Computer Vision

CS-E4850, 5 study credits

Lecturer: Juho Kannala

Lecture 2: Image Processing

- Lecture concentrates on image filtering
- Relevant reading: Chapter 3 of Szeliski's book

Acknowledgement: many slides from James Hays, Derek Hoiem, Svetlana Lazebnik, Steve Seitz, David Lowe, Kristen Grauman, Alexei Efros and others (detailed credits on individual slides)

Three views of filtering

- Image filters in spatial domain
  - Filter is a mathematical operation of a grid of numbers
  - Smoothing, sharpening, edge detection

- Image filters in the frequency domain
  - Filtering is a way to modify the frequencies of images
  - Hybrid images, sampling, image resizing

- Templates and image pyramids
  - Filtering is a way to match a template to the image
  - Detection, coarse-to-fine registration

Source: J. Hays

Image filtering

- Image filtering: compute function of local neighborhood at each position

- Really important!
  - Enhance images
    - Denoise, resize, increase contrast, etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching
  - Deep Convolutional Networks
    - Sequence of linear filters and non-linear functions

Source: J. Hays

Motivation: Image denoising

- How can we reduce noise in a photograph?

Source: S. Lazebnik

Moving average

- Let's replace each pixel with a *weighted* average of its neighborhood
- The weights are called the *filter kernel*
- What are the weights for the average of a 3x3 neighborhood?

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

"box filter"

Source: D. Lowe

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10							

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20						

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30					

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
				?					

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	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	

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood

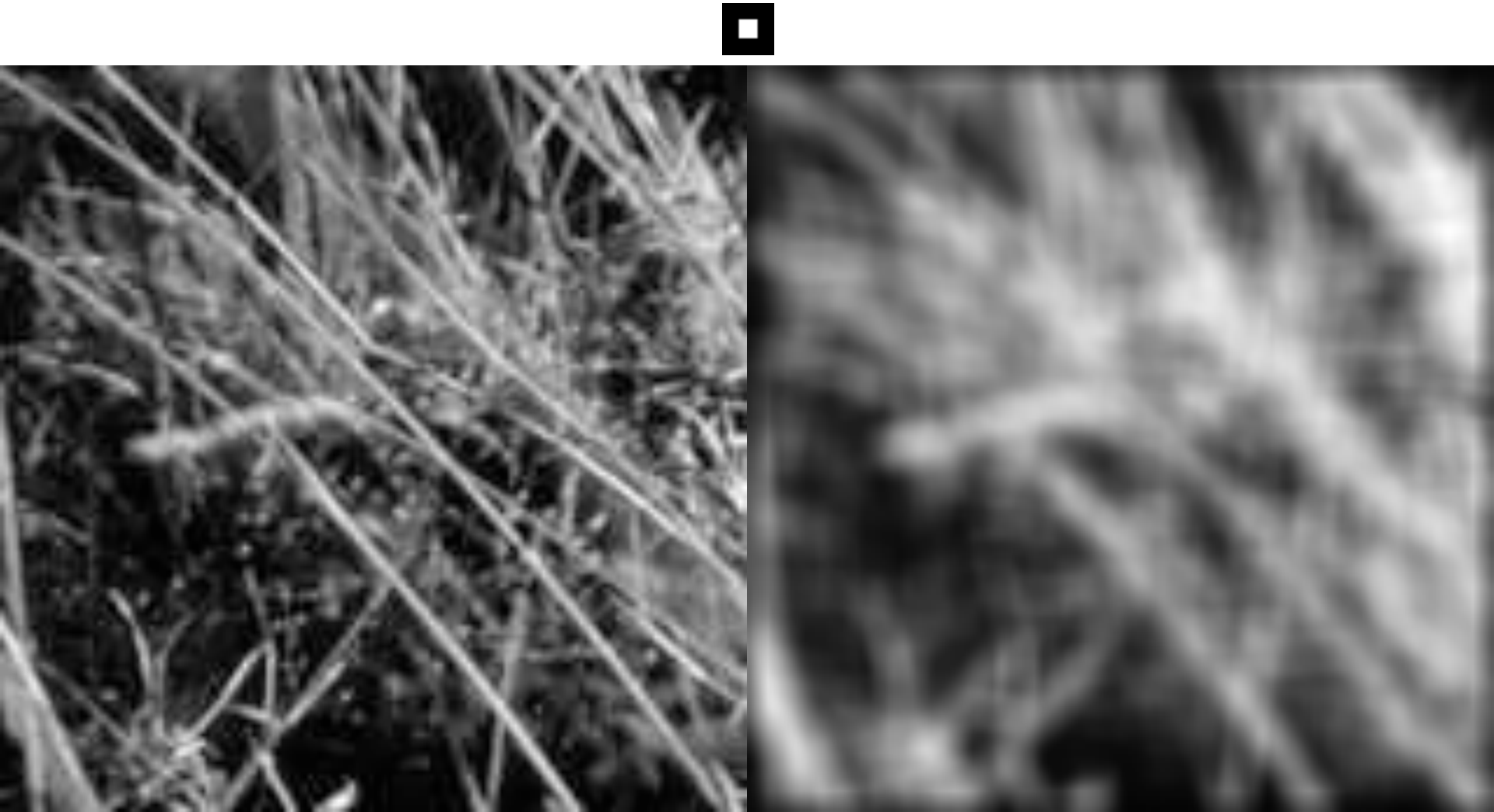
- Achieve smoothing effect (remove sharp features)

$$g[\cdot,\cdot]$$

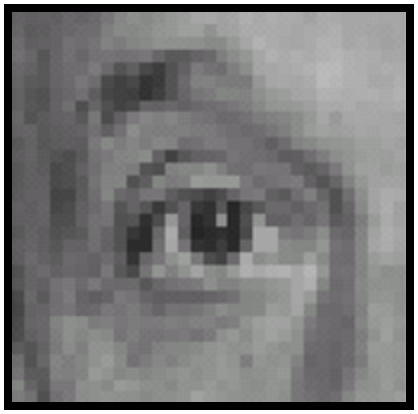
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slide credit: David Lowe

Smoothing with box filter



Practice with linear filters



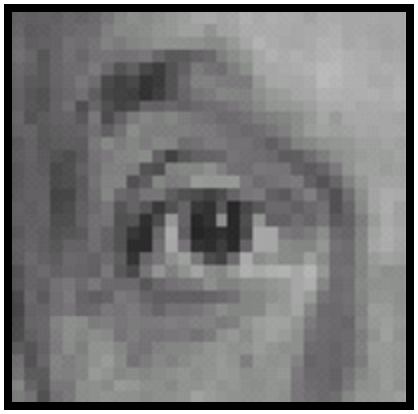
Original

0	0	0
0	1	0
0	0	0

?

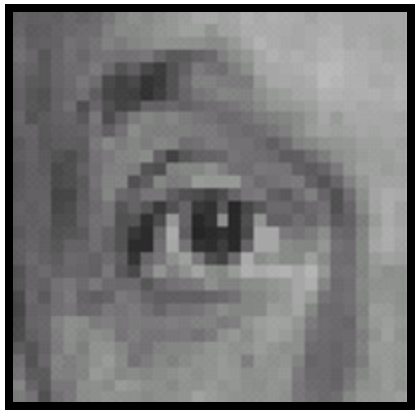
Source: D. Lowe

Practice with linear filters



Original

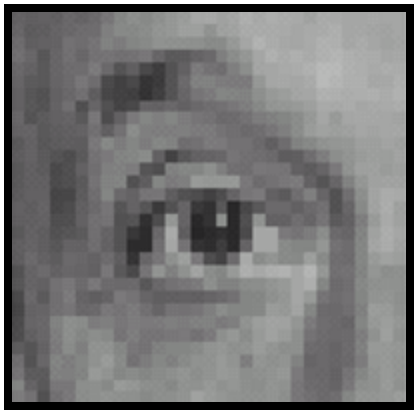
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Source: D. Lowe

Practice with linear filters



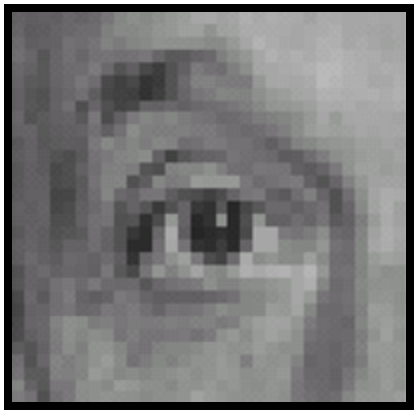
Original

0	0	0
0	0	1
0	0	0

?

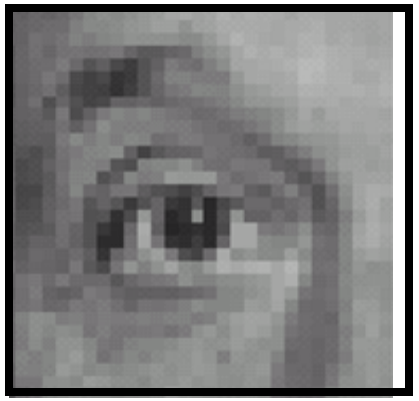
Source: D. Lowe

Practice with linear filters



Original

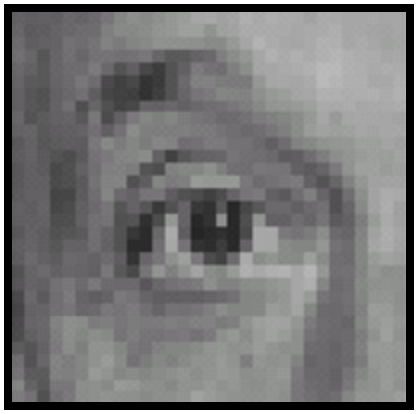
0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters



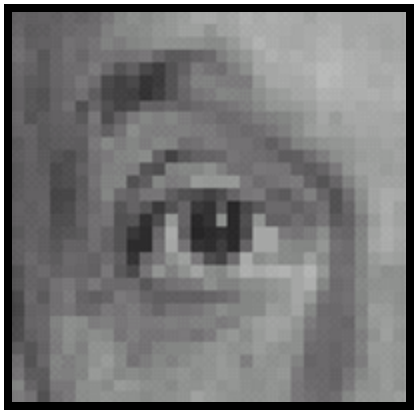
Original

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

(Note that filter sums to 1)

Source: D. Lowe

Practice with linear filters



Original

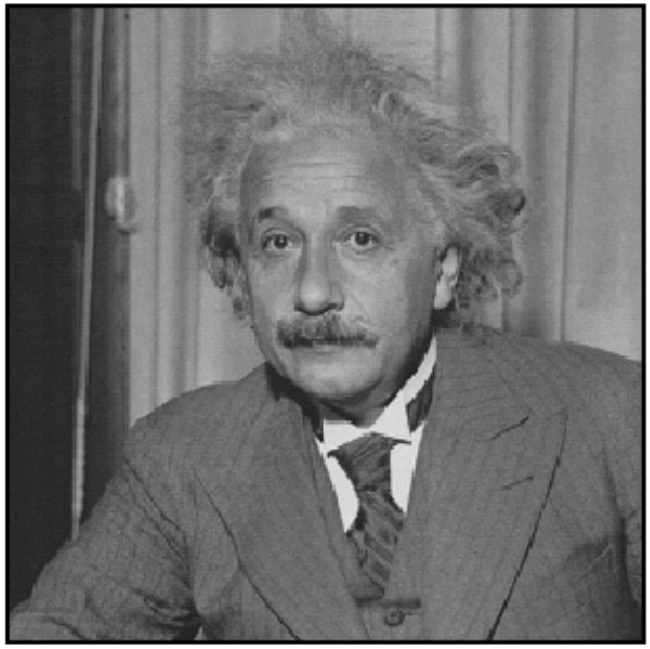
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

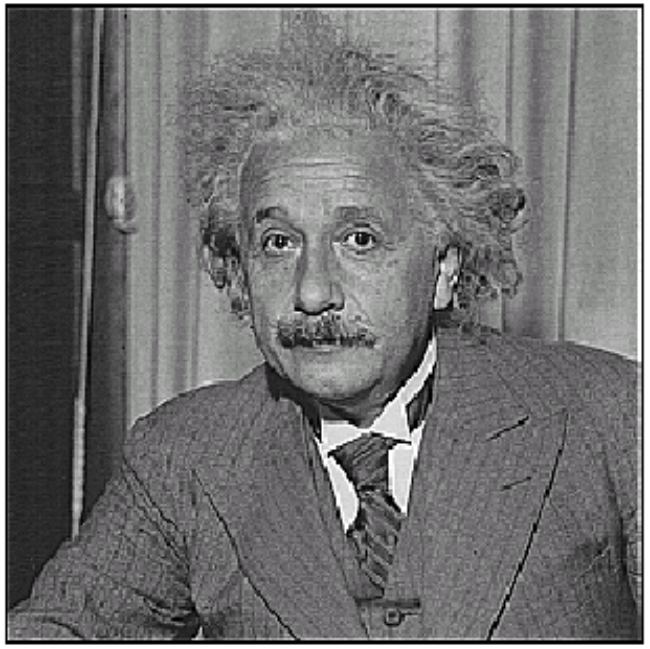
- Accentuates differences with local average

Source: D. Lowe

Sharpening



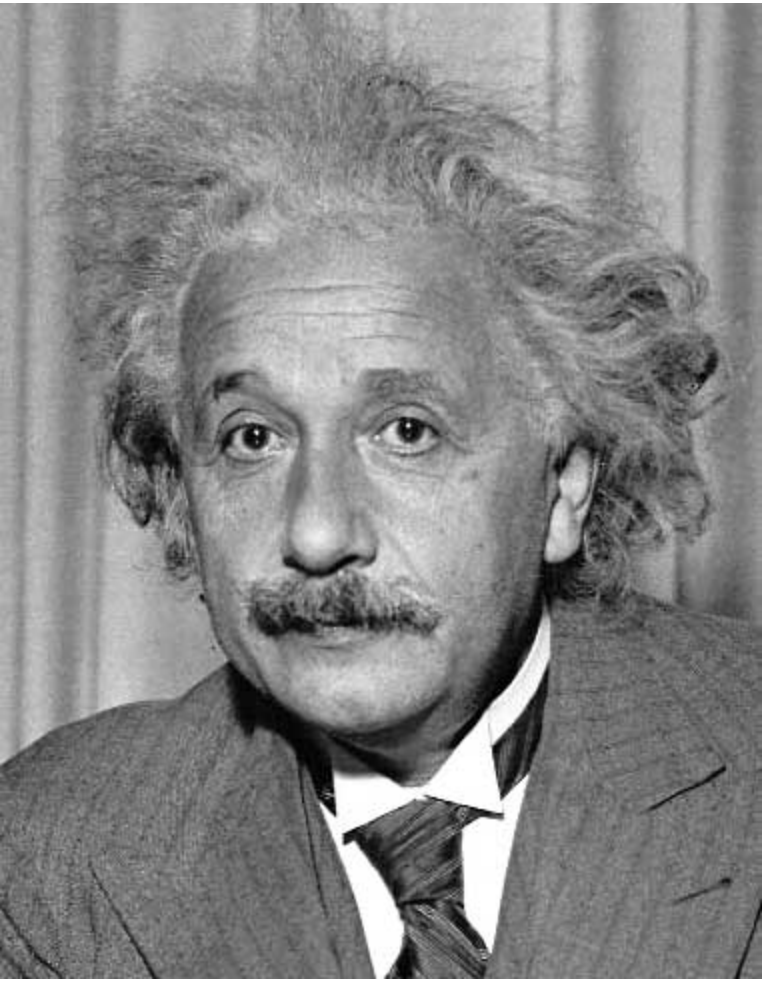
before



after

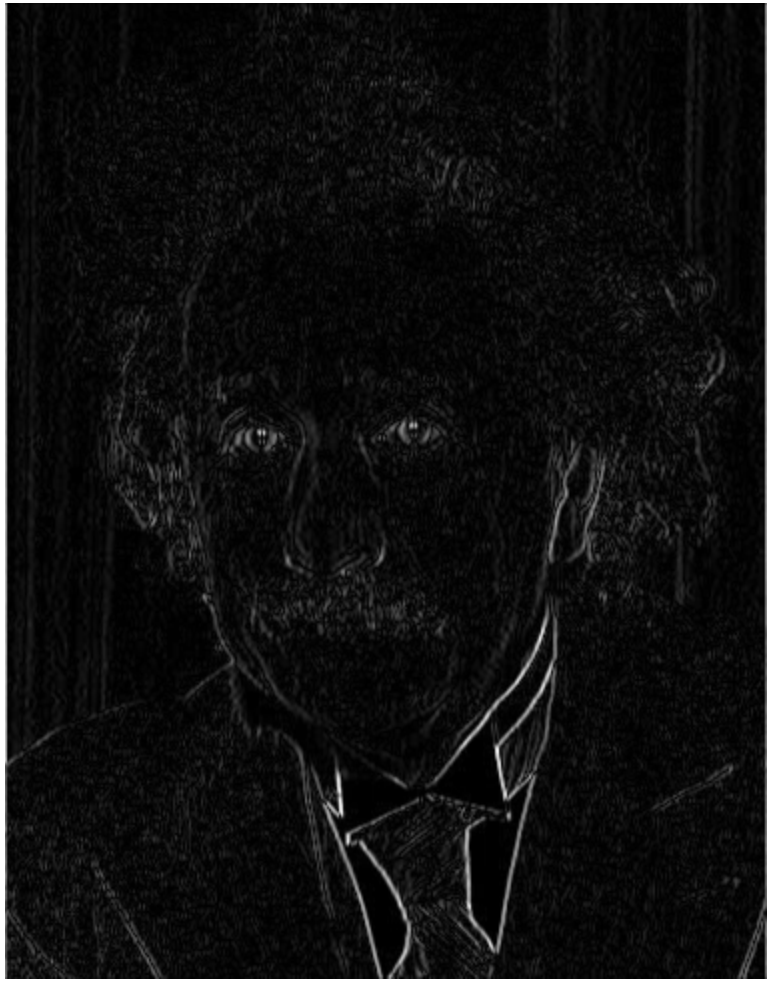
Source: D. Lowe

Other filters



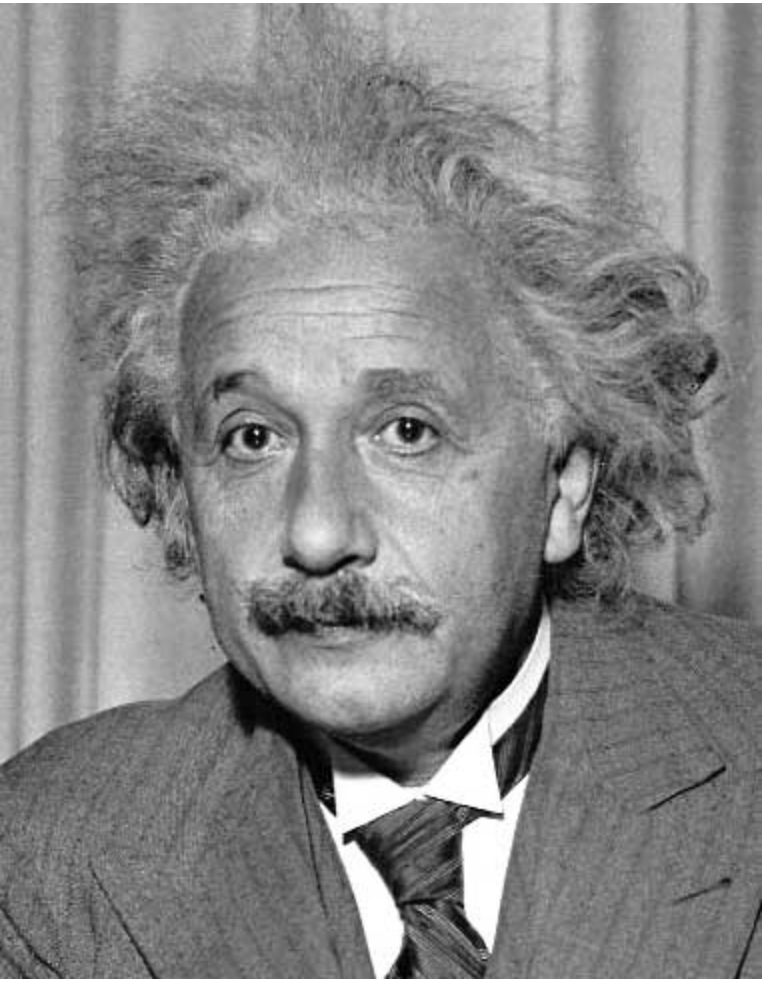
1	0	-1
2	0	-2
1	0	-1

Sobel



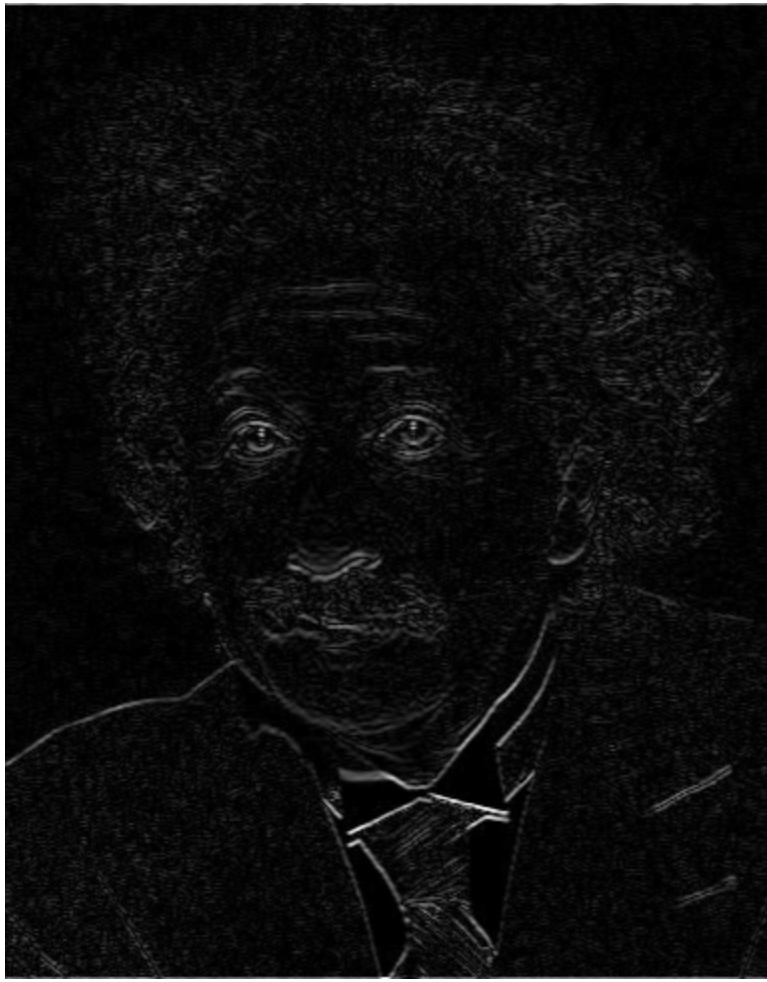
Vertical Edge
(absolute value)

Other filters



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

Sobel



Horizontal Edge
(absolute value)

Key properties

- **Linearity:** filter(f_1 + f_2) = filter(f_1) + filter(f_2)
- **Shift invariance:** same behavior regardless of pixel location: filter(shift(f)) = shift(filter(f))
- Theoretical result: any linear shift-invariant operator can be represented as a convolution

Source: S. Lazebnik

Properties in more detail

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k (a * b)$
- Identity: unit impulse $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$, $a * e = a$

Source: S. Lazebnik

Filtering vs. Convolution

- 2d filtering

f=filter I=image

  - `h=filter2(f,I);` or
    `h=imfilter(I,f);`

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

- 2d convolution
  - `h=conv2(f,I);`

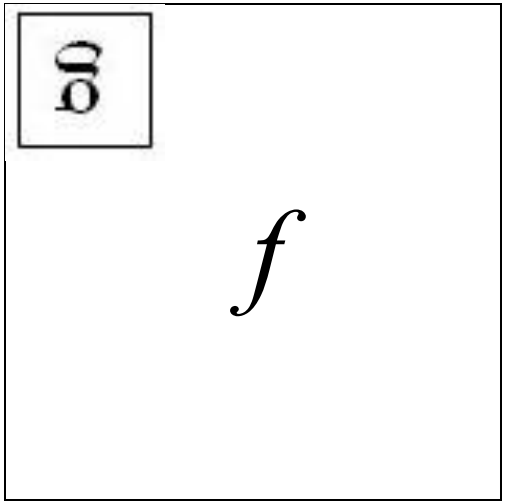
$$h[m,n] = \sum_{k,l} f[k,l]\, I[m-k, n-l]$$

Definition of convolution

- Let *f* be the image and *g* be the kernel. The output of convolving *f* with *g* is denoted *f* * *g*.

