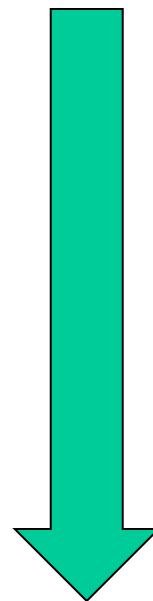


Image Processing



points, lines,
planes, transforms,
irradiance,
projections, ...



Pixels!
uint8 [0..255]

Image as a matrix (tensor) or numbers

row **column** → **R** **G** **B**

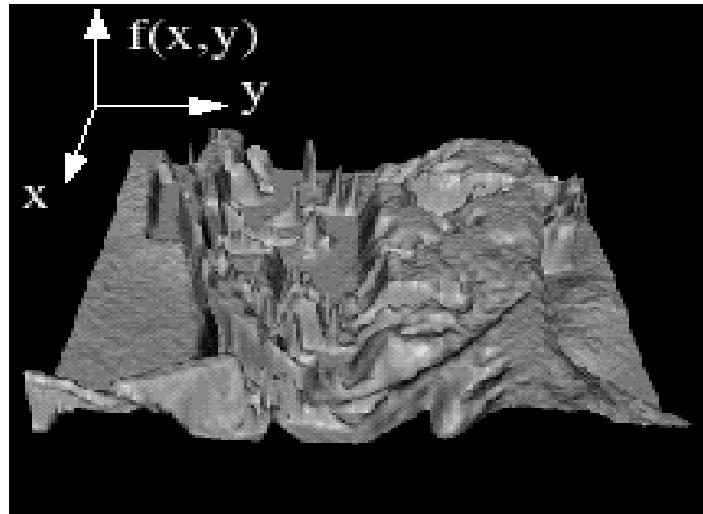
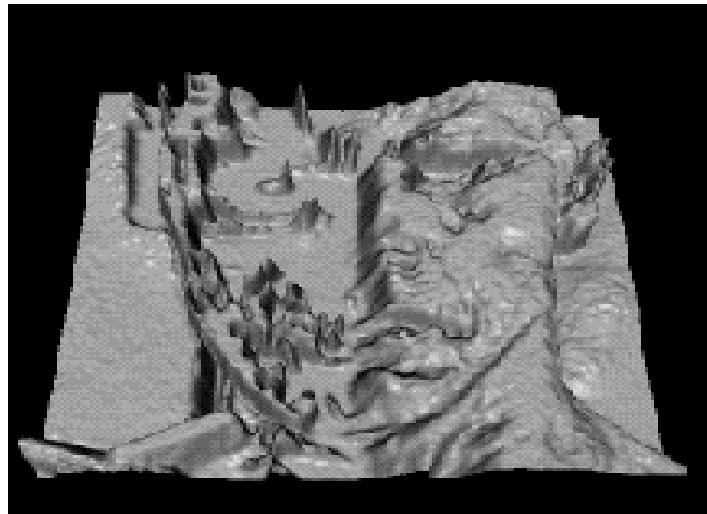
0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99	
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91	
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92	
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95	
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85	
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33	
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74	
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93	
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99	
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97	
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93	
	0.65	0.75	0.58	0.68	0.45	0.42	0.77	0.75	0.71	0.90	0.99
	0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
	0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93
	0.65	0.75	0.58	0.68	0.45	0.42	0.77	0.75	0.71	0.90	0.99
	0.79	0.73	0.90	0.67	0.33	0.33	0.61	0.69	0.79	0.73	0.93
	0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Image as a function

We can think of an **image** as a function, f , from \mathbb{R}^2 to \mathbb{R} :

- $f(x, y)$ gives the **intensity** at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
 - $f: [a,b] \times [c,d] \rightarrow [0,1]$

Image as a function



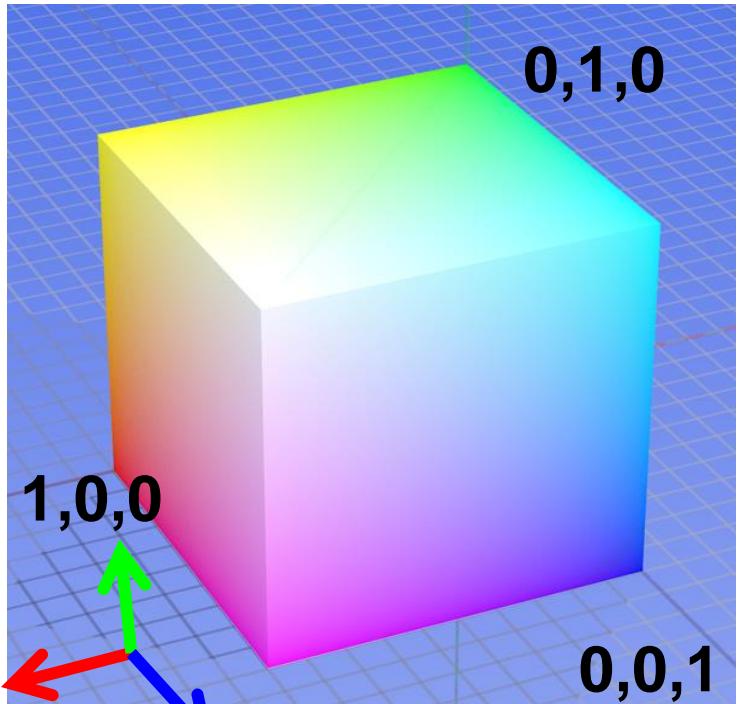
Color image as a function

A color image is just three functions pasted together.
We can write this as a “vector-valued” function:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

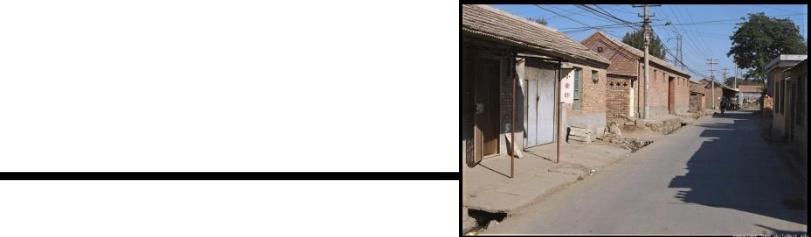
Color spaces: RGB

Default color space



RGB cube

- Easy for devices
- But not perceptual
- Where do the grays live?
- Where is hue and saturation?



R
(G=0,B=0)

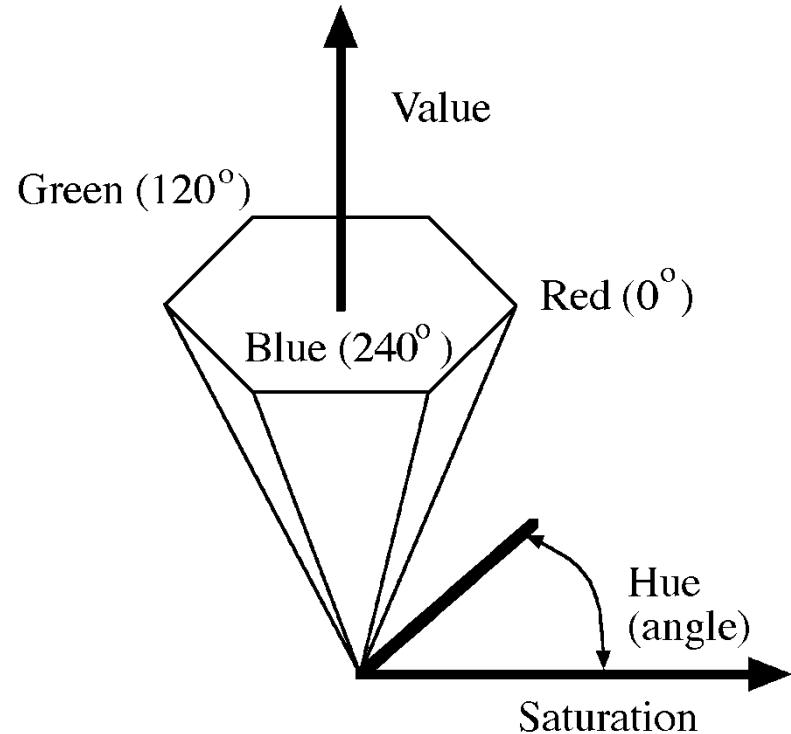
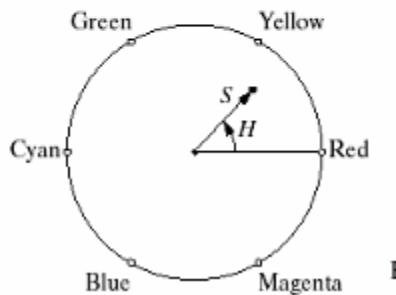
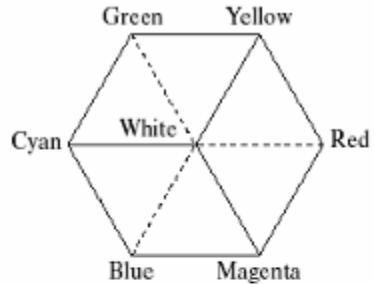


G
(R=0,B=0)



B
(R=0,G=0)

HSV



Hue, Saturation, Value (Intensity)

- RGB cube on its vertex

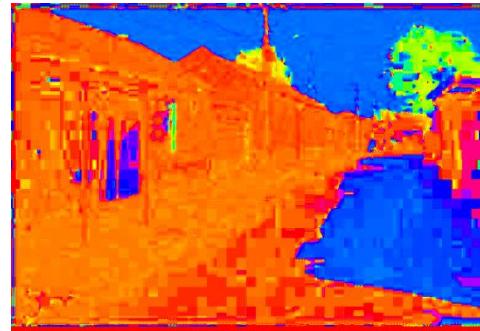
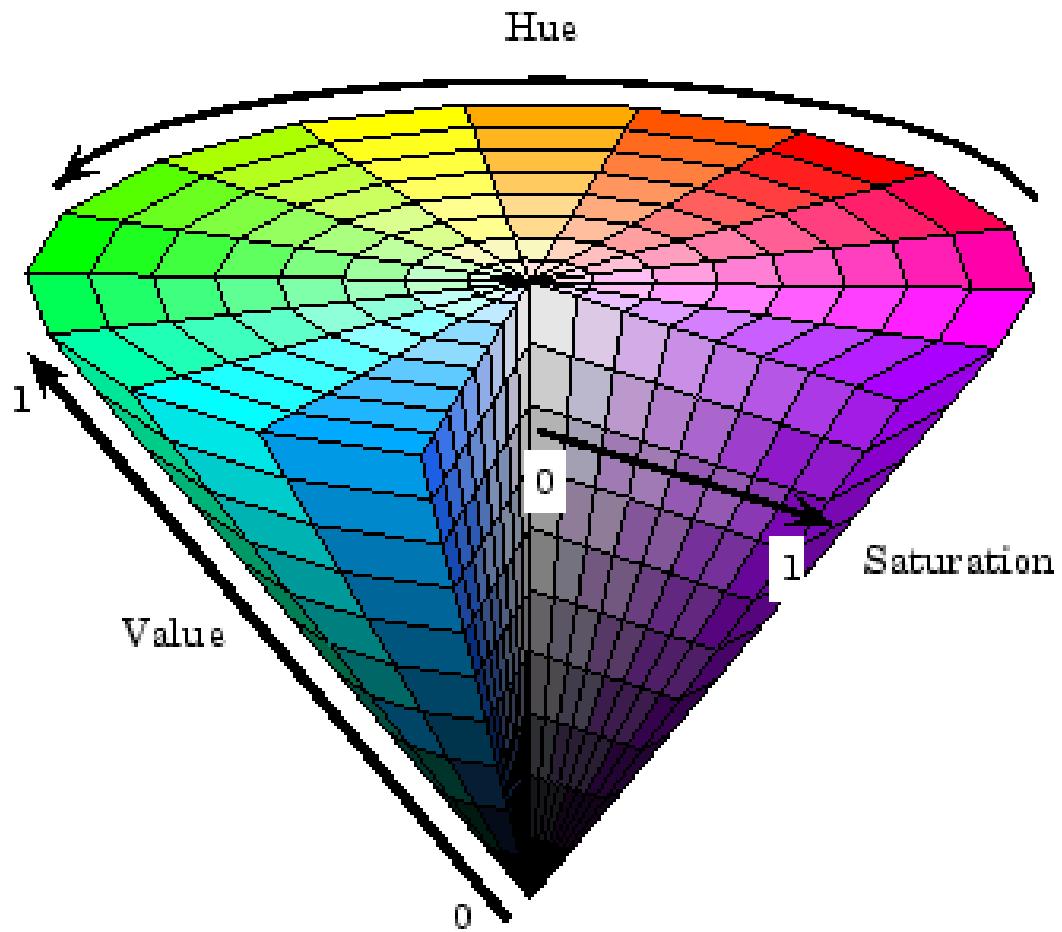
Decouples the three components (a bit)

Use `rgb2HSV()` and `HSV2RGB()` in Matlab / Python

Color spaces: HSV



Intuitive color space



H
($S=1, V=1$)



S
($H=1, V=1$)

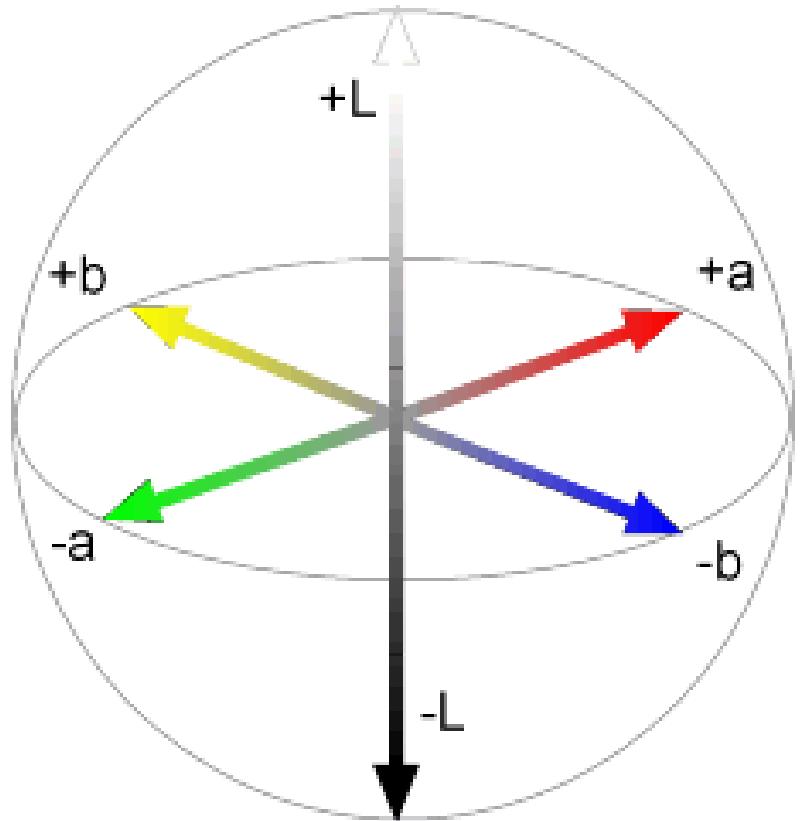


V
($H=1, S=0$)

Color spaces: L*a*b*



“Perceptually uniform”* color space



L
($a=0, b=0$)



a
($L=65, b=0$)



b
($L=65, a=0$)

What's in the “pixel intensity”?

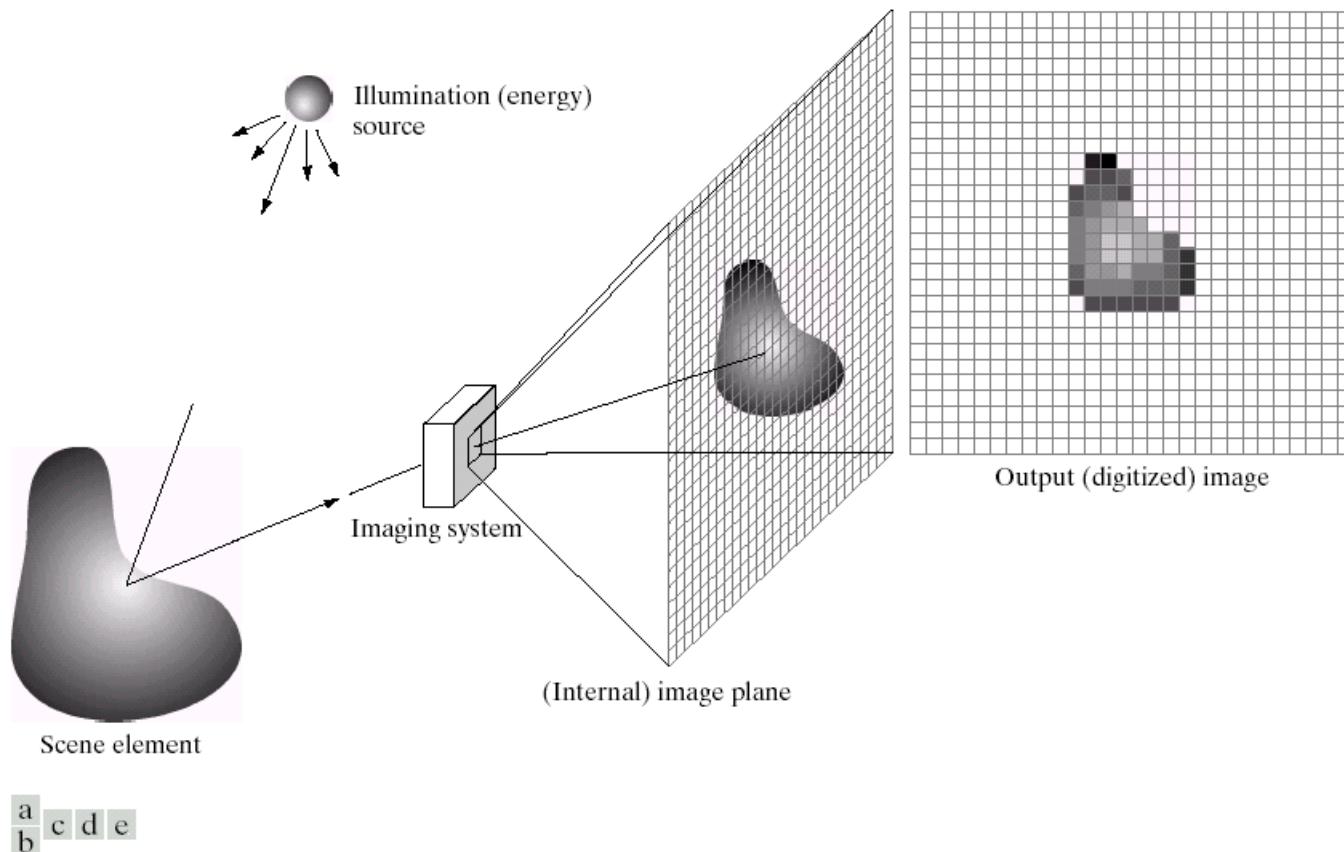


FIGURE 2.15 An example of the digital image acquisition process. (a) Energy (“illumination”) source. (b) An element of a scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

$$f(x,y) = \text{reflectance}(x,y) * \text{illumination}(x,y)$$

Reflectance in $[0, 1]$, illumination in $[0, \infty]$

Problem: Dynamic Range



1



1500



25,000



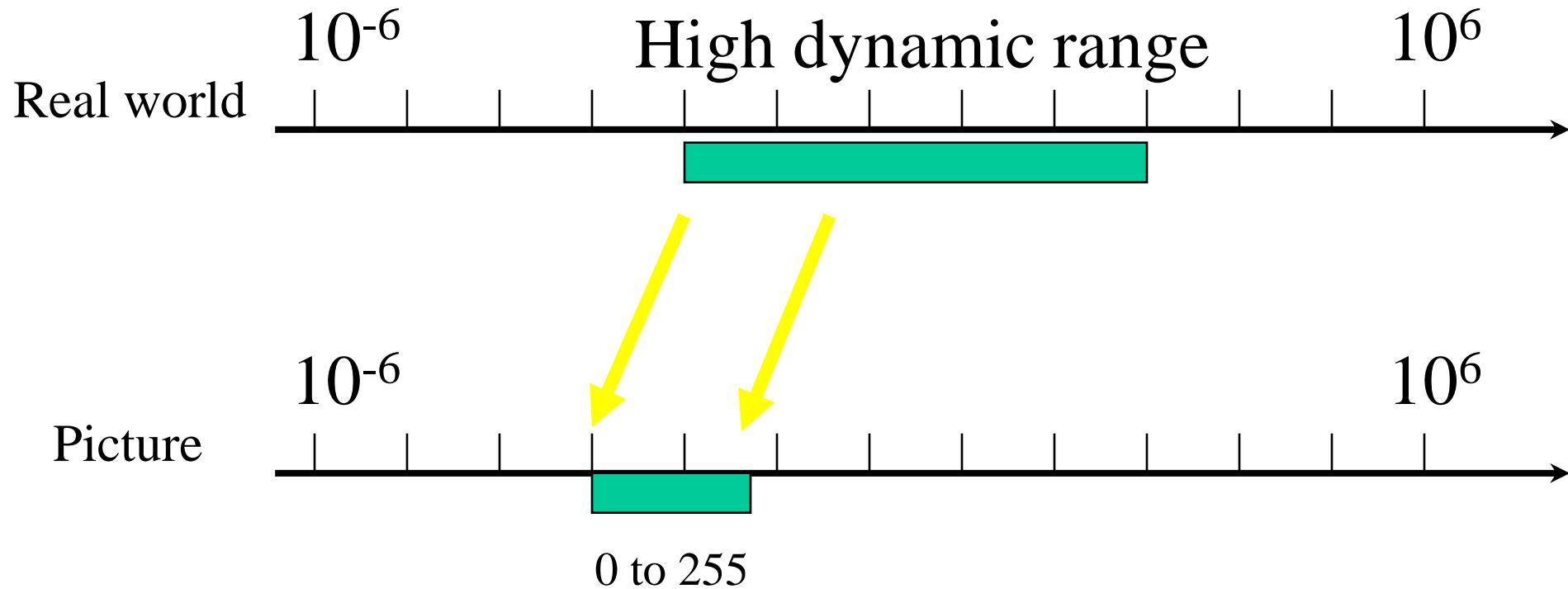
400,000



2,000,000,000

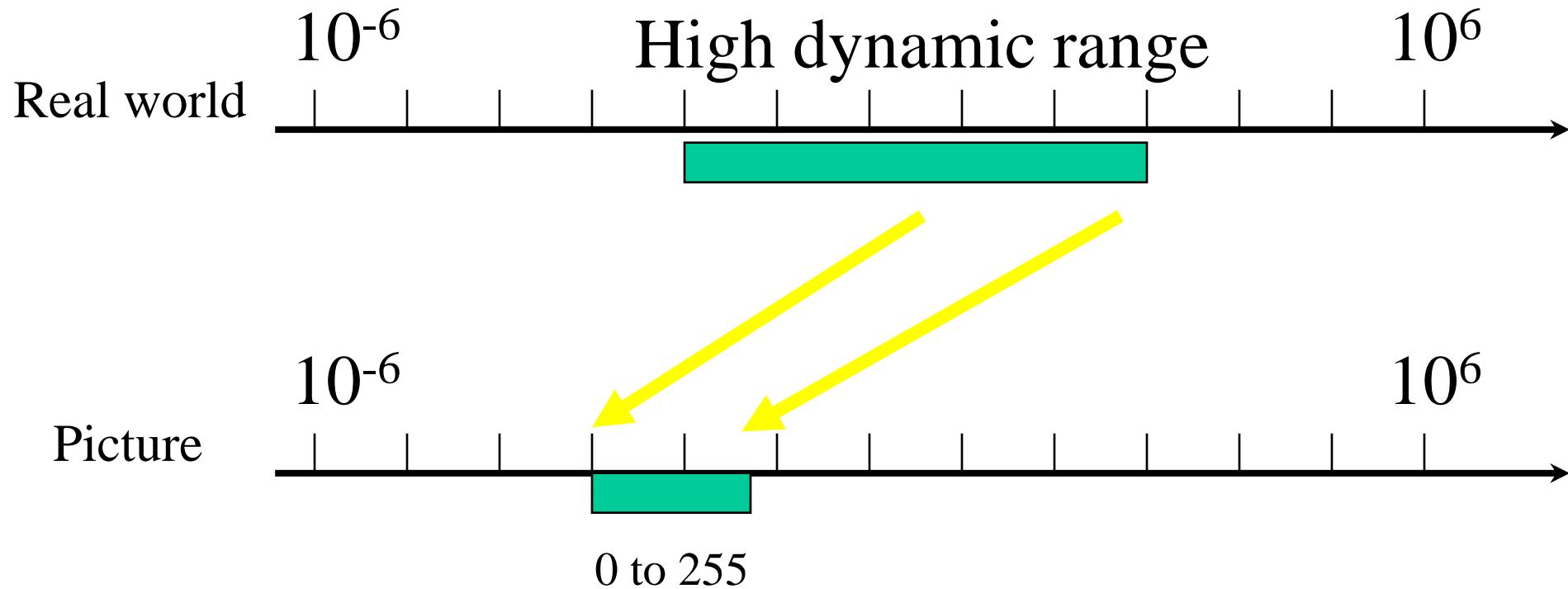
The real world is
High dynamic range

Long Exposure



- What does pixel value 255 mean?

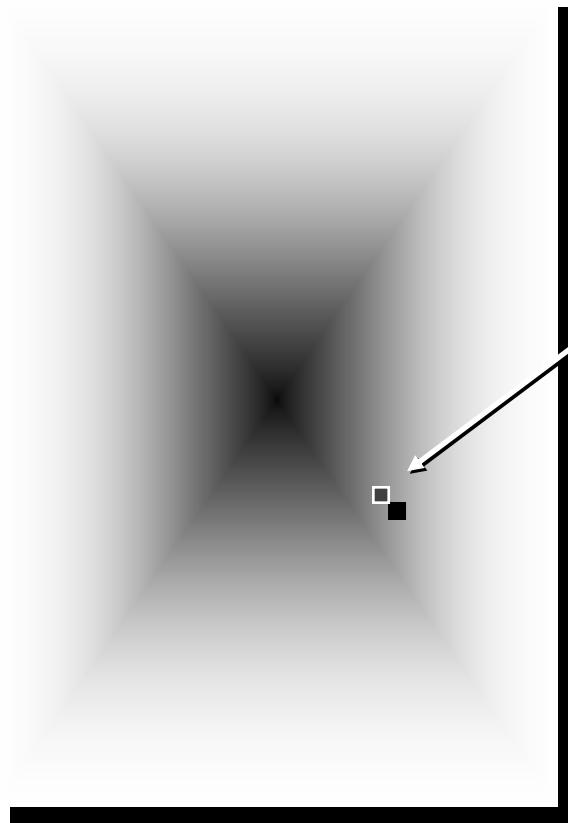
Short Exposure



- What does pixel value 0 mean?

Is Camera a photometer?

Image



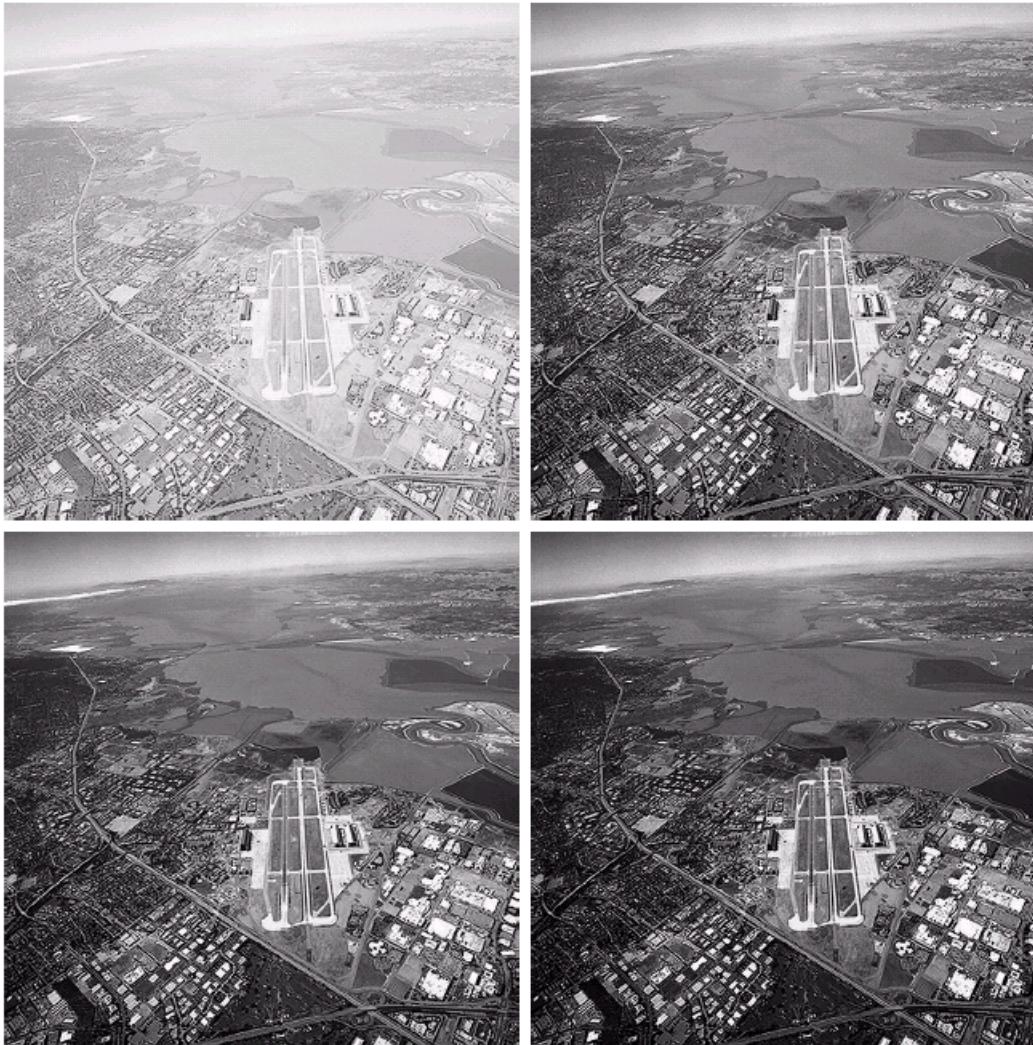
pixel (312, 284) = 42

42 photos?

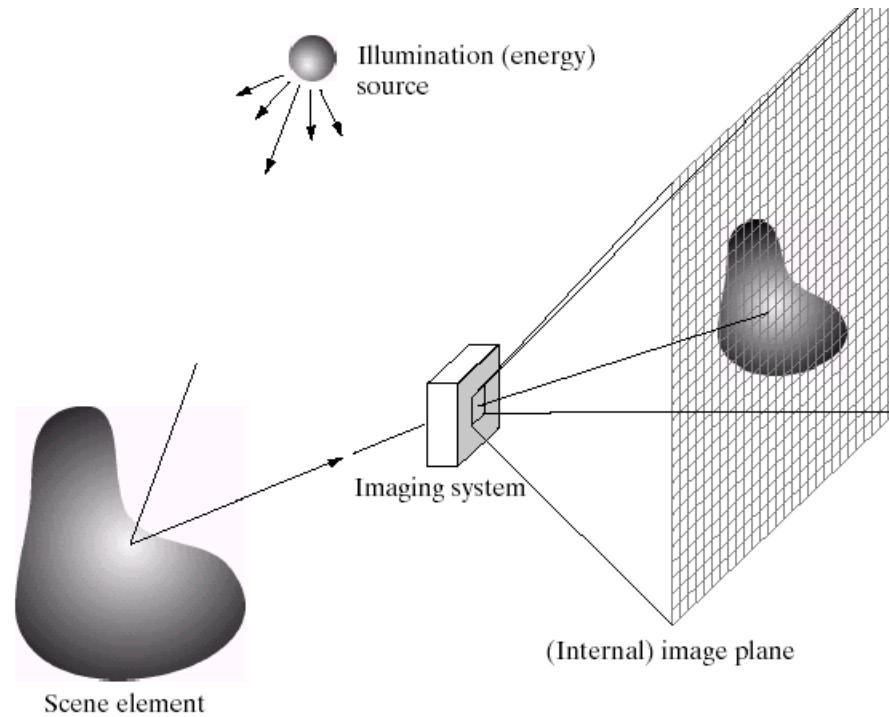
Don't trust the pixel values

a b
c d

FIGURE 3.9
(a) Aerial image.
(b)–(d) Results of
applying the
transformation in
Eq. (3.2-3) with
 $c = 1$ and
 $\gamma = 3.0, 4.0,$ and
 $5.0,$ respectively.
(Original image
for this example
courtesy of
NASA.)



Sampling and Quantization



Sampling and Quantization

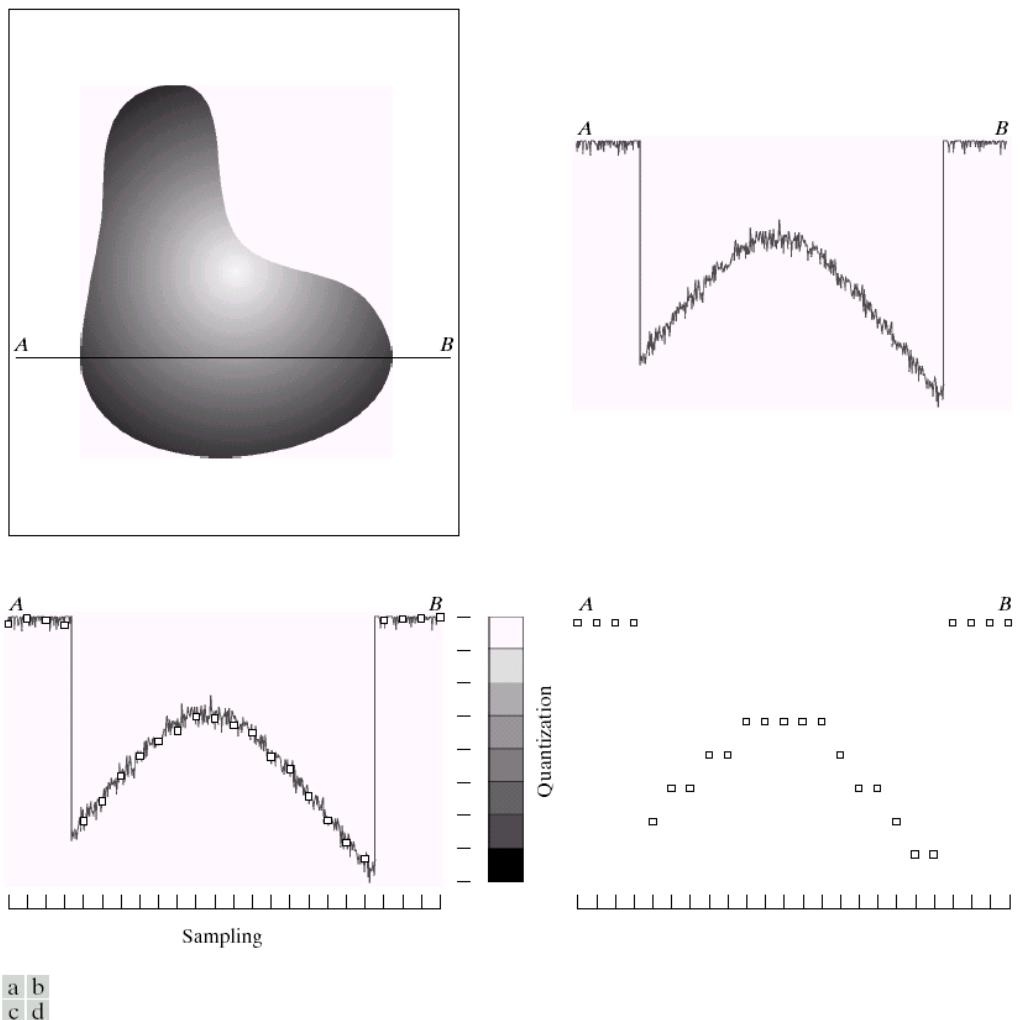
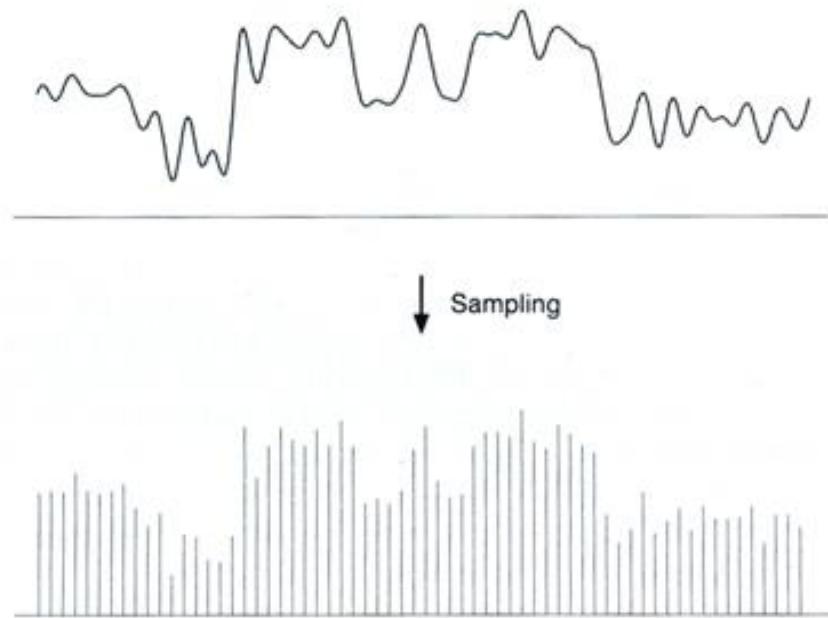


FIGURE 2.16 Generating a digital image. (a) Continuous image. (b) A scan line from A to B in the continuous image, used to illustrate the concepts of sampling and quantization. (c) Sampling and quantization. (d) Digital scan line.

