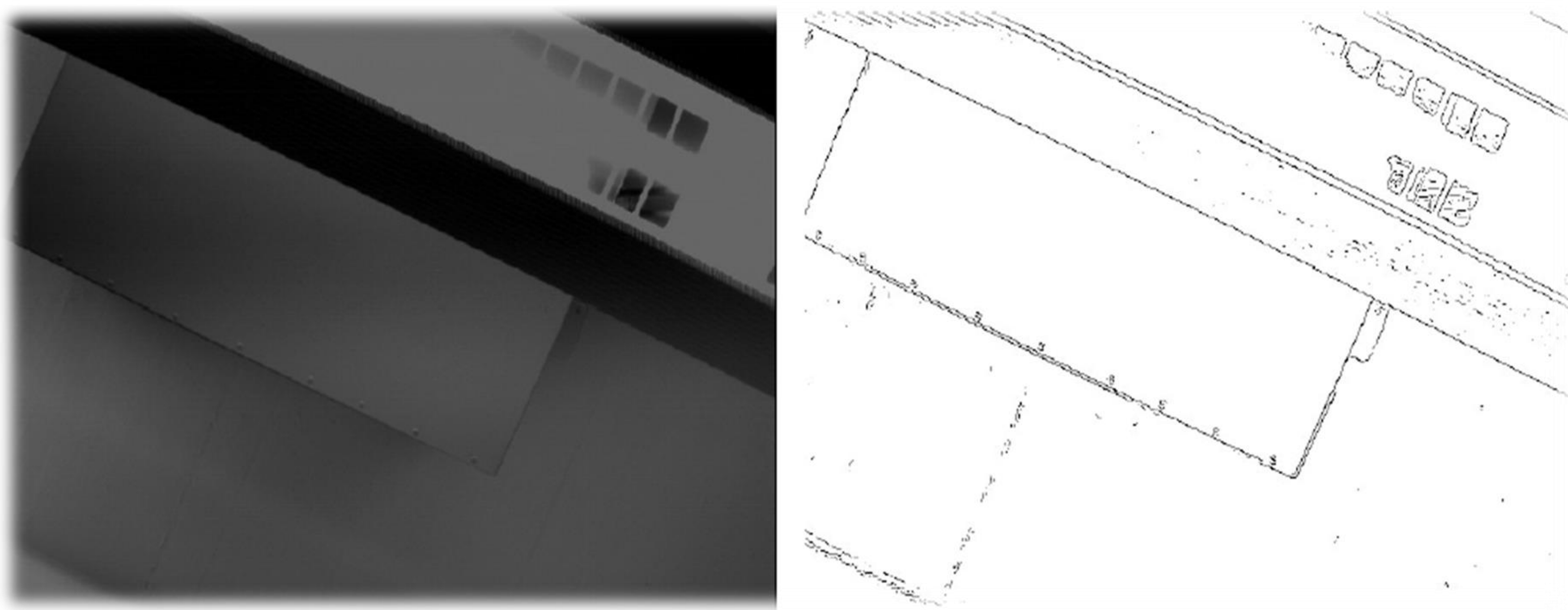
Median filter

- Median filter is edge preserving

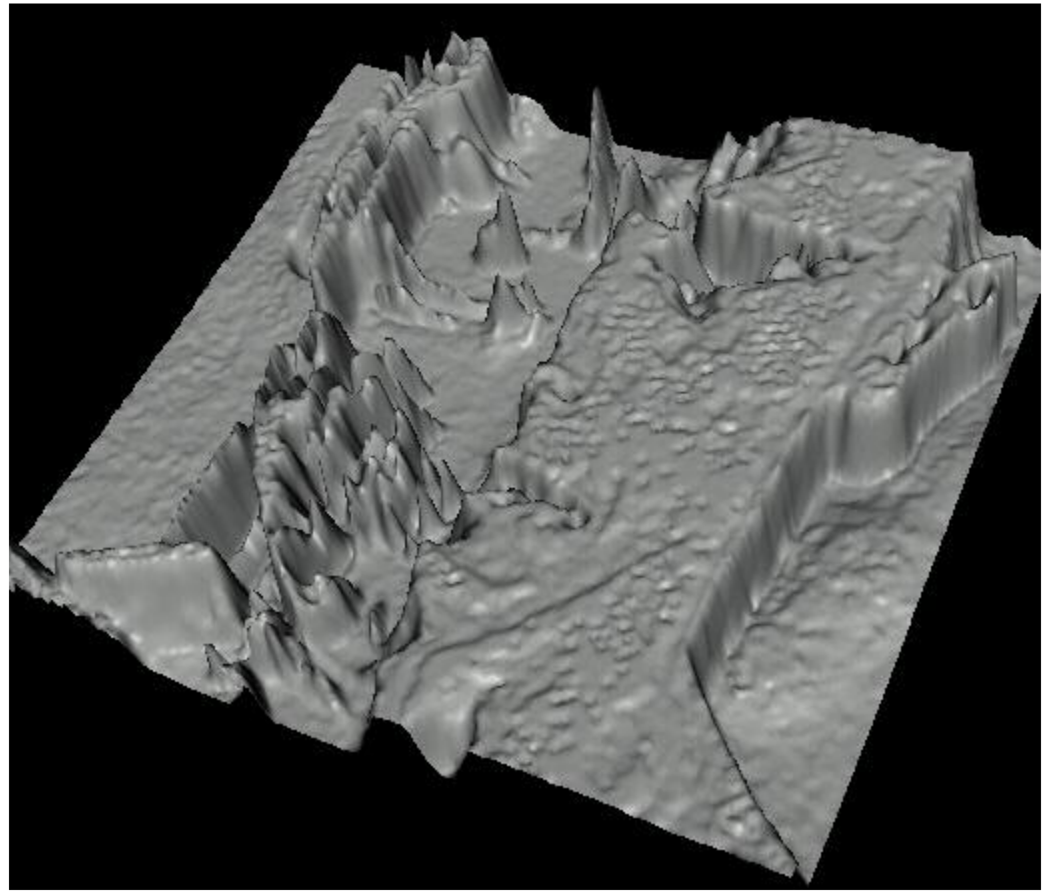


Edge detection

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



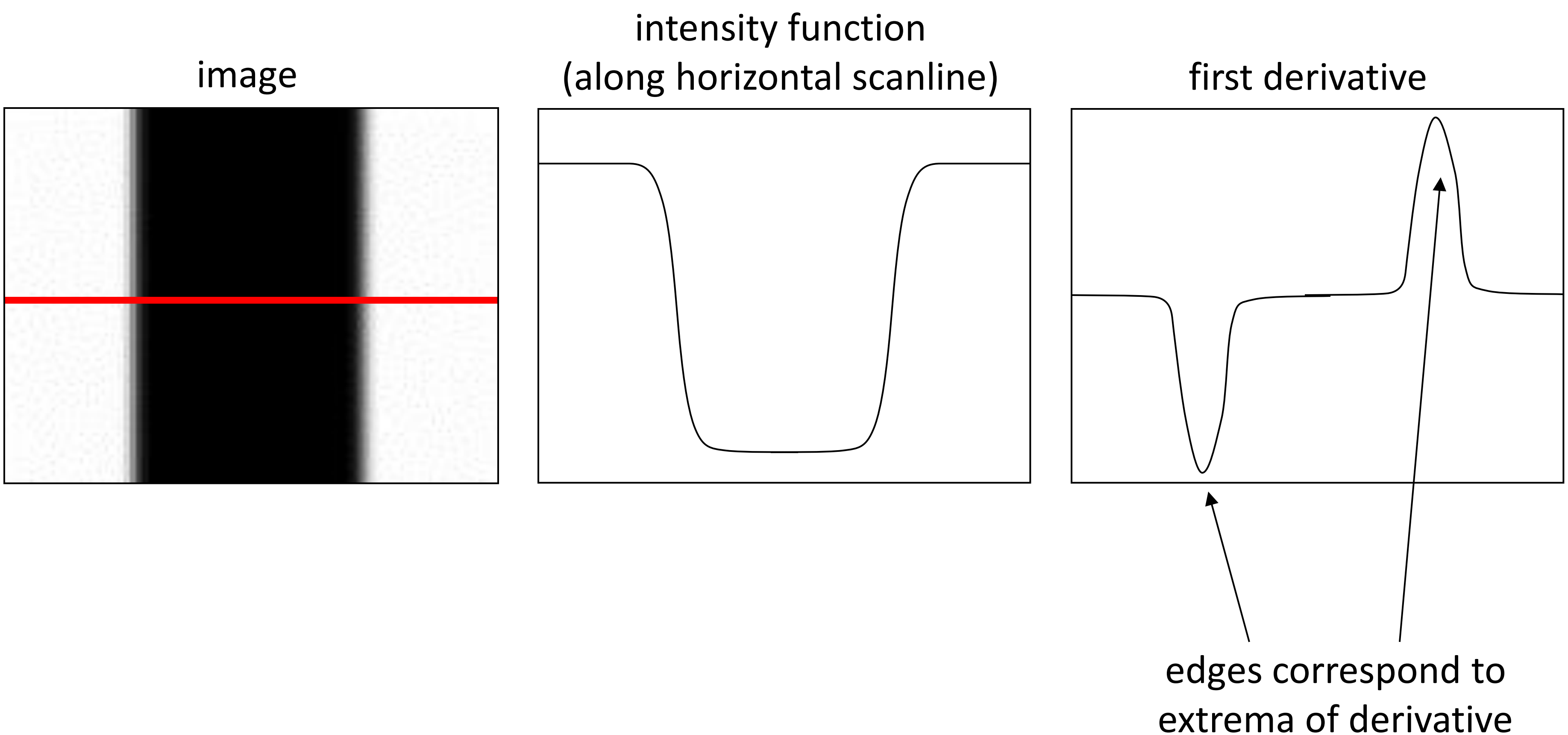
Images as functions $f(x, y)$



- Edges look like steep cliffs

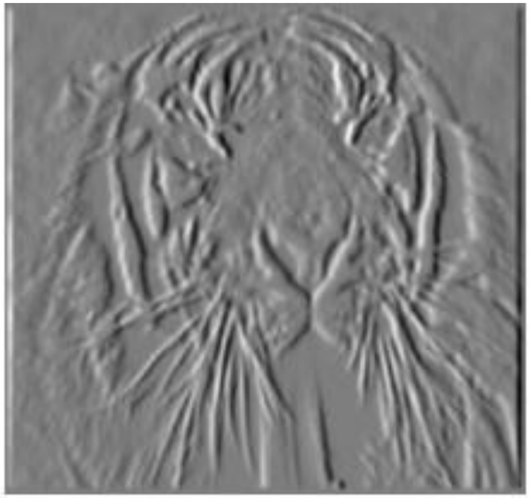