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l]\, g[k,l]$$

Convention:
kernel is "flipped"

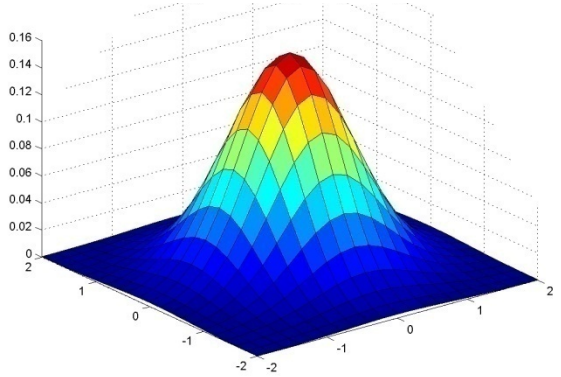


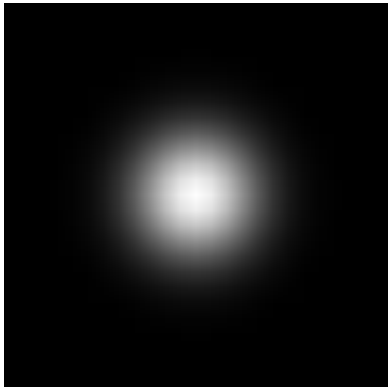
- See MATLAB functions: conv2, filter2, imfilter (the latter two don't flip the kernel)

Source: F. Durand

Important filter: Gaussian

- Spatially-weighted average





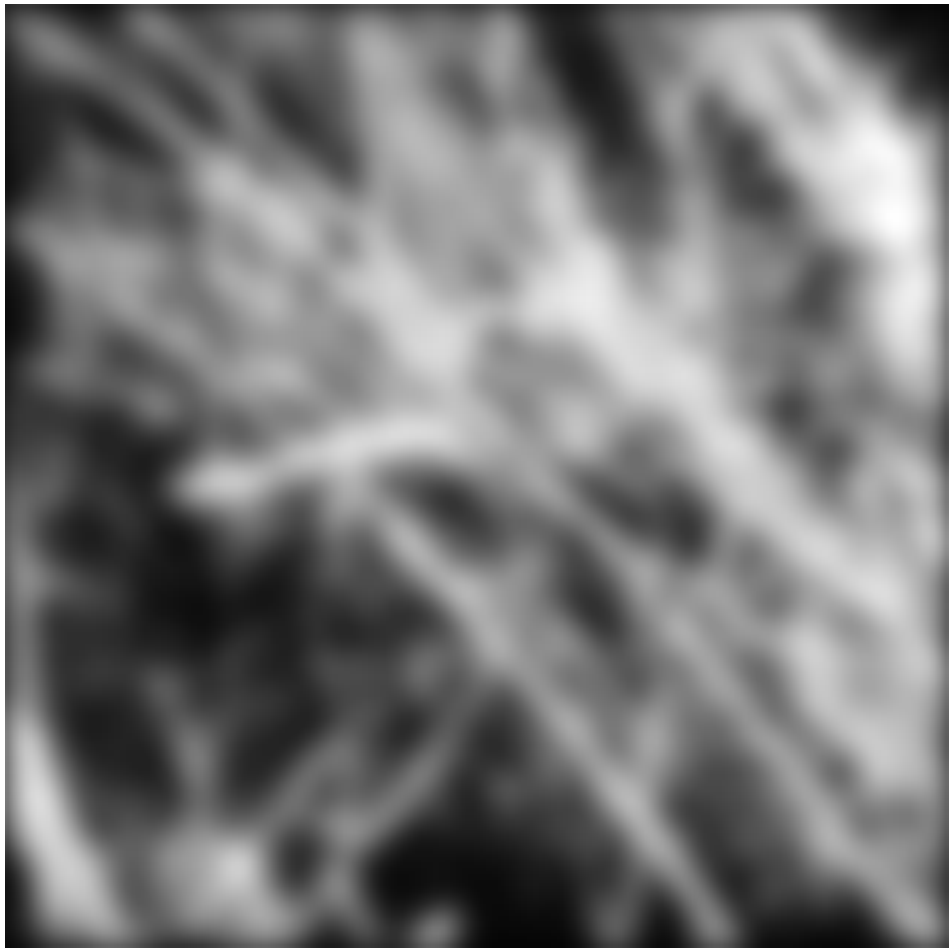
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

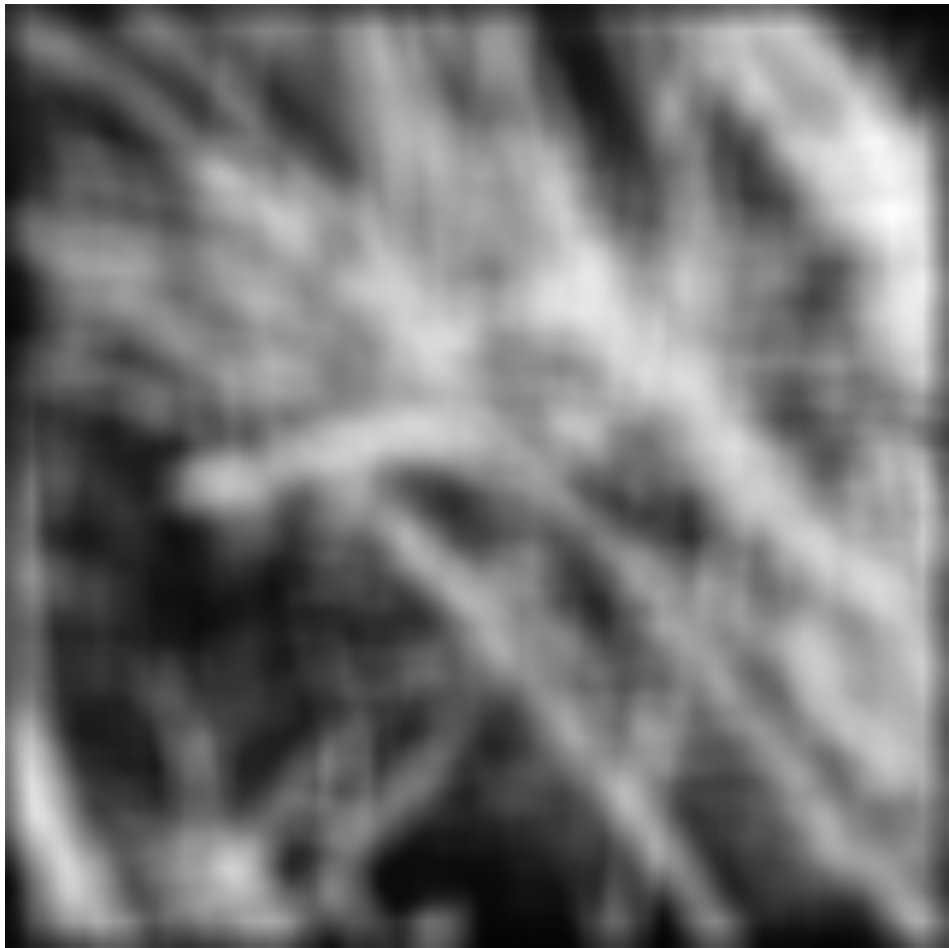
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Slide credit: Christopher Rasmussen

Smoothing with Gaussian filter



Smoothing with box filter



Gaussian filters

- Remove "high-frequency" components from the image (low-pass filter)
  - Images become more smooth
- Convolution with self is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into product of two 1D Gaussians

Source: K. Grauman

Separability of the Gaussian filter

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

$$= \left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{y^2}{2\sigma^2}}\right)$$

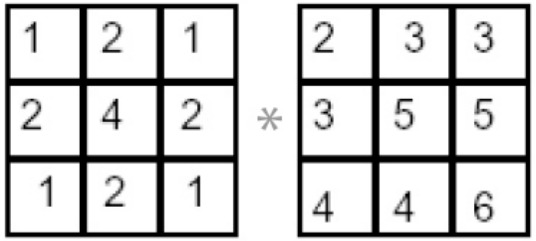
The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

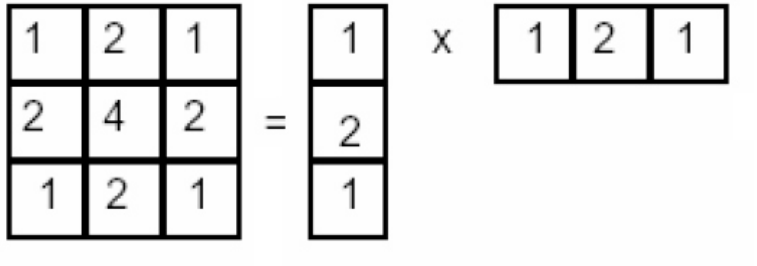
Source: D. Lowe

Separability example

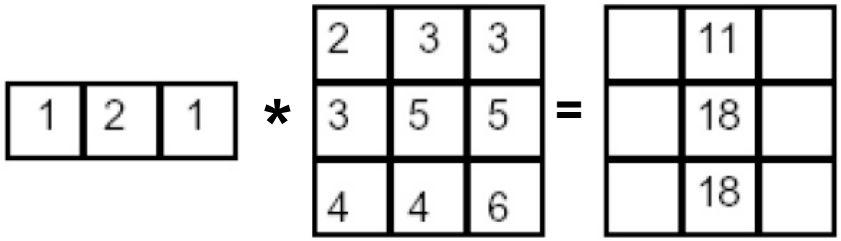
2D filtering
(center location only)



The filter factors
into a product of 1D
filters:



Perform filtering
along rows:



Followed by filtering
along the remaining column:

Source: K. Grauman

Separability

- Why is separability useful in practice?

Separability

- Why is separability useful in practice?

- Filter of size k*k requires k^2 operations per pixel

- Only 2k operations for separable kernels:

$$K = vh^T$$

Practical matters

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

Source: S. Marschner

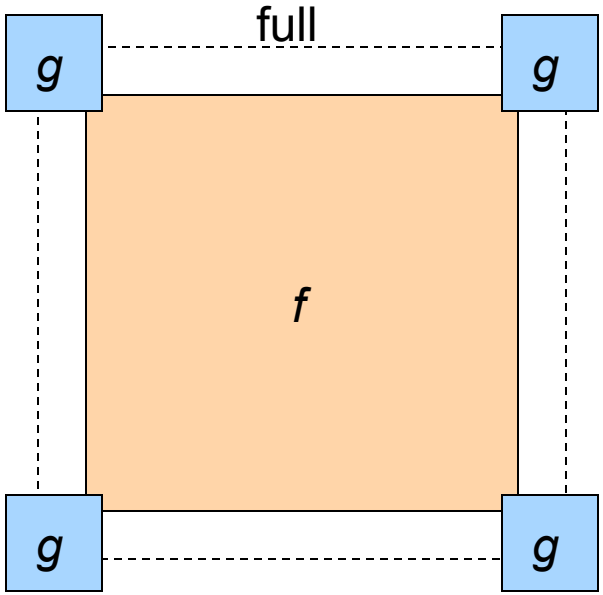
Practical matters

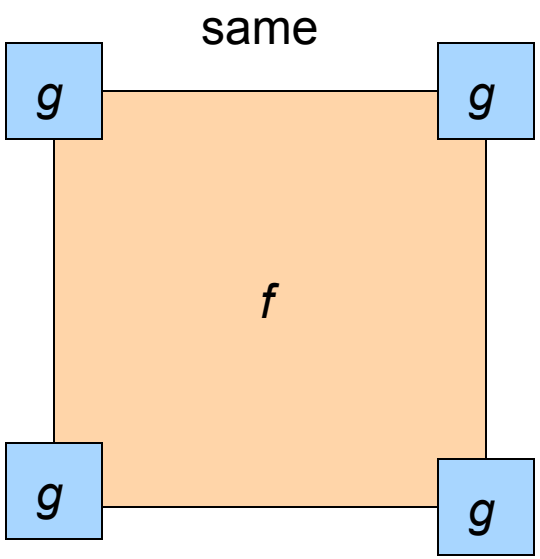
– methods (MATLAB):
  - clip filter (black): imfilter(f, g, 0)
  - wrap around: imfilter(f, g, ‘circular’)
  - copy edge: imfilter(f, g, ‘replicate’)
  - reflect across edge: imfilter(f, g, ‘symmetric’)

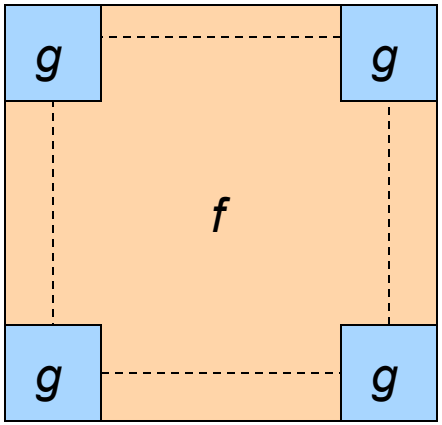
Source: S. Marschner

Practical matters

- 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







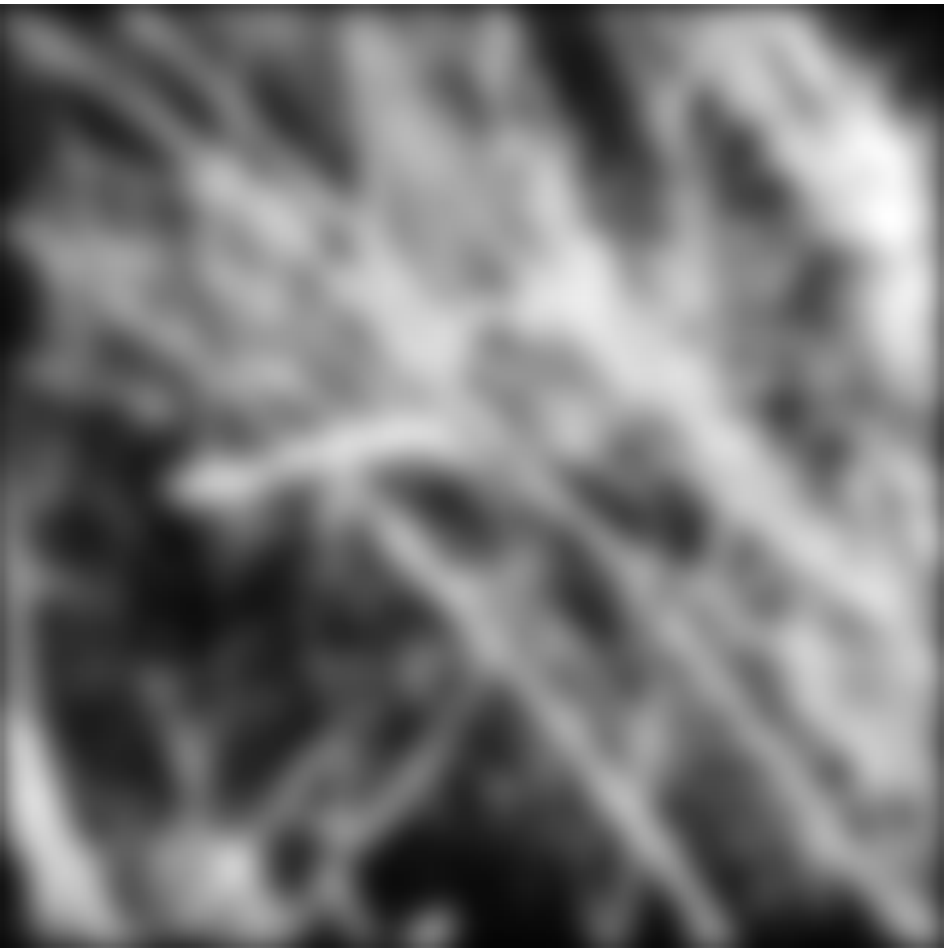
Source: S. Lazebnik

Image filters in frequency domain

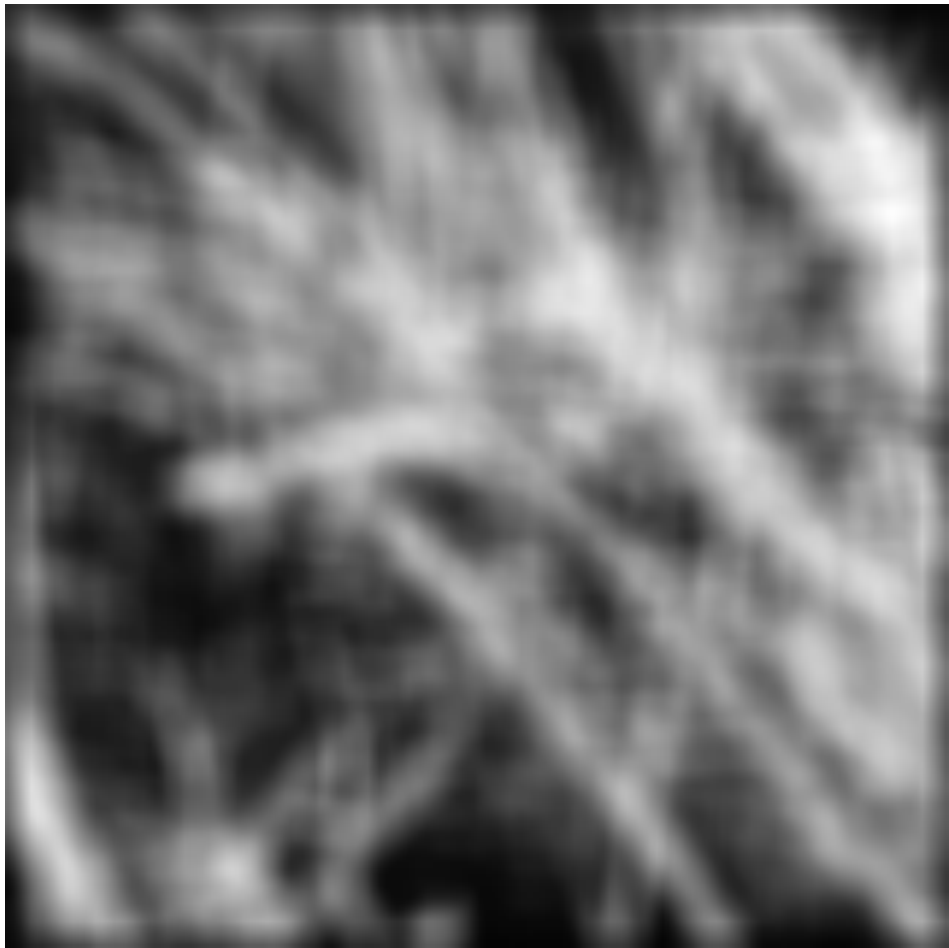
Source: D. Hoiem

Why does the Gaussian give a nice smooth image, but the square filter give edgy artifacts?