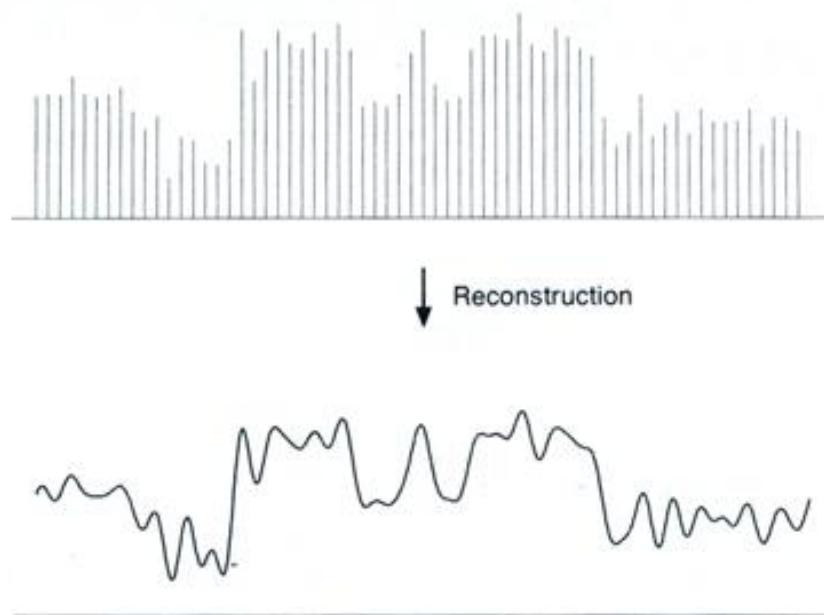
Sampled representations

- How to store and compute with continuous functions?
- Common scheme for representation: samples
 - write down the function's values at many points

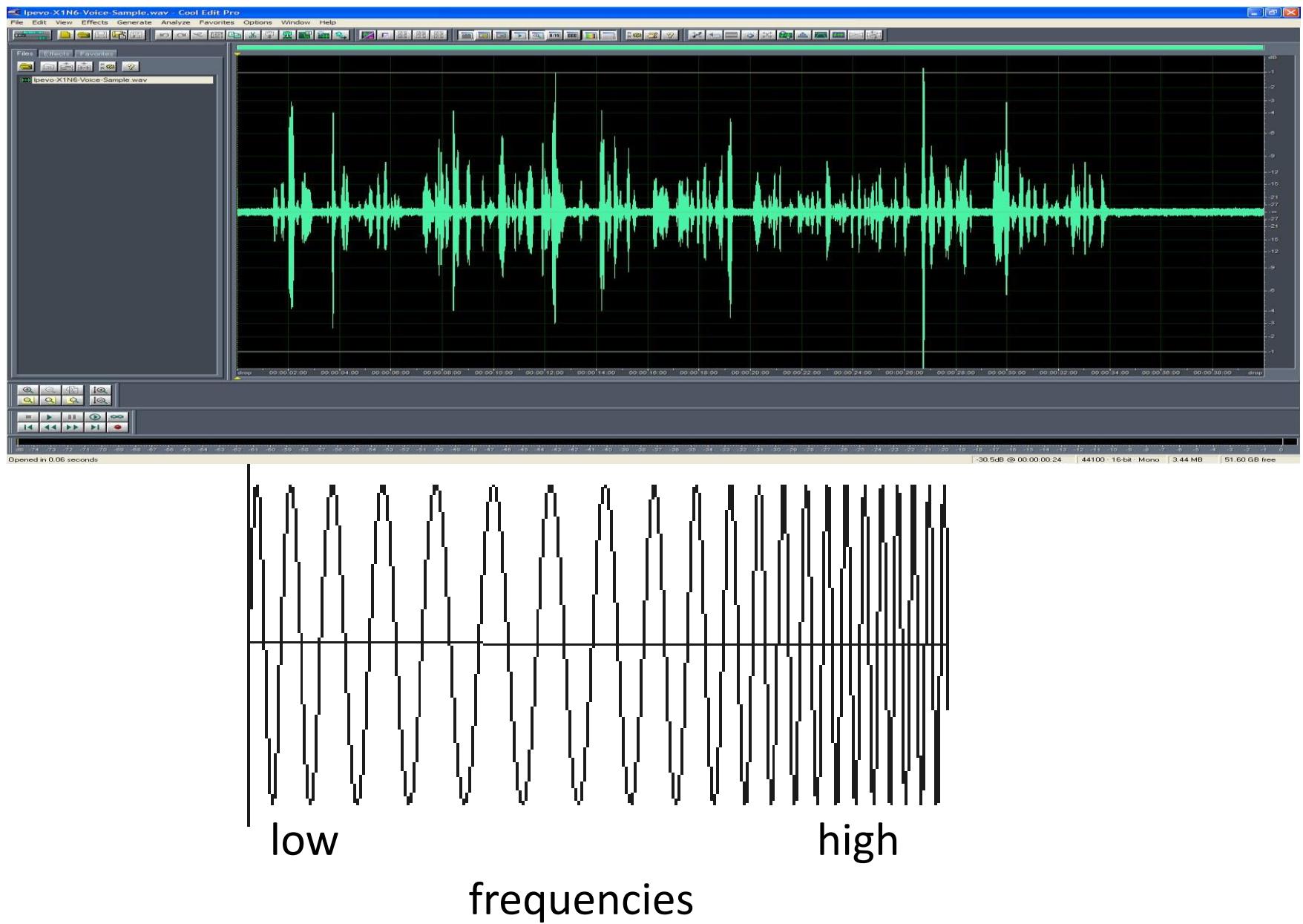


Reconstruction

- Making samples back into a continuous function
 - for output (need realizable method)
 - for analysis or processing (need mathematical method)
 - amounts to “guessing” what the function did in between

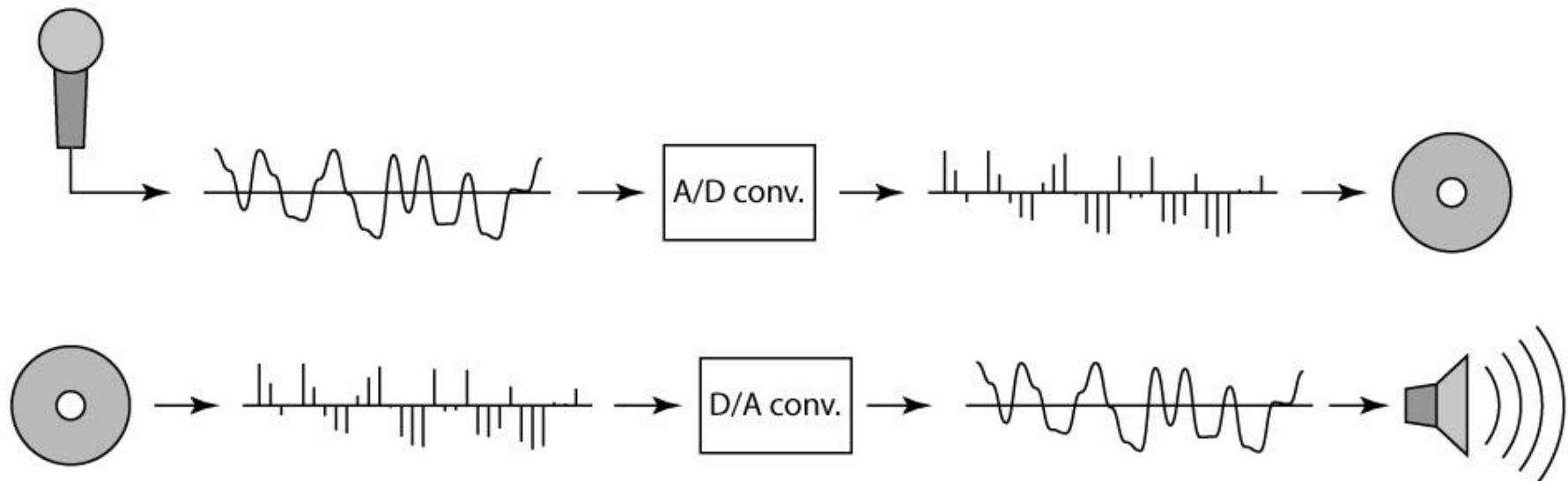


1D Example: Audio



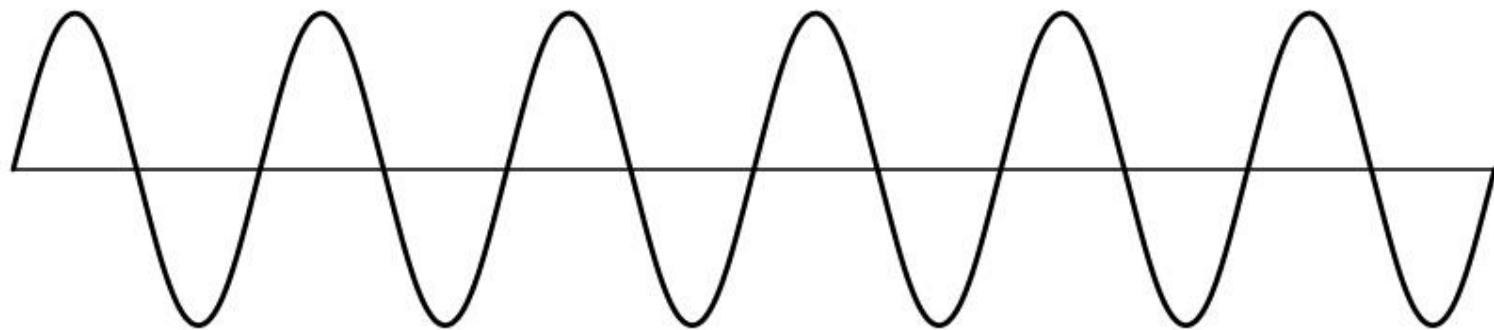
Sampling in digital audio

- Recording: sound to analog to samples to disc
- Playback: disc to samples to analog to sound again
 - how can we be sure we are filling in the gaps correctly?



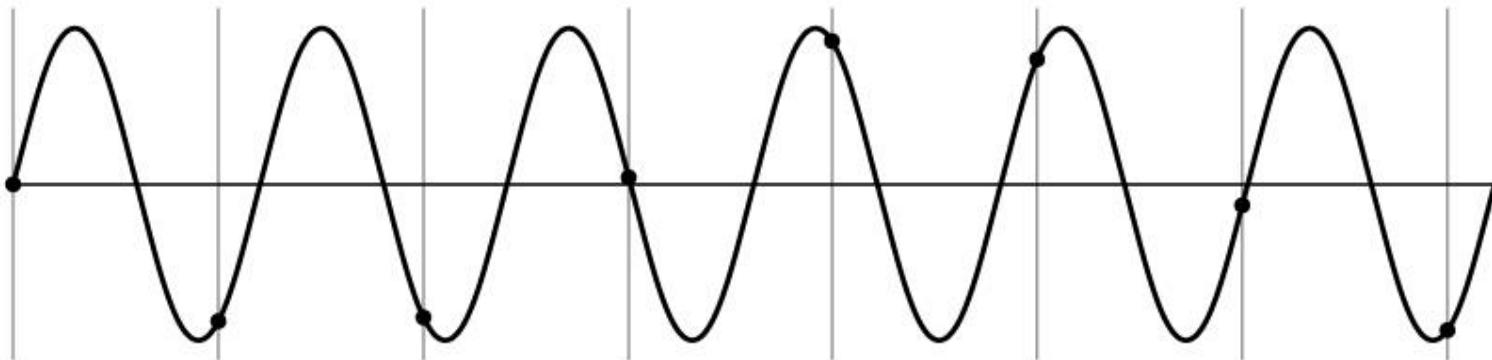
Sampling and Reconstruction

- Simple example: a sign wave



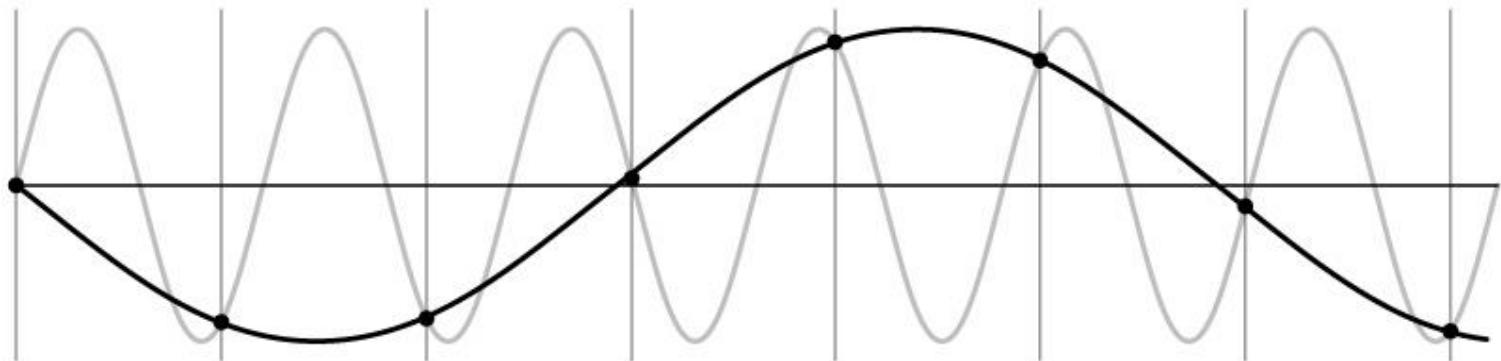
Undersampling

- What if we “missed” things between the samples?
- Simple example: undersampling a sine wave
 - unsurprising result: information is lost



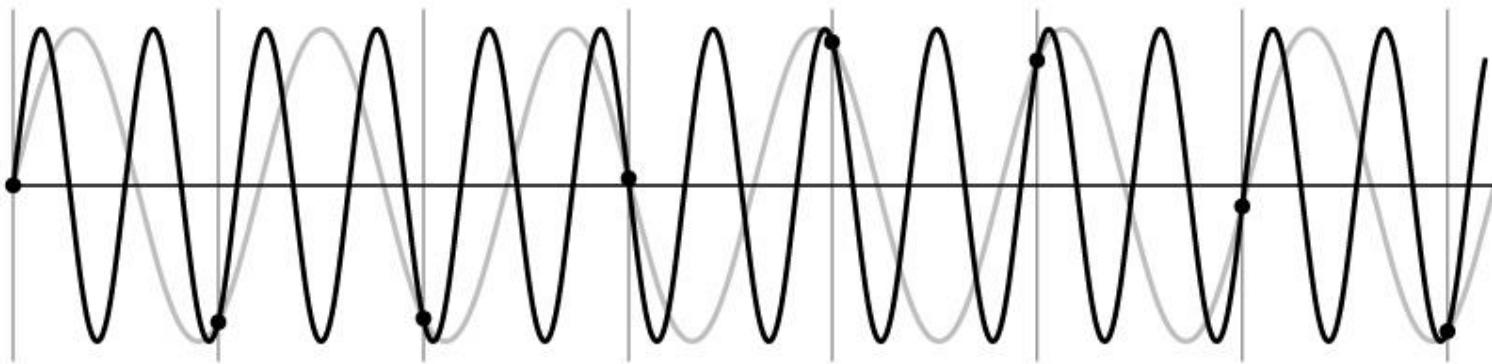
Undersampling

- What if we “missed” things between the samples?
- Simple example: undersampling a sine wave
 - unsurprising result: information is lost
 - surprising result: indistinguishable from lower frequency



Undersampling

- What if we “missed” things between the samples?
- Simple example: undersampling a sine wave
 - unsurprising result: information is lost
 - surprising result: indistinguishable from lower frequency
 - also, was always indistinguishable from higher frequencies
 - aliasing: signals “traveling in disguise” as other frequencies

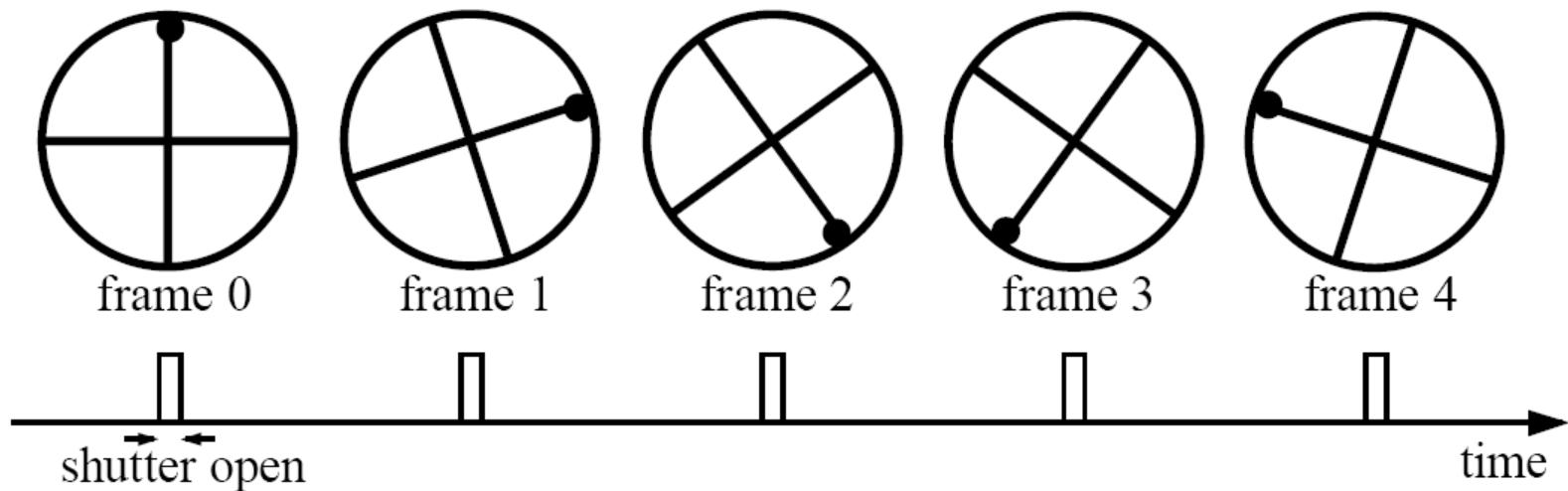


Aliasing in video

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)

Aliasing in images



Disintegrating textures

Antialiasing

What can we do about aliasing?

Sample more often

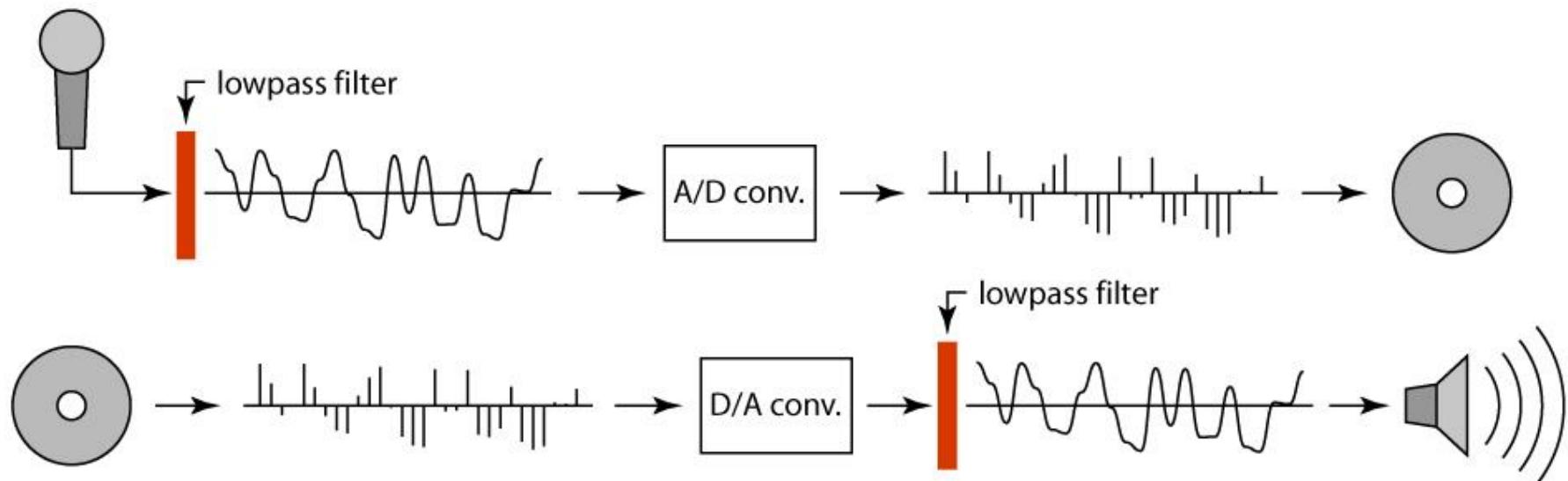
- Join the Mega-Pixel craze of the photo industry
- But this can't go on forever

Make the signal less “wiggly”

- Get rid of some high frequencies
- Will lose information
- But it's better than aliasing

Preventing aliasing

- Introduce lowpass filters:
 - remove high frequencies leaving only safe, low frequencies
 - choose lowest frequency in reconstruction (disambiguate)

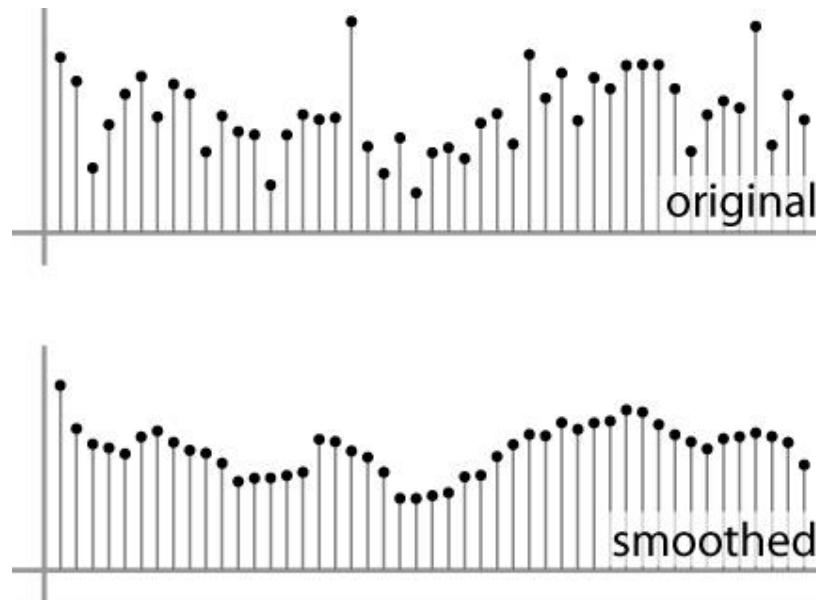


Linear filtering: a key idea

- Transformations on signals; e.g.:
 - bass/treble controls on stereo
 - blurring/sharpening operations in image editing
 - smoothing/noise reduction in tracking
- Key properties
 - linearity: $\text{filter}(f + g) = \text{filter}(f) + \text{filter}(g)$
 - shift invariance: behavior invariant to shifting the input
 - delaying an audio signal
 - sliding an image around
- Can be modeled mathematically by *convolution*

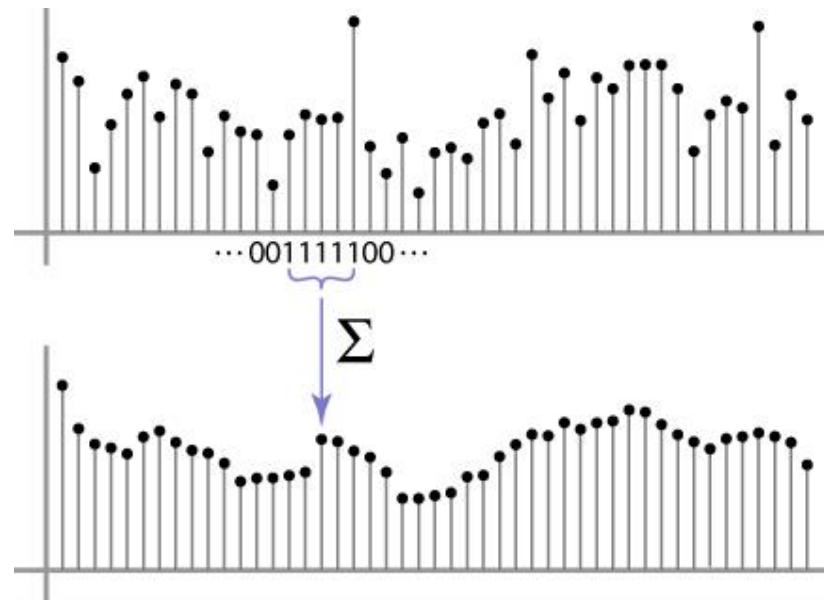
Moving Average

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing



Moving Average

- Can add weights to our moving average
- *Weights* $[..., 0, 1, 1, 1, 1, 1, 0, ...] / 5$



In 2D: box filter

$$h[\cdot, \cdot]$$

$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

Image filtering

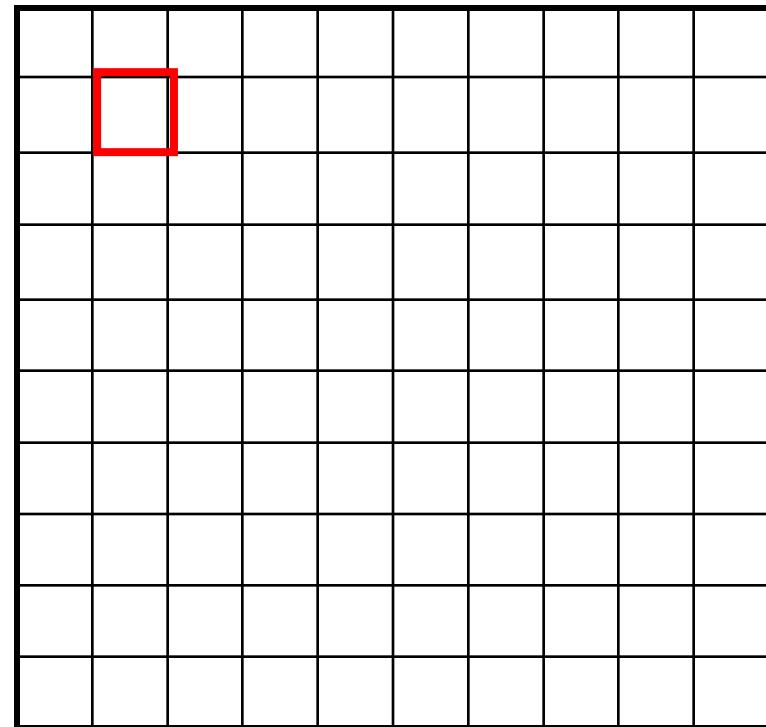
$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

$g[.,.]$

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



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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[.,.]$

$g[.,.]$

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

0			10							

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[.,.]$

$g[.,.]$

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

			0	10	20				

Image filtering

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

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

$g[.,.]$

			0	10	20	30			

Image filtering

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

1	1	1
1	1	1
1	1	1

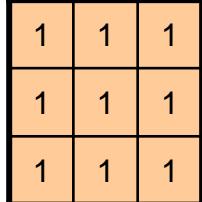
$f[.,.]$

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

$g[.,.]$

0 10 20 30 30

Image filtering

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


A 3x3 matrix with all entries equal to 1/9. The matrix is enclosed in a black border.

$f[.,.]$

$g[.,.]$

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

	0	10	20	30	30				

Image filtering

 $f[.,.]$

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

 $g[.,.]$

	0	10	20	30	30					

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

1	1	1
1	1	1
1	1	1

Image filtering

$$h[\cdot, \cdot] \quad \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$$f[\cdot, \cdot]$$

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

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

	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	

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

Credit: S. Seitz

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$h[\cdot, \cdot]$$

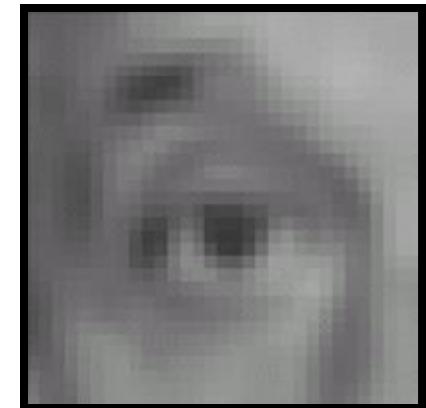
$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Linear filters: examples



Original

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



Blur (with a mean filter)

Cross-correlation

Let F be the image, H be the kernel (of size $2k+1 \times 2k+1$), and G be the output image

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

This is called a **cross-correlation** operation:

$$G = H \otimes F$$

- Can think of as a “dot product” between local neighborhood and kernel for each pixel

Practice with linear filters



Original

0	0	0
0	1	0
0	0	0

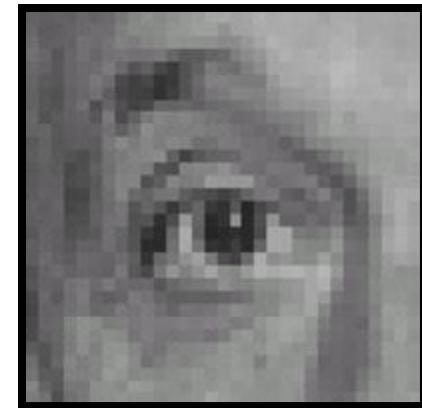
?

Practice with linear filters



Original

0	0	0
0	1	0
0	0	0



Filtered
(no change)

Practice with linear filters



Original

0	0	0
0	0	1
0	0	0

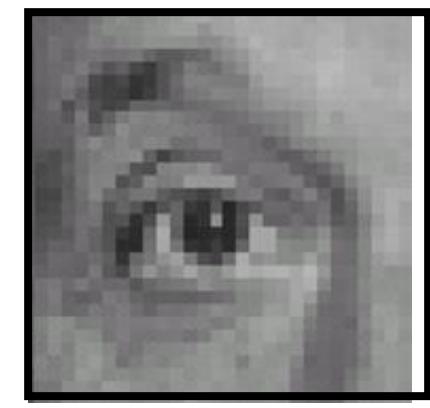
?