Derivatives and edges

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



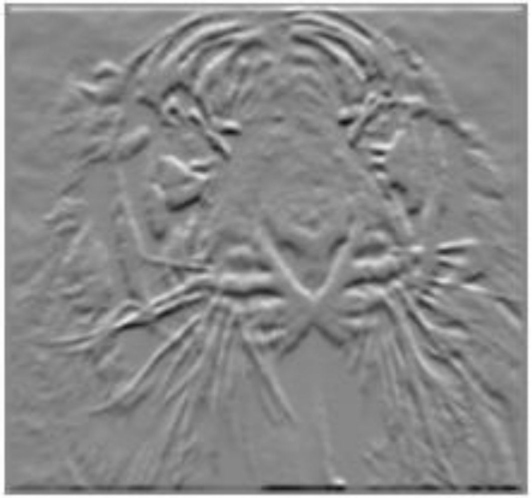
Partial derivatives of an image

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



-1	1

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



-1
1

Alternative Finite-difference filters

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

Sobel filter $\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$

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

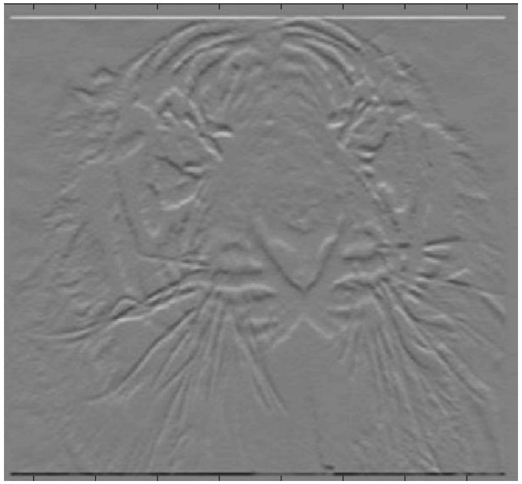
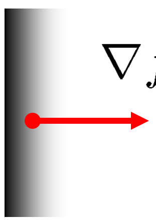


Image gradient

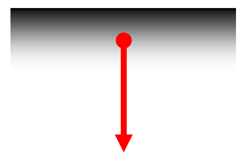
The gradient of an image:

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

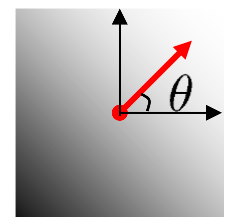
The gradient points in the direction of fastest intensity change



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



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



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

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

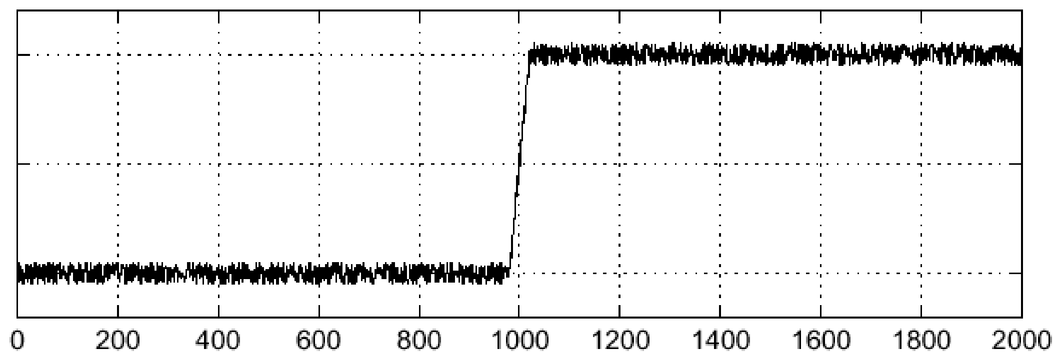
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Effects of noise

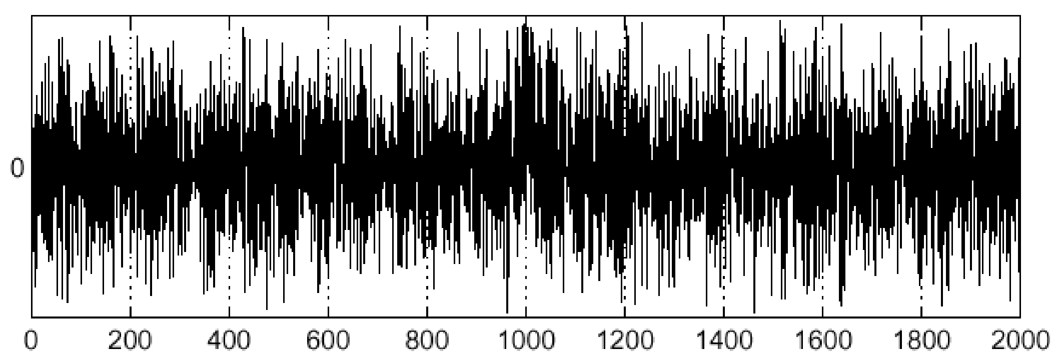
Consider a single row or column of the image

