Image Filtering
-Linear Filtering-

Computer Labs

- Lab enrolments are open again, please signup before 11:59 pm Thursday. No more extensions will be given!
- 3 Groups:
  - Clab-group-1: Thursday (10-12)
  - Clab-group-2: Friday (12-14)
  - Clab-group-3: Friday(14-16), new time!
- First lab assignment:
  - Due on August 23rd, 2020.
  - Late submissions will incur 10% per day (>7 days: 0 marks)
- No mid-term exam.

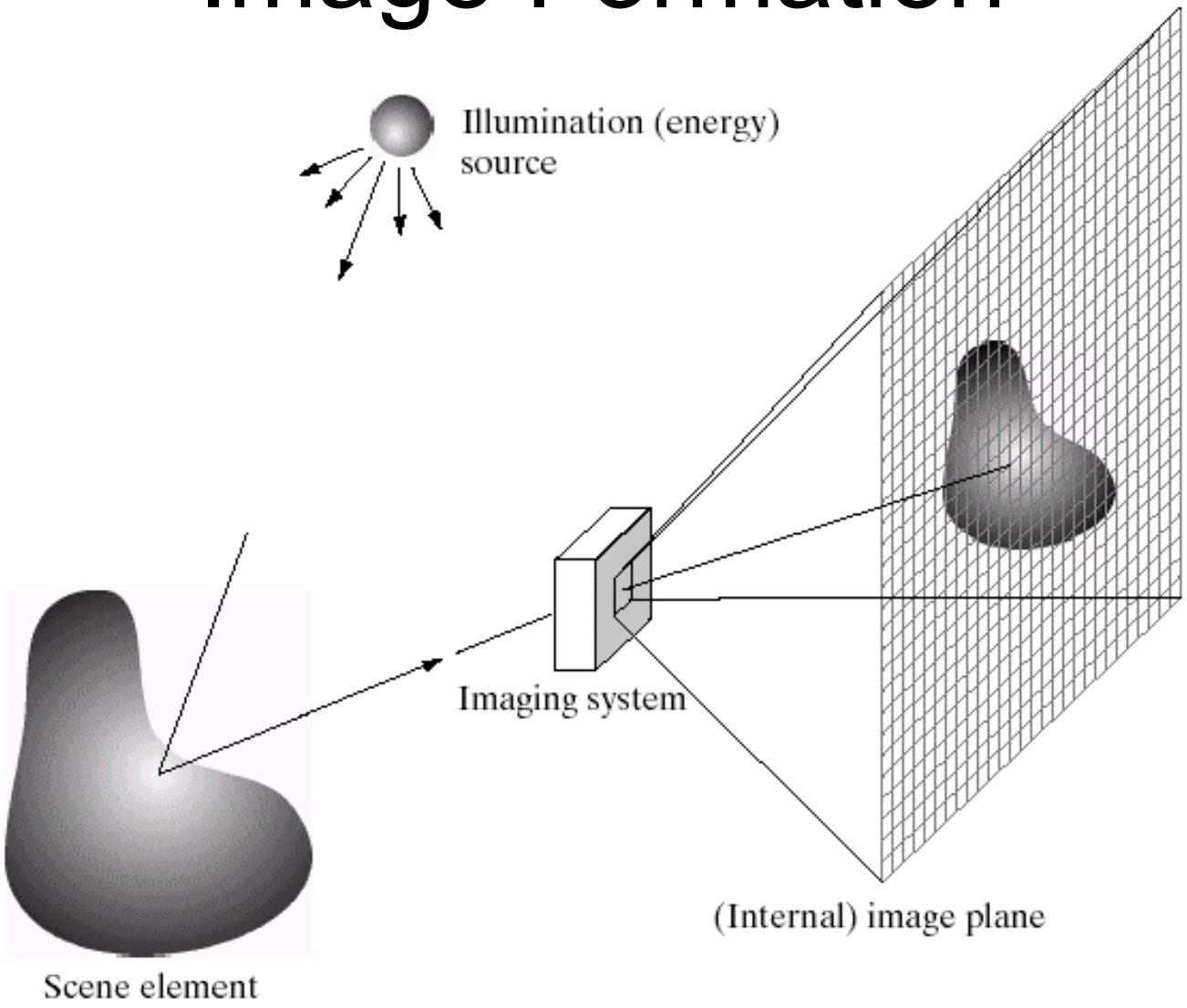
Lecture schedule (Tentative)

Week	**Tuesday (2-hr)**	**Lecturer**	**Wednesday (1-hr)**	**Lecturer**
Week-3	Image filtering	Ajanthan	Tutorial-2	Sameera
Week-4	Line fitting, corners	Richard	SIFT	Richard
Week-5	Segmentation, clustering	Richard	Tutorial-3, clab-2 intro	Sameera
Week-6	PCA, eigen faces	Richard/Ajanthan	TBD	TBD
Week-7	3D-vision-A	Richard	3D-vision-B	Richard
Week-8	3D-vision-C	Richard	Tutorial-4, clab-3 intro	Ryan
Week-9	Stereo, SLAM	Richard/Ajanthan	TBD	TBD
Week-10	Classification basics, SVM	Richard/Ajanthan	TBD	TBD
Week-11	Deep learning-A	Ajanthan	Deep learning-B	Ajanthan
Week-12	Guest lecture	TBD	Review	Ajanthan

Plan for today

- Review Photometric Image Formation
- Image Noise
- Linear Filtering
  - Correlation & Convolution
  - Gaussian Filter
  - Edge Detection

Image Formation



Slide credit: Derek Hoiem

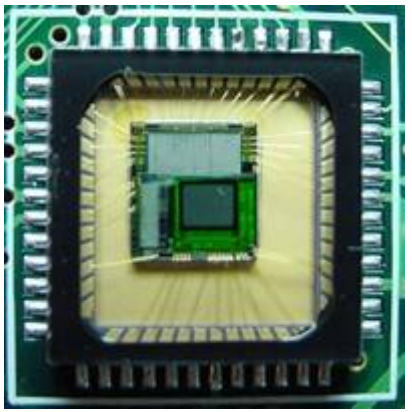
Digital camera

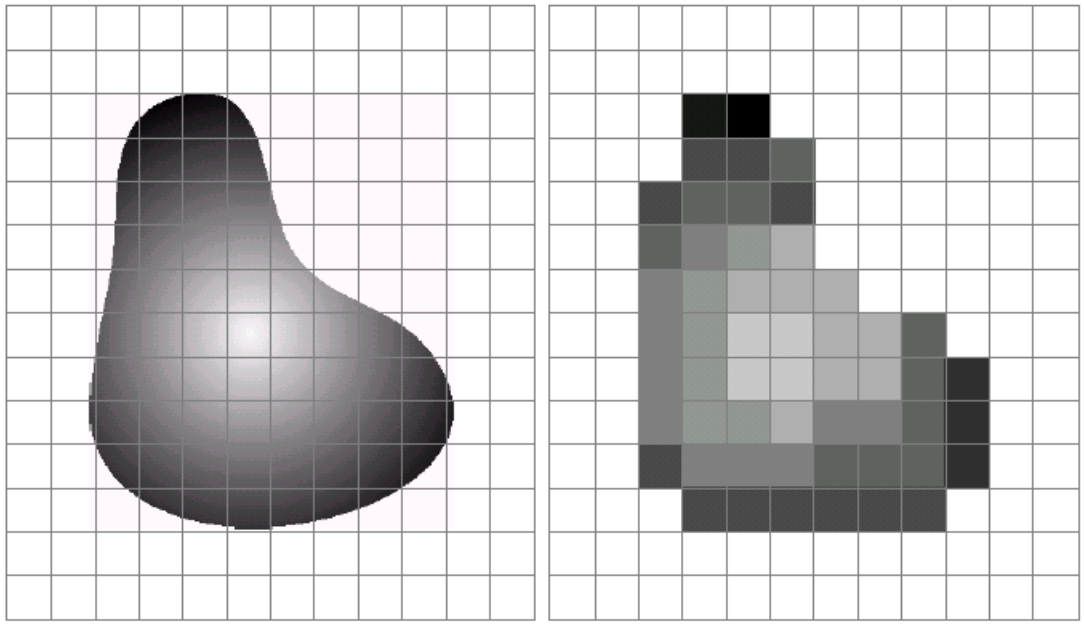
A digital camera replaces film with a sensor array

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

Slide by Steve Seitz

Digital images





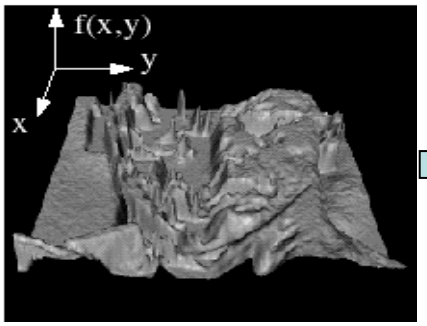
a b

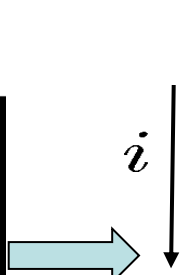
FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.

Slide credit: Derek Hoiem

Digital images

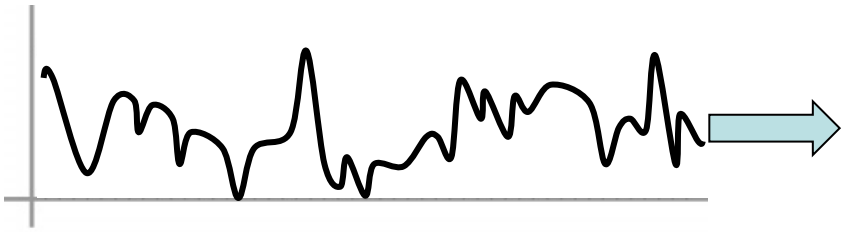
- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)
- Image thus represented as a matrix of integer values.

