

Image Filtering

Week - 2

Disclaimer: Many of the slides used here are obtained from online resources (including many open lecture materials given at MIT, etc.) without due acknowledgement. They are used here for the sole purpose of classroom teaching. All the credits and all the copyrights belong to the original authors. You should not copy it, redistribute it, put it online, or use it for any, any purposes other than helping you study the course of ENGN/COMP 4528/6528.

Announcement

- Mathematics Reference Book: Linear algebra, probability
 - <https://mml-book.github.io/book/mml-book.pdf>
 - eg. CDF (cumulative distribution function, see chapter 6.2 discrete and continuous distribution function)
- Lecture Series by Prof. Shree Nayar, Professor of Computer Science in the School of Engineering at Columbia University, Senior researcher in computer vision with high profile.
 - <https://fpcv.cs.columbia.edu/>

Outline

- Image Noise
- Linear filter
 - Gaussian filter
 - Edge detection
- Non-linear filter
 - median filter
 - bilateral filter

Outline

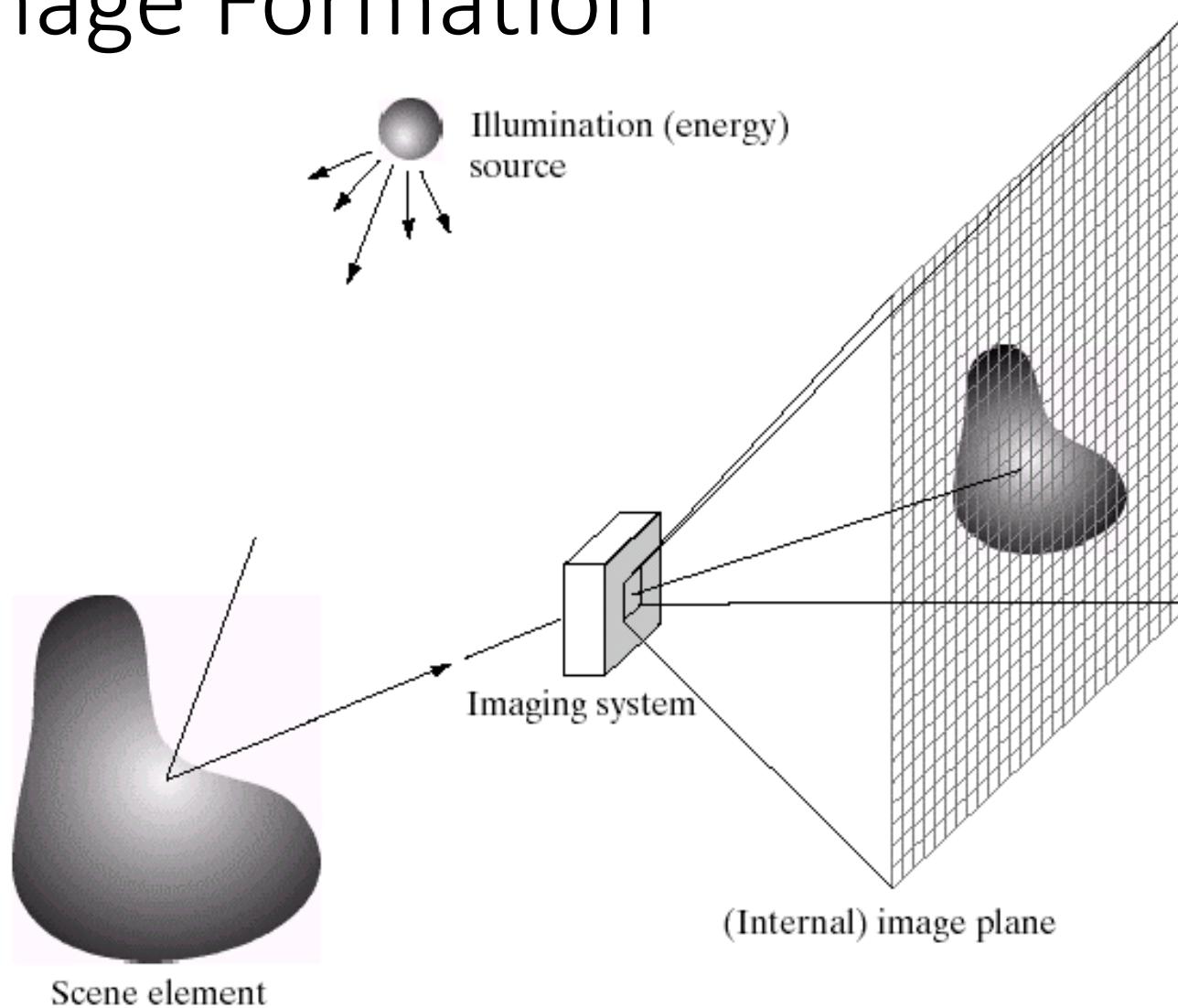
- Review
- Image Noise
- Linear filter
 - Gaussian filter
 - Edge detection
- Non-linear filter
 - median filter
 - bilateral filter

Image Filtering

- Filtering (correlation, convolution)
- Gaussian filter
- Application of filters (denoising, edge, contour, corner, texture, template matching and tracking ...,)

Image Noise

Recall: Image Formation



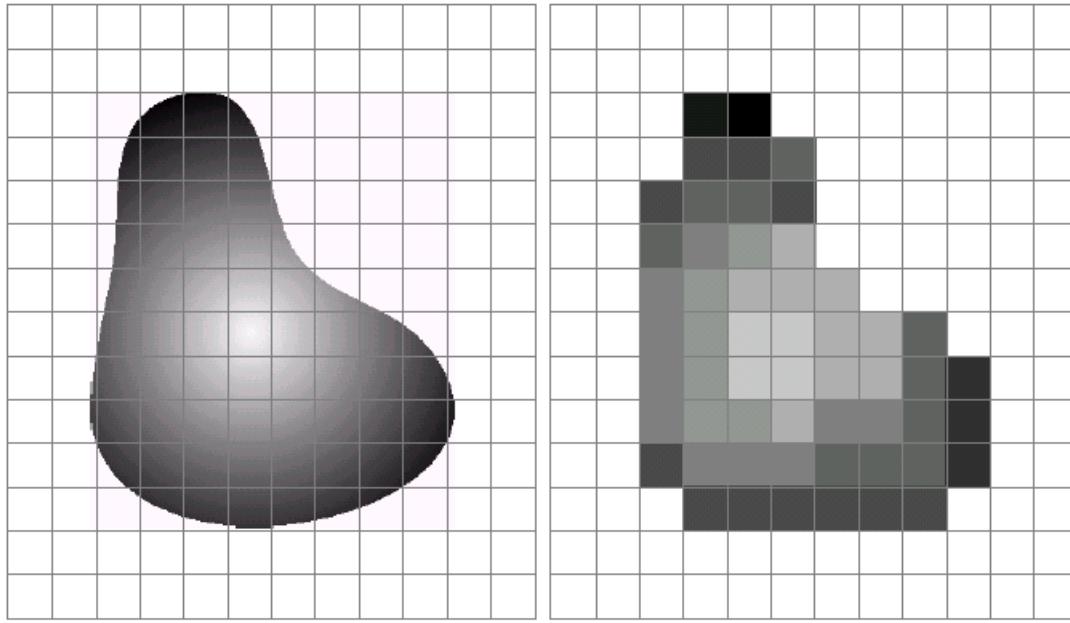
Recall: Digital camera



A digital camera replaces film with a sensor array

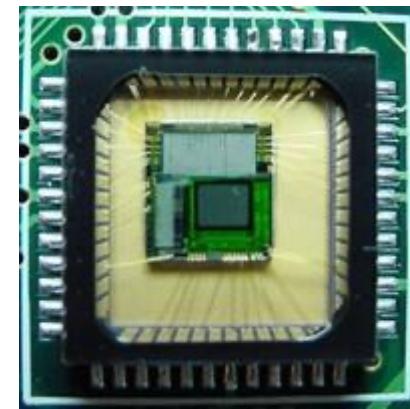
- Each cell in the array is light-sensitive diode that converts photons to electrons
- <http://electronics.howstuffworks.com/digital-camera.htm>

Digital images



a | b

FIGURE 2.17 (a) Continuous image projected onto a sensor array. (b) Result of image sampling and quantization.



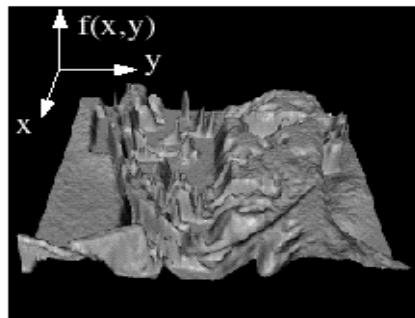
Recall: Images as functions

- 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:
 - $f: [a,b] \times [c,d] \rightarrow [0, 255]$
- A color image is just three functions pasted together. We can write this as a “vector-valued” function:

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

Recall: Digital images

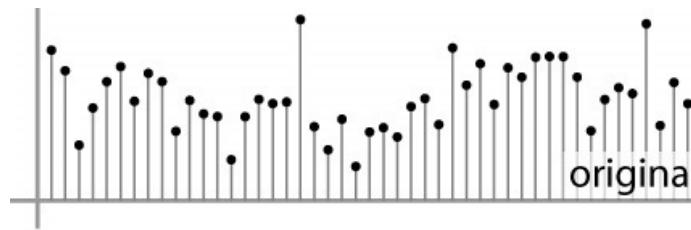
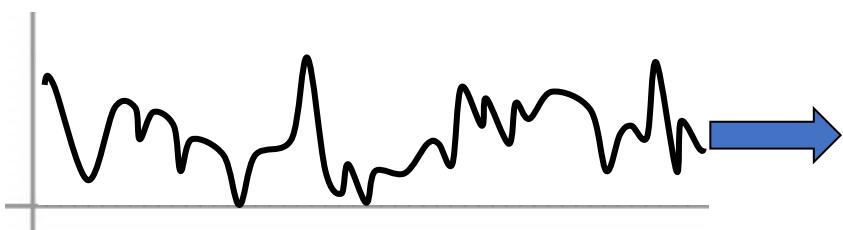
- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)
- Image is thus represented as a matrix of integer values.



j →
↓ i

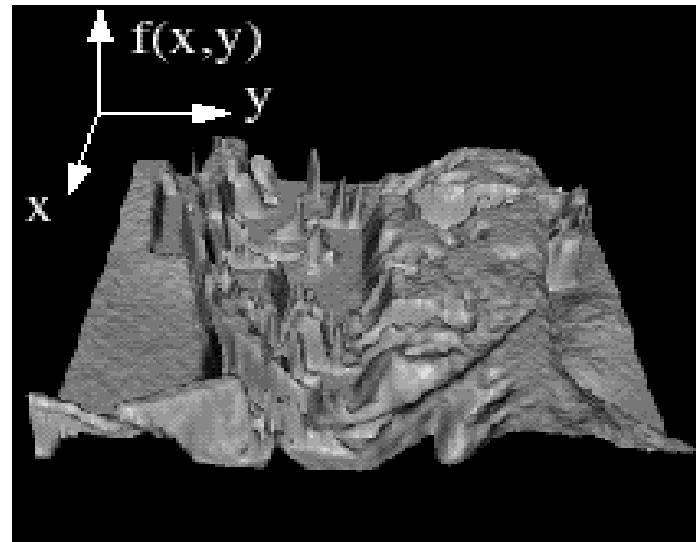
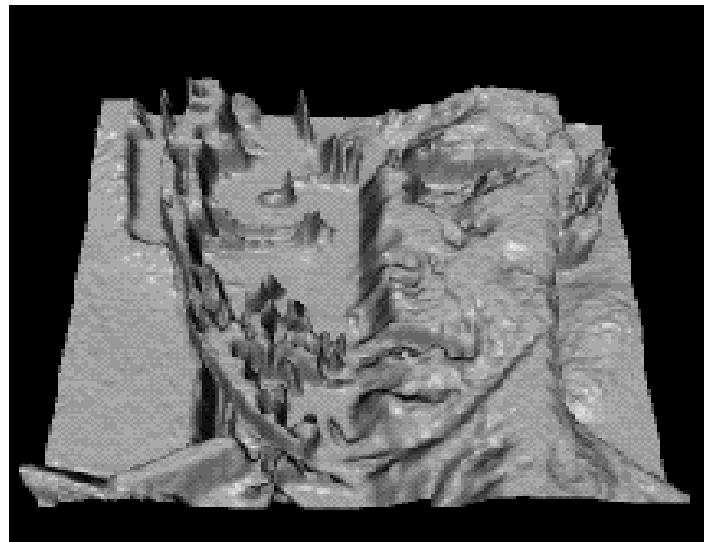
62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

2D

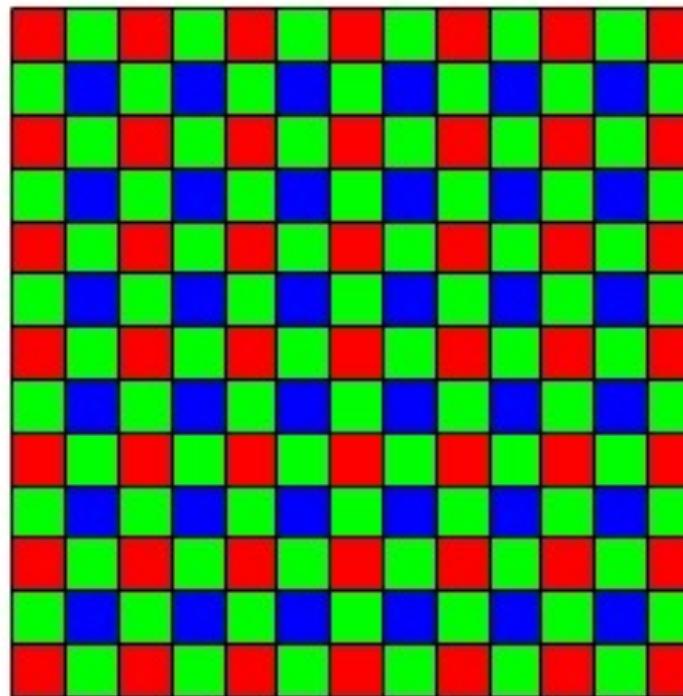


1D

Recall: Images as functions



Digital color images



Bayer filter

© 2000 How Stuff Works

Recall: Digital color images

Color images,
RGB color
space



R



G



B

Common types of noise

- **Salt and pepper noise:** random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels, sometimes, it is also referred to as Salt and pepper noise.
- **Gaussian noise:** variations in intensity drawn from a Gaussian normal distribution



Original



Salt and pepper noise

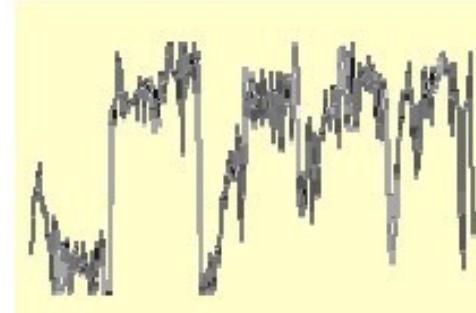
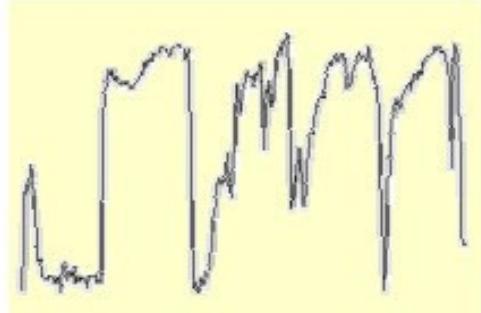
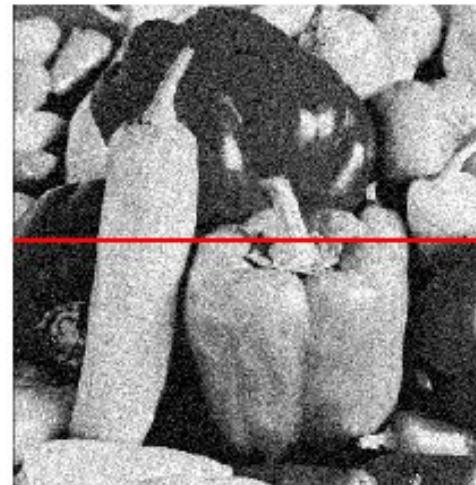
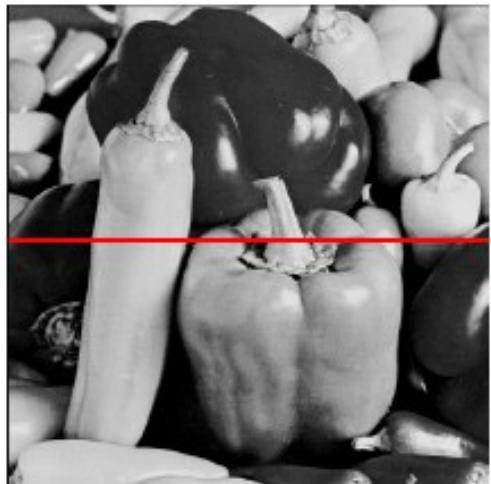


Impulse noise



Gaussian noise

Gaussian noise



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

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

```
>> noise = np.random.normal(mean, std_dev, shape)  
>> output = im + noise;
```

What is the impact of the sigma?

