

# Computer Vision - Lecture 3

## Linear Filters

23.10.2014

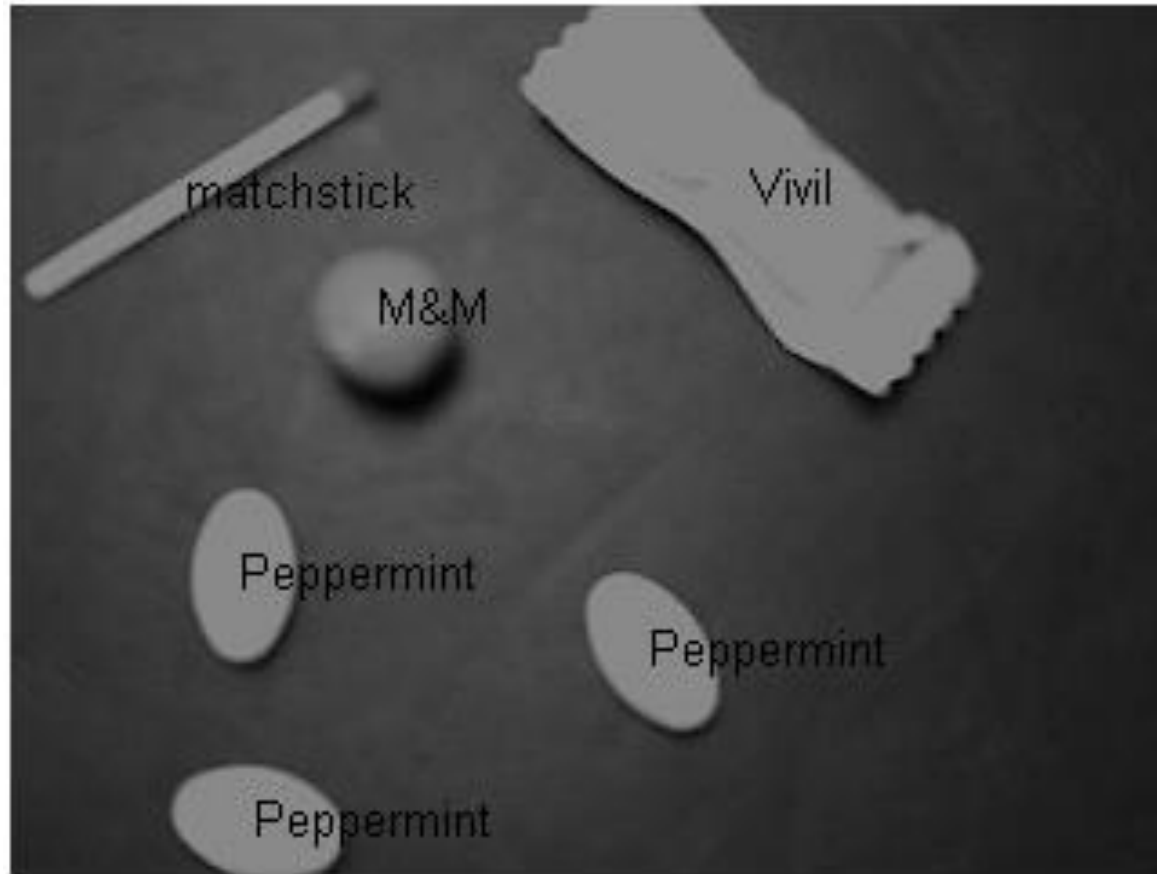
Bastian Leibe

RWTH Aachen

<http://www.vision.rwth-aachen.de>

[leibe@vision.rwth-aachen.de](mailto:leibe@vision.rwth-aachen.de)

# Demo “Haribo Classification”



*Code available on the class website...*

# You Can Do It At Home...

## Accessing a webcam in Matlab:

```
function out = webcam
% uses "Image Acquisition Toolbox"
adaptorName = 'winvideo';
vidFormat = 'I420_320x240';
vidObj1= videoinput(adaptorName, 1, vidFormat);
set(vidObj1, 'ReturnedColorSpace', 'rgb');
set(vidObj1, 'FramesPerTrigger', 1);
out = vidObj1 ;

cam = webcam();
img=getsnapshot(cam);
```



# Course Outline

- **Image Processing Basics**
  - Image Formation
  - Binary Image Processing
  - **Linear Filters**
  - Edge & Structure Extraction
  - Color
- **Segmentation**
- **Local Features & Matching**
- **Object Recognition and Categorization**
- **3D Reconstruction**
- **Motion and Tracking**

# Motivation

- Noise reduction/image restoration

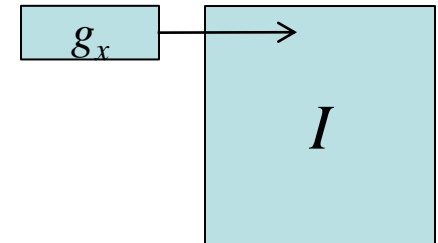


- Structure extraction



# Topics of This Lecture

- **Linear filters**
  - What are they? How are they applied?
  - Application: smoothing
  - Gaussian filter
  - What does it *mean* to filter an image?
- **Nonlinear Filters**
  - Median filter
- **Multi-Scale representations**
  - How to properly rescale an image?
- **Filters as templates**
  - Correlation as template matching



# Common Types of Noise

- Salt & 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.
- *Basic Assumption*
  - *Noise is i.i.d. (independent & identically distributed)*



Original



Salt and pepper noise



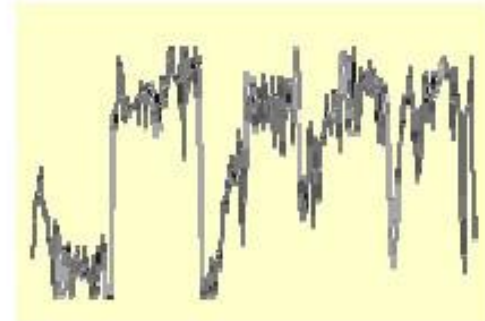
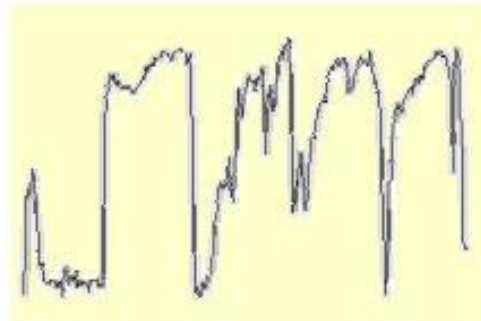
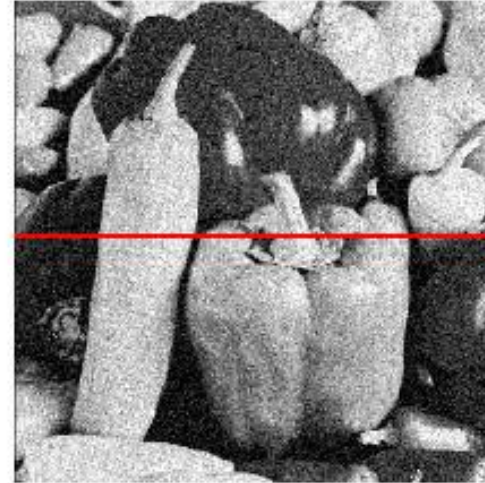
Impulse noise



Gaussian noise



# Gaussian Noise



$$f(x, y) = \underbrace{\hat{f}(x, y)}_{\text{Ideal Image}} + \underbrace{\eta(x, y)}_{\text{Noise process}}$$

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

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

```
>> output = im + noise;
```

B. Leibe



# First Attempt at a Solution

- **Assumptions:**
  - Expect pixels to be like their neighbors
  - Expect noise processes to be independent from pixel to pixel (“i.i.d. = independent, identically distributed”)
- **Let’s try to replace each pixel with an average of all the values in its neighborhood...**

# 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	



# Correlation Filtering

- Say the averaging window size is  $2k+1 \times 2k+1$ :

$$G[i, j] = \frac{1}{(2k+1)^2} \sum_{u=-k}^k \sum_{v=-k}^k F[i+u, j+v]$$

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:

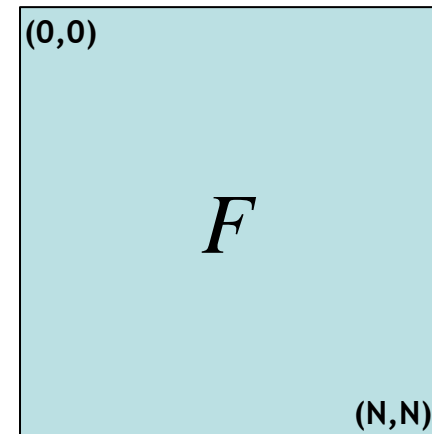
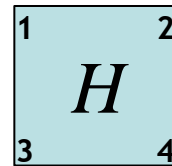
$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i+u, j+v]$$

Non-uniform weights

# Correlation Filtering

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

- This is called cross-correlation, denoted  $G = H \otimes F$
- Filtering an image
  - Replace each pixel by a weighted combination of its neighbors.
  - The filter “kernel” or “mask” is the prescription for the weights in the linear combination.

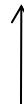


# Convolution

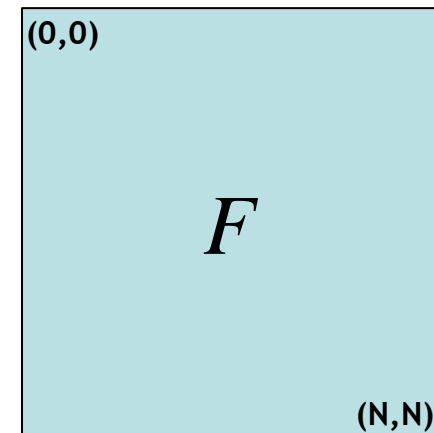
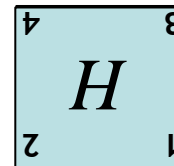
- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

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

$$G = H \star F$$



*Notation for  
convolution  
operator*



# Correlation vs. Convolution

- Correlation

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

$$G = H \otimes F$$

Matlab:  
`filter2`  
`imfilter`

Note the difference!

- Convolution

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

$$G = H \star F$$

Matlab:  
`conv2`

- Note

- If  $H[-u, -v] = H[u, v]$ , then correlation  $\equiv$  convolution.

# Shift Invariant Linear System

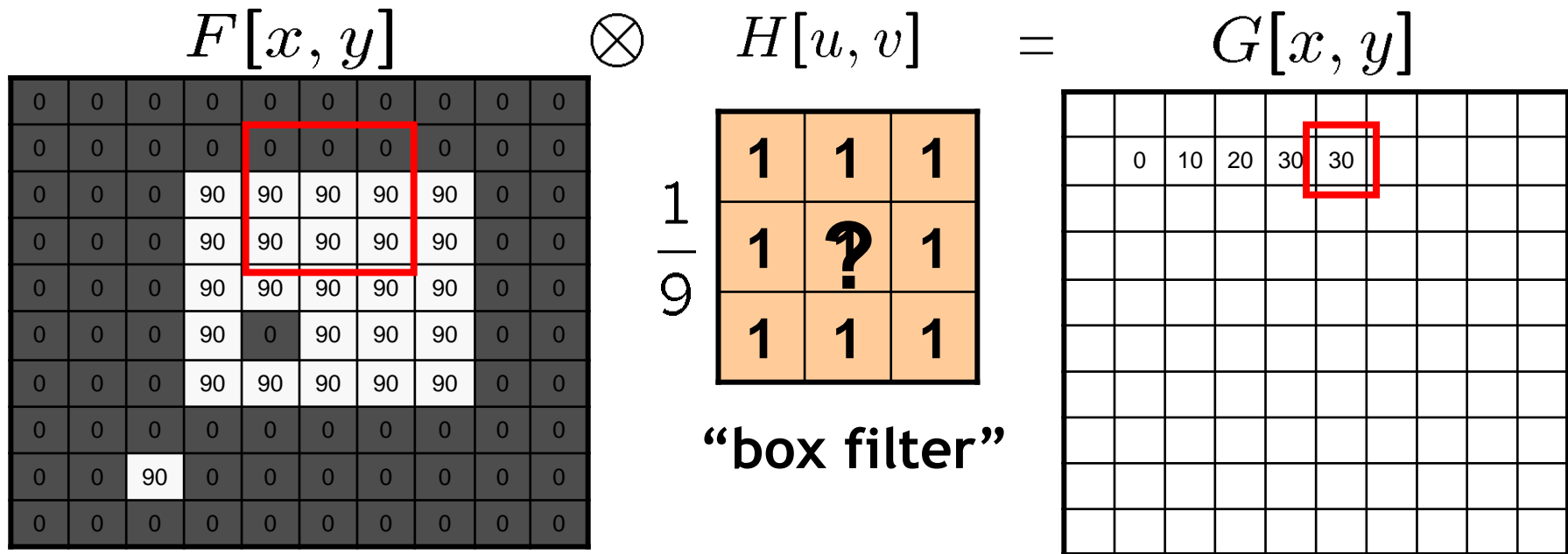
- **Shift invariant:**
  - Operator behaves the same everywhere, *i.e.* the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.
- **Linear:**
  - **Superposition:**  $h * (f_1 + f_2) = (h * f_1) + (h * f_2)$
  - **Scaling:**  $h * (kf) = k(h * f)$

# Properties of Convolution

- Linear & shift invariant
- Commutative:  $f \star g = g \star f$
- Associative:  $(f \star g) \star h = f \star (g \star h)$ 
  - Often apply several filters in sequence:  $((a \star b_1) \star b_2) \star b_3$
  - This is equivalent to applying one filter:  $a \star (b_1 \star b_2 \star b_3)$
- Identity:  $f \star e = f$ 
  - for unit impulse  $e = [\dots, 0, 0, 1, 0, 0, \dots]$ .
- Differentiation:  $\frac{\partial}{\partial x}(f \star g) = \frac{\partial f}{\partial x} \star g$

# Averaging Filter

- What values belong in the kernel  $H[u, v]$  for the moving average example?