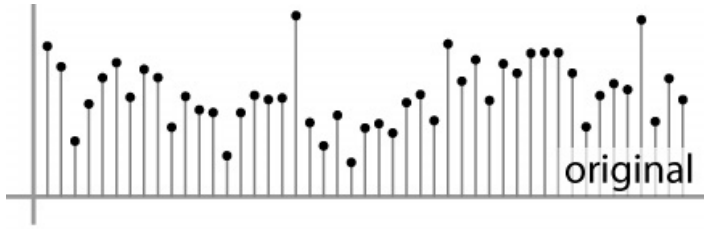




62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

2D

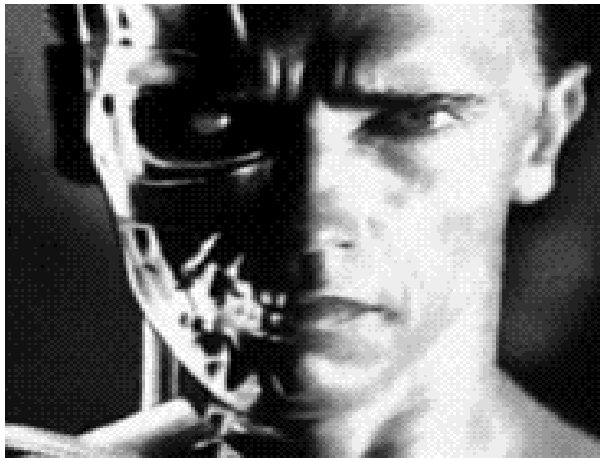


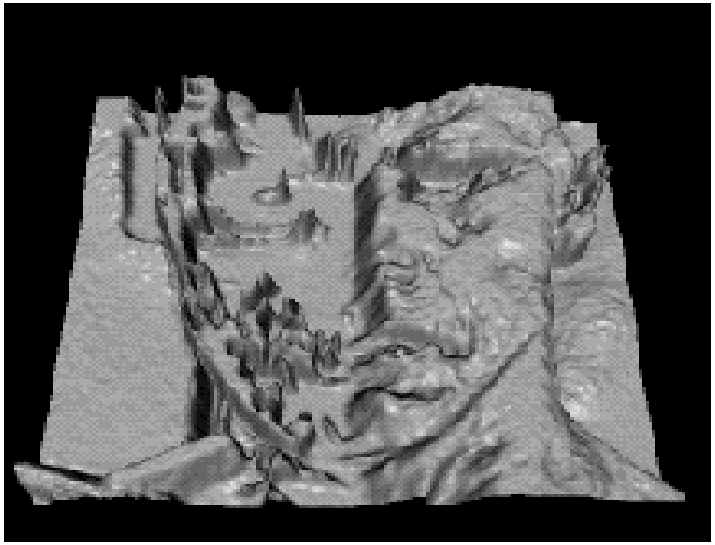


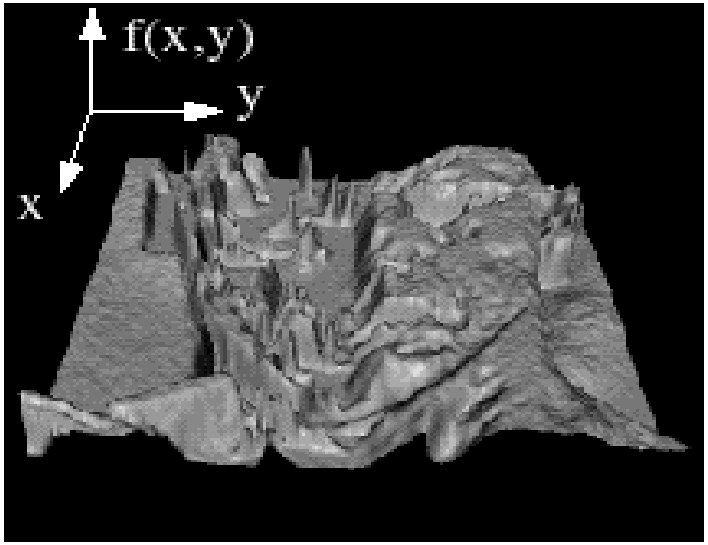
1D

Adapted from S. Seitz

Images as functions

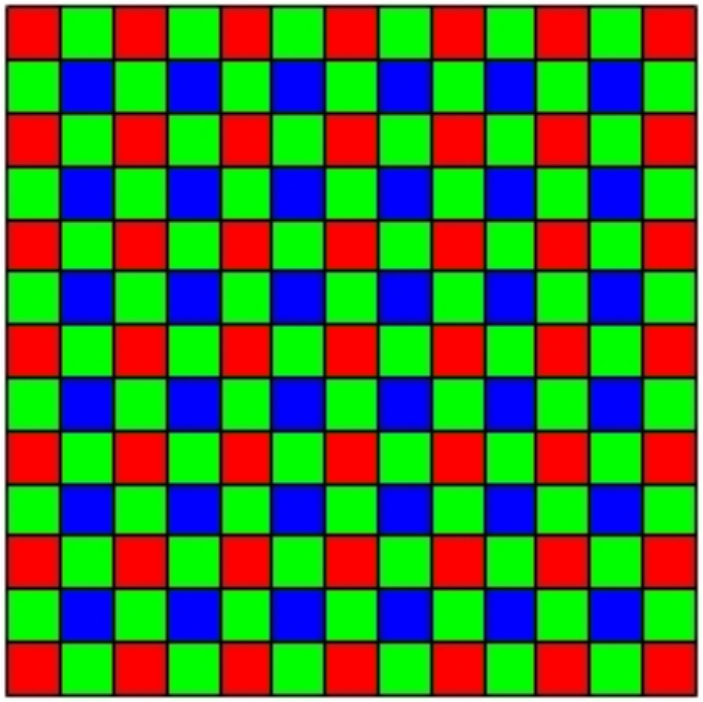






Source: S. Seitz

Digital color images



Bayer filter

Slide credit:
Kristen Grauman

Digital color images

Color images, RGB color space

R

G

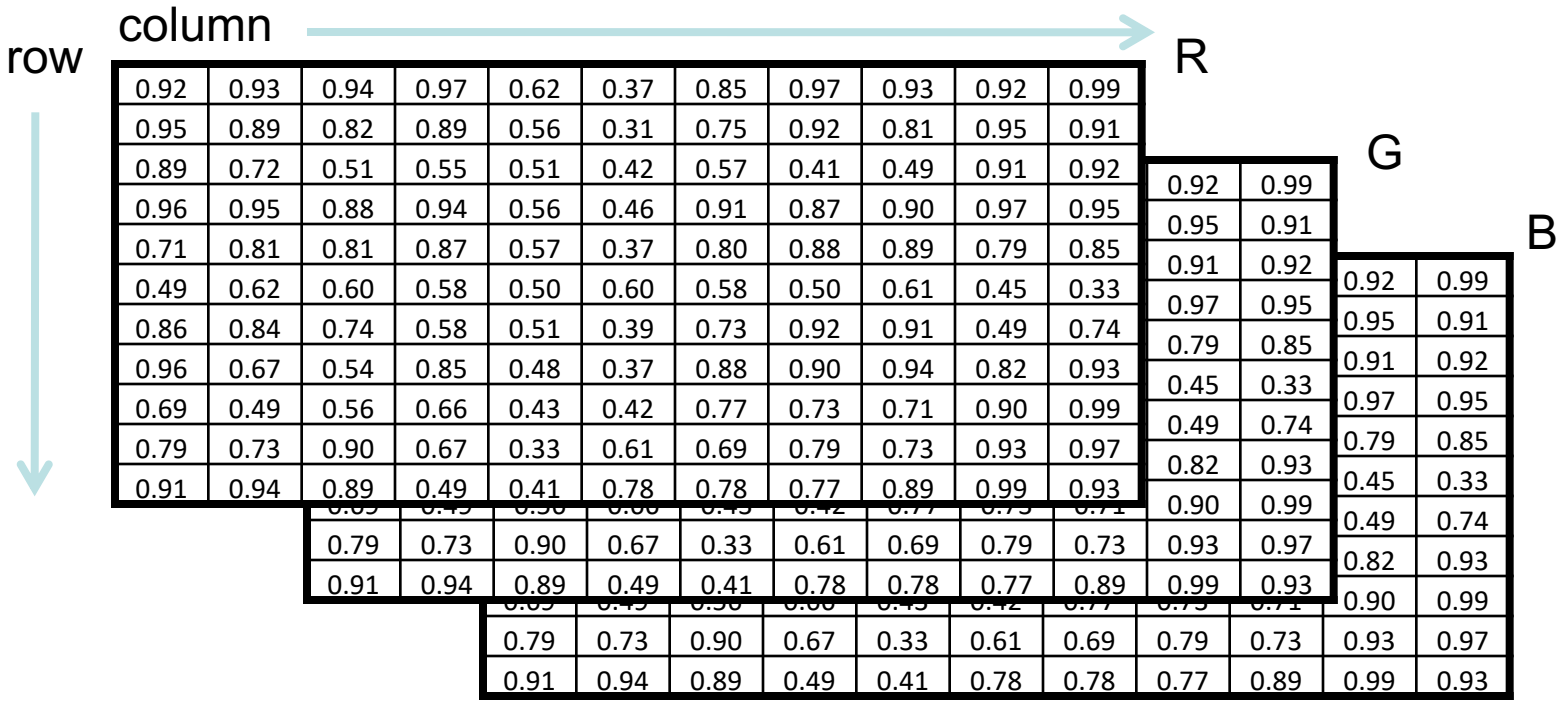
B

Images as functions

- We can think of an **image** as a function, *f*, from R^2 to 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:
    - $f: [a, b] \times [c, d] \rightarrow [0,1]$

Images in Matlab

- Images represented as a matrix
- Suppose we have a NxM RGB image called “im”
  - im(1,1,1) = top-left pixel value in R-channel
  - im(y, x, b) = y pixels down, x pixels to right in the bth channel
  - im(N, M, 3) = bottom-right pixel in B-channel
- imread(filename) returns a uint8 image (values 0 to 255)
  - Convert to double format (values 0 to 1) with im2double



Slide credit: Derek Hoiem

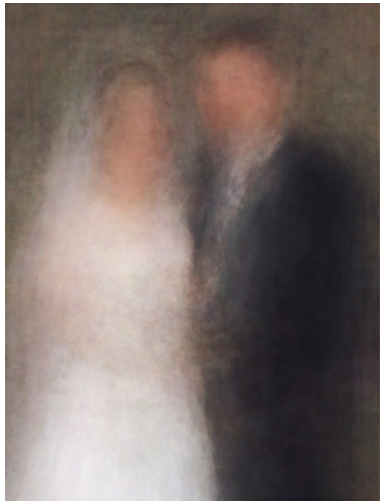
Images as matrices

Result of averaging 100 similar snapshots

Little Leaguer

Kids with Santa

The Graduate



Newlyweds

From: *100 Special Moments*, by Jason Salavon (2004)
http://salavon.com/SpecialMoments/SpecialMoments.shtml

Slide credit:
Kristen Grauman

Image Noise

Common types of noise

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Original

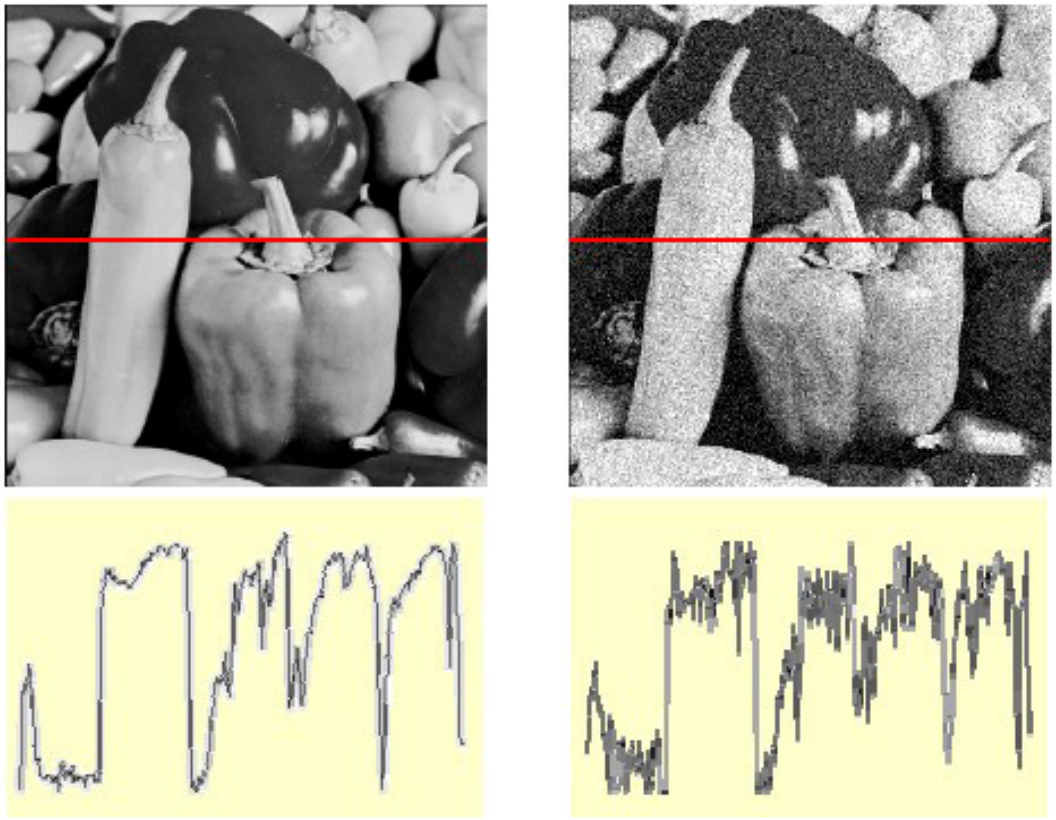
Salt and pepper noise

Impulse noise

Gaussian noise

Source: S. Seitz

Gaussian noise



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

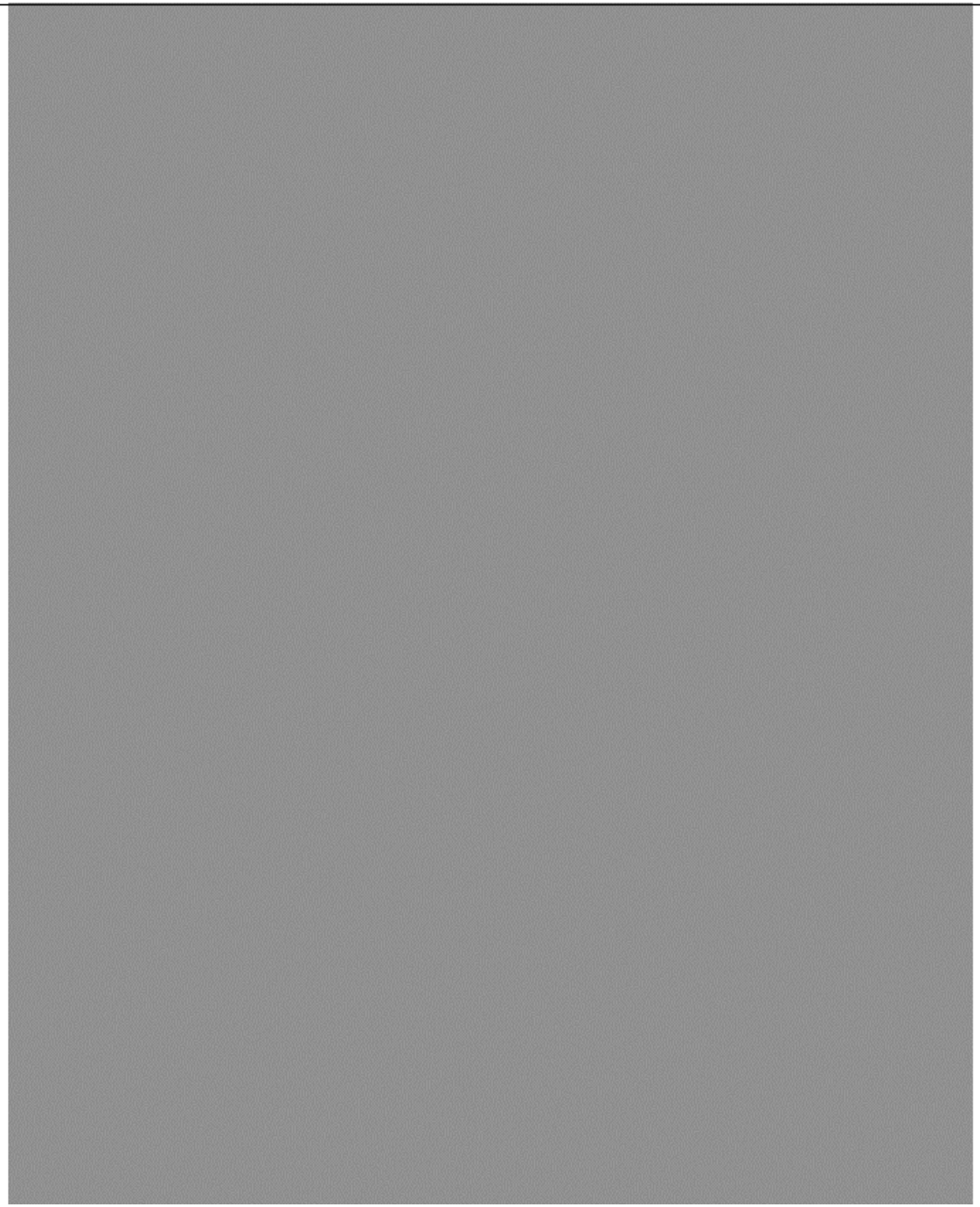
Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

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

What is impact of the sigma?