– Plotting intensity as a function of position gives a signal

$f(x)$

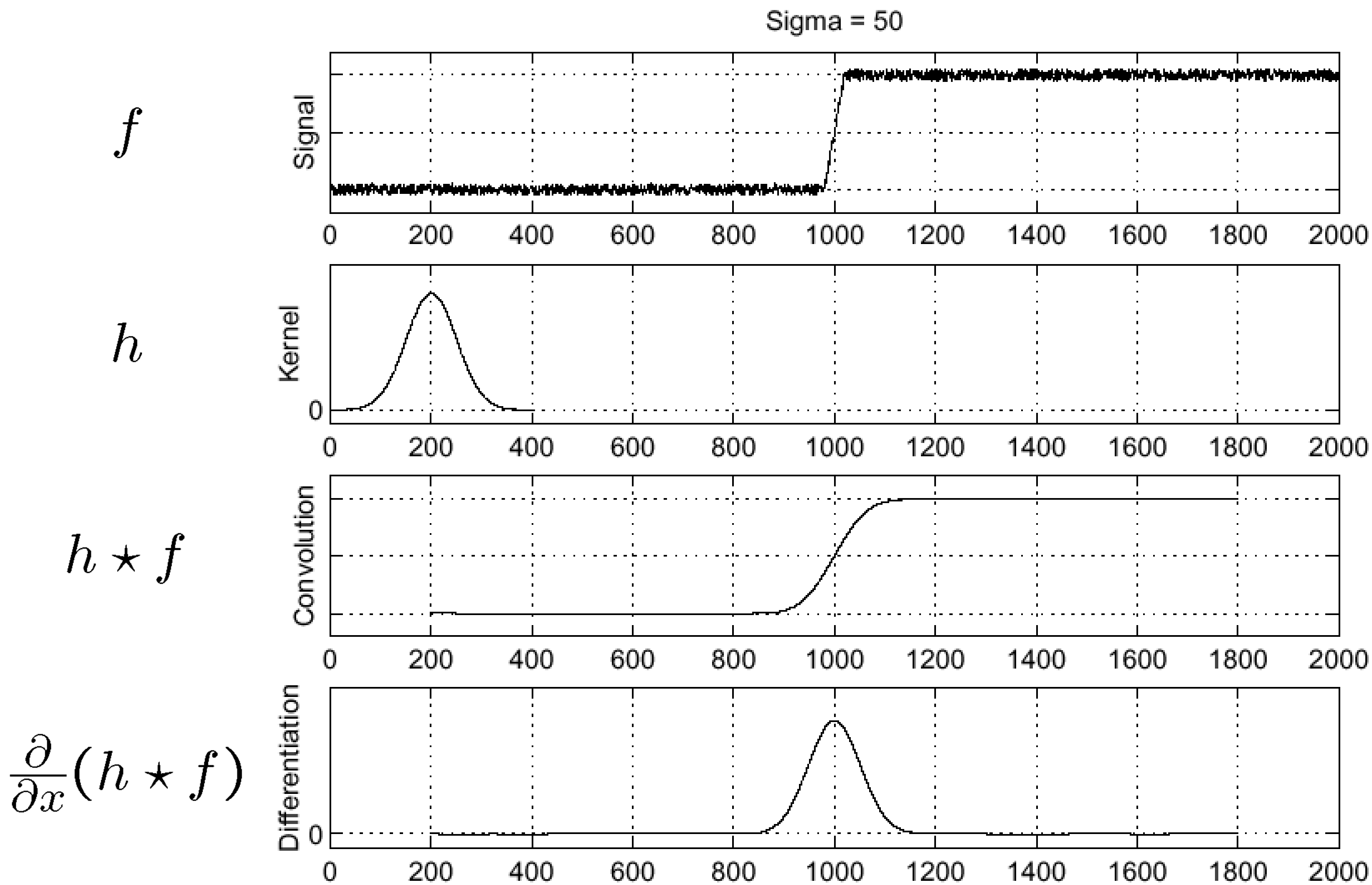


$\frac{d}{dx}f(x)$



Where is the edge?

Solution: smooth first



Where is the edge?