$$G = H \otimes F$$



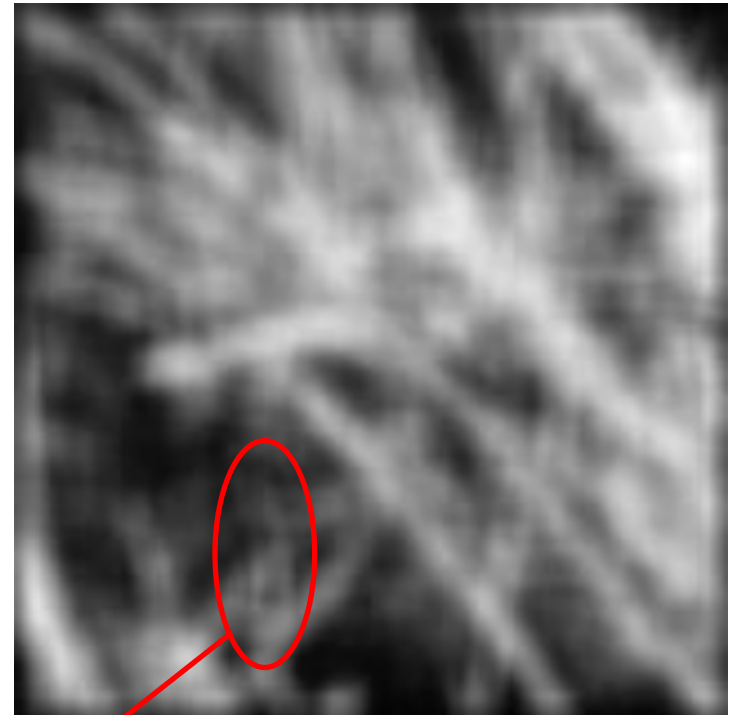
# Smoothing by Averaging



depicts box filter:  
white = high value, black = low value



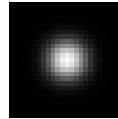
Original



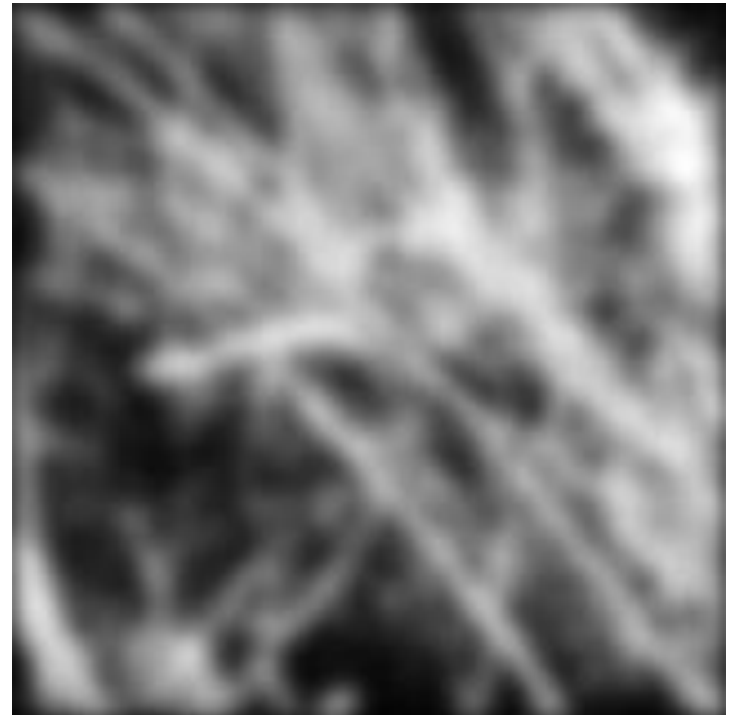
Filtered

**“Ringing” artifacts!**

# Smoothing with a Gaussian

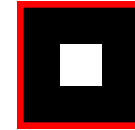
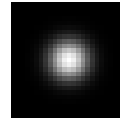


**Original**



**Filtered**

# Smoothing with a Gaussian - Comparison



Original



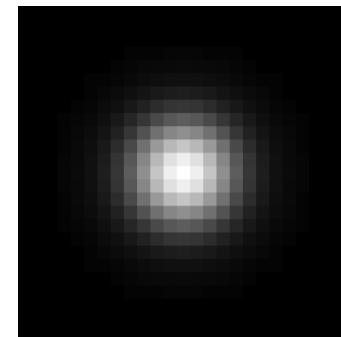
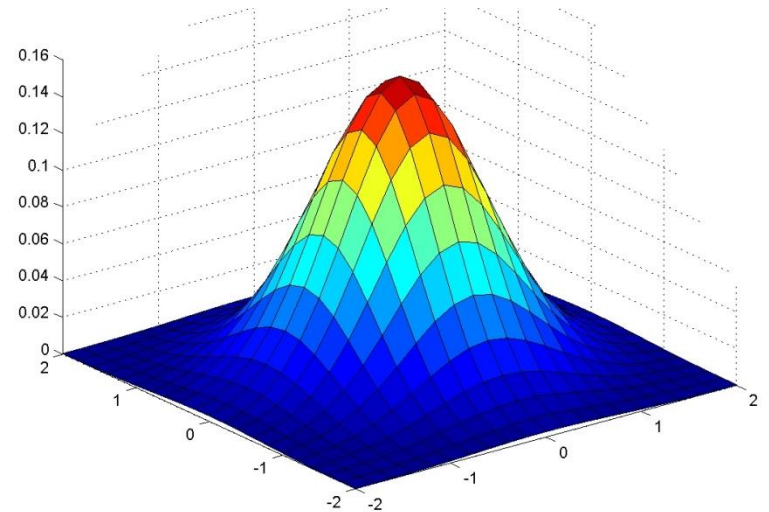
Filtered

# Gaussian Smoothing

- Gaussian kernel

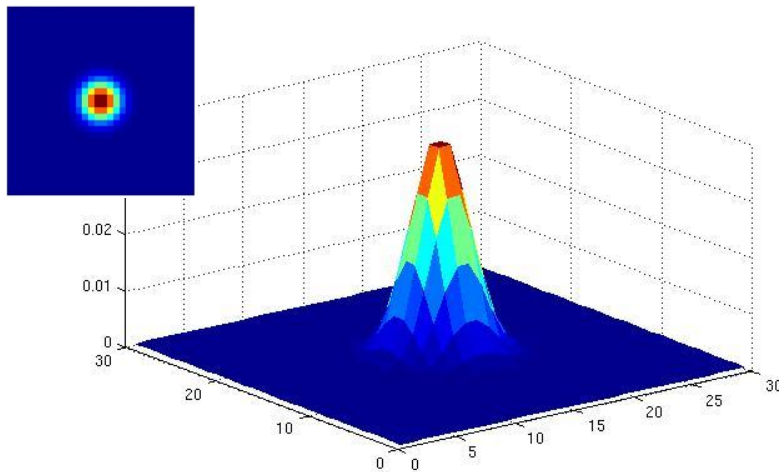
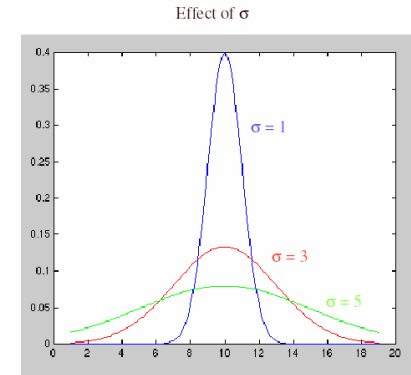
$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Rotationally symmetric
- Weights nearby pixels more than distant ones
  - This makes sense as ‘probabilistic’ inference about the signal
- A Gaussian gives a good model of a fuzzy blob

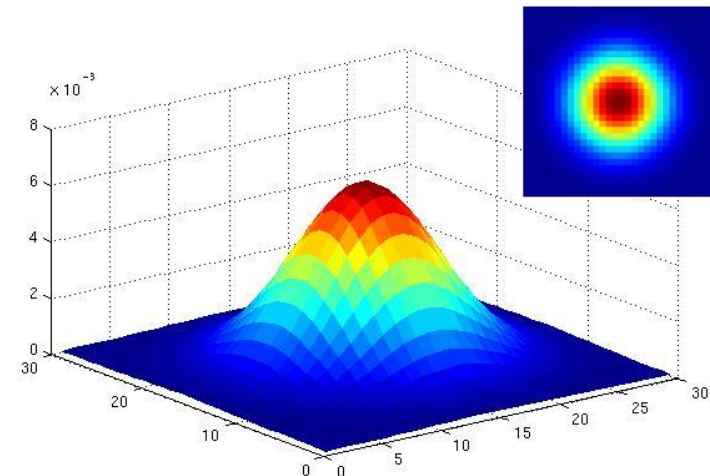


# Gaussian Smoothing

- What parameters matter here?
- *Variance*  $\sigma$  of Gaussian
  - Determines extent of smoothing



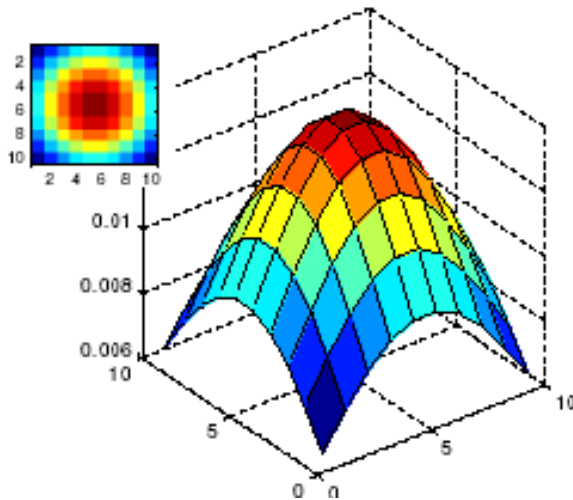
$\sigma = 2$  with  $30 \times 30$   
kernel



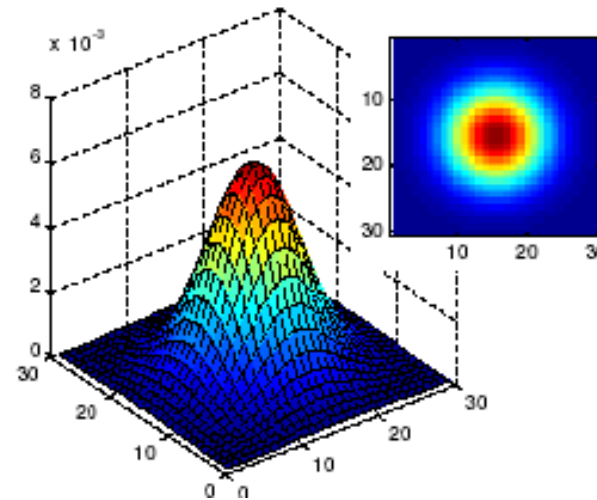
$\sigma = 5$  with  $30 \times 30$   
kernel

# Gaussian Smoothing

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



$\sigma = 5$  with  $10 \times 10$   
kernel



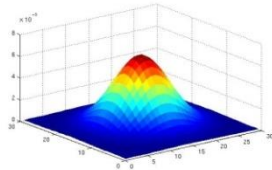
$\sigma = 5$  with  $30 \times 30$   
kernel

- **Rule of thumb: set filter half-width to about  $3\sigma$ !**

# Gaussian Smoothing in Matlab

```
>> hsize = 10;  
>> sigma = 5;  
>> h = fspecial('gaussian' hsize, sigma);
```

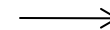
```
>> mesh(h);
```



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



```
>> outim = imfilter(im, h);  
>> imshow(outim);
```



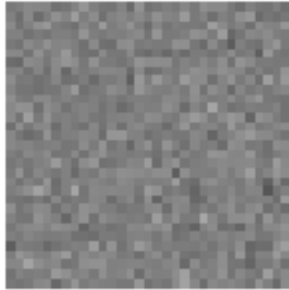
outim



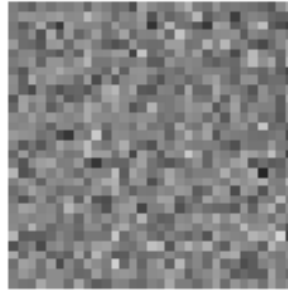
# Effect of Smoothing

More noise →

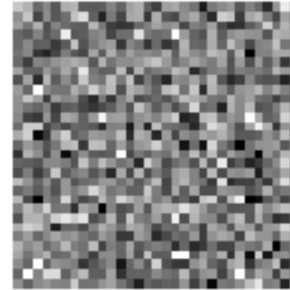
$\sigma=0.05$



$\sigma=0.1$



$\sigma=0.2$



no  
smoothing

Wider smoothing kernel →

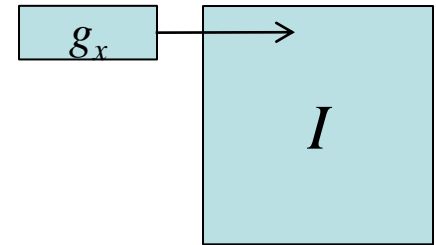


# Efficient Implementation

- Both, the BOX filter and the Gaussian filter are separable:

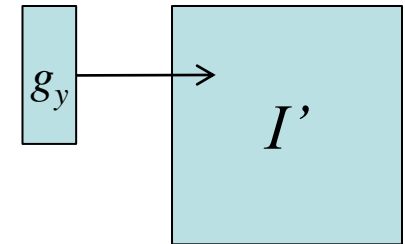
- First convolve each row with a 1D filter

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-x^2 / (2\sigma^2))$$



- Then convolve each column with a 1D filter

$$g(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-y^2 / (2\sigma^2))$$



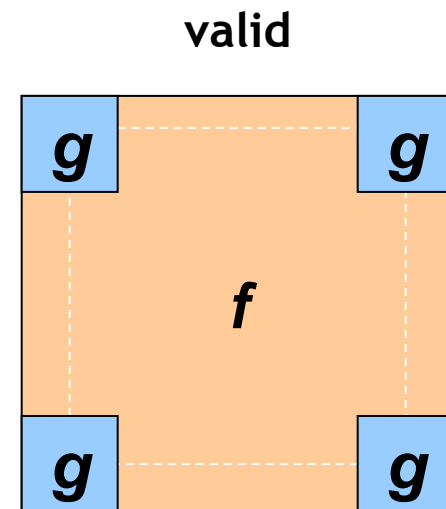
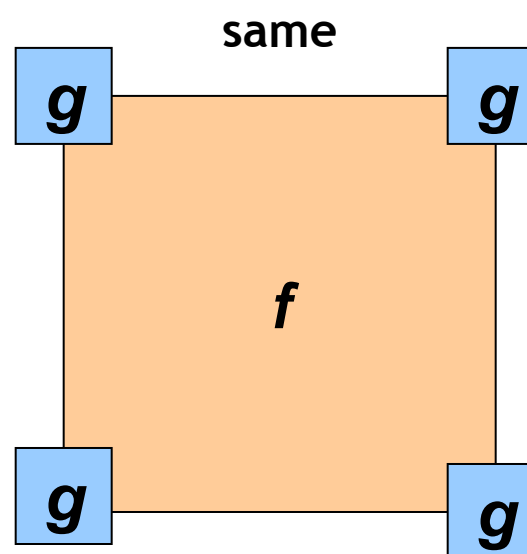
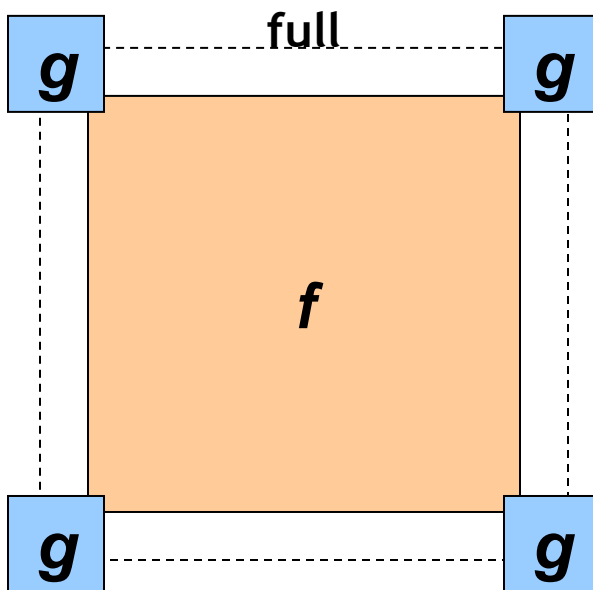
- Remember:

- Convolution is linear - associative and commutative

$$g_x \star g_y \star I = g_x \star (g_y \star I) = (g_x \star g_y) \star I$$

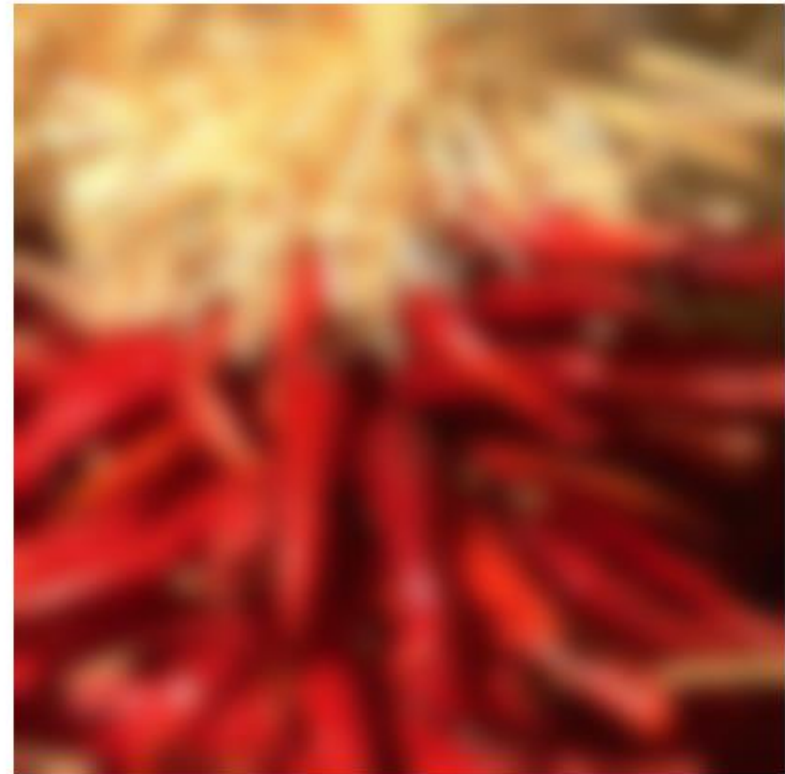
# Filtering: Boundary Issues

- What is the size of the output?
- MATLAB: `filter2(g, f, shape)`
  - *shape* = 'full': output size is sum of sizes of *f* and *g*
  - *shape* = 'same': output size is same as *f*
  - *shape* = 'valid': output size is difference of sizes of *f* and *g*



# Filtering: Boundary Issues

- How should the filter behave near the image boundary?
  - The filter window falls off the edge of the image
  - Need to extrapolate
  - Methods:
    - Clip filter (black)
    - Wrap around
    - Copy edge
    - Reflect across edge



# Filtering: Boundary Issues

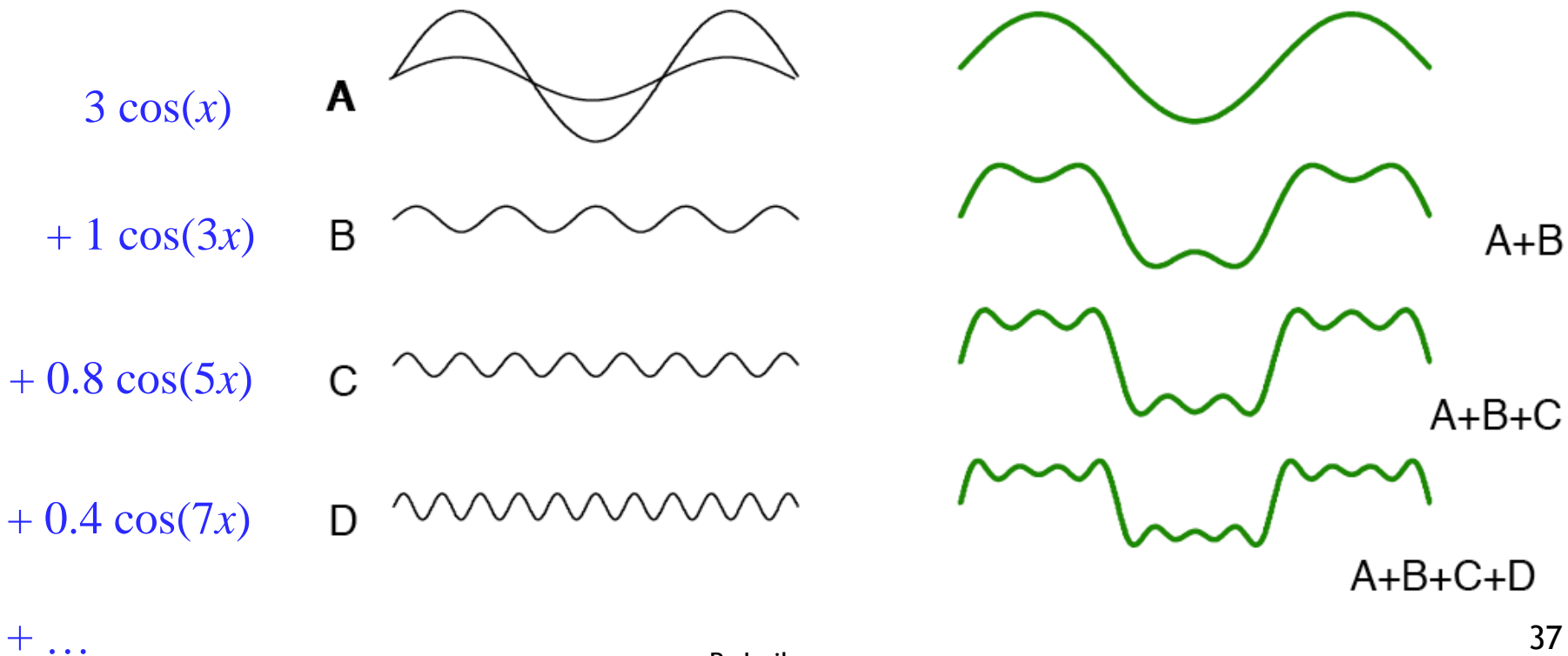
- How should the filter behave near the image boundary?
  - The filter window falls off the edge of the image
  - Need to extrapolate
  - Methods (MATLAB):
    - Clip filter (black): `imfilter(f,g,0)`
    - Wrap around: `imfilter(f,g,'circular')`
    - Copy edge: `imfilter(f,g,'replicate')`
    - Reflect across edge: `imfilter(f,g,'symmetric')`

# Topics of This Lecture

- **Linear filters**
  - What are they? How are they applied?
  - Application: smoothing
  - Gaussian filter
  - What does it *mean* to filter an image?
- Nonlinear Filters
  - Median filter
- Multi-Scale representations
  - How to properly rescale an image?
- Filters as templates
  - Correlation as template matching

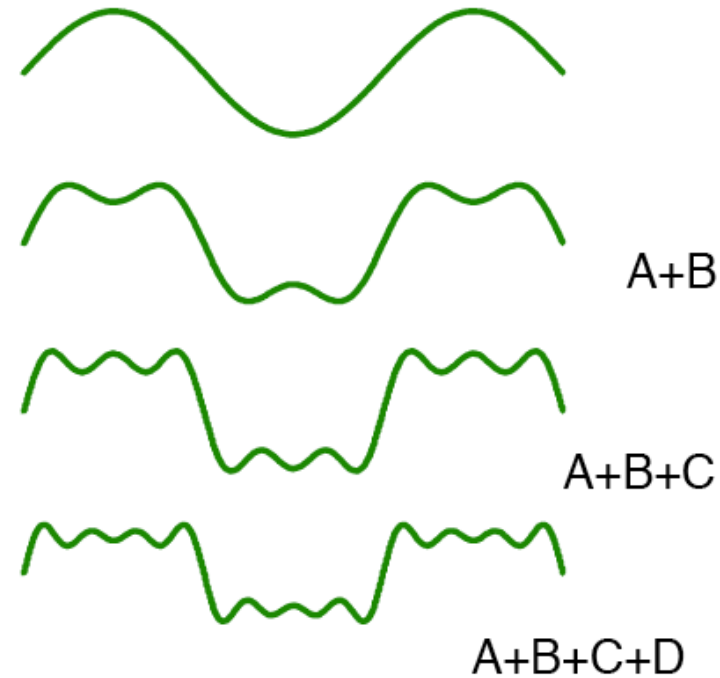
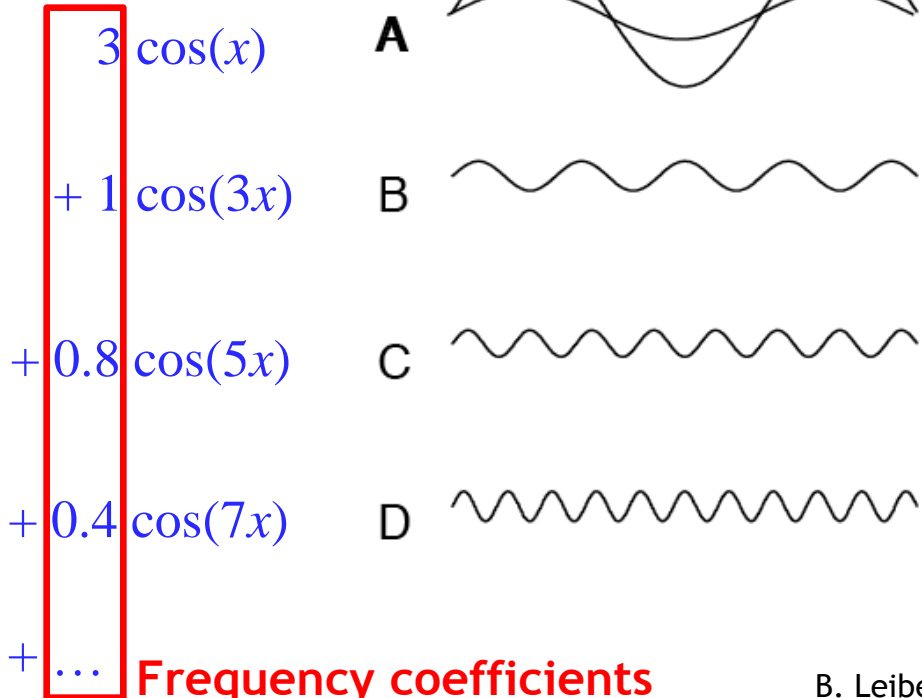
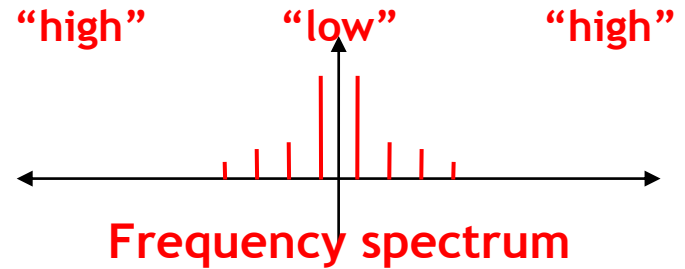
# Why Does This Work?

- A small excursion into the Fourier transform to talk about spatial frequencies...



# The Fourier Transform in Cartoons

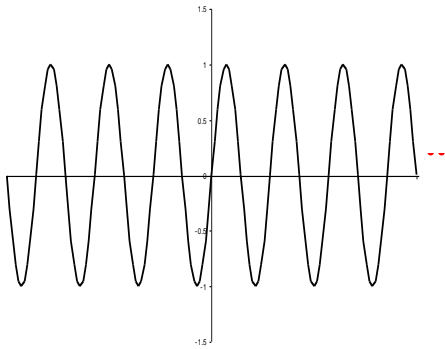
- A small excursion into the Fourier transform to talk about spatial frequencies...



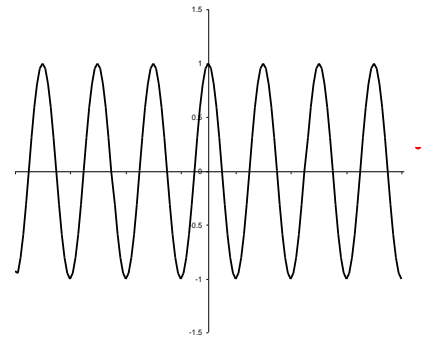


# Fourier Transforms of Important Functions

- Sine and cosine transform to...



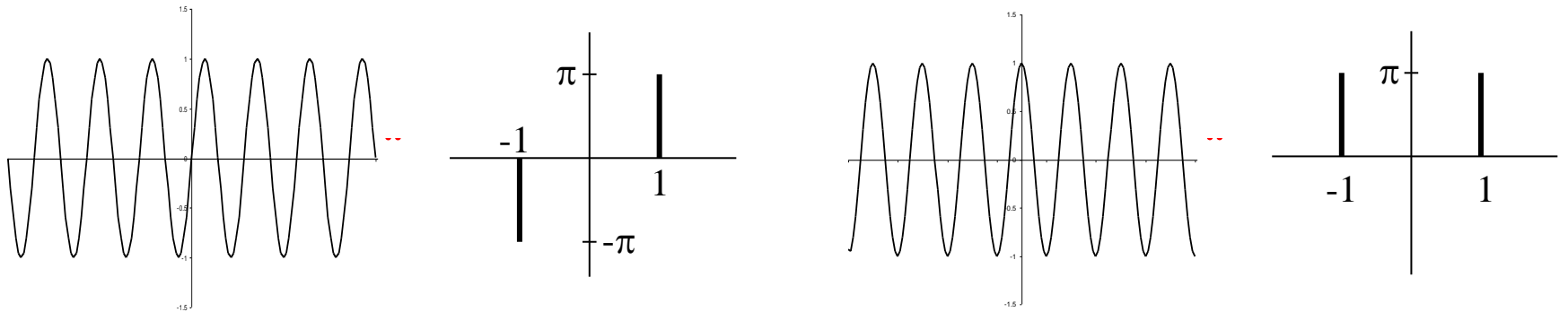
?



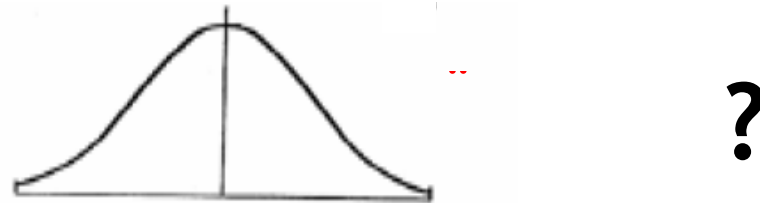
?

# Fourier Transforms of Important Functions

- Sine and cosine transform to “frequency spikes”

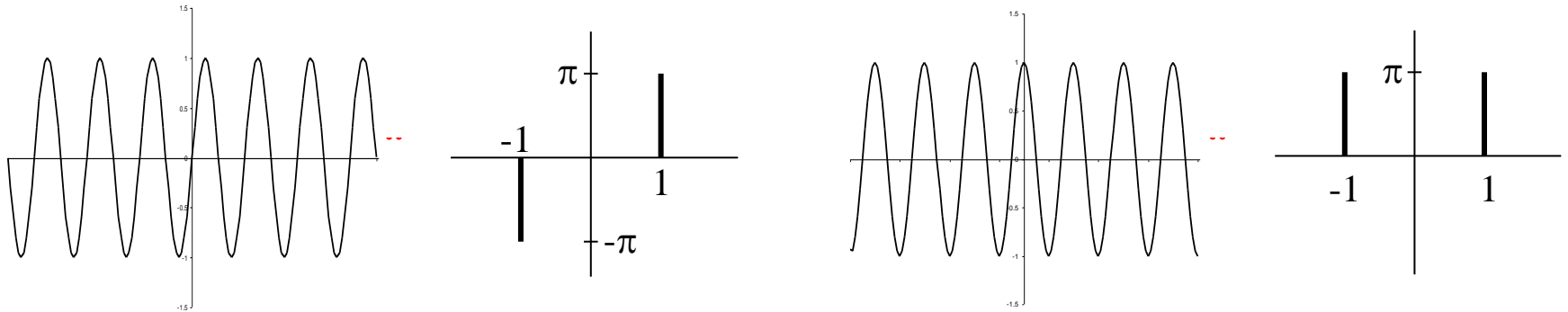


- A Gaussian transforms to...