Practice with linear filters



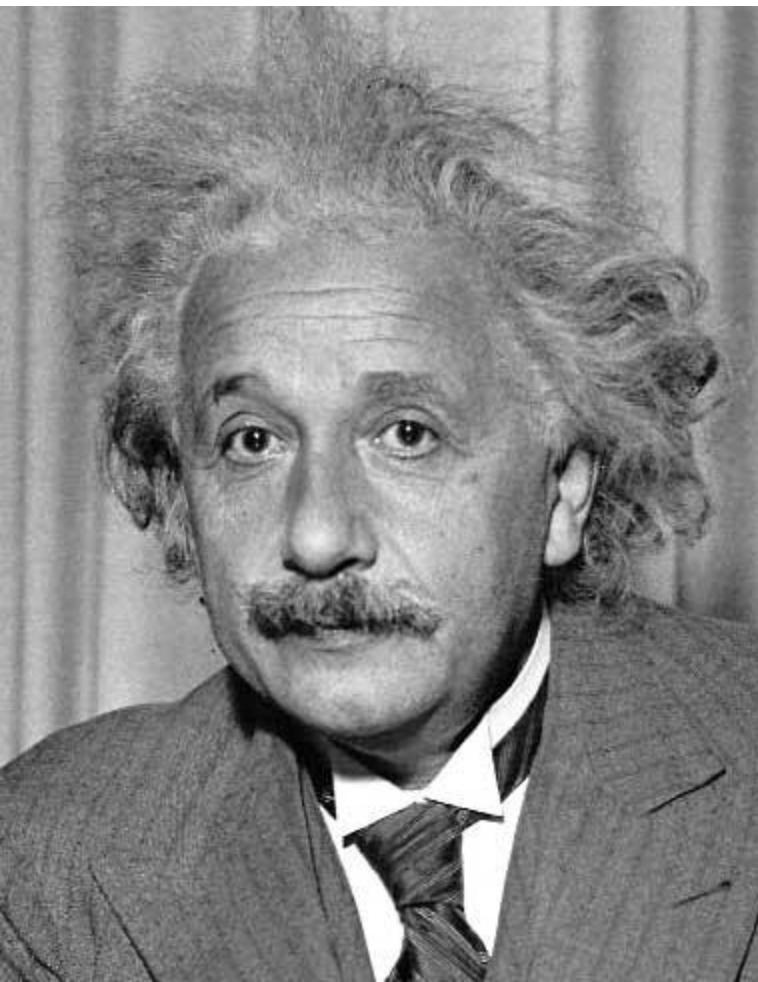
Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Other filters



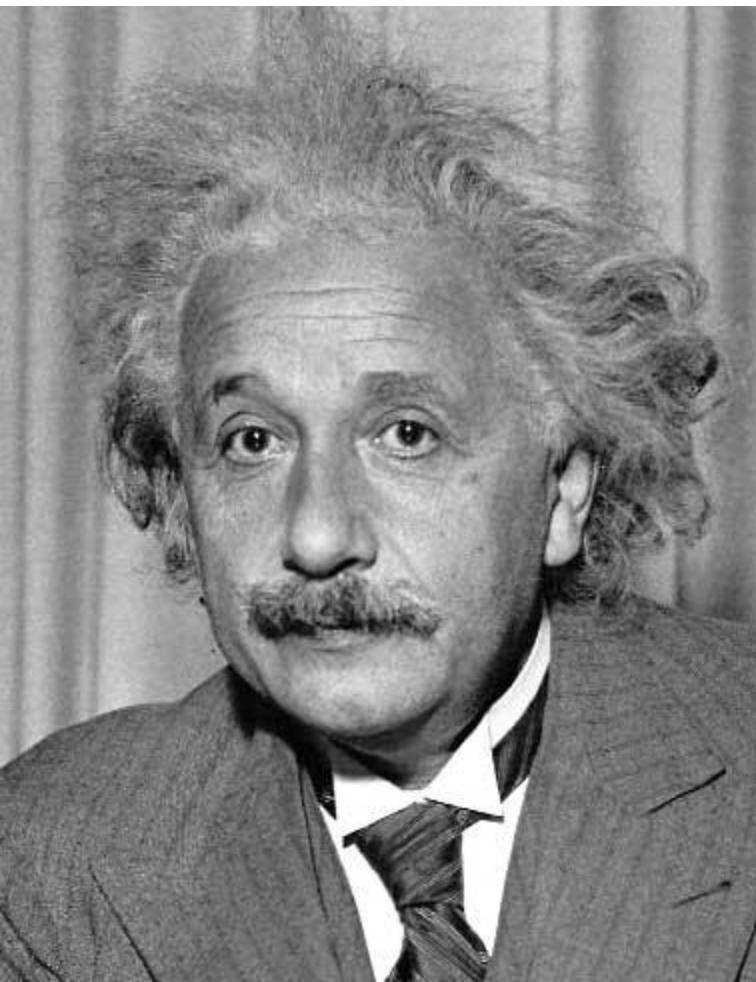
1	0	-1
2	0	-2
1	0	-1

Sobel



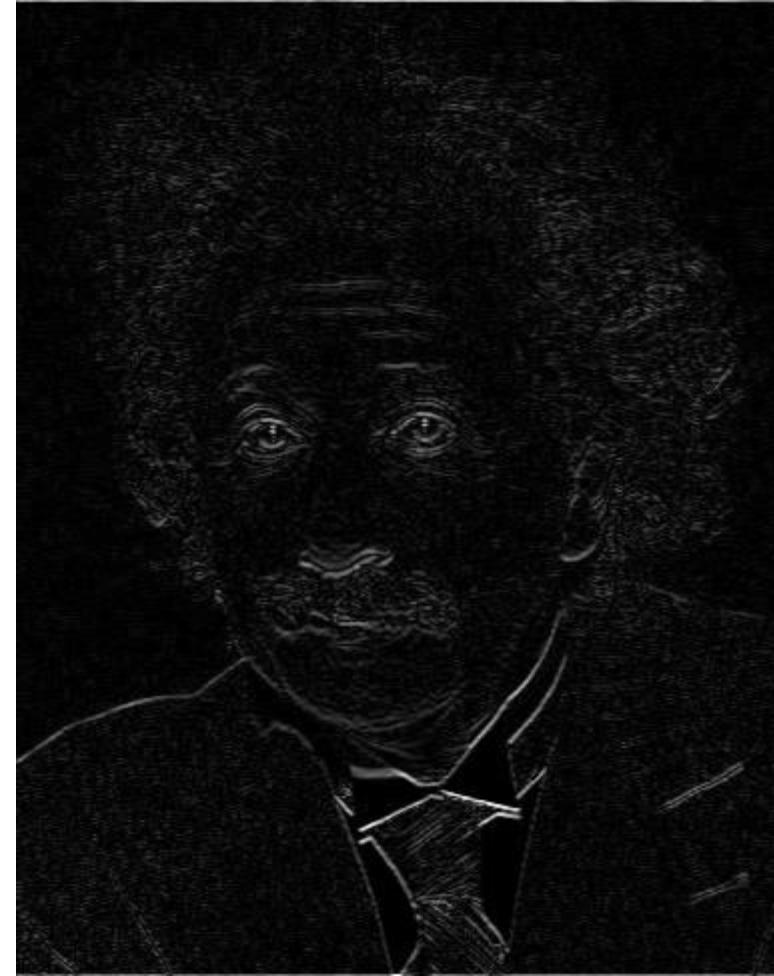
Vertical Edge
(absolute value)

Other filters



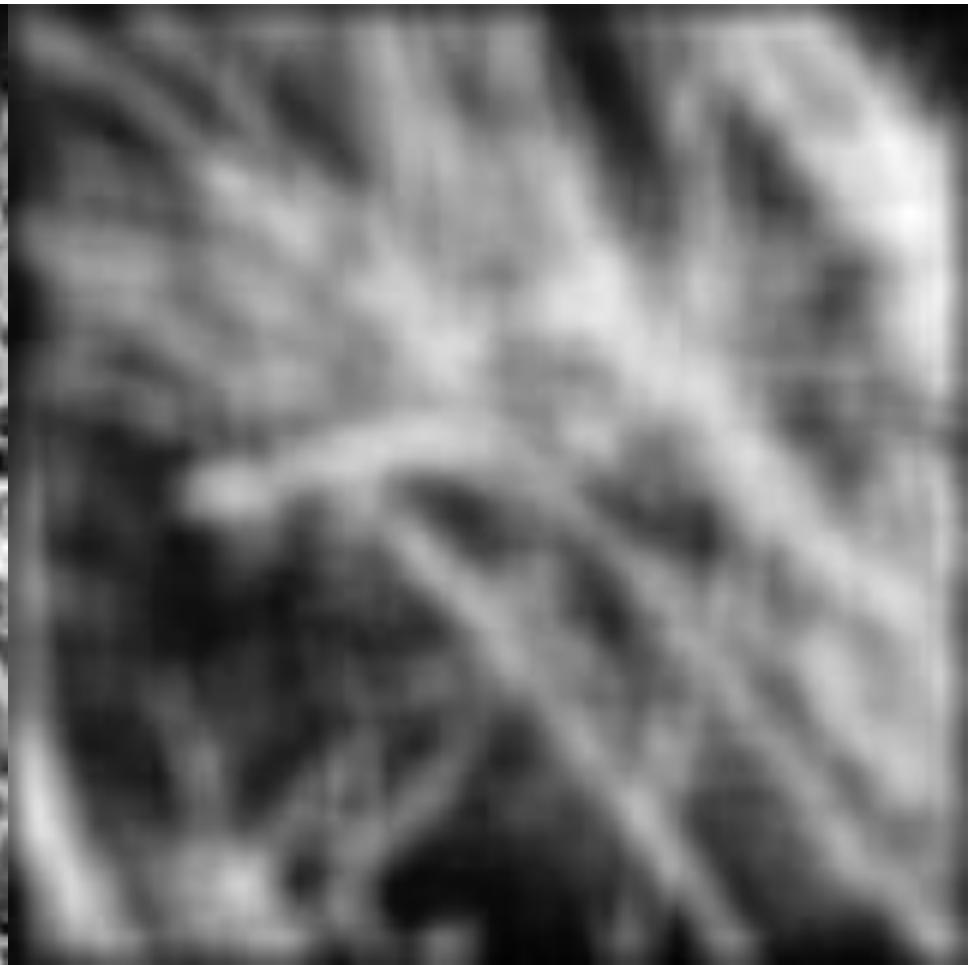
1	2	1
0	0	0
-1	-2	-1

Sobel



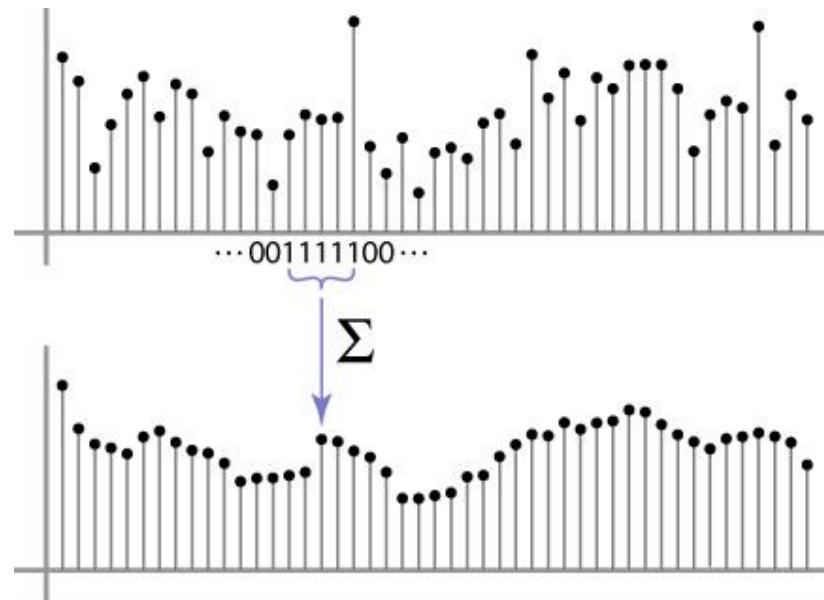
Horizontal Edge
(absolute value)

Back to the box filter



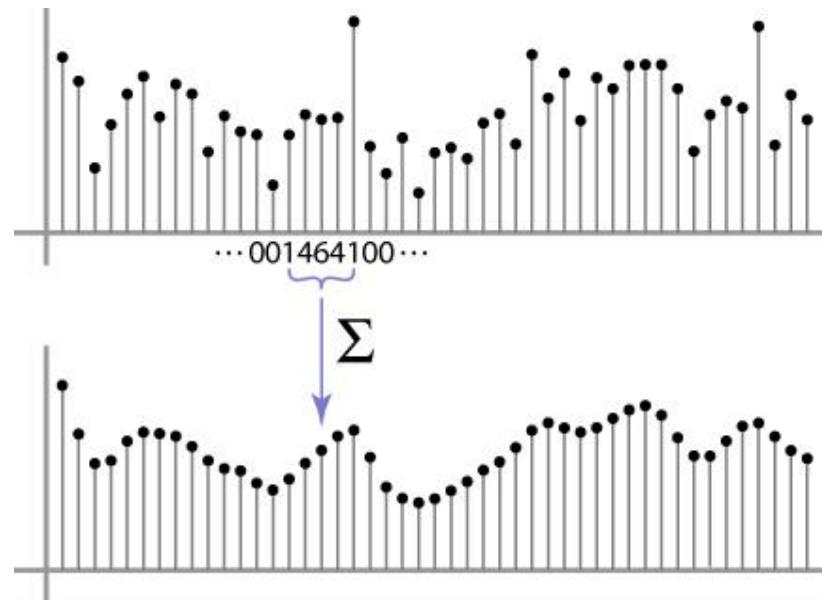
Moving Average

- Can add weights to our moving average
- *Weights* $[..., 0, 1, 1, 1, 1, 1, 0, ...] / 5$



Weighted Moving Average

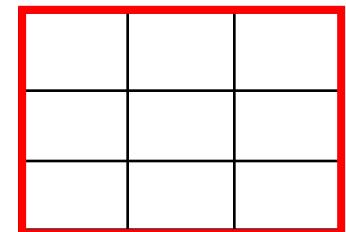
- bell curve (gaussian-like) weights $[..., 1, 4, 6, 4, 1, ...]$



Moving Average In 2D

What are the weights H?

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

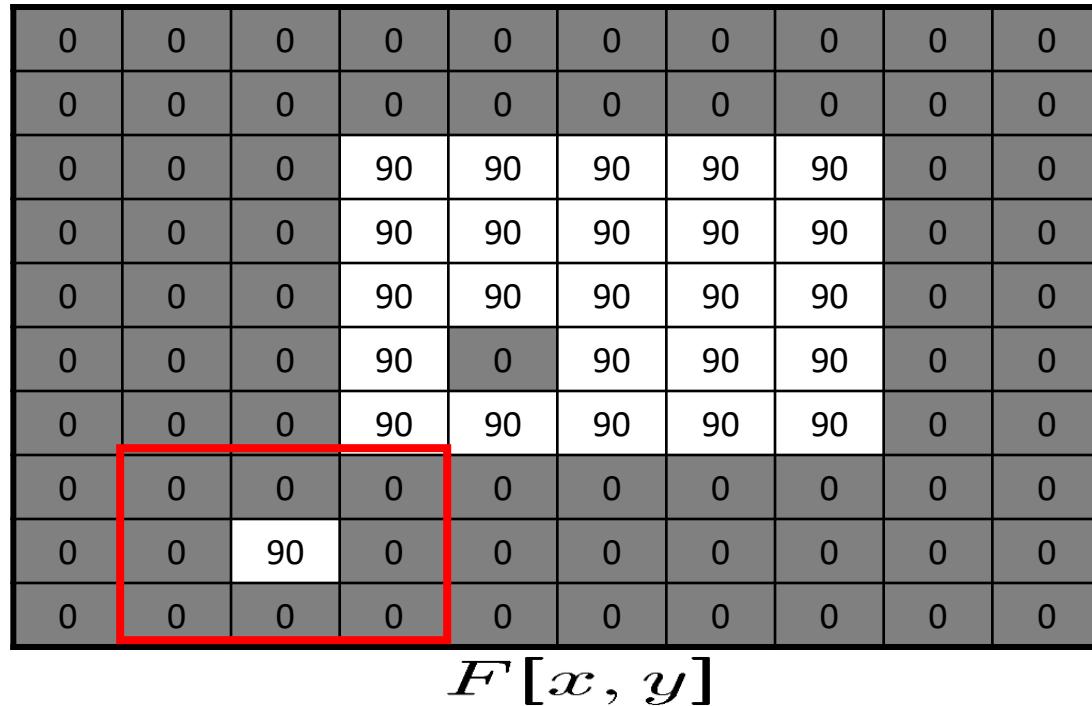


$$H[u, v]$$

$$F[x, y]$$

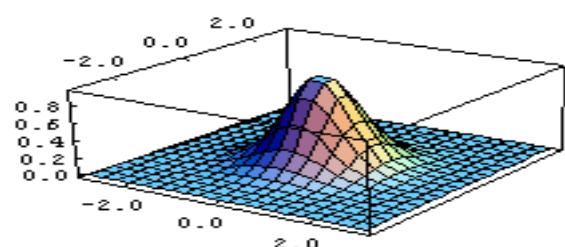
Gaussian filtering

A Gaussian kernel gives less weight to pixels further from the center of the window

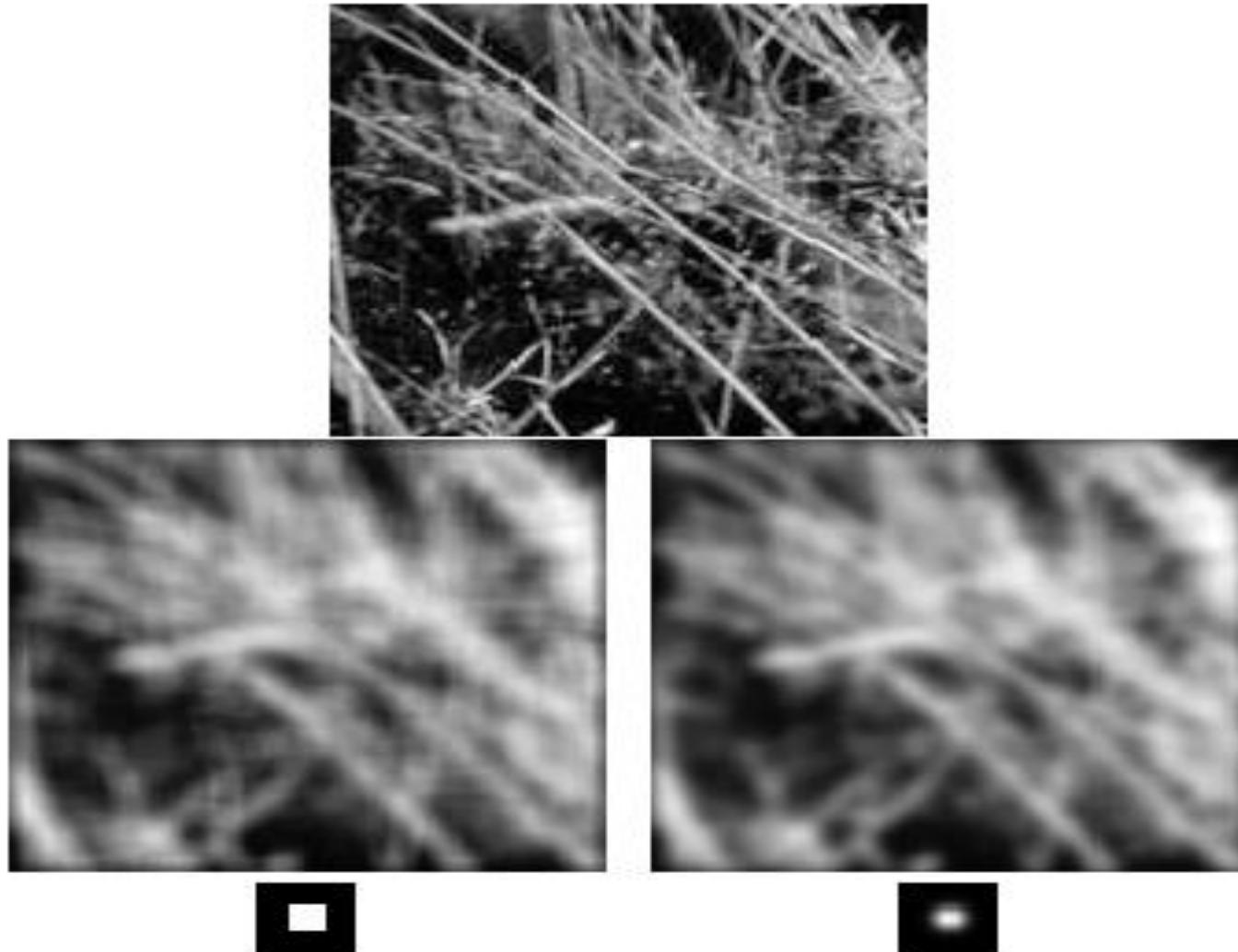


$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

This kernel is an approximation of a Gaussian function:

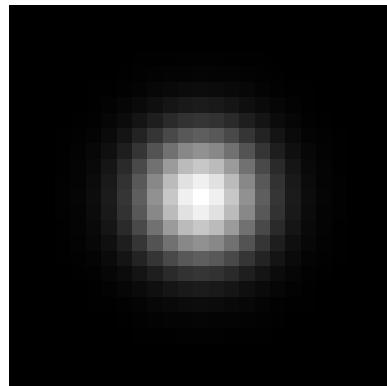
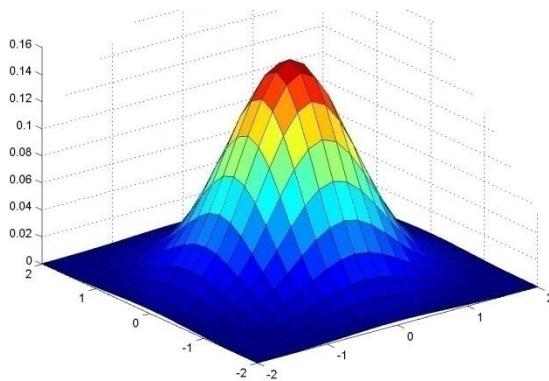


Mean vs. Gaussian filtering



Important filter: Gaussian

Weight contributions of neighboring pixels by nearness



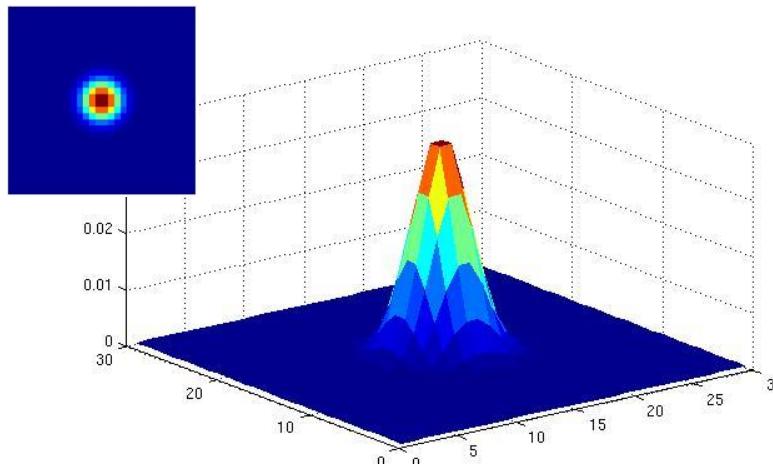
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

$5 \times 5, \sigma = 1$

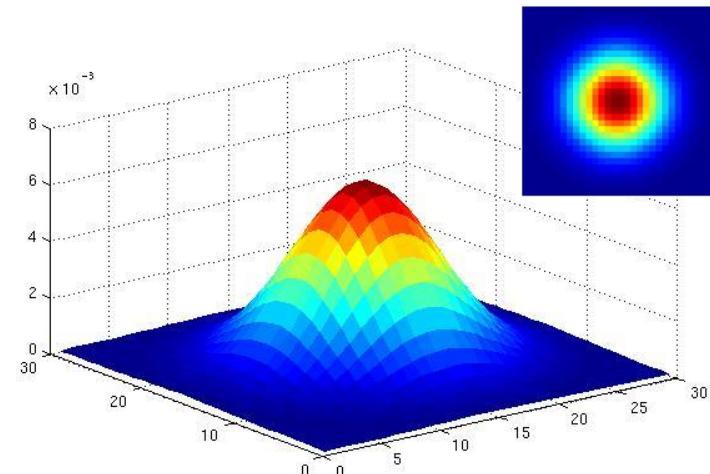
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Gaussian Kernel

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



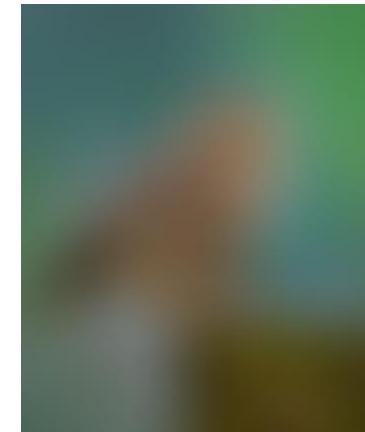
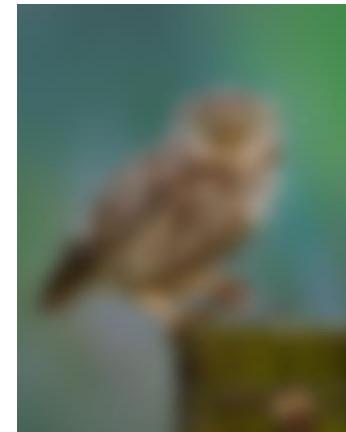
$\sigma = 2$ with 30×30 kernel



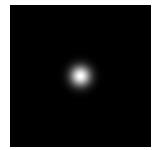
$\sigma = 5$ with 30×30 kernel

- Standard deviation σ : determines extent of smoothing

Gaussian filters



$\sigma = 1$ pixel



$\sigma = 5$ pixels



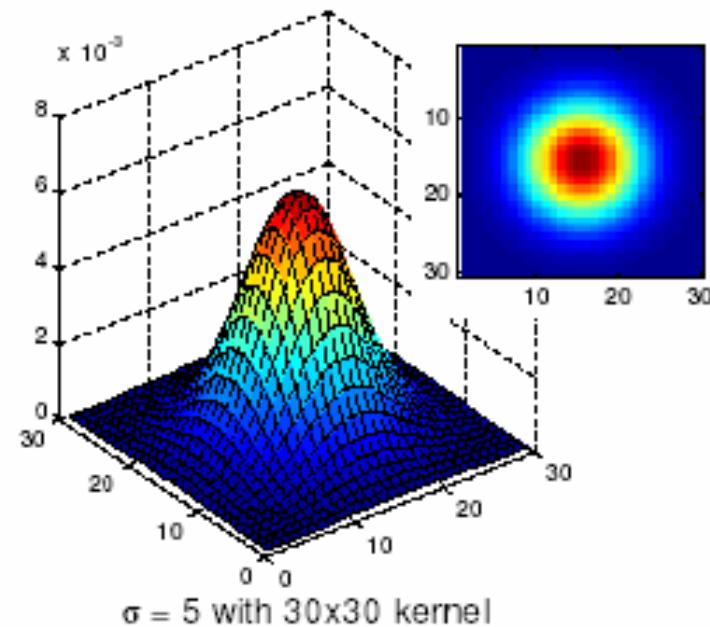
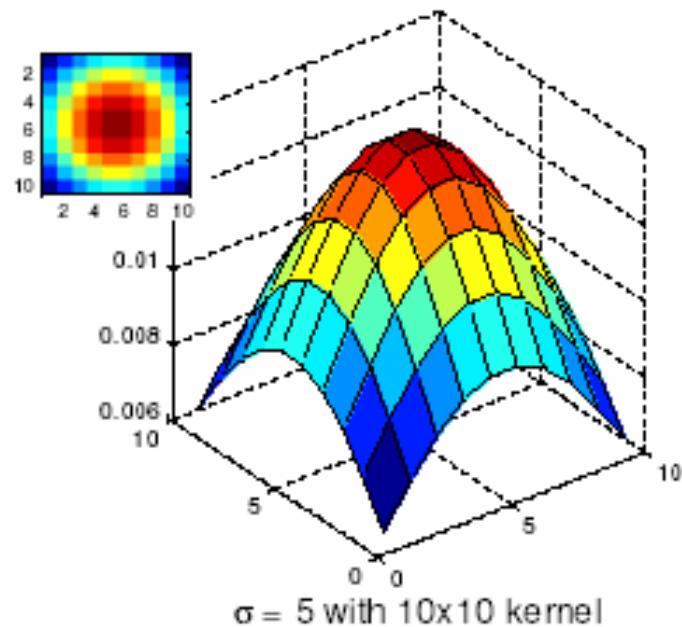
$\sigma = 10$ pixels



$\sigma = 30$ pixels

Choosing kernel width

- The Gaussian function has infinite support, but discrete filters use finite kernels

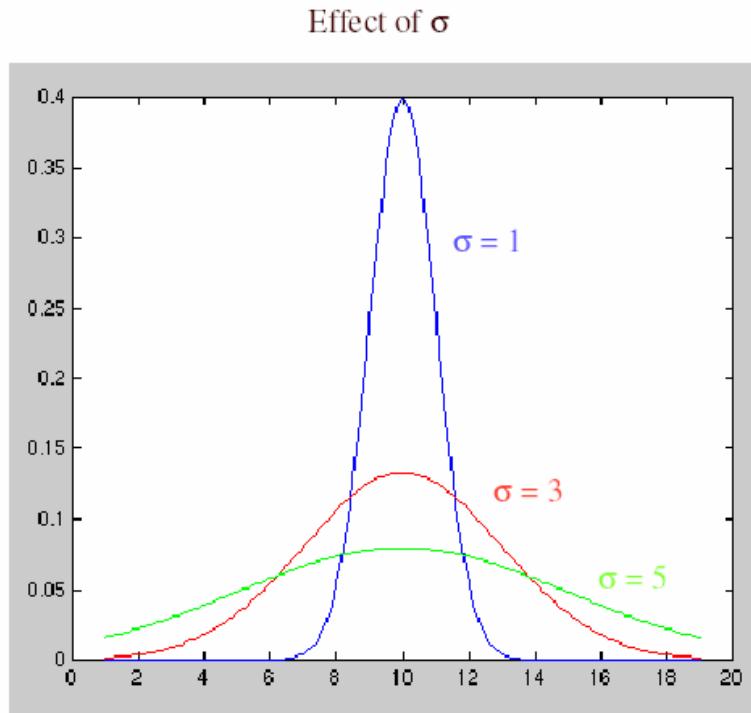


Practical matters

How big should the filter be?

Values at edges should be near zero

Rule of thumb for Gaussian: set filter half-width to about 3σ



Cross-correlation vs. Convolution

cross-correlation: $G = H \otimes F$

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

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

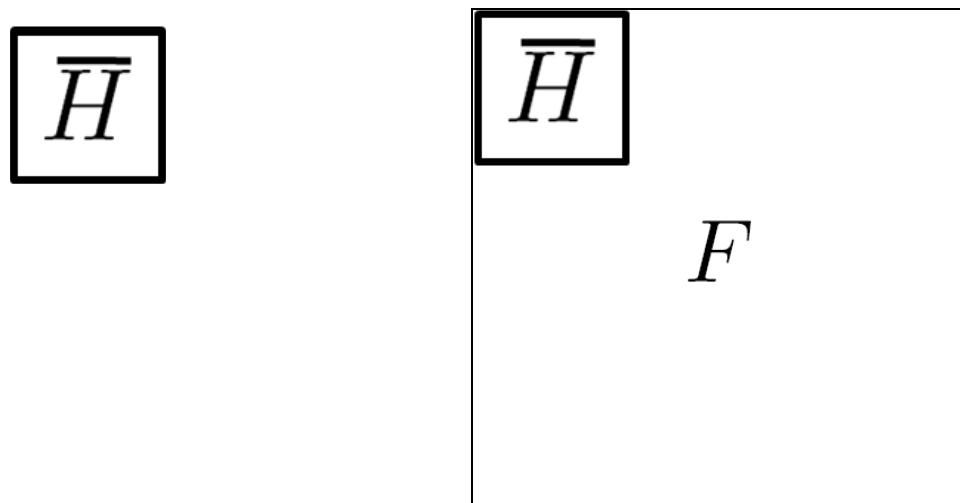
$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v]F[i - u, j - v]$$

It is written:

$$G = H \star F$$

Convolution is **commutative** and **associative**

Convolution



Convolution is nice!

- Notation: $b = c \star a$
- Convolution is a multiplication-like operation
 - commutative $a \star b = b \star a$
 - associative $a \star (b \star c) = (a \star b) \star c$
 - distributes over addition $a \star (b + c) = a \star b + a \star c$
 - scalars factor out $\alpha a \star b = a \star \alpha b = \alpha(a \star b)$
 - identity: unit impulse $e = [..., 0, 0, 1, 0, 0, ...]$
$$a \star e = a$$
- Conceptually no distinction between filter and signal
- Usefulness of associativity
 - often apply several filters one after another: $((a * b_1) * b_2) * b_3$
 - this is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

The Convolution Theorem

The greatest thing since sliced (banana) bread!

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

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

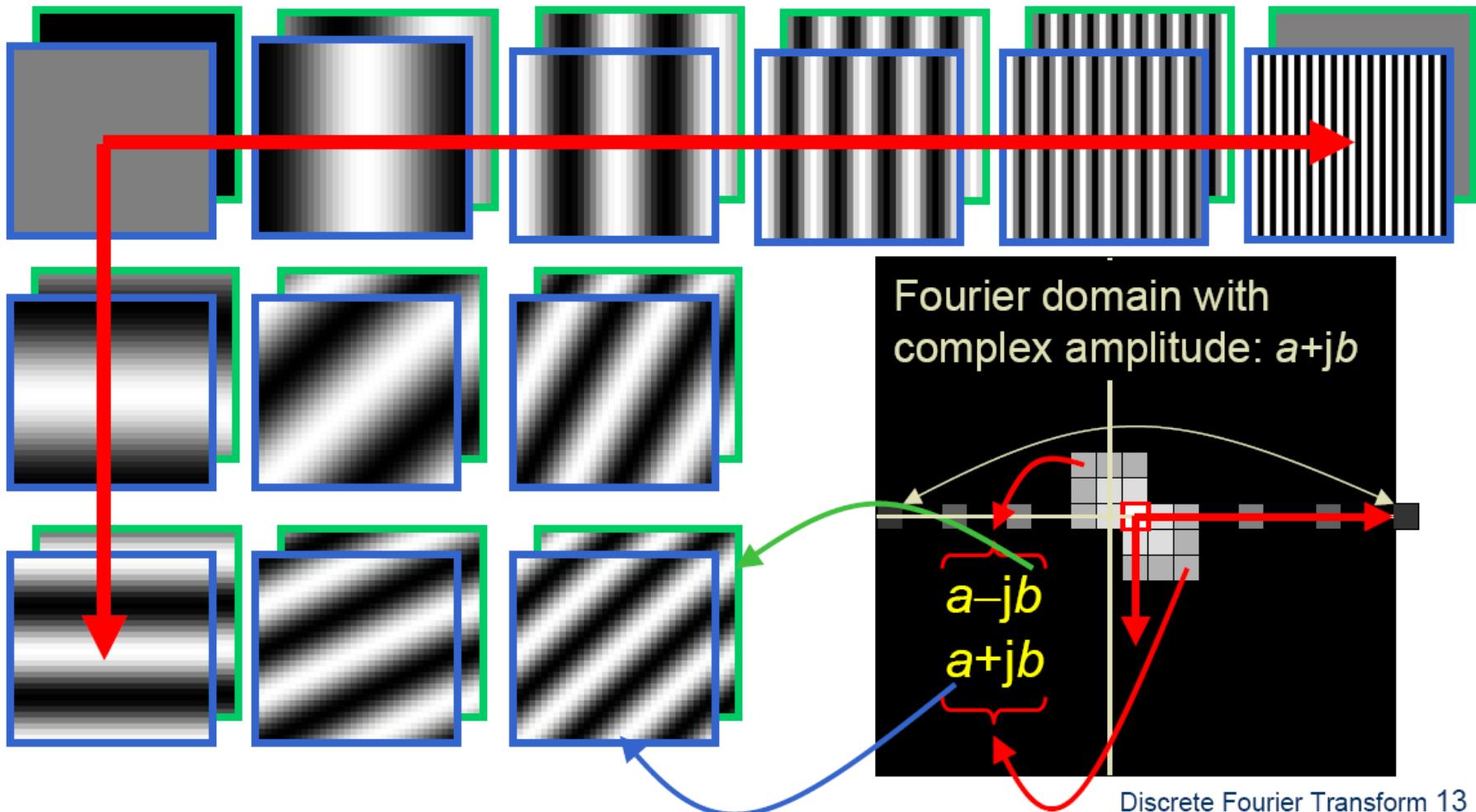
- The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms

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

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

2D Fourier transform

Teases away fast vs. slow changes in the image.