Fig: M. Hebert

Effect of
sigma on
Gaussian
noise:

Image
shows the
noise values
themselves.

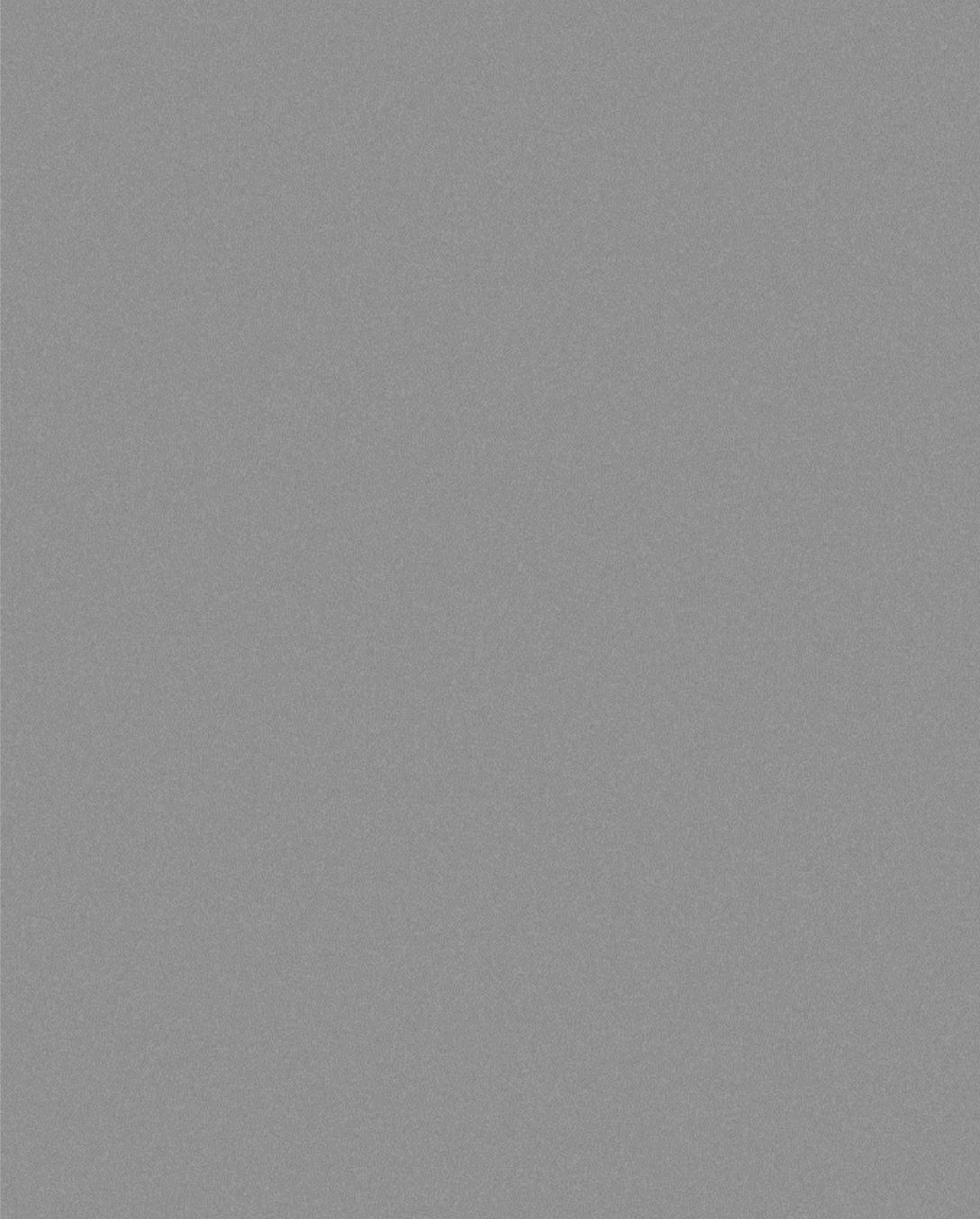
sigma=1



Effect of
sigma on
Gaussian
noise:

Image
shows the
noise values
themselves.

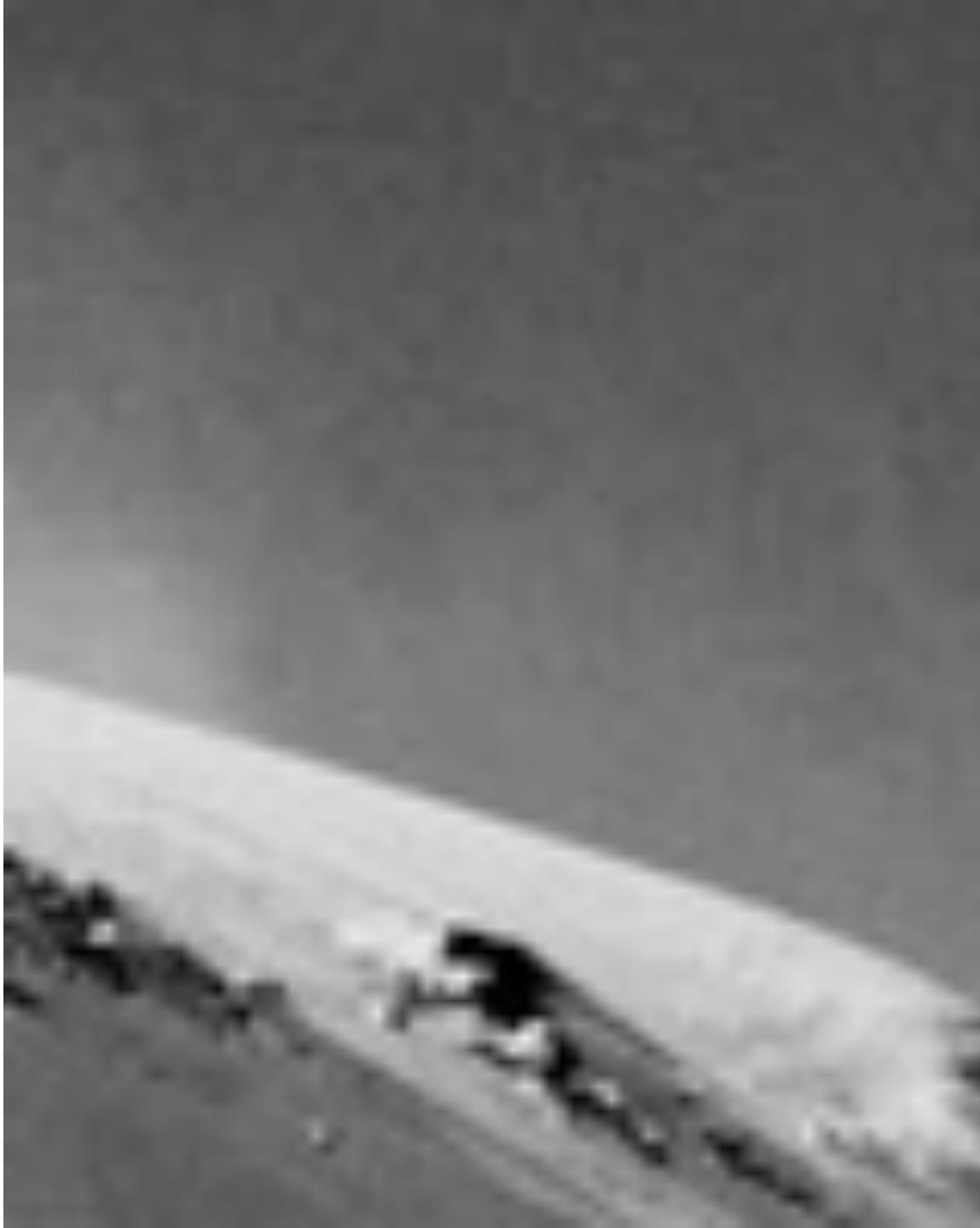
sigma=4



Effect of
sigma on
Gaussian
noise:

This shows
the noise
values
added to the
raw
intensities of
an image.

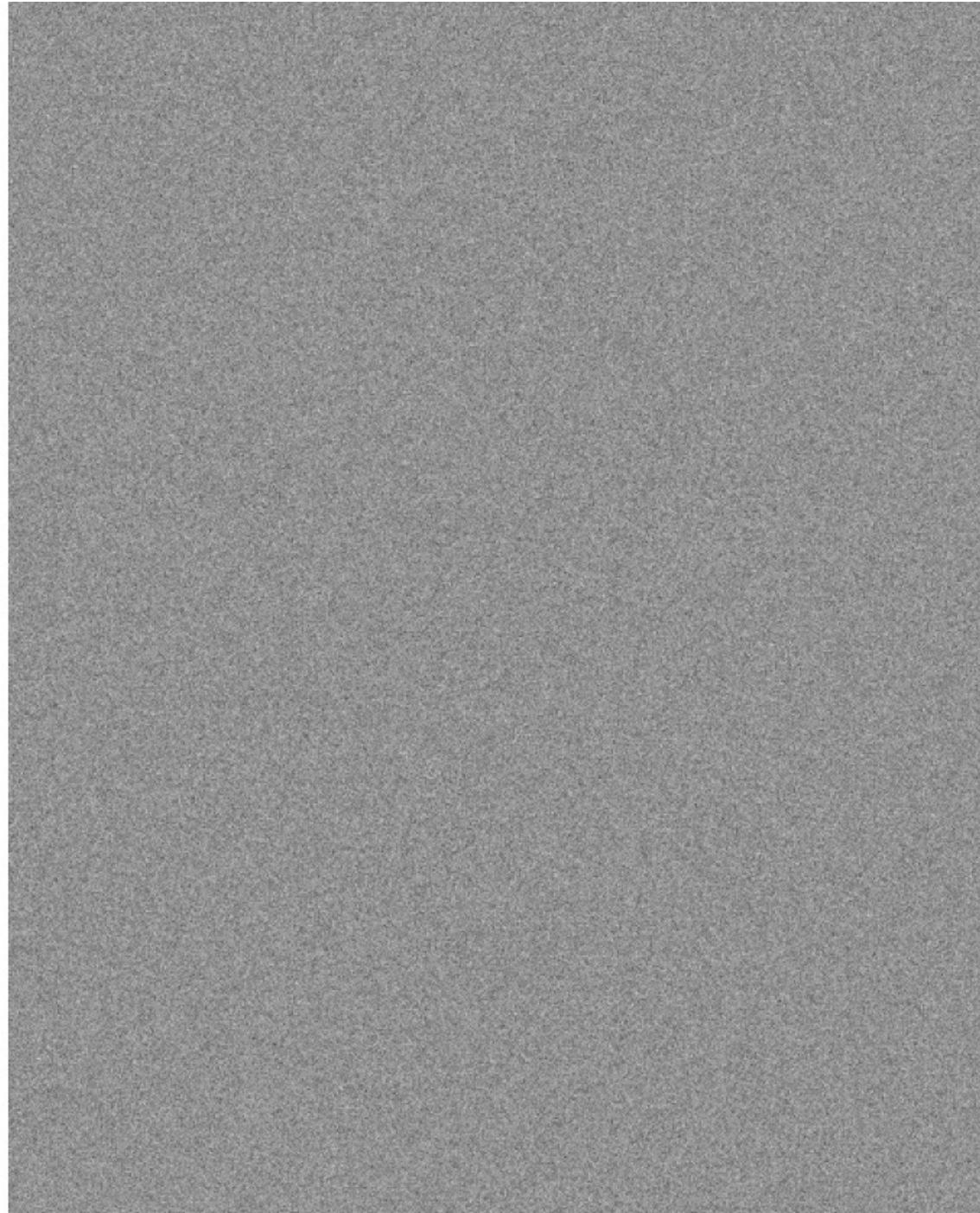
sigma=1



Effect of
sigma on
Gaussian
noise:

Image
shows the
noise values
themselves.

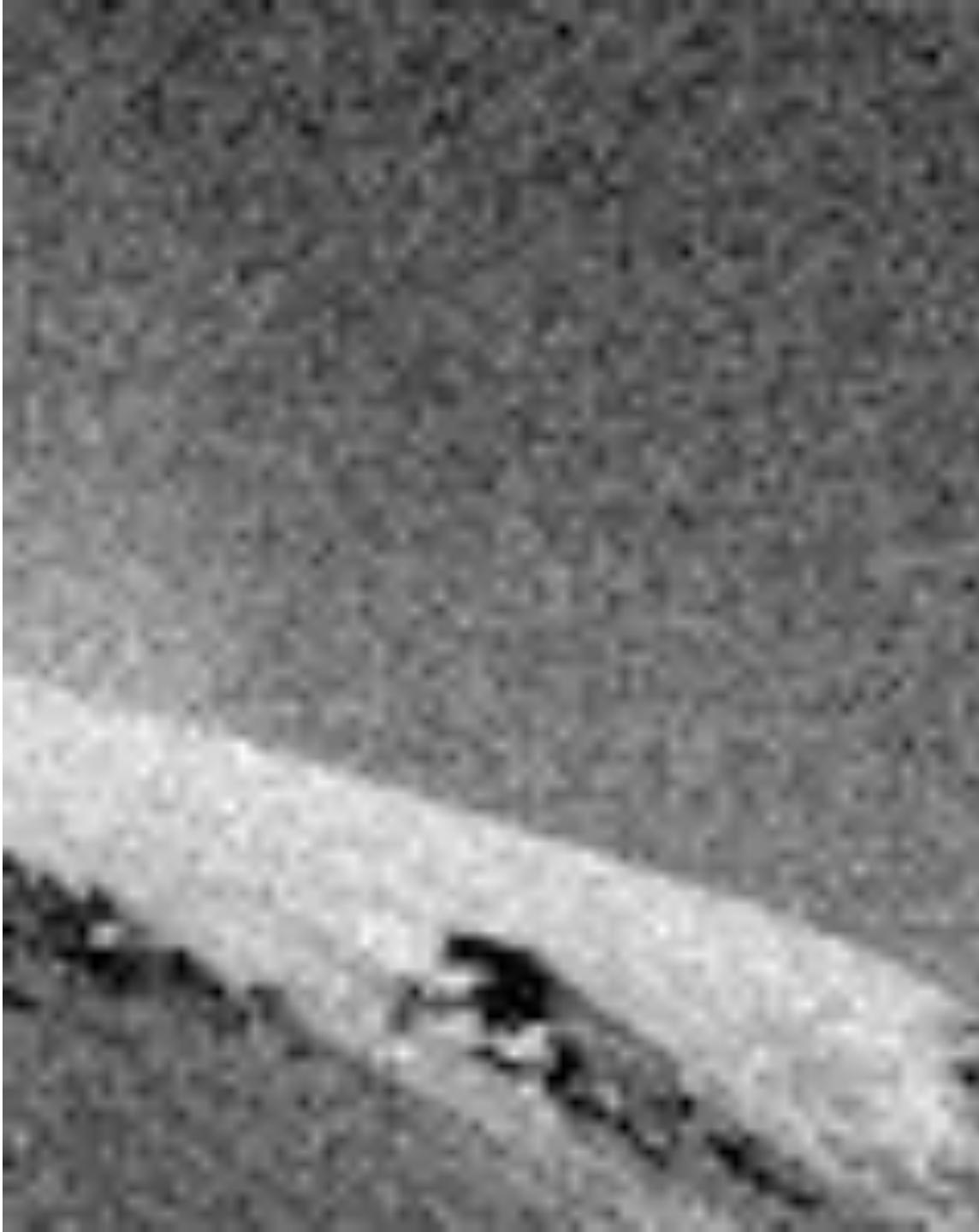
sigma=
16



Effect of sigma on Gaussian noise

This shows
the noise
values
added to the
raw
intensities of
an image.

sigma=16



Noise Reduction

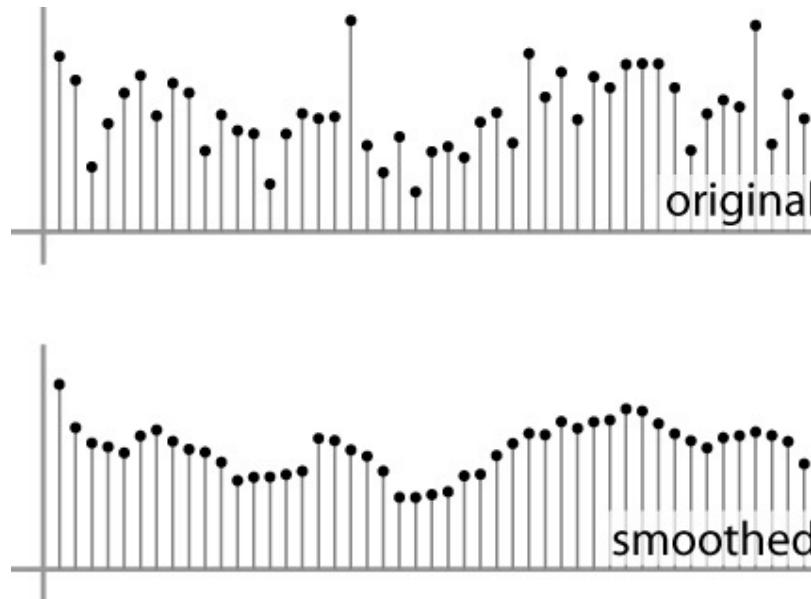
- How should we reduce the noise?