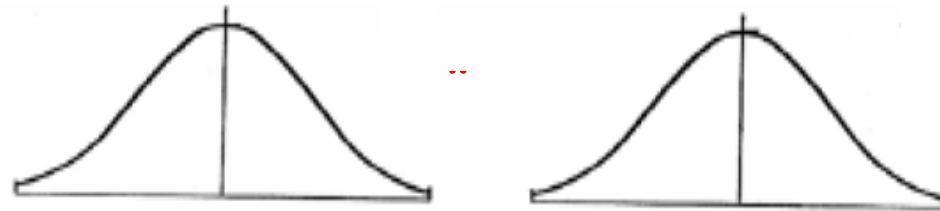


# Fourier Transforms of Important Functions

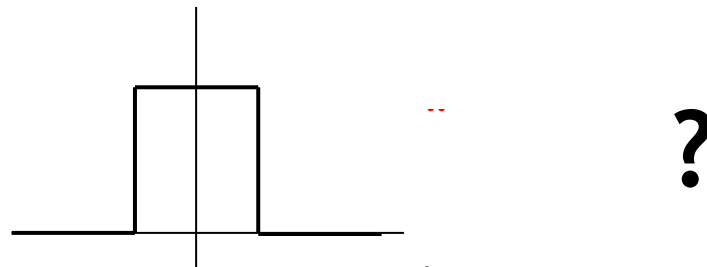
- Sine and cosine transform to “frequency spikes”



- A Gaussian transforms to a Gaussian

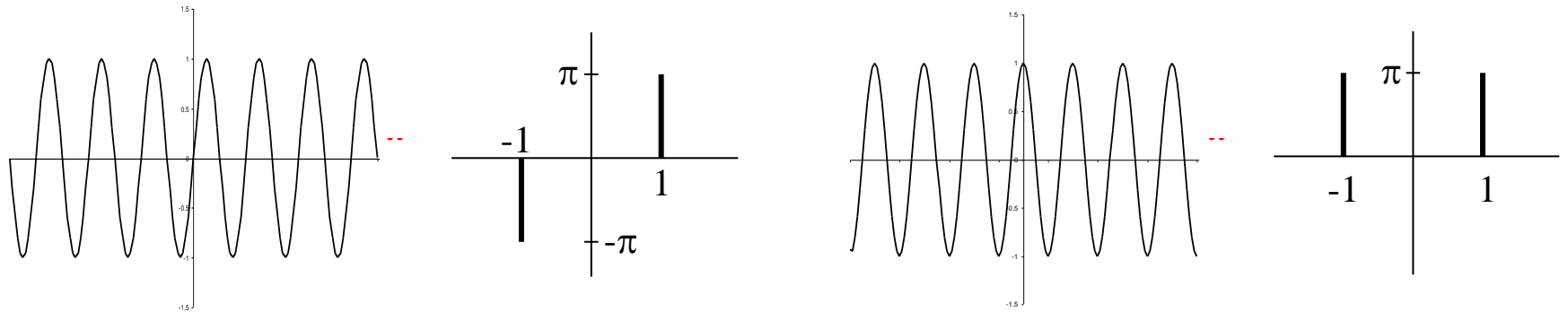


- A box filter transforms to...

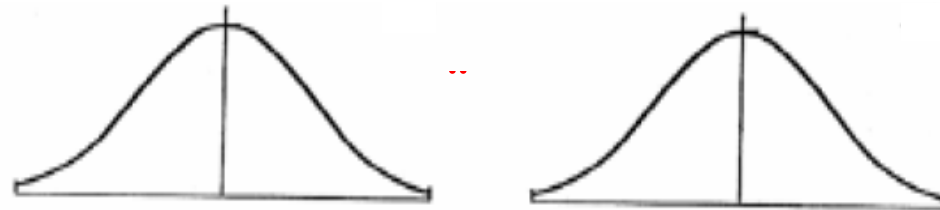


# Fourier Transforms of Important Functions

- Sine and cosine transform to “frequency spikes”

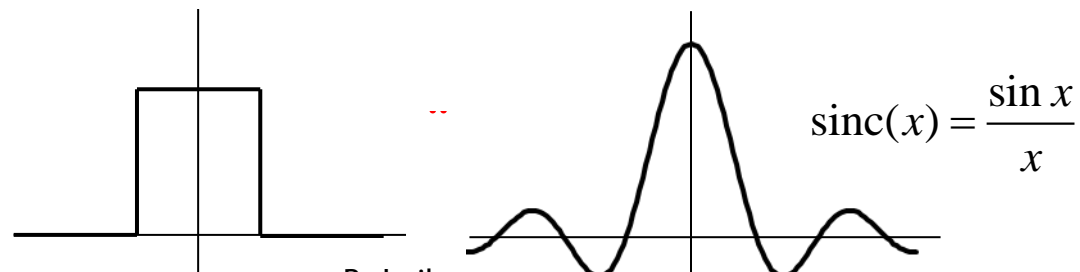


- A Gaussian transforms to a Gaussian



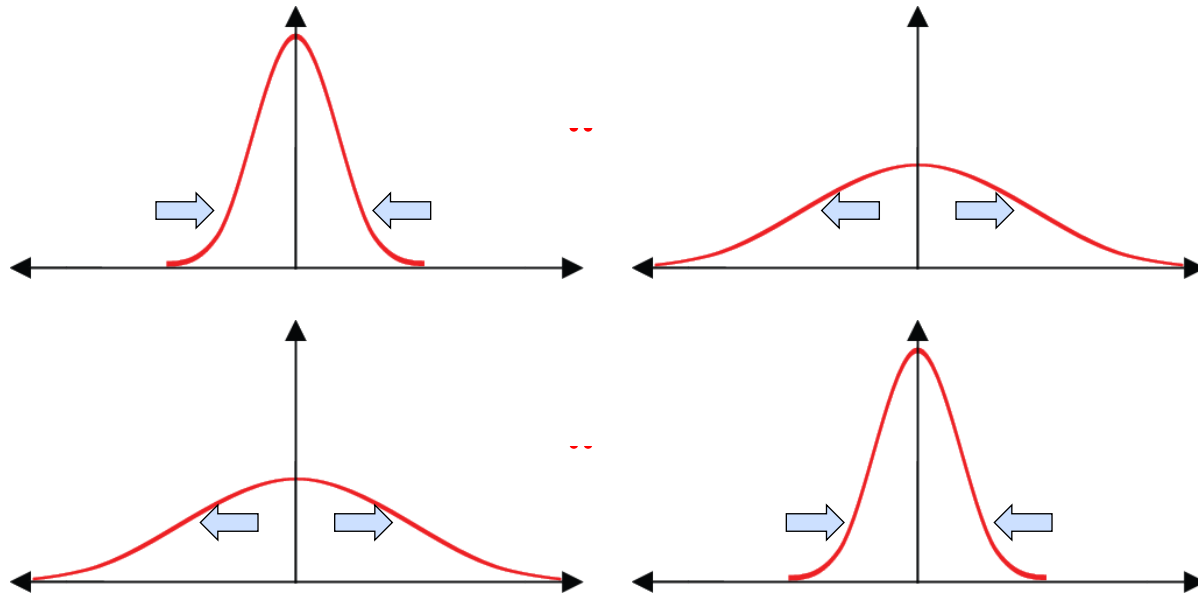
All of this is symmetric!

- A box filter transforms to a sinc

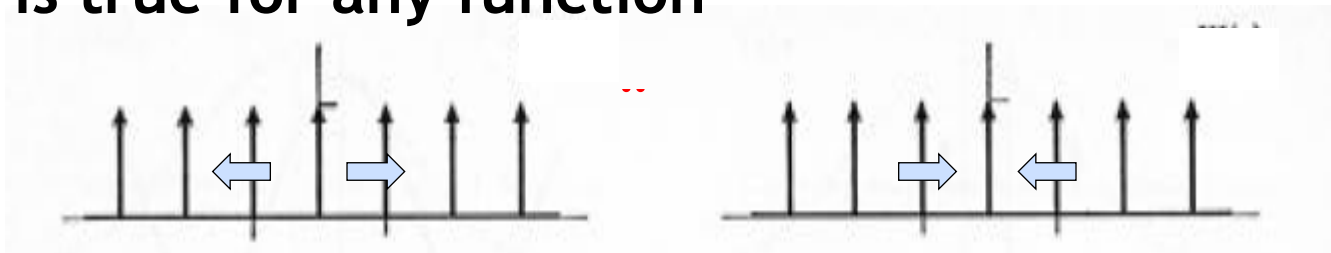


# Duality

- The better a function is localized in one domain, the worse it is localized in the other.



- This is true for any function



# Effect of Convolution

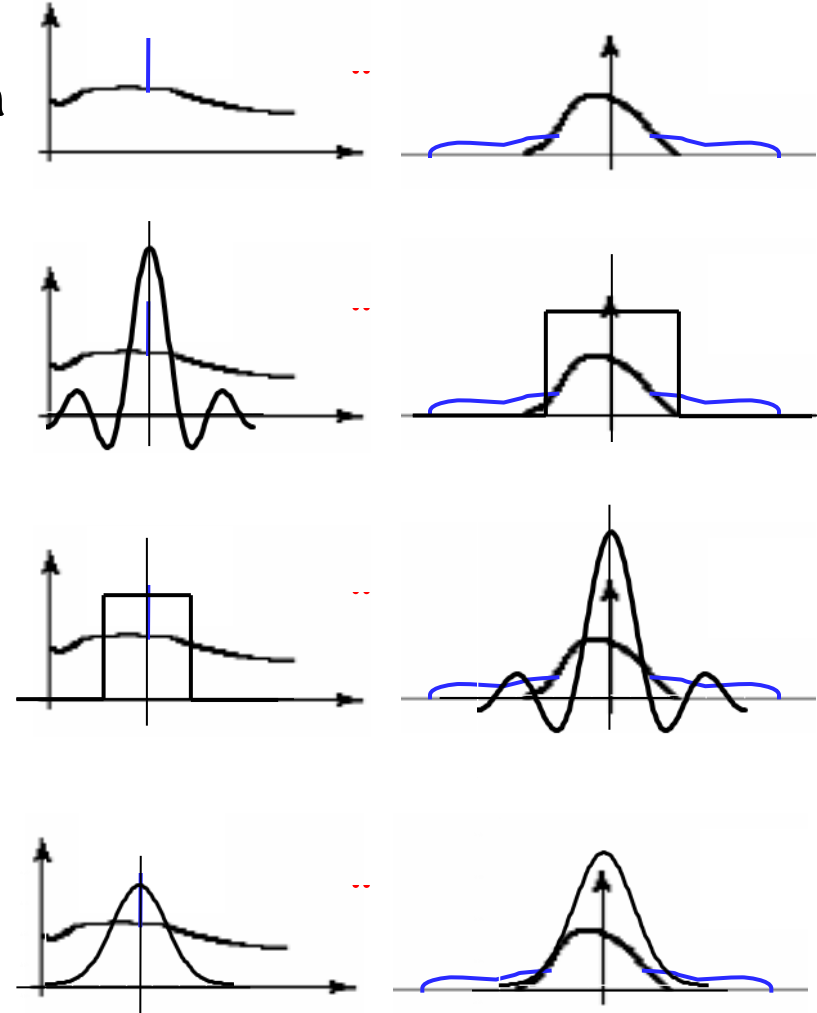
- Convoluting two functions in the image domain corresponds to taking the product of their transformed versions in the frequency domain.

$$f \star g \xrightarrow{\circ} \mathcal{F} \cdot \mathcal{G}$$

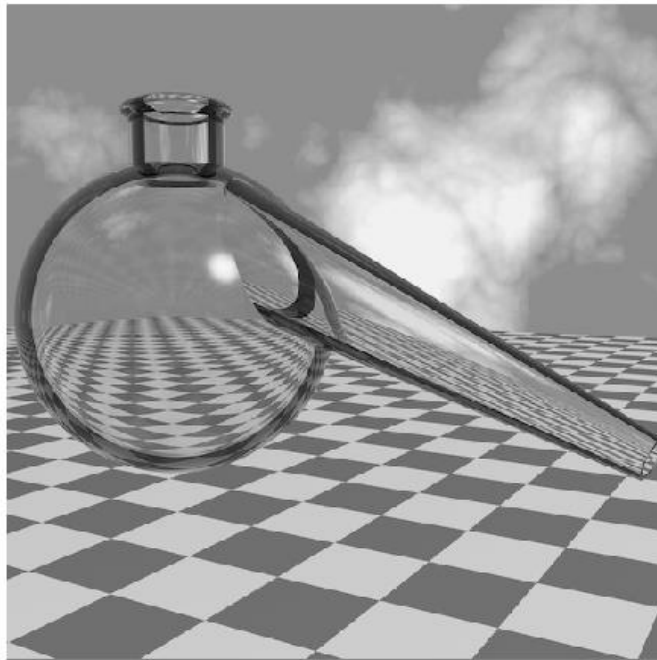
- This gives us a tool to manipulate image spectra.
  - A filter attenuates or enhances certain frequencies through this effect.

# Effect of Filtering

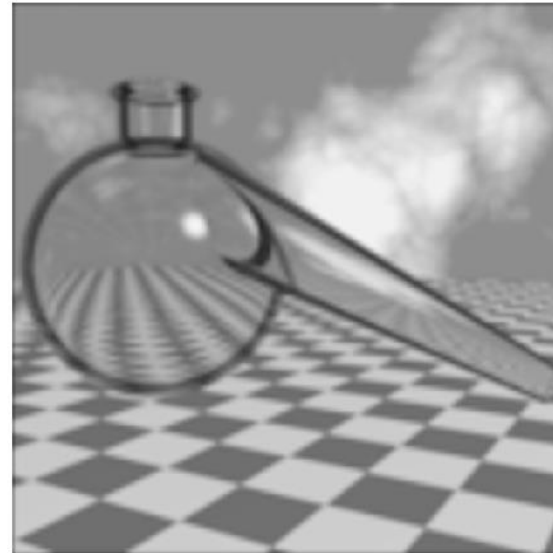
- Noise introduces high frequencies. To remove them, we want to apply a “low-pass” filter.
- The ideal filter shape in the frequency domain would be a box. But this transfers to a spatial sinc, which has infinite spatial support.
- A compact spatial box filter transfers to a frequency sinc, which creates artifacts.
- A Gaussian has compact support in both domains. This makes it a convenient choice for a low-pass filter.



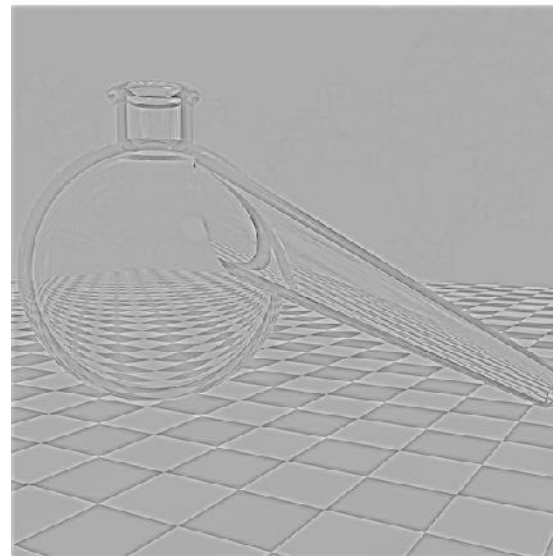
# Low-Pass vs. High-Pass



Original image



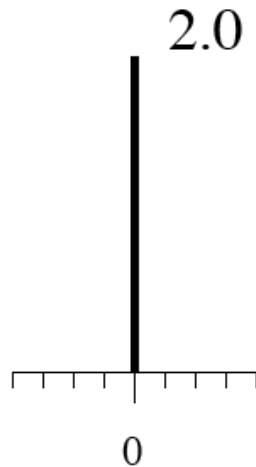
Low-pass  
filtered



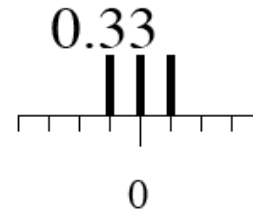
High-pass  
filtered



# Quiz: What Effect Does This Filter Have?



—

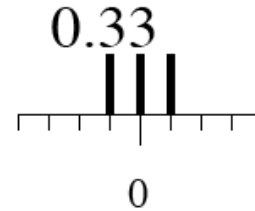
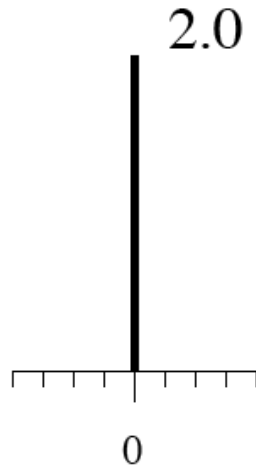


?