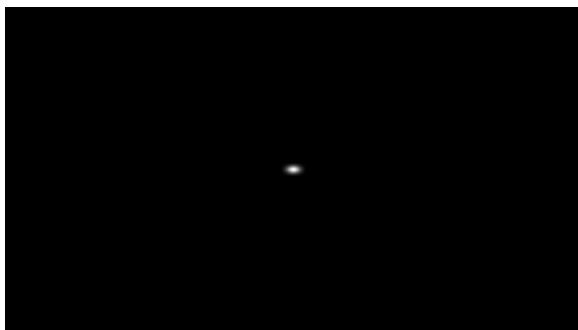
2D convolution theorem example

$f(x,y)$



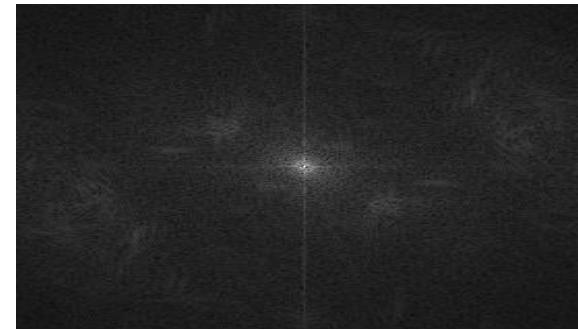
*

$h(x,y)$



↓↓

$g(x,y)$



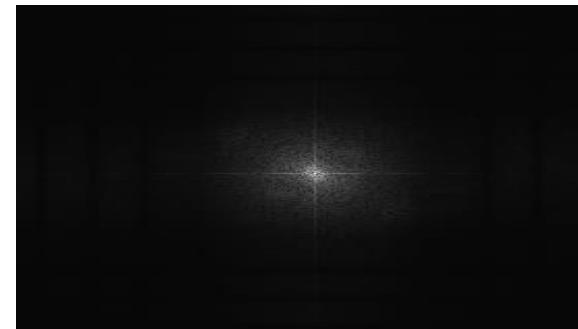
×

$|F(s_x, s_y)|$



↓↓

$|H(s_x, s_y)|$

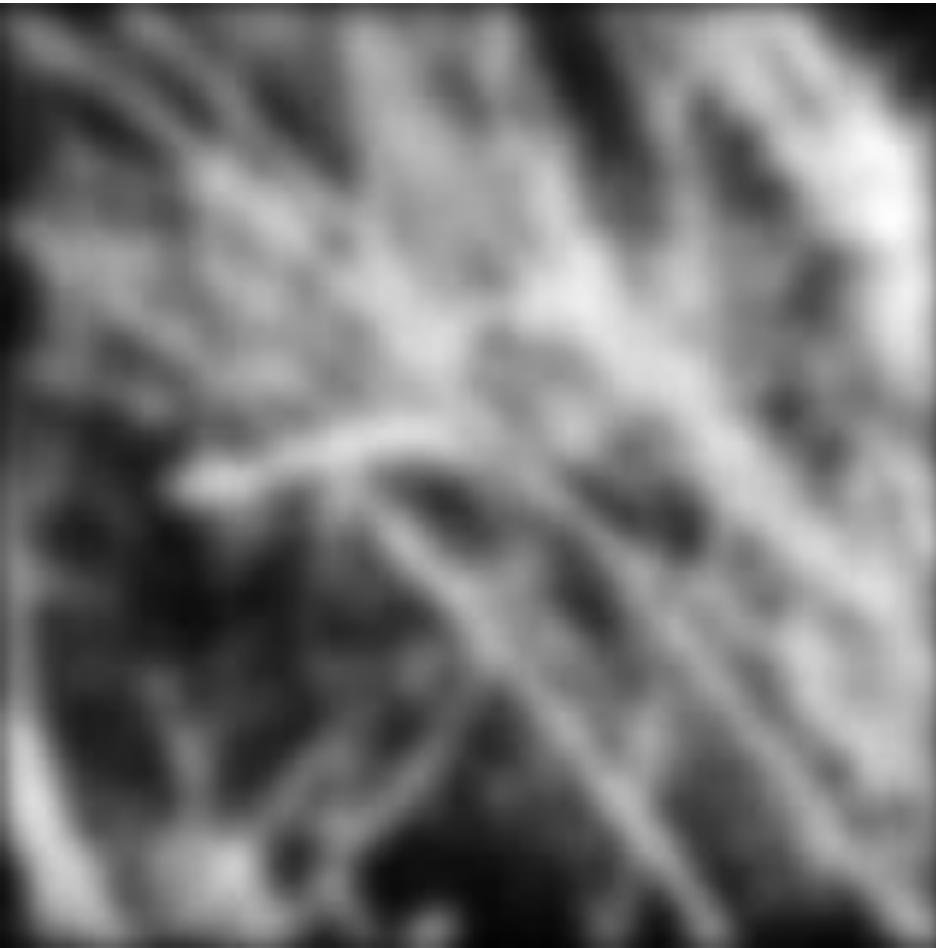


$|G(s_x, s_y)|$

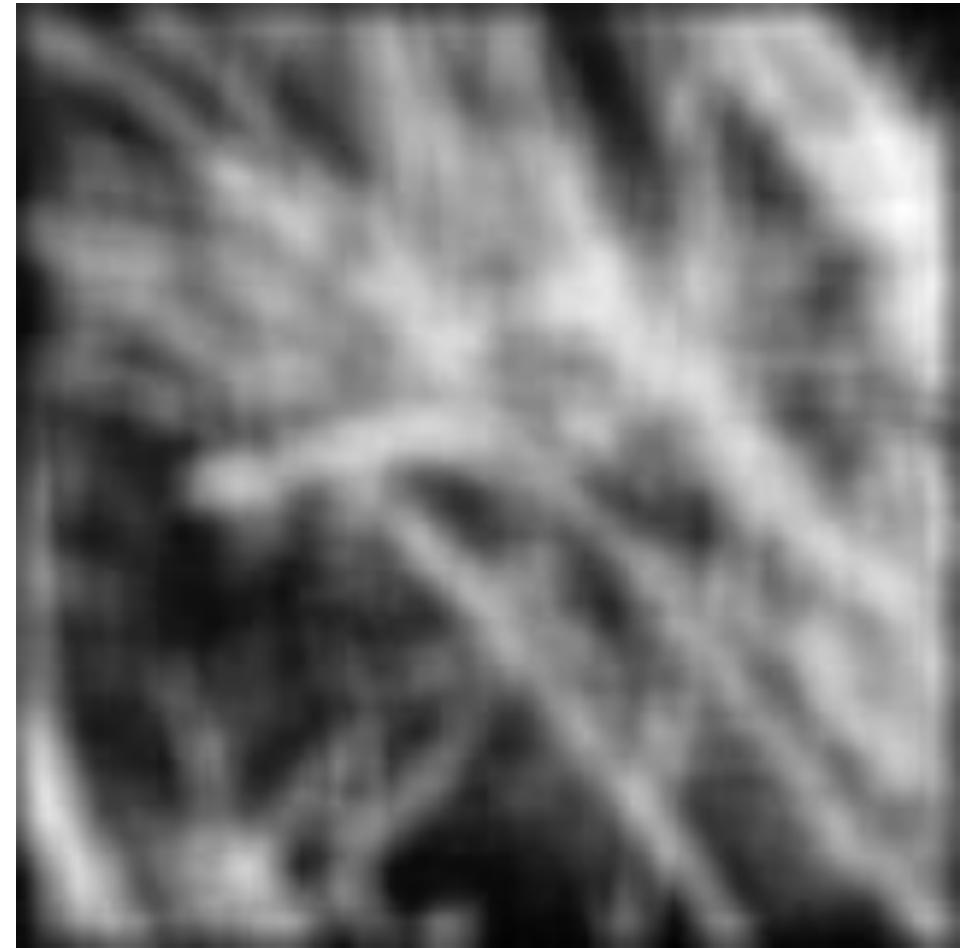
Filtering

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

Gaussian



Box filter

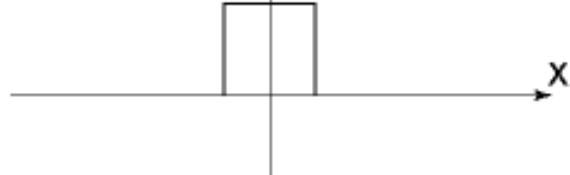


Fourier Transform pairs

Spatial domain

$$f(x)$$

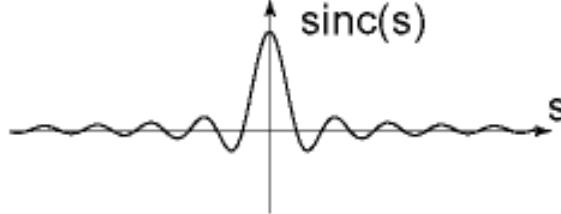
↑
box(x)



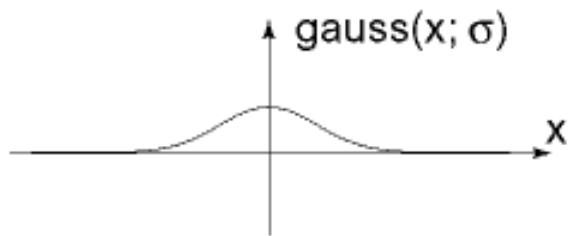
Frequency domain

$$F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi s x} dx$$

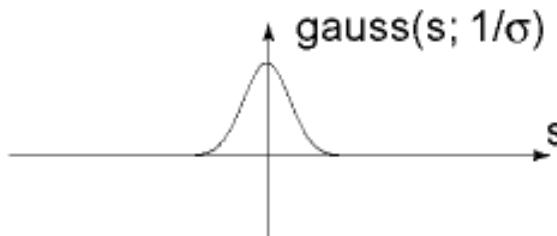
sinc(s)



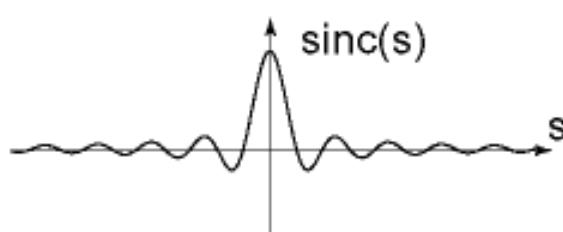
$$\text{gauss}(x; \sigma)$$



$$\text{gauss}(s; 1/\sigma)$$



$$\text{sinc}(s)$$



$$\text{box}(x)$$

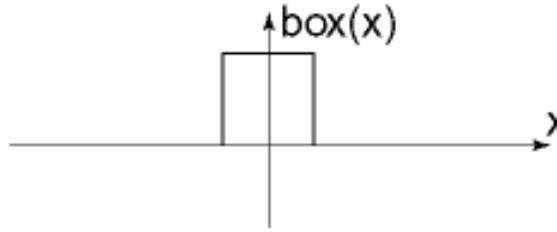


Image half-sizing

This image is too big to fit on the screen. How can we reduce it?

How to generate a half-sized version?

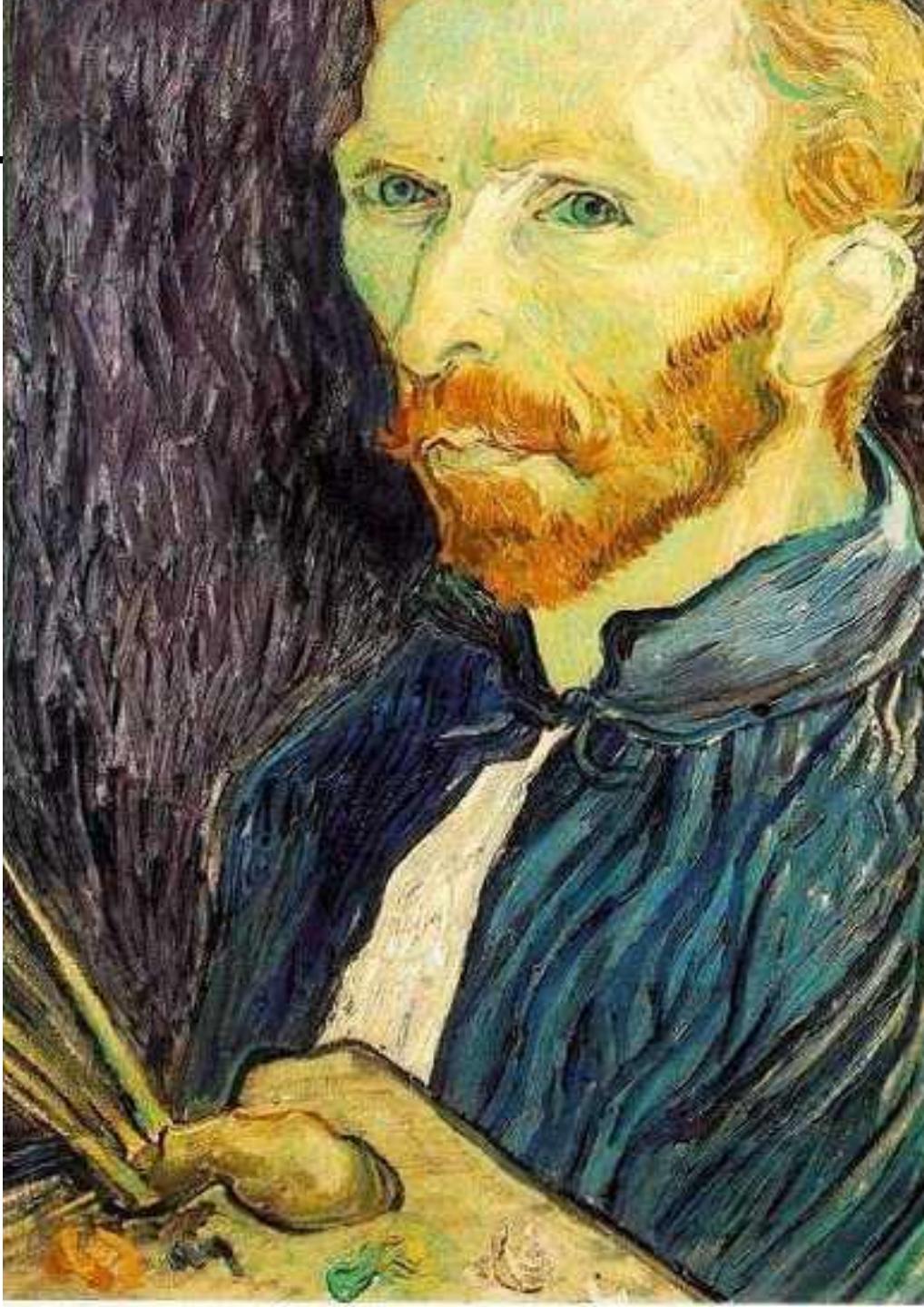
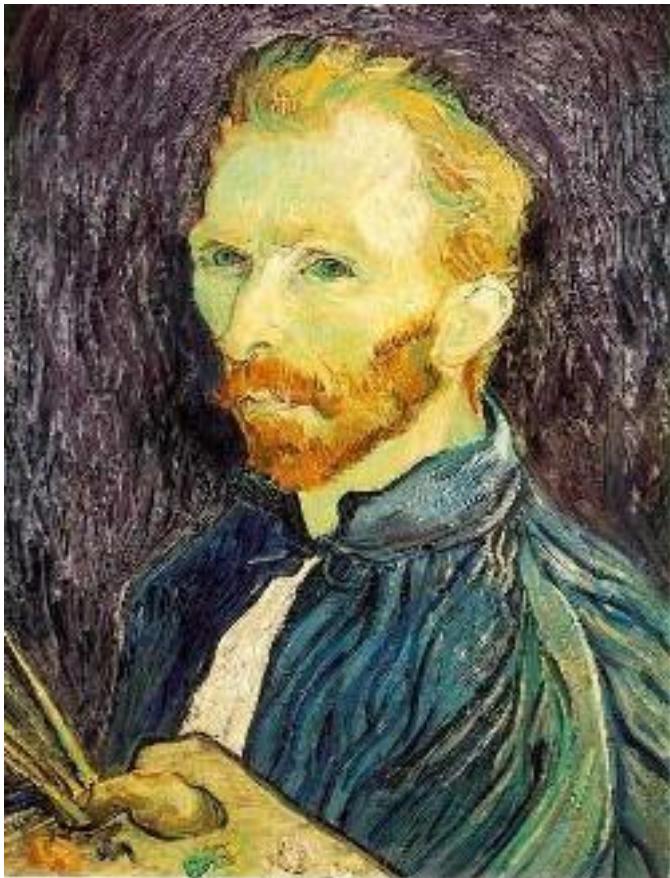


Image sub-sampling



1/2



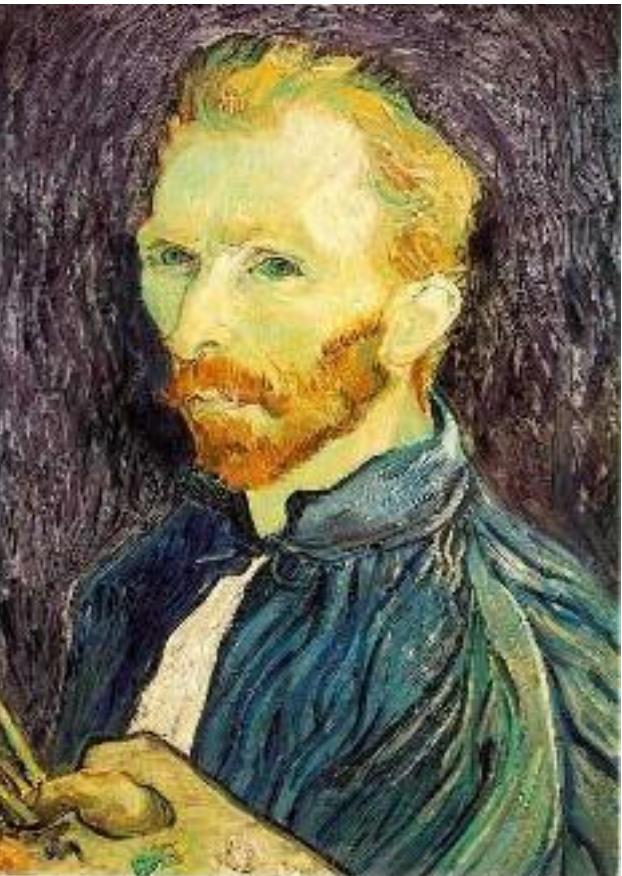
1/4



1/8

Throw away every other row and
column to create a $1/2$ size image
- called *image sub-sampling*

Image sub-sampling



1/2



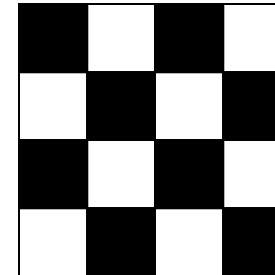
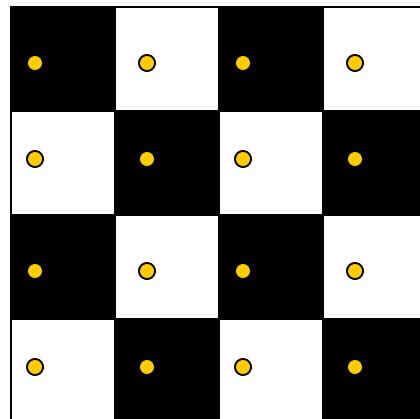
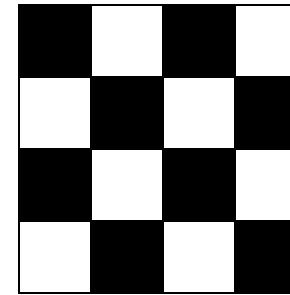
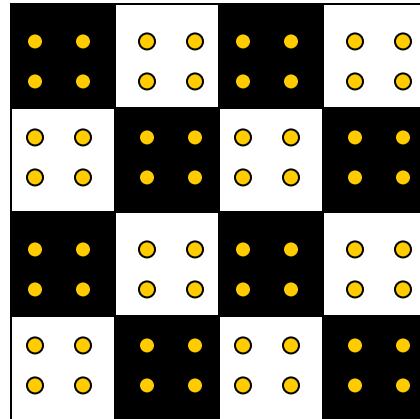
1/4 (2x zoom)



1/8 (4x zoom)

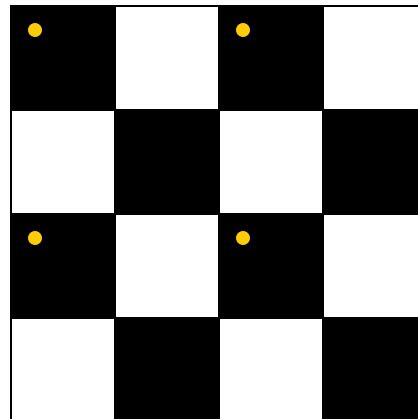
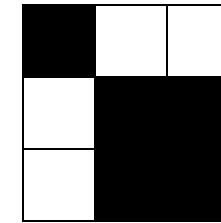
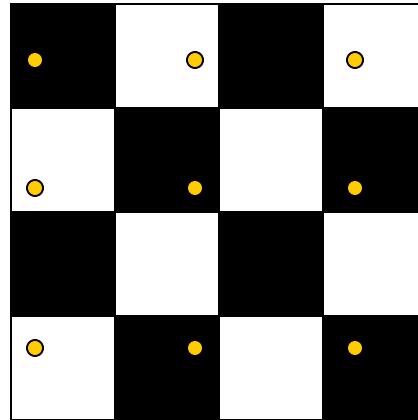
Aliasing! What do we do?

Sampling an image



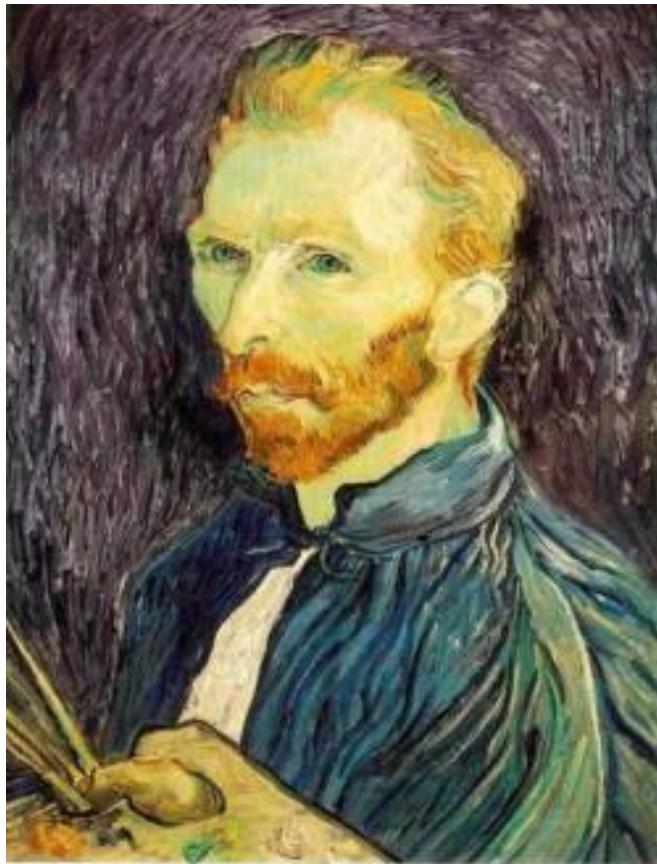
Examples of GOOD sampling

Undersampling

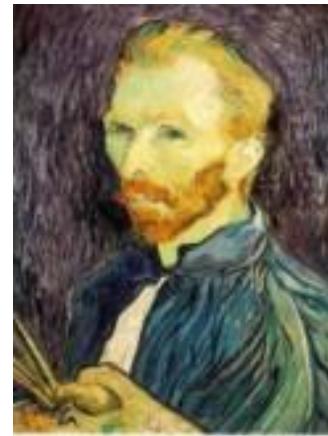


Examples of BAD sampling -> Aliasing

Gaussian (lowpass) pre-filtering



Gaussian 1/2



G 1/4

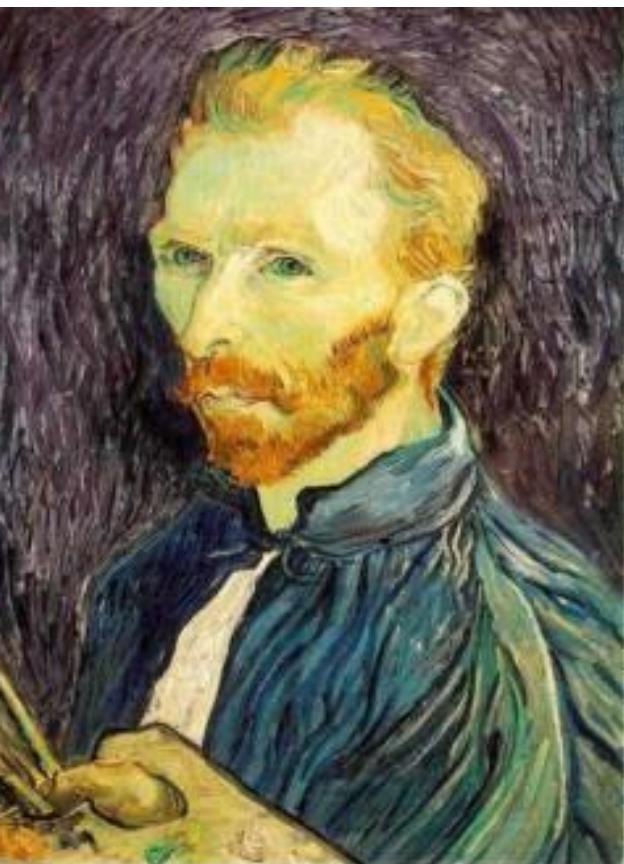


G 1/8

Solution: filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?

Subsampling with Gaussian pre-filtering



Gaussian 1/2

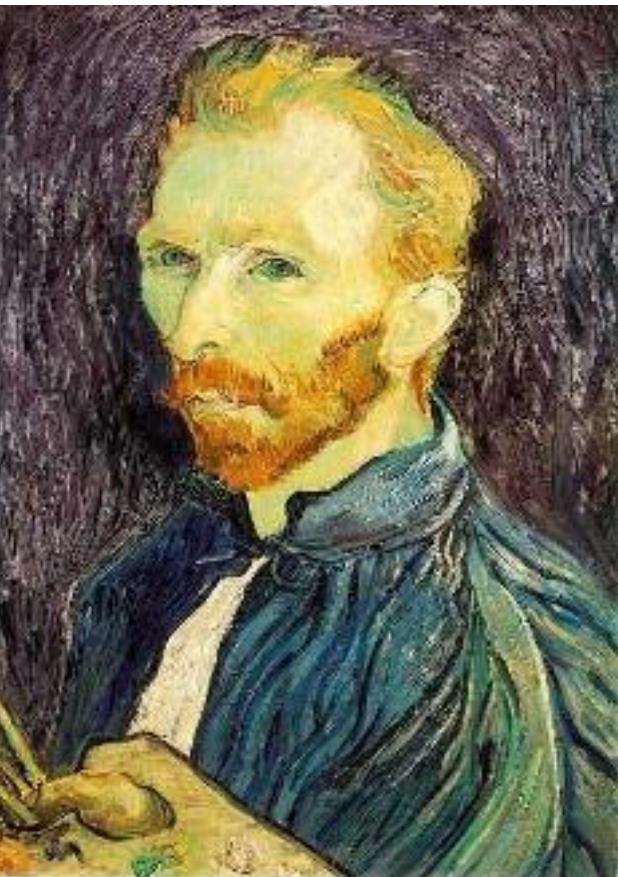


G 1/4



G 1/8

Compare with...



1/2

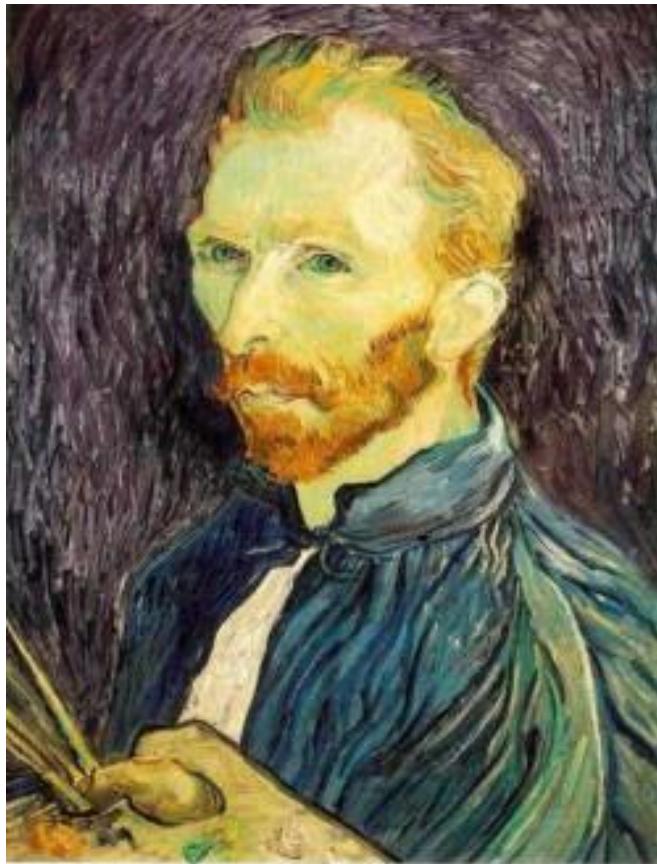


1/4 (2x zoom)

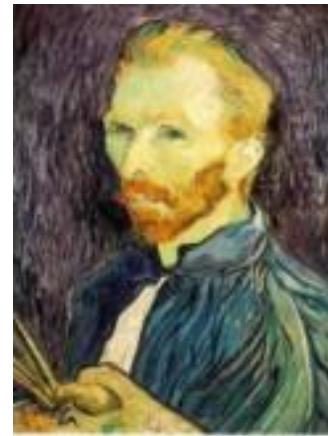


1/8 (4x zoom)

Gaussian (lowpass) pre-filtering



Gaussian 1/2



G 1/4



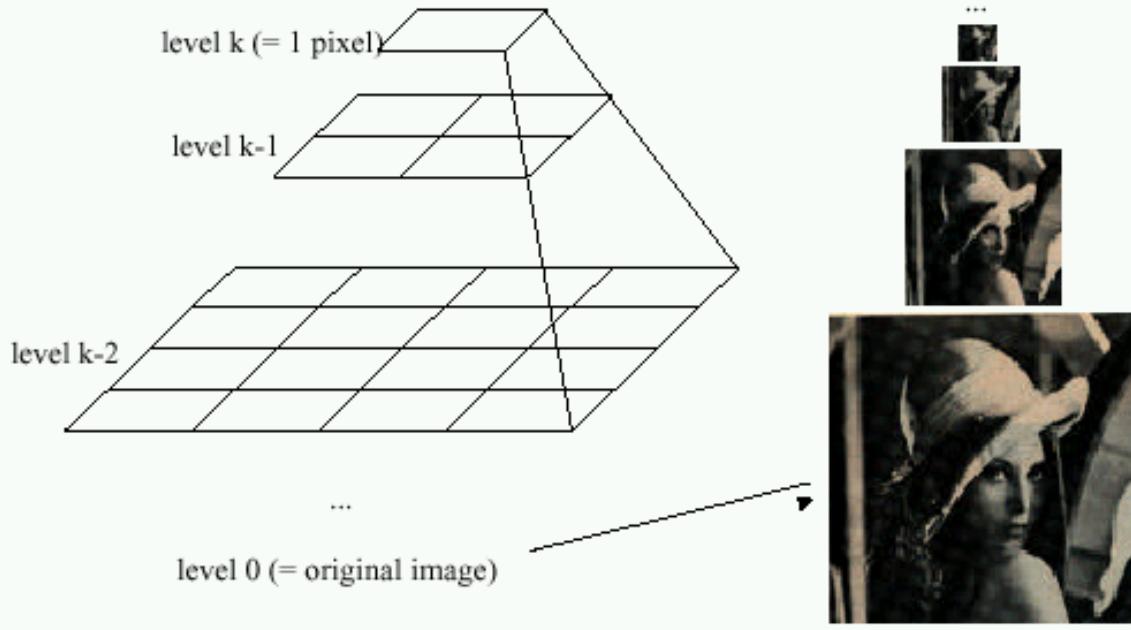
G 1/8

Solution: filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?
- How can we speed this up?

Image Pyramids

Idea: Represent $N \times N$ image as a “pyramid” of $1 \times 1, 2 \times 2, 4 \times 4, \dots, 2^k \times 2^k$ images (assuming $N=2^k$)



Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to *wavelet transform*



512

256

128

64

32

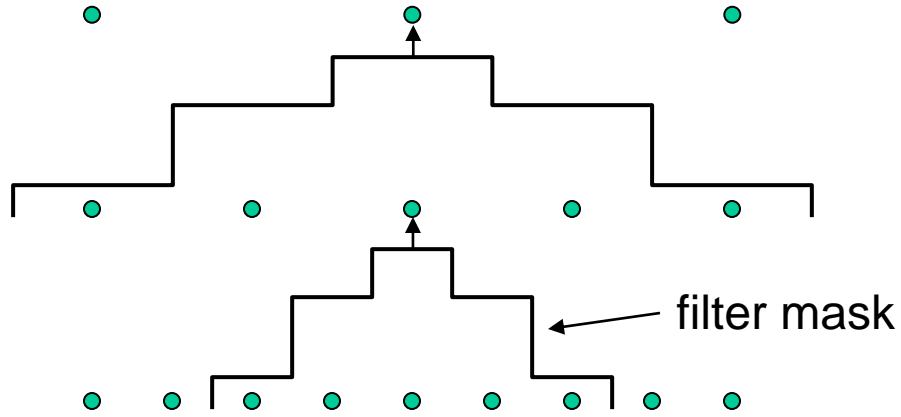
16

8



A bar in the big images is a hair on the zebra's nose; in smaller images, a stripe; in the smallest, the animal's nose

Gaussian pyramid construction



Repeat

- Filter
- Subsample

Until minimum resolution reached

- can specify desired number of levels (e.g., 3-level pyramid)

The whole pyramid is only $4/3$ the size of the original image!

What are they good for?