First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood (neighborhood operation)
- Assumptions:
 - Expect pixels to be like their neighbors
 - Expect noise processes to be independent from pixel to pixel

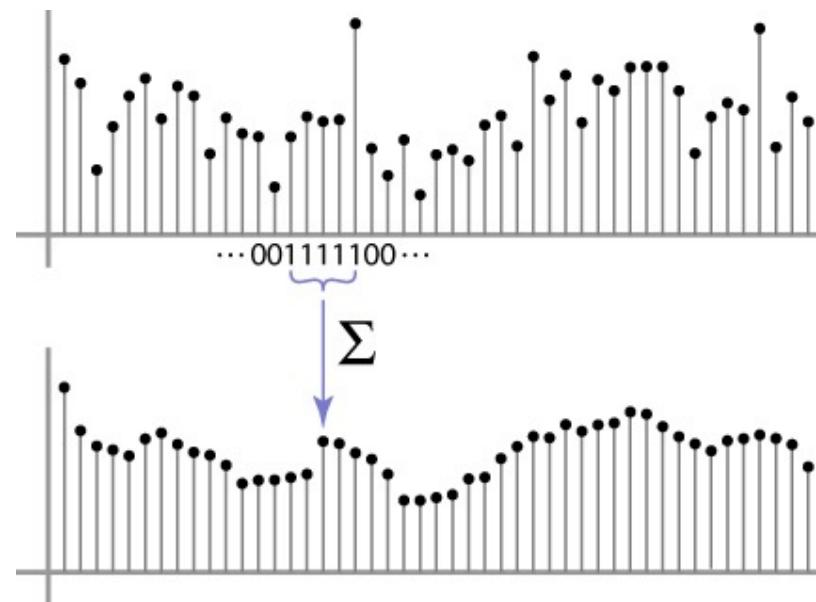
First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



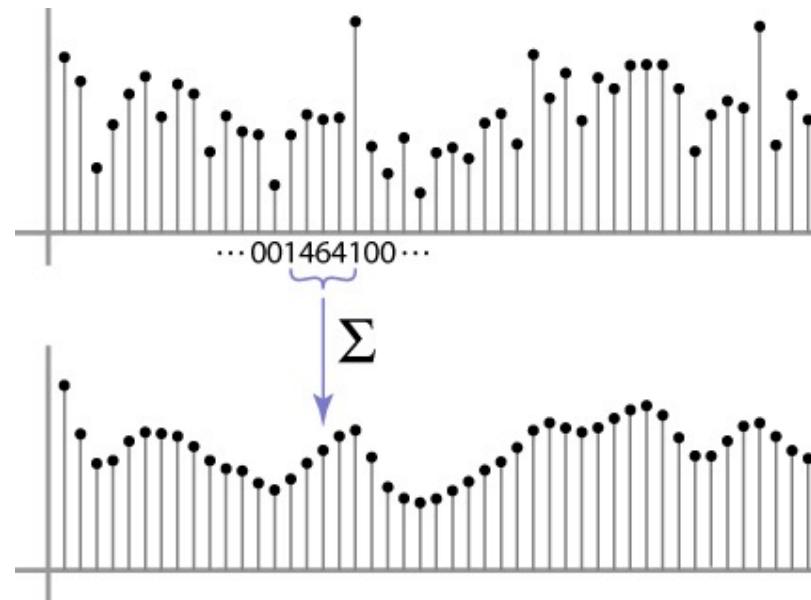
Weighted Moving Average

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



Weighted Moving Average

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



Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	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

$$G[x, y]$$

Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	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

$G[x, y]$

0	10									

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

			0	10	20					

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	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

$$G[x, y]$$

			0	10	20	30	30			

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	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

$$G[x, y]$$

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

Correlation filtering

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

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

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

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

Correlation filtering

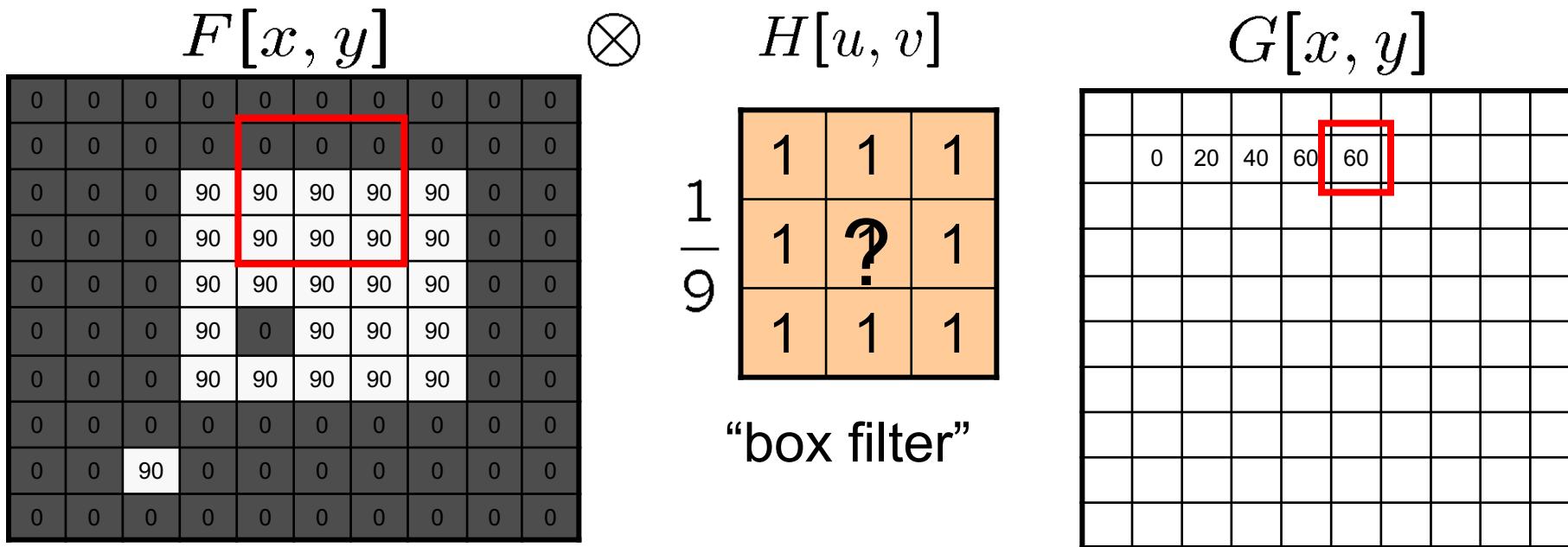
$$G[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[x + u, y + v]$$

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

- Filtering an image: replace each pixel with a linear combination of its neighbors.
- The filter “**kernel**” or “**mask**” $H [u, v]$ defines the weights in the linear combination.

Averaging filter

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



$$G = H \otimes F$$

Smoothing by averaging



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



original



filtered

We can see the ‘Block’ effect on the filtered image.

Image Filtering

- Image filtering: compute the function value of local neighbourhood at each pixel.
- Very useful
 - **Enhance images**
 - De-noise, resize, increase contrast, etc.
 - **Extract information from images**
 - Texture, edges, distinctive points, etc.
 - **Detect patterns**
 - Template matching

Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	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	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

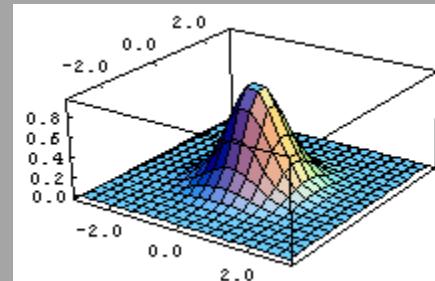
$F[x, y]$

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

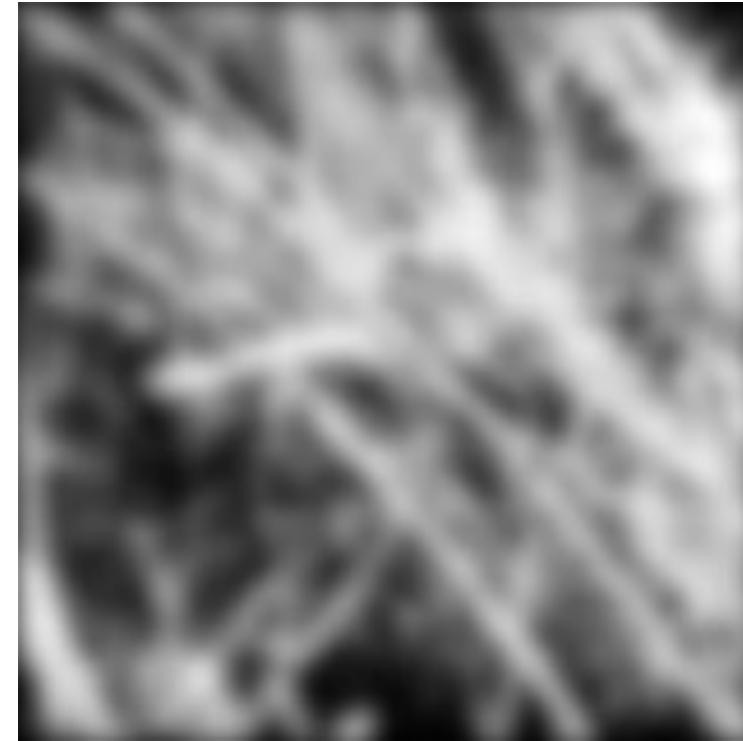
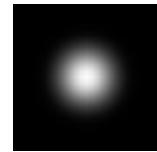
$H[u, v]$

- This kernel is an approximation of a 2d Gaussian function:

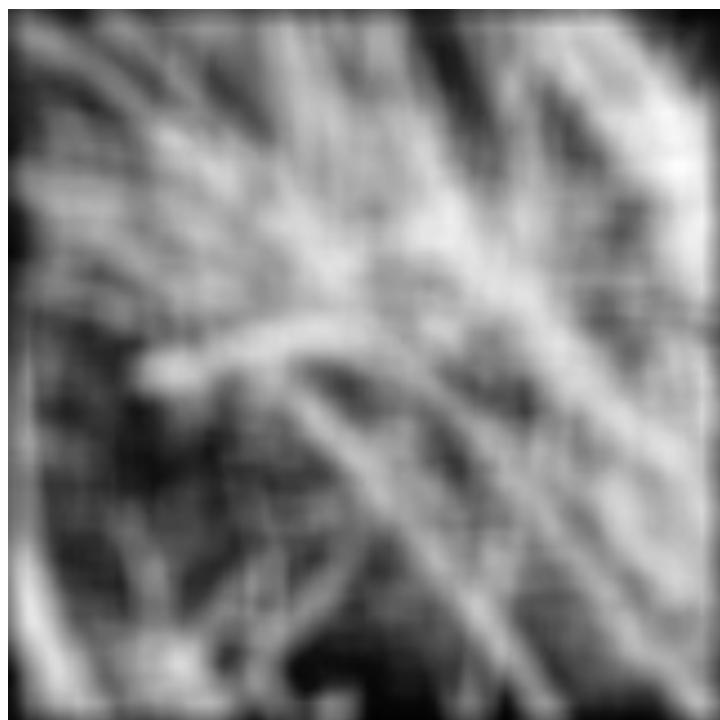
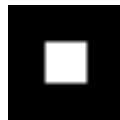
$$h(u, v) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{u^2+v^2}{2\sigma^2}}$$



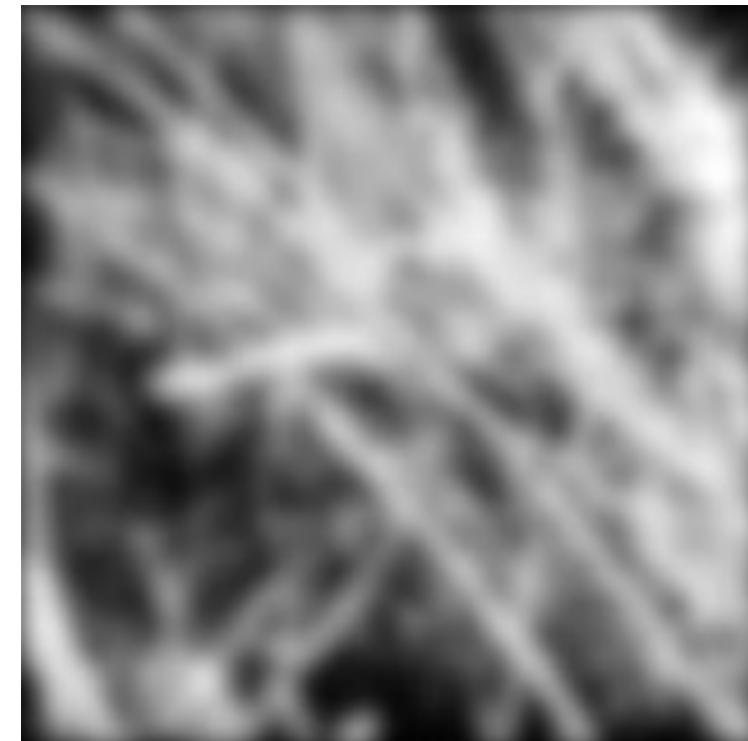
Smoothing with a Gaussian



Compare box filter with Gaussian filter effects



Smoothed image by box filter

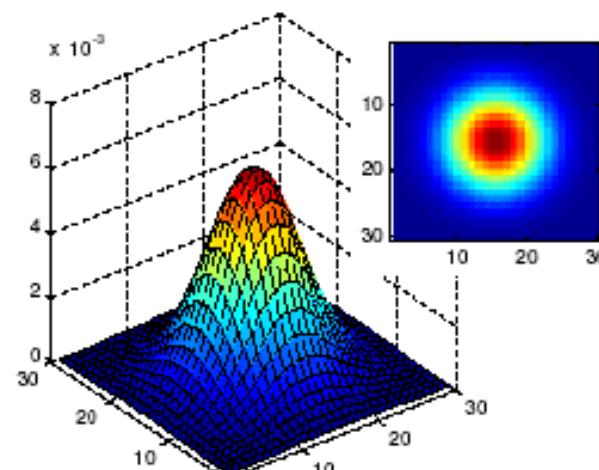
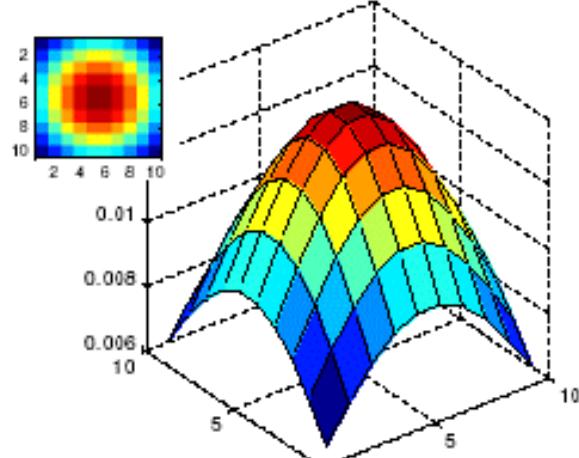
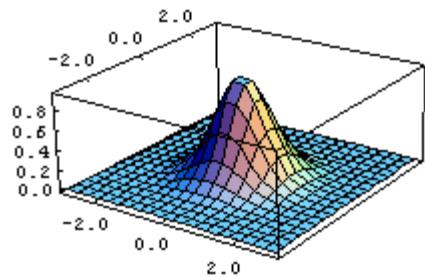


Smoothed image by Gaussian filter

Gaussian filters

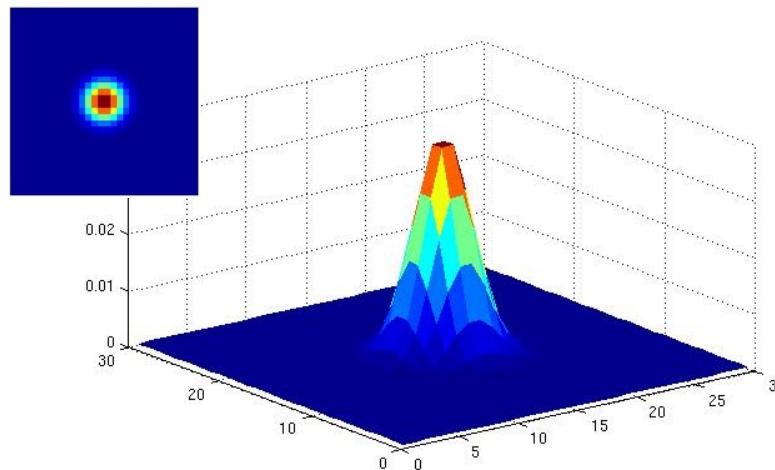
- What parameters matter here?
- **Size** of kernel or mask
 - Note, Gaussian function has infinite support, but discrete filters use finite kernels