# Sharpening Filter

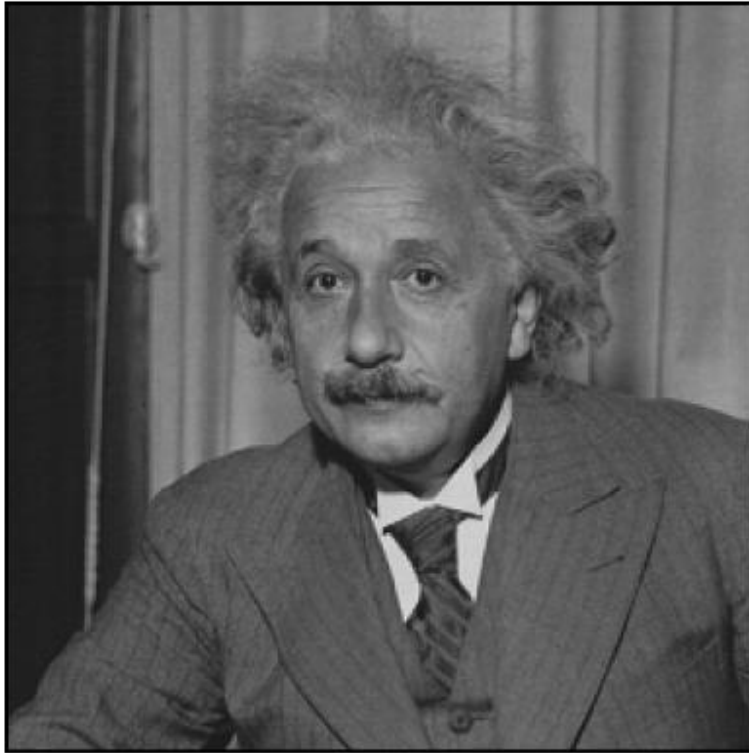


Original

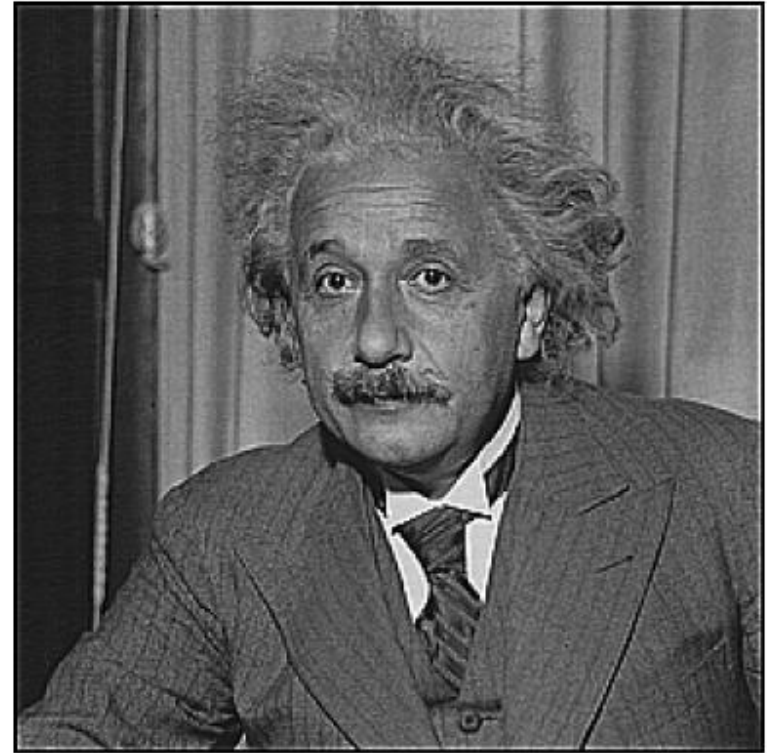


**Sharpening filter**  
– Accentuates differences  
with local average

# Sharpening Filter



**before**



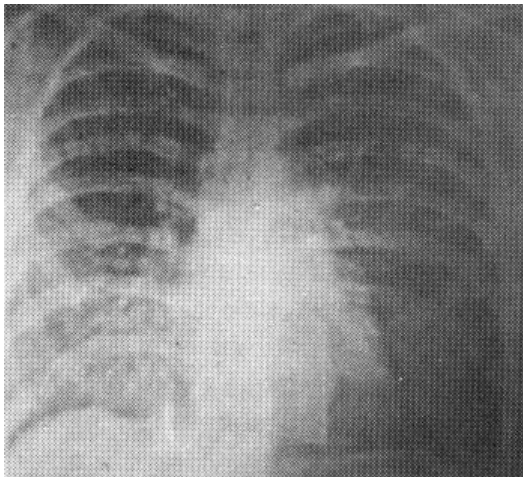
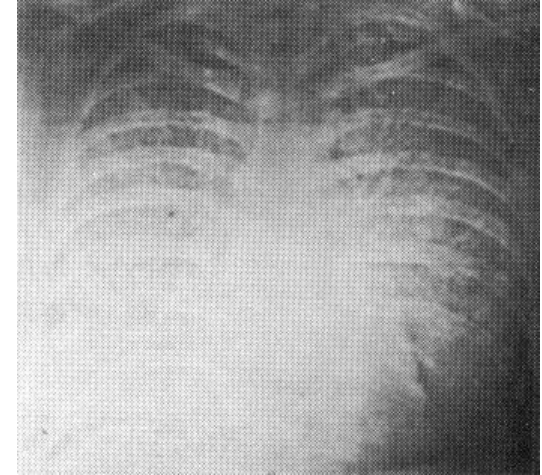
**after**

# Application: High Frequency Emphasis

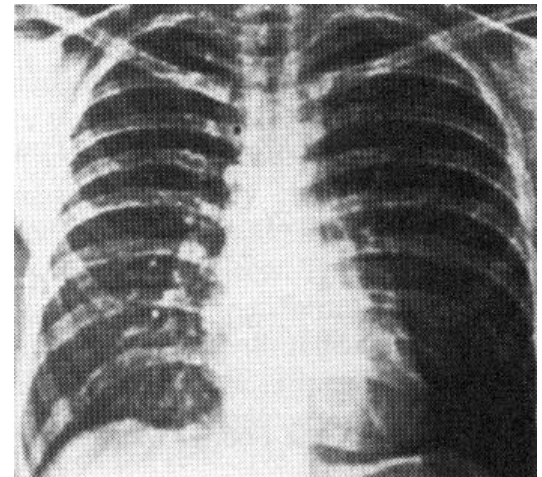
Original



High pass Filter



High Frequency  
Emphasis



High Frequency Emphasis  
+  
Histogram Equalization

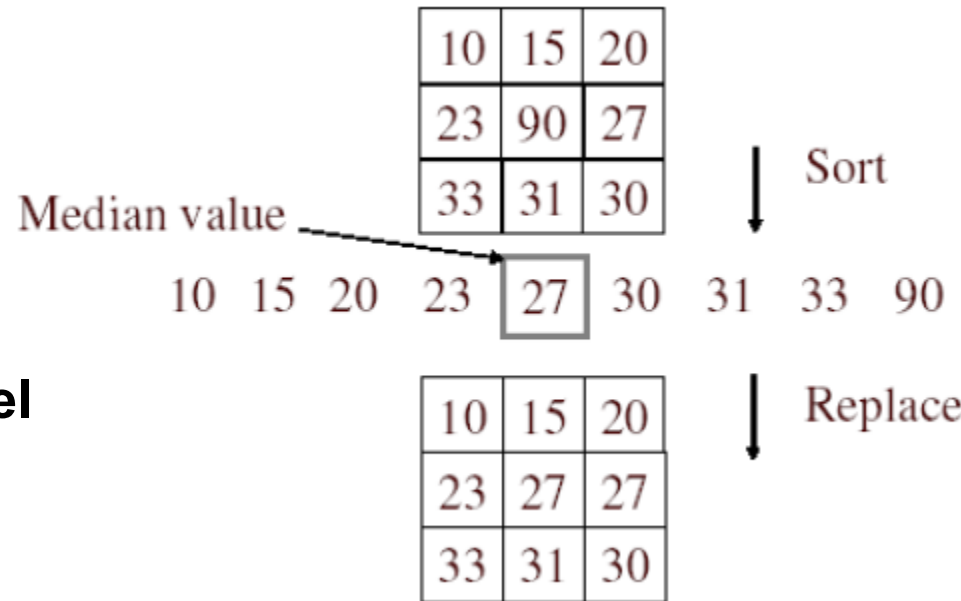
# Topics of This Lecture

- Linear filters
  - What are they? How are they applied?
  - Application: smoothing
  - Gaussian filter
  - What does it *mean* to filter an image?
- **Nonlinear Filters**
  - **Median filter**
- Multi-Scale representations
  - How to properly rescale an image?
- Image derivatives
  - How to compute gradients robustly?

# Non-Linear Filters: Median Filter

- **Basic idea**

- Replace each pixel by the median of its neighbors.



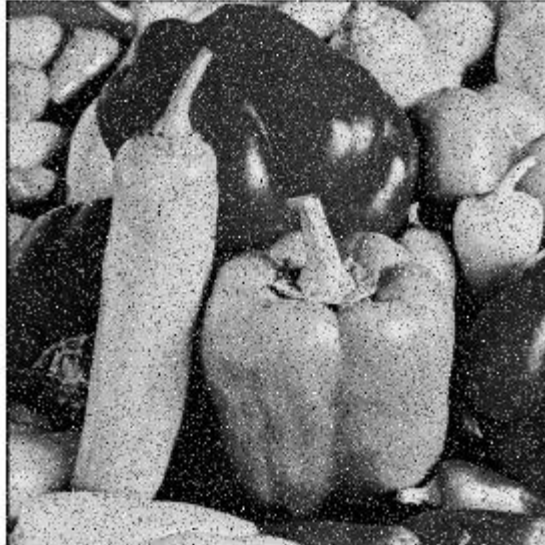
- **Properties**

- Doesn't introduce new pixel values
- Removes spikes: good for impulse, salt & pepper noise
- Linear?

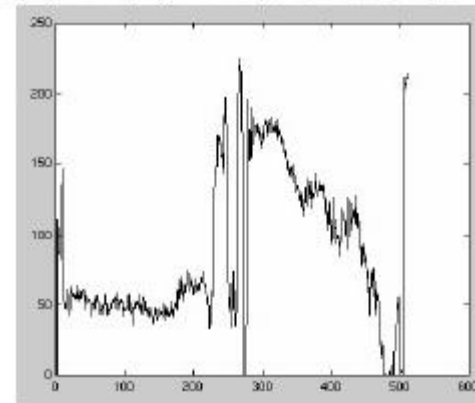
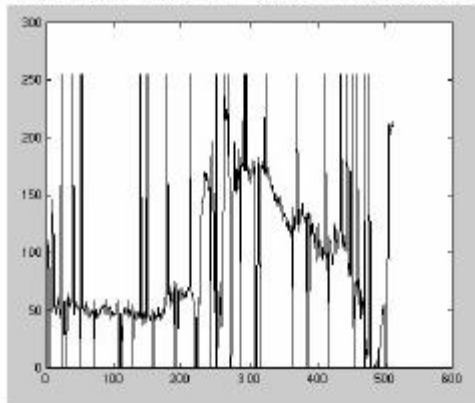


# Median Filter

Salt and pepper noise



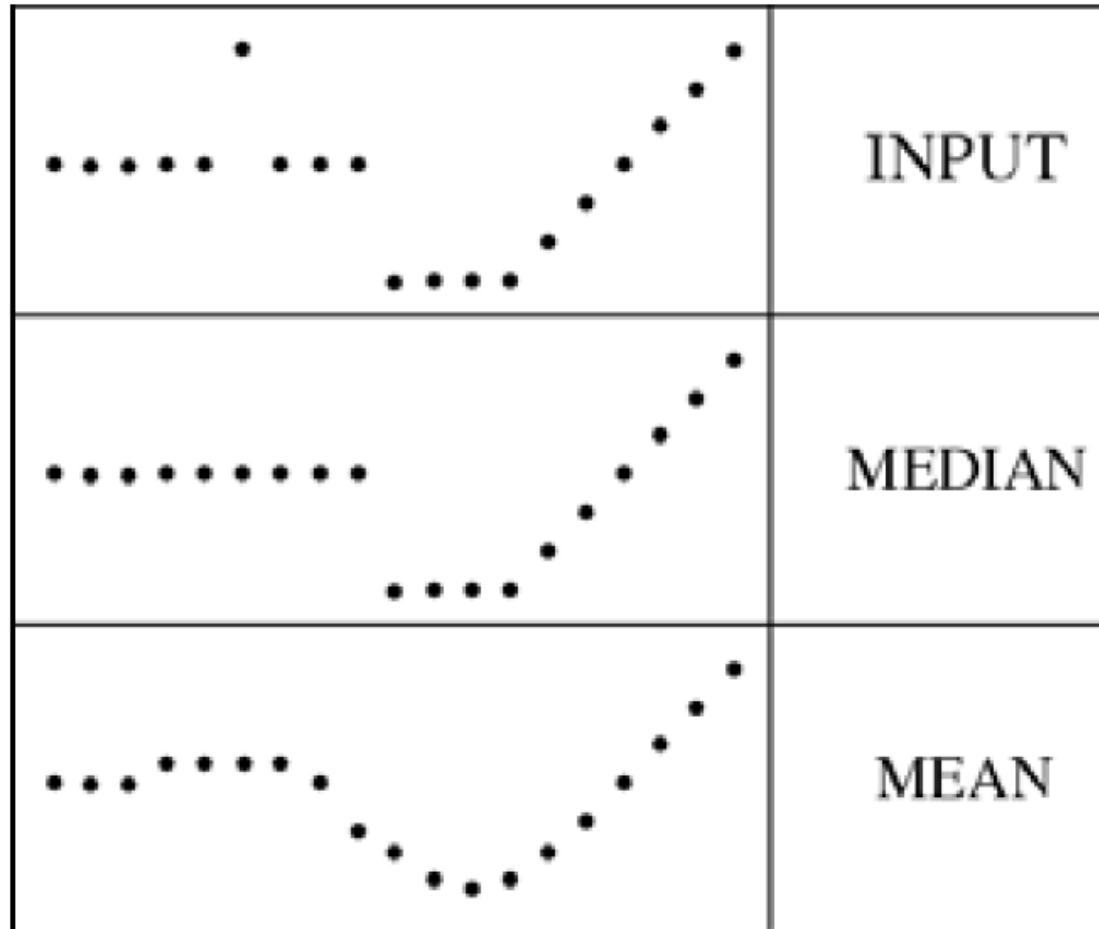
Median filtered



Plots of a row of the image

# Median Filter

- The Median filter is **edge preserving**.