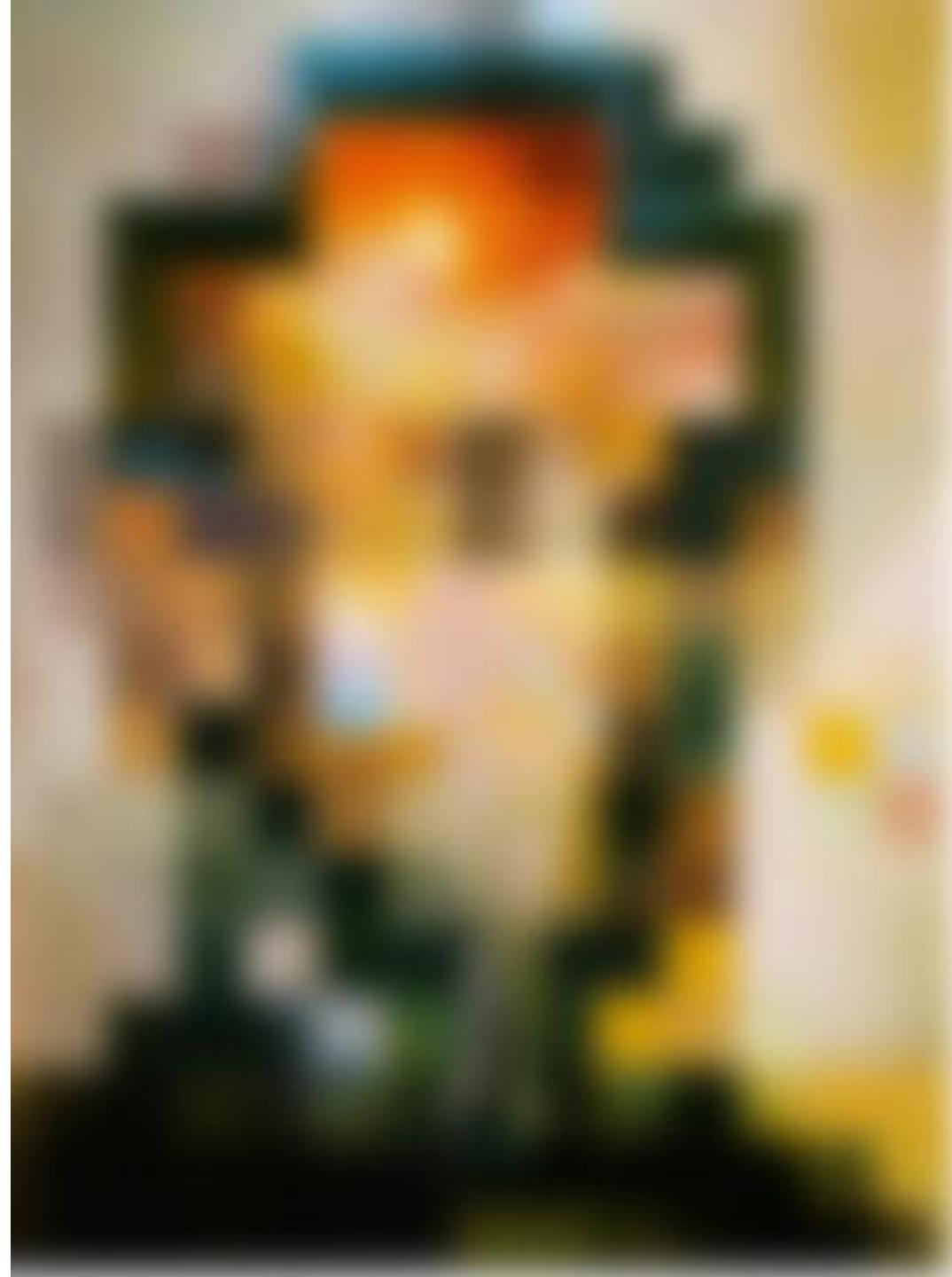
Improve Search

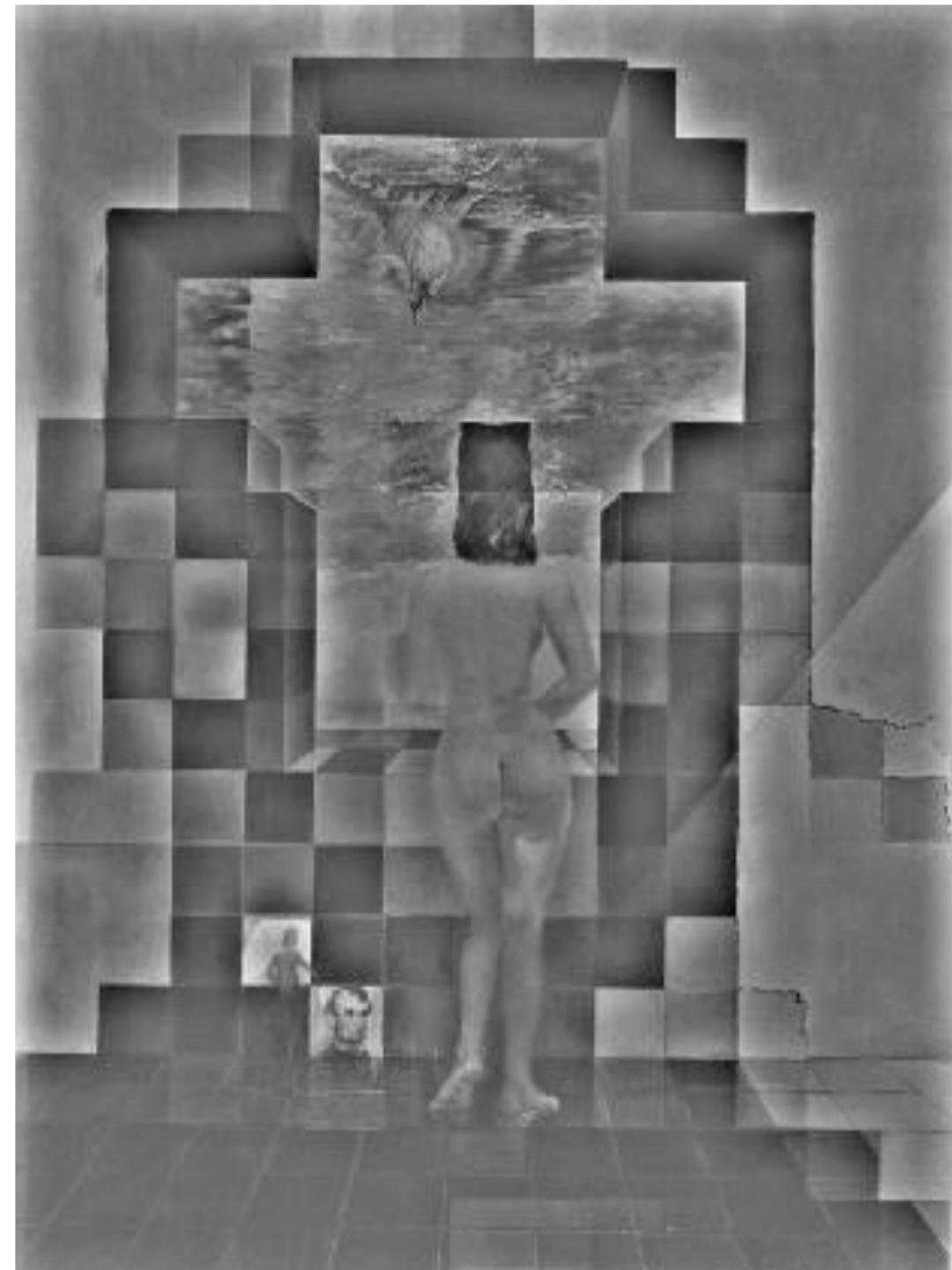
- Search over translations
 - Classic coarse-to-fine strategy
- Search over scale
 - Template matching
 - E.g. find a face at different scales



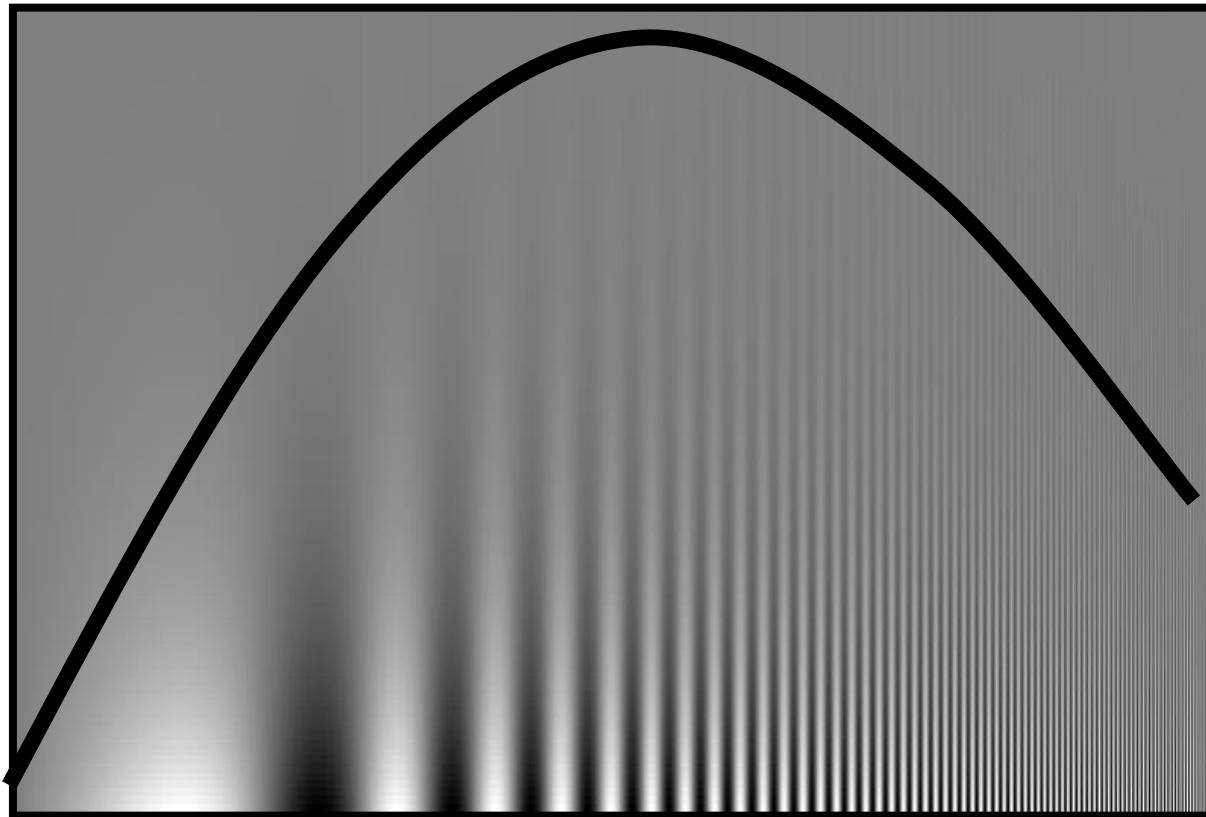
Salvador Dali

*"Gala Contemplating the Mediterranean Sea,
which at 30 meters becomes the portrait
of Abraham Lincoln", 1976*





Spatial Frequencies and Perception



Campbell-Robson contrast sensitivity curve

Filtering – Sharpening

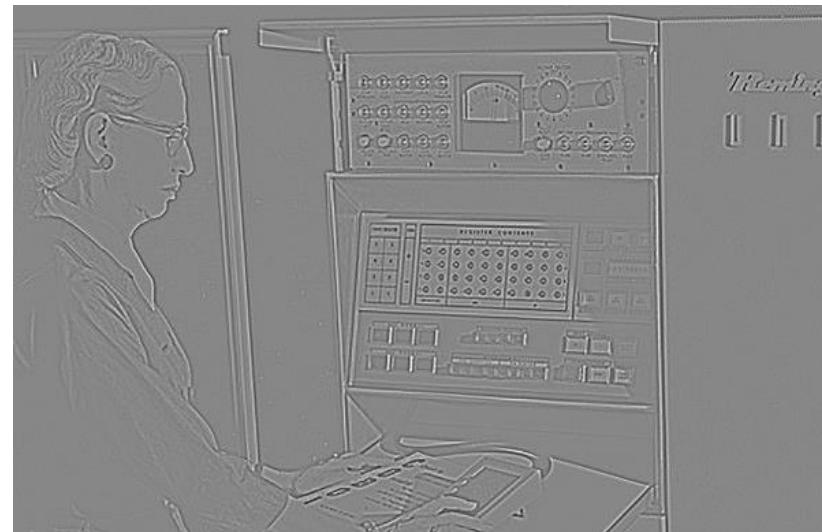
Image



Smoothed



Details

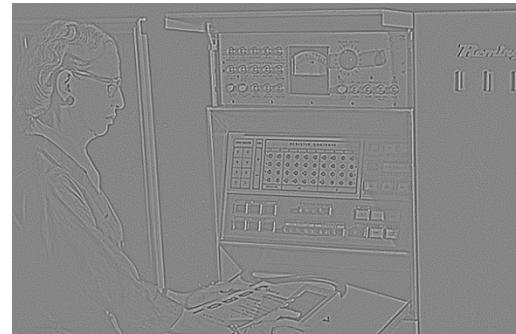


Filtering – Sharpening

Image



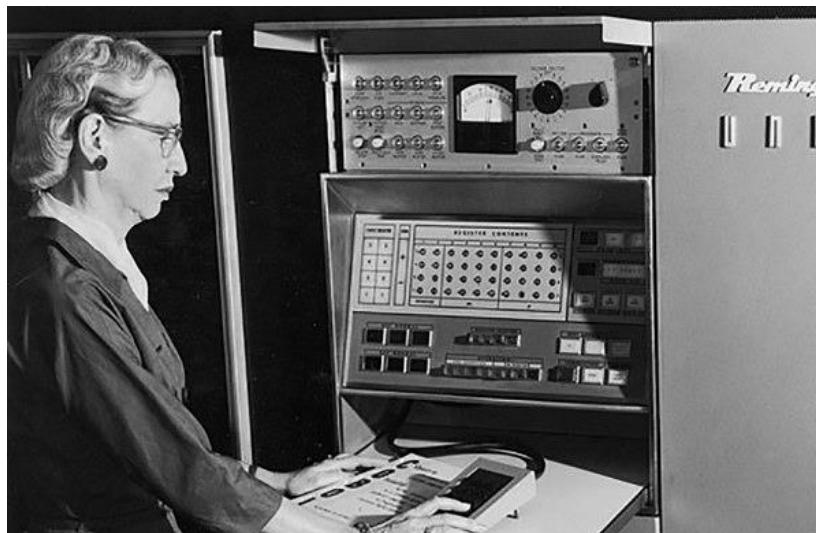
Details



$+ \alpha$

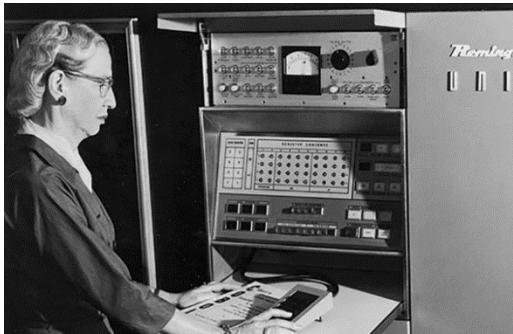
“Sharpened” $\alpha=1$

=

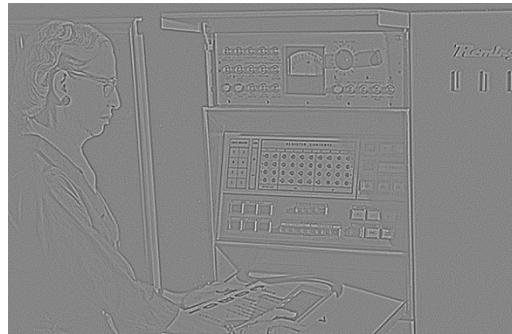


Filtering – Sharpening

Image



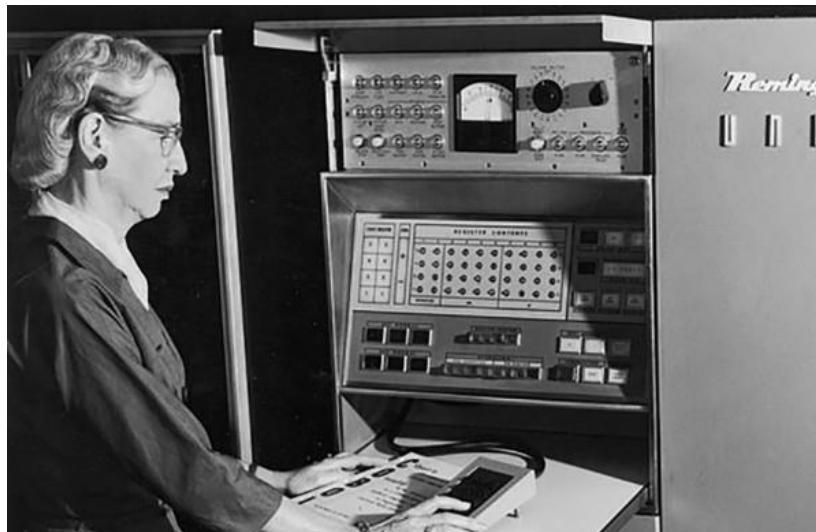
Details



$+ \alpha$

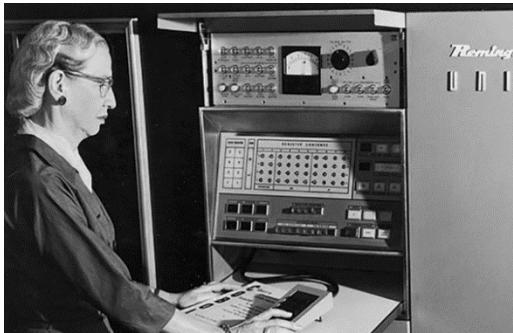
“Sharpened” $\alpha=0$

=

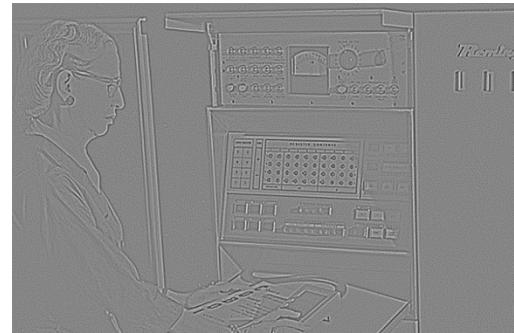


Filtering – Sharpening

Image



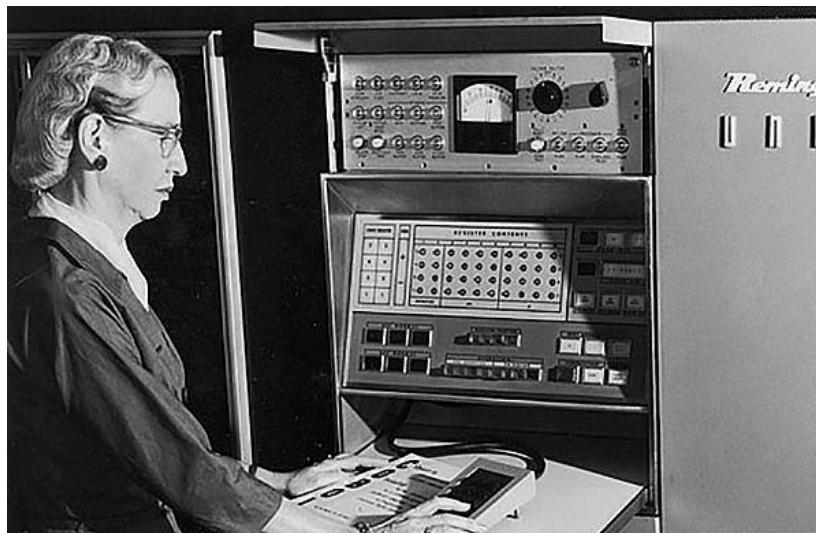
Details



$+ \alpha$

“Sharpened” $\alpha=2$

=



Filtering – Sharpening

Image



Details



$+ \alpha$

“Sharpened” $\alpha=0$

=

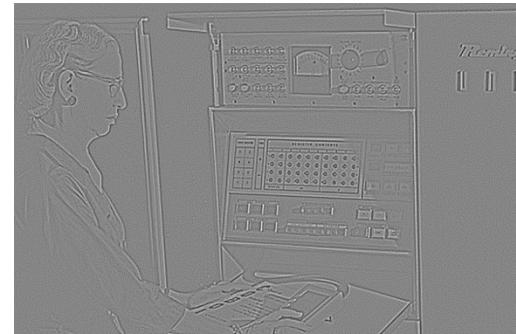


Filtering – Extreme Sharpening

Image



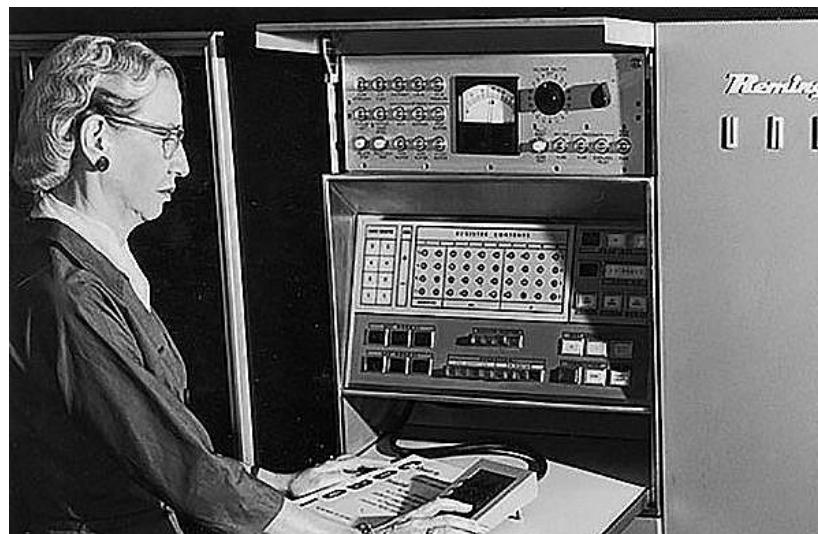
Details



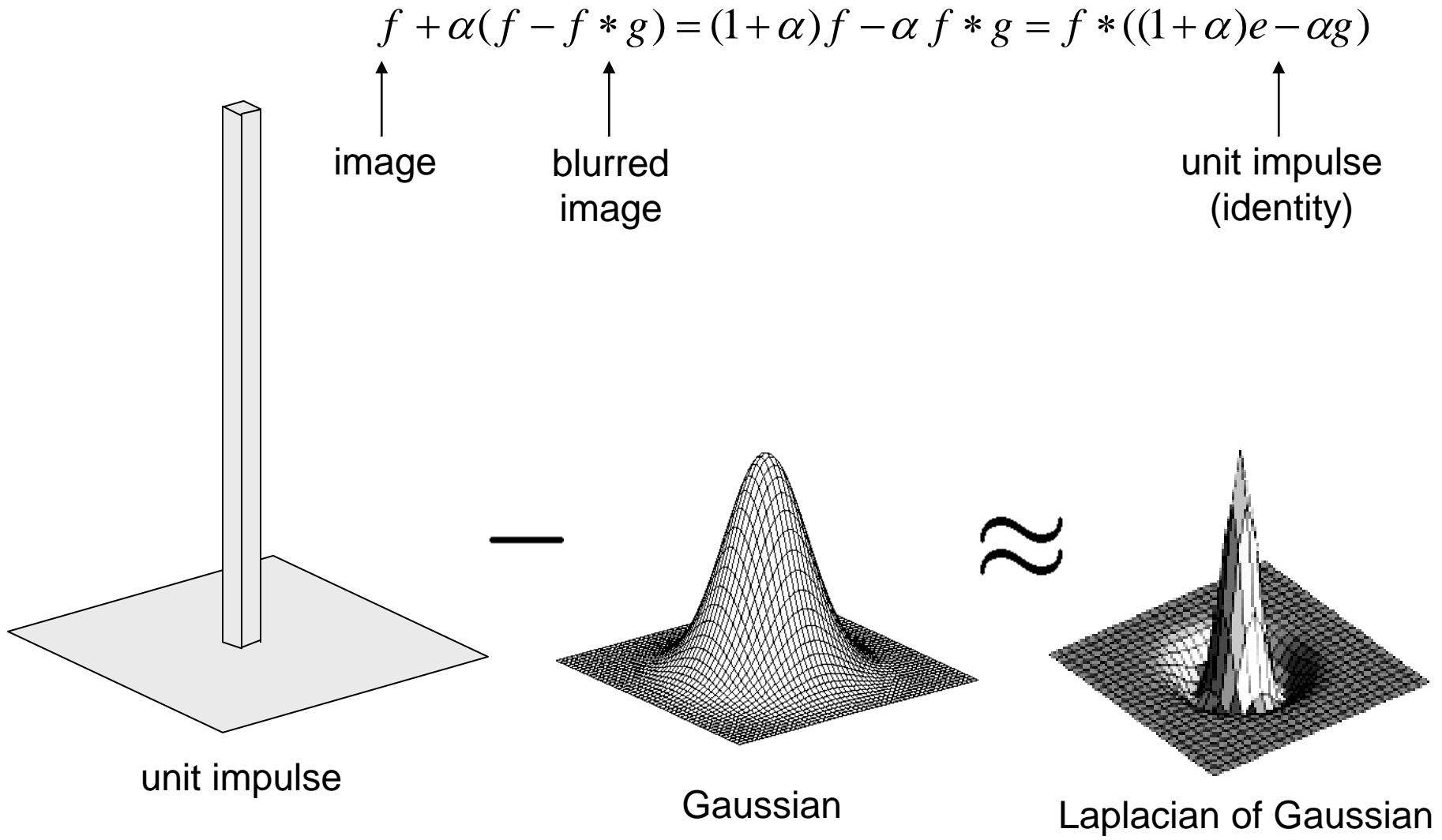
$+ \alpha$

“Sharpened” $\alpha=10$

=



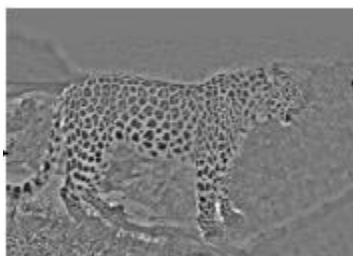
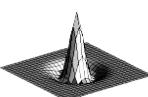
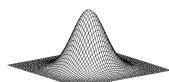
Unsharp mask filter



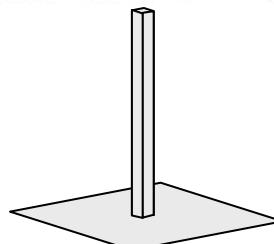
Application: Hybrid Images

Gaussian Filter

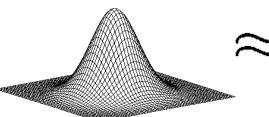
A. Oliva, A. Torralba, P.G. Schyns,
[“Hybrid Images,” SIGGRAPH 2006](#)



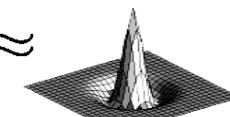
Laplacian Filter



unit impulse

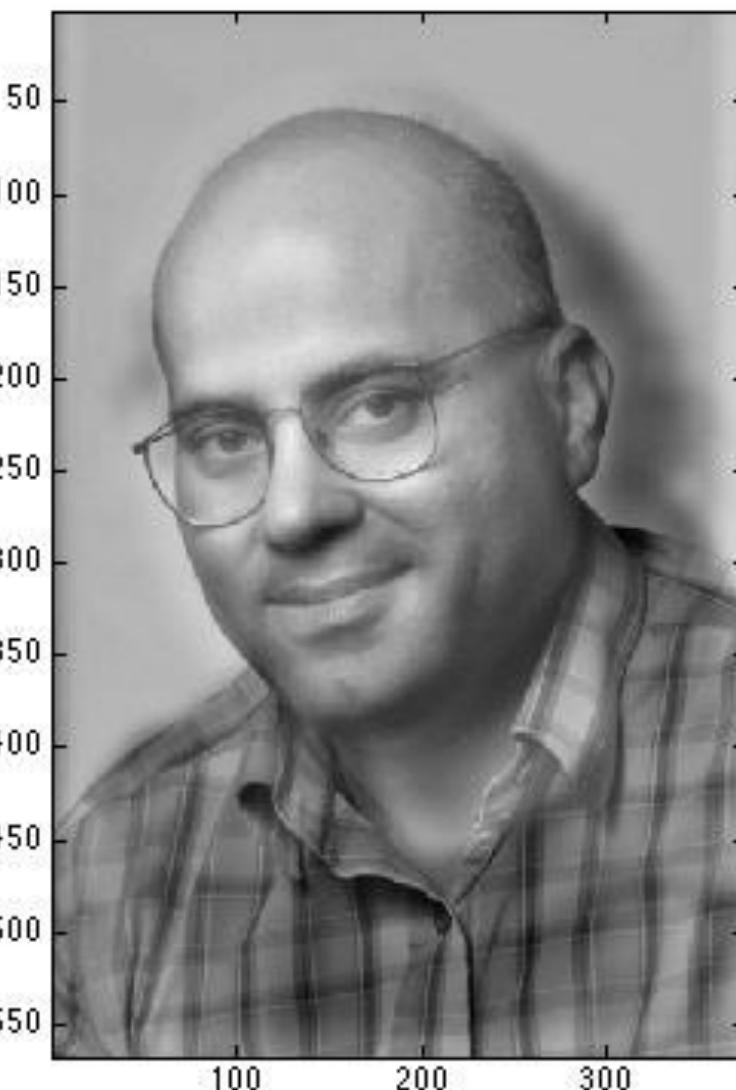


Gaussian



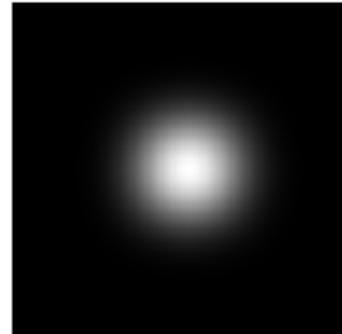
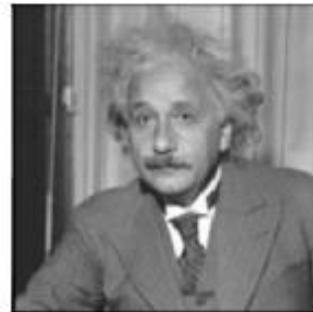
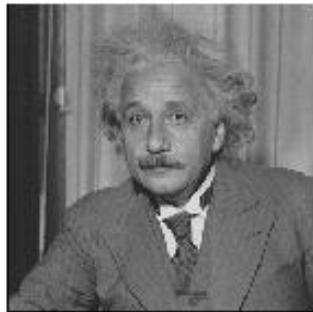
Laplacian of Gaussian

Prof. Jitendros Papadimalik

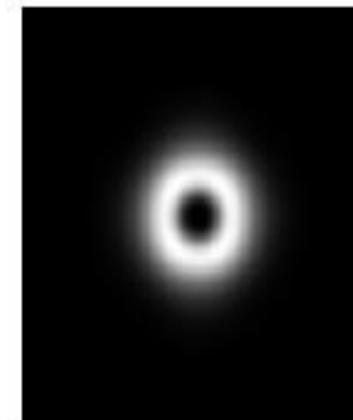
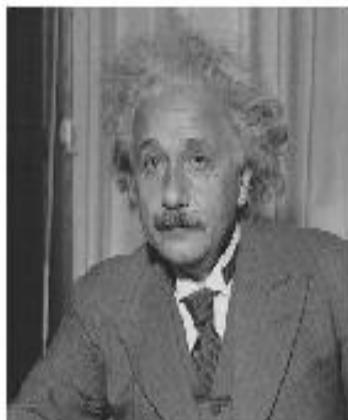


Low-pass, Band-pass, High-pass filters

low-pass:



High-pass / band-pass:

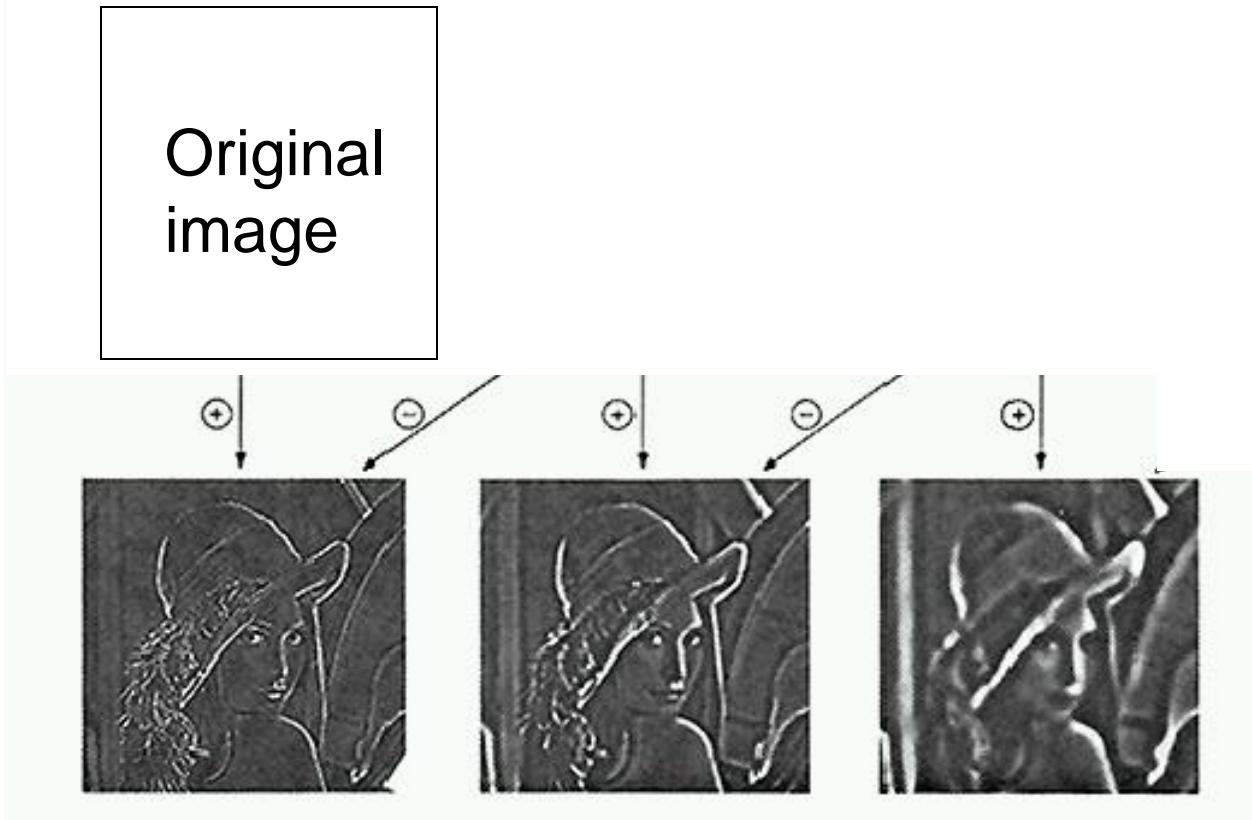


Band-pass filtering

Gaussian Pyramid (low-pass images)



Laplacian Pyramid (Burt and Adelson, 83)

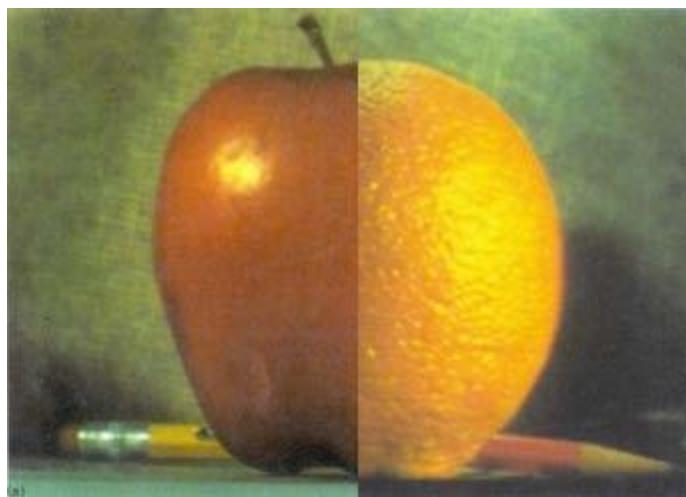
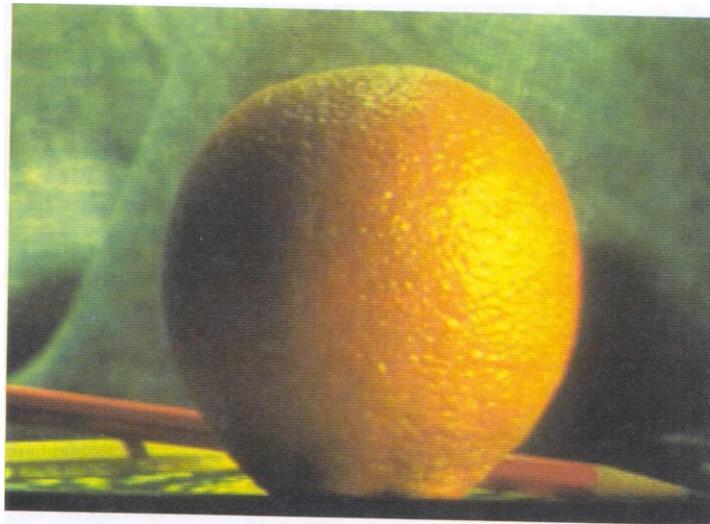


How can we reconstruct (collapse) this pyramid into the original image?