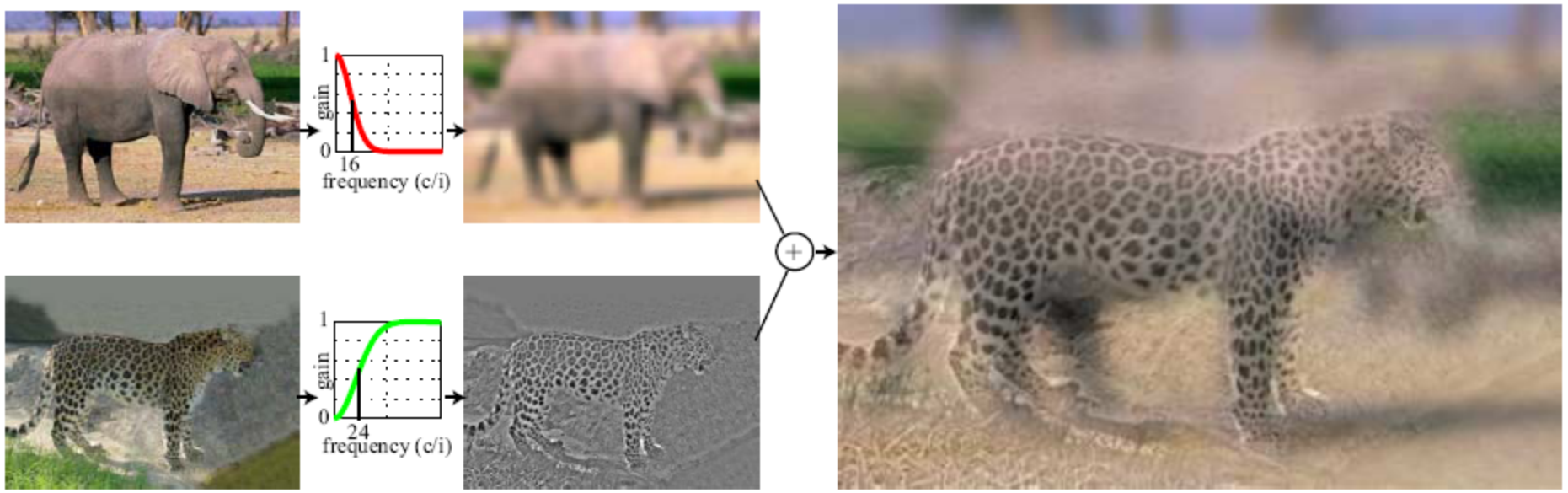
Gaussian 



Box filter 



Hybrid Images



- A. Oliva, A. Torralba, P.G. Schyns, "Hybrid Images," SIGGRAPH 2006

Source: D. Hoiem

Why do we get different, distance-dependent interpretations of hybrid images?

?

Adapted from a slide by D. Hoiem

Why does a lower resolution image still make sense to us? What do we lose?

Source: D. Hoiem

Image: http://www.flickr.com/photos/igorms/136916757/

Thinking in terms of frequency

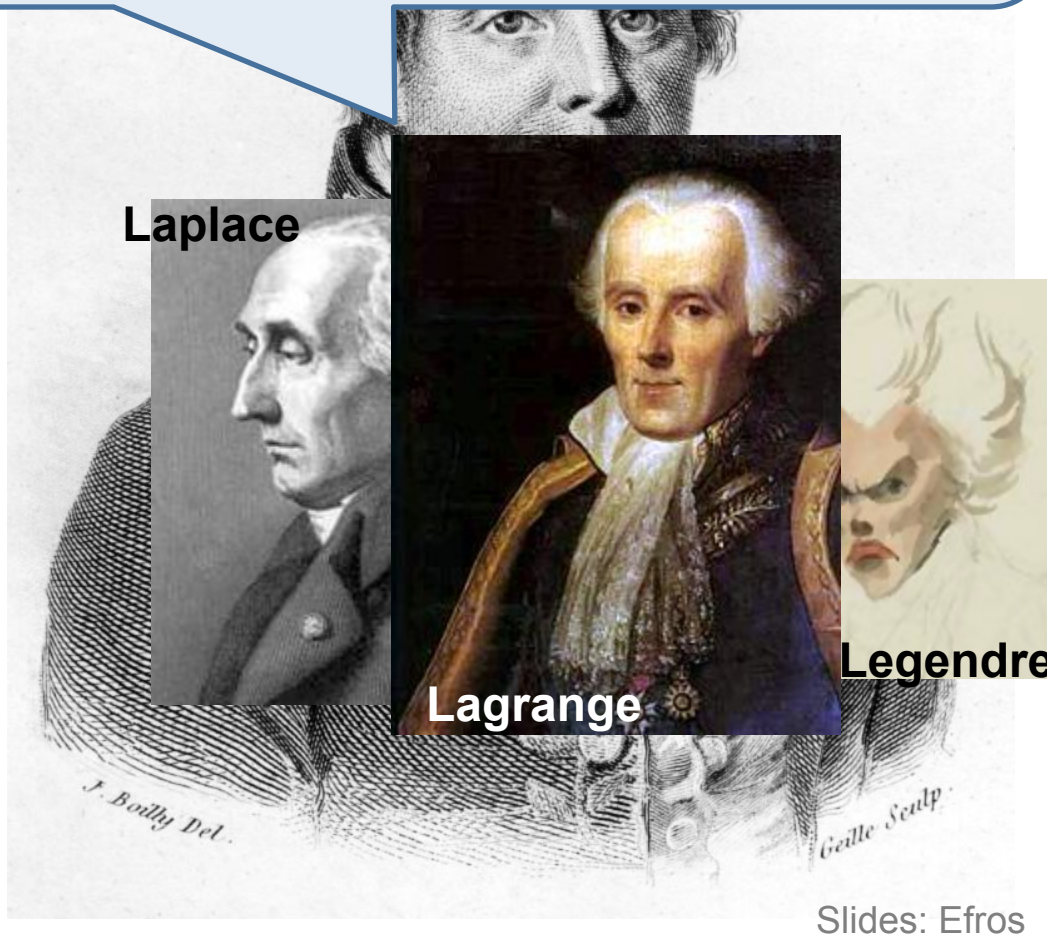
Jean Baptiste Joseph Fourier (1768-1830)

had crazy idea (1807):

Any *univariate function can rewritten as a weighted sum sines and cosines of differen frequencies.*

...the manner in which the author arrives at these equations is not exempt of difficulties and...his analysis to integrate them still leaves something to be desired on the score of generality and even rigour.

- Don't believe it?
  - Neither did Lagrange, Laplace, Poisson and other big wigs
  - Not translated into English until 1878!
- But it's (mostly) true!
  - called Fourier Series
  - there are some subtle restrictions



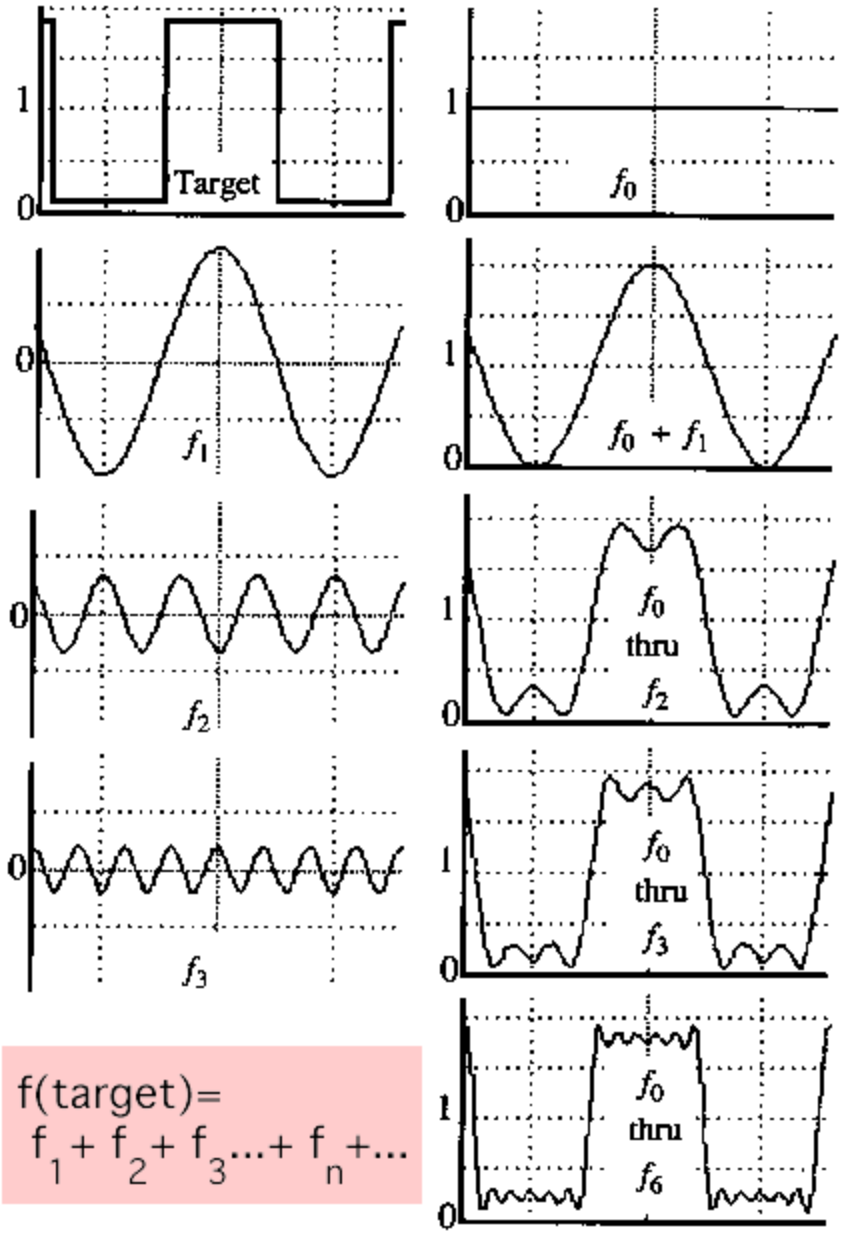
Slides: Efros

A sum of sines

Our building block:

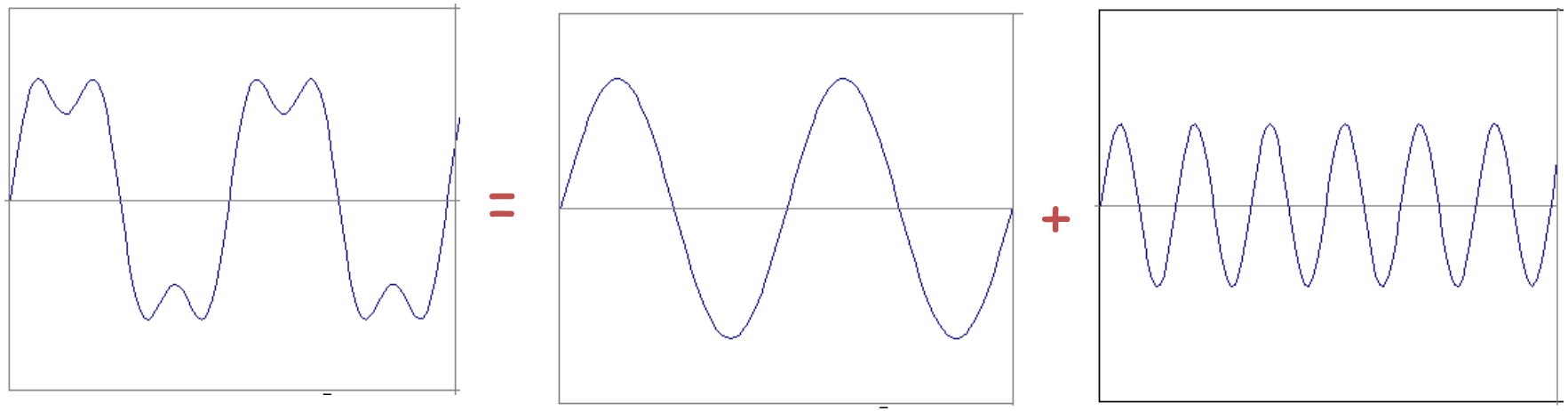
$$A\sin(\omega x + \phi)$$

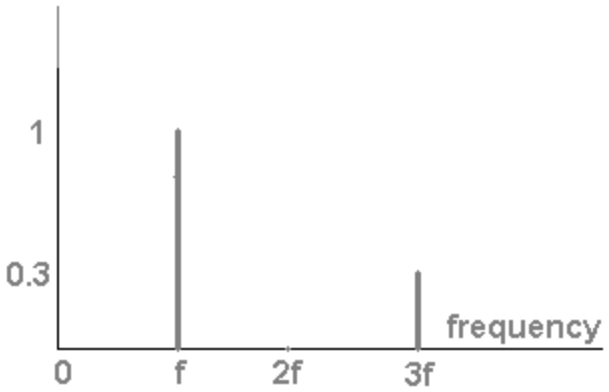
Add enough of them to get any signal *f(x)* you want!



Frequency Spectra

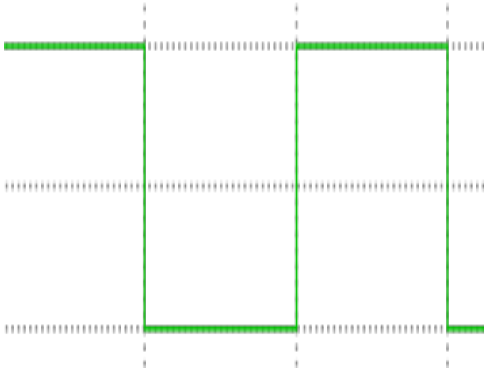
- example : $g(t) = \sin(2\pi f\, t) + (1/3)\sin(2\pi(3f)\, t)$