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

Derivative theorem of convolution

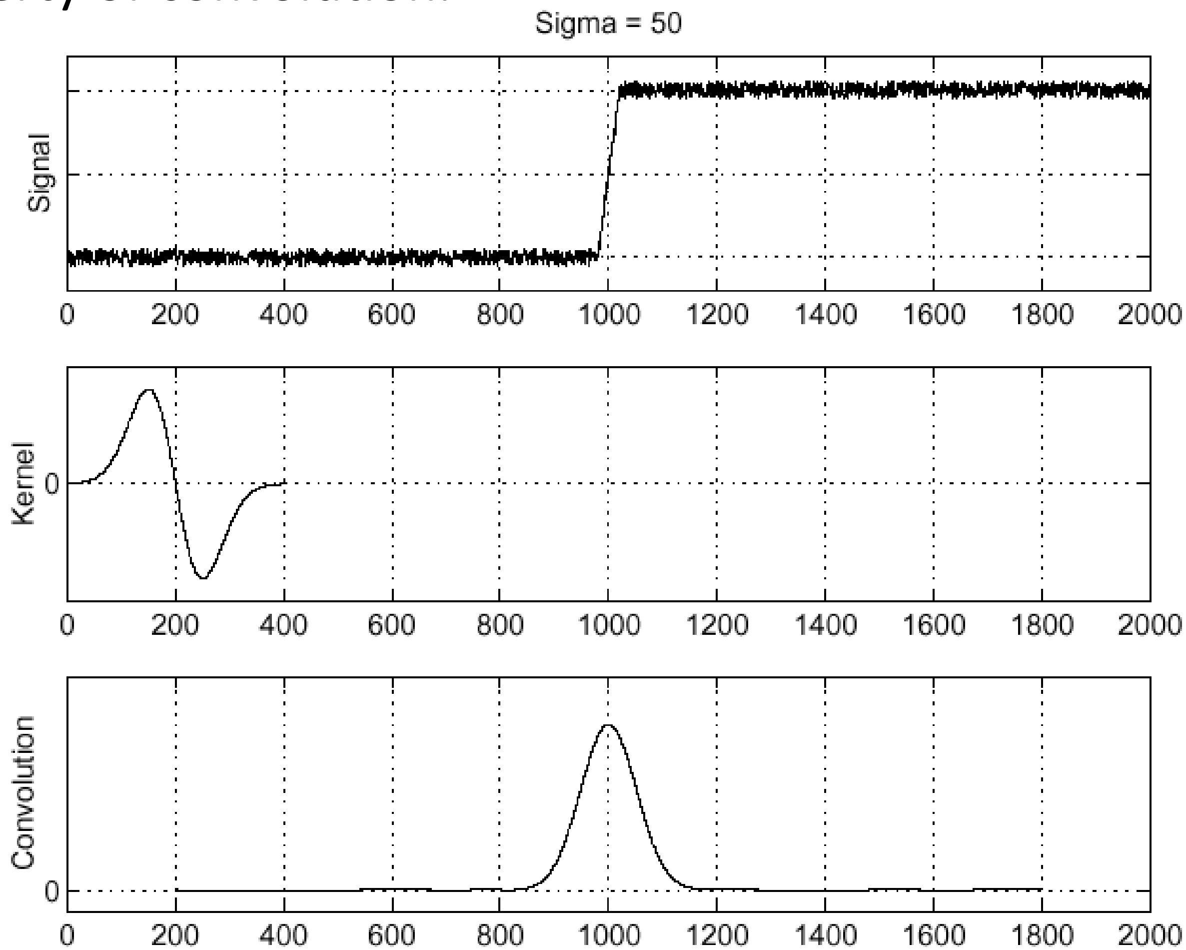
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Differentiation property of convolution.

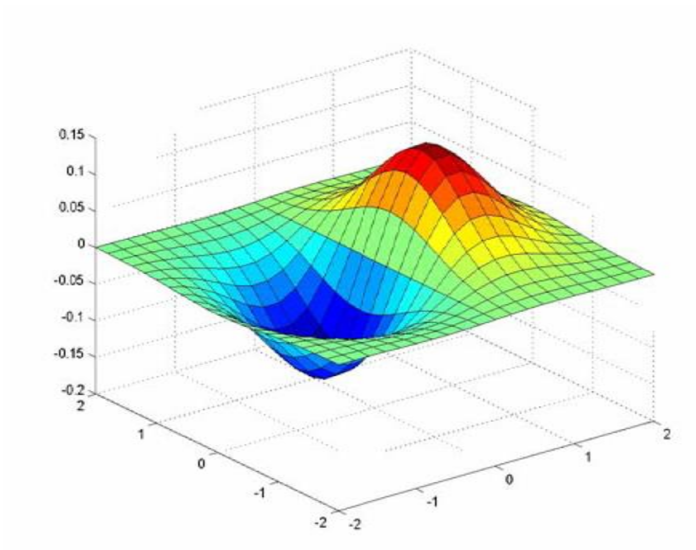
f

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

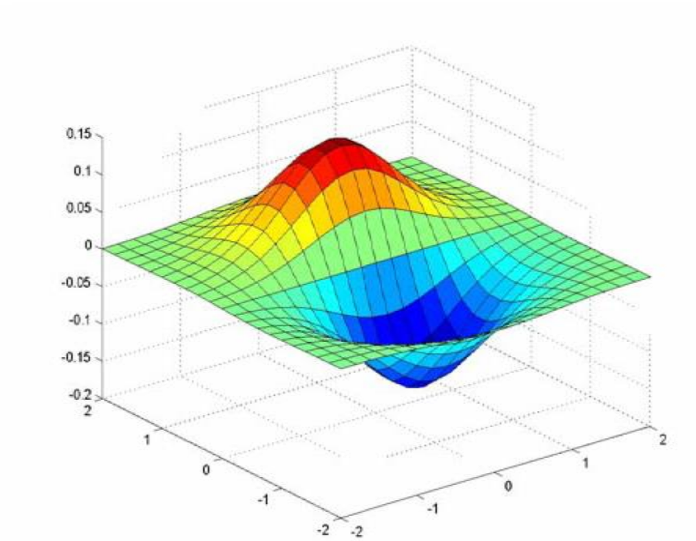
$(\frac{\partial}{\partial x}h) \star f$