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

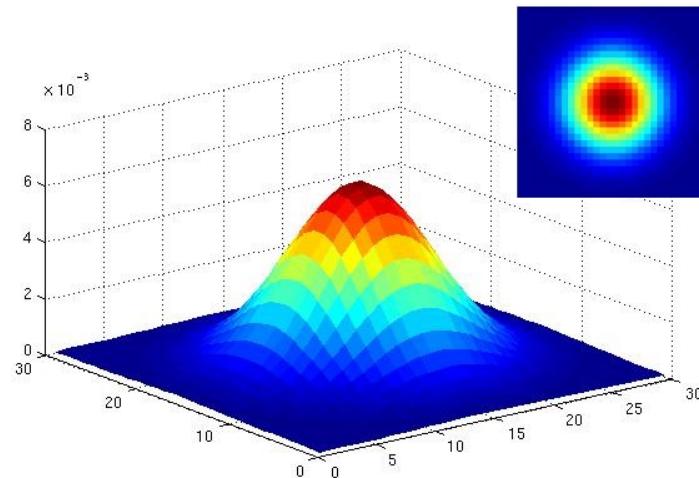


Gaussian filters

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



$\sigma = 2$ with
30 x 30
kernel

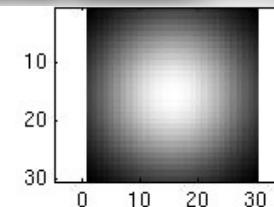
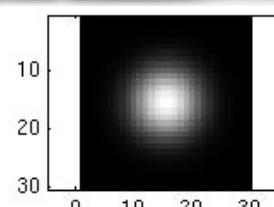
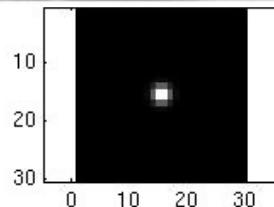


$\sigma = 5$ with
30 x 30
kernel

• Slide credit:
Kristen Grauman

Smoothing with a Gaussian

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



•Slide credit:
Kristen Grauman

Convolution

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

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

$$G = H \star F$$



*Notation for
convolution
operator*

Convolution

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

$$G(x, y) = \sum_{u=0}^M \sum_{v=0}^N F(u, v) H(x - u, y - v)$$

$$G = F \star H$$



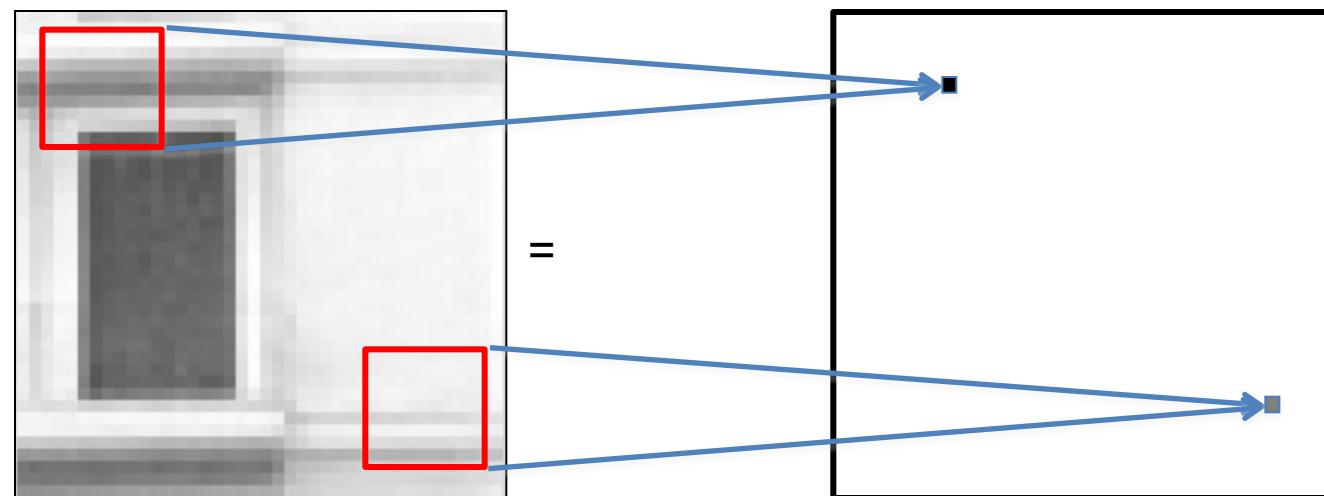
*Notation for
convolution
operator*

Discrete linear system



In vision, many times, we are interested in operations that are spatially invariant.
For a linear spatially invariant system:

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



Properties of convolution

- **Shift invariant:**
 - Operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.

Properties of convolution

- Commutative:

$$f * g = g * f$$

- Associative

$$(f * g) * h = f * (g * h)$$

- Distributes over addition

$$f * (g + h) = (f * g) + (f * h)$$

- Scalars factor out

$$kf * g = f * kg = k(f * g)$$

- Identity:

$$\text{unit impulse } e = [..., 0, 0, 1, 0, 0, ...]. \quad f * e = f$$

Separability

- In some cases, the filter is separable, and we can factor into two steps:
 - Convolve all rows with a 1D filter
 - Convolve all columns with a 1D filter

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 \end{bmatrix}$$

Separability example

- 2D convolution (center location only)(65)

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{l} 2 + 6 + 3 = 11 \\ 6 + 20 + 10 = 36 \\ 4 + 8 + 6 = 18 \\ \hline 65 \end{array}$$

h f

Separability example

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{l} 2 + 6 + 3 = 11 \\ 6 + 20 + 10 = 36 \\ 4 + 8 + 6 = 18 \\ \hline 65 \end{array}$$

- The filter is factored into a product of 1D filters.

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

Separability example

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{l} 2 + 6 + 3 = 11 \\ 6 + 20 + 10 = 36 \\ 4 + 8 + 6 = 18 \\ \hline 65 \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

- Perform convolutions along rows.

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 11 & & \\ \hline & 18 & \\ \hline & 18 & \\ \hline \end{array}$$

Separability example

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{l} 2 + 6 + 3 = 11 \\ 6 + 20 + 10 = 36 \\ 4 + 8 + 6 = 18 \\ \hline 65 \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

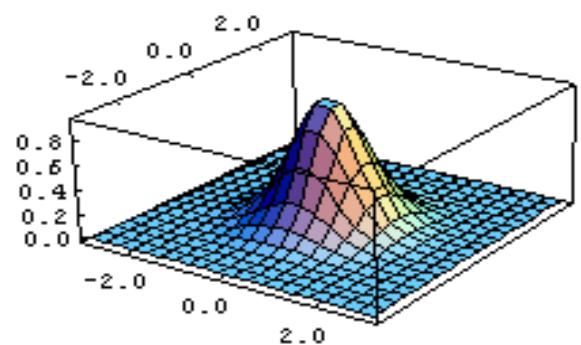
$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & 11 & \\ \hline & 18 & \\ \hline & 18 & \\ \hline \end{array}$$

- Perform convolutions along columns.

$$\begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline & 11 & \\ \hline & 18 & \\ \hline & 18 & \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline & 65 & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

This kernel is an approximation of
a 2d Gaussian function:

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



Properties of Gaussian filter

- Rotational symmetry treats features of all orientations equally (isotropy).
- Convolution with self gives 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 the product of two 1D Gaussians

Separability of the Gaussian filter

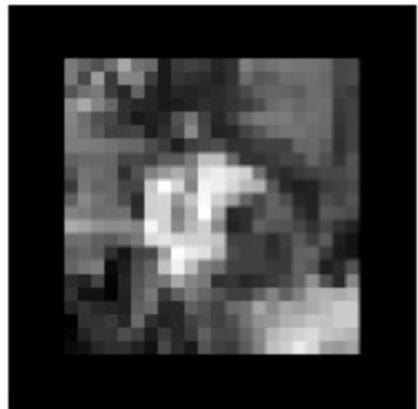
$$\begin{aligned} G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right) \end{aligned}$$

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

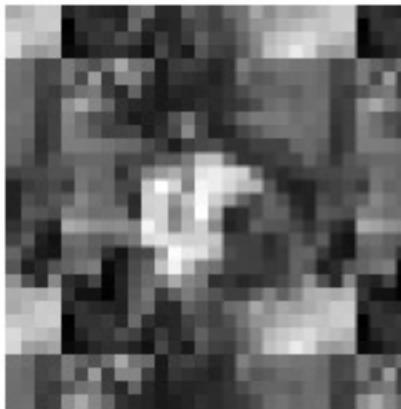
In this case, the two functions are the (identical) 1D Gaussian

- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
 - $O(n^2 m^2)$
- What if the kernel is separable?
 - $O(n^2 m)$

How to handle borders



zero



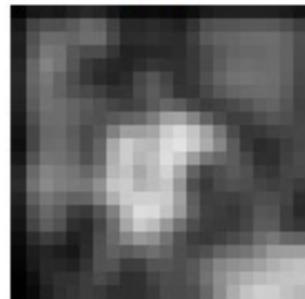
wrap



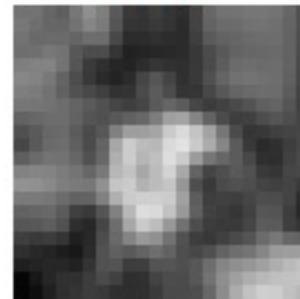
clamp



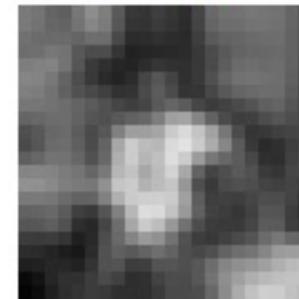
mirror



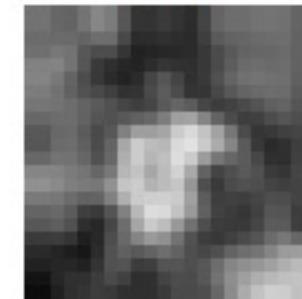
blurred: zero



normalized zero



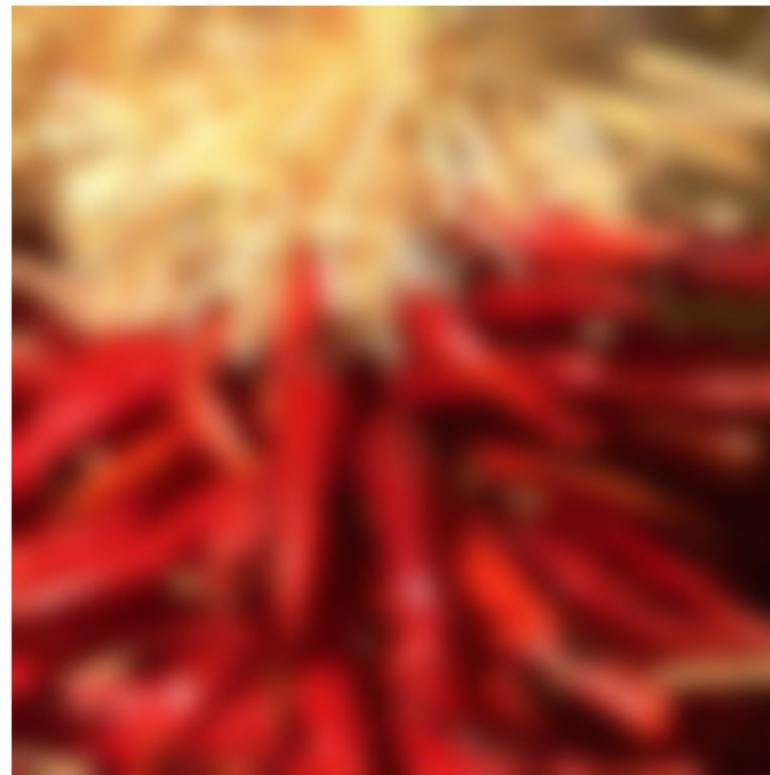
clamp



mirror

From Szeliski, Computer Vision, 2010

Example: Border handling



Source: S. Marschner

Python's border handling

- What is the size of the output?
- **cv2.filter2D(h,g,-1, bordertype)**

Enumerator	
BORDER_CONSTANT Python: cv.BORDER_CONSTANT	iiiiii abcdefgh iiiiii with some specified <code>i</code>
BORDER_REPLICATE Python: cv.BORDER_REPLICATE	aaaaaaaa abcdefgh hhhhhhh
BORDER_REFLECT Python: cv.BORDER_REFLECT	fedcba abcdefgh hgfedcb
BORDER_WRAP Python: cv.BORDER_WRAP	cdefgh abcdefgh abcdefg

Discrete Filtering

- Linear filter: Weighted sum of pixels over rectangular neighborhood—*kernel* defines weights
- Think of kernel as template being matched by **correlation** (Python: `cv2.filter2D`)
- **Convolution:** Correlation with kernel rotated 180° (Matlab: `conv2`)

1	-1	-1
1	2	-1
1	1	1

Practice with linear filters

A taxonomy of useful filters

- Impulse, Shifts,
- Blur
 - Box filter
 - Gaussian filter
 - Bilateral filter
 - Asymmetrical filter: motion blur
- Edges
 - [-1 1]
 - Derivative filter
 - Derivative of a gaussian
 - Oriented filters
 - Gabor filter
 - Quadrature filters: phase and magnitude.
 - Elongated edges: filling gaps...

Practice with linear filters

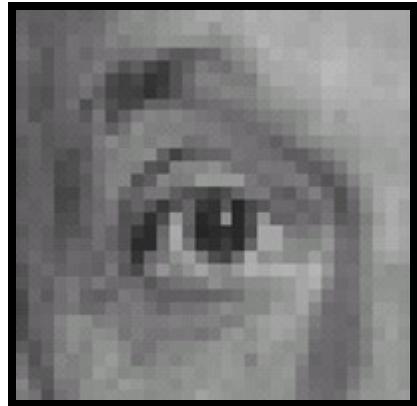


Original

0	0	0
0	1	0
0	0	0

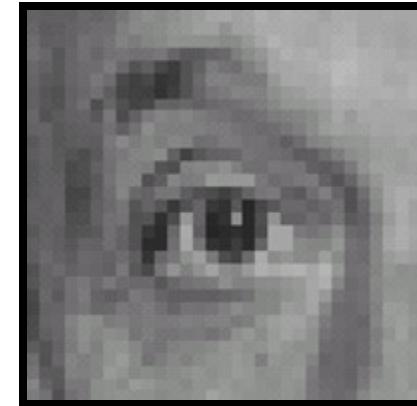
?

Practice with linear filters



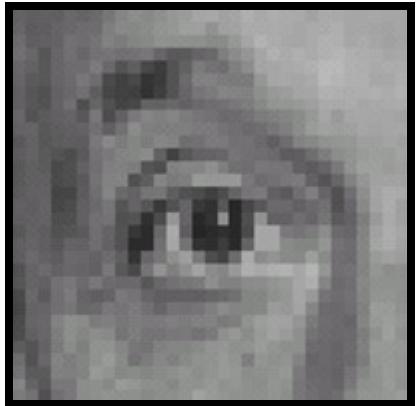
Original

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

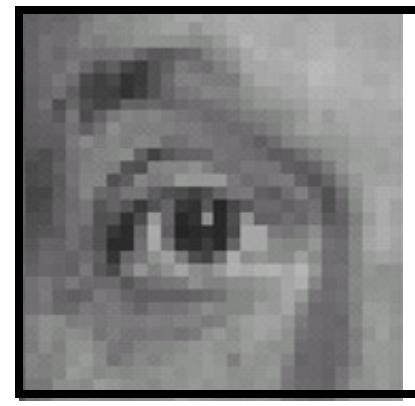
?

Practice with linear filters



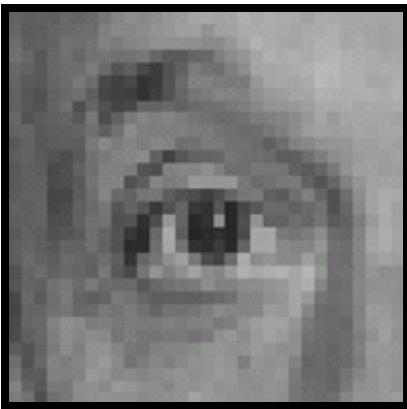
Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

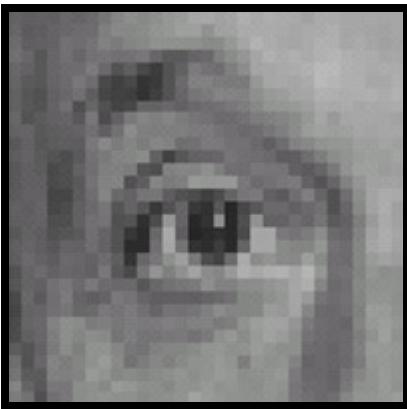
-

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

?