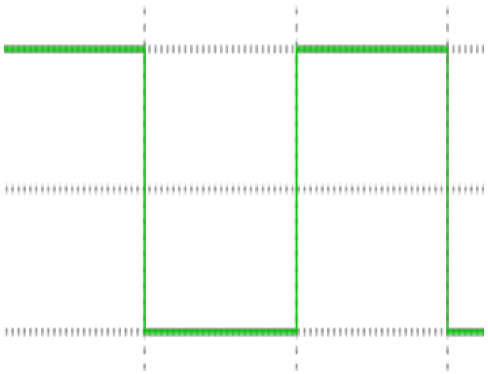
Slides: Efros

Frequency Spectra

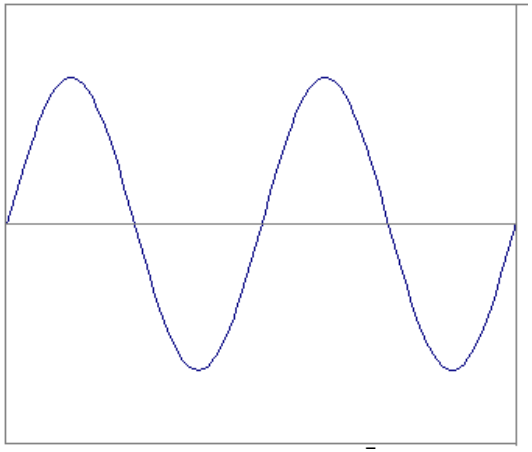


Source: J. Hays

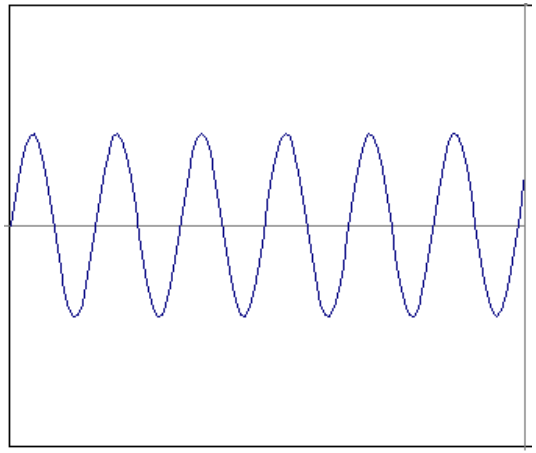
Frequency Spectra



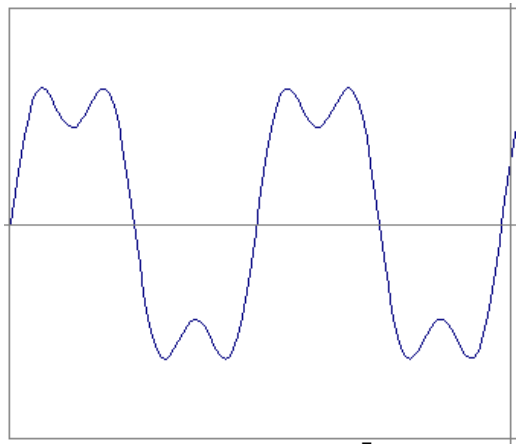
=



+

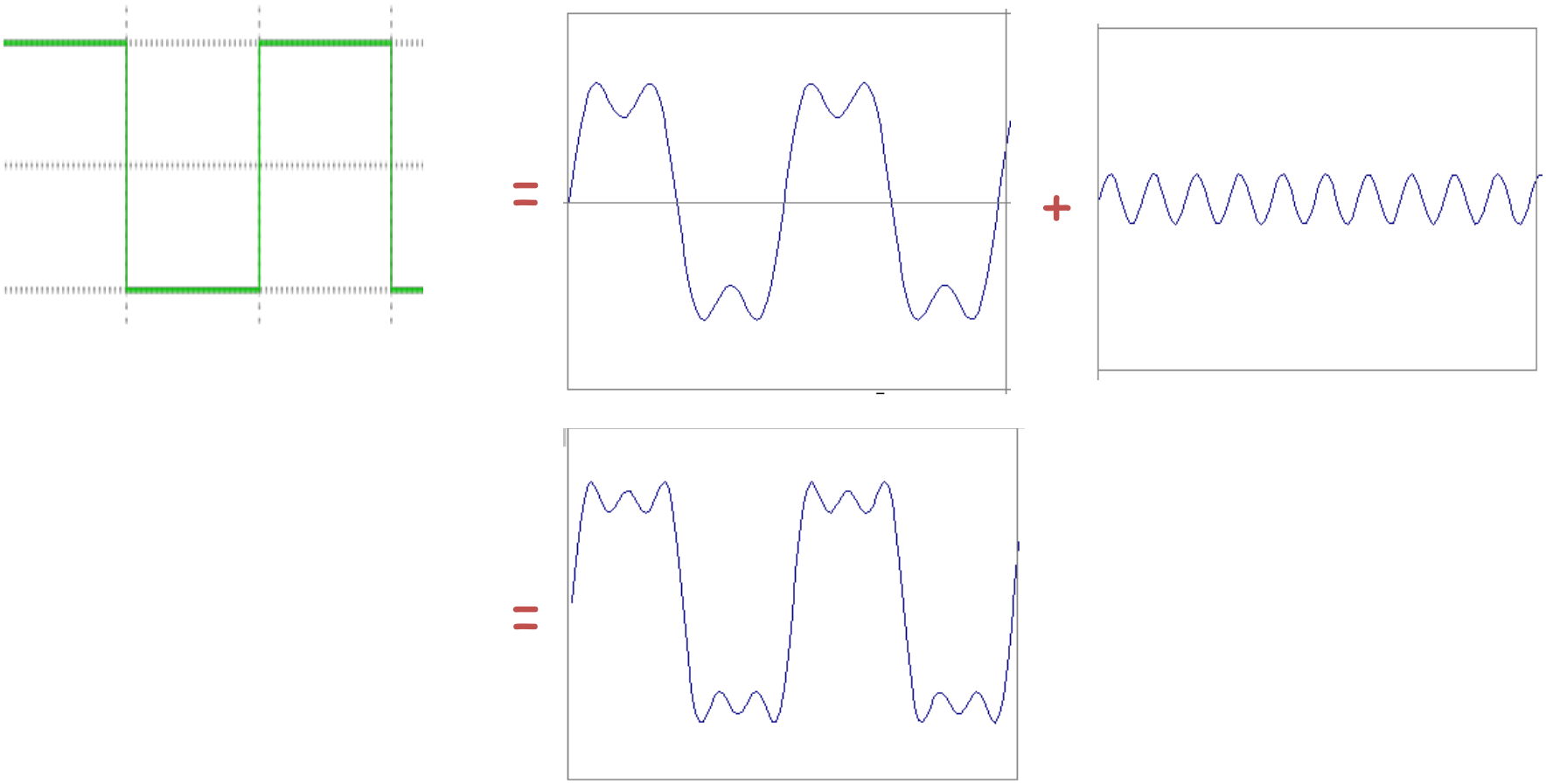


=



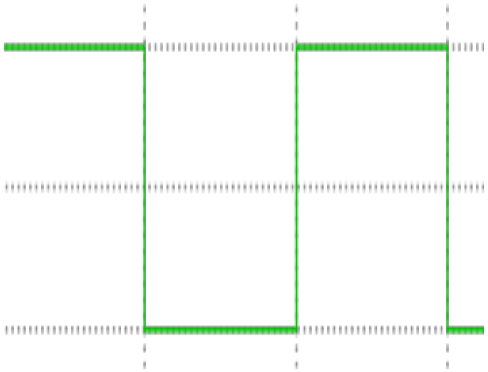
Source: J. Hays

Frequency Spectra

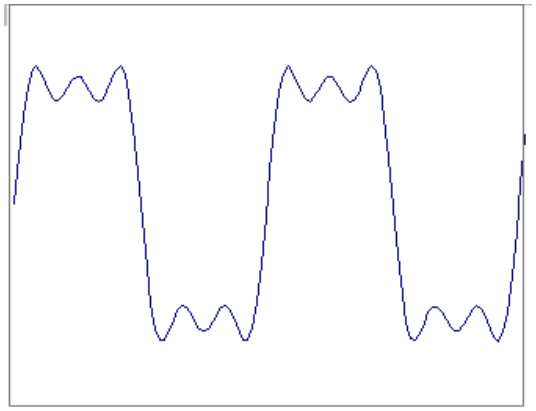


Source: J. Hays

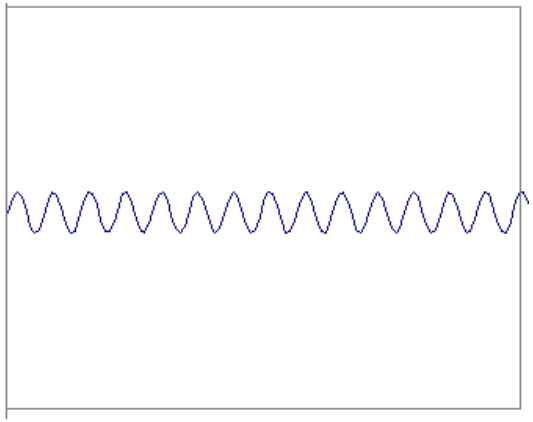
Frequency Spectra



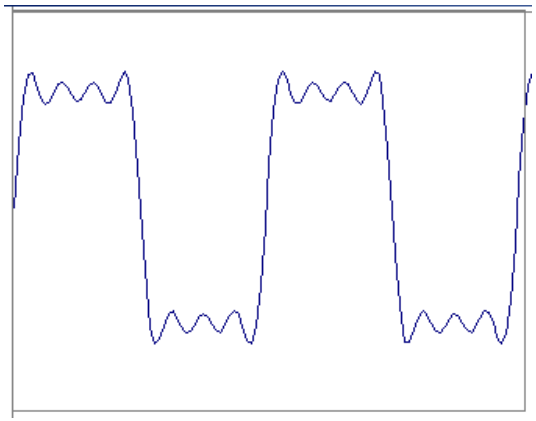
=



+

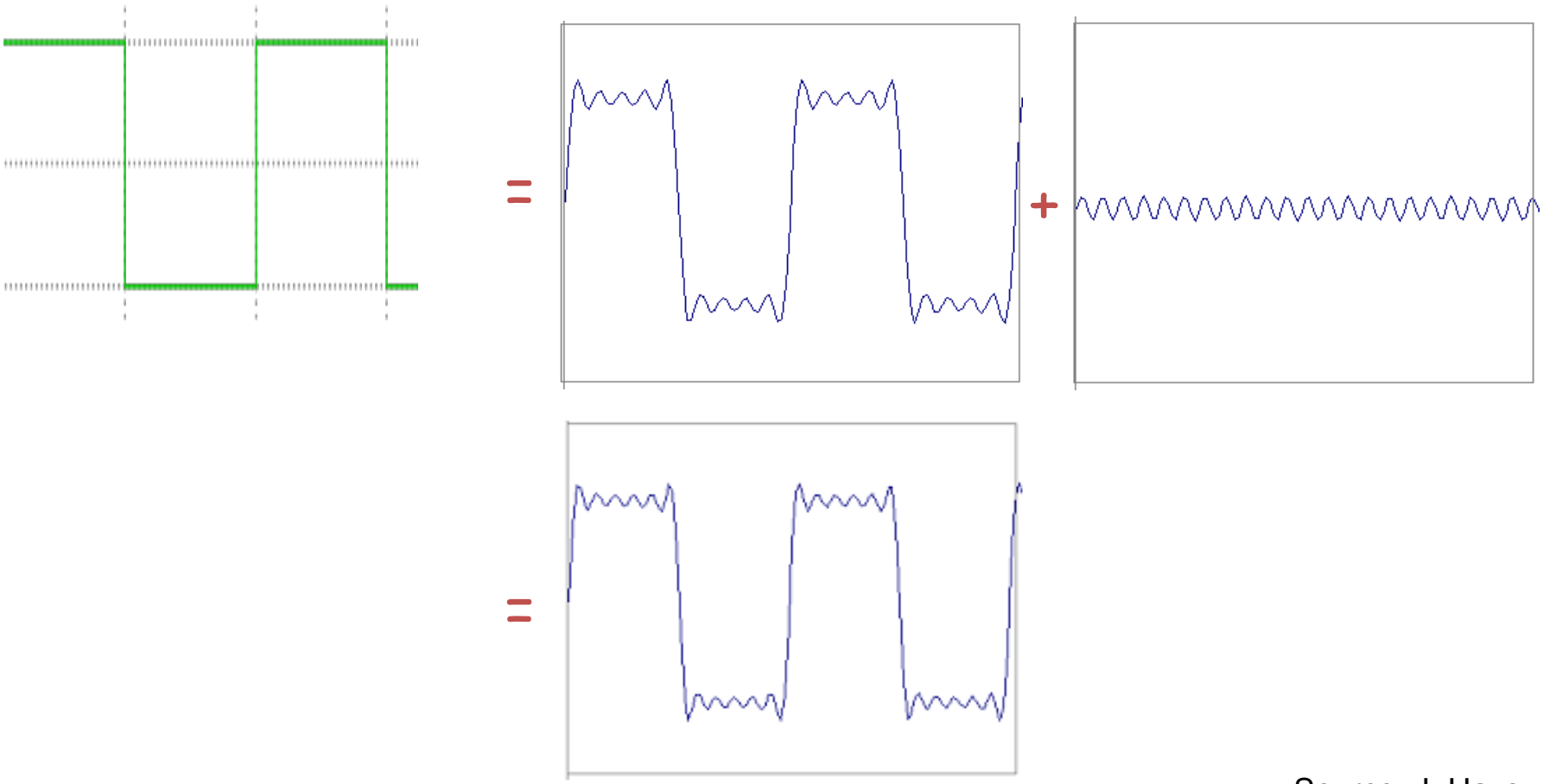


=



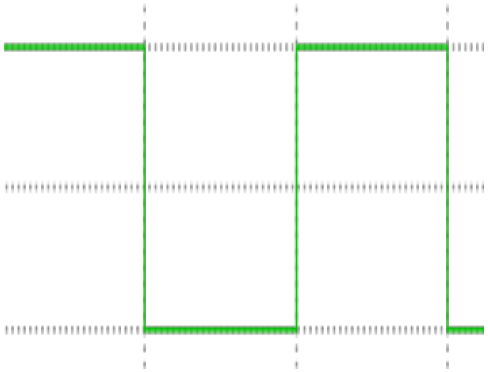
Source: J. Hays

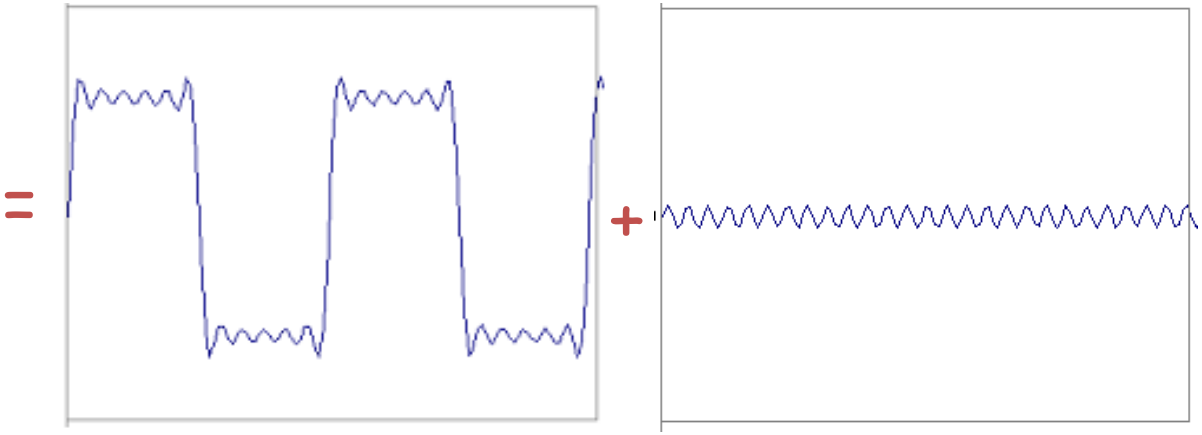
Frequency Spectra

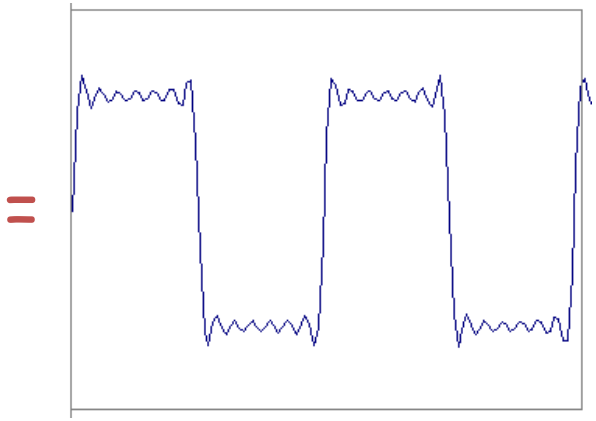


Source: J. Hays

Frequency Spectra

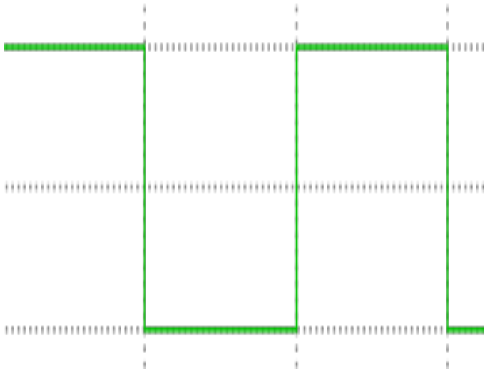






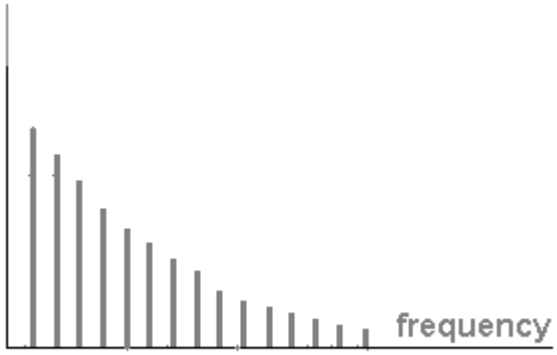
Source: J. Hays

Frequency Spectra



=

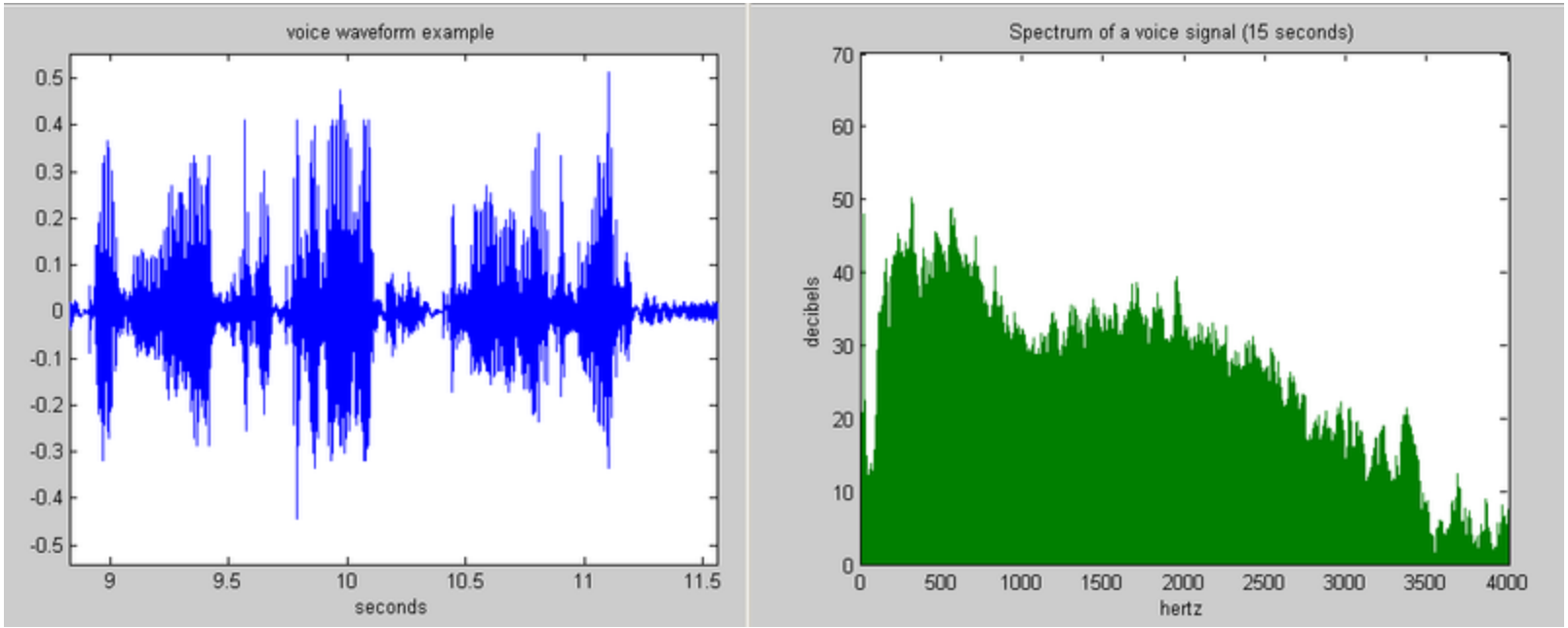
$$A\sum_{k=1}^{\infty}\frac{1}{k}\sin(2\pi kt)$$



Source: J. Hays

Example: Music

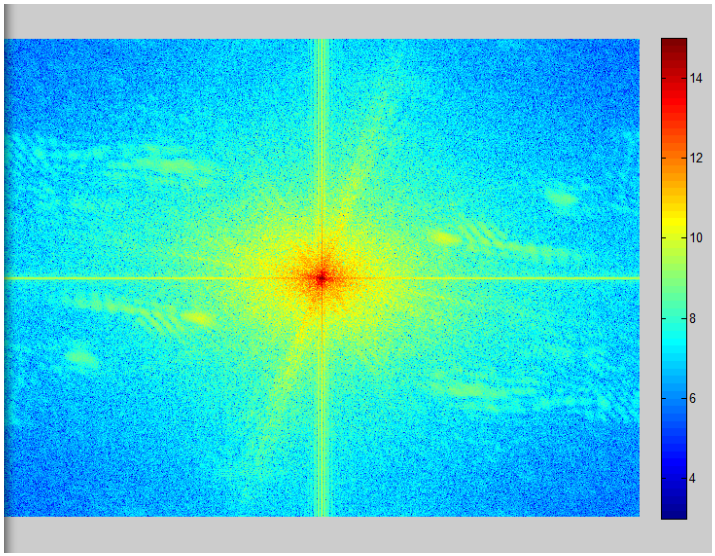
- We think of music in terms of frequencies at different magnitudes



Source: D. Hoiem

Other signals

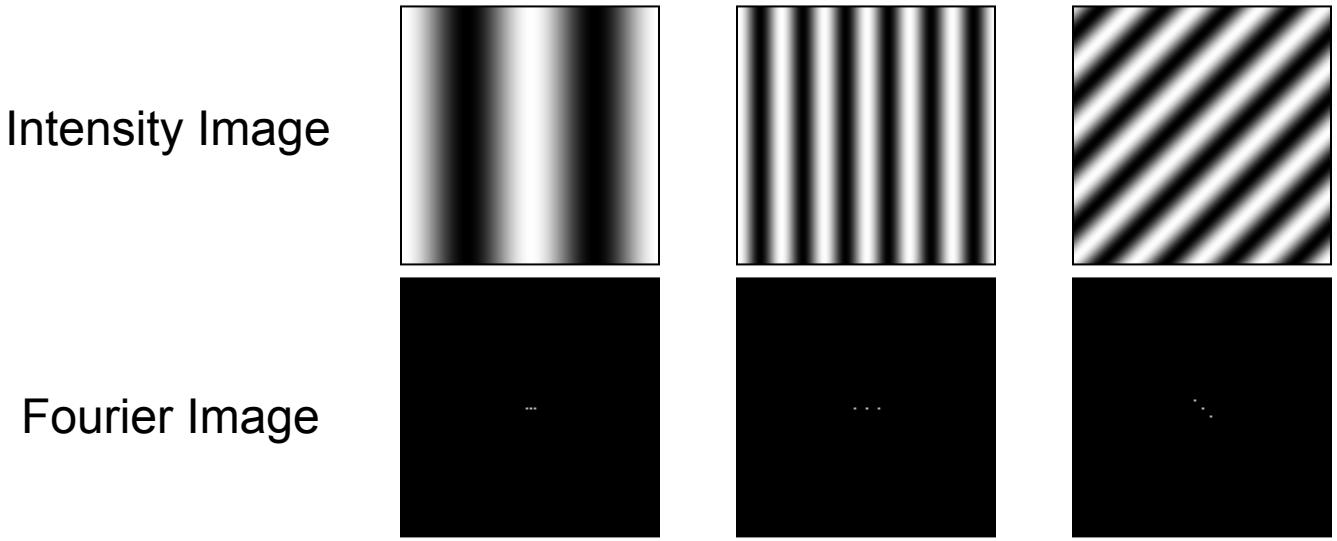
- We can also think of all kinds of other signals the same way



Source: D. Hoiem

Fourier analysis in images

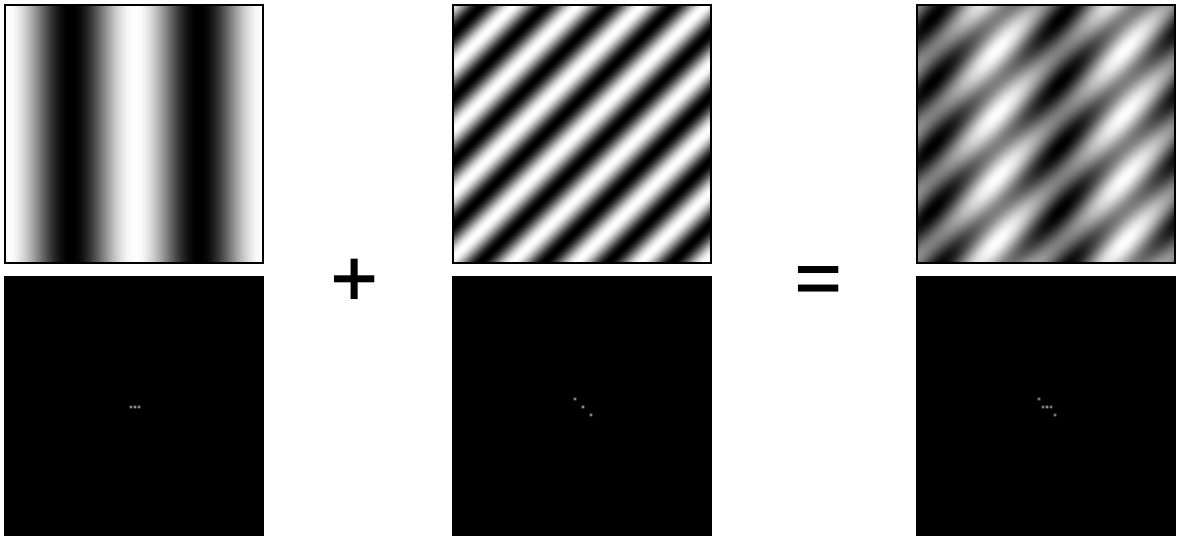
- In 2D case we have two-dimensional frequency (which encodes also the 2D orientation of the sine wave)



Slide adapted from D. Hoiem

http://sharp.bu.edu/~slehar/fourier/fourier.html#filtering

Signals can be composed

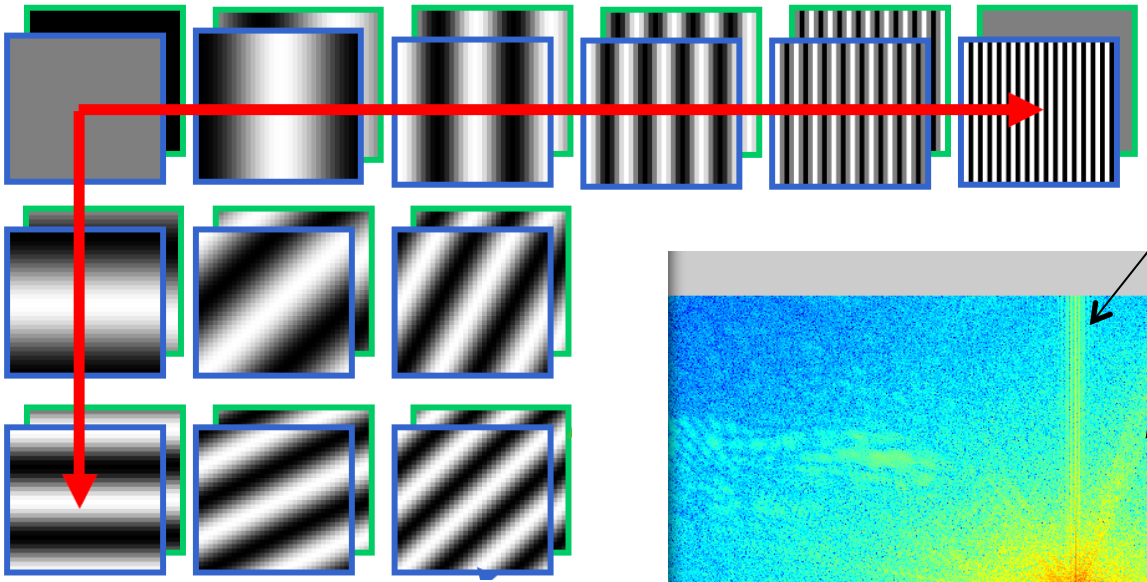


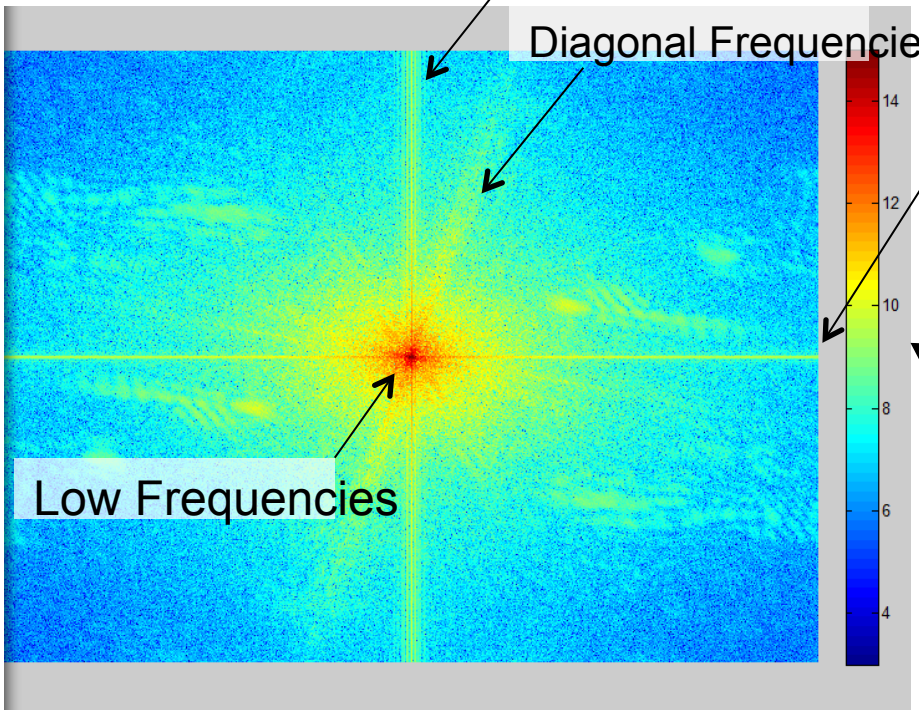
http://sharp.bu.edu/~slehar/fourier/fourier.html#filtering
More: http://www.cs.unm.edu/~brayer/vision/fourier.html

Source: D. Hoiem

Fourier Bases

Teases away fast vs. slow changes in the image.