Cut and Paste Blending

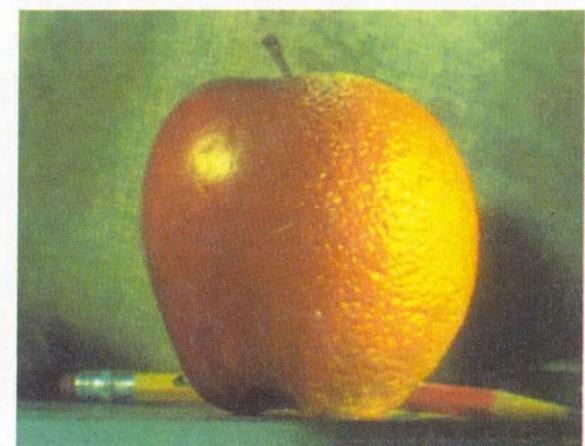
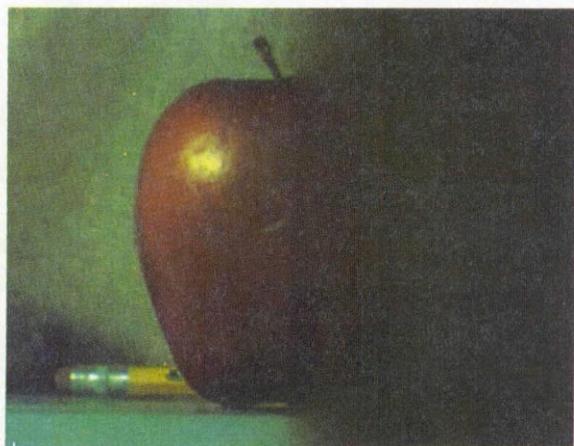
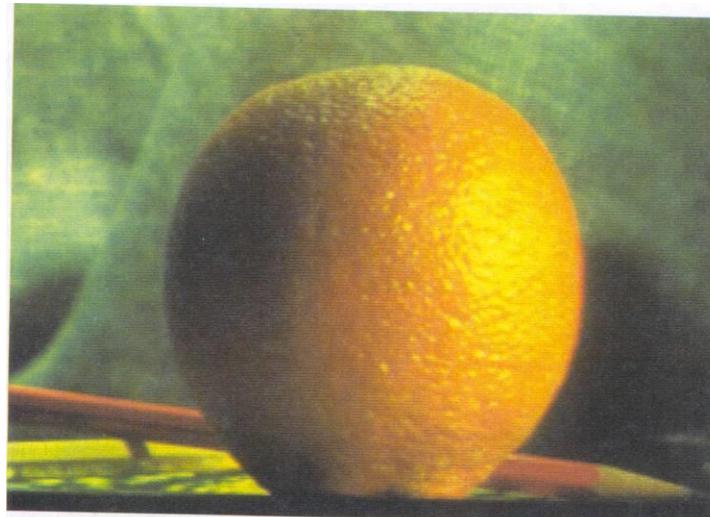
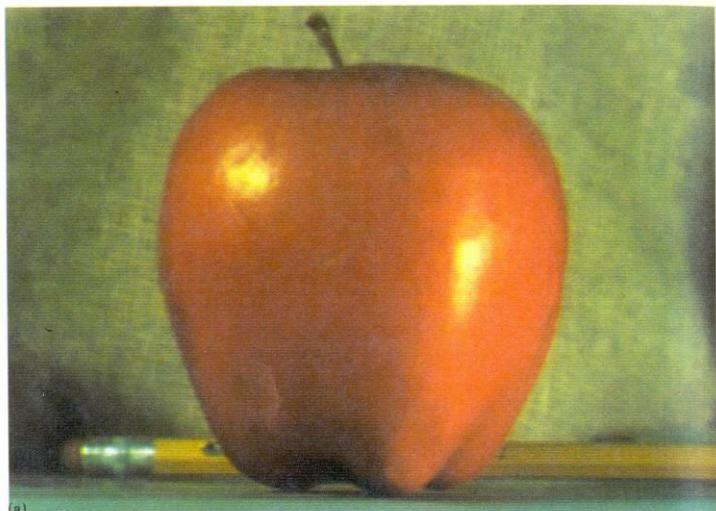


(a)

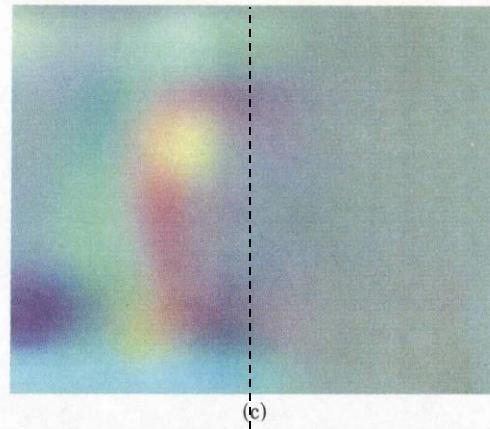


(a)

Pyramid Blending

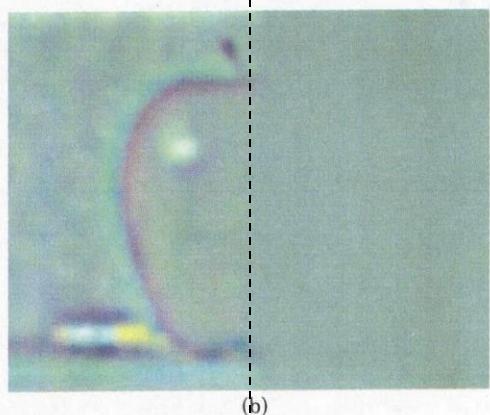


laplacian
level
4



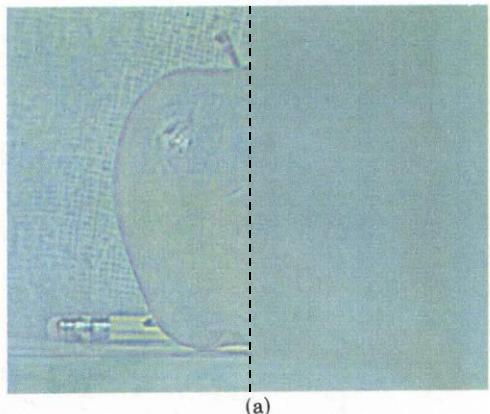
(c)

laplacian
level
2



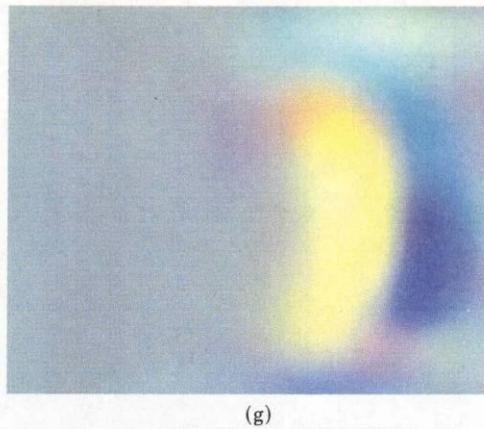
(b)

laplacian
level
0

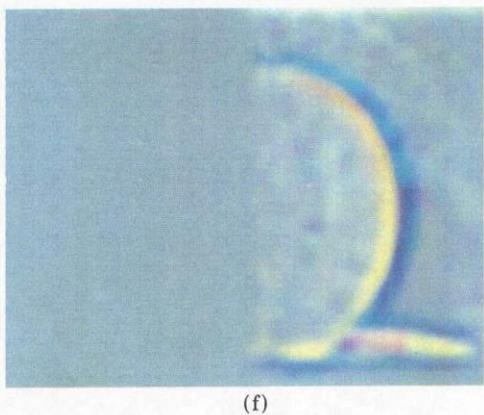


(a)

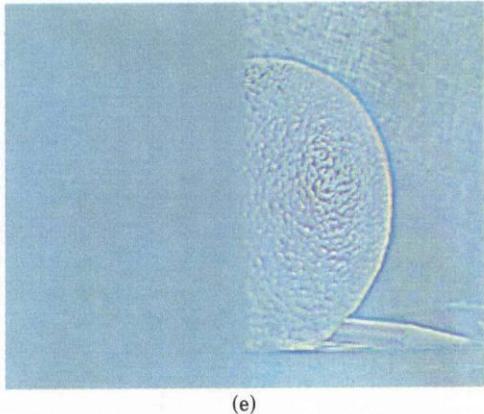
left pyramid



(g)

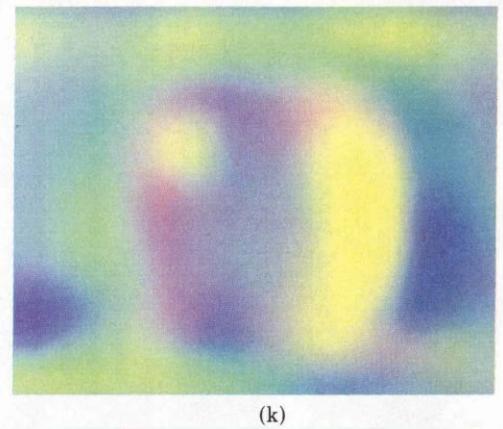


(f)

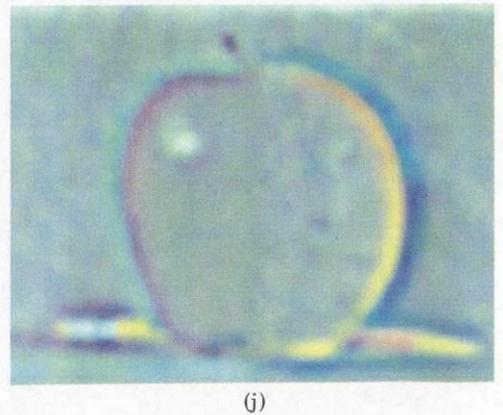


(e)

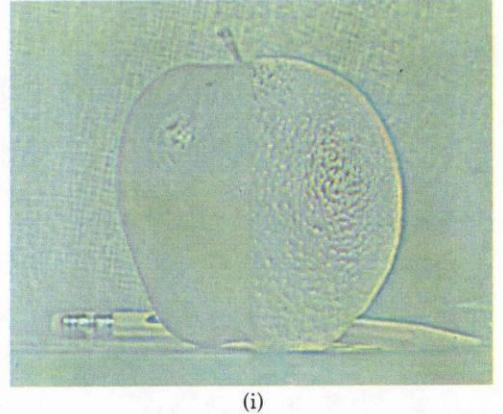
right pyramid



(k)



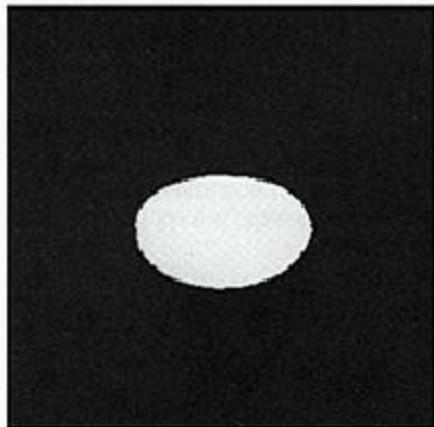
(j)



(i)

blended pyramid

Blending Regions

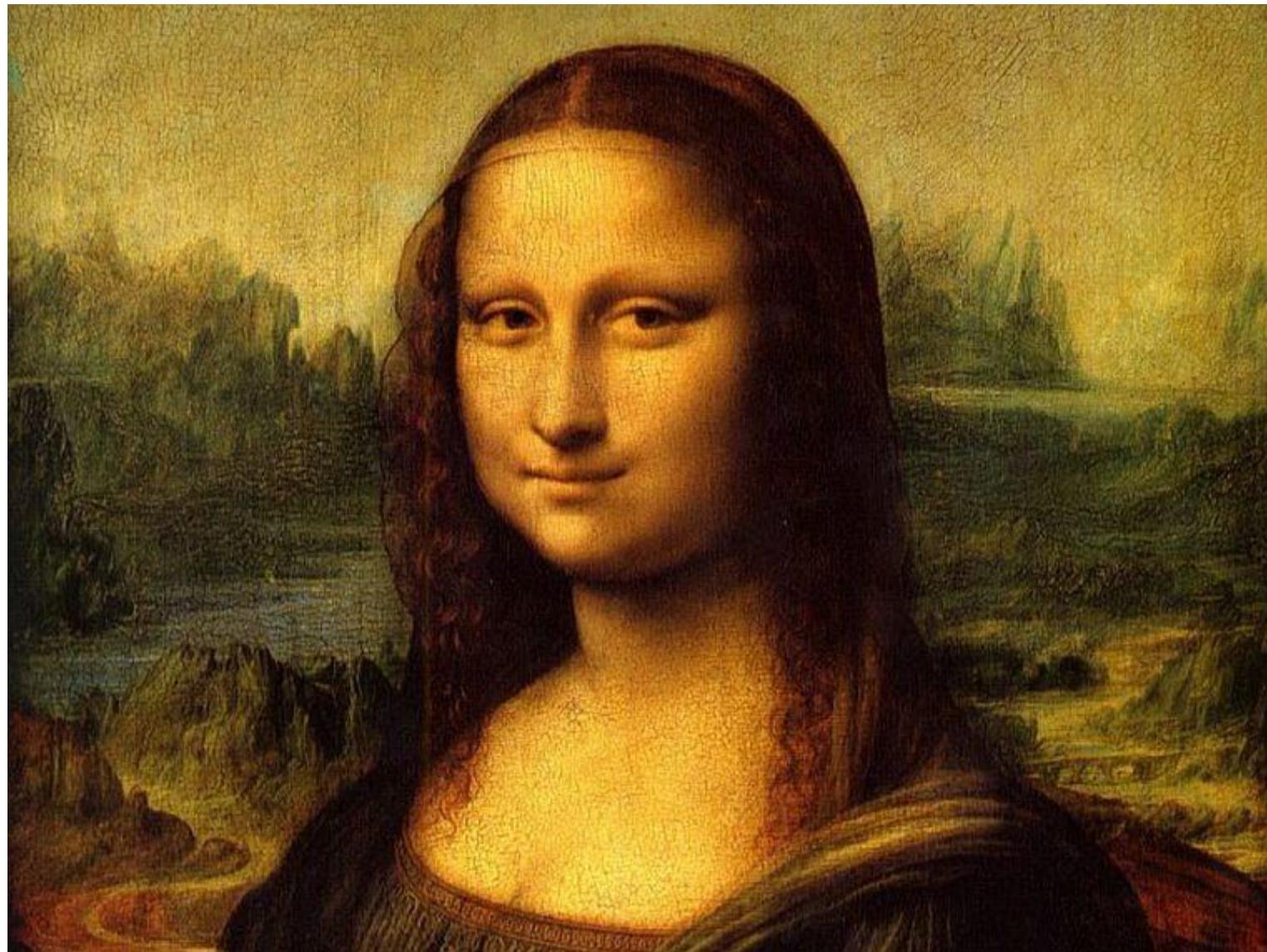


Results from previous class

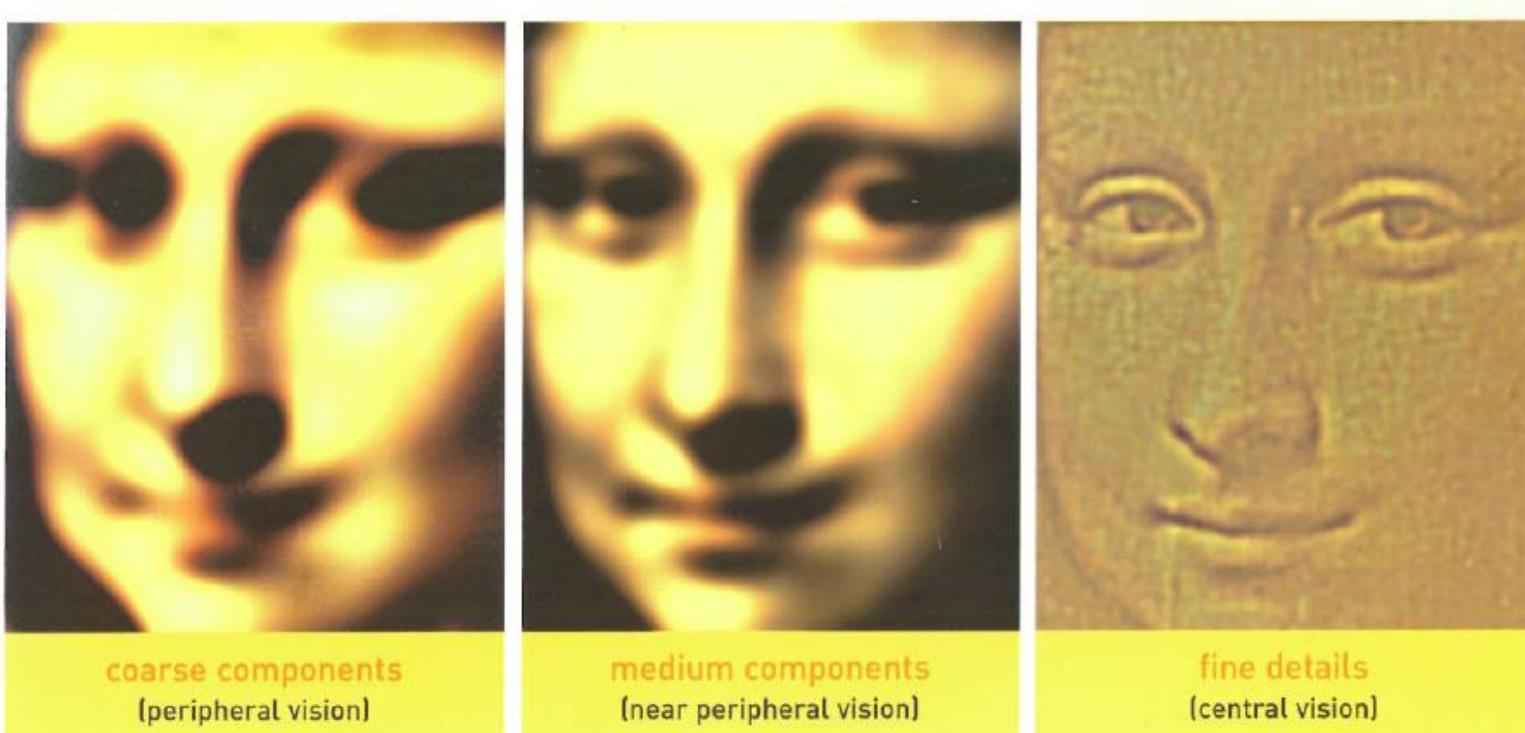


© Chris Cameron

Da Vinci and The Laplacian Pyramid



Da Vinci and The Laplacian Pyramid



Leonardo playing with peripheral vision

Taking derivative by convolution

Partial derivatives with convolution

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

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

For discrete data, we can approximate using finite differences:

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

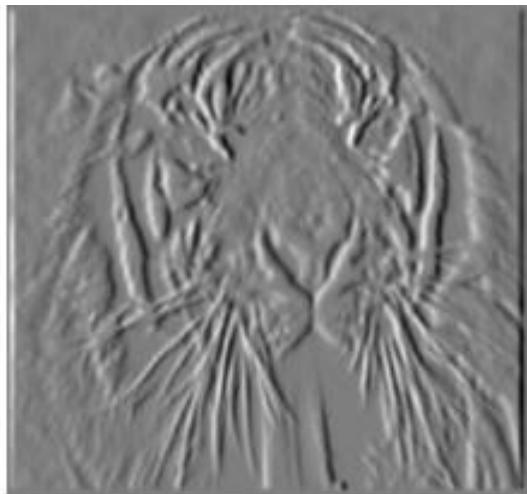
To implement above as convolution, what would be the associated filter?

Partial derivatives of an image



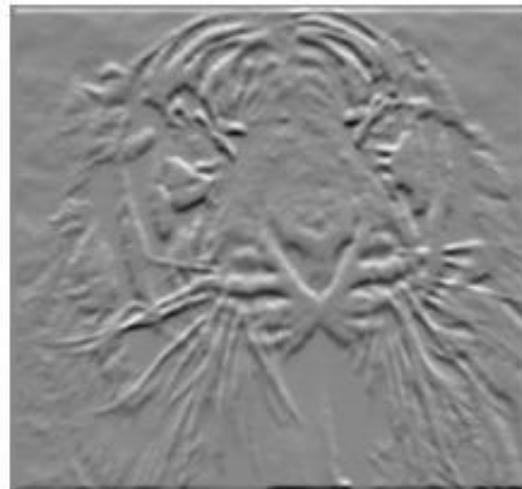
$$\frac{\partial f(x, y)}{\partial x}$$

-1	1
----	---



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

-1	or	1
1		-1



Which shows changes with respect to x?

Finite difference filters

Other approximations of derivative filters exist:

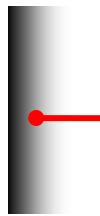
Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

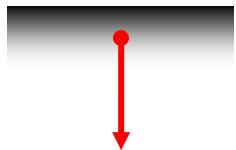
Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

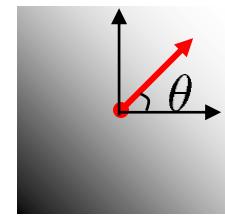
Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Image gradient

The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$


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


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


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

The gradient points in the direction of most rapid increase in intensity

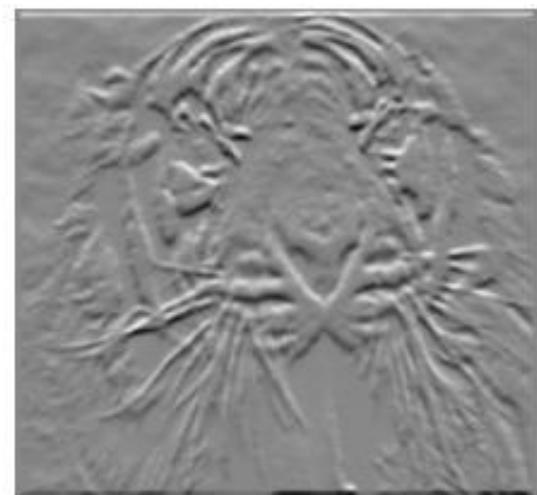
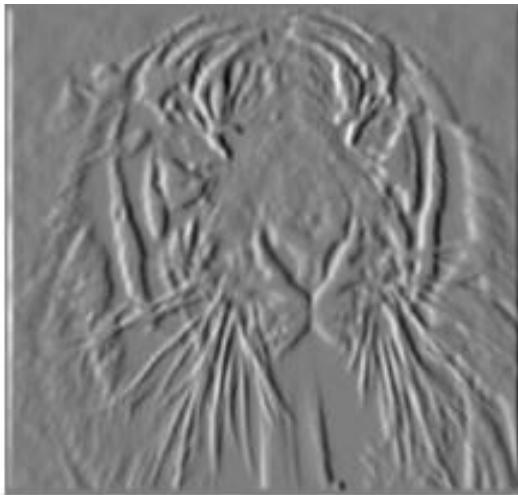
- How does this direction relate to the direction of the edge?

The gradient direction 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

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

Image Gradient



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

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

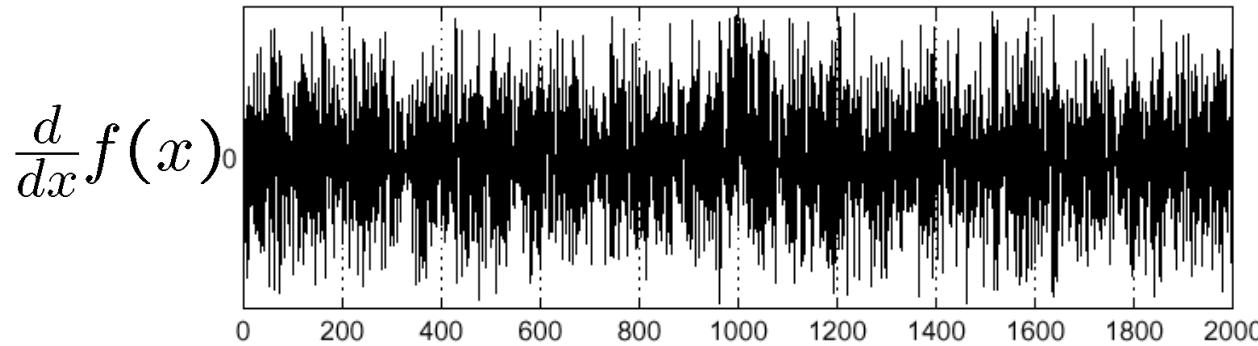
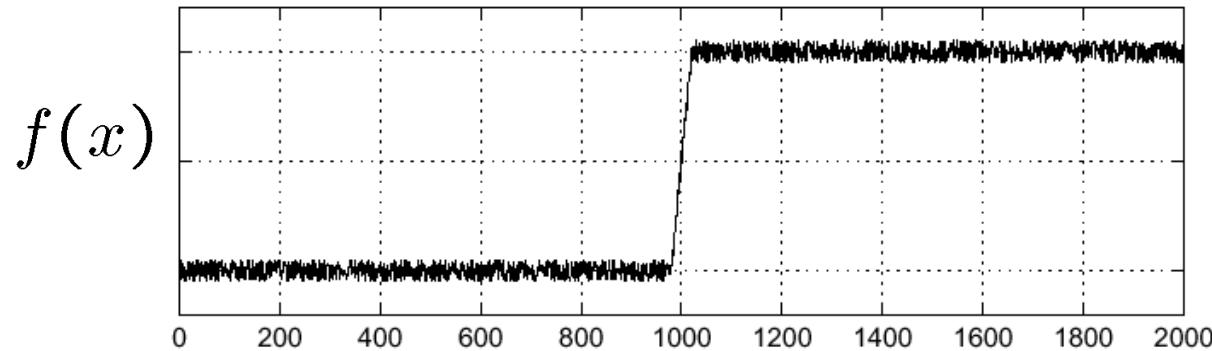


$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Effects of noise

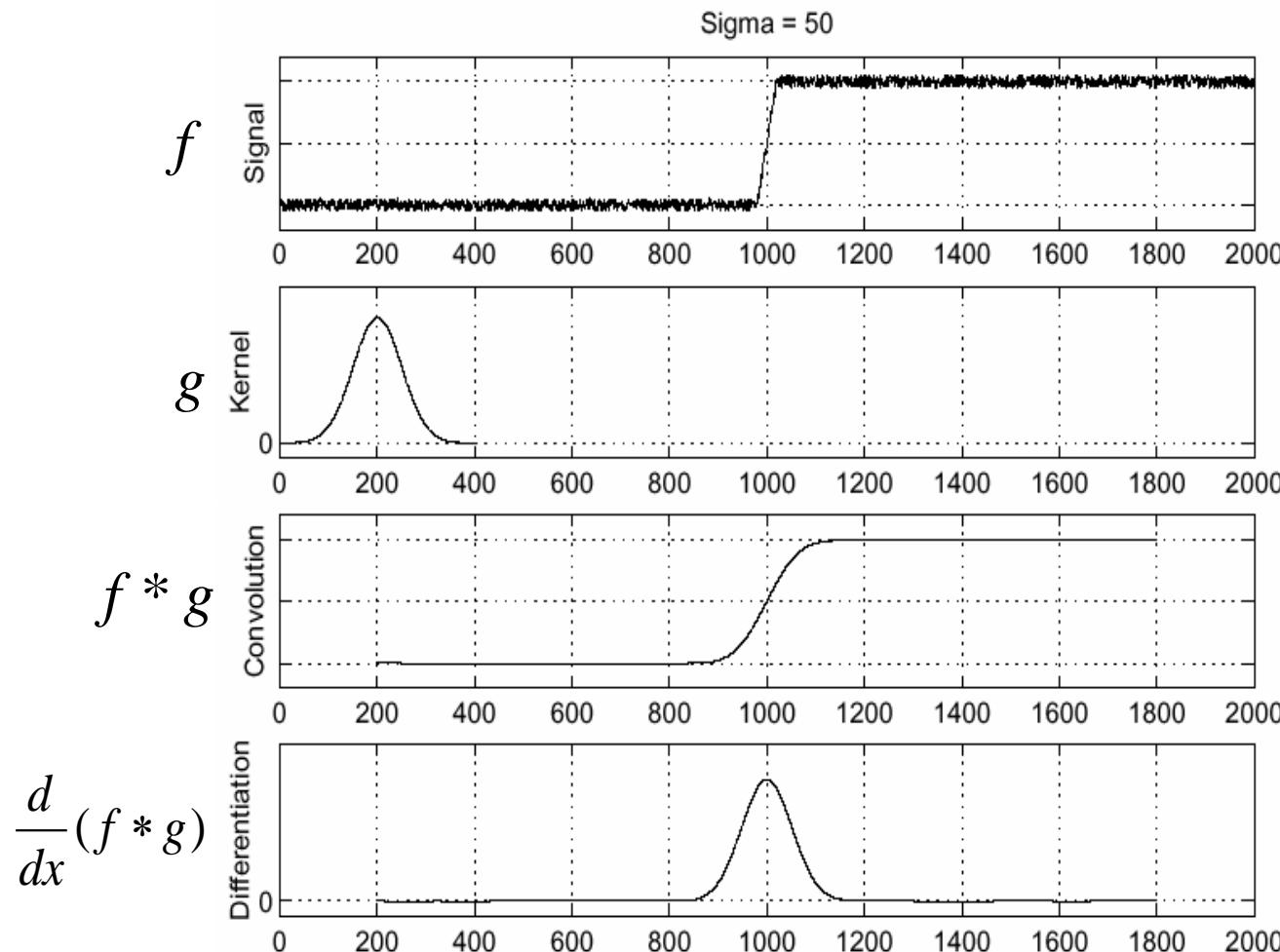
Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal



Where is the edge?

Solution: smooth first



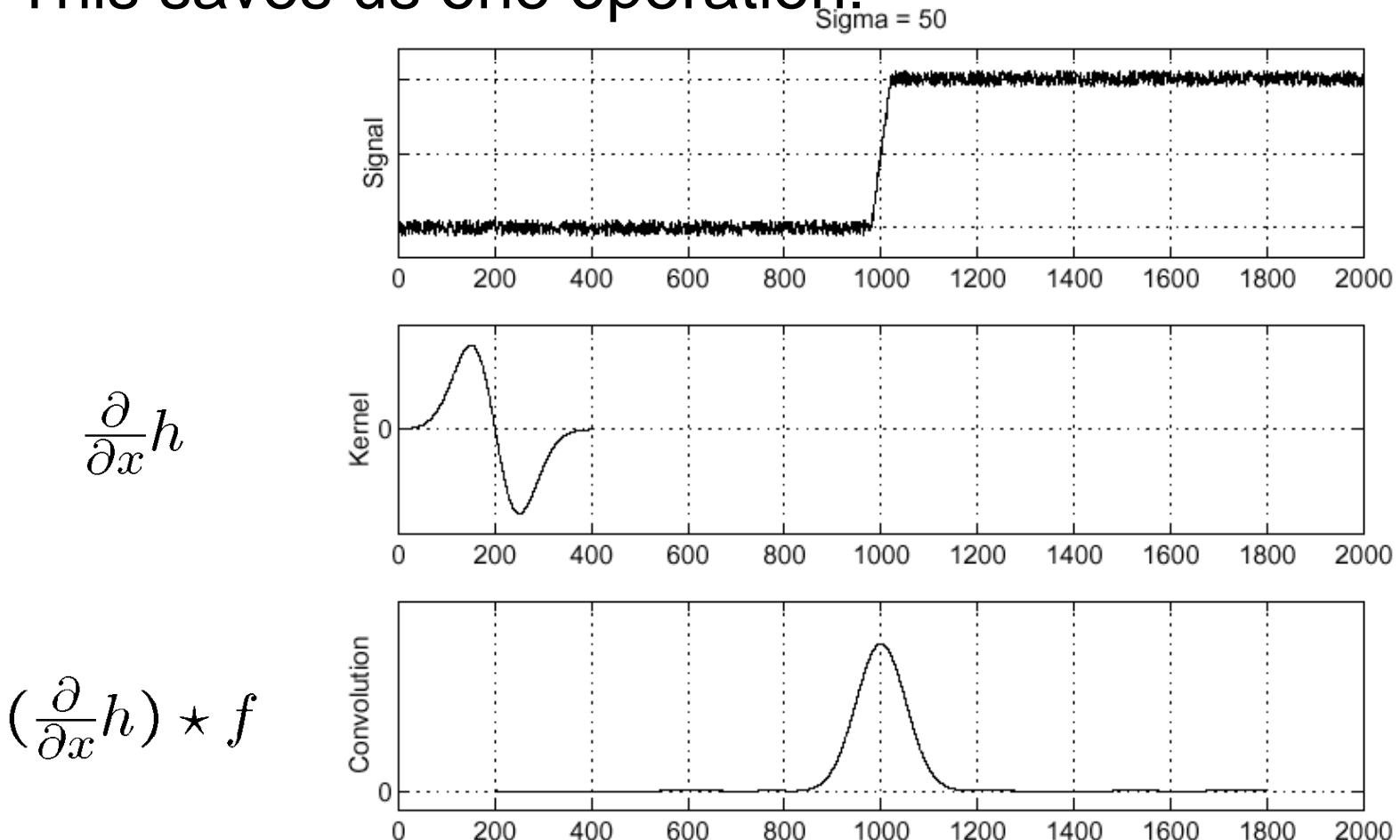
- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

Source: S. Seitz

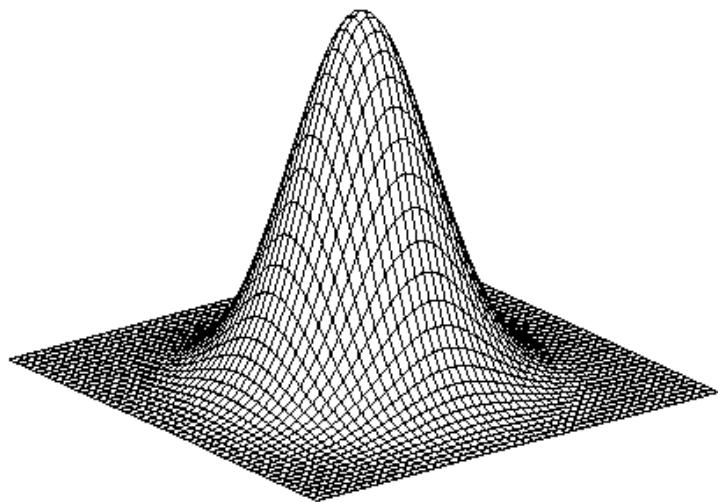
Derivative theorem of convolution

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

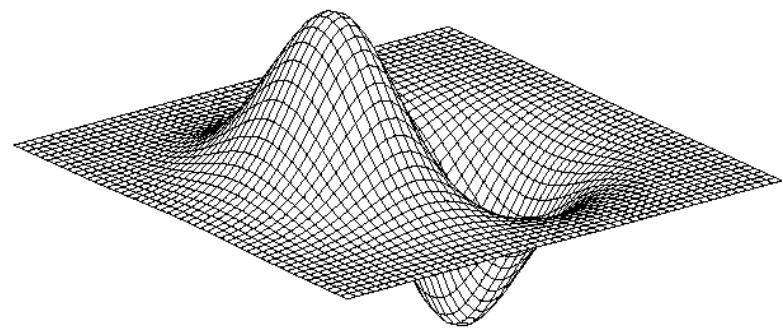
This saves us one operation:



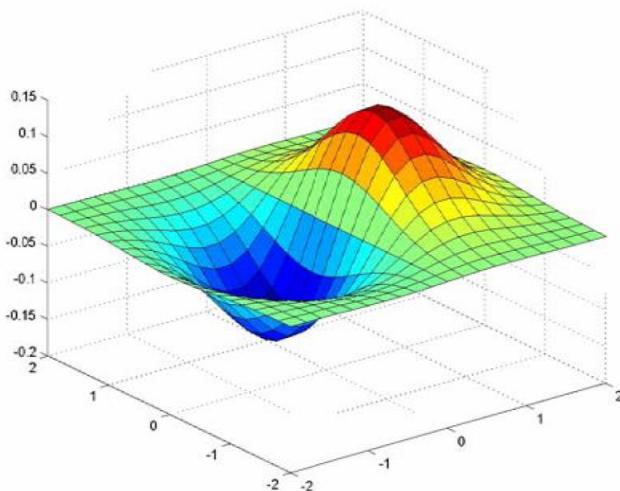
Derivative of Gaussian filter



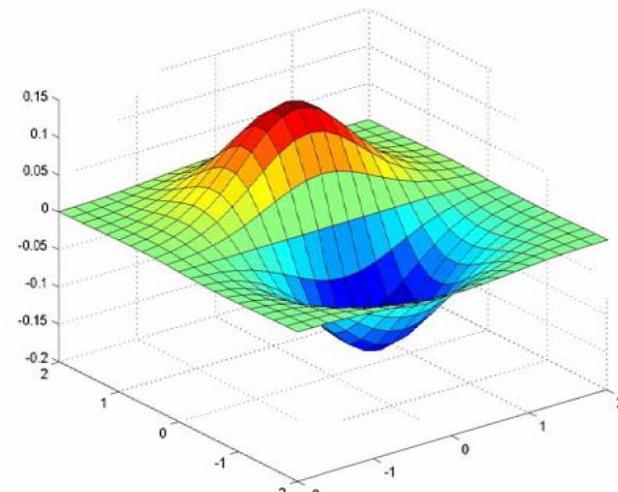
$$* [1 \ -1] =$$



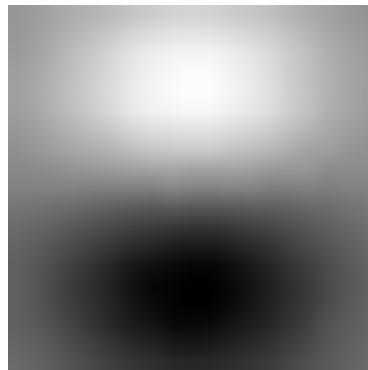
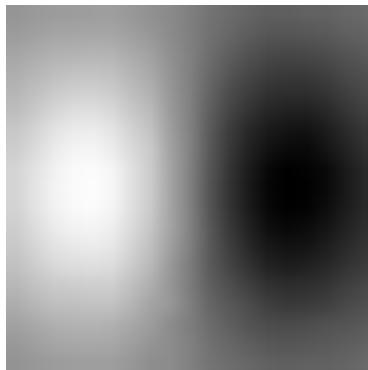
Derivative of Gaussian filter



x-direction



y-direction

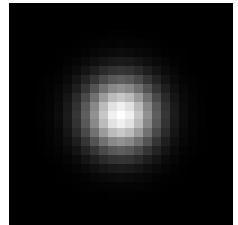


Which one finds horizontal/vertical edges?