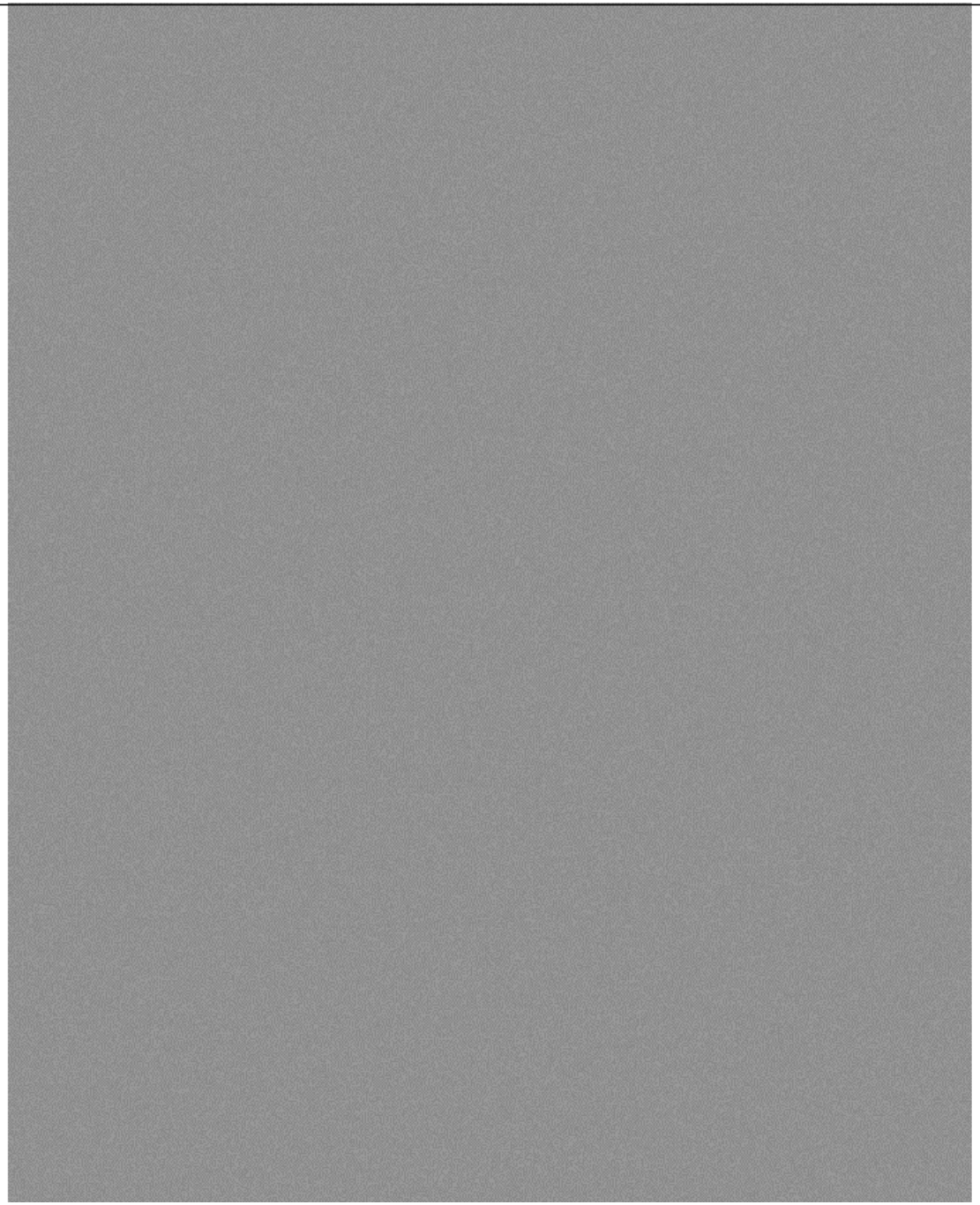
Fig: M. Hebert

sigma=1



Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

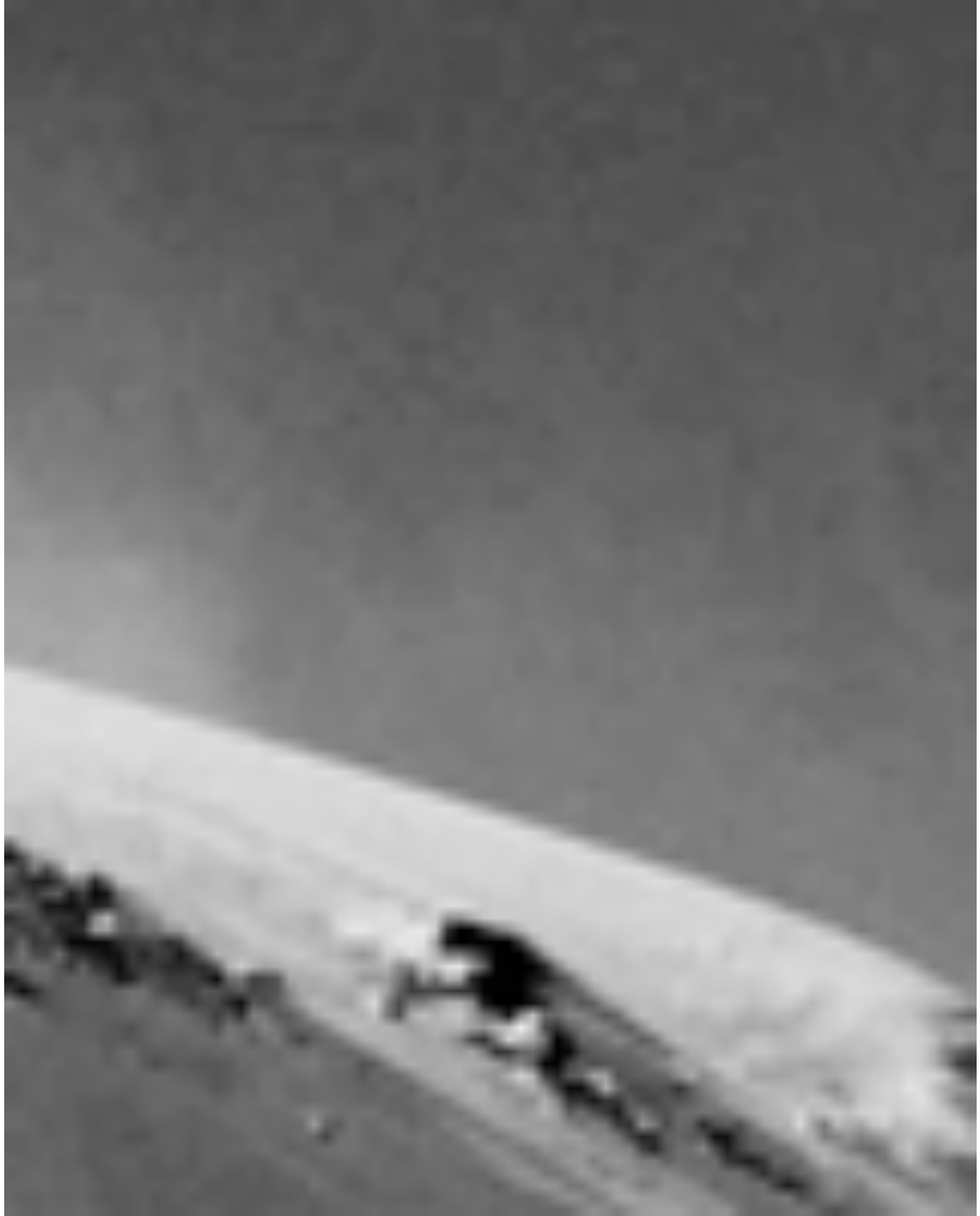


sigma=4

Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

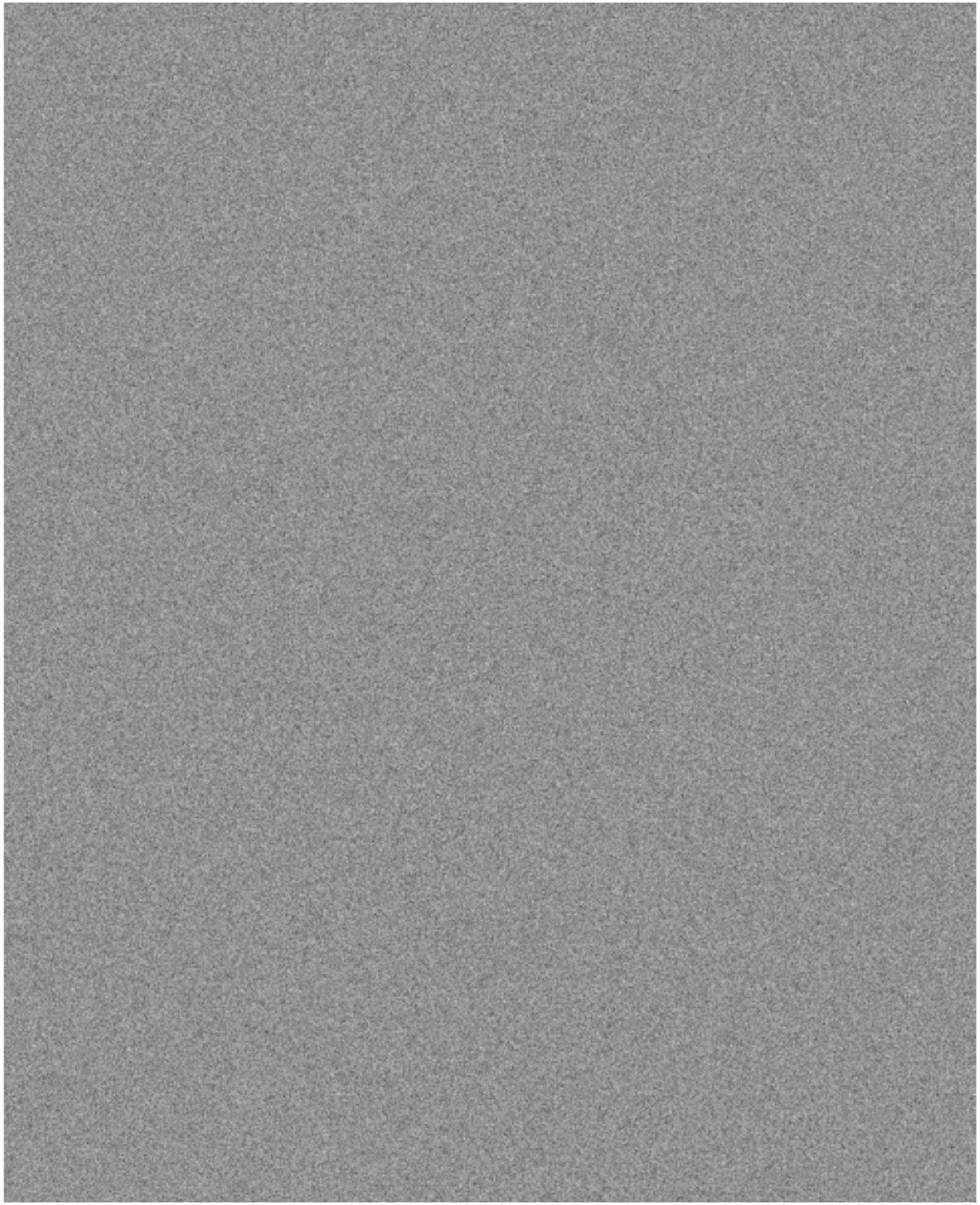
sigma=1



Effect of sigma on Gaussian noise:

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

sigma=
16



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.

Noise Reduction

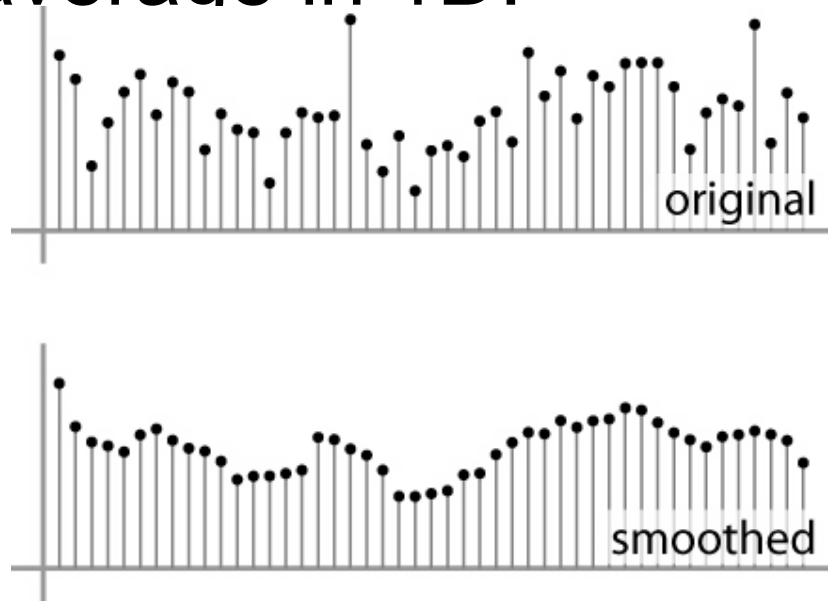
- How should we reduce the noise?

First attempt at a solution

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

First attempt at a solution

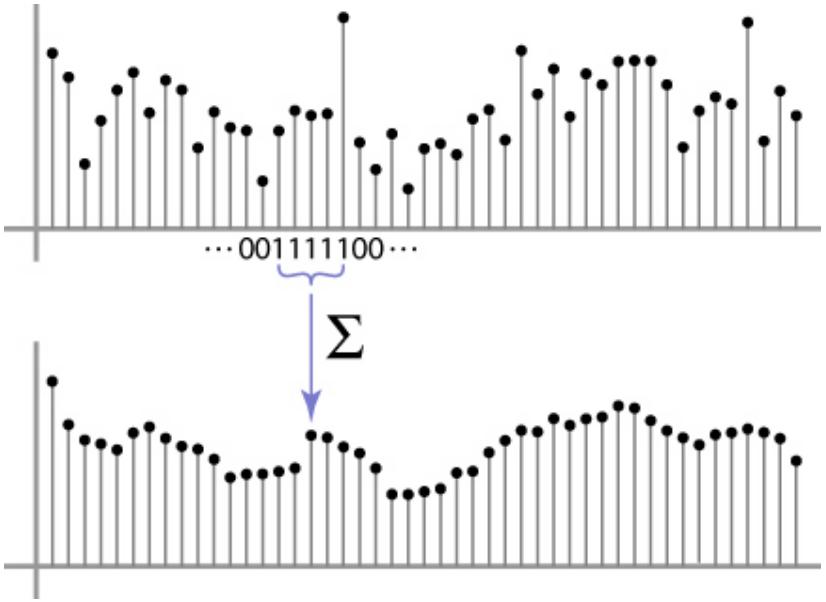
- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



•Source: S. Marschner

Weighted Moving Average

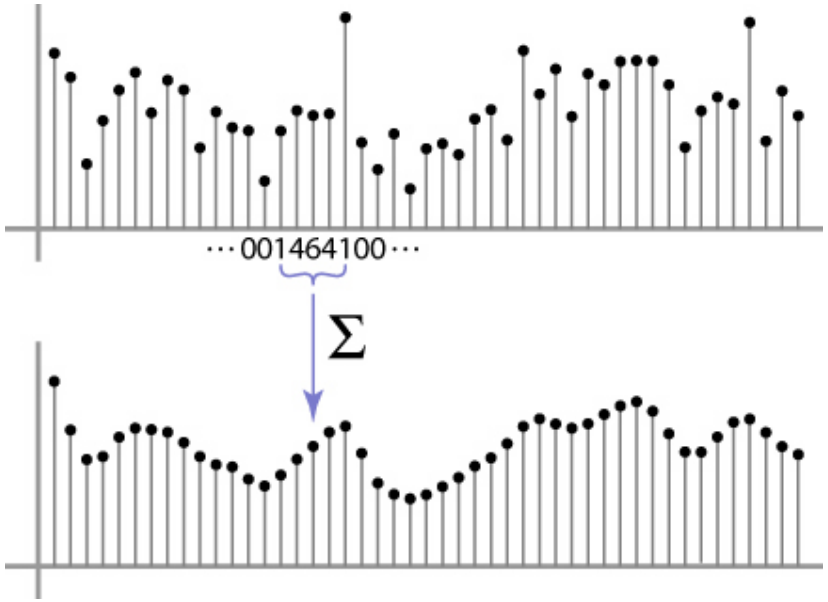
- Can add weights to our moving average
- *Weights* [1, 1, 1, 1, 1] / 5



•Source: S. Marschner

Weighted Moving Average

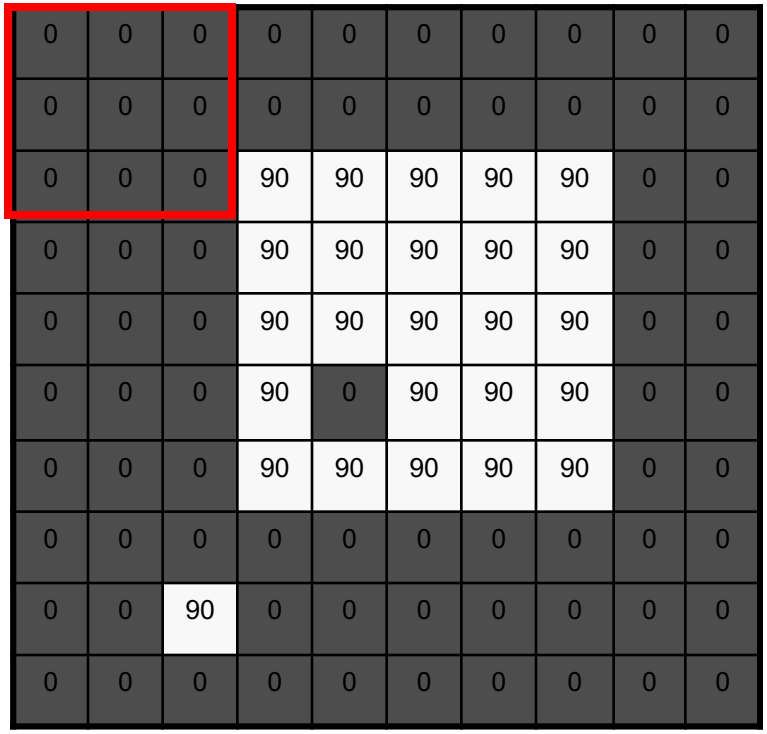
- Non-uniform weights [1, 4, 6, 4, 1] / 16



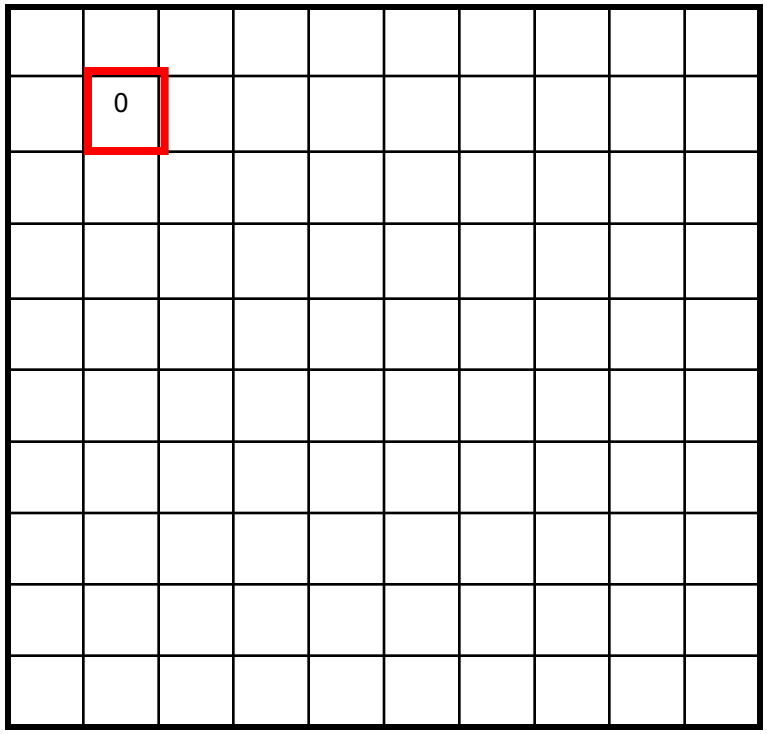
•Source: S. Marschner

Moving Average In 2D

$F[x, y]$



$G[x, y]$



•Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10							

•Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20						

•Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20	30					

•Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30				

•Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

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

•Source: S. Seitz

Correlation filtering

Say the averaging window size is (2k+1) x (2k+1):

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

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

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

Slide credit:
Kristen Grauman

Correlation filtering

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

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

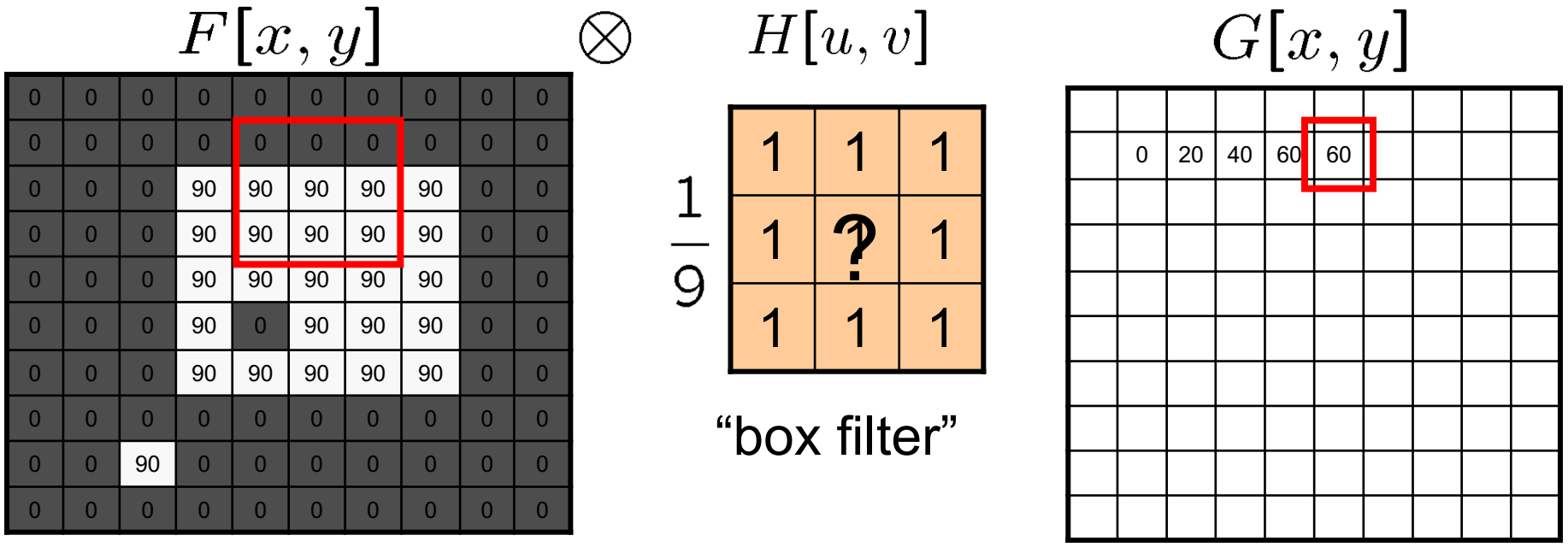
Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter “kernel” or “mask” *H*[*u,v*] is the prescription for the weights in the linear combination.