(Note that filter sums to 1)

Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

-

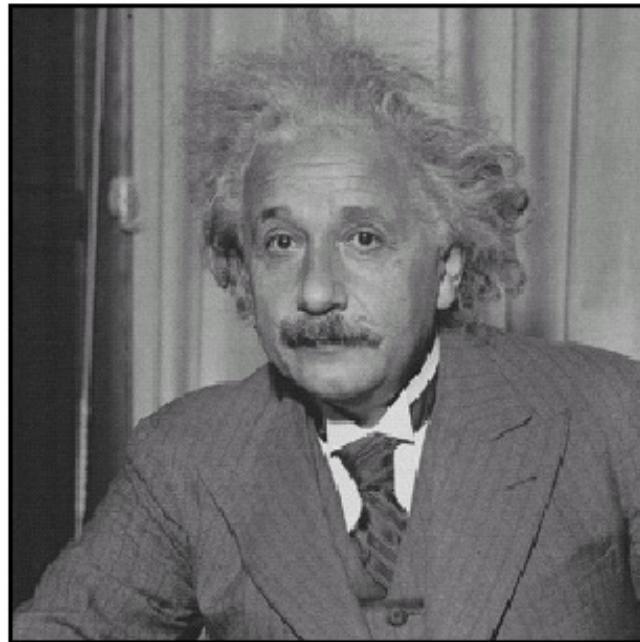
$\frac{1}{9}$	1	1	1
1	1	1	1
1	1	1	1



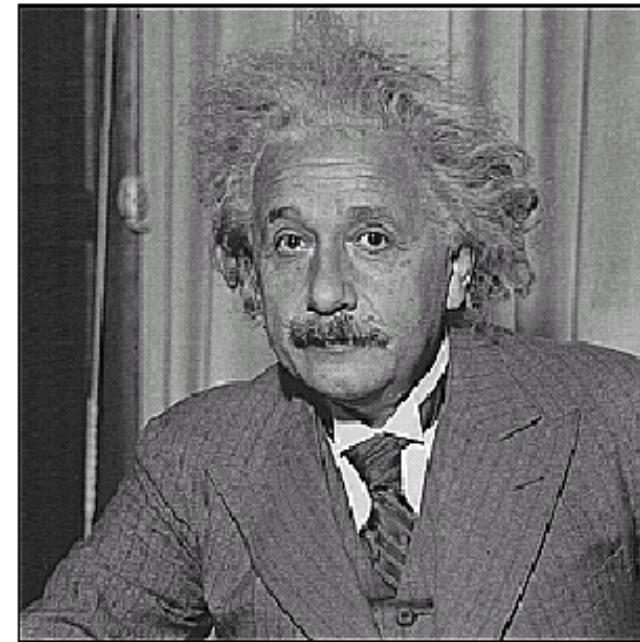
Sharpening filter

- Accentuates differences with local average

Sharpening

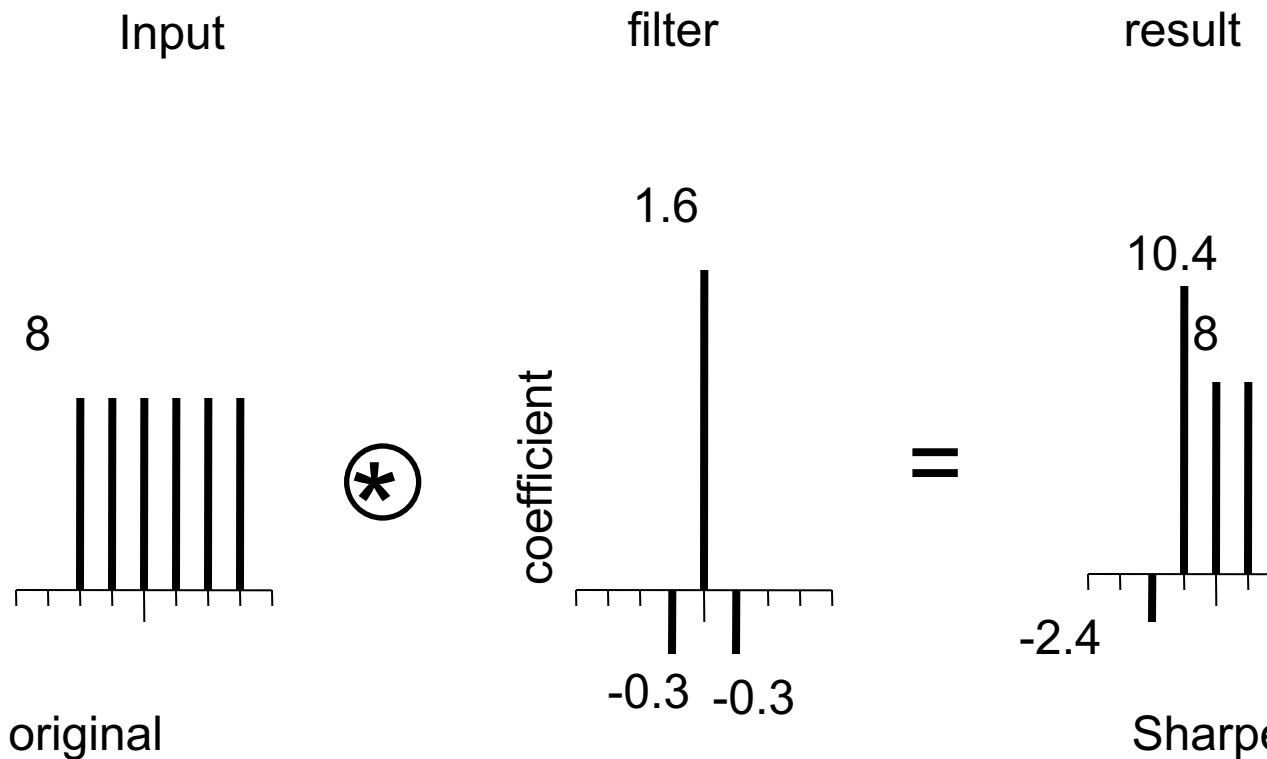


before



after

1-D sharpening example



Sharpened
(differences are
accentuated; constant
areas are left untouched).

$[-1 \ 1]$

$$\frac{\partial \mathbf{I}}{\partial x} \simeq \mathbf{I}(x, y) - \mathbf{I}(x - 1, y)$$

 \otimes $[-1, 1]$ $h[m,n]$ 

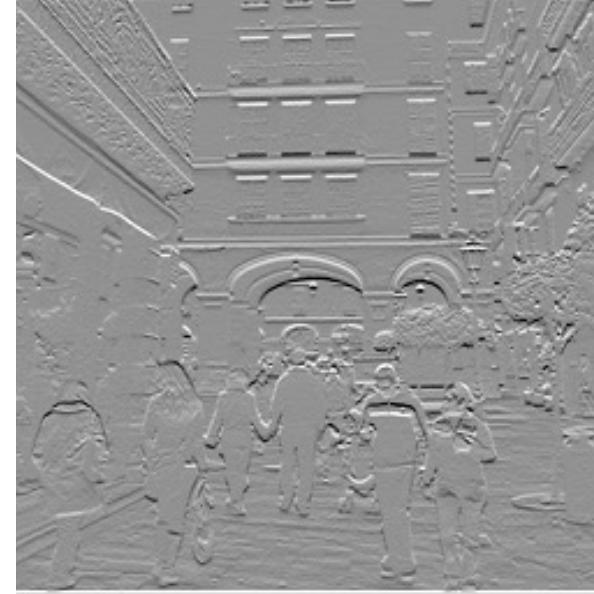
$$[-1 \ 1]^\top$$

$$\frac{\partial \mathbf{I}}{\partial y} \approx \mathbf{I}(x, y) - \mathbf{I}(x, y - 1)$$

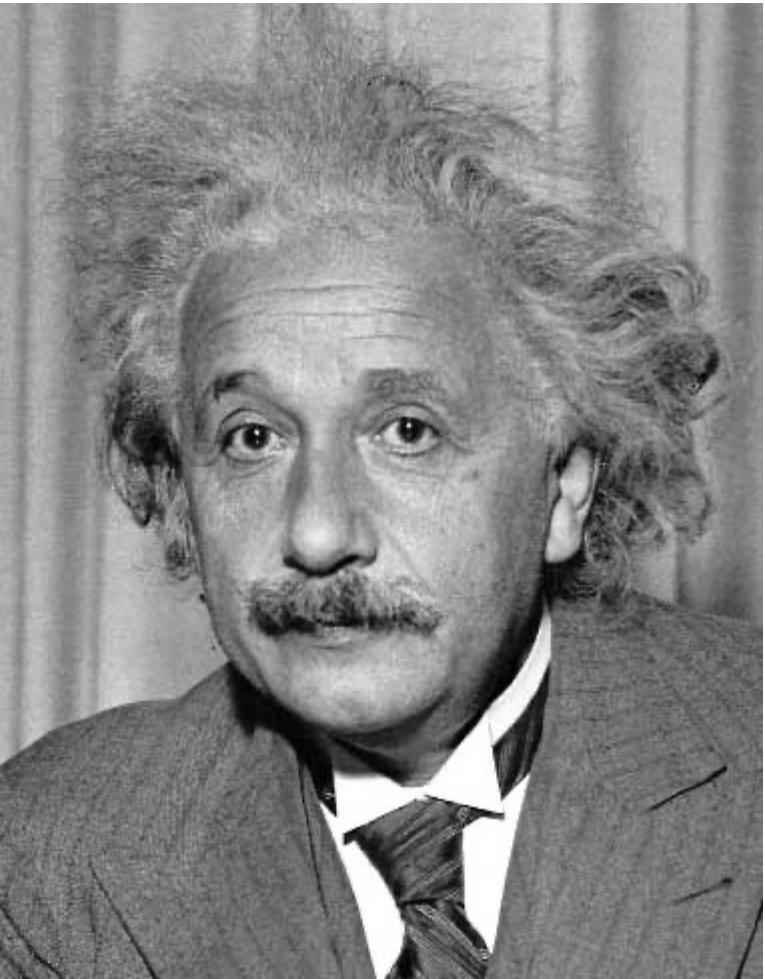
 \otimes

$$[-1, 1]^\top =$$

$$h[m,n]$$

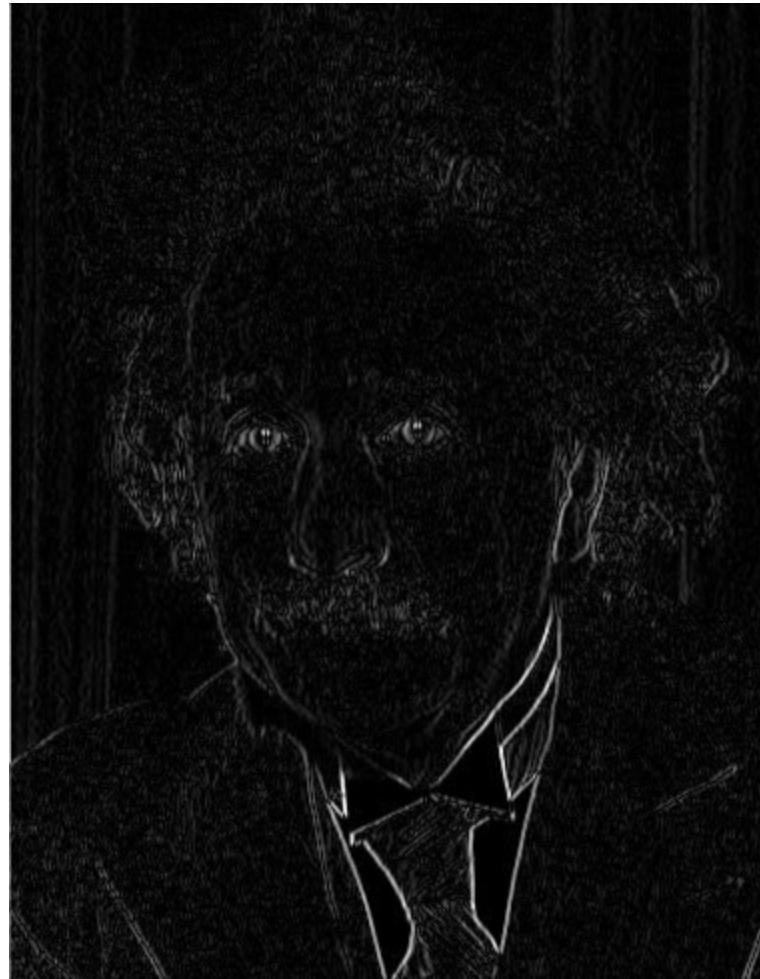


Derivative filters



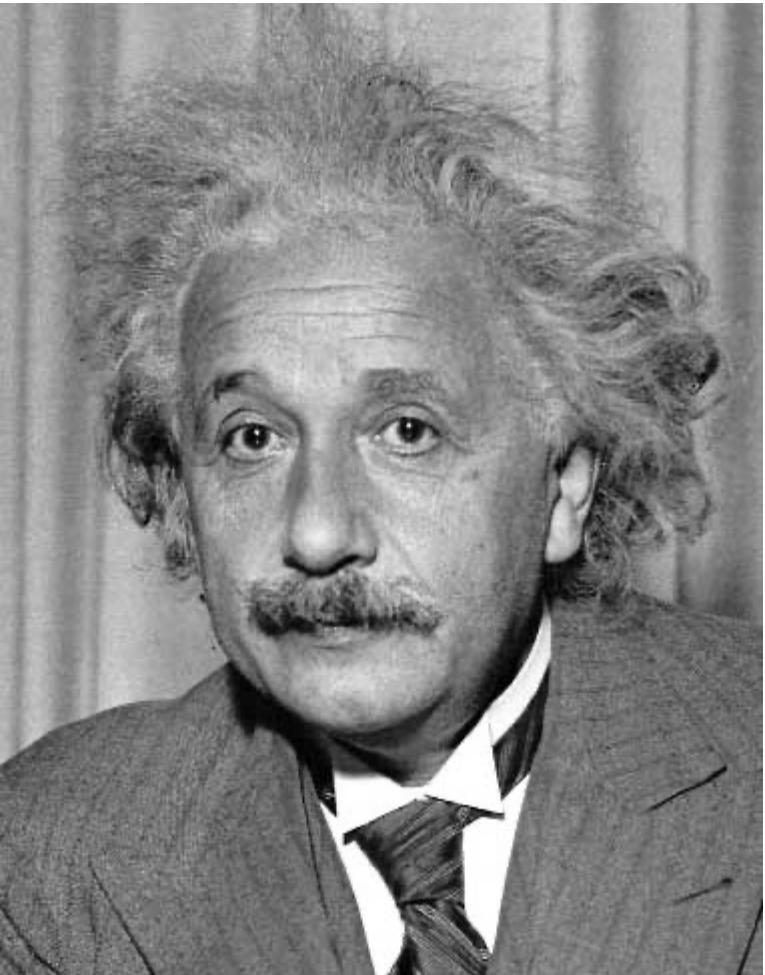
-1	0	1
-2	0	2
-1	0	1

Sobel



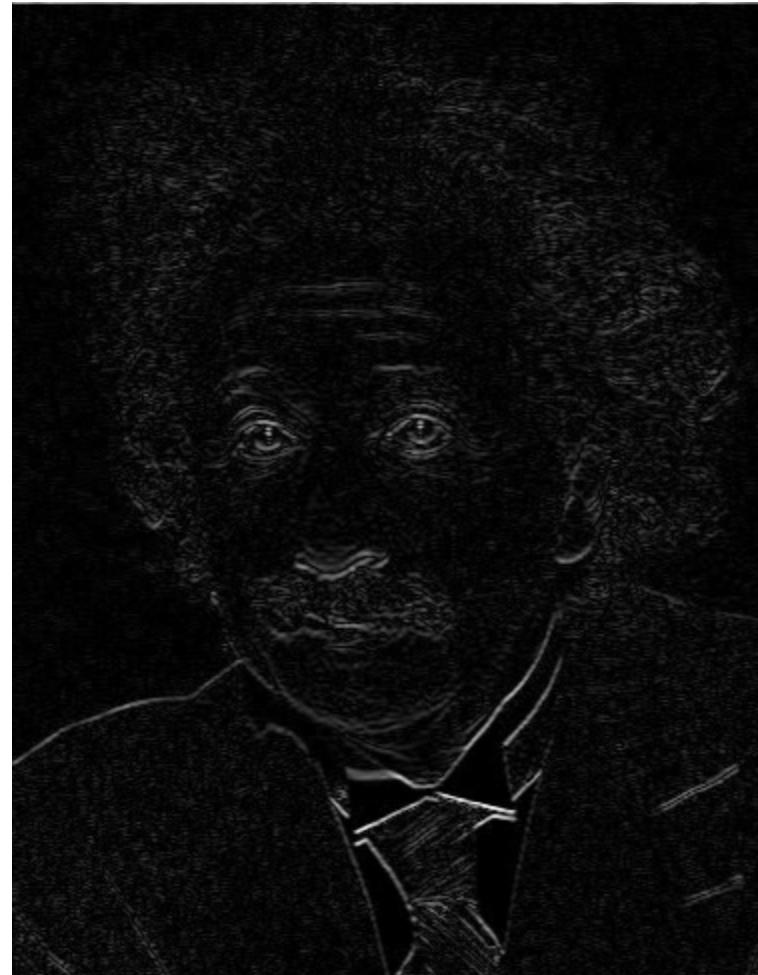
Vertical Edge
(absolute value)