# Median vs. Gaussian Filtering

3x3

5x5

7x7

**Gaussian**



**Median**

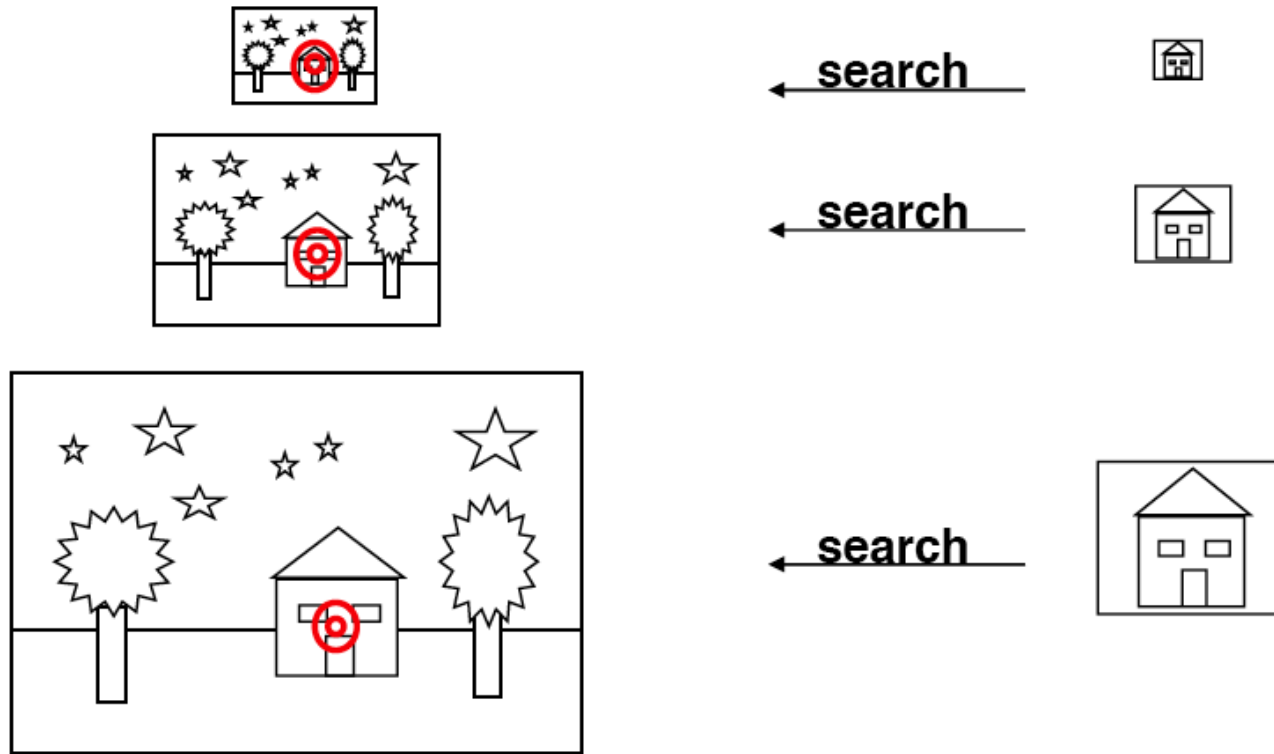


# Topics of This Lecture

- Linear filters
  - What are they? How are they applied?
  - Application: smoothing
  - Gaussian filter
  - What does it *mean* to filter an image?
- Nonlinear Filters
  - Median filter
- **Multi-Scale representations**
  - How to properly rescale an image?
- Filters as templates
  - Correlation as template matching

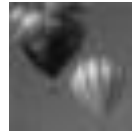


# Motivation: Fast Search Across Scales



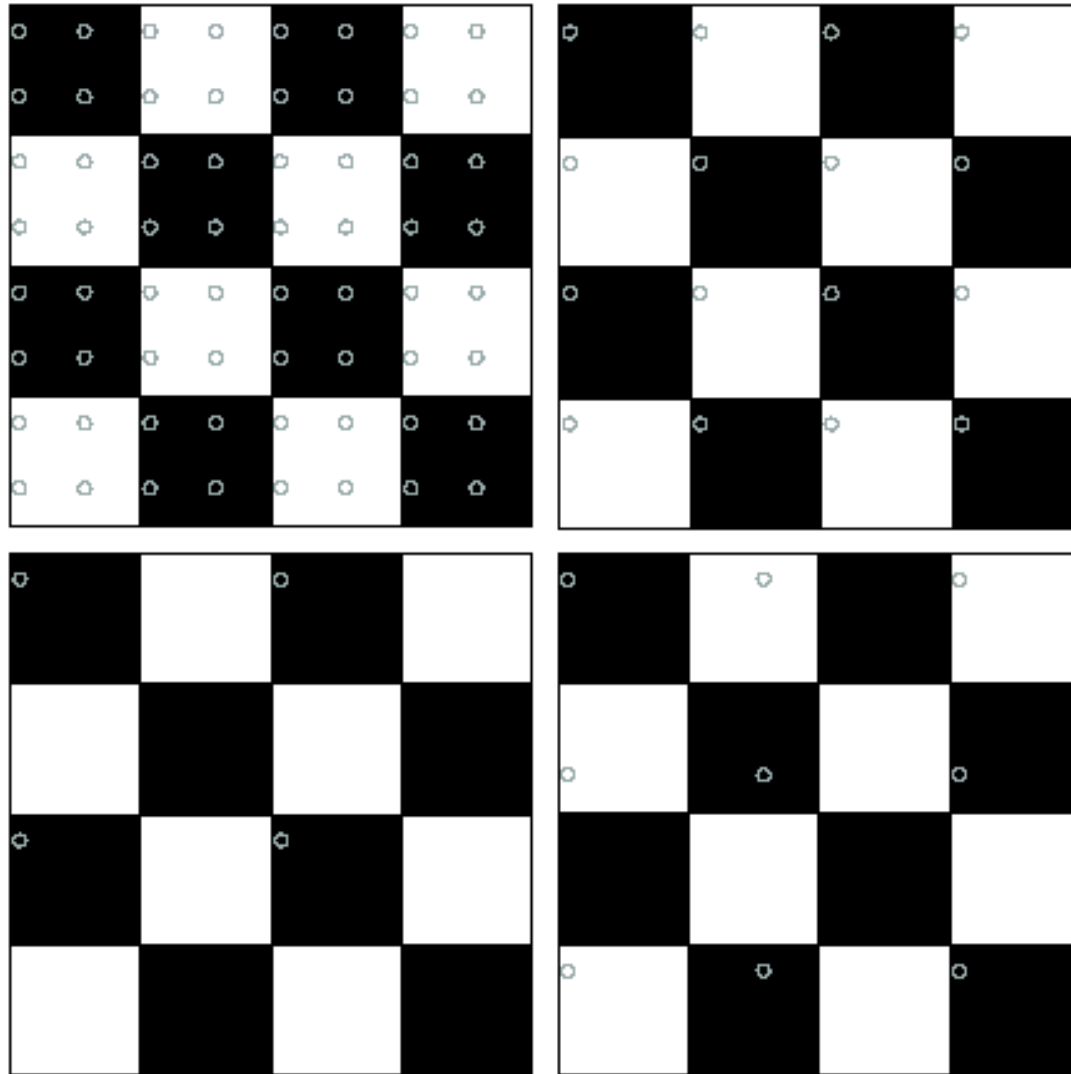
# Image Pyramid

Low resolution



High resolution

# How Should We Go About Resampling?



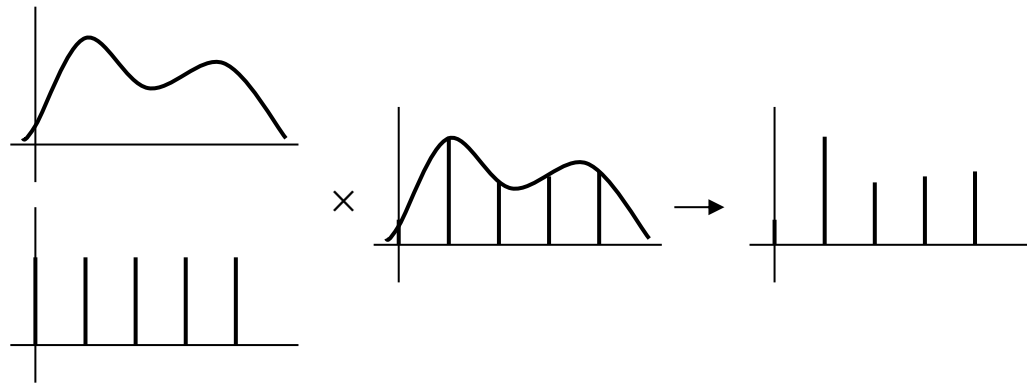
Let's resample the checkerboard by taking one sample at each circle.

In the top left board, the new representation is reasonable. Top right also yields a reasonable representation.

Bottom left is all black (dubious) and bottom right has checks that are too big.

# Fourier Interpretation: Discrete Sampling

- Sampling in the spatial domain is like multiplying with a spike function.

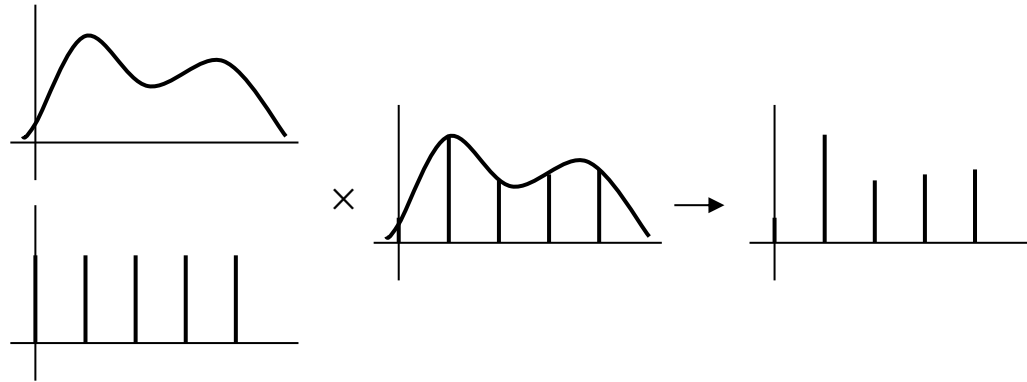


- Sampling in the frequency domain is like...

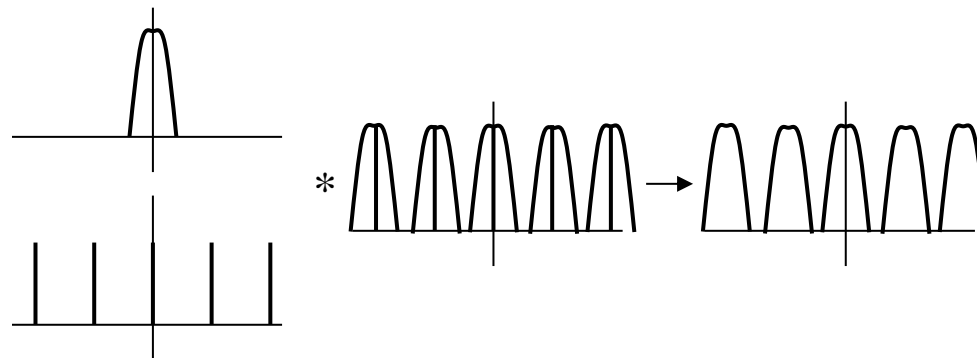
?

# Fourier Interpretation: Discrete Sampling

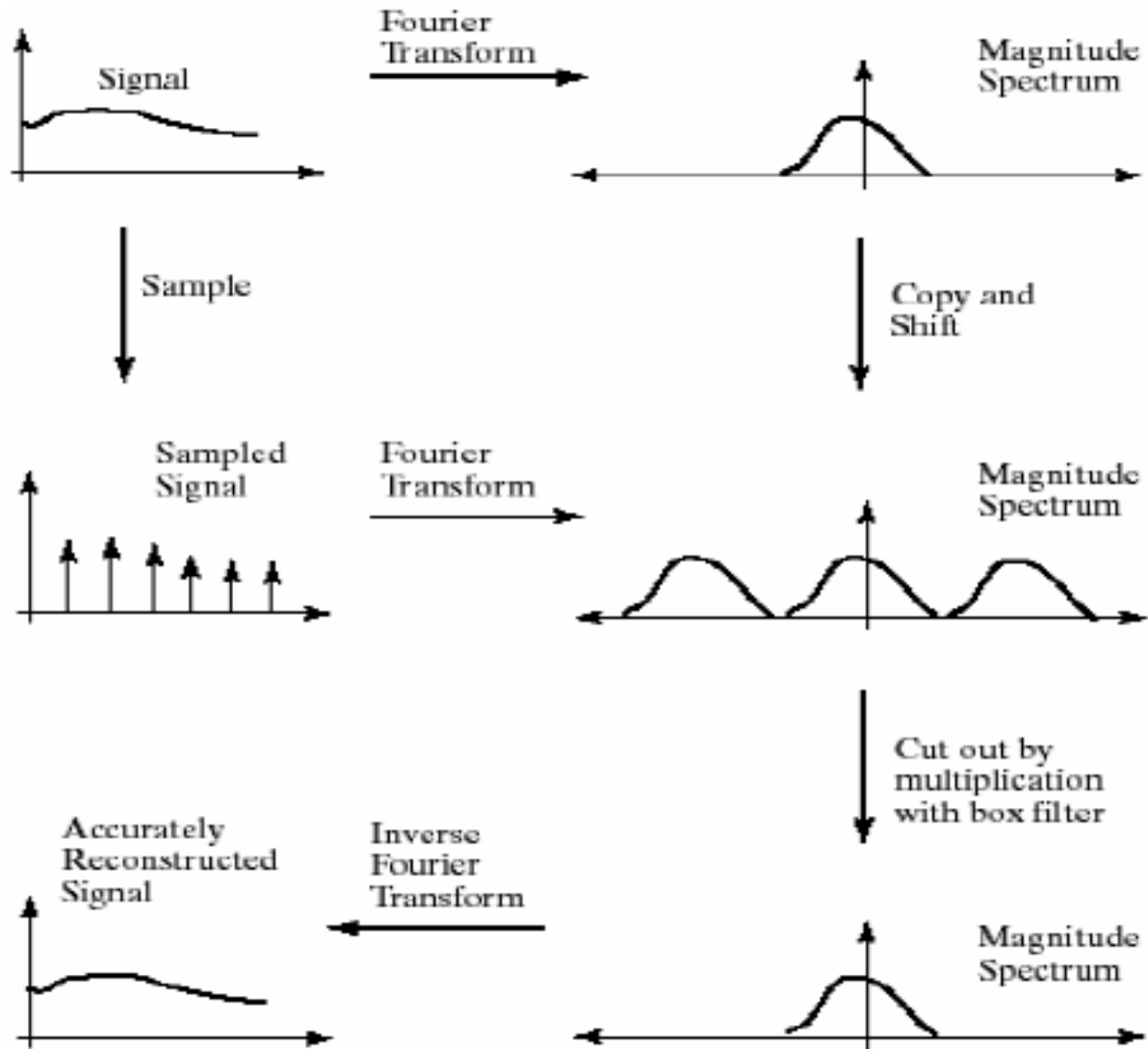
- Sampling in the spatial domain is like multiplying with a spike function.



- Sampling in the frequency domain is like convolving with a spike function.