Review: Smoothing vs. derivative filters

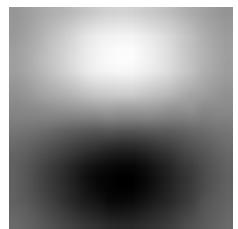
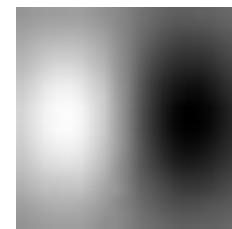
Smoothing filters

- Gaussian: remove “high-frequency” components; “low-pass” filter
- Can the values of a smoothing filter be negative?
- What should the values sum to?
 - **One**: constant regions are not affected by the filter



Derivative filters

- Derivatives of Gaussian
- Can the values of a derivative filter be negative?
- What should the values sum to?
 - **Zero**: no response in constant regions
- High absolute value at points of high contrast

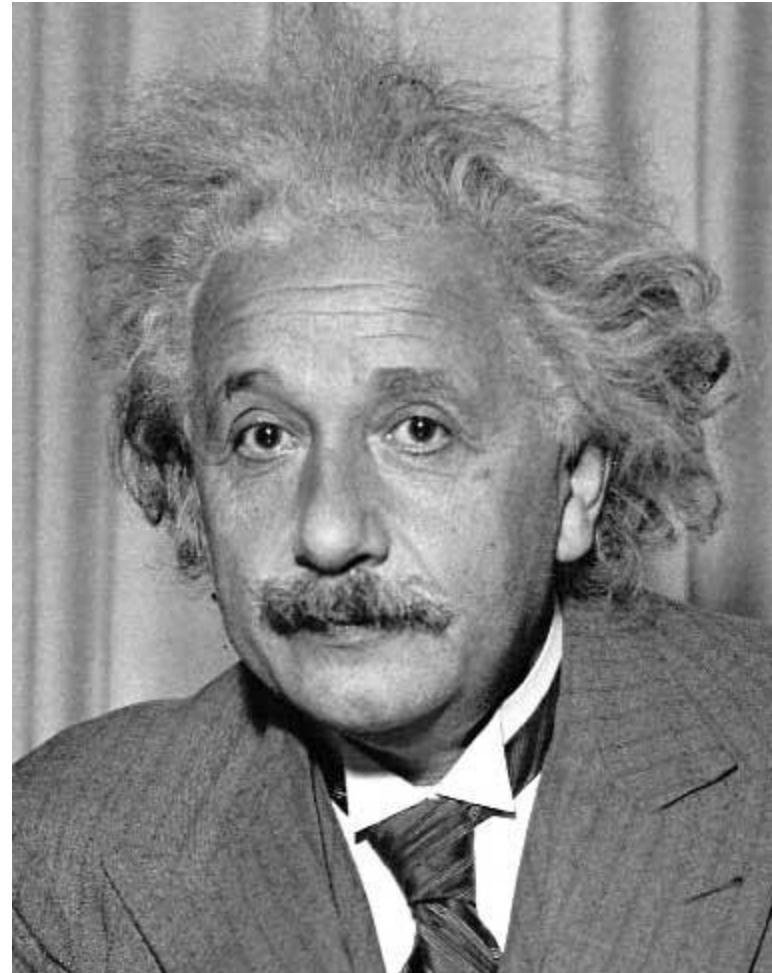


Template matching

Goal: find  in image

Main challenge: What is a good similarity or distance measure between two patches?

- Correlation
- Zero-mean correlation
- Sum Square Difference
- Normalized Cross Correlation

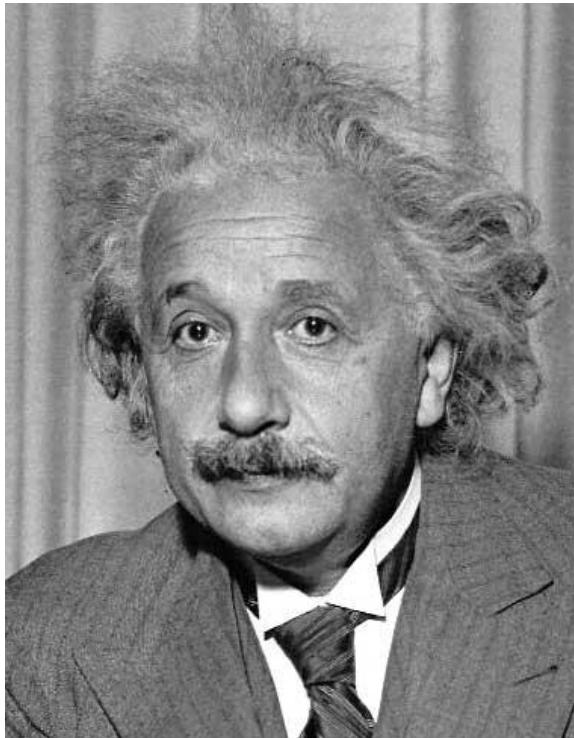


Matching with filters

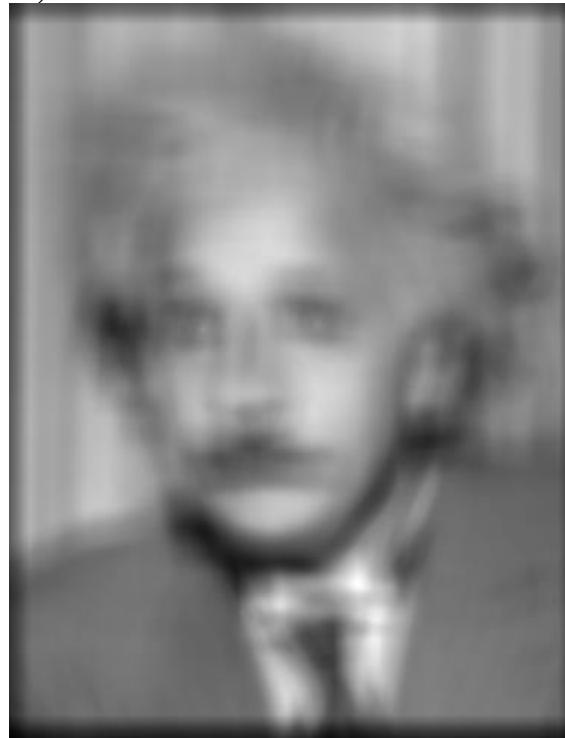
Goal: find  in image

Method 0: filter the image with eye patch

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



Input



Filtered Image

f = image
g = filter

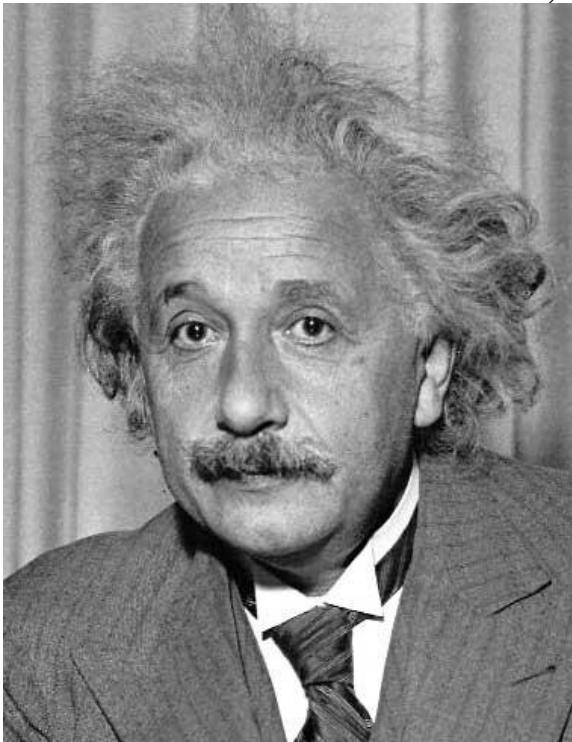
What went wrong?

Matching with filters

Goal: find  in image

Method 1: filter the image with zero-mean eye

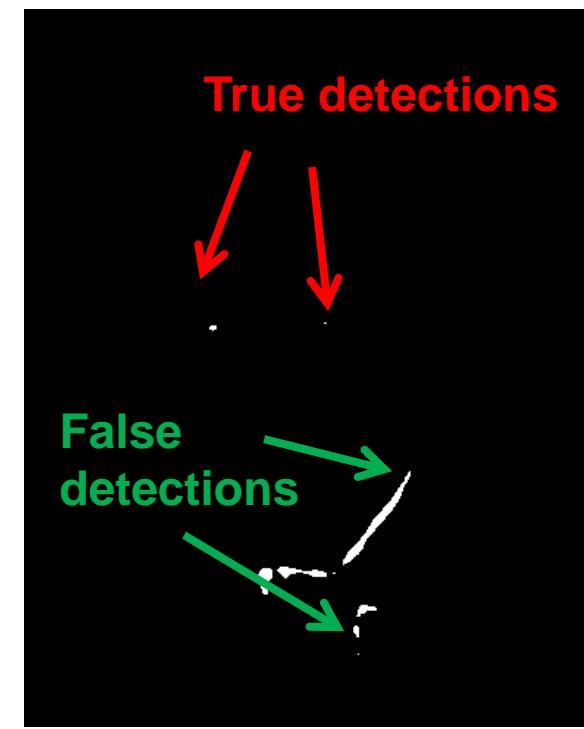
$$h[m, n] = \sum_{k, l} (f[k, l] - \bar{f}) \underbrace{(g[m + k, n + l])}_{\text{mean of } f}$$



Input



Filtered Image (scaled)



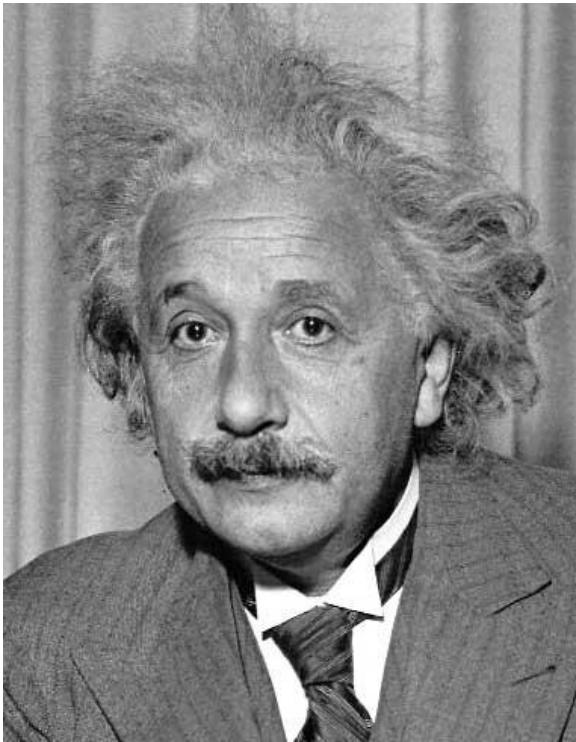
Thresholded Image

Matching with filters

Goal: find  in image

Method 2: SSD

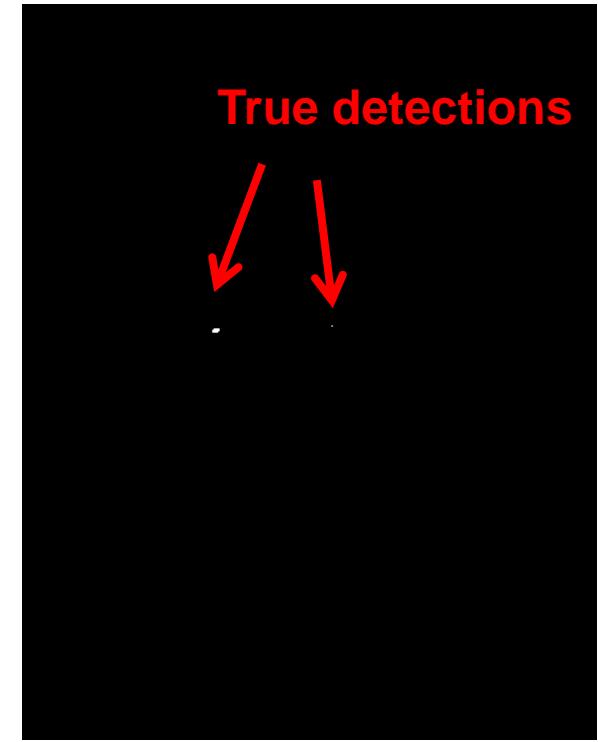
$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$



Input



1 - $\sqrt{\text{SSD}}$



Thresholded Image

True detections

Matching with filters

Can SSD be implemented with linear filters?

$$h[m, n] = \sum_{k, l} (g[k, l] - f[m + k, n + l])^2$$

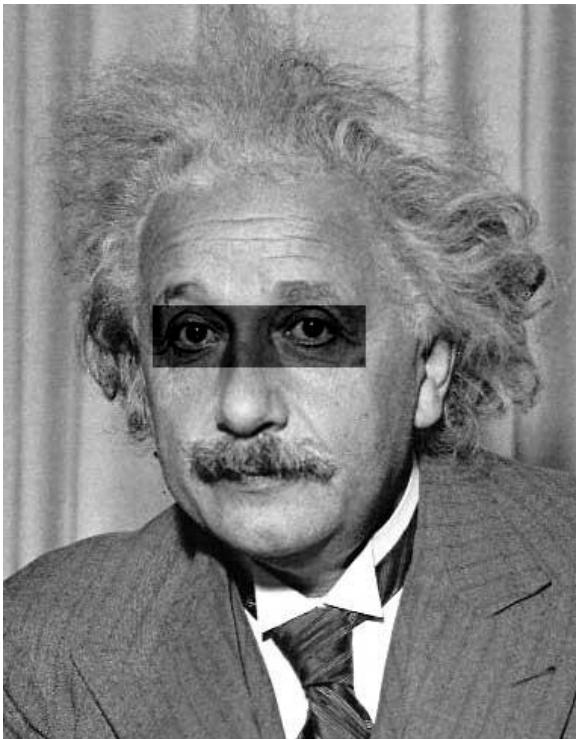
Matching with filters

Goal: find  in image

What's the potential downside of SSD?

Method 2: SSD

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$



Input



1 - $\sqrt{\text{SSD}}$

Matching with filters

Goal: find  in image

Method 3: Normalized cross-correlation

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

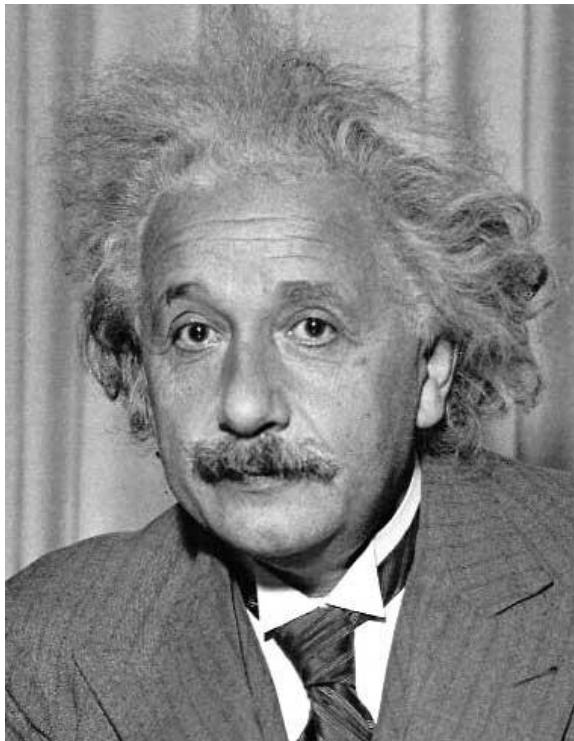
mean template
↓
 $\sum_{k,l} (g[k, l] - \bar{g})(f[m+k, n+l] - \bar{f}_{m,n})$

mean image patch
↓

Matching with filters

Goal: find  in image

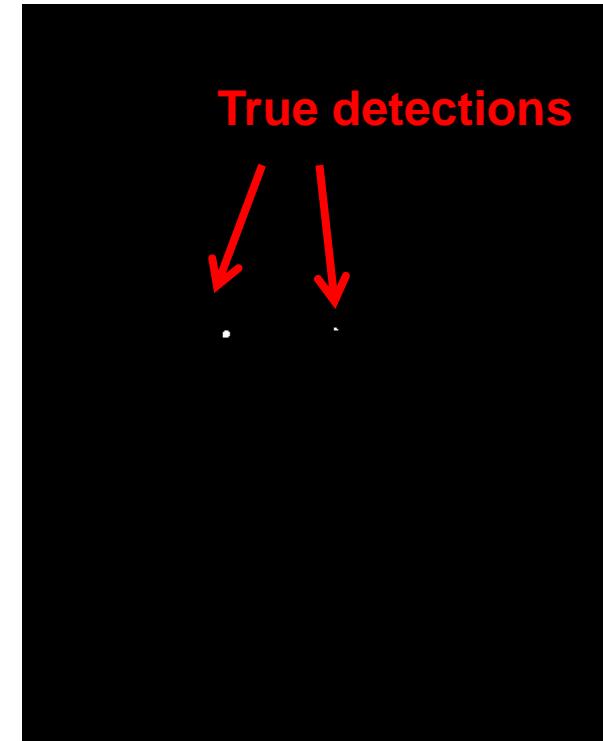
Method 3: Normalized cross-correlation



Input



Normalized X-Correlation

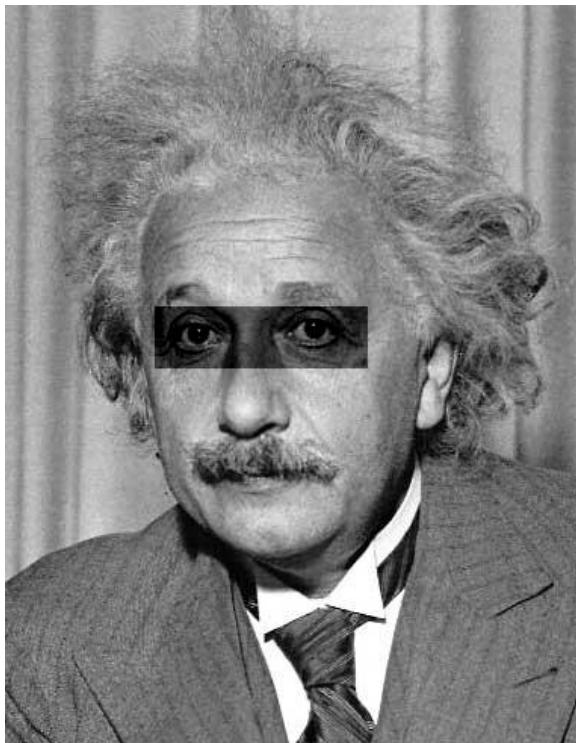


Thresholded Image

Matching with filters

Goal: find  in image

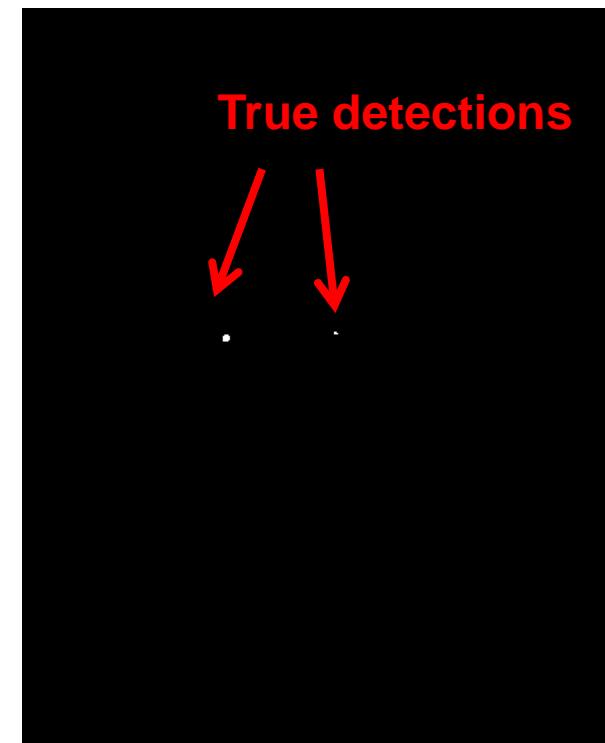
Method 3: Normalized cross-correlation



Input



Normalized X-Correlation



Thresholded Image

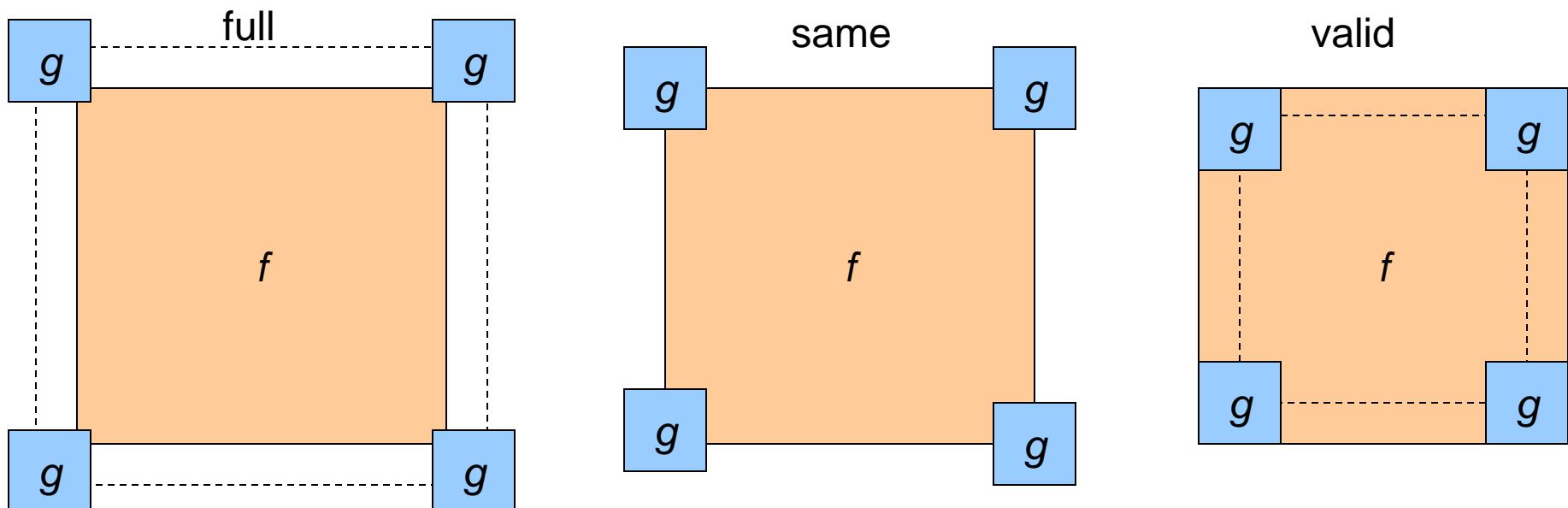
True detections

Practical matters

What is the size of the output?

MATLAB: `filter2(g, f, shape)` or `conv2(g,f,shape)`

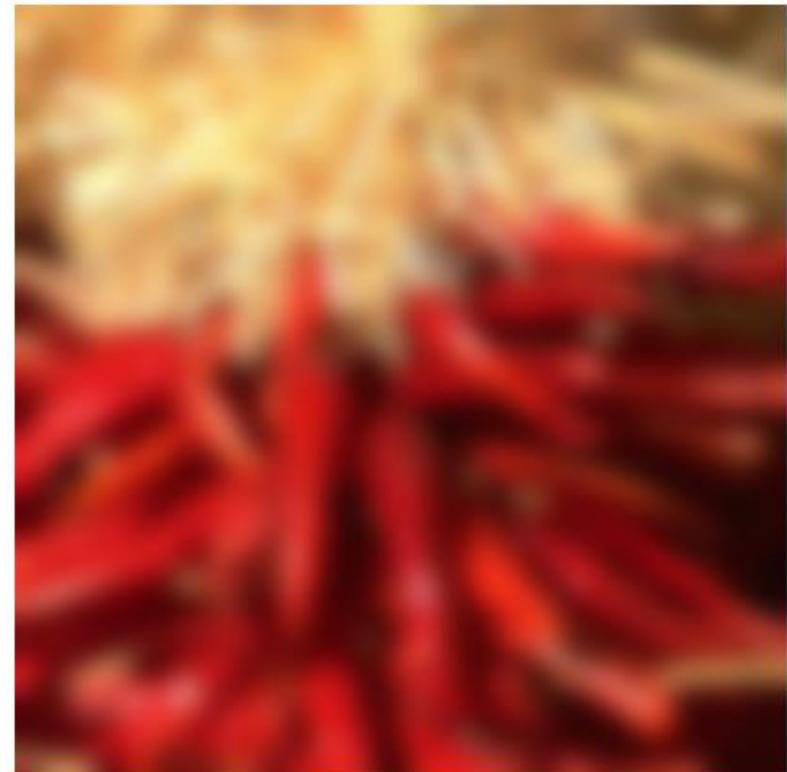
- `shape = 'full'`: output size is sum of sizes of `f` and `g`
- `shape = 'same'`: output size is same as `f`
- `shape = 'valid'`: output size is difference of sizes of `f` and `g`



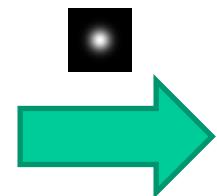
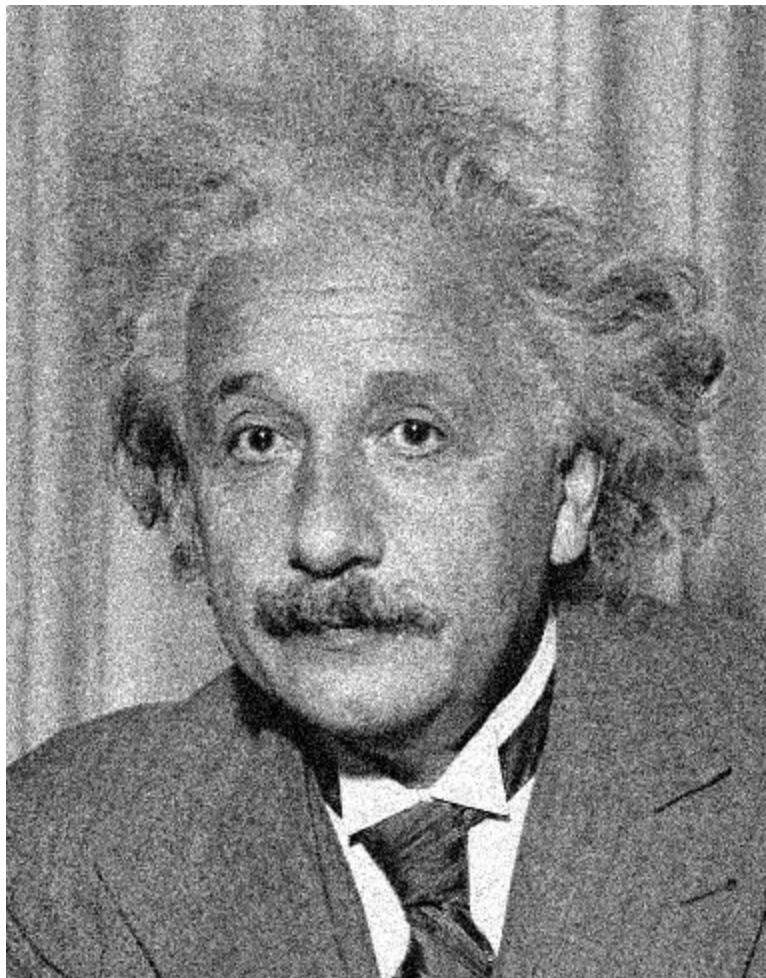
Practical matters

What about near the edge?

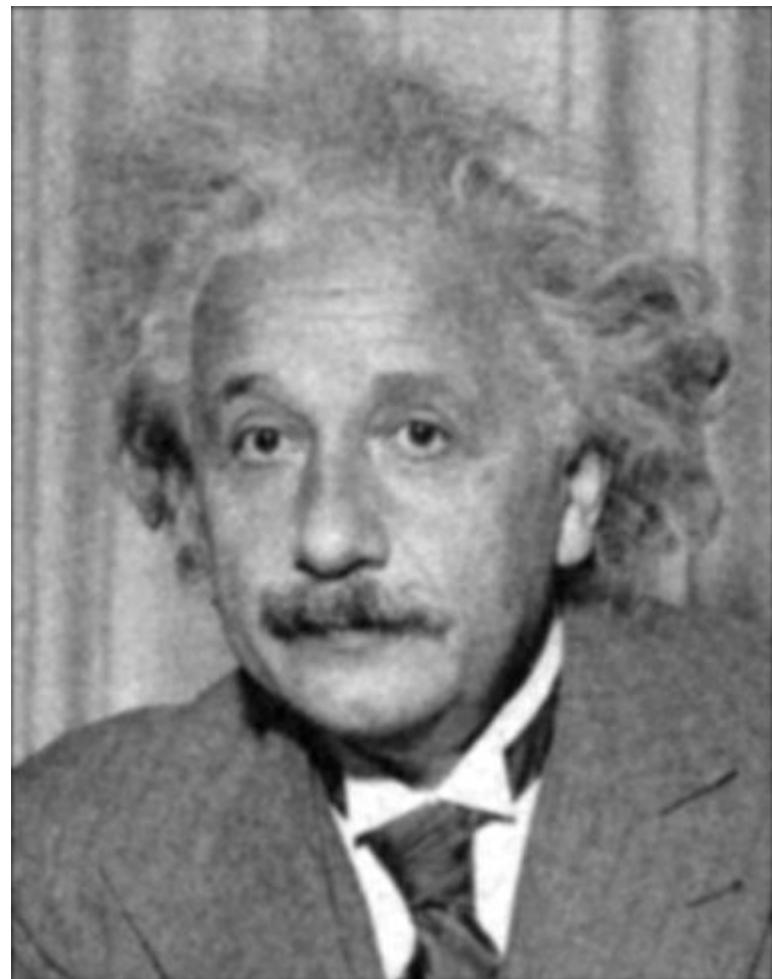
- the filter window falls off the edge of the image
- need to extrapolate
- methods:
 - clip filter (black)
 - wrap around
 - copy edge
 - reflect across edge



Non-linear Filters



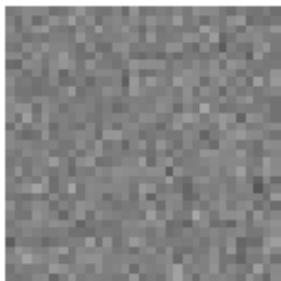
Gaussian
Filter



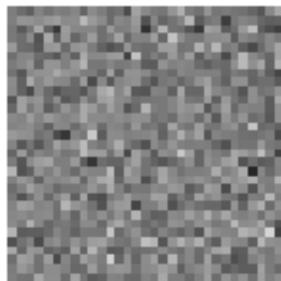
Additive Gaussian Noise

Reducing Gaussian noise

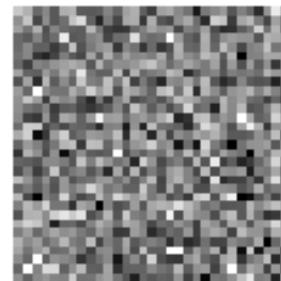
$\sigma=0.05$



$\sigma=0.1$



$\sigma=0.2$



no
smoothing



$\sigma=1$ pixel



$\sigma=2$ pixels



Smoothing with larger standard deviations suppresses noise, but also blurs the image

Reducing salt-and-pepper noise by Gaussian smoothing

3x3



5x5

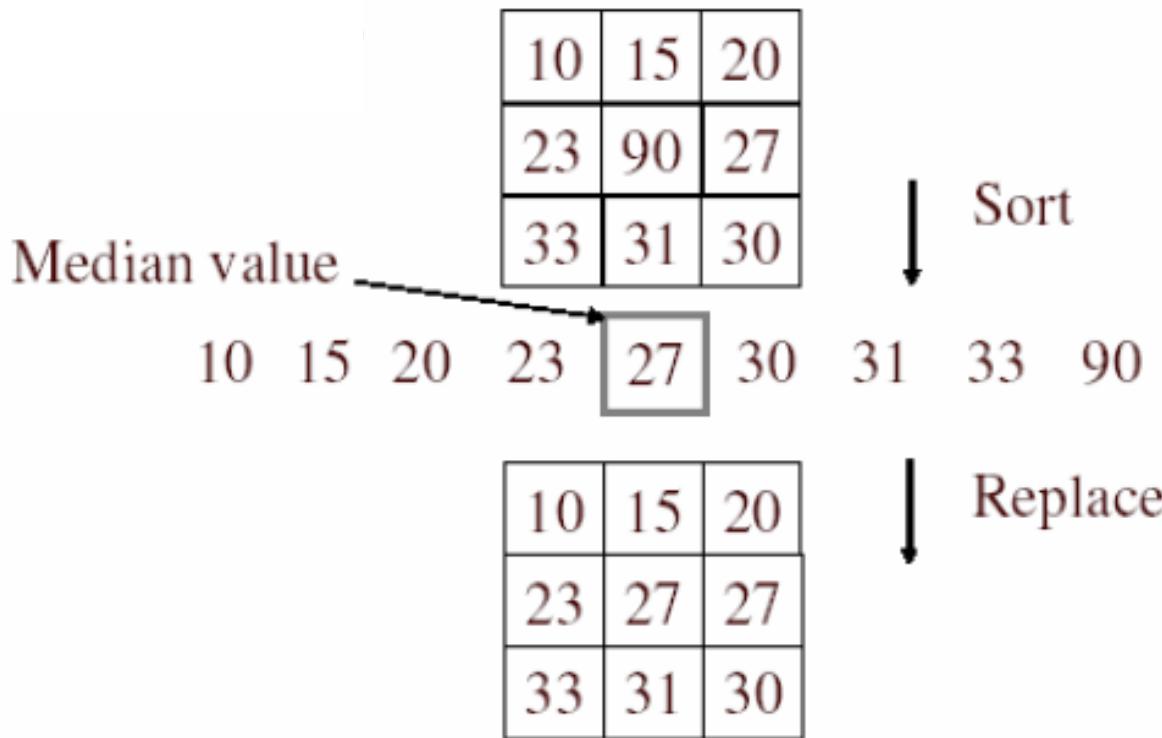


7x7



Alternative idea: Median filtering

A **median filter** operates over a window by selecting the median intensity in the window



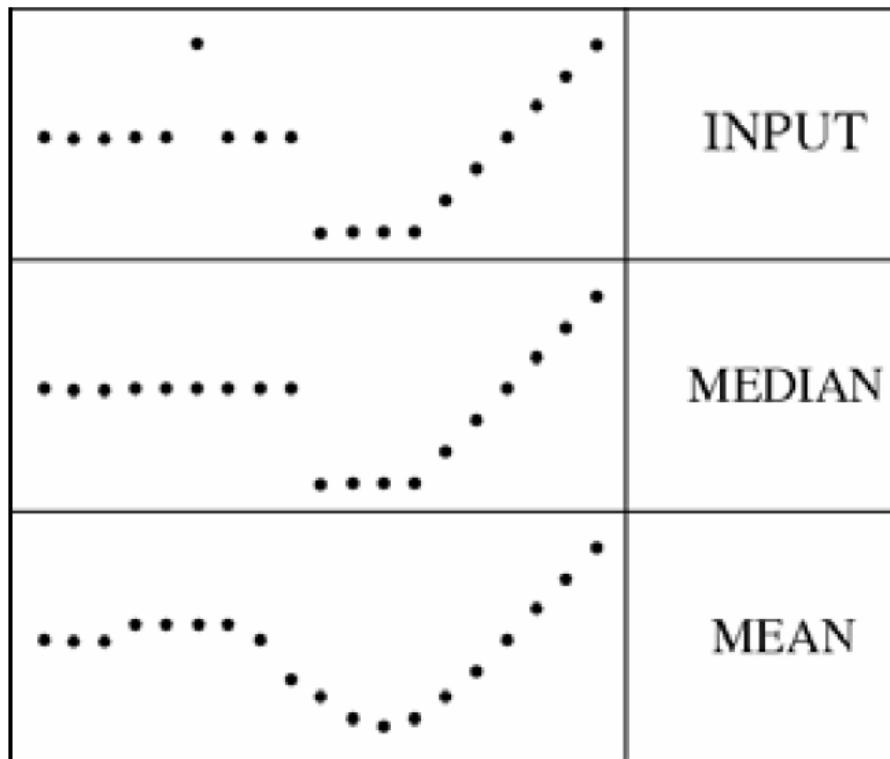
- Is median filtering linear?