Slide credit:
Kristen Grauman

Averaging filter

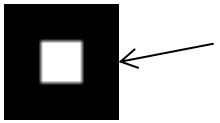
- What values belong in the kernel *H* for the moving average example?



"box filter"

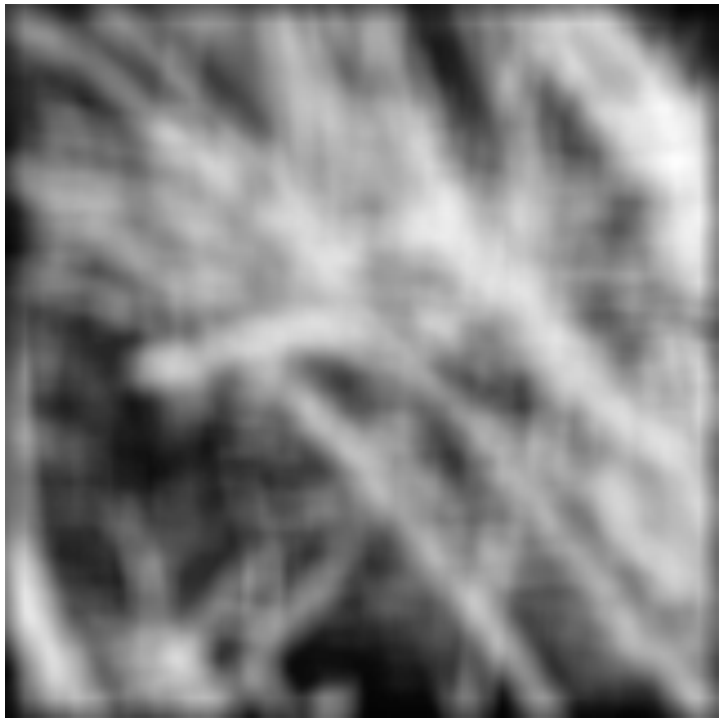
$$G = H \otimes F$$

Smoothing by averaging



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

original



filtered

What if the filter size was 5 x 5 instead of 3 x 3?

Image Filtering

- Image filtering: compute the function value of local neighborhood 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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

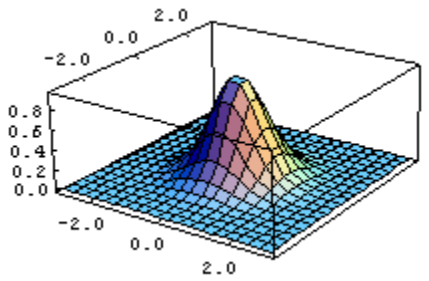
$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$H[u, v]$

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

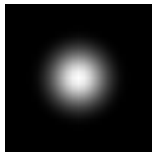
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

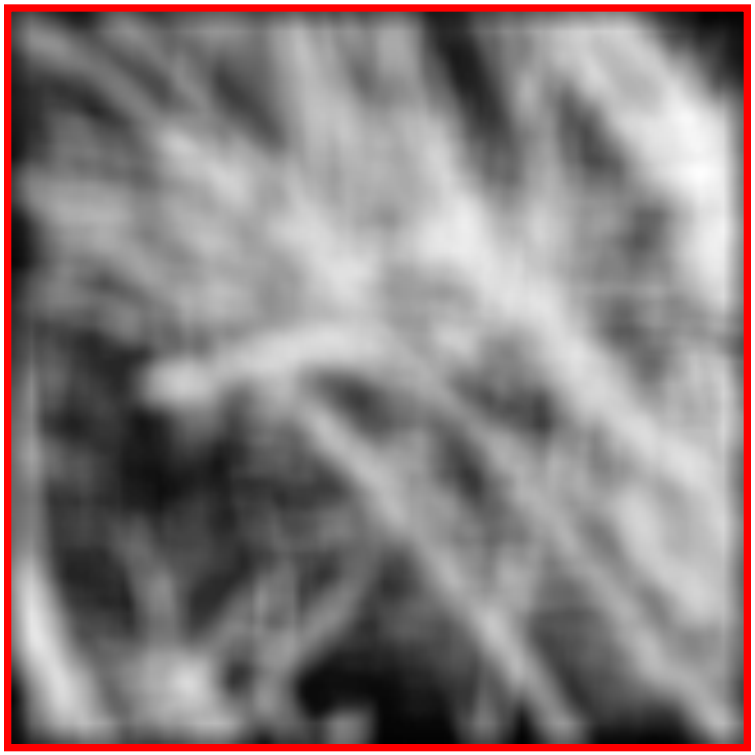


- Removes high-frequency components from the image ("low-pass filter").

Source: S. Seitz

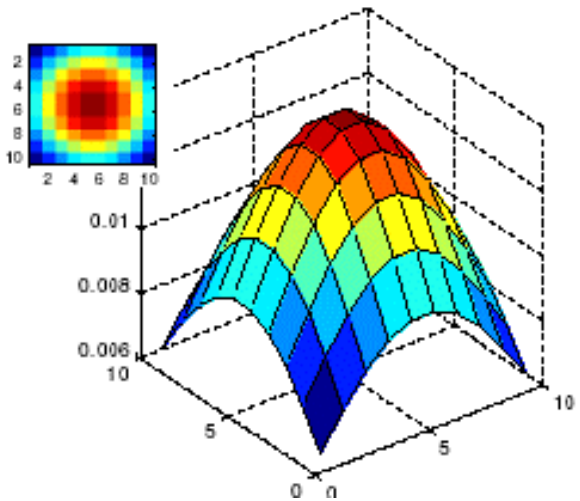
Smoothing with a Gaussian



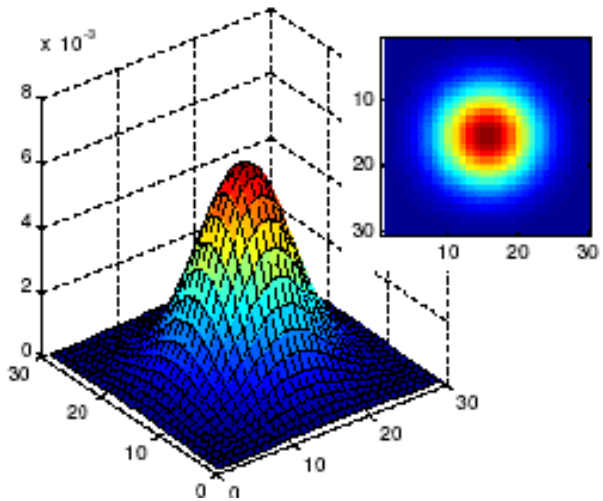


Gaussian filters

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



σ = 5 with 10 x 10 kernel

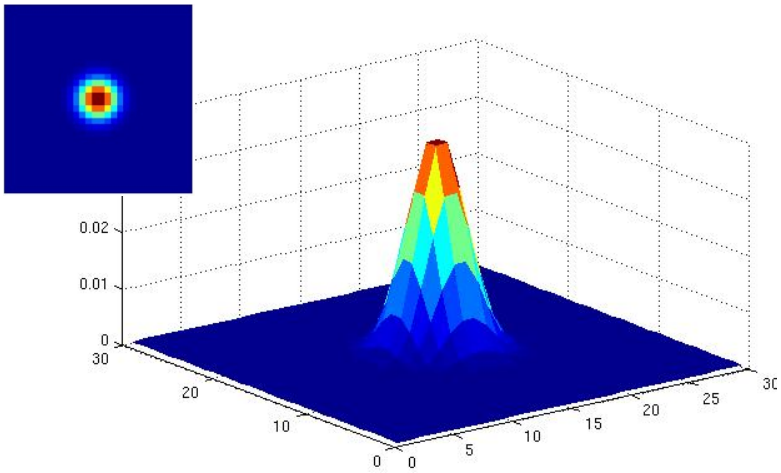


σ = 5 with 30 x 30 kernel

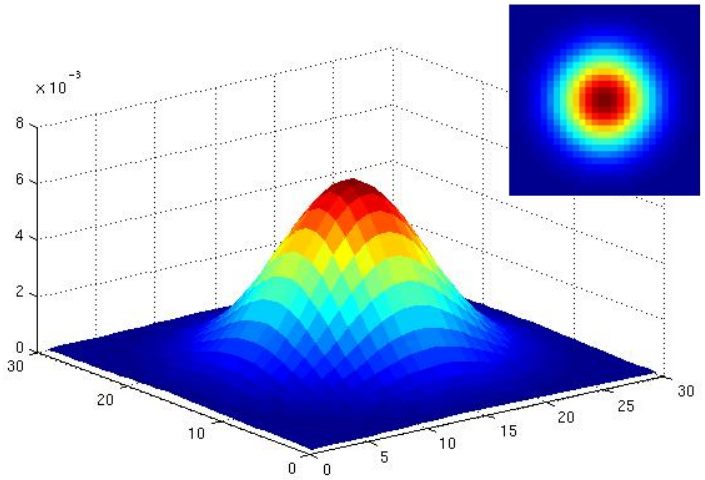
•Slide credit: Kristen Grauman

Gaussian filters

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



σ = 2 with 30 x 30 kernel



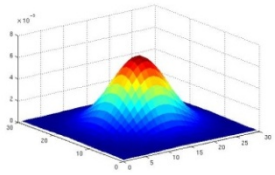
σ = 5 with 30 x 30 kernel

•Slide credit: Kristen Grauman

Matlab

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

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



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

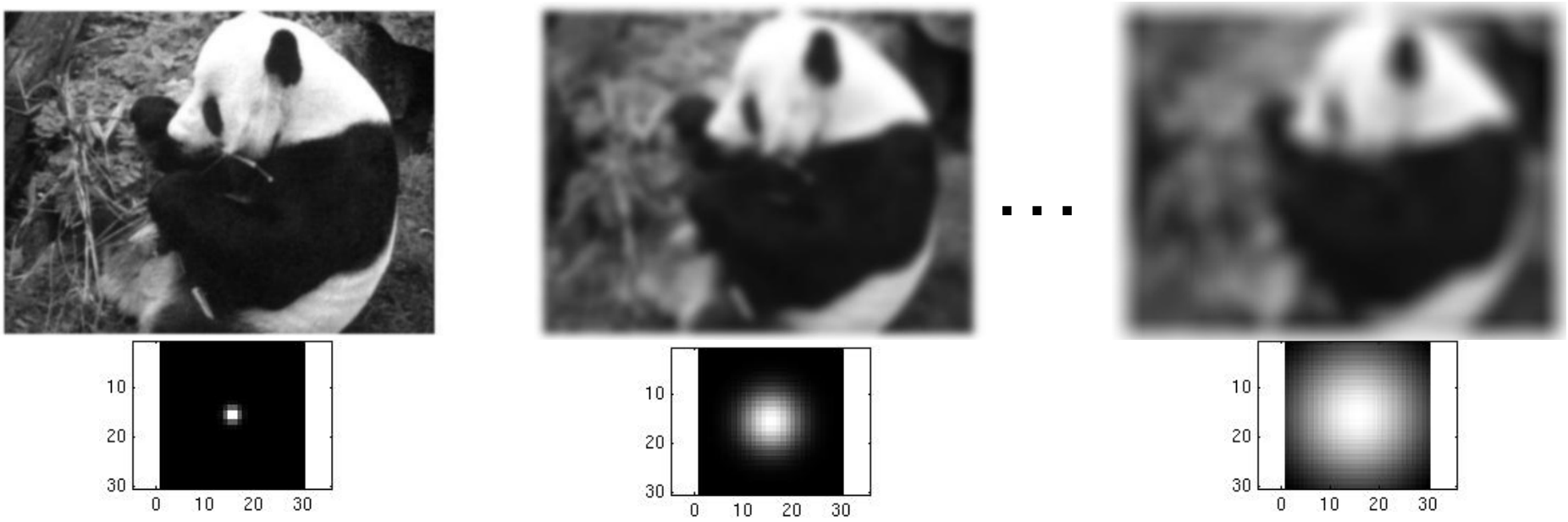
```
>> outim = imfilter(im, h); % correlation
>> imshow(outim);
```

outim

•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.




```
for sigma=1:3:10
   h = fspecial('gaussian', fsize, sigma);
   out = imfilter(im, h);
   imshow(out);
   pause;
end
```

•Slide credit: Kristen Grauman

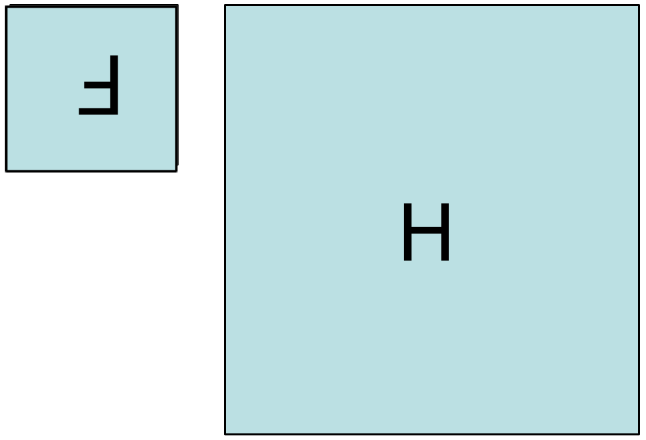
Convolution

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

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

$$G = H \star F$$

Notation for convolution operator



Convolution

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

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

$$G = F \star H$$

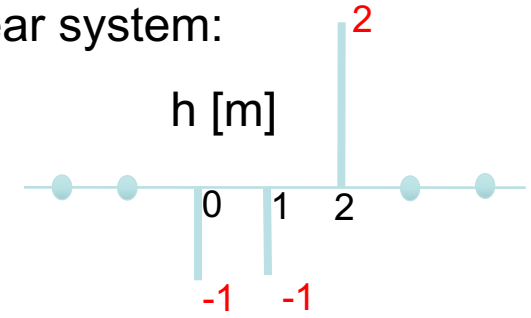
Notation for convolution operator

Convolution vs Cross correlation

- **Cross correlation:** The measurement of similarity between the two signals H and F.
- **Convolution:** The mesaurement of the effect of a signal (H) on the other signal (F).
- When the filter (H) is symmetric, the results are identical.