This change of basis is the Fourier Transform

Source: Hays, Hoiem

Fourier Transform

Source: S. Narasimhan

- We want to understand the frequency ω of our signal. So, let's reparametrize the signal by ω instead of *x*:

- For every ω from 0 to inf, ***F(ω)*** holds the amplitude *A* and phase ϕ of the corresponding sine $A\sin(\omega x + \phi)$
  - How can *F* hold both? Complex number trick!

$$F(\omega) = R(\omega) + iI(\omega)$$

$$A = \pm\sqrt{R(\omega)^2 + I(\omega)^2} \qquad \phi = \tan^{-1}\frac{I(\omega)}{R(\omega)}$$

Fourier Transform

- Fourier transform stores the magnitude and phase at each frequency
  - Magnitude encodes how much signal there is at a particular frequency
  - Phase encodes spatial information (indirectly)
  - For mathematical convenience, this is often notated in terms of real and complex numbers

Amplitude: $A = \pm\sqrt{R(\omega)^2 + I(\omega)^2}$ Phase: $\phi = \tan^{-1}\frac{I(\omega)}{R(\omega)}$

Euler's formula: $e^{inx} = \cos(nx) + i\sin(nx)$

Source: D. Hoiem

Computing the Fourier Transform

$$H(\omega) = \mathcal{F}\{h(x)\} = Ae^{j\phi}$$

Continuous

$$H(\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x}dx$$

Discrete

$$H(k) = \frac{1}{N}\sum_{x=0}^{N-1} h(x)e^{-j\frac{2\pi kx}{N}}$$

k = -N/2..N/2

Fast Fourier Transform (FFT): NlogN

Source: D. Hoiem

Source: L. Xie

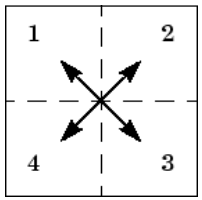
Computing 2D-DFT

DFT $$y(u,v) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m,n) e^{\frac{-j2\pi um}{M}} e^{\frac{-j2\pi vn}{N}}$$

IDFT $$x(m,n) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} y(u,v) e^{\frac{j2\pi um}{M}} e^{\frac{j2\pi vn}{N}}$$

- Discrete, 2-D Fourier & inverse Fourier transforms are implemented in `fft2` and `ifft2`, respectively
- `fftshift`: Move origin (DC component) to image center for display
- Example:

```
>> I = imread('test.png');   % Load grayscale image
>> F = fftshift(fft2(I));    % Shifted transform
>> imshow(log(abs(F)),[]);   % Show log magnitude
>> imshow(angle(F),[]);      % Show phase angle
```



The Convolution Theorem

- The Fourier transform of the convolution of two functions is the product of their Fourier transforms

$$\mathrm{F}[g * h] = \mathrm{F}[g]\mathrm{F}[h]$$

- The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms

$$\mathrm{F}^{-1}[gh] = \mathrm{F}^{-1}[g] * \mathrm{F}^{-1}[h]$$

- **Convolution** in spatial domain is equivalent to **multiplication** in frequency domain!

Source: D. Hoiem

Properties of Fourier Transforms

- Linearity $\mathcal{F}[ax(t) + by(t)] = a\mathcal{F}[x(t)] + b\mathcal{F}[y(t)]$

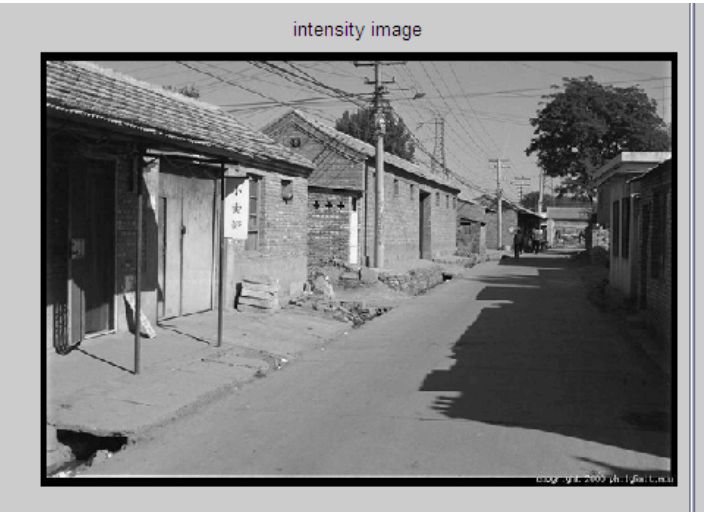
- Fourier transform of a real signal is symmetric about the origin

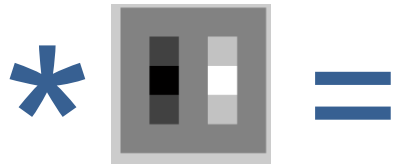
- The energy of the signal is the same as the energy of its Fourier transform

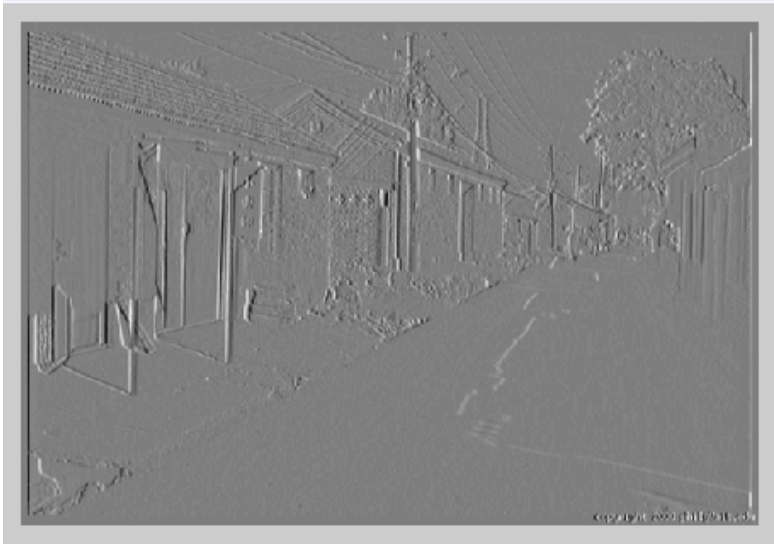
See Szeliski Book (3.4)

Filtering in spatial domain

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



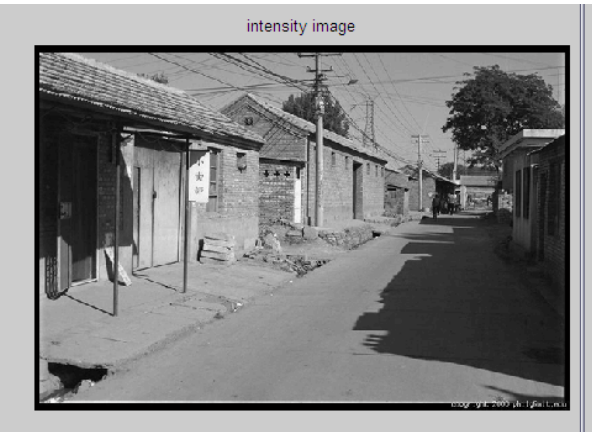




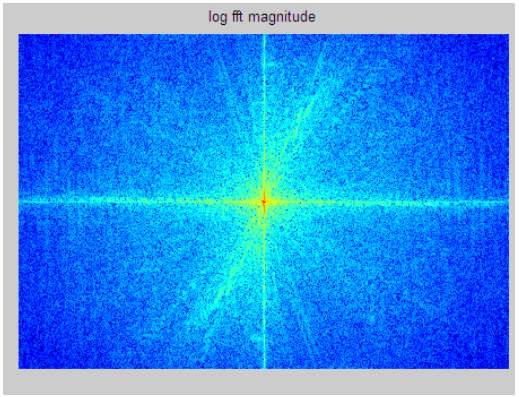
Source: D. Hoiem

Filtering in frequency domain

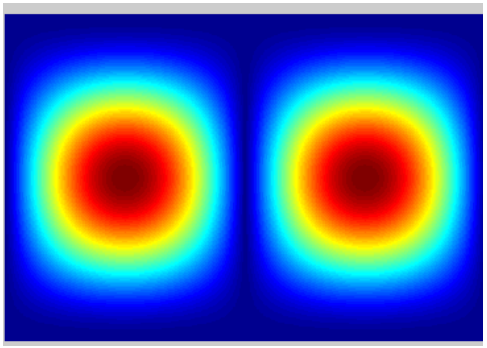
FFT



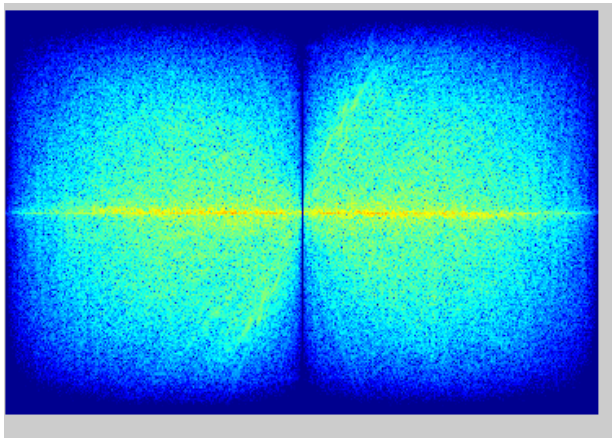
FFT



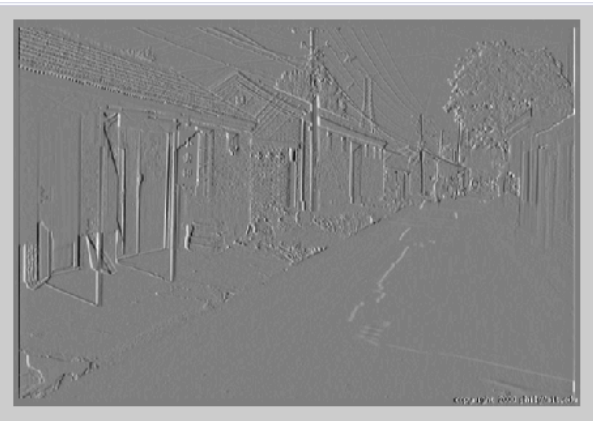
×



=



Inverse FFT



Source: Hoiem

Questions

Which has more information, the phase or the magnitude?

What happens if you take the phase from one image and combine it with the magnitude from another image?

Example: amplitude vs. phase

Source: L. Xie

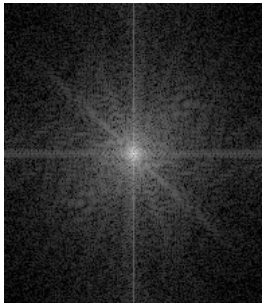
A = “Aron”

P = “Phyllis”

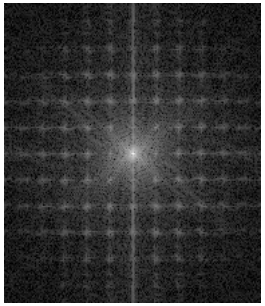
FA = fft2(A)

FP = fft2(P)

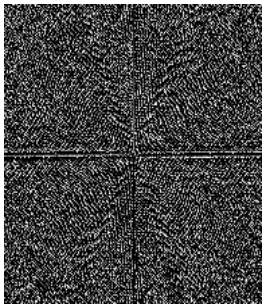
log(abs(FA))



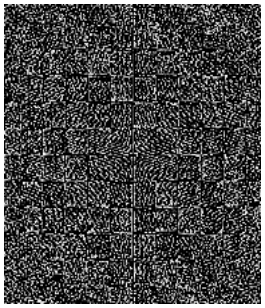
log(abs(FP))



angle(FA)



angle(FP)



ifft2(abs(FA), angle(FP))

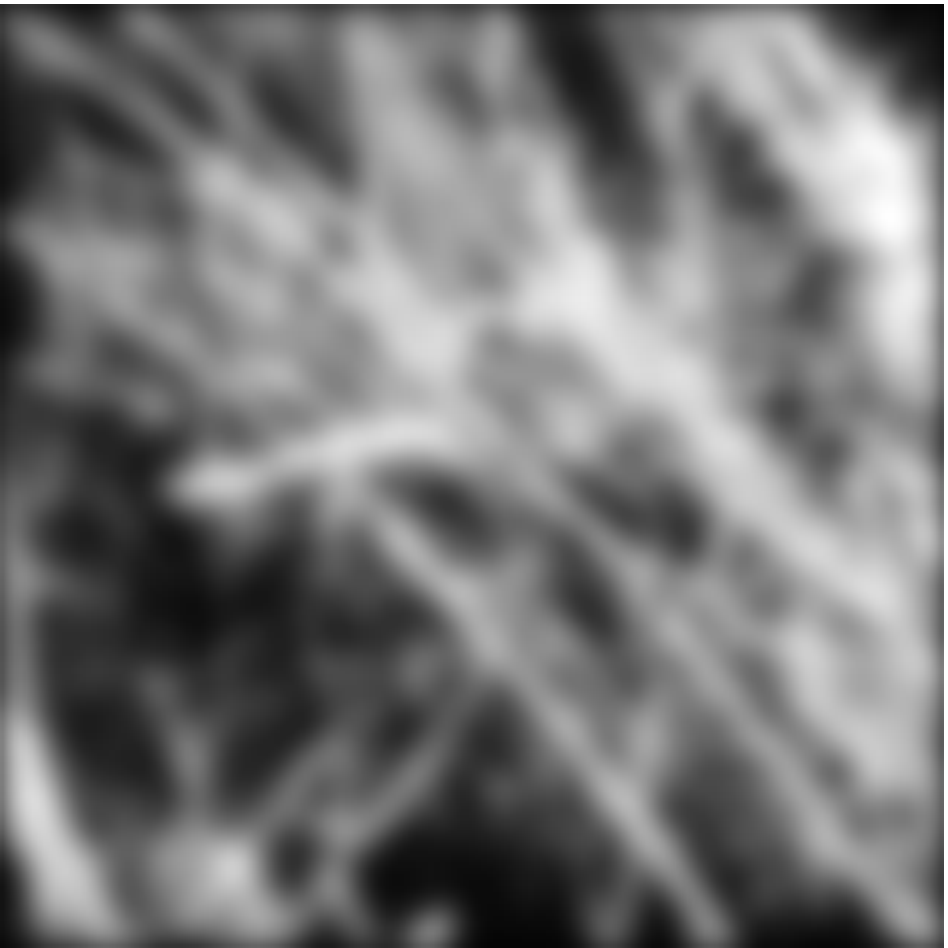
ifft2(abs(FP), angle(FA))

Adpated from http://robotics.eecs.berkeley.edu/~sastry/ee20/vision2/vision2.html

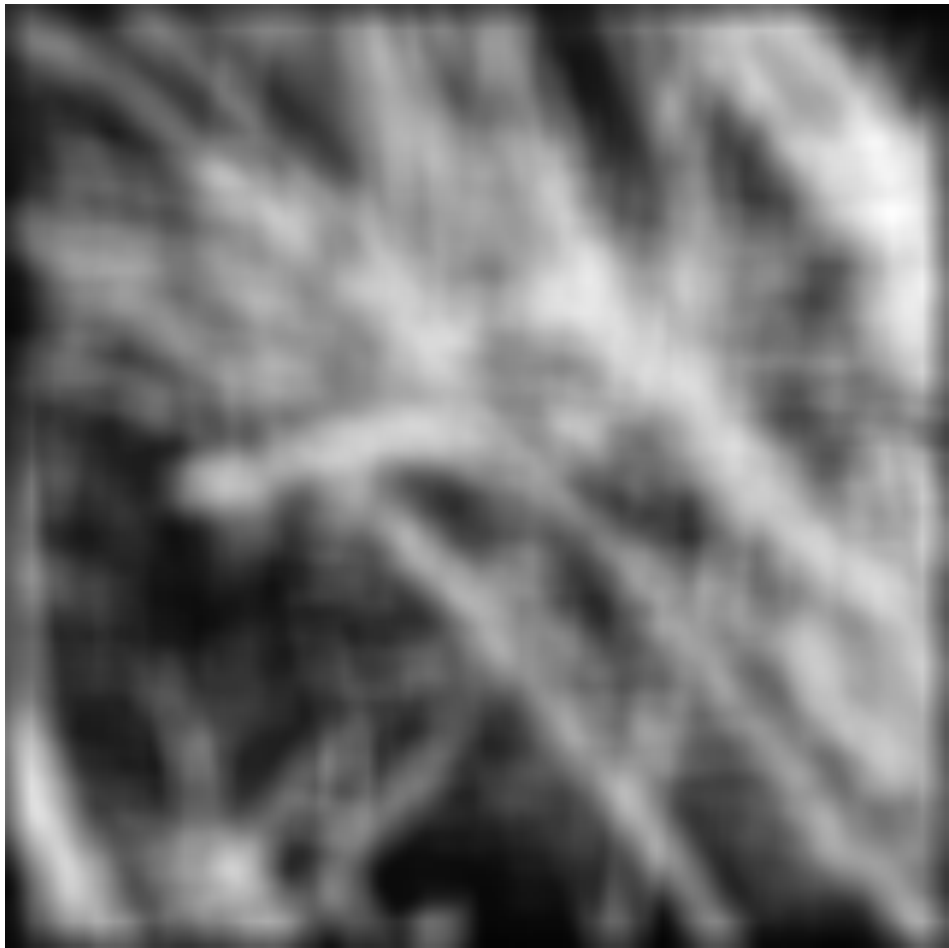
Filtering

Why does the Gaussian give a nice smooth image, but the square filter give edgy artifacts?