Median filter

What advantage does median filtering have over Gaussian filtering?

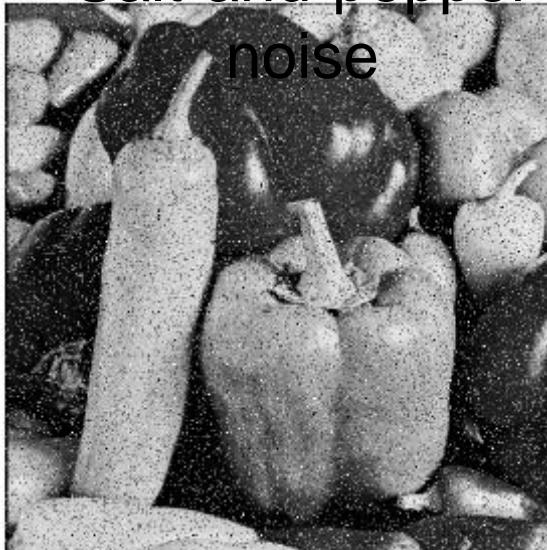
- Robustness to outliers

filters have width 5 :

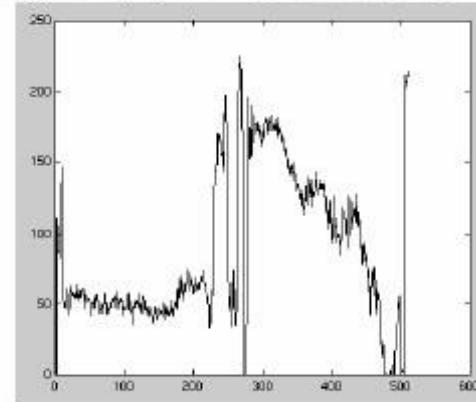
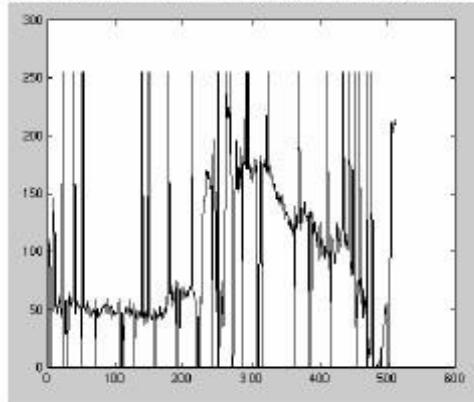


Median filter

Salt-and-pepper
noise



Median filtered



MATLAB: `medfilt2(image, [h w])`

Source: M. Hebert

Median vs. Gaussian filtering

3x3



5x5



7x7



Gaussian

Median



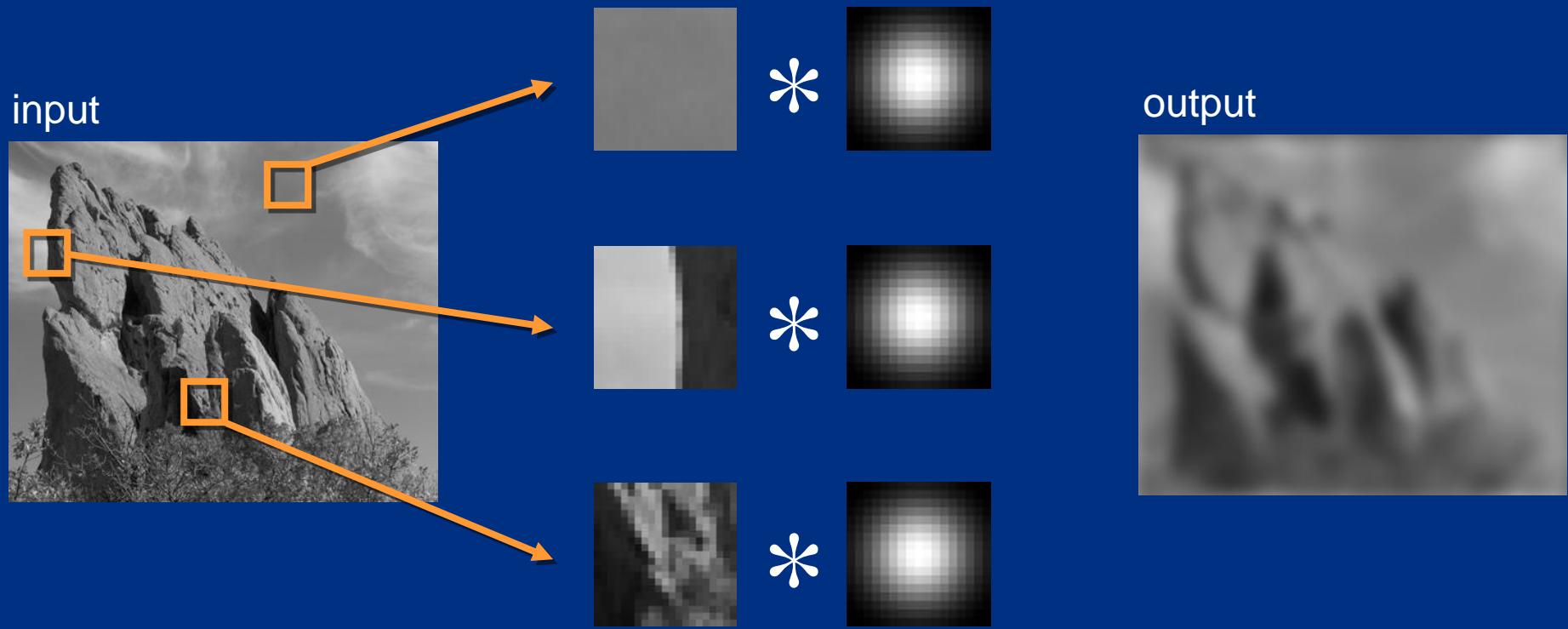
A Gentle Introduction to Bilateral Filtering and its Applications



“Fixing the Gaussian Blur”: the Bilateral Filter

Sylvain Paris – MIT CSAIL

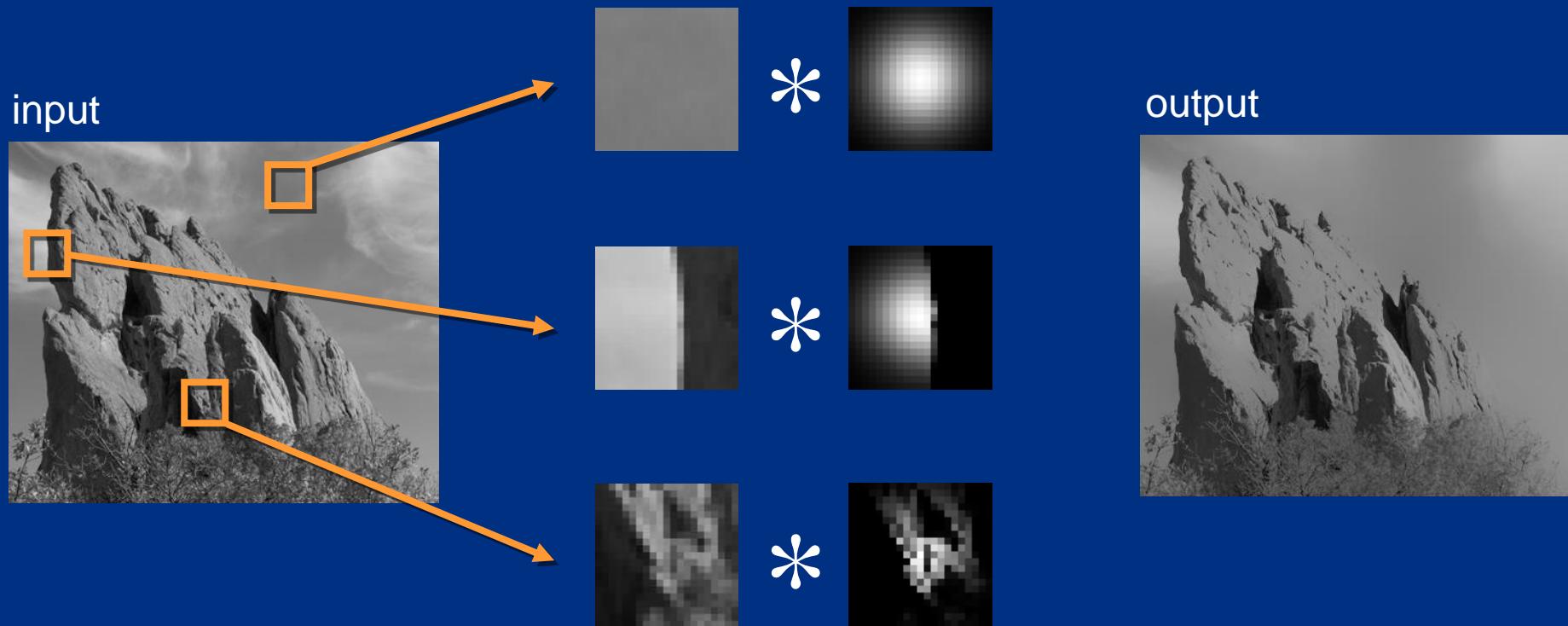
Blur Comes from Averaging across Edges



Same Gaussian kernel everywhere.

Bilateral Filter [Aurich 95, Smith 97, Tomasi 98]

No Averaging across Edges



The kernel shape depends on the image content.

Bilateral Filter Definition: an Additional Edge Term

Same idea: **weighted average of pixels.**

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\| p - q \|) G_{\sigma_r}(|I_p - I_q|) I_q$$

new
not new
new

normalization factor space weight range weight

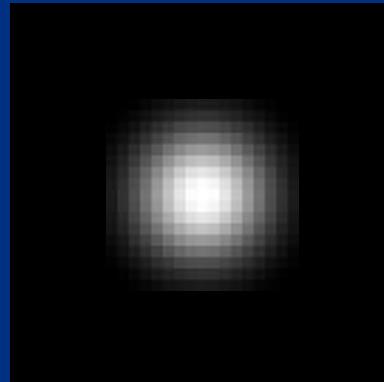
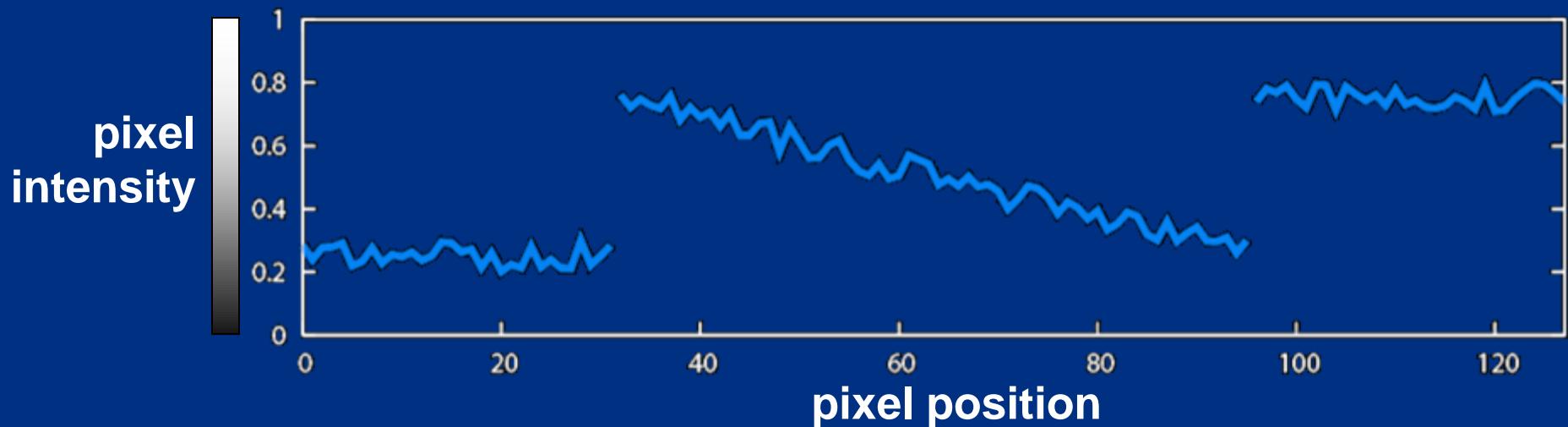


Illustration a 1D Image

- 1D image = line of pixels

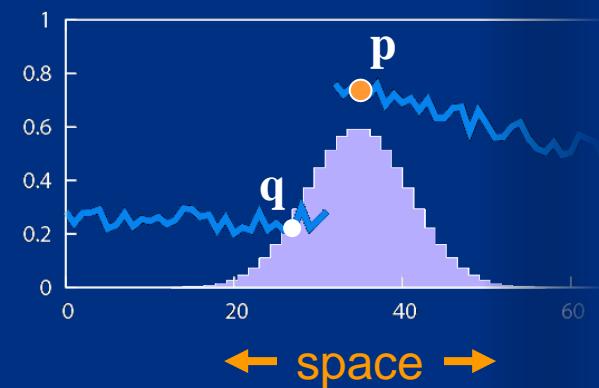


- Better visualized as a plot



Gaussian Blur and Bilateral Filter

Gaussian blur

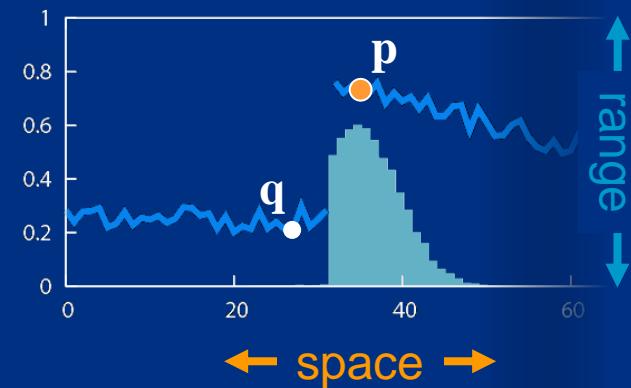


$$GB[I]_p = \sum_{q \in S} G_\sigma(\|p - q\|) I_q$$

space

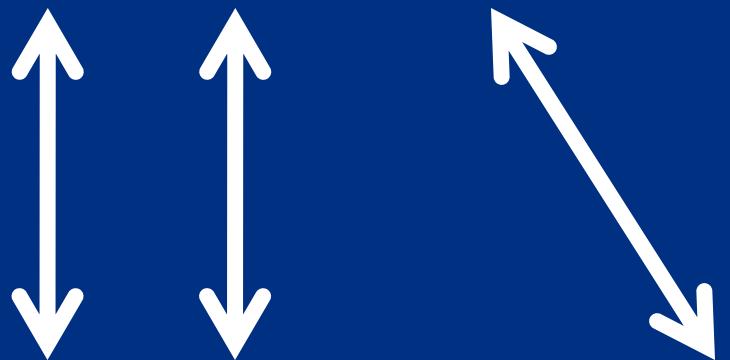
Bilateral filter

[Aurich 95, Smith 97, Tomasi 98]



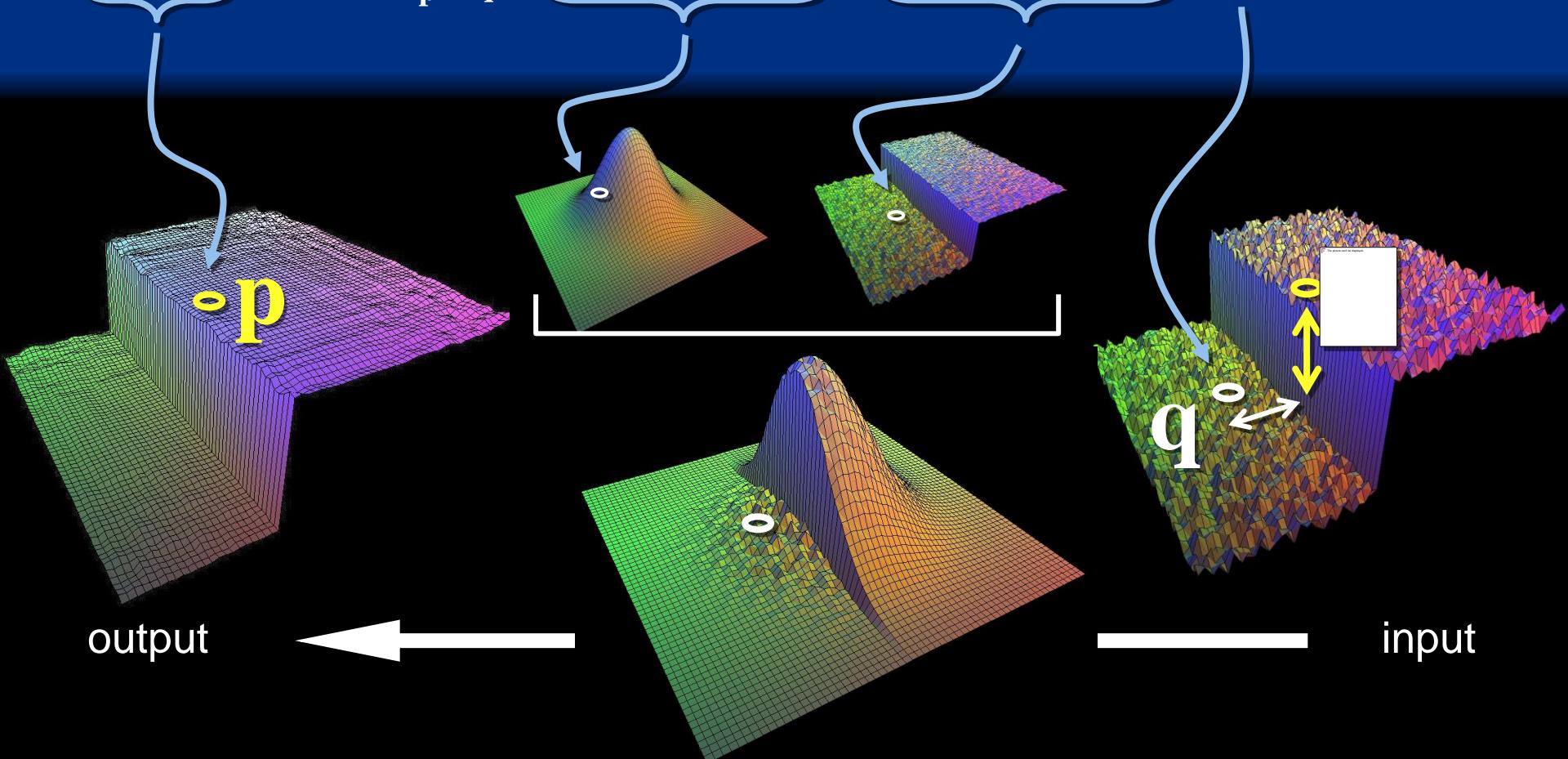
$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

space range
normalization



Bilateral Filter on a Height Field

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$



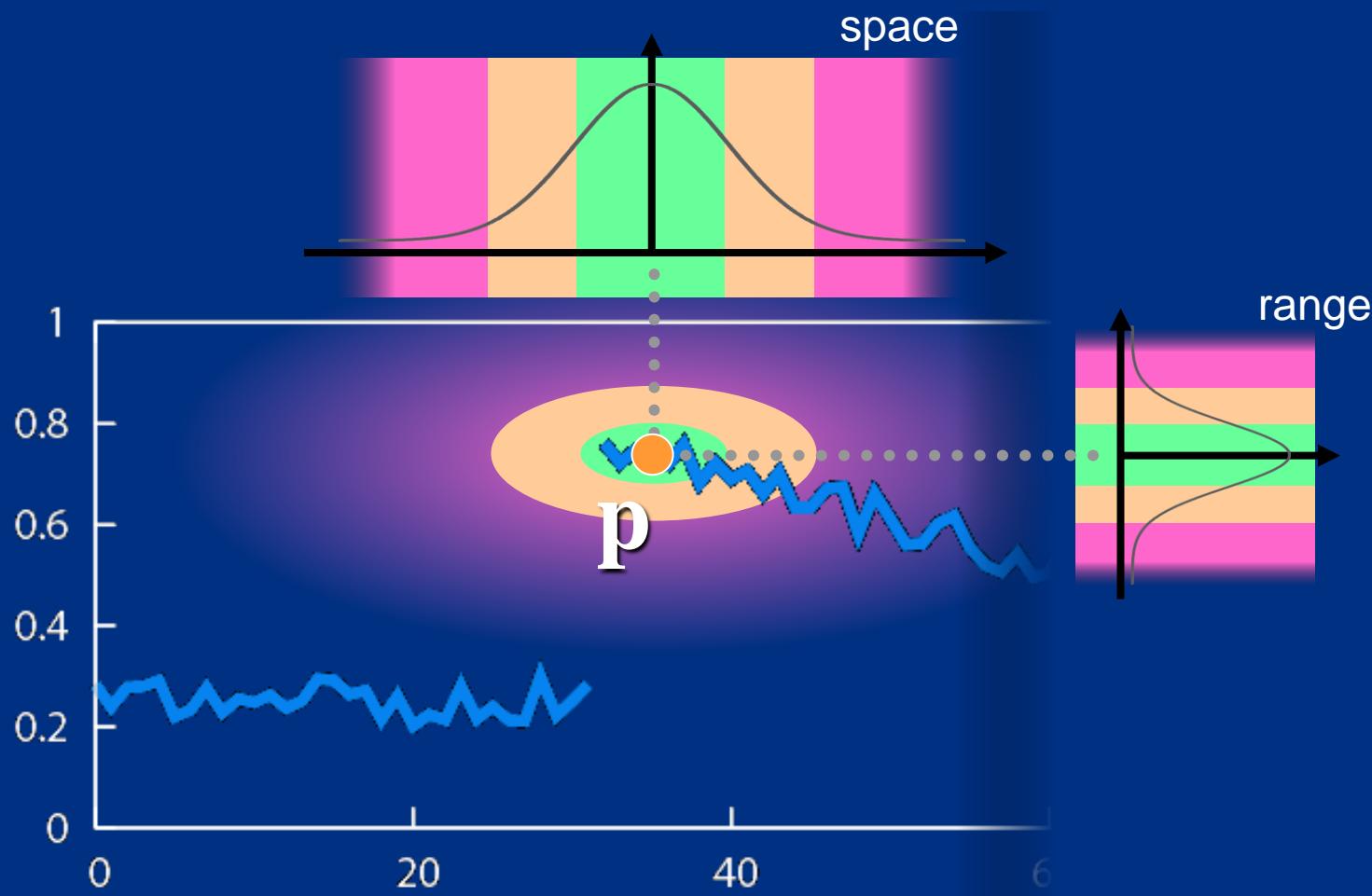
Space and Range Parameters

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$


- space σ_s : spatial extent of the kernel, size of the considered neighborhood.
- range σ_r : “minimum” amplitude of an edge

Influence of Pixels

Only pixels close in space and in range are considered.



Exploring the Parameter Space



input

$\sigma_s = 2$



$\sigma_r = 0.1$

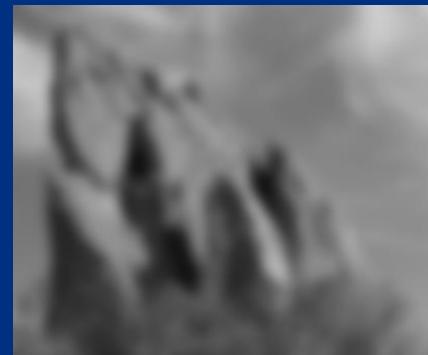
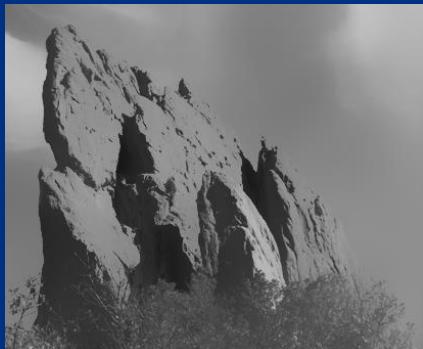
$\sigma_r = 0.25$

$\sigma_r = \infty$
(Gaussian blur)

$\sigma_s = 6$



$\sigma_s = 18$



Varying the Range Parameter



input

$\sigma_s = 2$



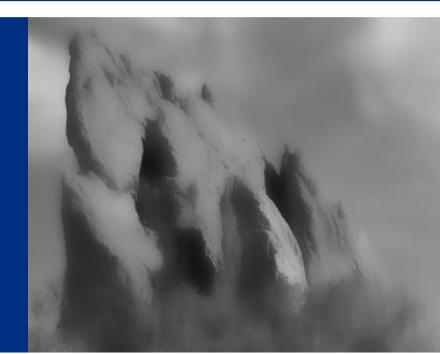
$\sigma_r = 0.1$

$\sigma_r = 0.25$

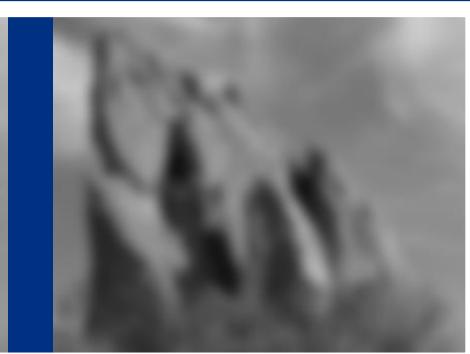
$\sigma_r = \infty$
(Gaussian blur)



$\sigma_s = 6$



$\sigma_s = 18$



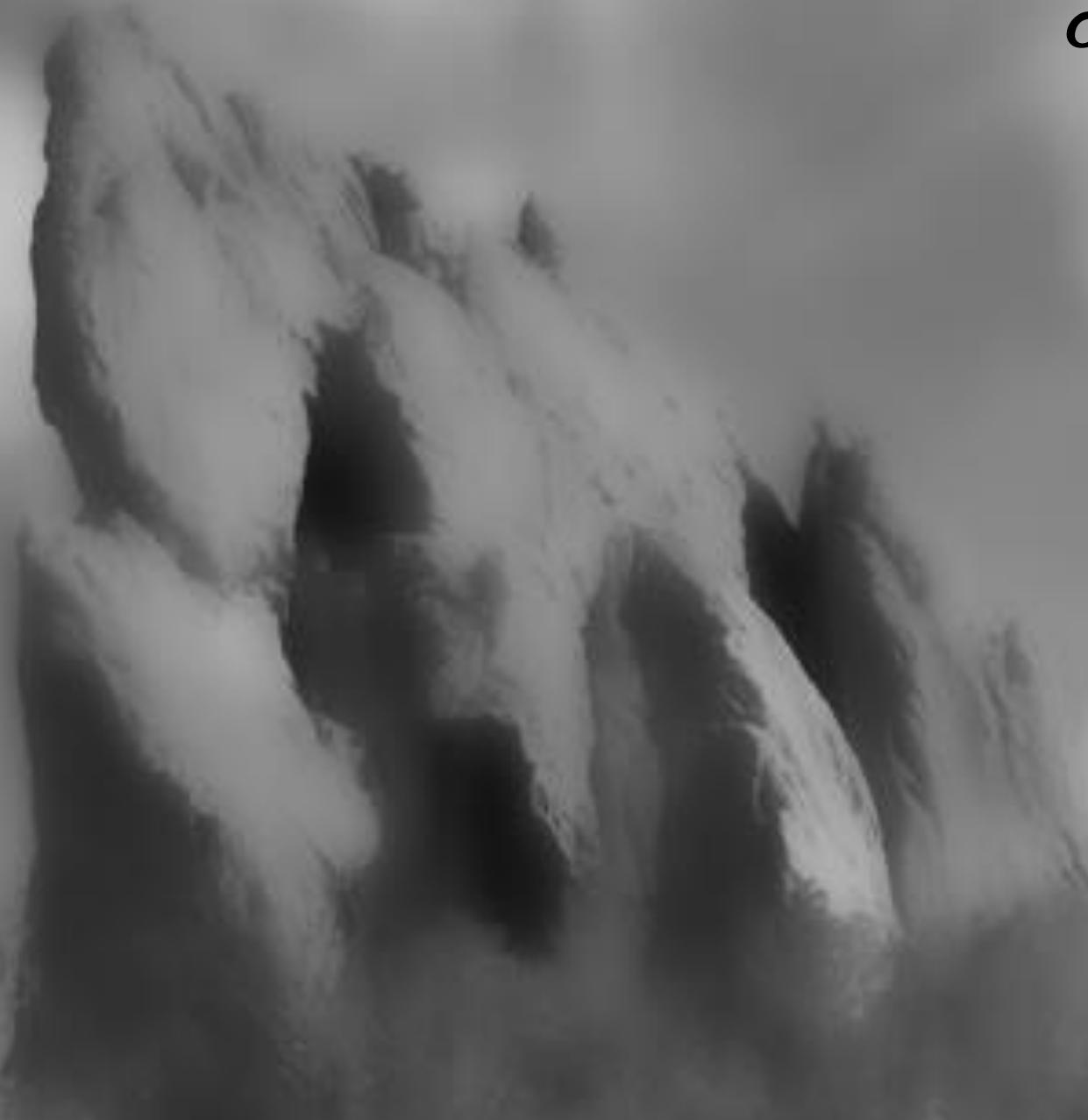
input



$\sigma_r = 0.1$



$\sigma_r = 0.25$



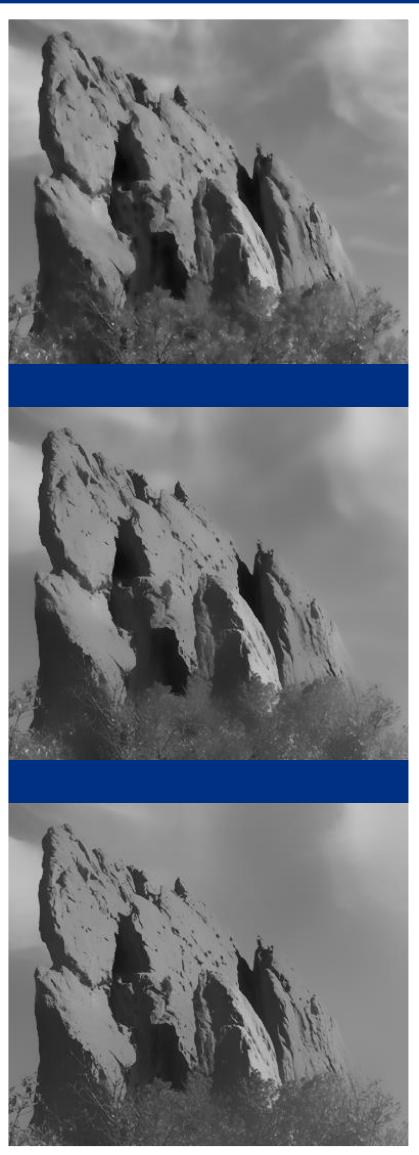
$\sigma_r = \infty$
(Gaussian blur)

Varying the Space Parameter



input

$\sigma_s = 2$



$\sigma_r = 0.1$

$\sigma_s = 6$



$\sigma_r = 0.25$

$\sigma_s = 18$



$\sigma_r = \infty$
(Gaussian blur)

input



A black and white photograph of a rugged, snow-covered mountain peak under a cloudy sky. The mountain's surface is heavily textured with deep shadows and bright highlights, suggesting a mix of sunlight and overcast conditions. The foreground is dark and out of focus, showing some low-lying vegetation or rocks. The background consists of a dense layer of clouds. $\sigma_s = 2$

 $\sigma_s = 6$

$\sigma_s = 18$