1-D linear filtering

Linear system:



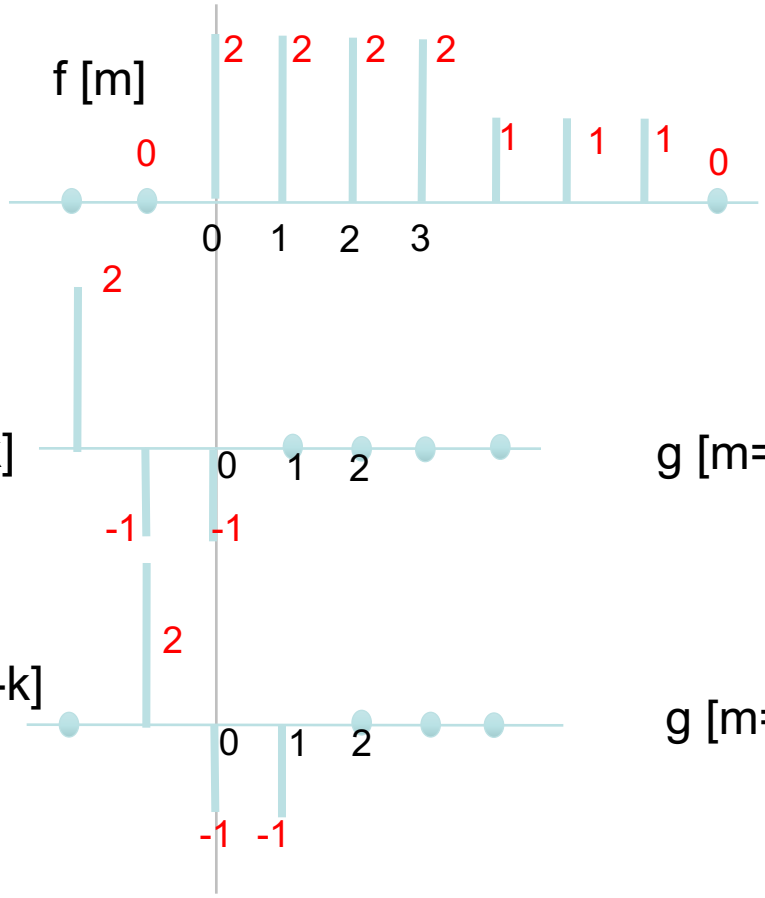
Output?

$$g(m=0) = \sum_k h(-k) f(k)$$

$$g(m=1) = \sum_k h(1-k) f(k)$$

$$g(m=2) = \sum_k h(2-k) f(k)$$

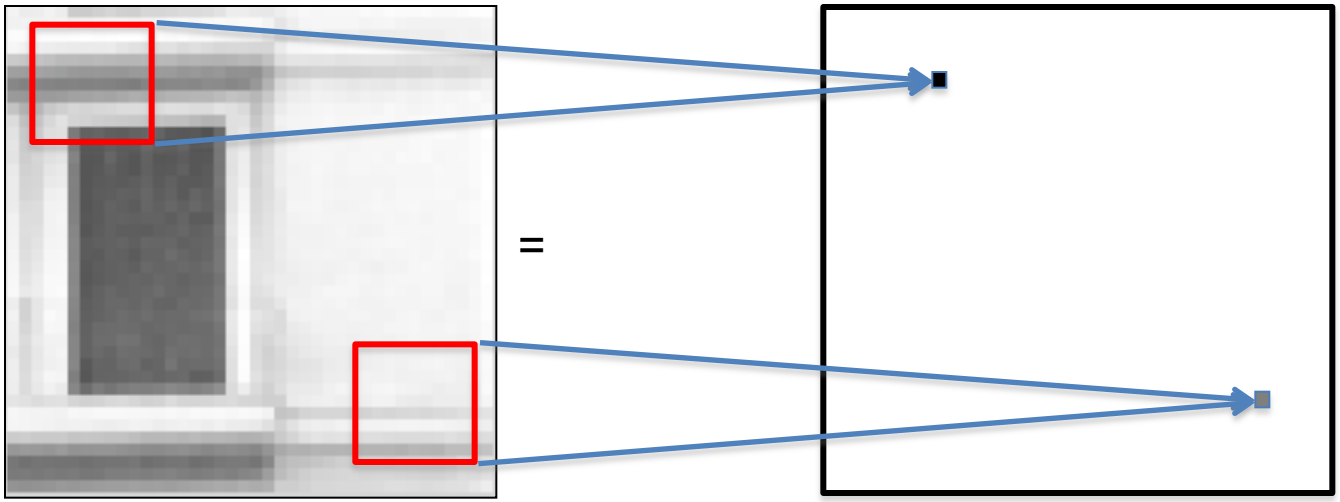
Input:



Discrete linear system

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

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



Matlab's Filter vs. Convolution

- 2D filtering

  h=filter g=image

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

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

- 2D convolution
  - `f=conv2(h,g);`

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

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.
- **Superposition:**
  - h * (f1 + f2) = (h * f1) + (h * f2)

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:

  unit impulse e = […, 0, 0, 1, 0, 0, …]. $f * e = f$

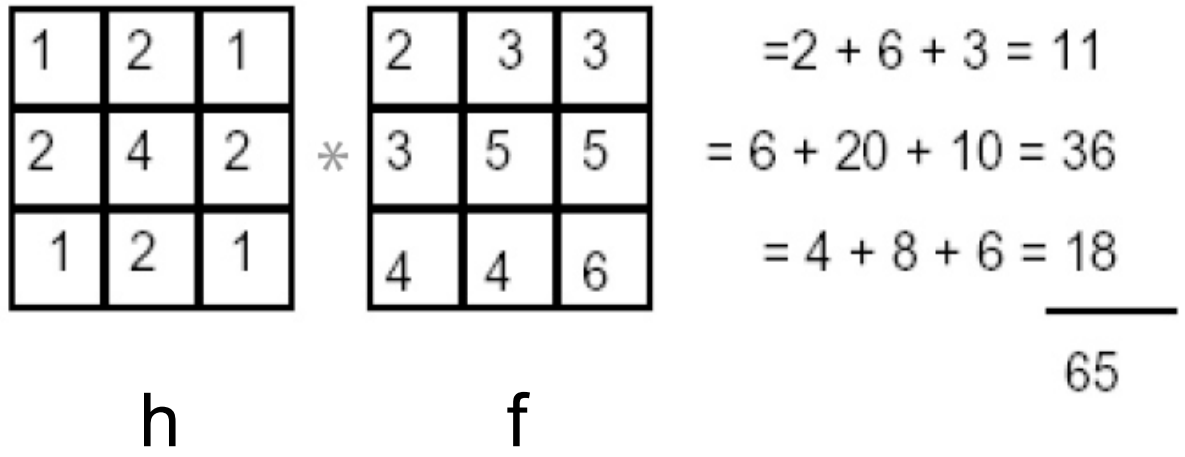
Separability

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

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

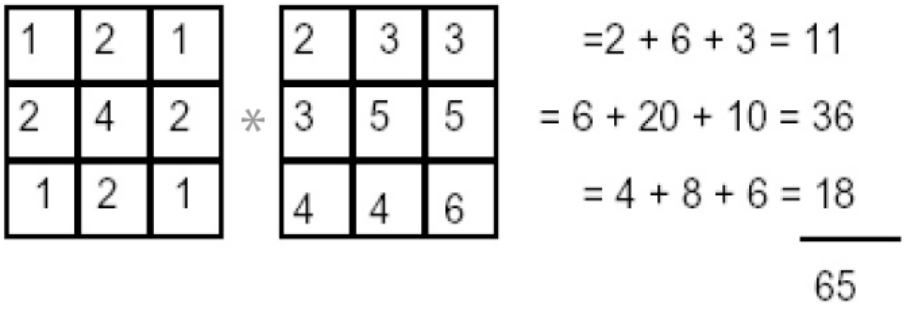
Separability example

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

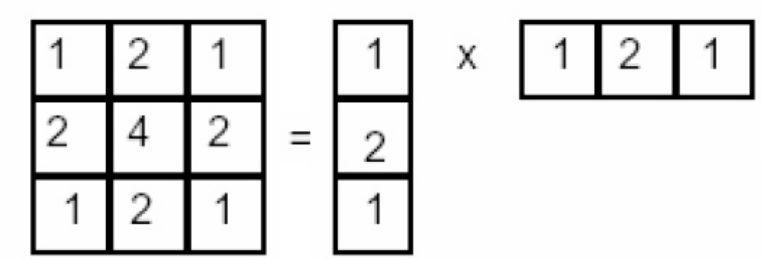


Source: K. Grauman

Separability example

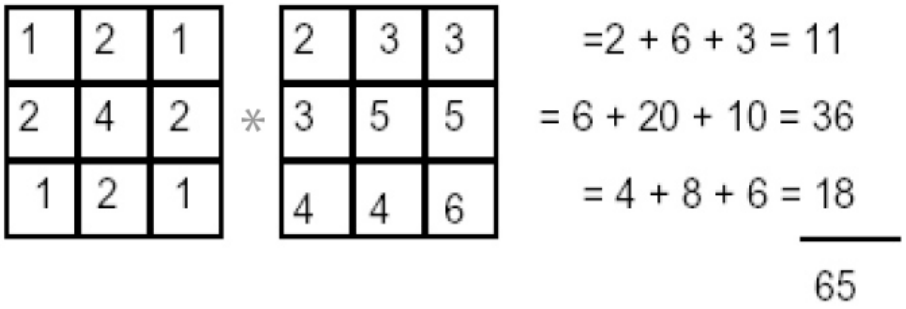


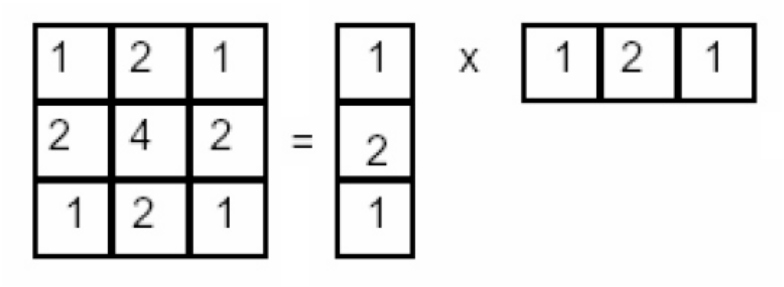
- The filter is factored into a product of 1D filters.



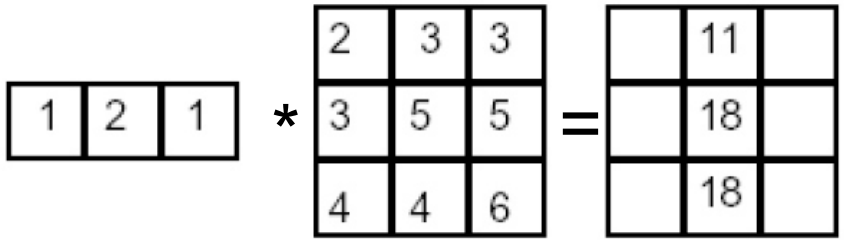
Source: K. Grauman

Separability example



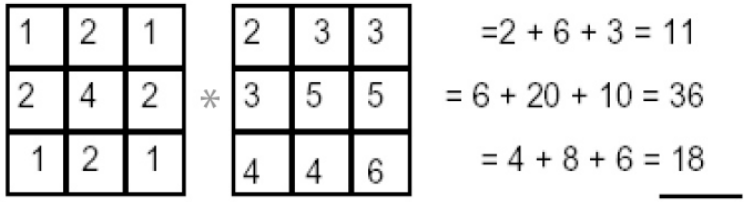


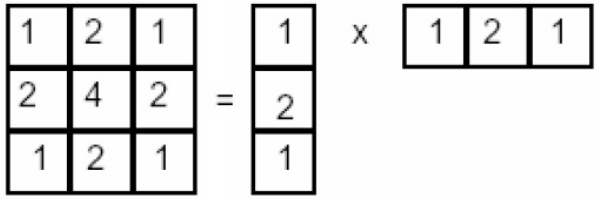
- Perform convolutions along rows.

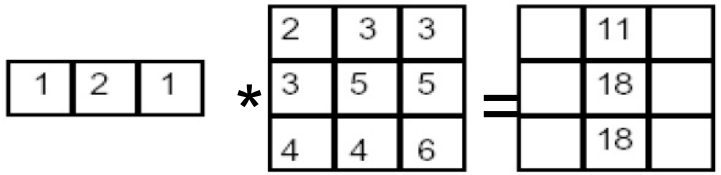


Source: K. Grauman

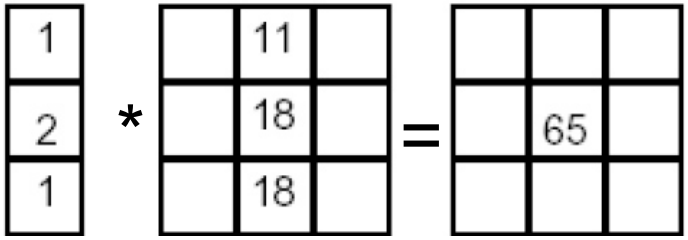
Separability example







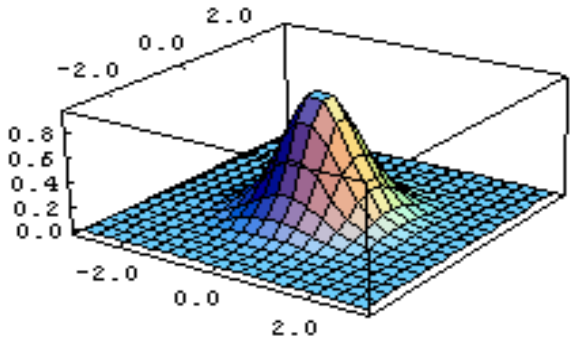
- Perform convolutions along columns.



Source: K. Grauman

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

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



Properties of Gaussian filter

- 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

Source: K. Grauman

Separability of the Gaussian filter

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

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

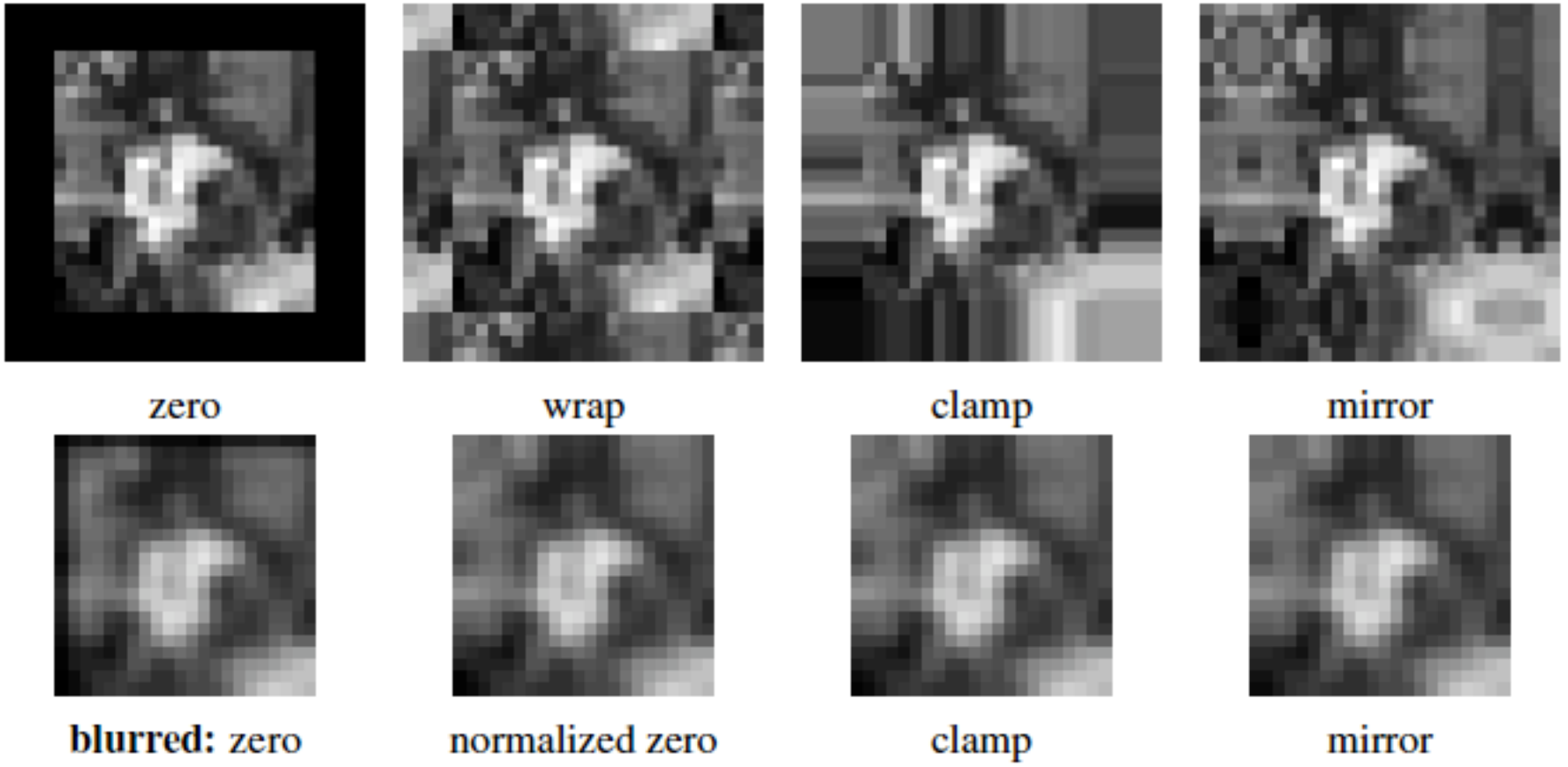
The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