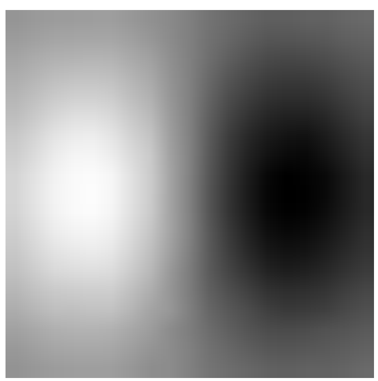
Derivative of Gaussian filters

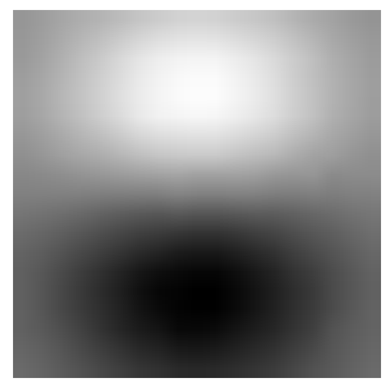


x-direction



y-direction





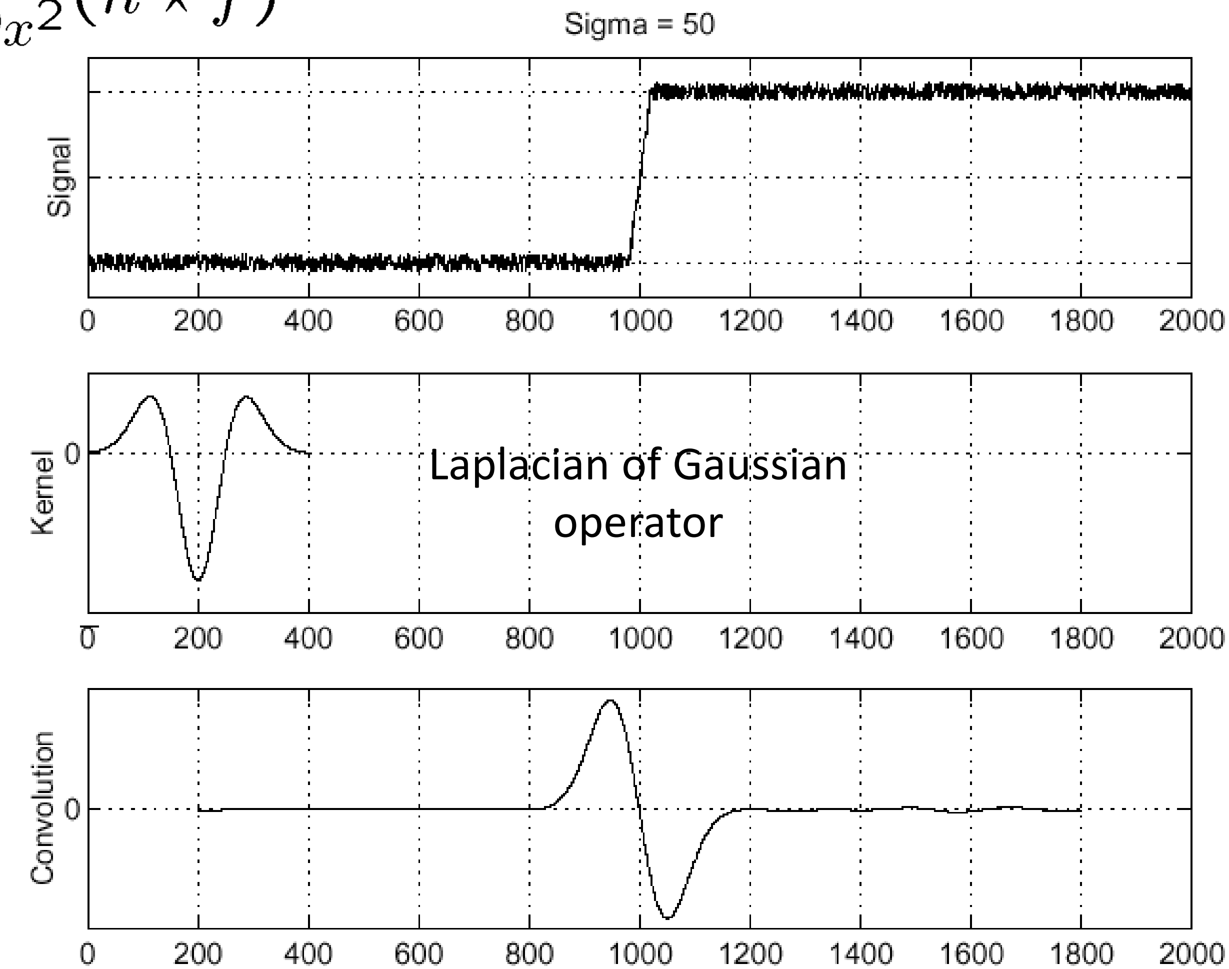
Laplacian of Gaussian

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

f

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

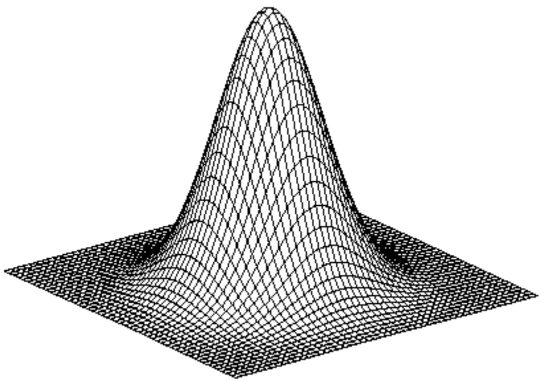
$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge?

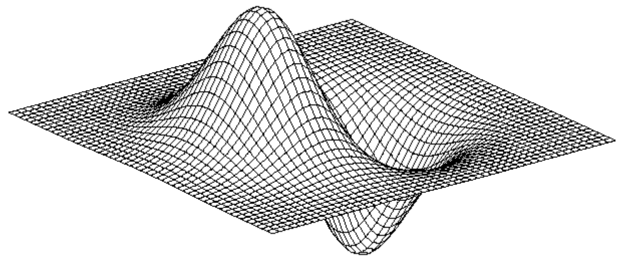
Zero-crossings of bottom graph

2D edge detection filters



Gaussian

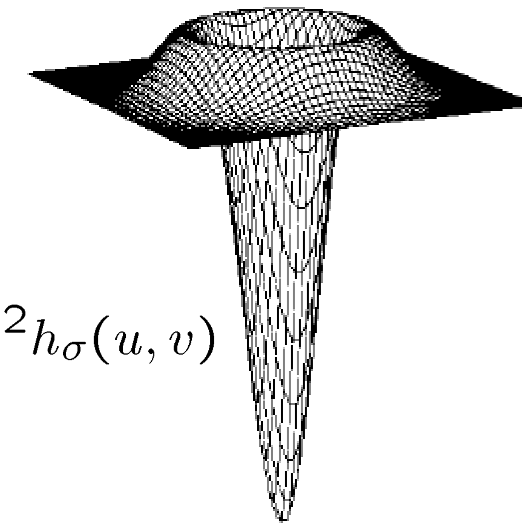
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



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 filters

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter

- Derivatives
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 → no response in constant regions
  - High absolute value at points of high contrast

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)

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)

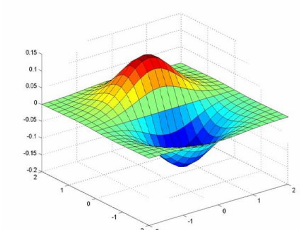
Take a grayscale image. If not grayscale (i.g., RGB), convert it into a grayscale by replacing each pixel by the mean value of its R, G, B components.

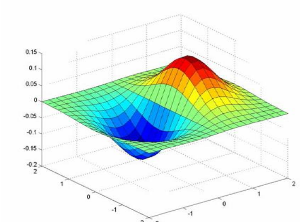
Original image (Lenna image: https://en.wikipedia.org/wiki/Lenna)

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 who 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, sharpening filter
  - Differences and properties
  - Boundary issues
  - Correlation vs convolution
- Non-linear filters
  - Median filter and its applications
- Edge detection
  - Derivating filters (Prewitt, Sobel)
  - Convolution theorem
  - Laplacian of Gaussian
  - Canny edge detector
- Book chapters 3.2, pages 108-109, 386-387, 4.2.1, 11.3.1