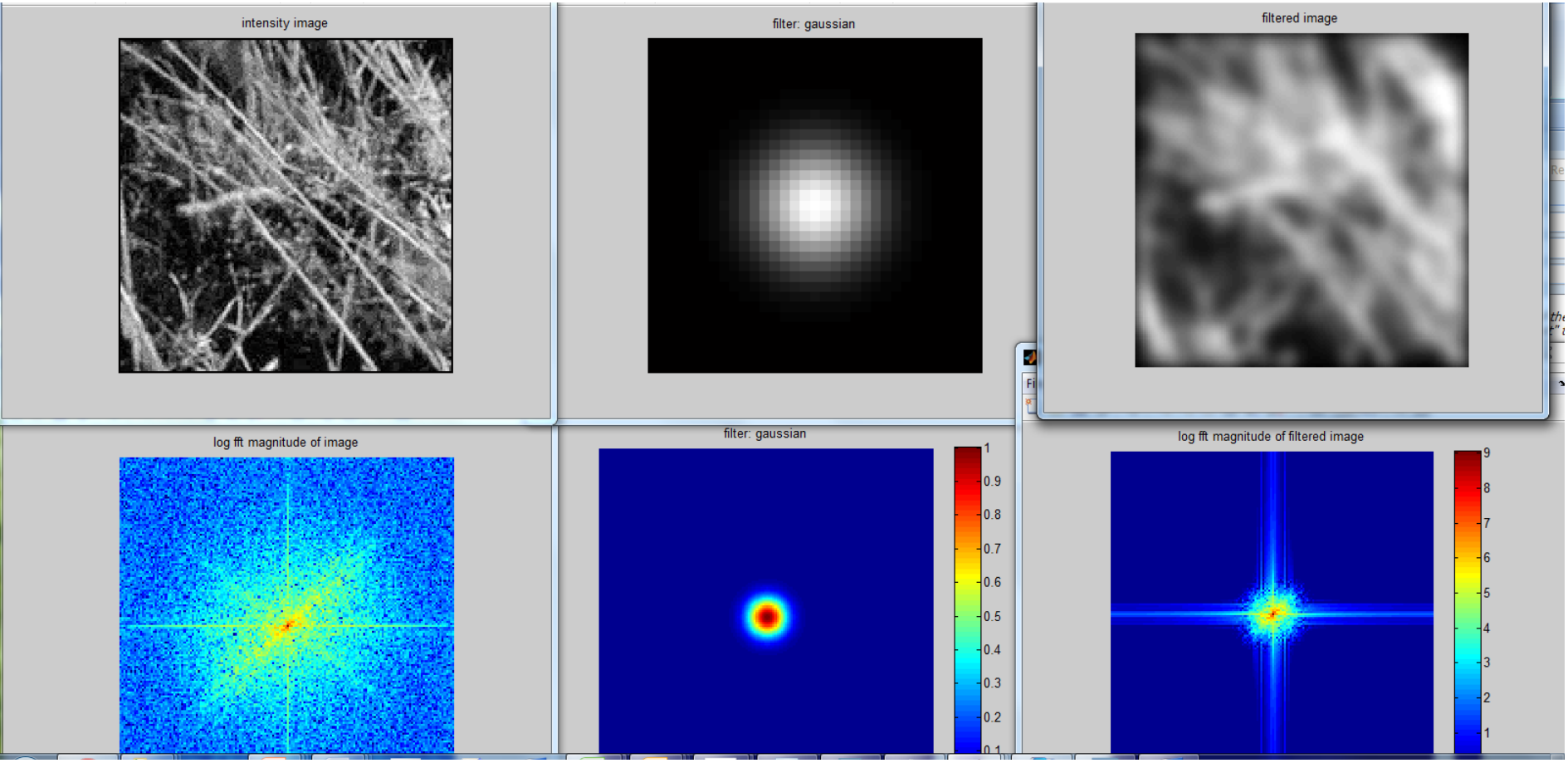
Gaussian 



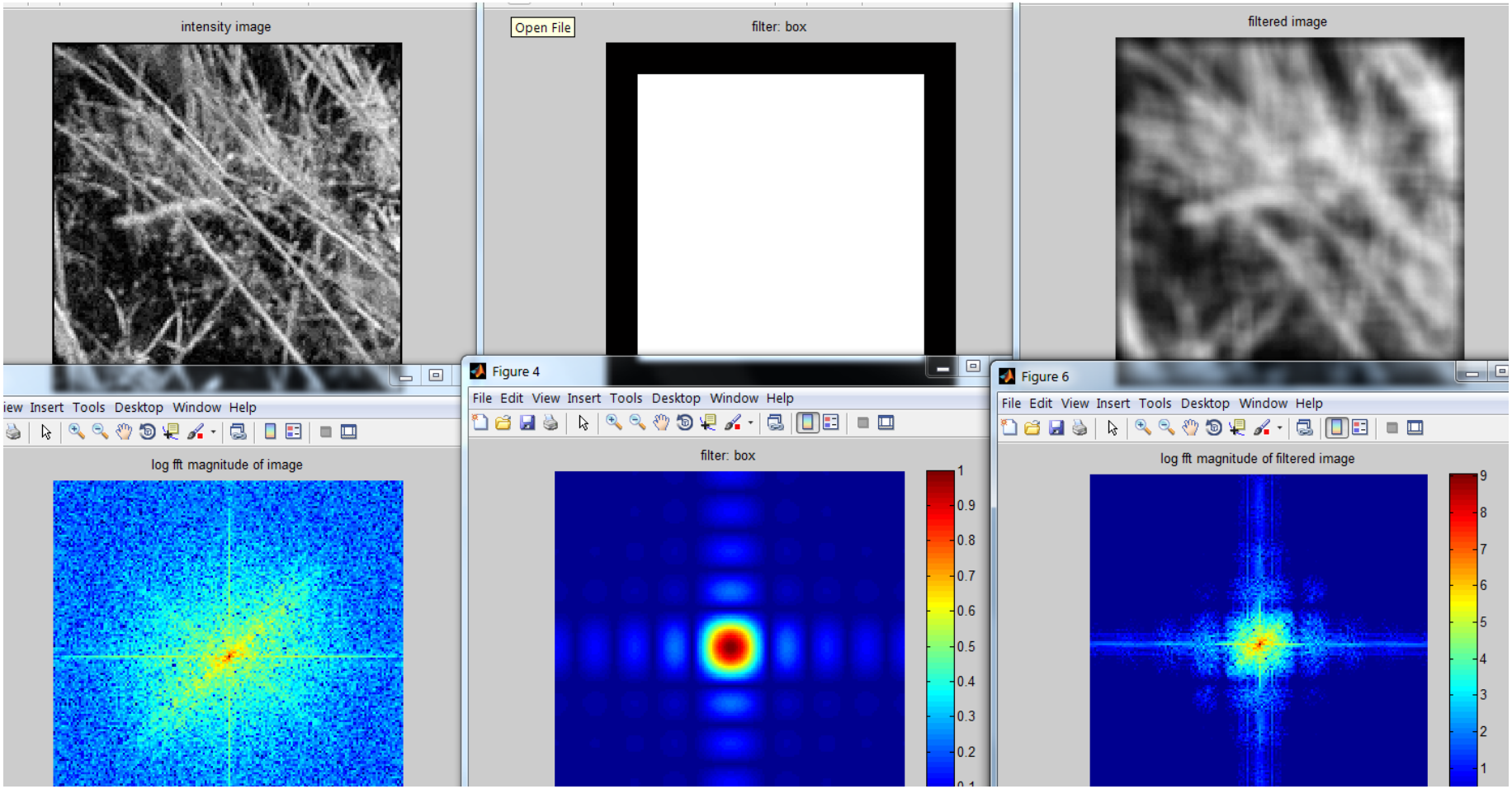
Box filter 



Gaussian


intensity image
filter: gaussian
filtered image
log fft magnitude of image
filter: gaussian
log fft magnitude of filtered image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
9
8
7
6
5
4
3
2
1

Box Filter


intensity image
Open File
filter: box
filtered image
iew Insert Tools Desktop Window Help
log fft magnitude of image
Figure 4
File Edit View Insert Tools Desktop Window Help
filter: box
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Figure 6
File Edit View Insert Tools Desktop Window Help
log fft magnitude of filtered image
9
8
7
6
5
4
3
2
1

Sampling

Why does a lower resolution image still make sense to us? What do we lose?

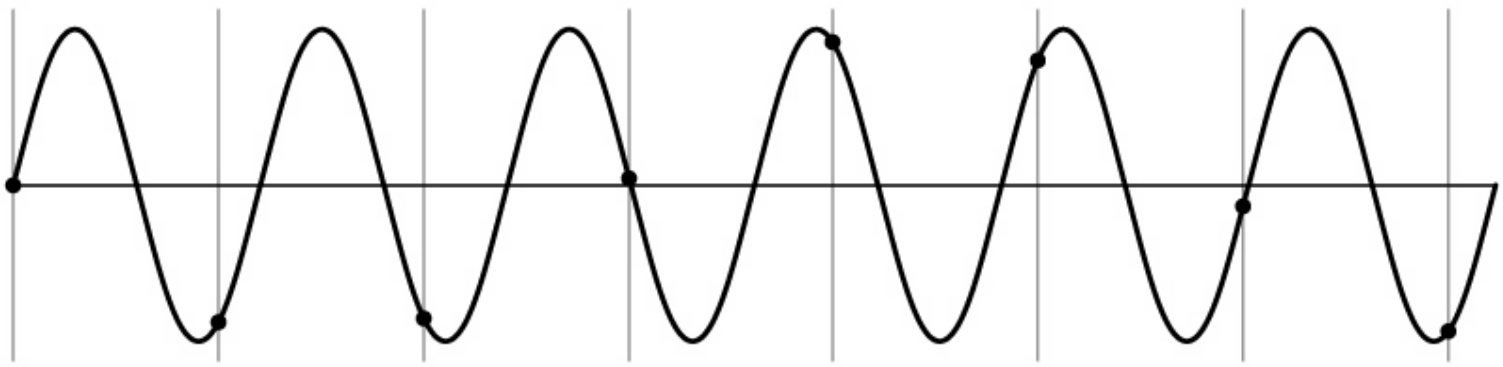
Image: http://www.flickr.com/photos/igorms/136916757/

Subsampling by a factor of 2

Throw away every other row and column to create a 1/2 size image

Aliasing problem

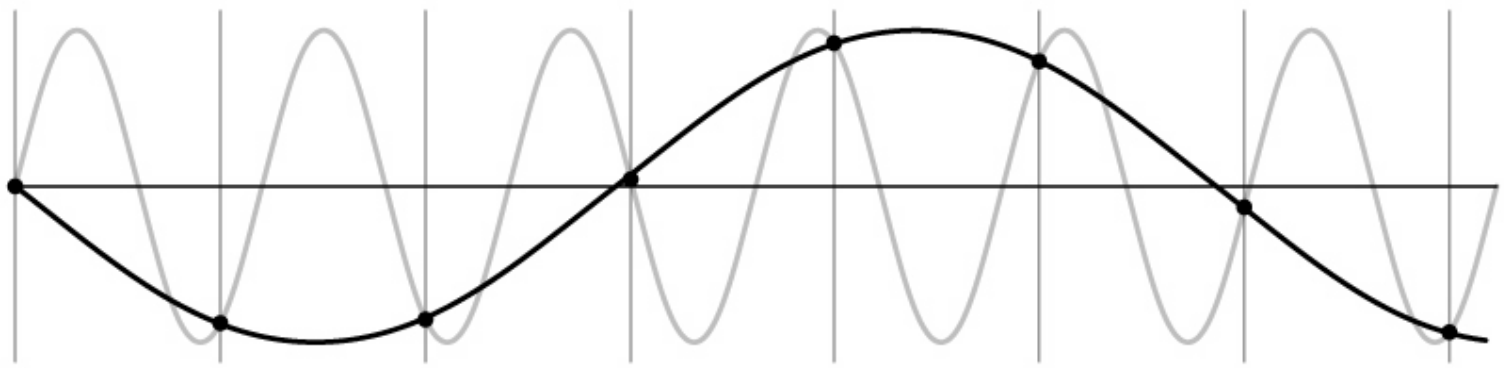
- 1D example (sinewave):



Source: S. Marschner

Aliasing problem

- 1D example (sinewave):



Source: S. Marschner

Aliasing problem

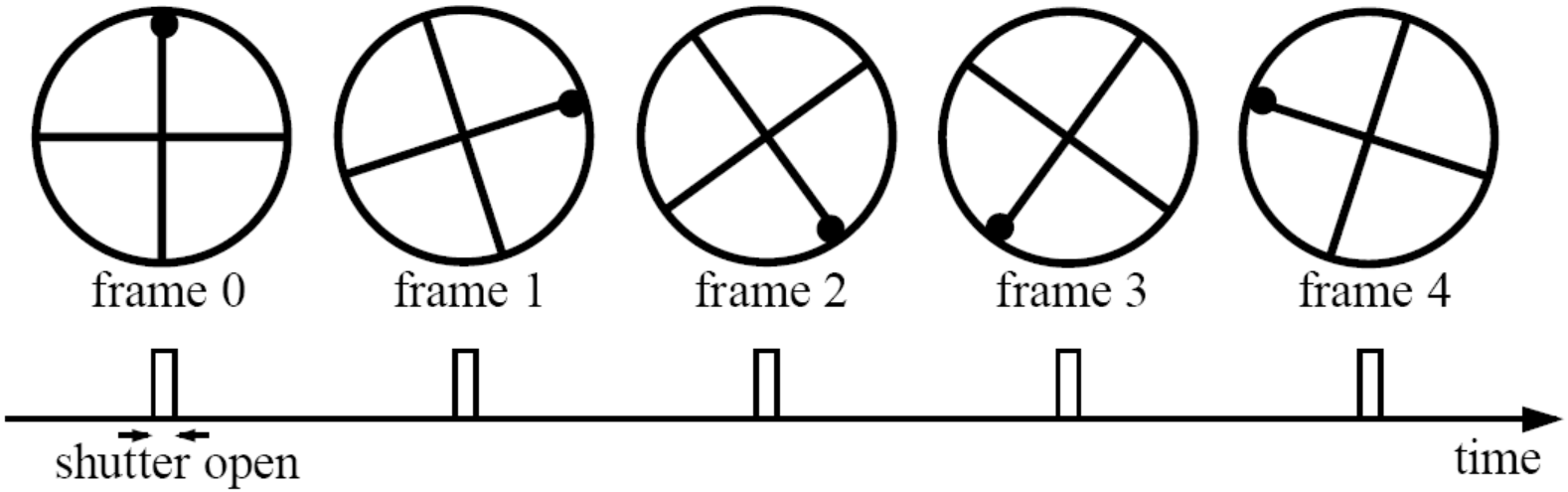
- Sub-sampling may be dangerous….
- Characteristic errors may appear:
  - “Wagon wheels rolling the wrong way in movies”
  - “Checkerboards disintegrate in ray tracing”
  - “Striped shirts look funny on color television”

Source: D. Forsyth

Aliasing in video

Imagine a spoked wheel moving to the right (rotating clockwise). Mark wheel with dot so we can see what's happening.

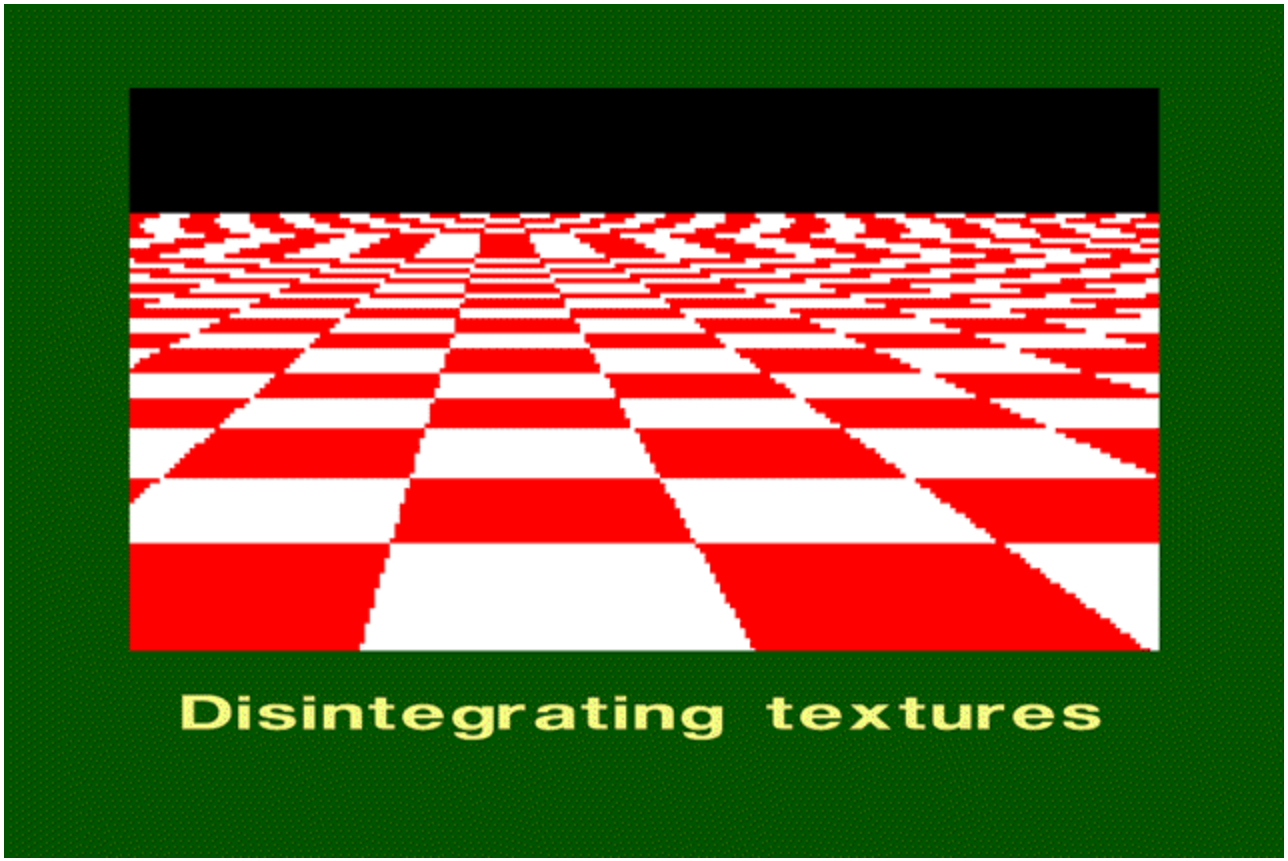
If camera shutter is only open for a fraction of a frame time (frame time = 1/30 sec. for video, 1/24 sec. for film):



Without dot, wheel appears to be rotating slowly backwards! (counterclockwise)

Slide by Steve Seitz

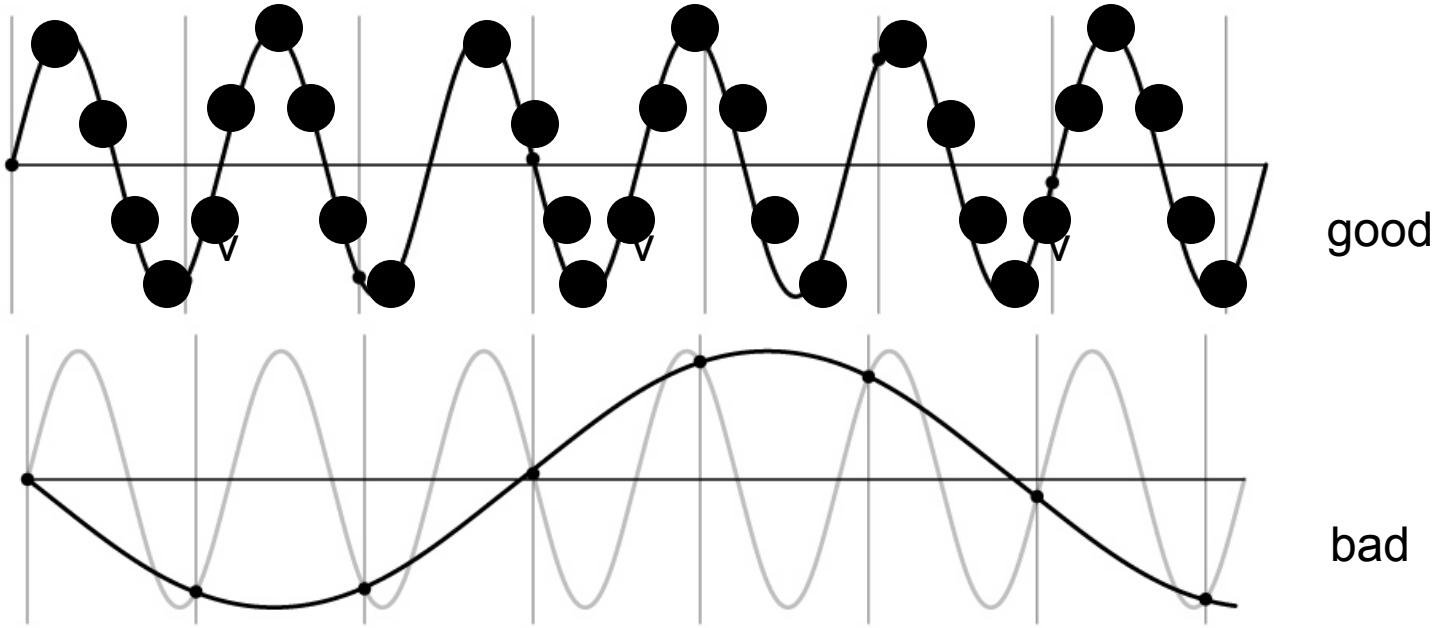
Aliasing in graphics



Source: A. Efros

Nyquist-Shannon Sampling Theorem

- When sampling a signal at discrete intervals, the sampling frequency must be $\geq 2 \times f_{max}$
- f_{max} = max frequency of the input signal
- This will allows to reconstruct the original perfectly from the sampled version



Source: D. Hoiem

Anti-aliasing

Solutions:

- Sample more often

- Get rid of all frequencies that are greater than half the new sampling frequency
  - Will lose information
  - But it's better than aliasing
  - Apply a smoothing filter

Source: D. Hoiem

Algorithm for downsampling by factor of 2

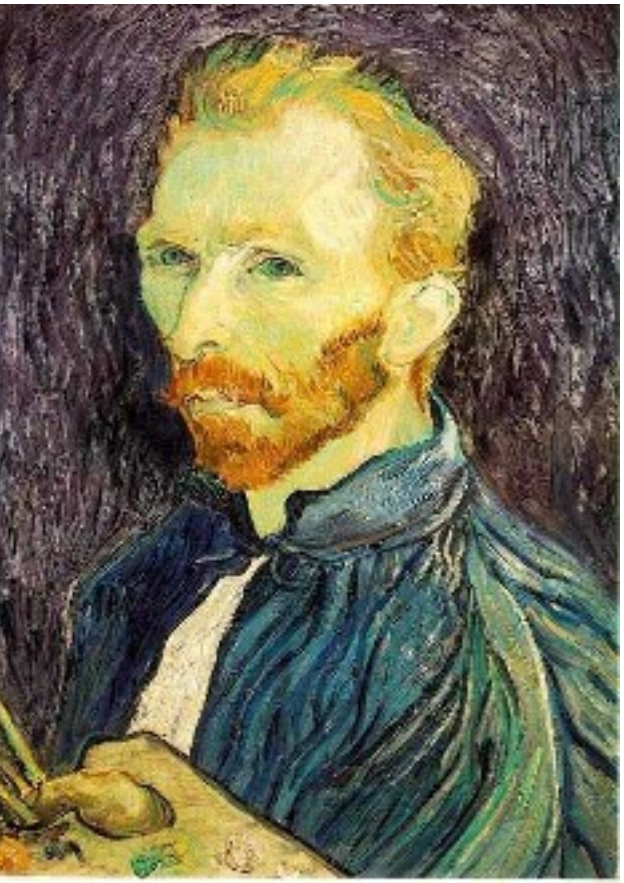
1. Start with image(h, w)
2. Apply low-pass filter

   im_blur = imfilter(image, fspecial('gaussian', 7, 1))
3. Sample every other pixel

   im_small = im_blur(1:2:end, 1:2:end);

Source: D. Hoiem

Subsampling without pre-filtering



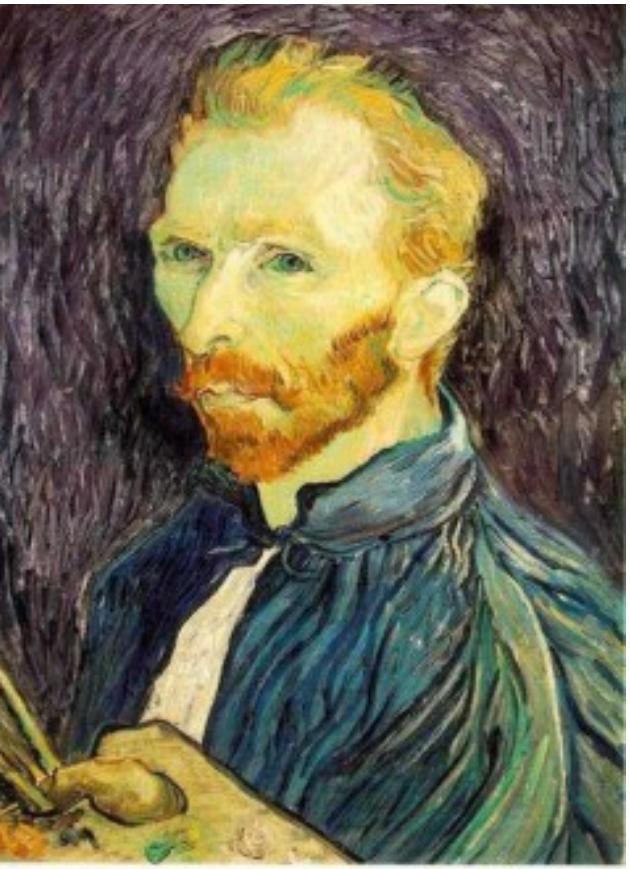
1/2

1/4 (2x zoom)

1/8 (4x zoom)

Slide by Steve Seitz

Subsampling with Gaussian pre-filtering



Gaussian 1/2

G 1/4

G 1/8

Slide by Steve Seitz

Why does a lower resolution image still make sense to us? What do we lose?