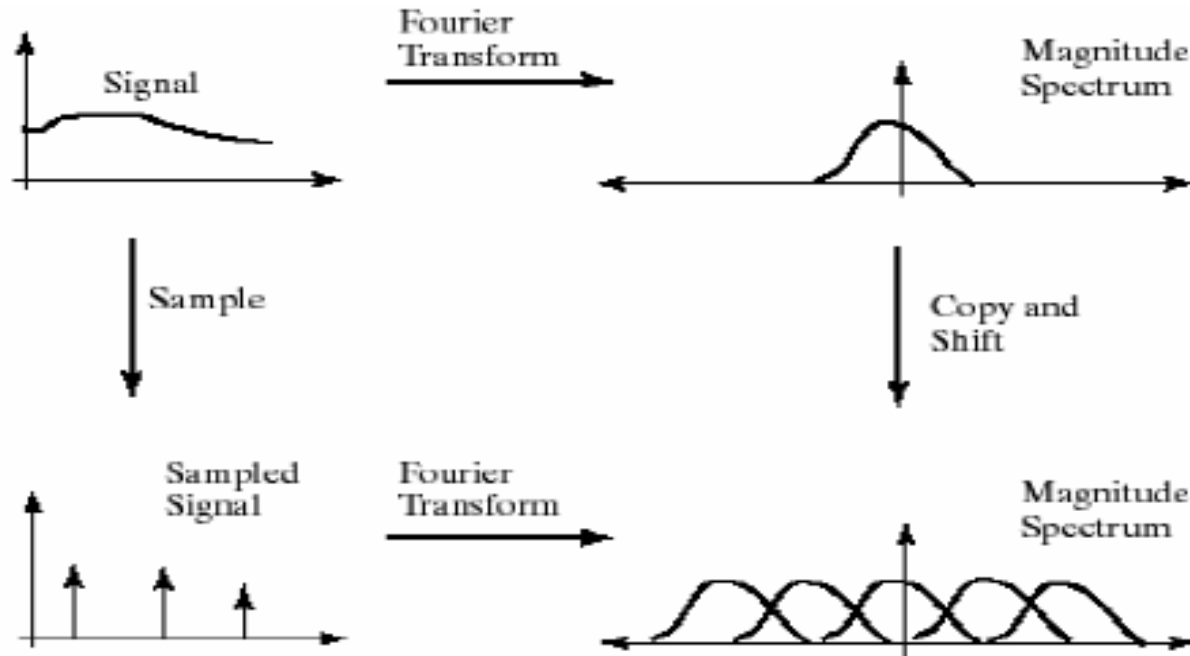


# Sampling and Aliasing





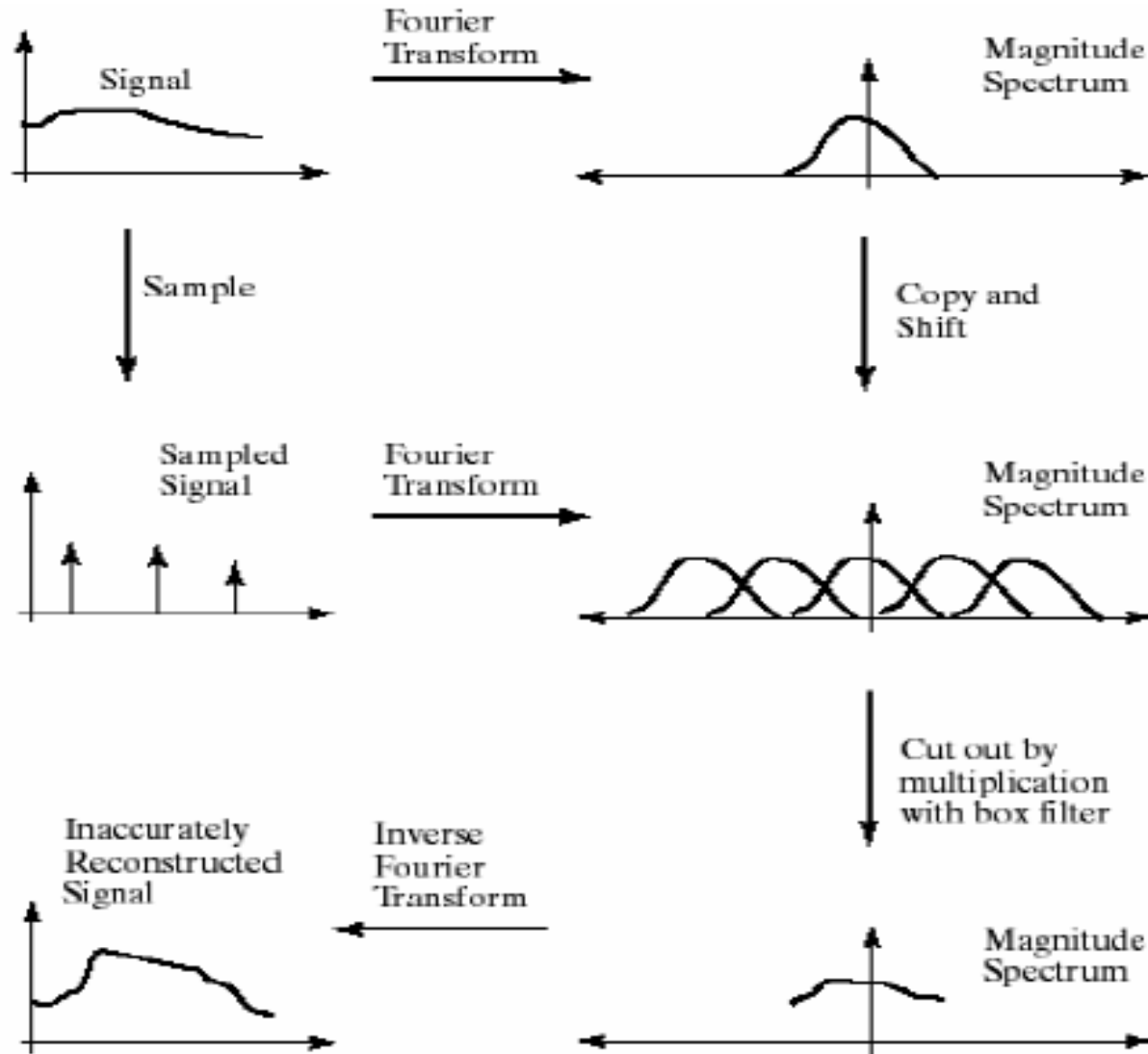
# Sampling and Aliasing



- Nyquist theorem:

- In order to recover a certain frequency  $f$ , we need to sample with at least  $2f$ .
- This corresponds to the point at which the transformed frequency spectra start to overlap (the **Nyquist limit**)

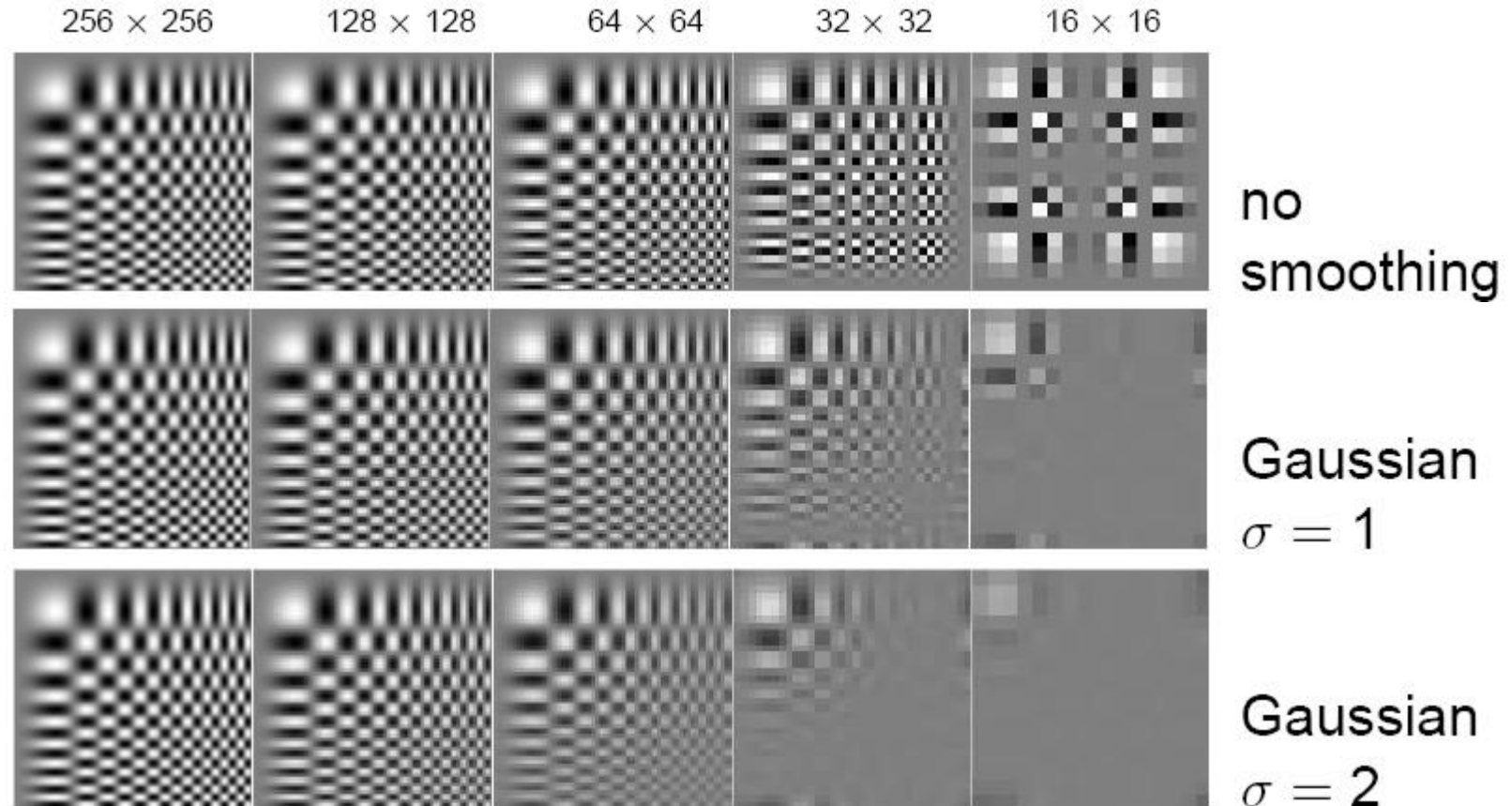
# Sampling and Aliasing



# Aliasing in Graphics



# Resampling with Prior Smoothing



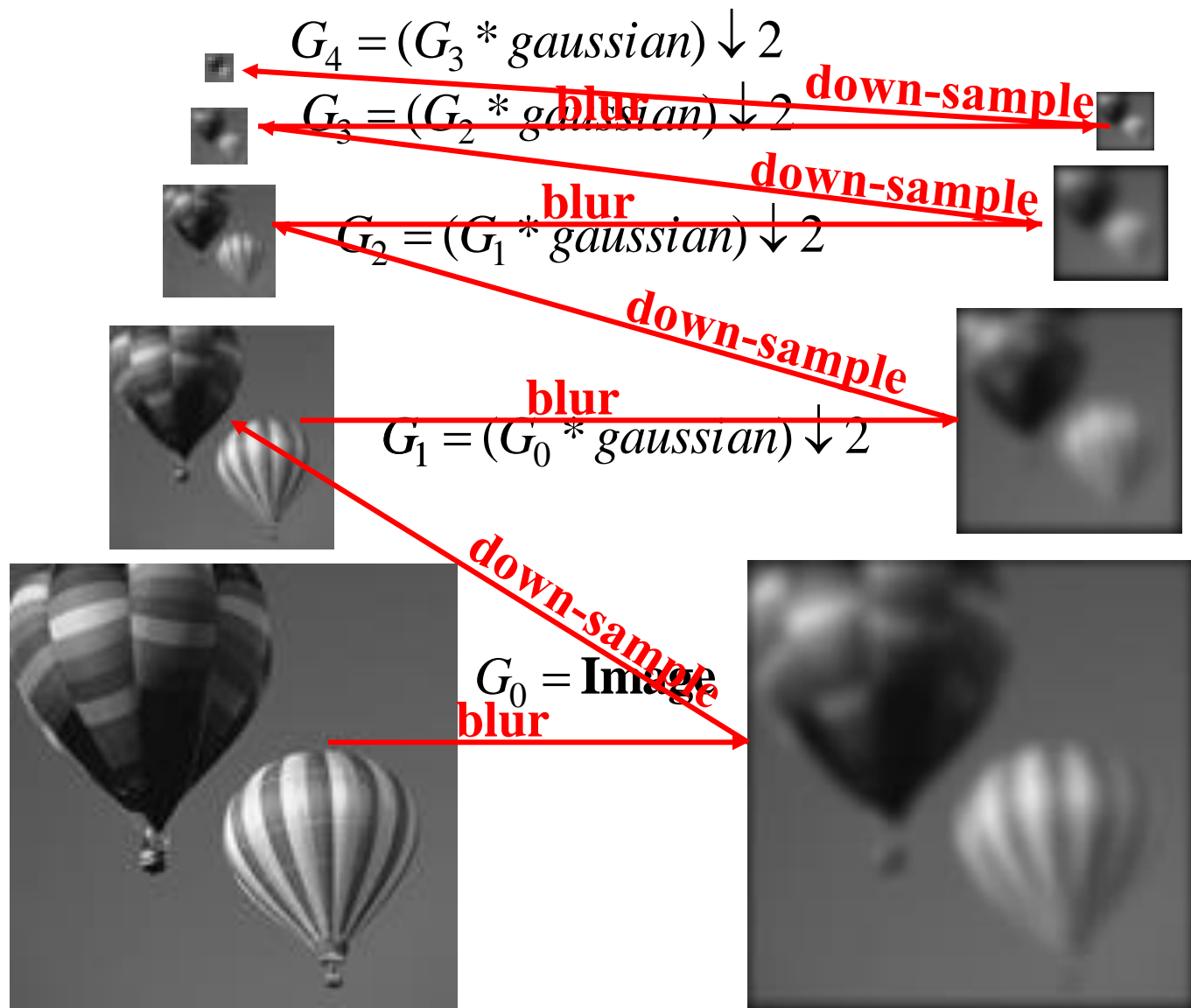
- **Note: We cannot recover the high frequencies, but we can avoid artifacts by smoothing before resampling.**