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

Source: D. Lowe

- Separability means that a 2D convolution can be reduced to two 1D convolutions (one among rows and one among columns)
  - What is the complexity of filtering an n×n image with an m×m kernel?
    - $O(n^2 m^2)$
  - What if the kernel is separable?
    - $O(n^2 m)$

How to handle borders



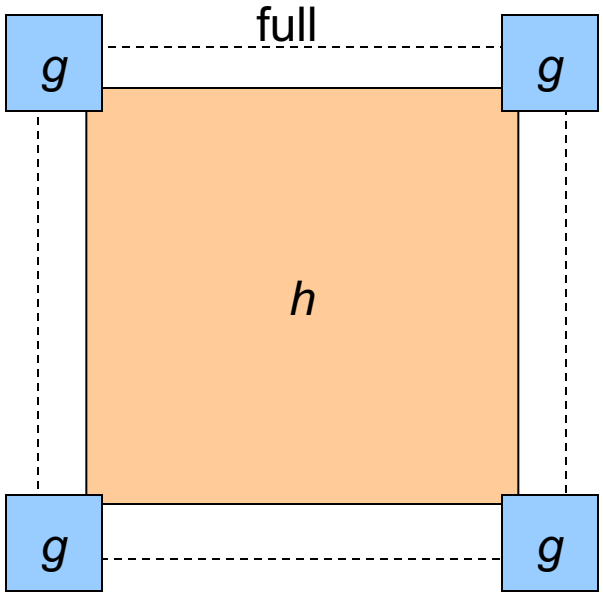
From Szeliski, Computer Vision, 2010

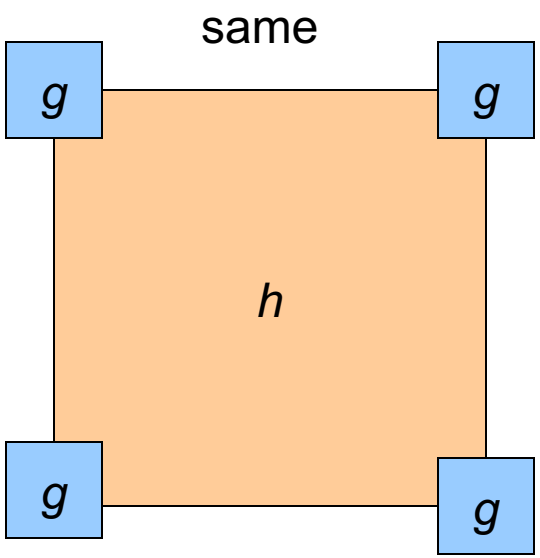
Example: Border handling

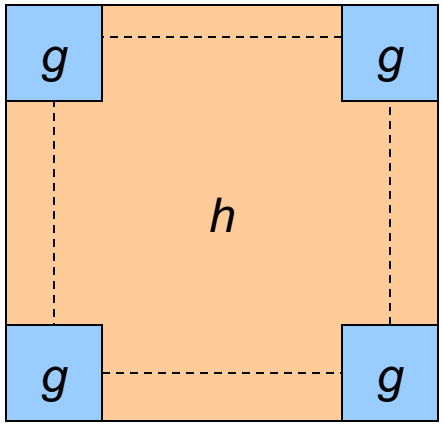
Source: S. Marschner

Matlab's border handling

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







Source: S. Lazebnik

Discrete Filtering

- Linear filter: Weighted sum of pixels over rectangular neighborhood—*kernel* defines weights

- Think of kernel as template being matched by correlation (Matlab: `imfilter, filter2`)

- Convolution: Correlation with kernel rotated 180° (Matlab: `conv2`)

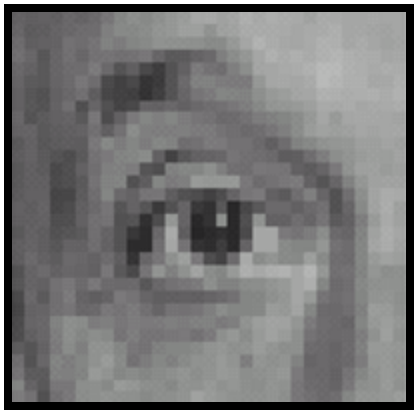
1	-1	-1
1	2	-1
1	1	1

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



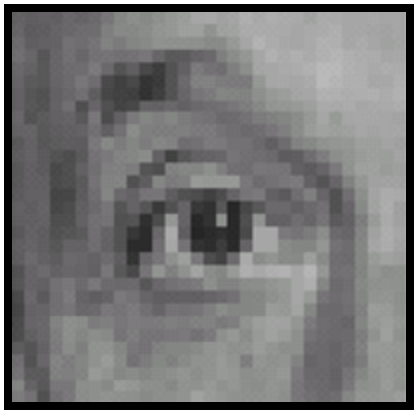
Original

0	0	0
0	1	0
0	0	0

?

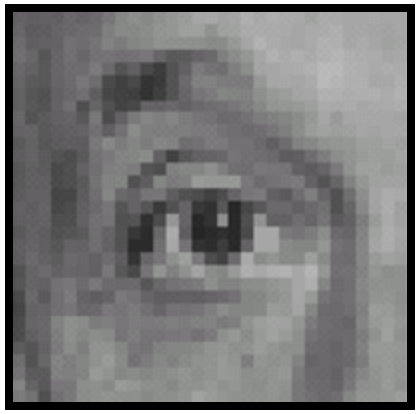
Source: D. Lowe

Practice with linear filters



Original

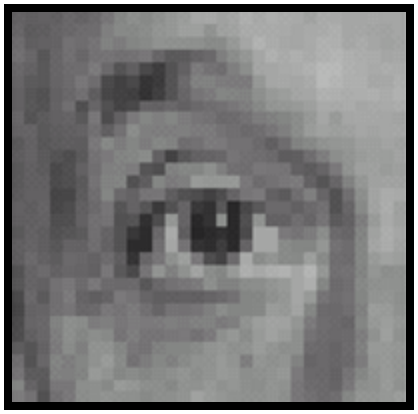
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Source: D. Lowe

Practice with linear filters



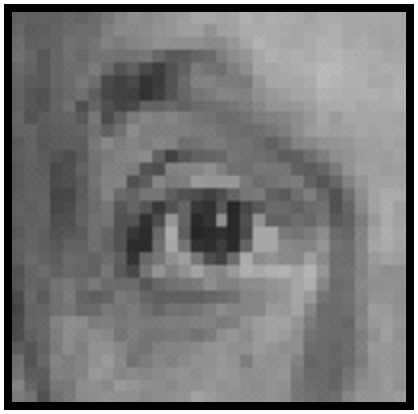
Original

0	0	0
0	0	1
0	0	0

?

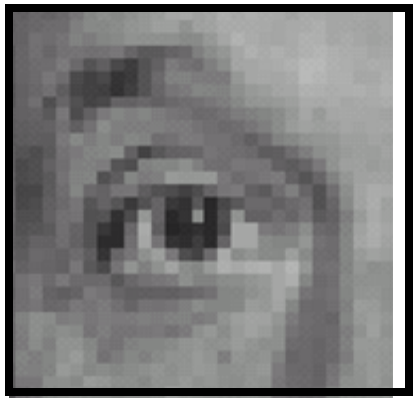
Source: D. Lowe

Practice with linear filters



Original

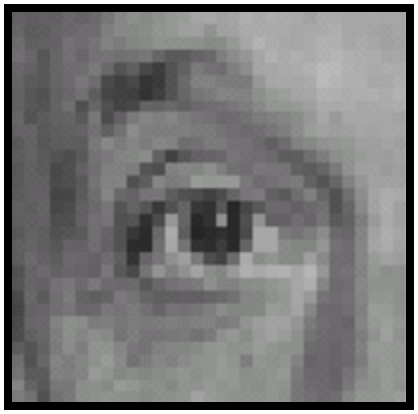
0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters



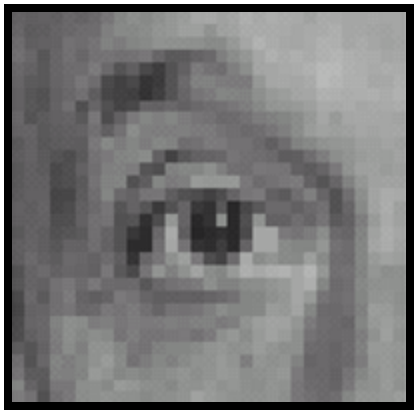
Original

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

(Note that filter sums to 1)

Source: D. Lowe

Practice with linear filters



Original

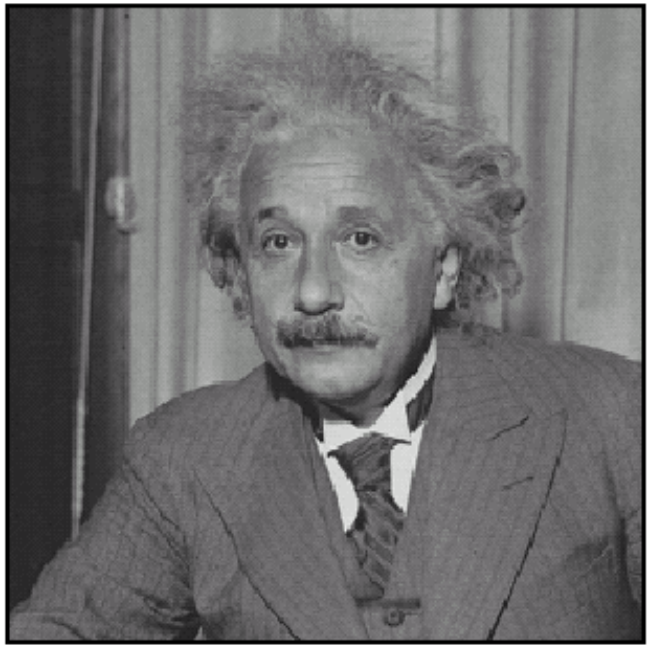
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

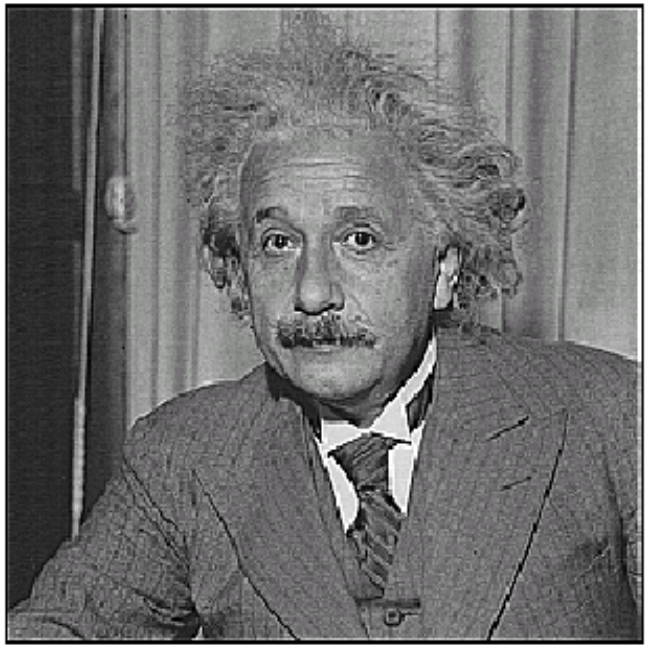
- Accentuates differences with local average

Source: D. Lowe

Sharpening



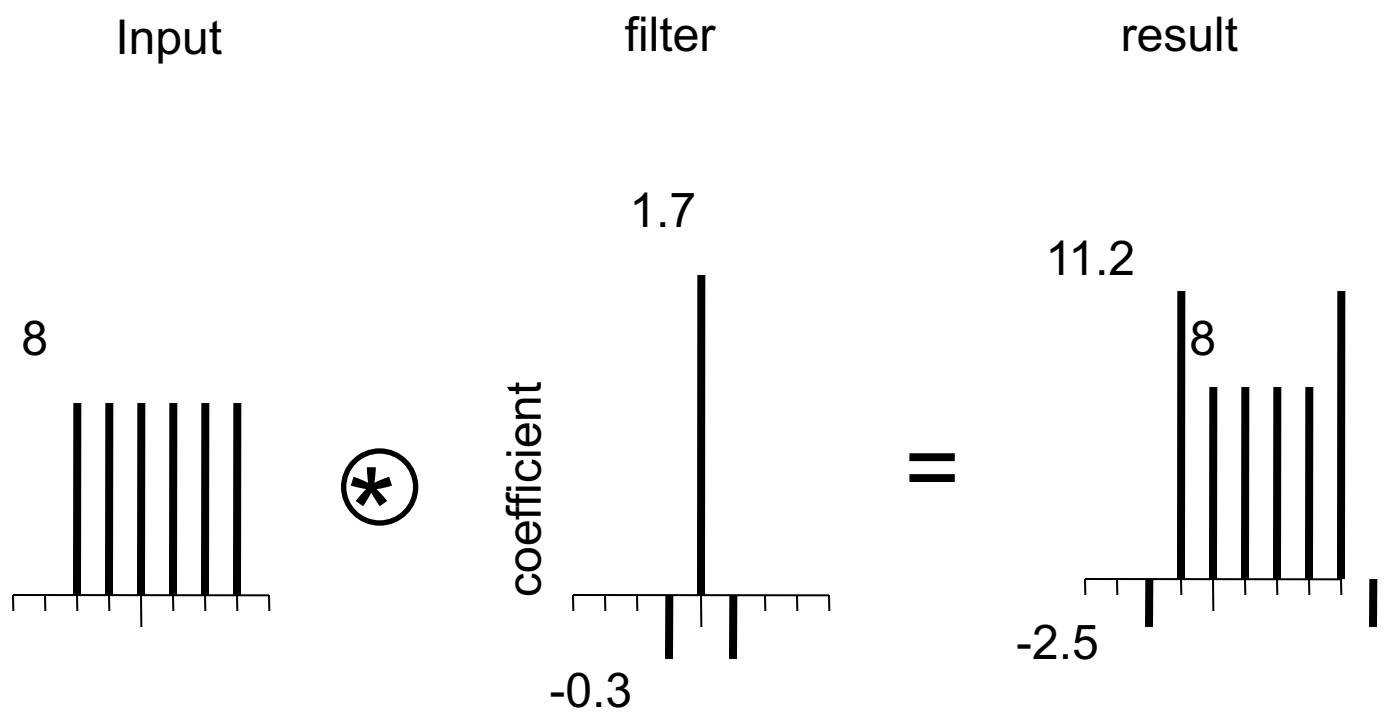
before



after

Source: D. Lowe

1-D sharpening example



[-1 1]

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

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

g[m,n]

f[m,n]

$[-1\ \ 1]^T$

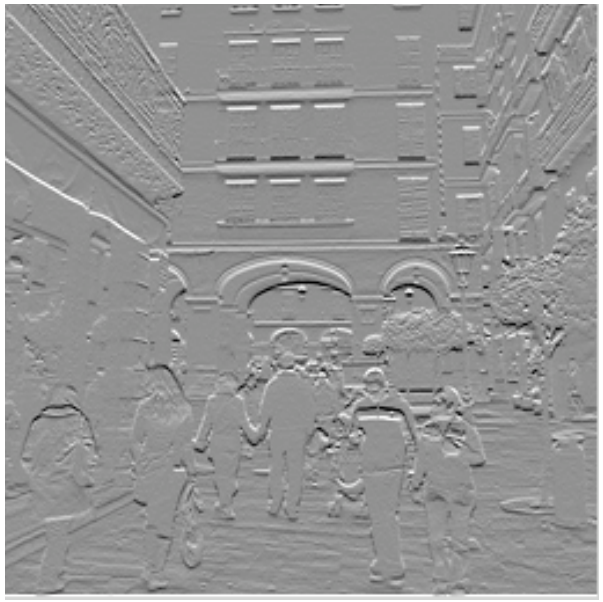
g[m,n]