Derivative filters



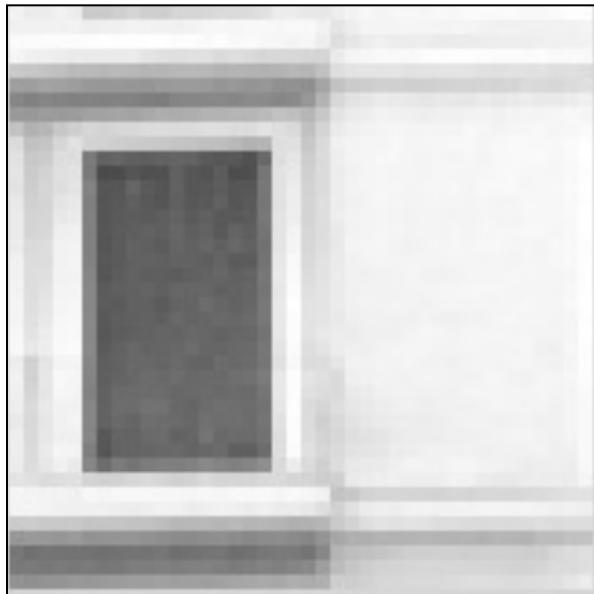
1	2	1
0	0	0
-1	-2	-1

Sobel



Horizontal Edge
(absolute value)

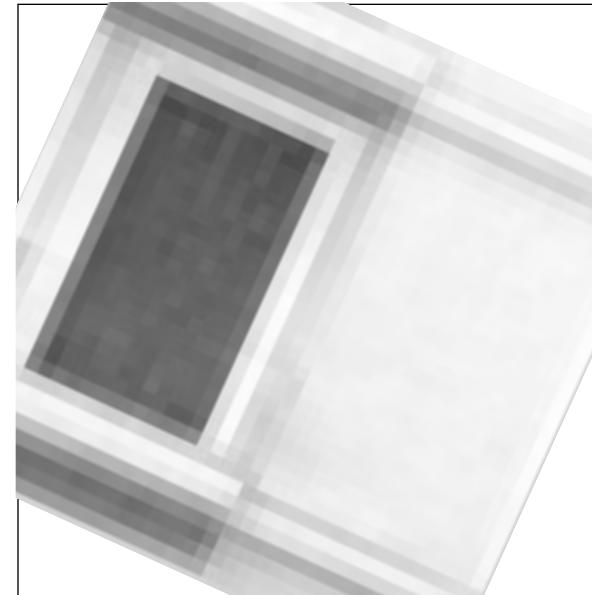
Image rotation



$g[m,n]$

\otimes ? =

$h[m,n]$

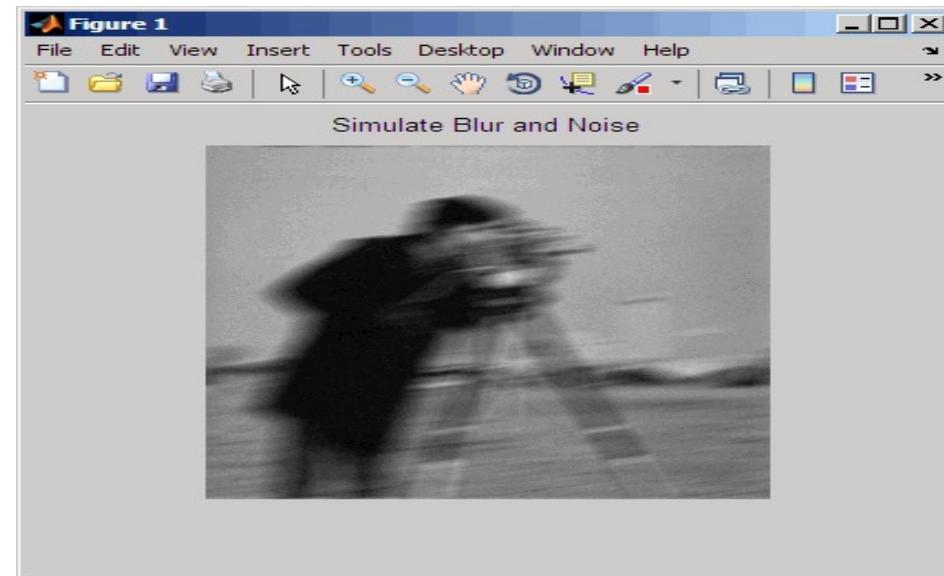


$f[m,n]$

It is linear, but not a spatially invariant operation. There is not convolution.

How could we synthesize motion blur?

```
theta = 30; len = 20;  
fil = imrotate(ones(1, len), theta, 'bilinear');  
fil = fil / sum(fil(:));  
figure(2), imshow(imfilter(im, fil));
```



Motion blur filter



$g[m,n]$

\otimes



=

$h[m,n]$



$f[m,n]$

Motion blur filter



$g[m,n]$

\otimes

$h[m,n]$

=



$f[m,n]$

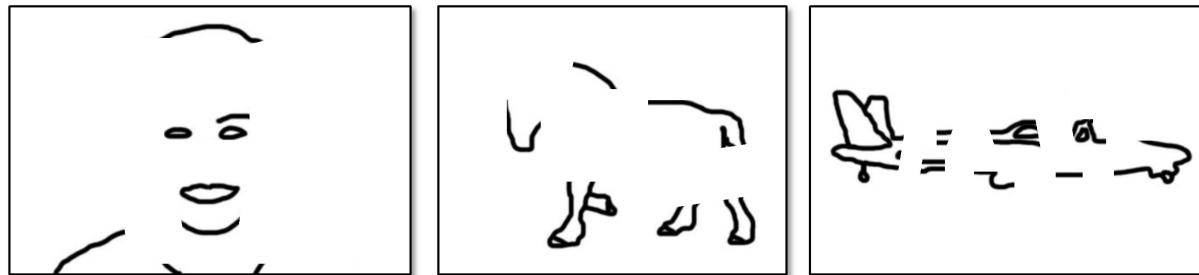
Summary

- Linear filters and convolution useful for
 - Enhancing images (smoothing, removing noise)
 - Box filter
 - Gaussian filter
 - Separable filters more efficient

Edge Detection

Edge detection

- **Goal:** map image from 2d array of pixels to a set of curves or line segments or contours.
- **Why?**



•Figure from J. Shotton et al., PAMI 2007

- **Main idea:** look for strong gradients, post-process

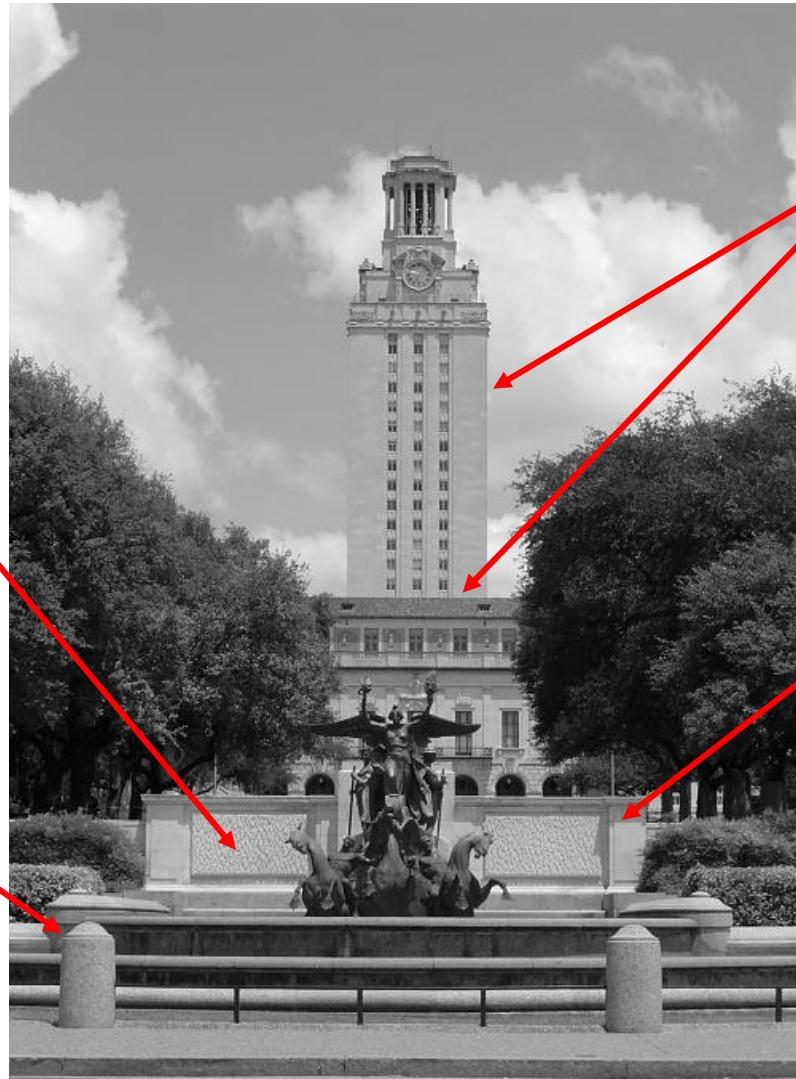
What causes an edge?

- Reflectance change:
appearance
information, texture

- Change in surface
orientation: shape

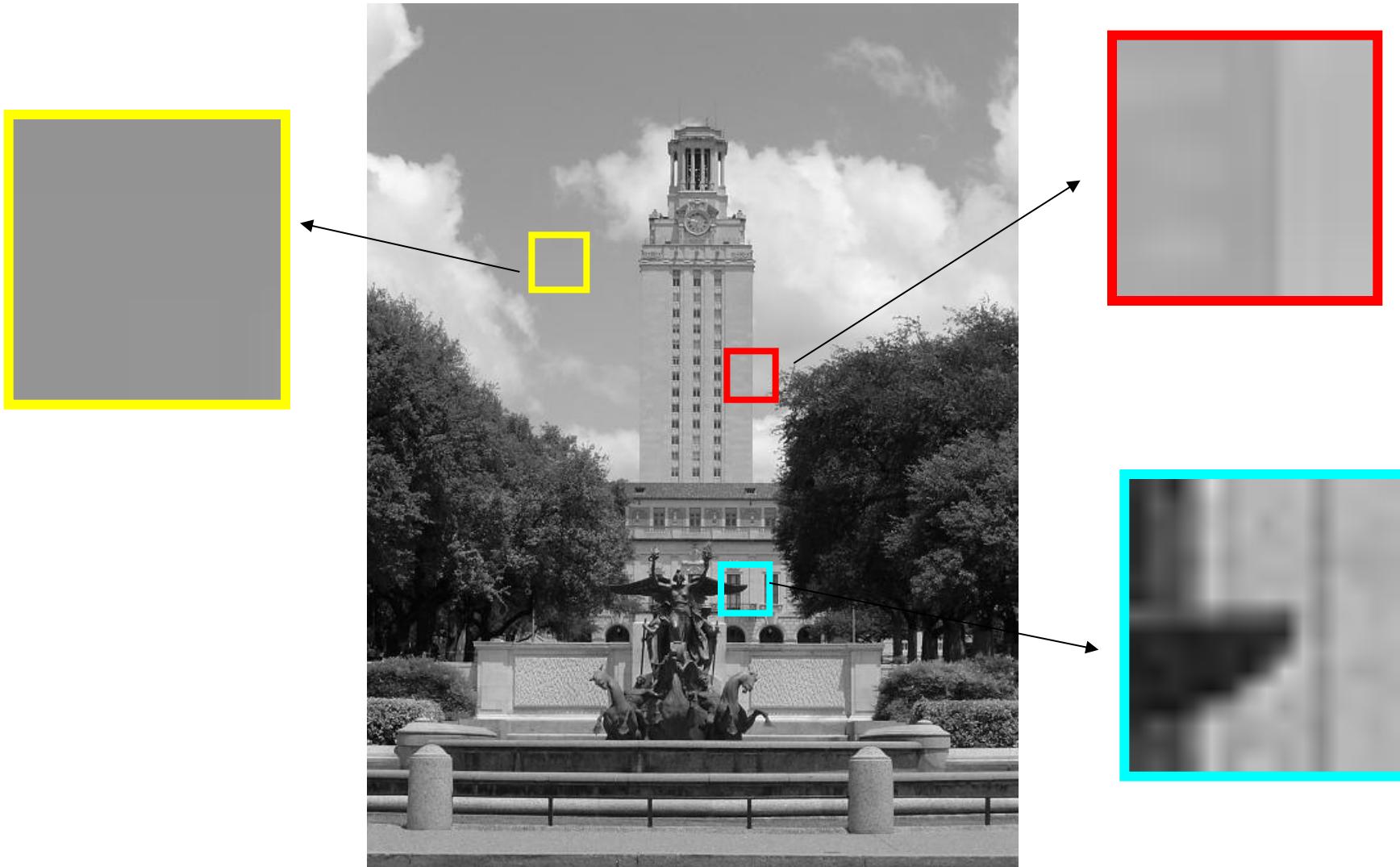
- Depth discontinuity:
object boundary

- Cast shadows



• Slide credit:
Kristen Grauman

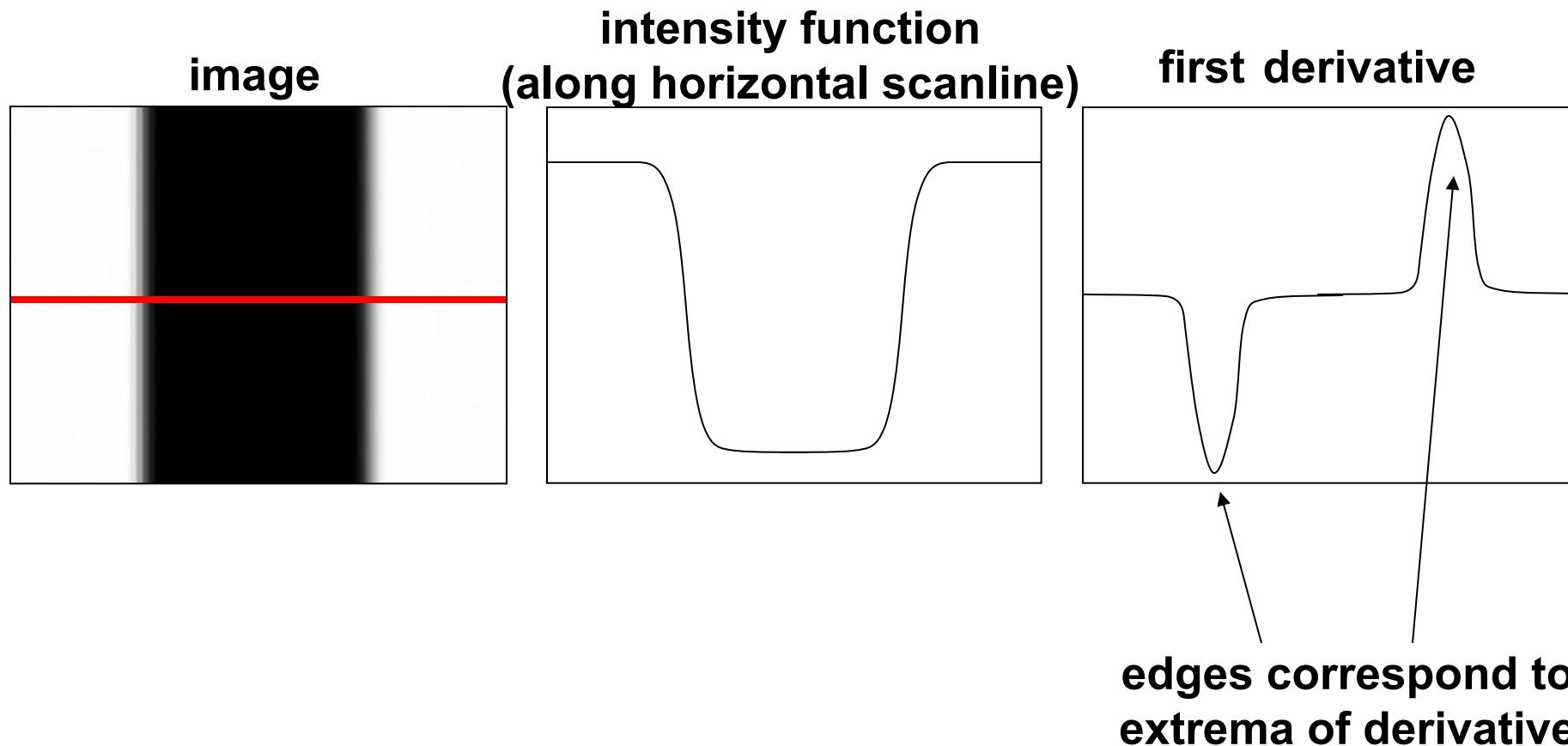
Edges/gradients and invariance



•Slide credit:
Kristen Grauman

Derivatives and edges

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



Methods

- Using first order derivative: *Look for extrema*
 - Sobel operator : `edge(I, 'sobel');`
 - Prewitt, Roberts,...
 - Derivative of Gaussian
- Using second order derivative: *Look for zero-crossings*
 - Laplacian : isotropic
 - Second directional derivative
 - LOG/DOG (Laplace of Gaussian/Difference of Gaussians)

$$\nabla^2 \mathbf{I}$$

Common edge detectors

- Basic gradient edge detectors
 - **Sobel**, Prewitt, Gaussian derivative ...
- **LoG/Marr** edge detector
- **Canny** edge detector