Image: http://www.flickr.com/photos/igorms/136916757/

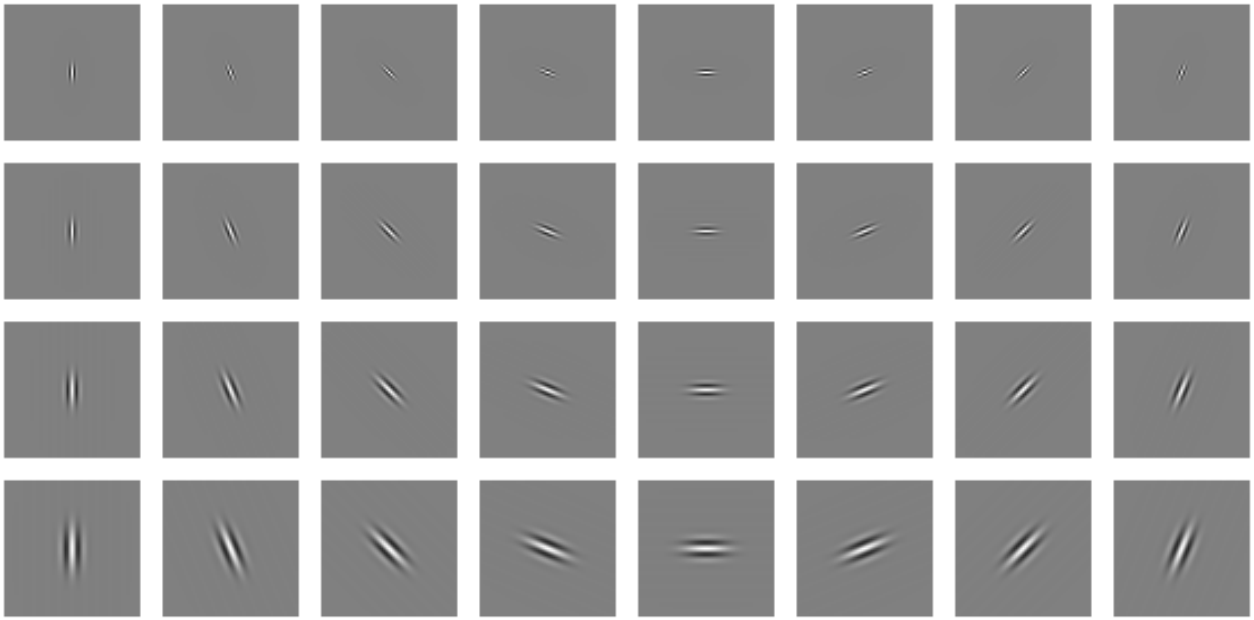
Why do we get different, distance-dependent interpretations of hybrid images?

?

Adapted from a slide by D. Hoiem

Clues from Human Perception

- Early processing in humans filters for various orientations and scales of frequency
- Perceptual cues in the mid-high frequencies dominate perception
- When we see an image from far away, we are effectively subsampling it

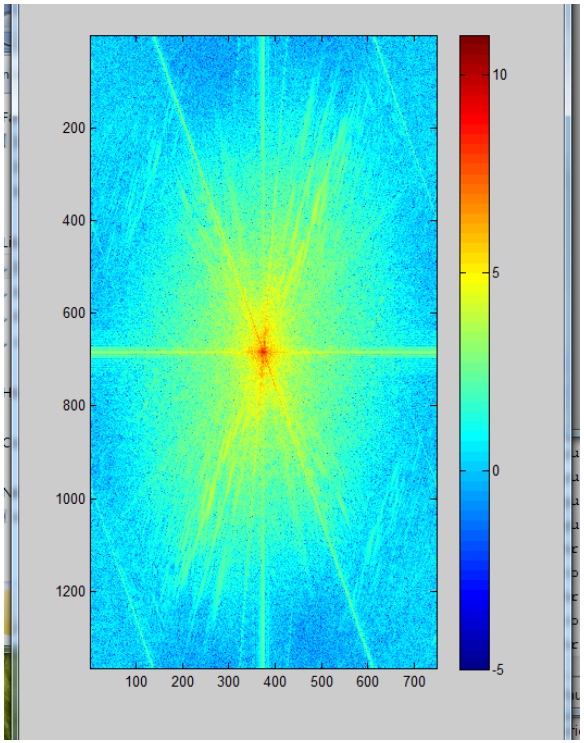


Early Visual Processing: Multi-scale edge and blob filters

Source: D. Hoiem

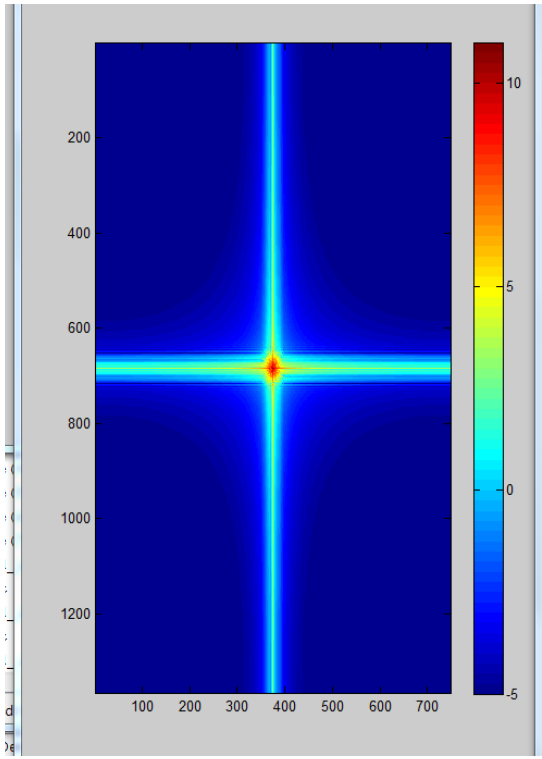
Hybrid Image in FFT

Hybrid Image



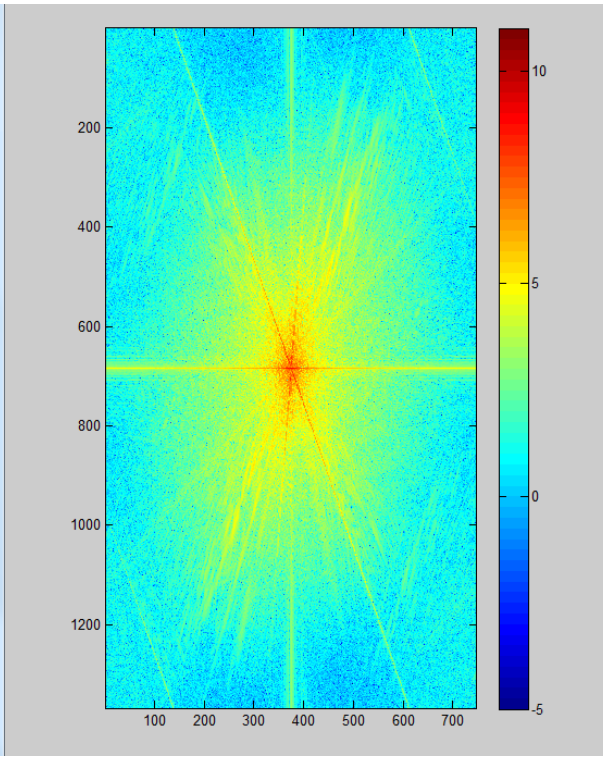
=

Low-passed Image



+

High-passed Image



Source: D. Hoiem

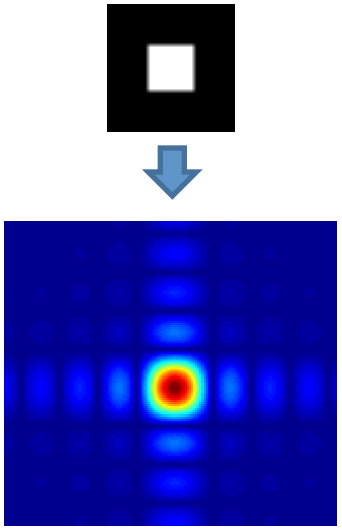
Perception

Why do we get different, distance-dependent interpretations of hybrid images?

?

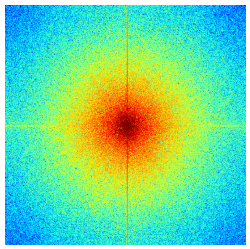
Adapted from a slide by D. Hoiem

Things to Remember



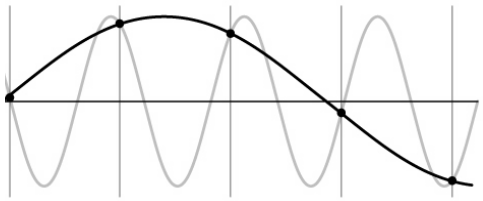
- Sometimes it makes sense to think of images and filtering in the frequency domain
  - Fourier analysis

- Can be faster to filter using FFT for large images (N logN vs. N^2 for auto-correlation)



- Images are mostly smooth
  - Basis for compression

- Remember to low-pass before sampling



Source: D. Hoiem

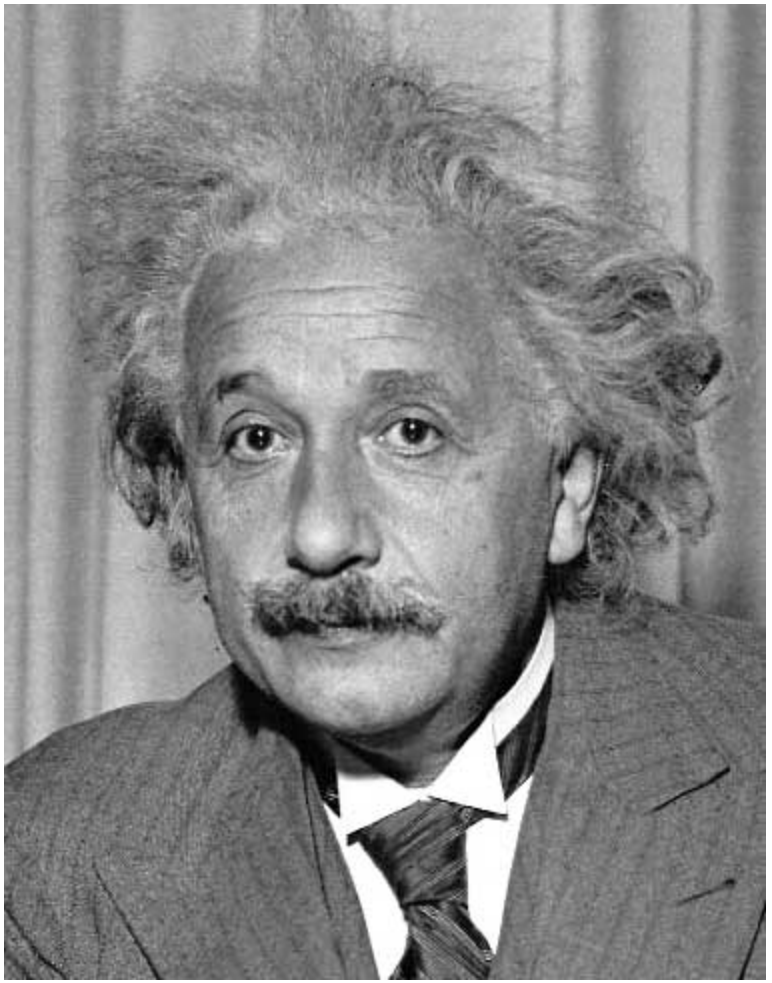
Next

- Template matching
- Image Pyramids

Template matching

- Goal: find  in image

- Main challenge: What is a good similarity or distance measure between two patches?
  - Correlation
  - Zero-mean correlation
  - Normalized Cross Correlation



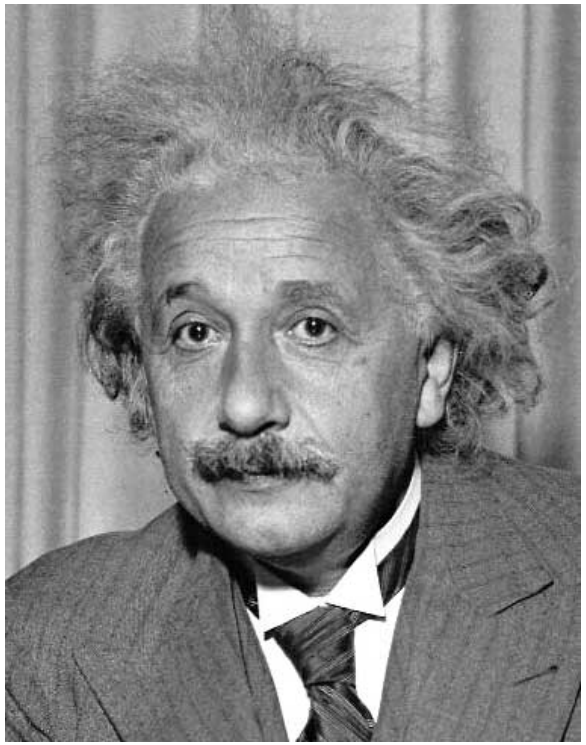
Source: D. Hoiem

Matching with filters

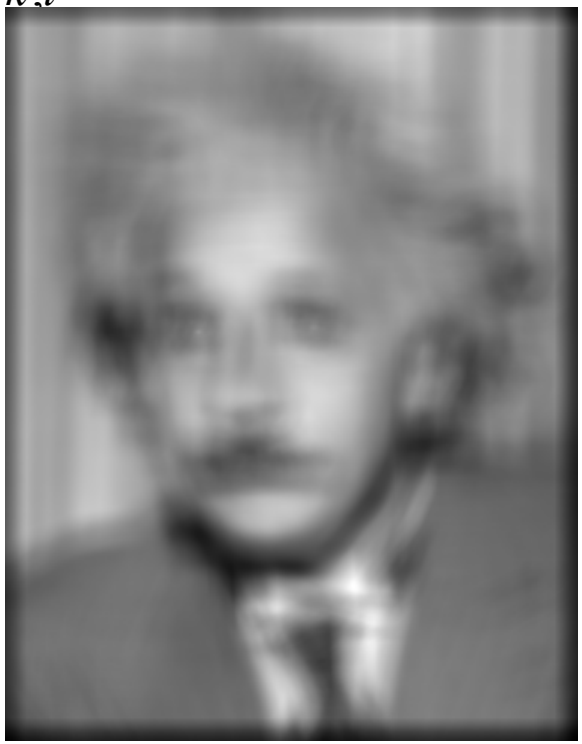
- Goal: find  in image
- Method 1: filter the image with eye patch

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

f = image
g = filter



Input



Filtered Image

What went wrong?

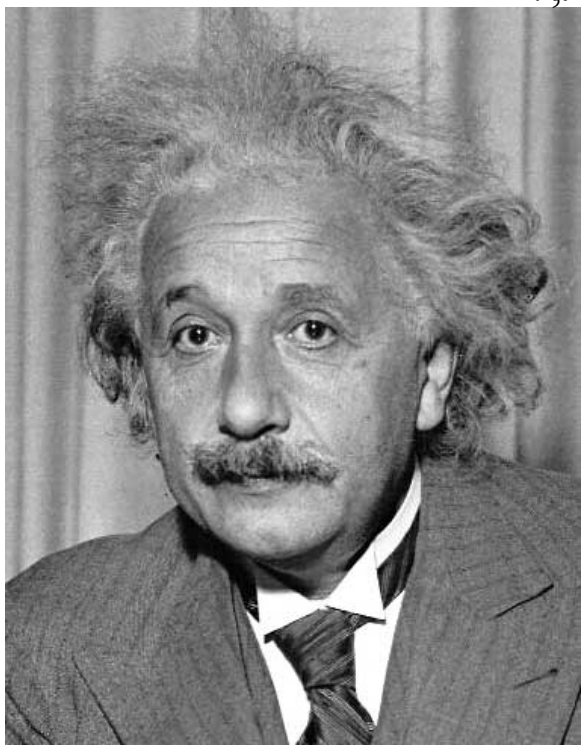
Source: D. Hoiem

Matching with filters

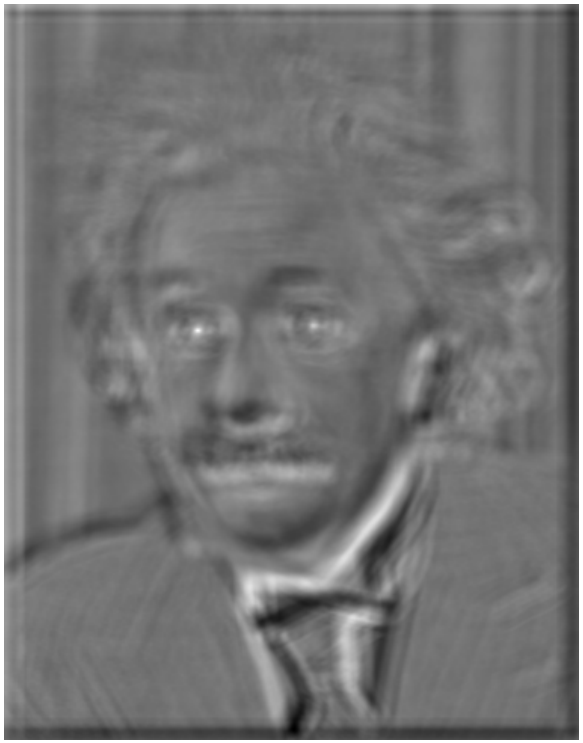
- Goal: find  in image
- Method 2: filter the image with zero-mean eye

$$h[m,n] = \sum_{k,l} (g[k,l] - \bar{g})\,(f[m+k, n+l])$$

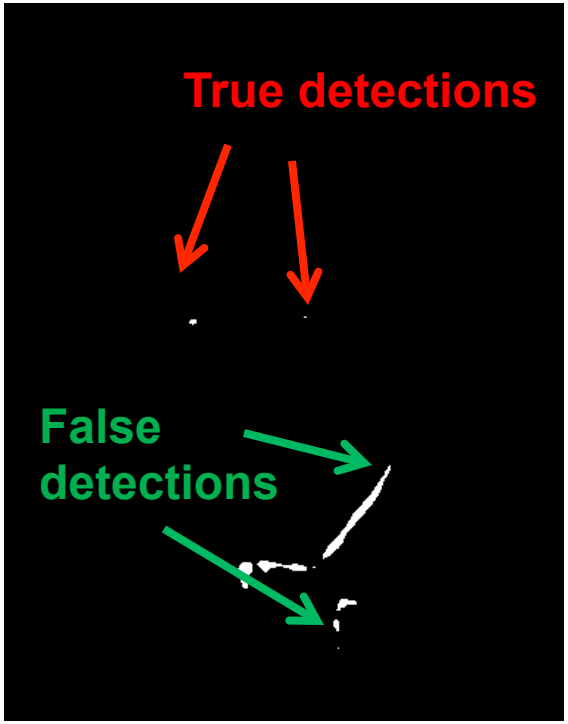
mean of template g



Input



Filtered Image (scaled)