# The Gaussian Pyramid

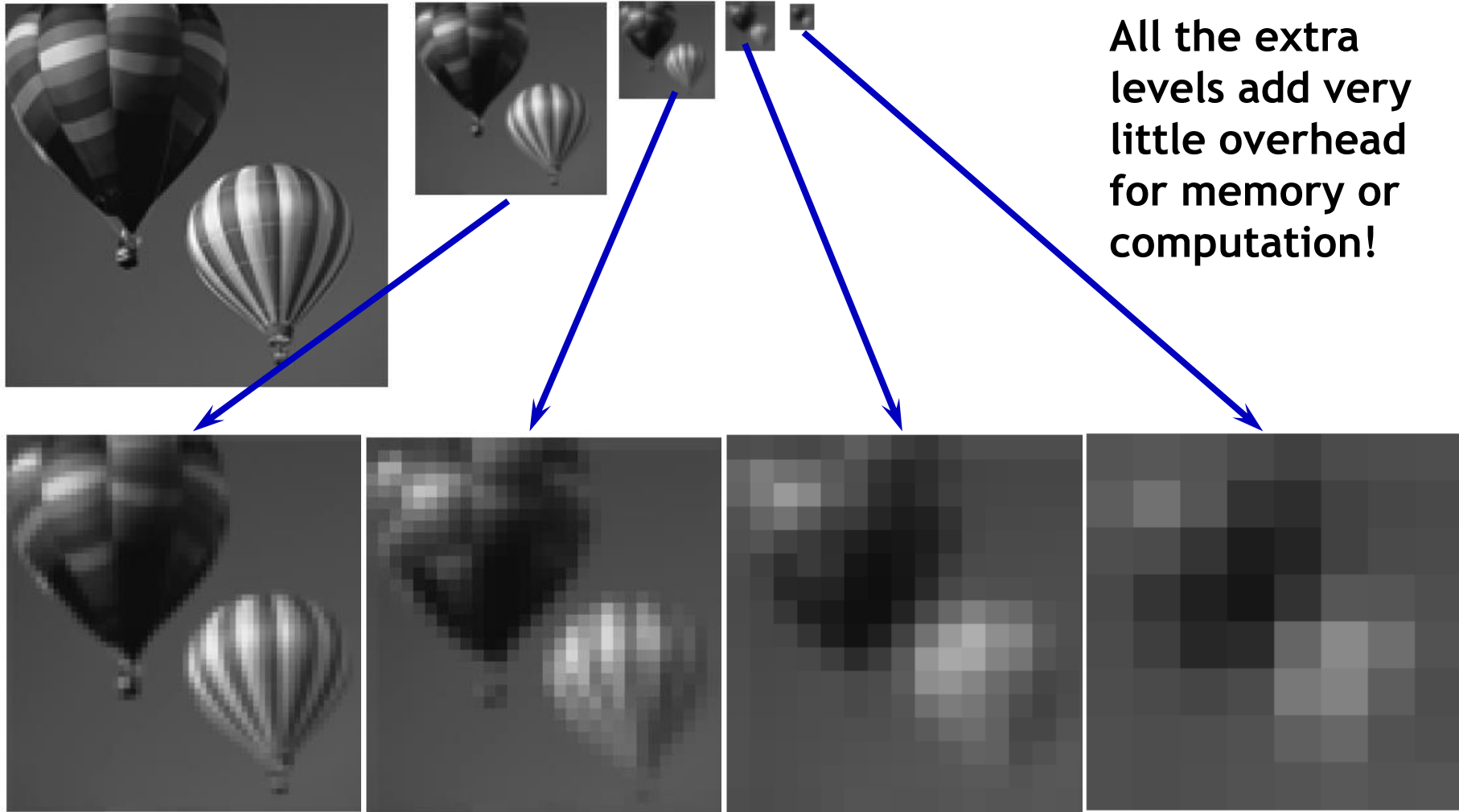
Low resolution



High resolution



# Gaussian Pyramid - Stored Information



# Summary: Gaussian Pyramid

- **Construction: create each level from previous one**
  - Smooth and sample
- **Smooth with Gaussians, in part because**
  - a Gaussian\*Gaussian = another Gaussian
  - $G(\sigma_1) * G(\sigma_2) = G(\sqrt{\sigma_1^2 + \sigma_2^2})$
- **Gaussians are low-pass filters, so the representation is redundant once smoothing has been performed.**
  - ⇒ There is no need to store smoothed images at the full original resolution.

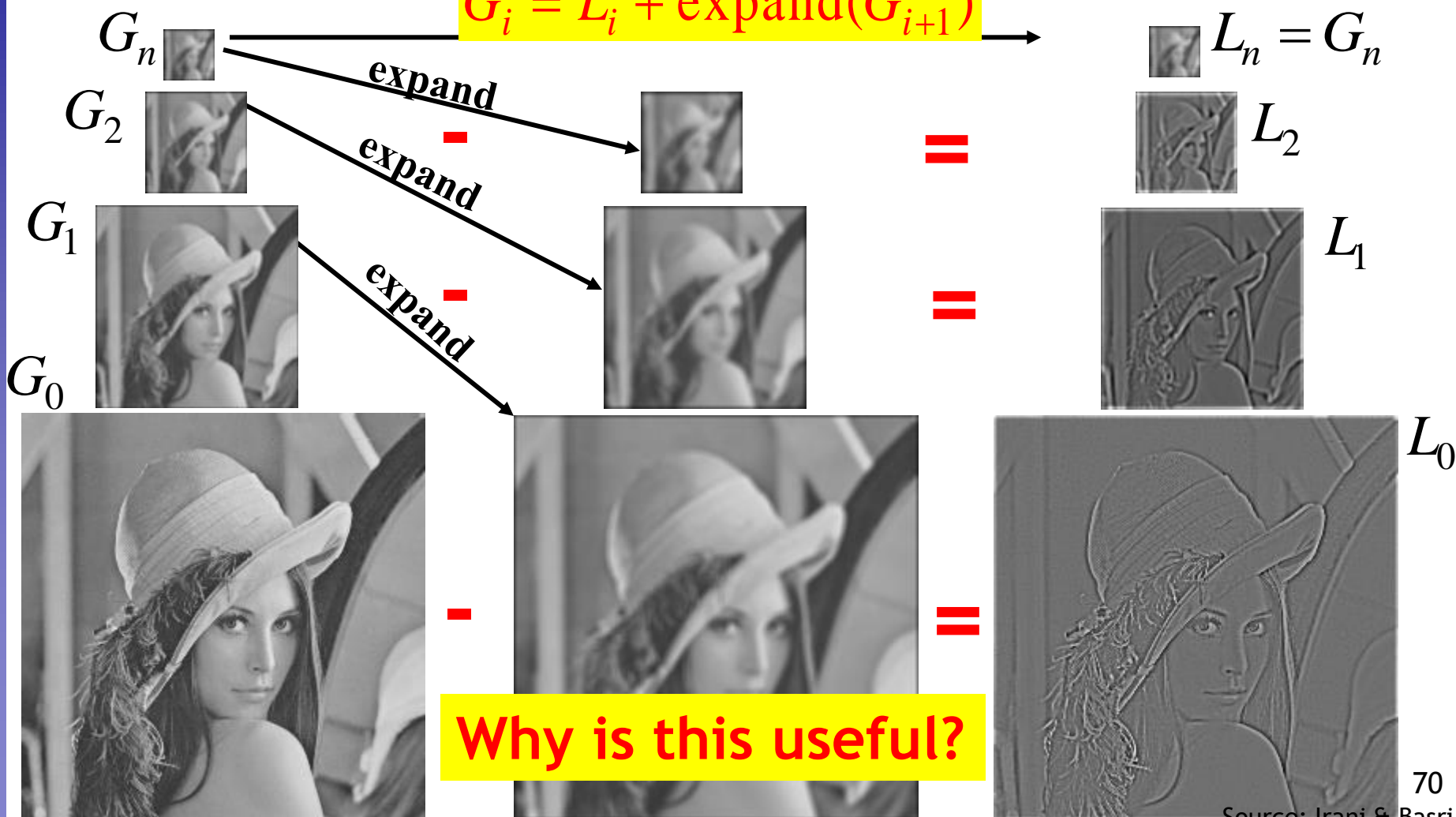
# The Laplacian Pyramid

Gaussian Pyramid

$$L_i = G_i - \text{expand}(G_{i+1})$$

$$G_i = L_i + \text{expand}(G_{i+1})$$

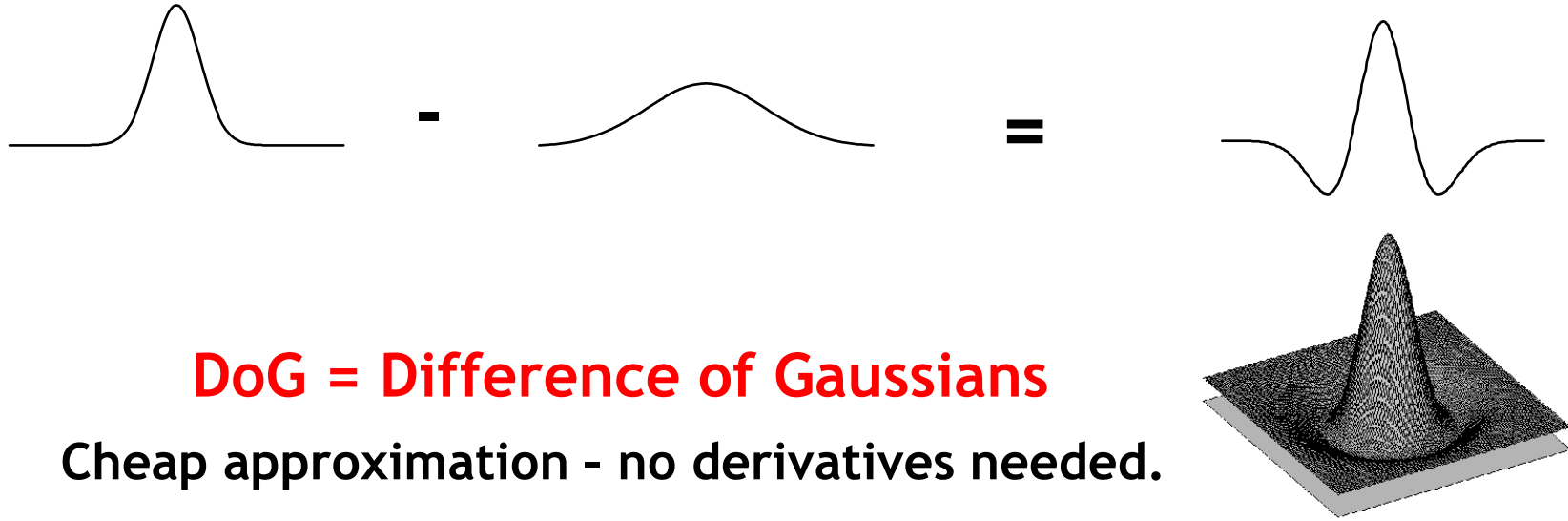
Laplacian Pyramid



Why is this useful?



# Laplacian ~ Difference of Gaussian



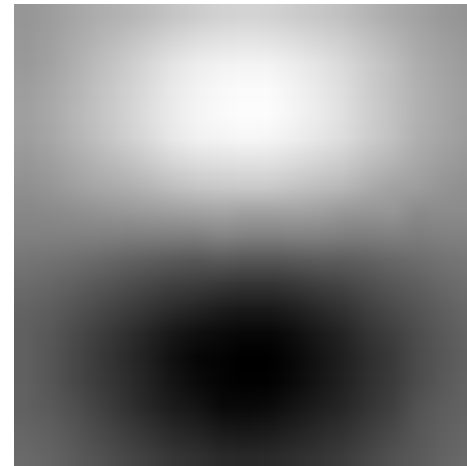
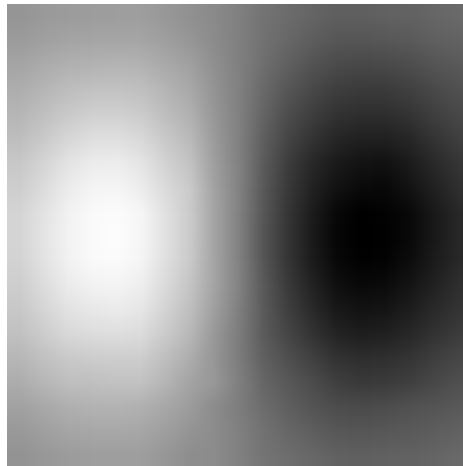
# Topics of This Lecture

- Linear filters
  - What are they? How are they applied?
  - Application: smoothing
  - Gaussian filter
  - What does it *mean* to filter an image?
- Nonlinear Filters
  - Median filter
- Multi-Scale representations
  - How to properly rescale an image?
- **Filters as templates**
  - **Correlation as template matching**



# Note: Filters are Templates

- Applying a filter at some point can be seen as taking a dot-product between the image and some vector.
- Filtering the image is a set of dot products.
- Insight
  - Filters look like the effects they are intended to find.
  - Filters find effects they look like.



# Where's Waldo?



Scene



Template



# Where's Waldo?



Template

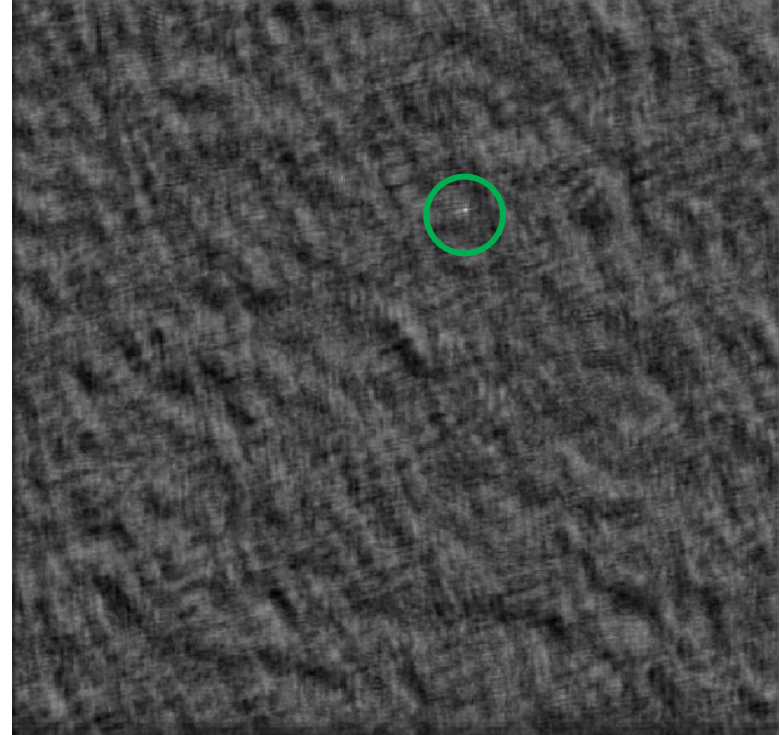
Detected template



# Where's Waldo?



**Detected template**



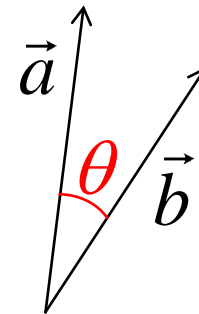
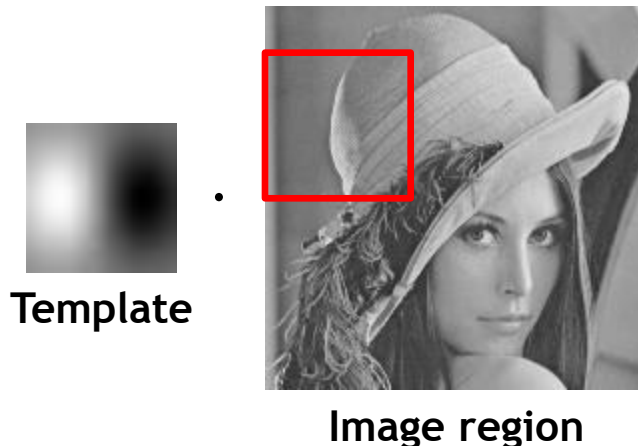
**Correlation map**

# Correlation as Template Matching

- Think of filters as a dot product of the filter vector with the image region
  - Now measure the angle between the vectors

$$a \cdot b = |a| |b| \cos \theta \quad \cos \theta = \frac{a \cdot b}{|a| |b|}$$

- Angle (similarity) between vectors can be measured by normalizing length of each vector to 1.



# Summary: Mask Properties

- **Smoothing**
  - Values positive
  - Sum to 1  $\Rightarrow$  constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter
- **Filters act as templates**
  - Highest response for regions that “look the most like the filter”
  - Dot product as correlation



# Summary Linear Filters

- **Linear filtering:**
  - Form a new image whose pixels are a weighted sum of original pixel values
- **Properties**
  - Output is a shift-invariant function of the input (same at each image location)

## Examples:

- Smoothing with a box filter
- Smoothing with a Gaussian
- Finding a derivative
- Searching for a template

## Pyramid representations

- Important for describing and searching an image at all scales

# References and Further Reading

- Background information on linear filters and their connection with the Fourier transform can be found in Chapters 7 and 8 of
  - D. Forsyth, J. Ponce, *Computer Vision - A Modern Approach*. Prentice Hall, 2003