$\otimes$

$[-1, 1]^T$

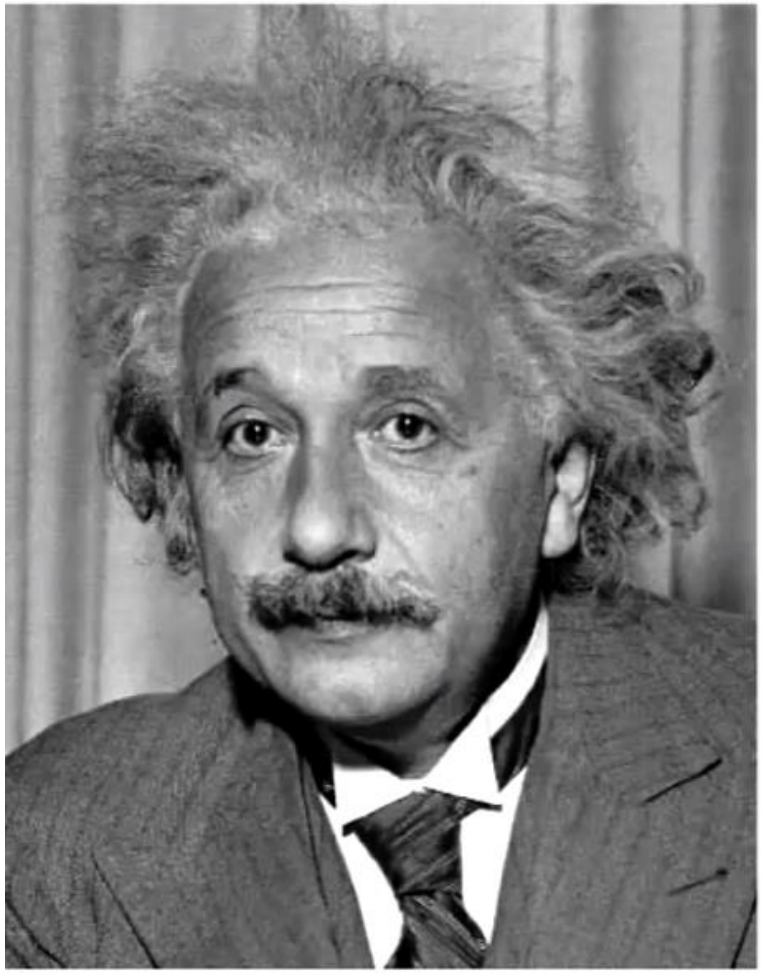
h[m,n]

=



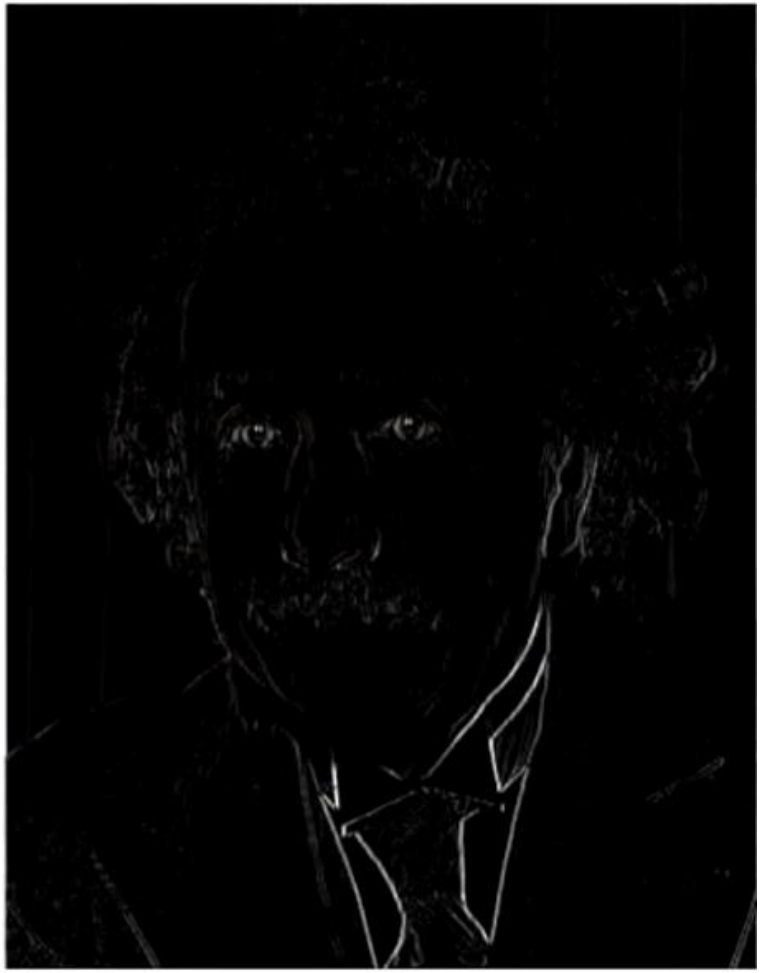
f[m,n]

Derivative filters



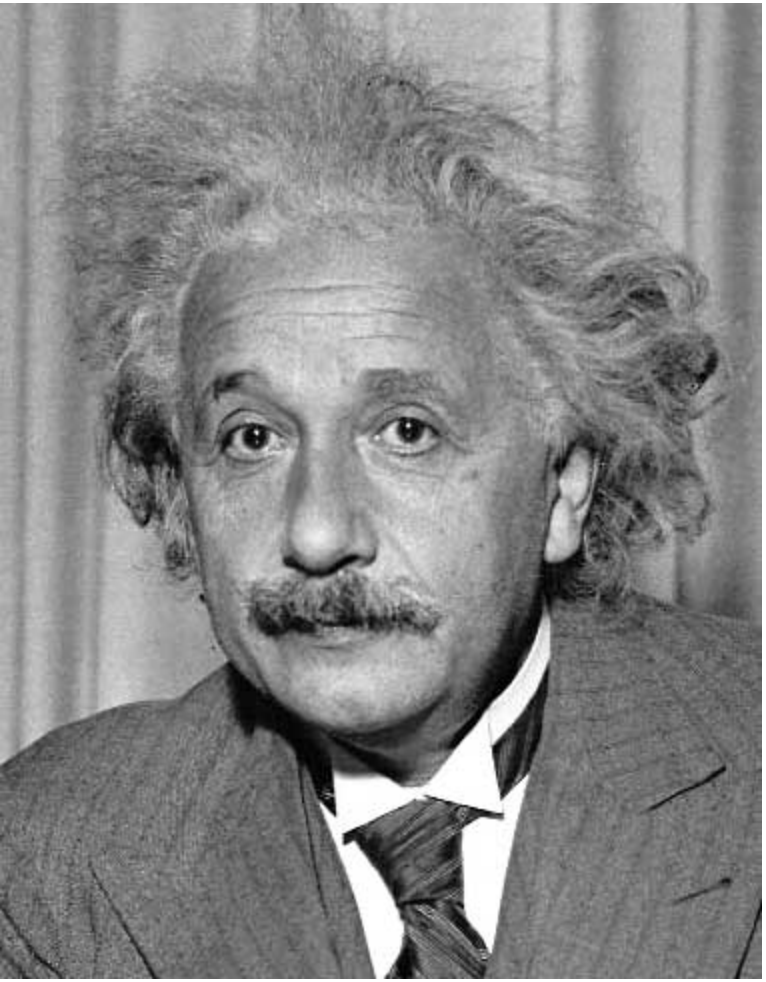
1	0	-1
2	0	-2
1	0	-1

Sobel



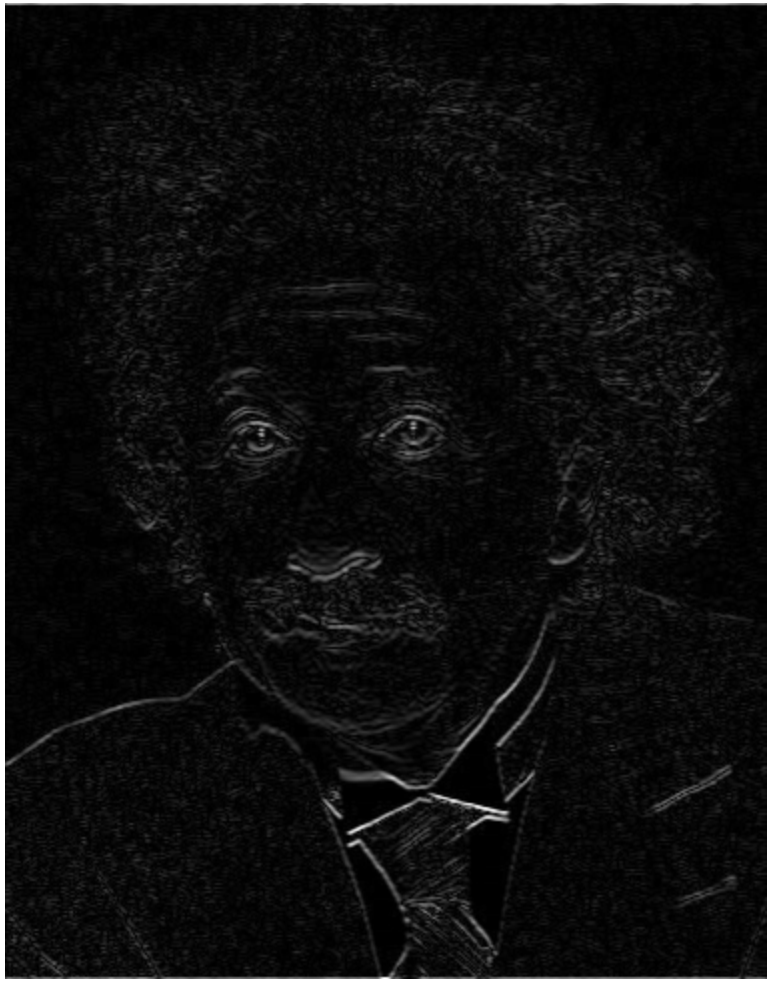
Vertical Edge
(absolute value)

Derivative filters



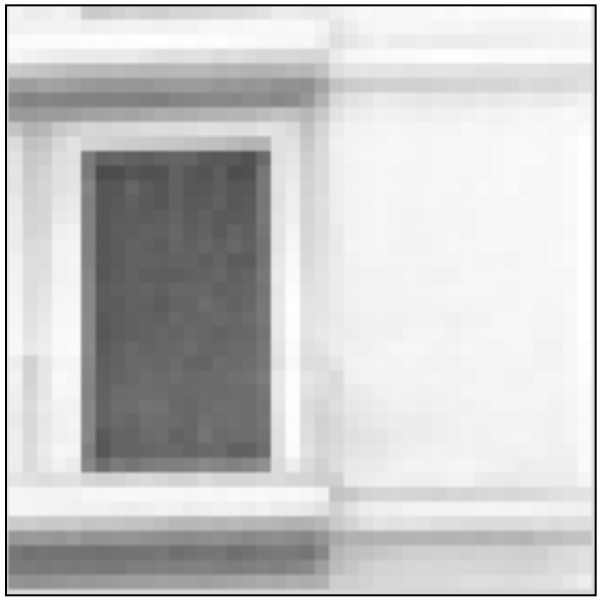
1	2	1
0	0	0
-1	-2	-1

Sobel



Horizontal Edge
(absolute value)

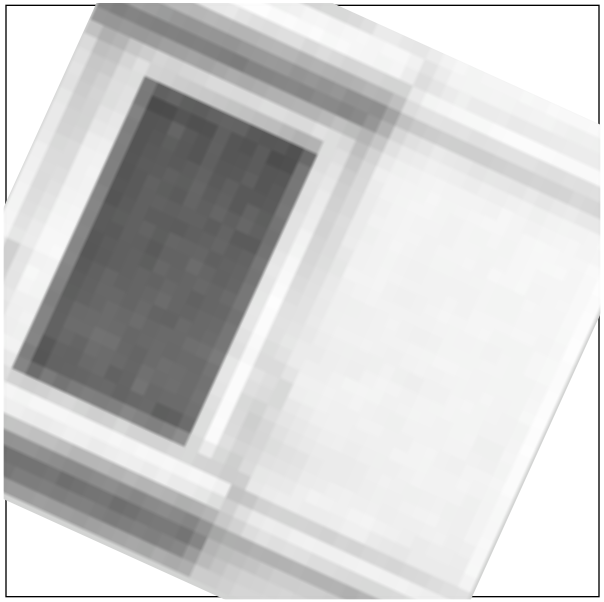
Image rotation



g[m,n]

$\otimes$? =

h[m,n]

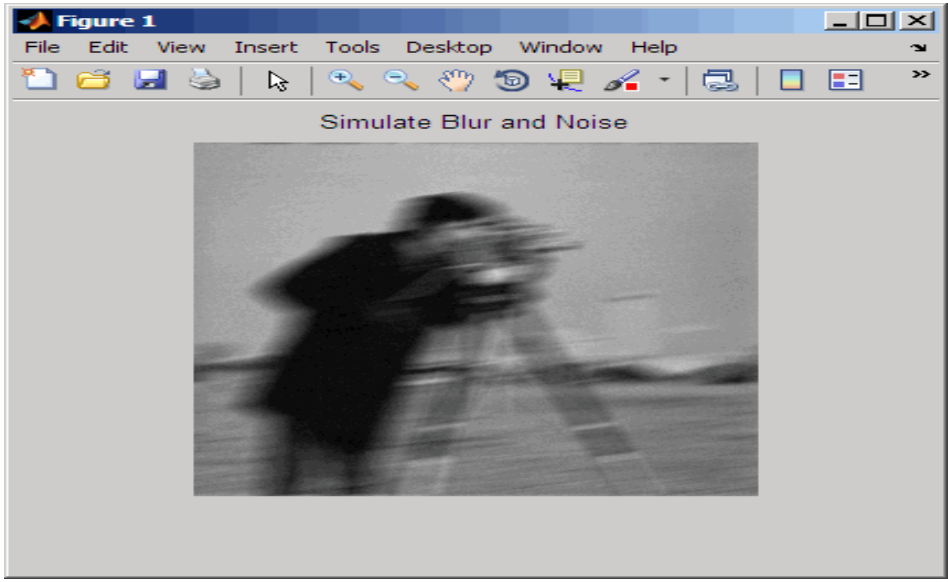


f[m,n]

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

How could we synthesize motion blur?