Thresholded Image

Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation

mean template

mean image patch

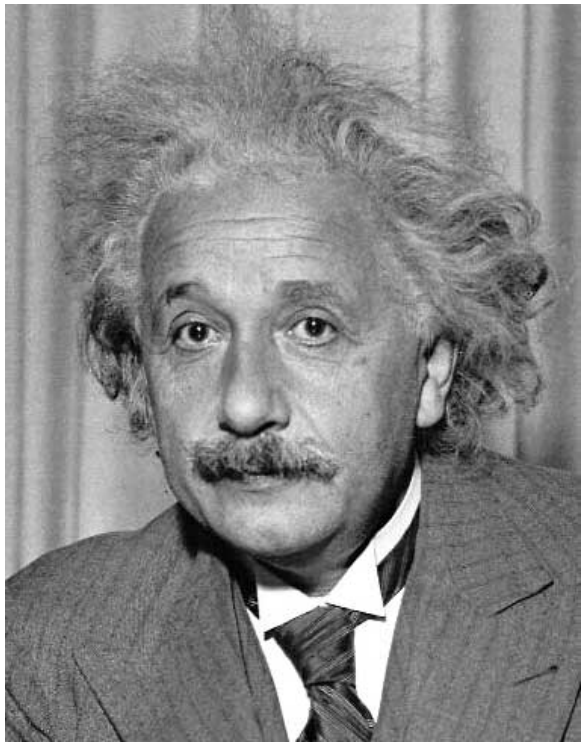
$$h[m,n] = \frac{\sum_{k,l}(g[k,l]-\bar{g})(f[m+k,n+l]-\bar{f}_{m,n})}{\left(\sum_{k,l}(g[k,l]-\bar{g})^2 \sum_{k,l}(f[m+k,n+l]-\bar{f}_{m,n})^2\right)^{0.5}}$$

Matlab: `normxcorr2(template, im)`

Source: D. Hoiem

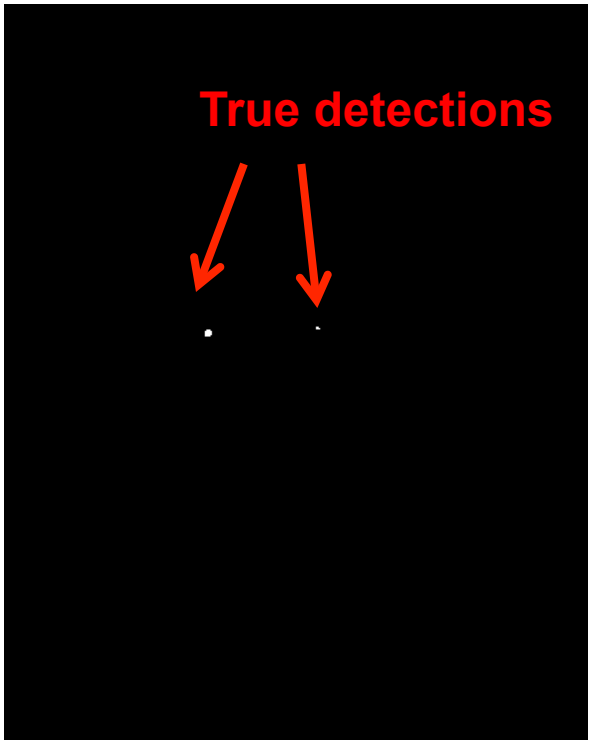
Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

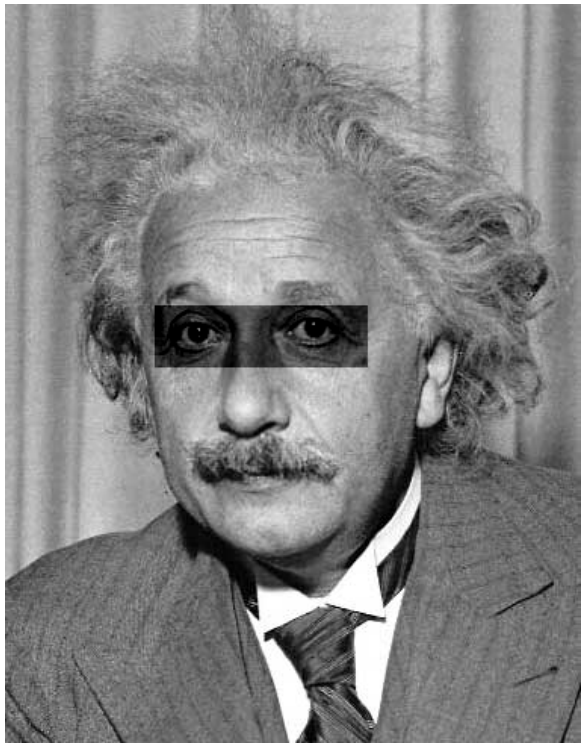
Normalized X-Correlation



Thresholded Image

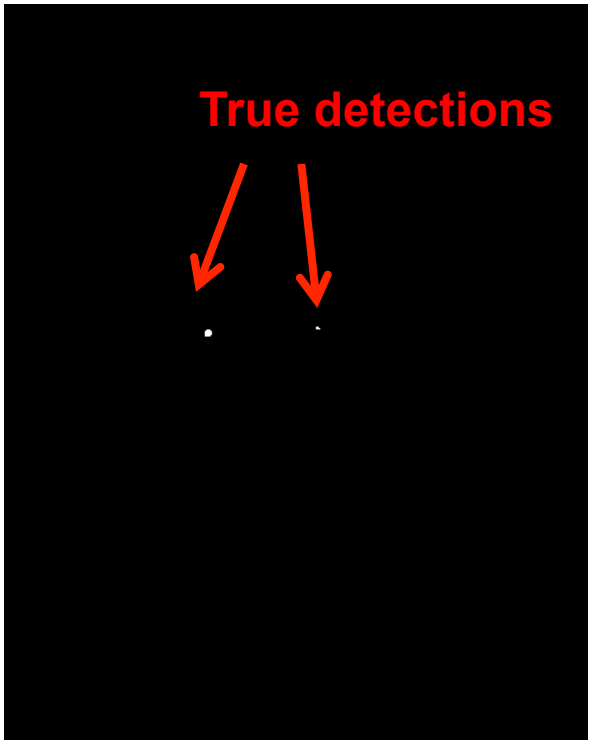
Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

Normalized X-Correlation



Thresholded Image

Q: What is the best method to use?

A: Depends

- Zero-mean filter: fast but not a great matcher
- Normalized cross-correlation: slow but invariant to local average intensity and contrast

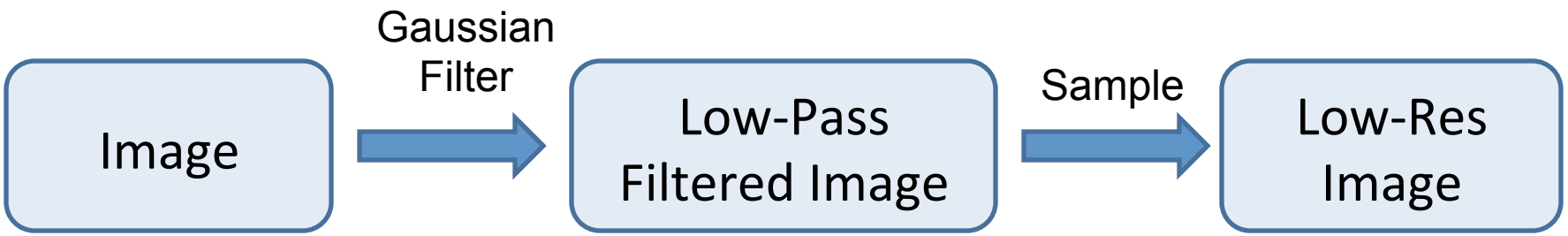
Source: D. Hoiem

Q: What if we want to find larger or smaller eyes?

A: Image Pyramid

Source: D. Hoiem

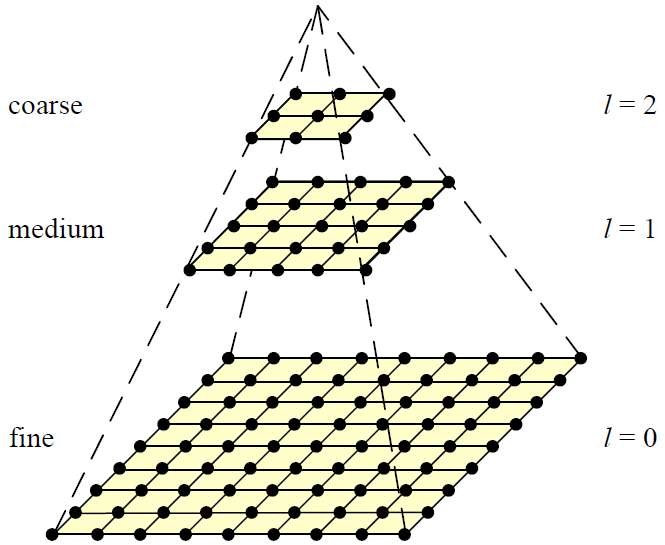
Review of Sampling

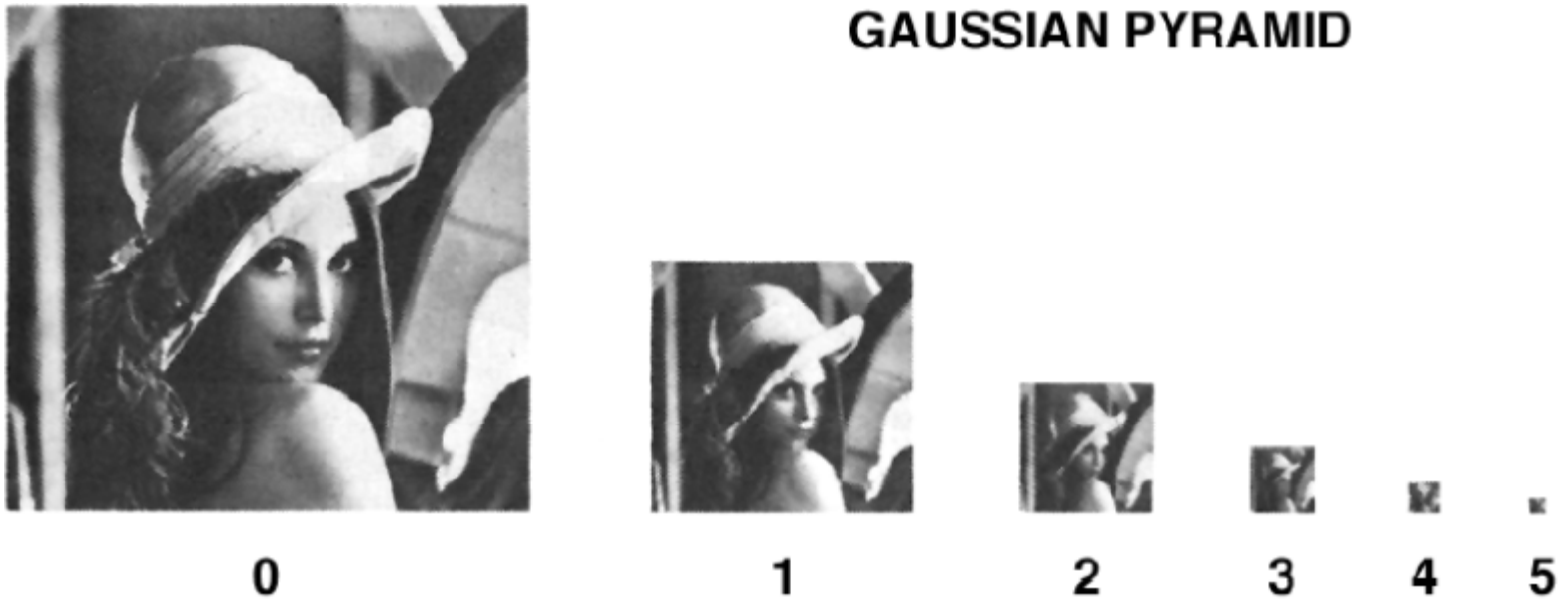


Source: D. Hoiem

Image pyramids

- Image pyramids are multi-resolution image representations.
- Repeated decimations with a Gaussian low-pass filter give a Gaussian pyramid:





Template Matching with Image Pyramids

Input: Image, Template

1. Match template at current scale

2. Downsample image
   - In practice, scale step of 1.1 to 1.2

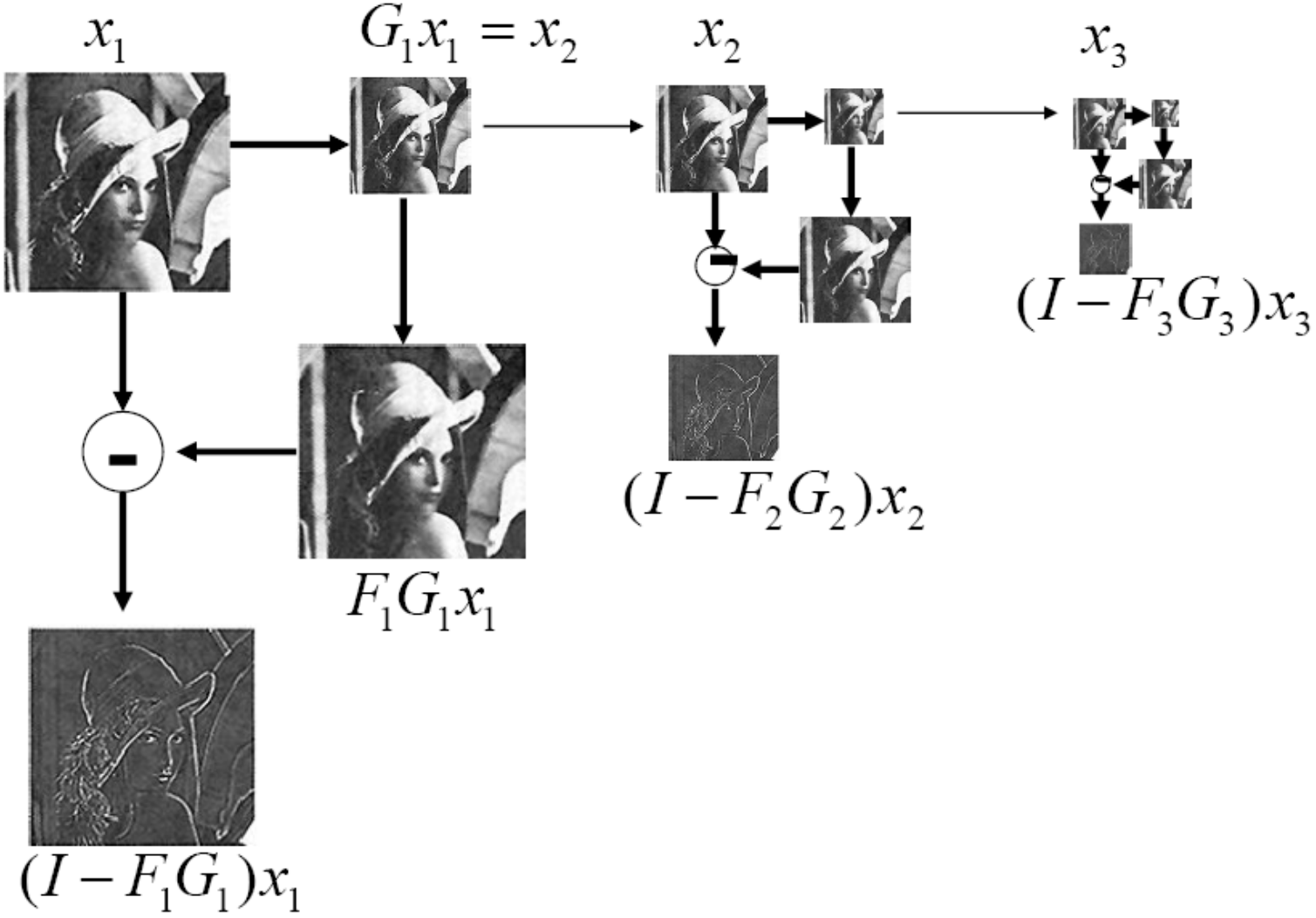
3. Repeat 1-2 until image is very small

4. Take responses above some threshold, perhaps with non-maxima suppression

Source: D. Hoiem

Laplacian pyramid

- Contains the difference images between two successive Gaussian pyramid levels:



Showing, at full resolution, the information captured at each level of a Gaussian (top) and Laplacian (bottom) pyramid.

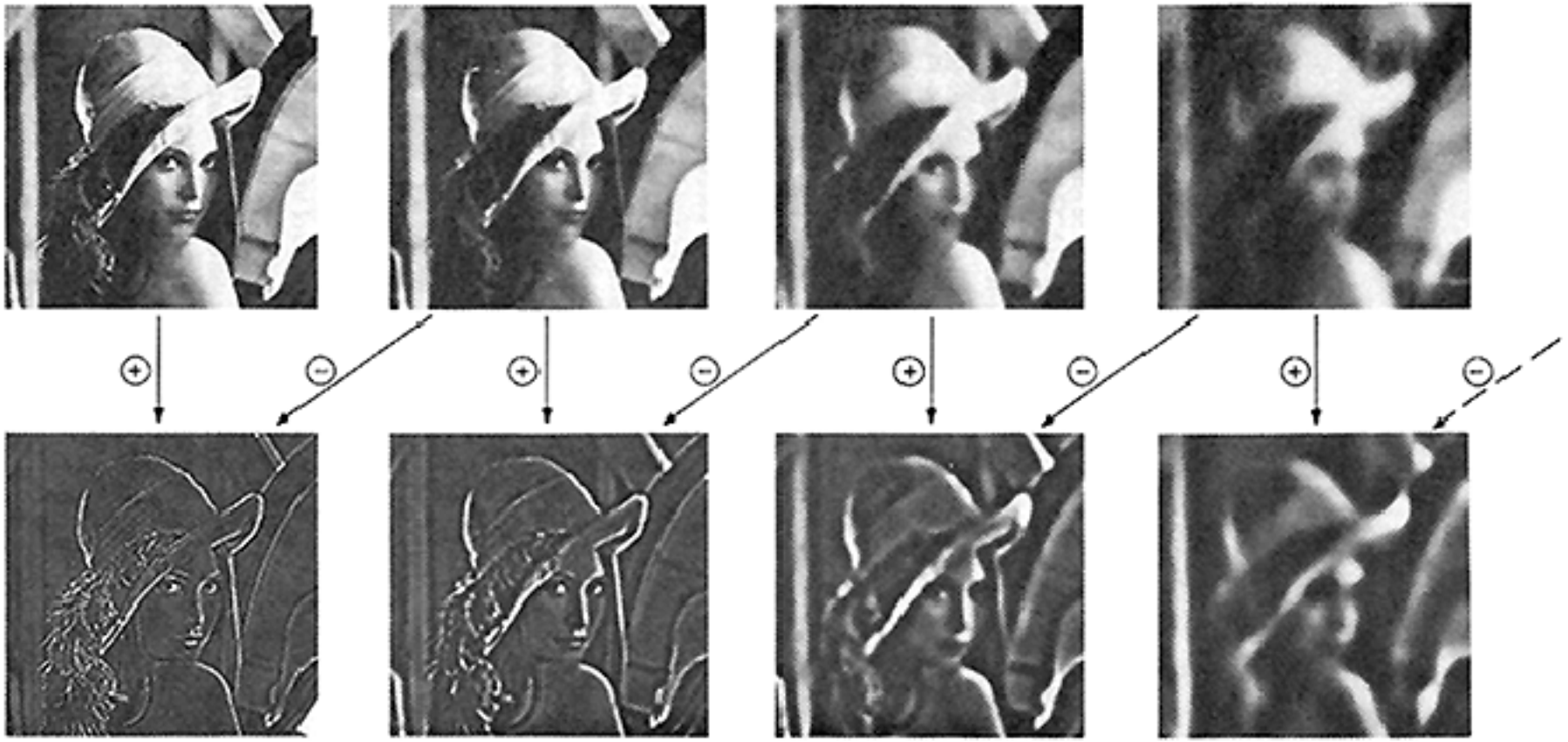


Fig 5. First four levels of the Gaussian and Laplacian pyramid. Gaussian images, upper row, were obtained by expanding pyramid arrays (Fig. 4) through Gaussian interpolation. Each level of the Laplacian pyramid is the difference between the corresponding and next higher levels of the Gaussian pyramid.

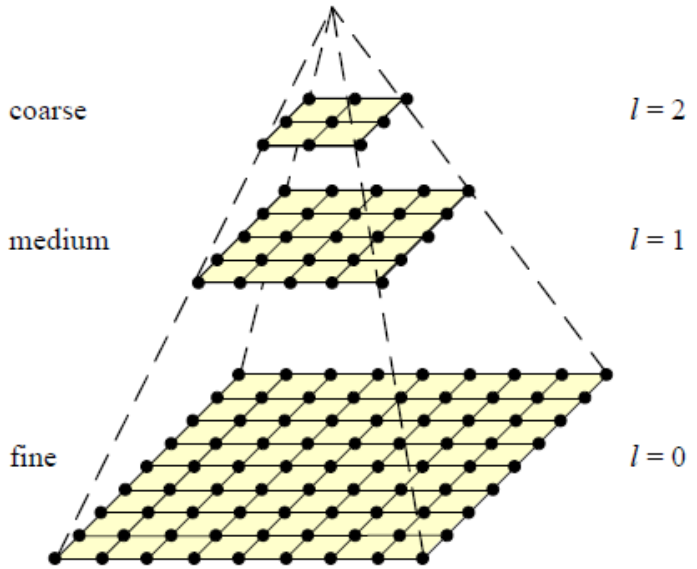
Major uses of image pyramids

- Compression
- Object detection
  - Scale search
  - Features
- Detecting stable interest points
- Registration
  - Coarse-to-fine

Source: D. Hoiem

Coarse-to-fine Image Registration

1. Compute Gaussian pyramid
2. Align with coarse pyramid
3. Successively align with finer pyramids
   - Search smaller range



Why is this faster?

Are we guaranteed to get the same result?

Source: D. Hoiem