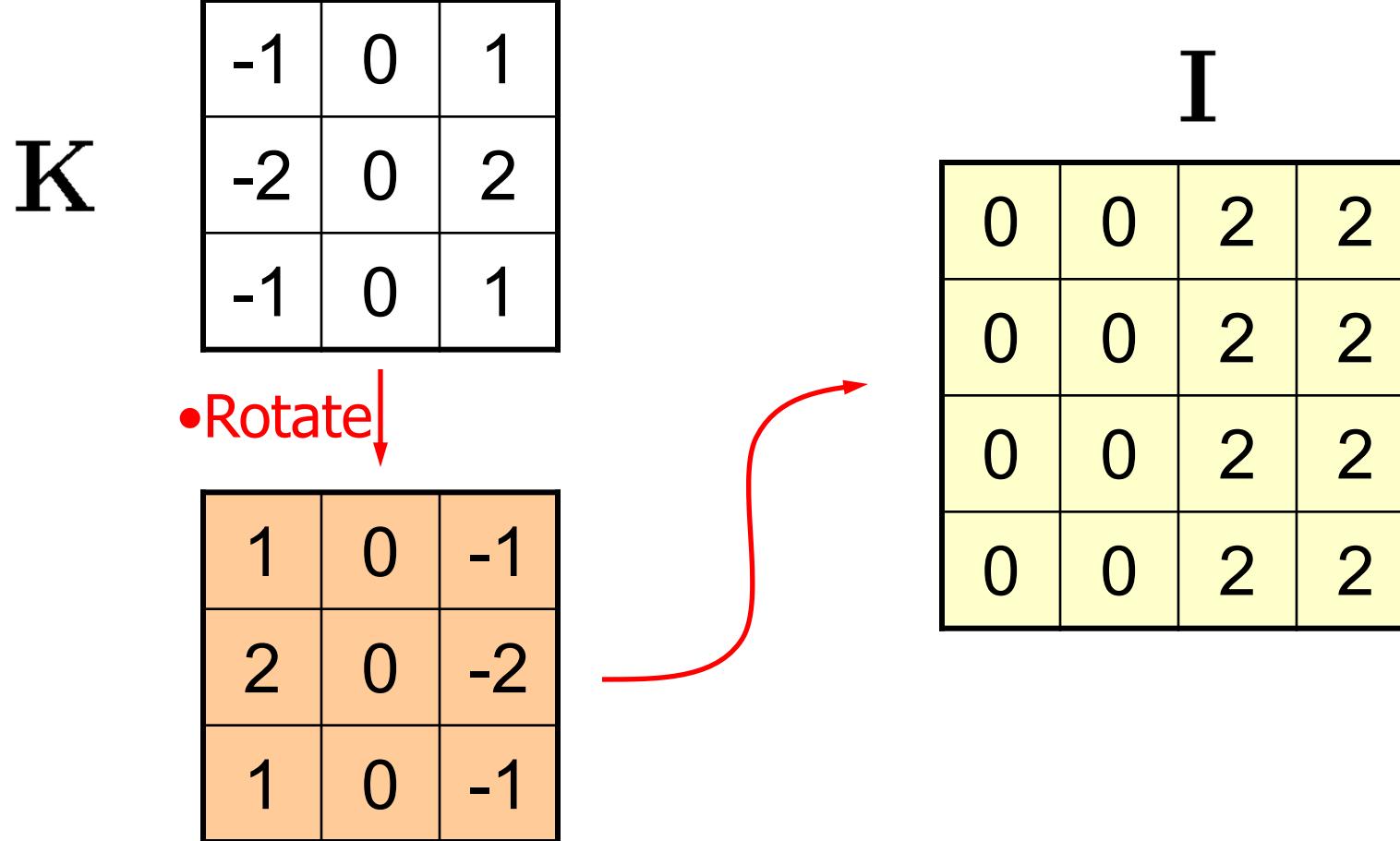
Gradient Edge Detectors

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

Eg: Convolute an image with a Sobel kernel

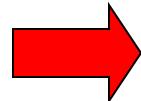


Step

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

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2

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



0			

I

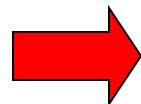
I'

Step

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

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2

1	0	-1	
0	0	-4	2
0	0	-2	2
0	0	2	2
0	0	2	2



0	-6		

I

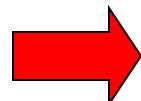
I'

Step

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

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2

	1	0	-1
0	0	0	-4
0	0	0	-2
0	0	2	2
0	0	2	2



0	-6	-6	

I

I'

Step

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

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2

1	0	-1
0	0	4
0	0	2
0	0	2
0	0	2

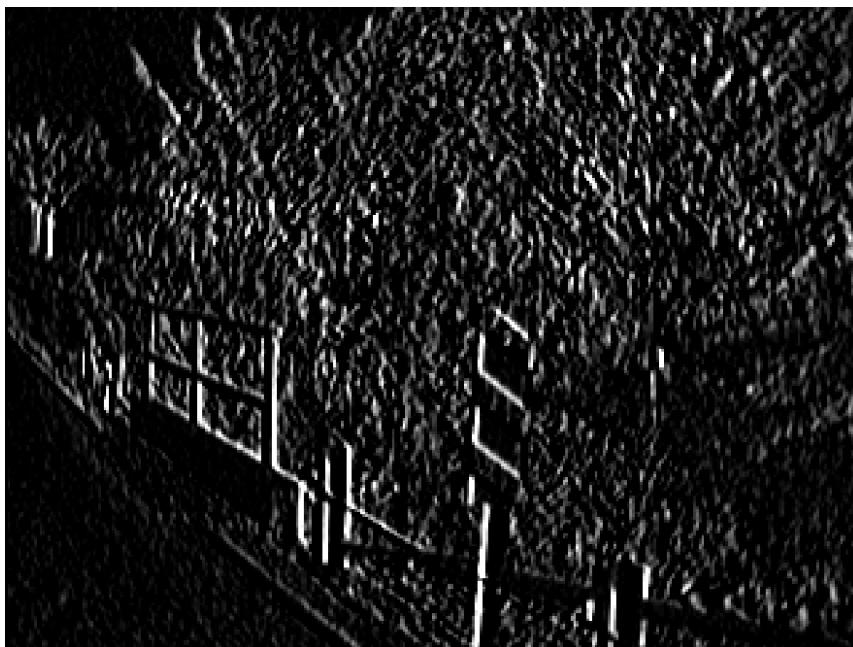
I

0	-6	-6	6

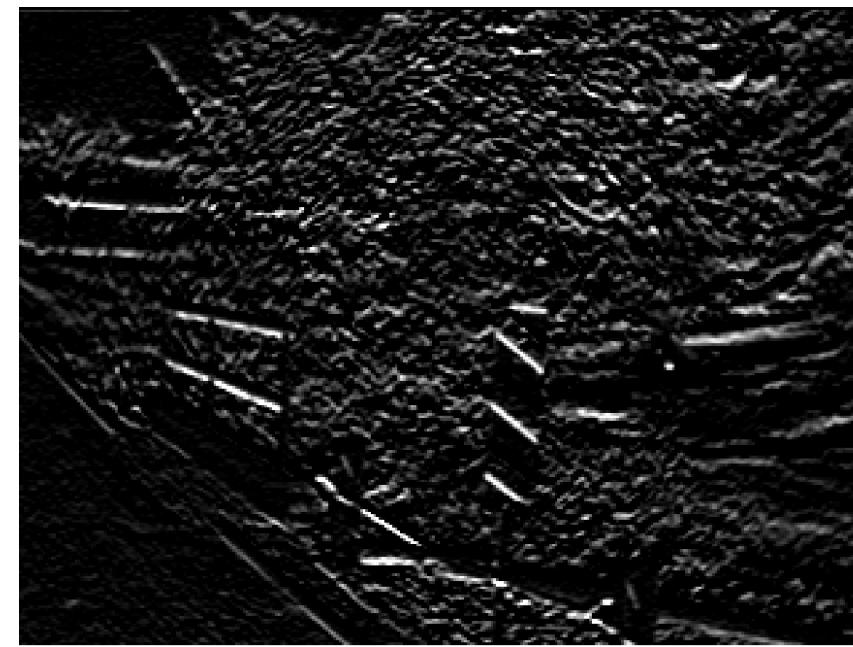
I'

border
artifact

Sobel edge detection: example



- Vertical edges



- Horizontal edges

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 it using finite differences:

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

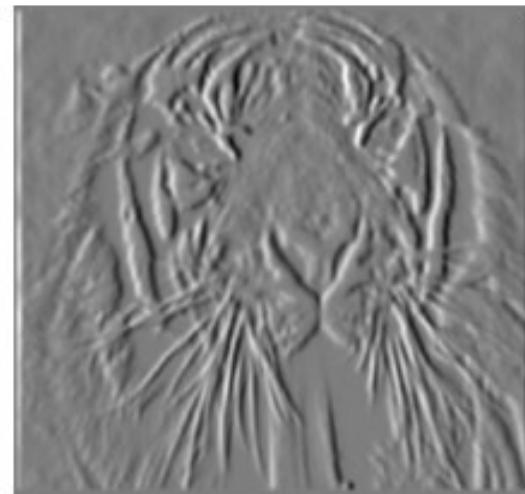
To implement above as convolution, what would be the associated filter?

Partial derivatives of an image



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

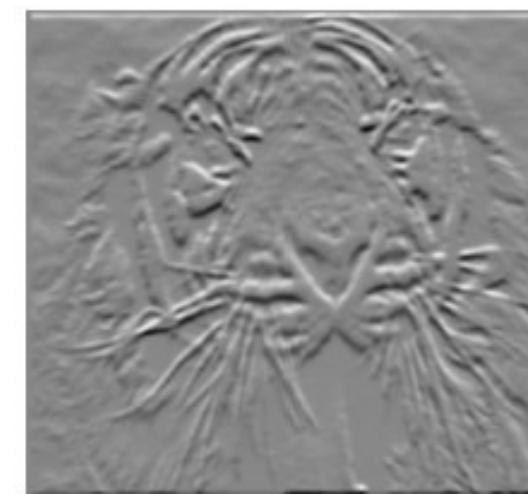
1	-1
---	----



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

-1 or 1

-1	or	1
1		-1



- Which shows changes with respect to x?

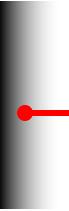
- (showing filters for correlation)

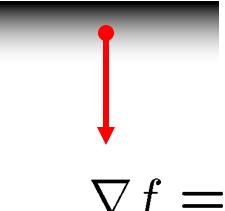
Image gradient

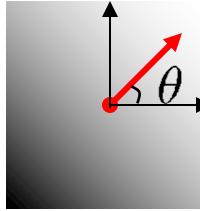
The gradient of an image:

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

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


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


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

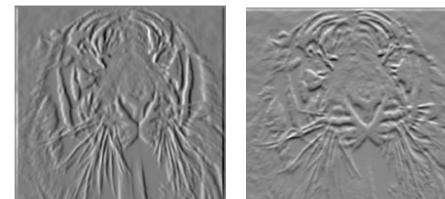

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

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

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

- The **edge strength** is given by the gradient magnitude

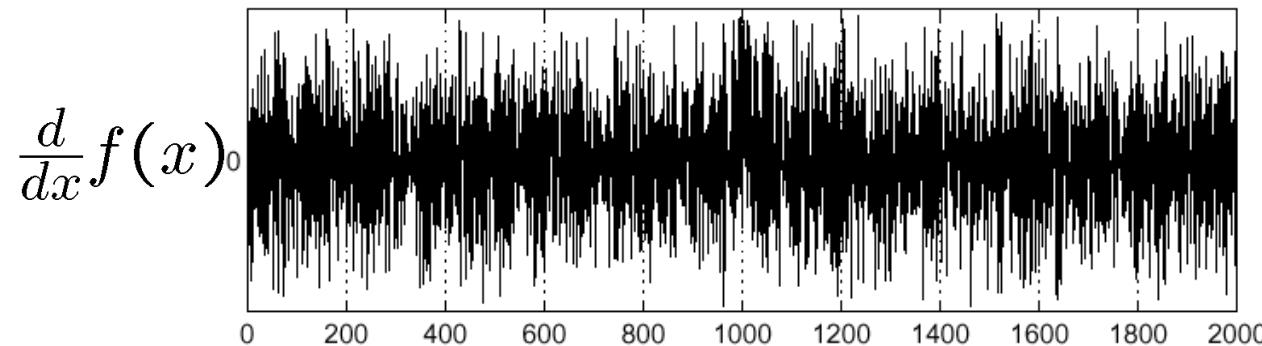
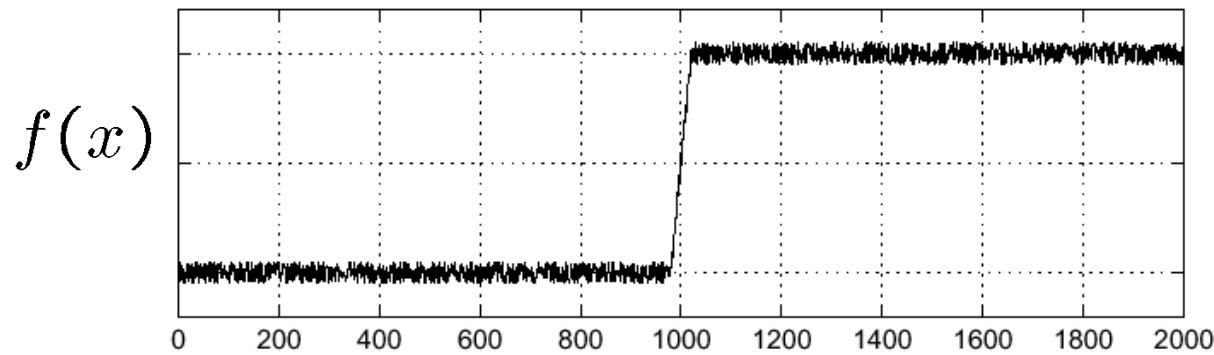
$$\| \nabla f \| = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2}$$



Effects of noise

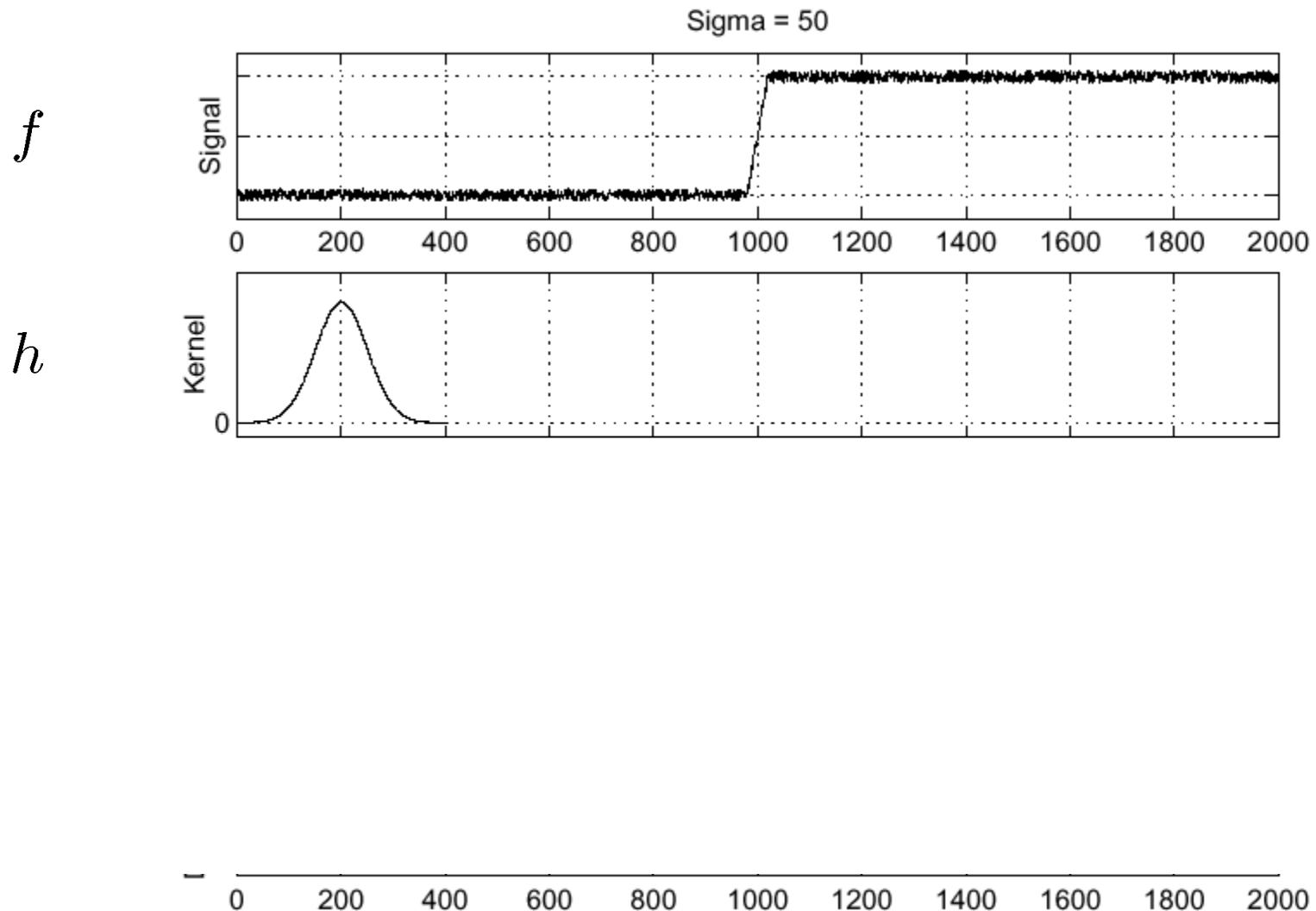
Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal



- Where is the edge?

Solution: smooth first



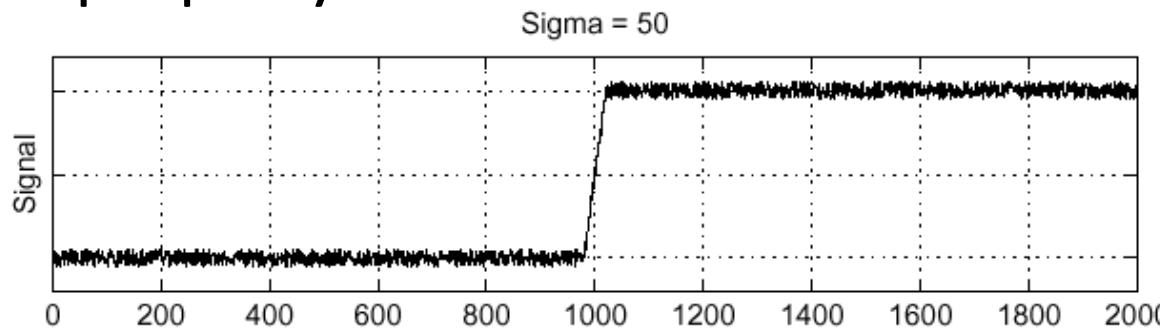
- Where is the edge?
- Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

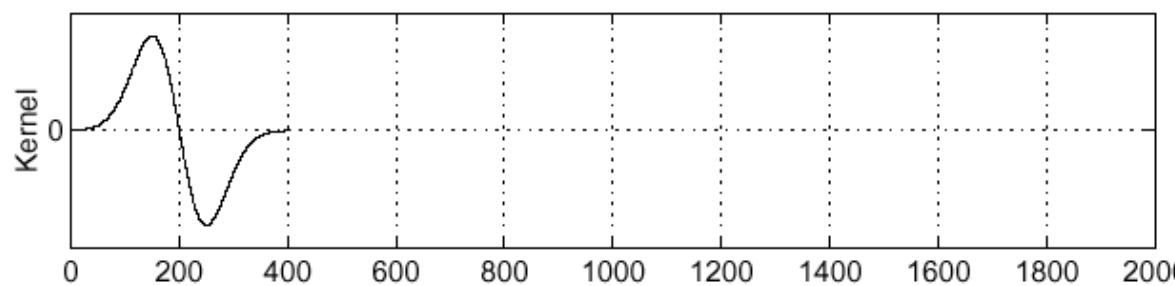
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Differentiation property of convolution.

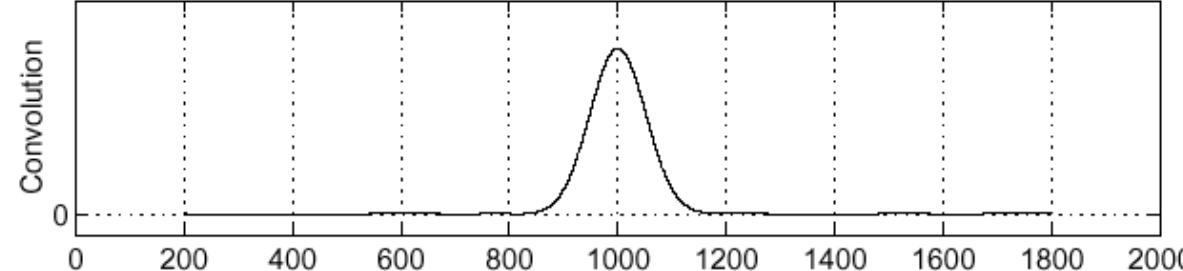
f



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



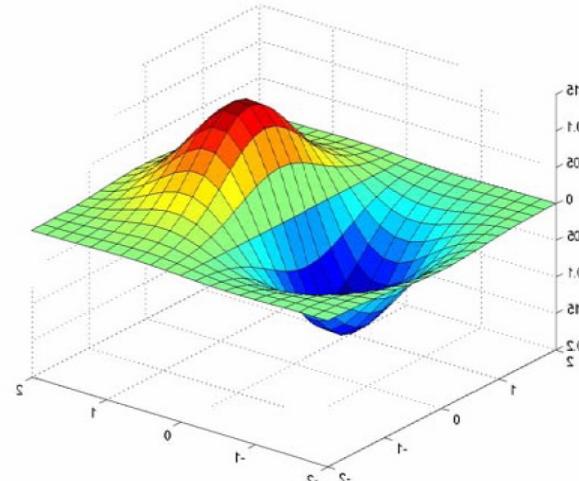
$(\frac{\partial}{\partial x}h) \star f$



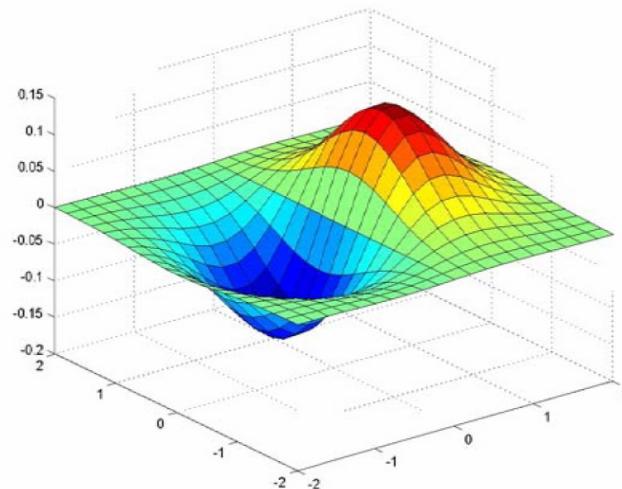
Derivative of Gaussian filters

$$(I \otimes g) \otimes h = I \otimes (g \otimes h)$$

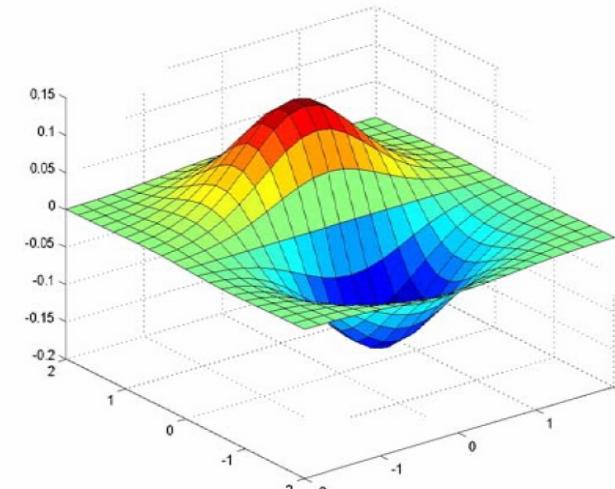
$$\begin{bmatrix} \bullet & 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ \bullet & 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ \bullet & 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ \bullet & 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ \bullet & 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \quad \otimes \quad \begin{bmatrix} 1 & -1 \end{bmatrix}$$



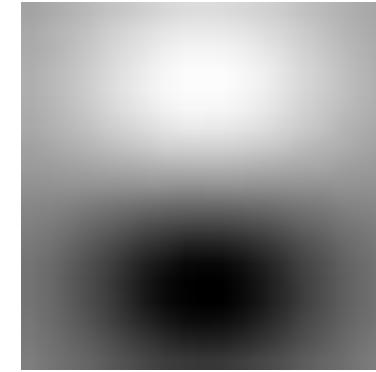
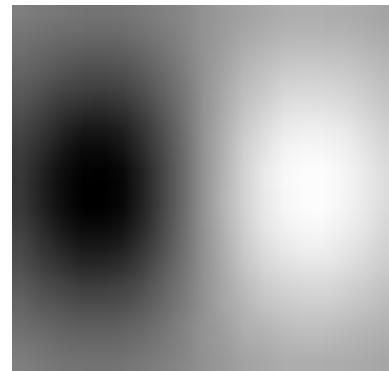
Derivative of Gaussian filters



•x-direction



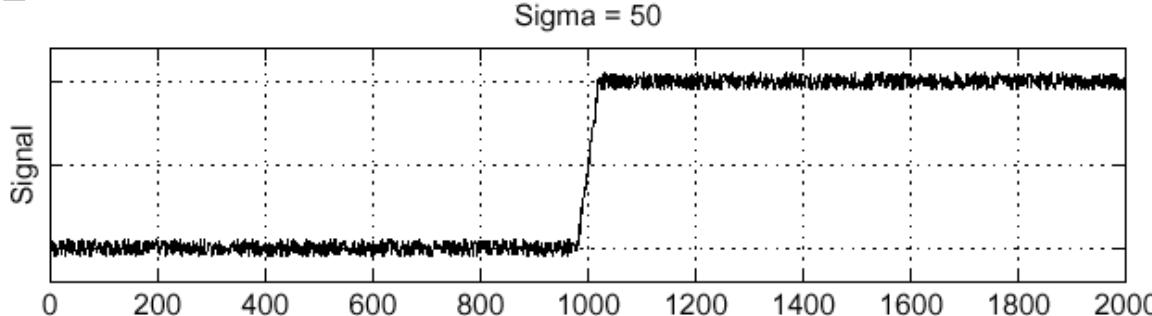
•y-direction



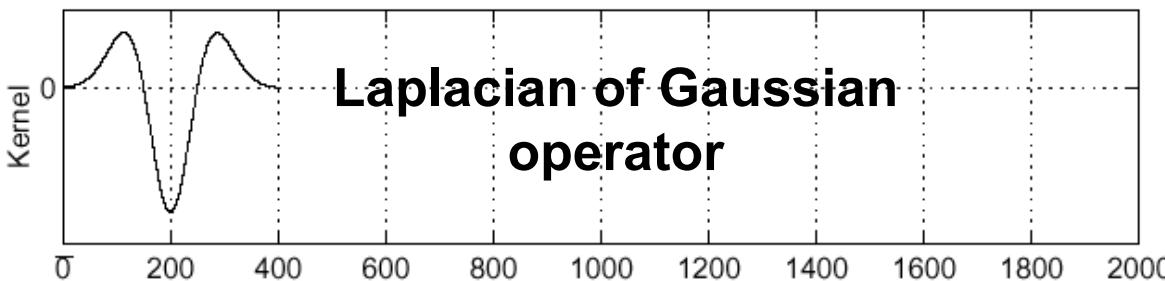
Laplacian of Gaussian

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

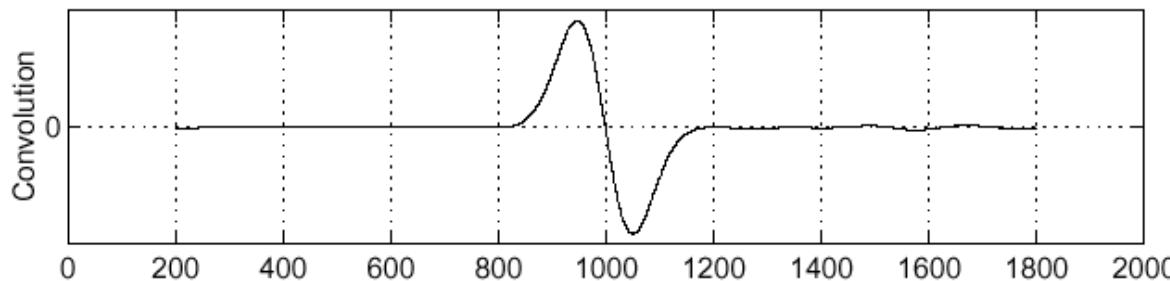
f



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



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

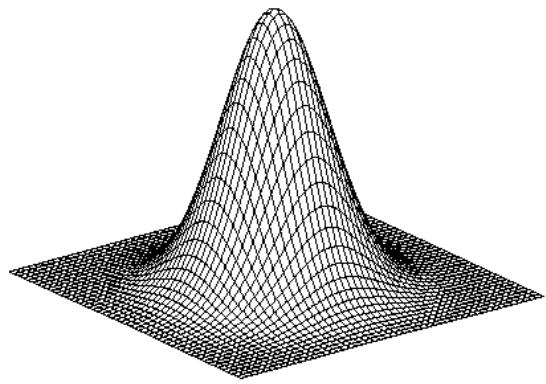


- Where is the edge?

- Zero-crossings of bottom graph

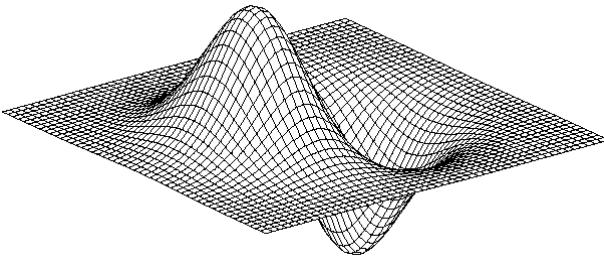
• Slide credit: Steve Seitz

2D edge detection filters



Gaussian

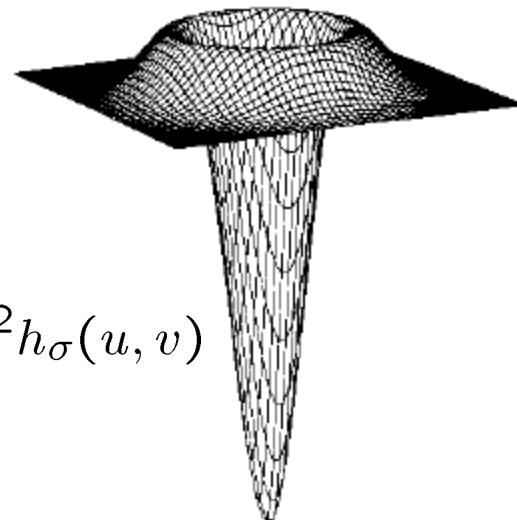
$$h_\sigma(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



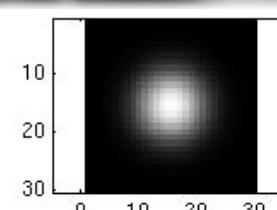
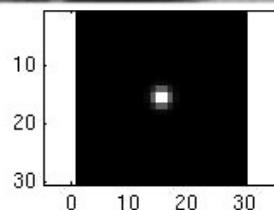
$$\nabla^2 h_\sigma(u, v)$$

- ∇^2 is the Laplacian operator:

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

Smoothing with a Gaussian

- Recall: parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



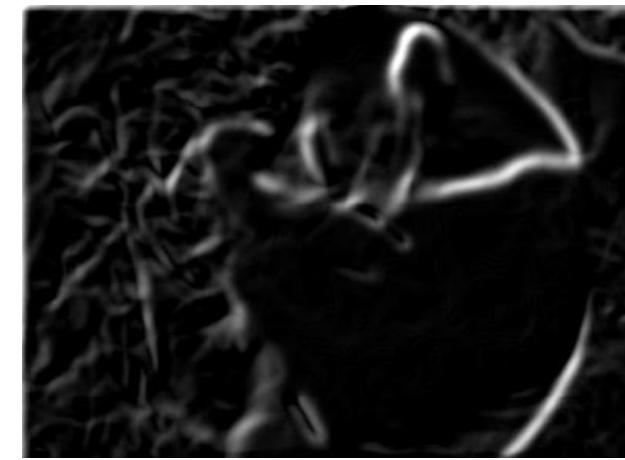
• ...



Effect of σ on derivatives



$\sigma = 1$ pixel



$\sigma = 3$ pixels

- The apparent structures differ depending on Gaussian's scale parameter.
- Larger values: larger scale edges detected
- Smaller values: finer features detected

Readings

- Computer Vision: Algorithms and Applications, Chapter 3.1, 3.2 and 3.3