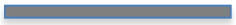
```
theta = 30; len = 20;
fil = imrotate(ones(1, len), theta, 'bilinear');
fil = fil / sum(fil(:));
figure(2), imshow(imfilter(im, fil));
```



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]

Three views of linear filtering

- Linear filters in the spatial domain
  - Filter is a mathematical operation on matrix
  - Smoothing, sharpening, measuring texture

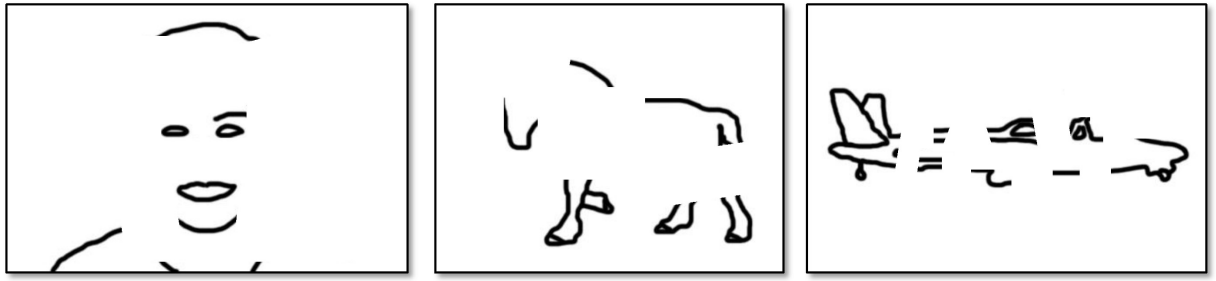
- Linear filters in the **frequency domain [Self-study]**
  - Filtering is a way to modify the frequencies of images
  - Denoising, sampling, image compression

- Linear filters as **templates matching**
  - Filtering is a way to match a template to the image
  - Detection, registration

Edge Detection

Edge detection

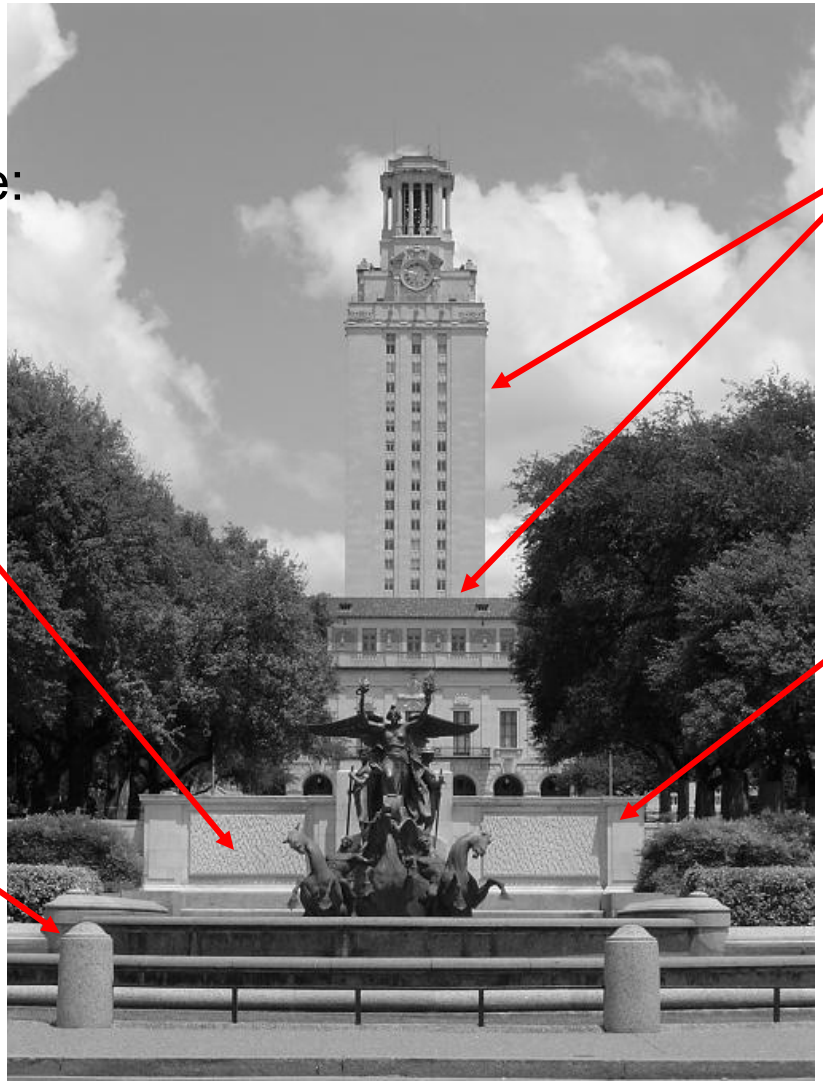
- **Goal**: map image from 2d array of pixels to a set of curves or line segments or contours.
- **Why?**



•**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

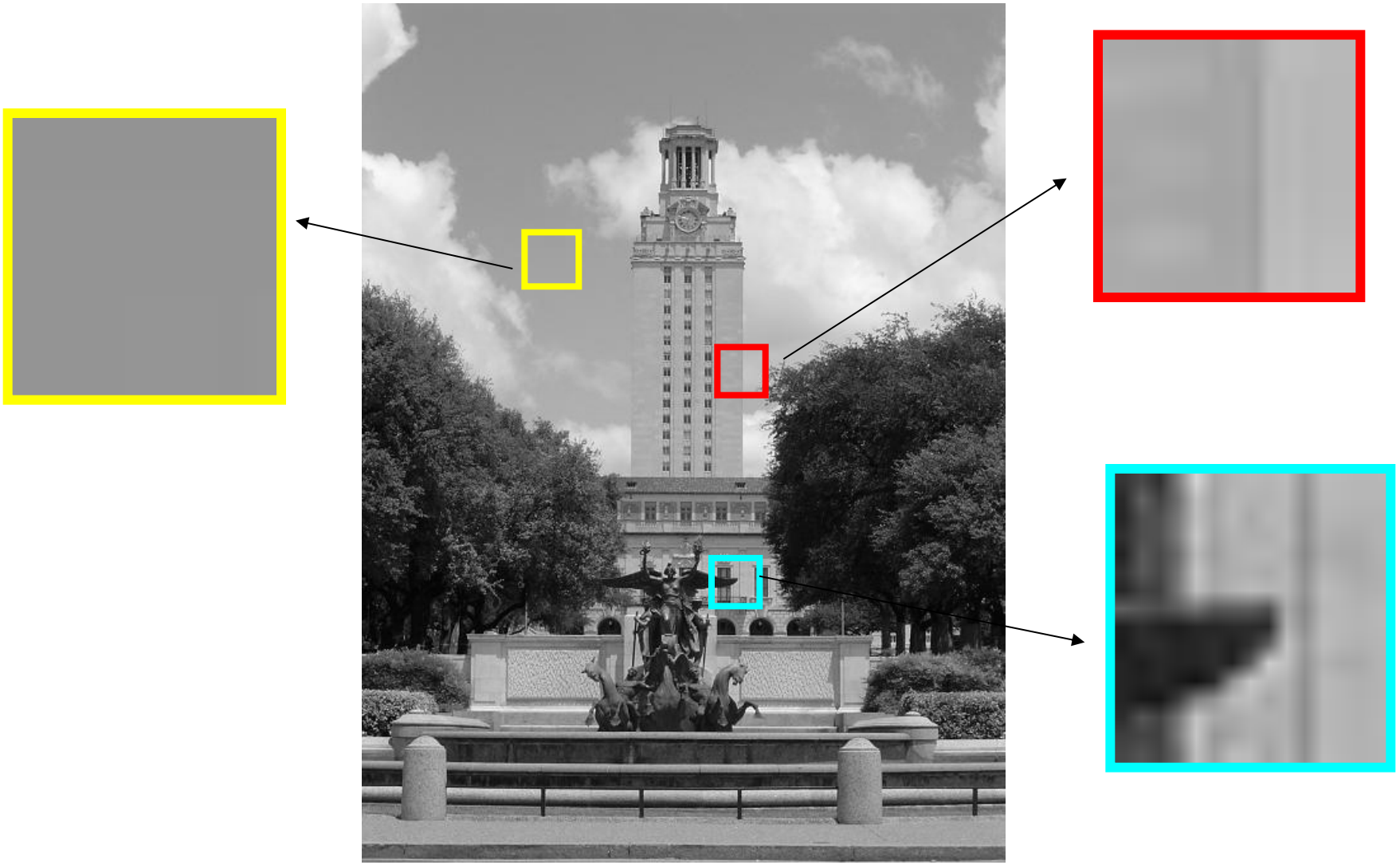
•Depth discontinuity: object boundary

•Cast shadows

•Change in surface orientation: shape

•Slide credit: Kristen Grauman

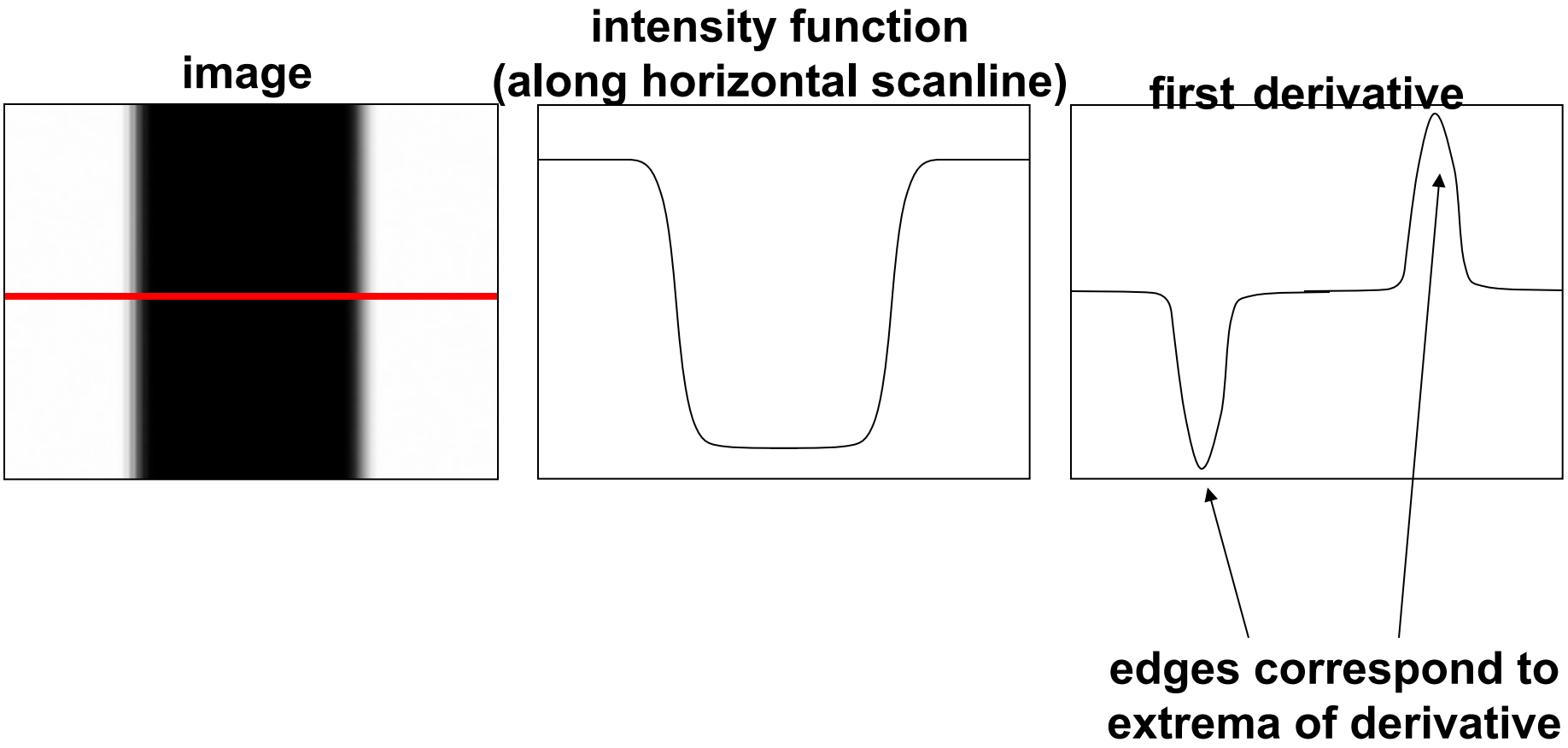
Edges/gradients and invariance



•Slide credit: Kristen Grauman

Derivatives and edges

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



•Source: L. Lazebnik

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 $\nabla^2 \mathbf{I}$
  - LOG/DOG (Laplace of Gaussian/Difference of Gaussians)

Common edge detectors

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

Derivatives with convolution

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

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

For discrete data, we can approximate using finite differences:

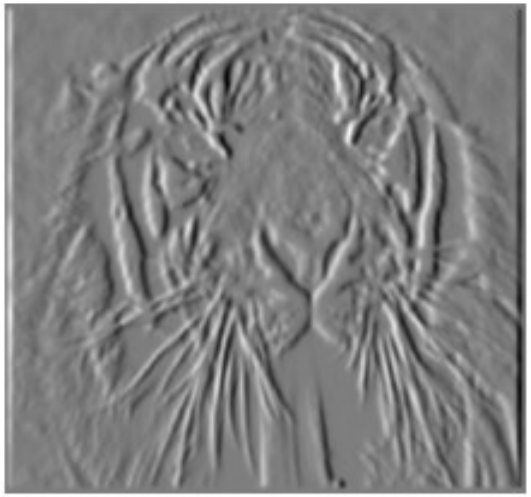
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

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

•Slide credit: Kristen Grauman

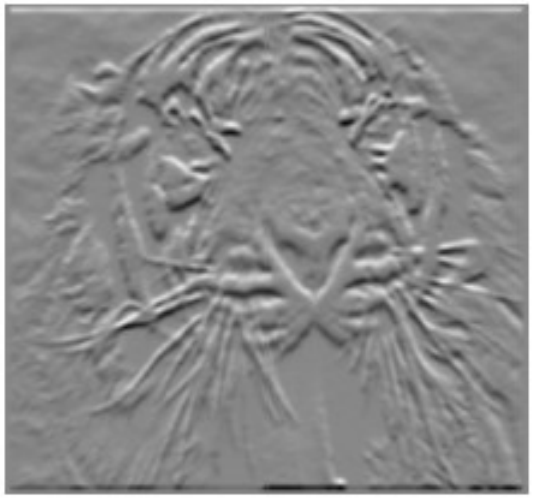
Partial derivatives of an image

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



-1	1

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



-1
1

or

1
-1

- Which shows changes with respect to x?

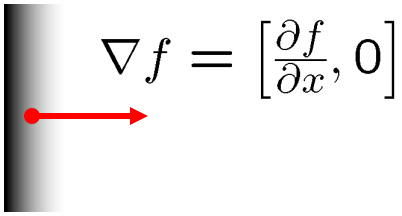
- (showing filters for correlation)

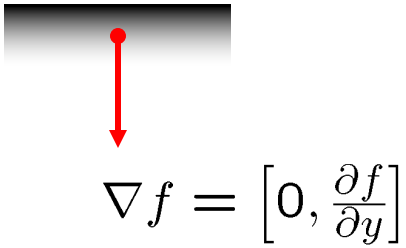
Image gradient

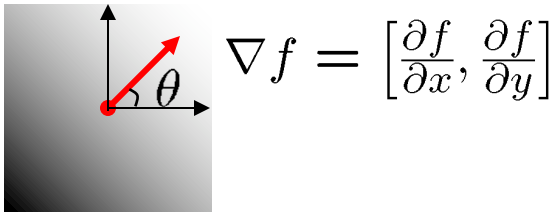
The gradient of an image:

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

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







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

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

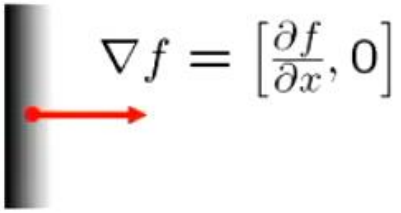
•Slide credit Steve Seitz

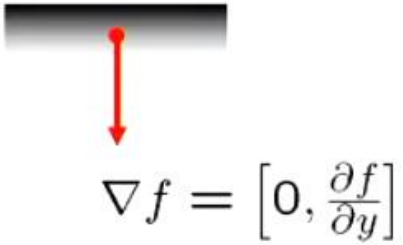
Image gradient

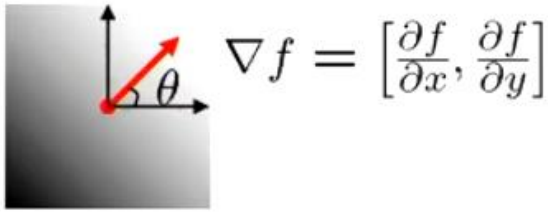
The gradient of an image:

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

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







- 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

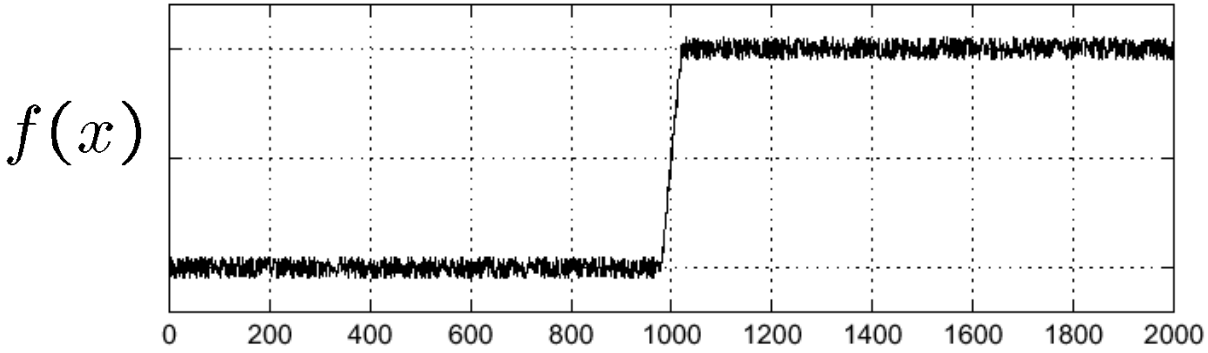
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

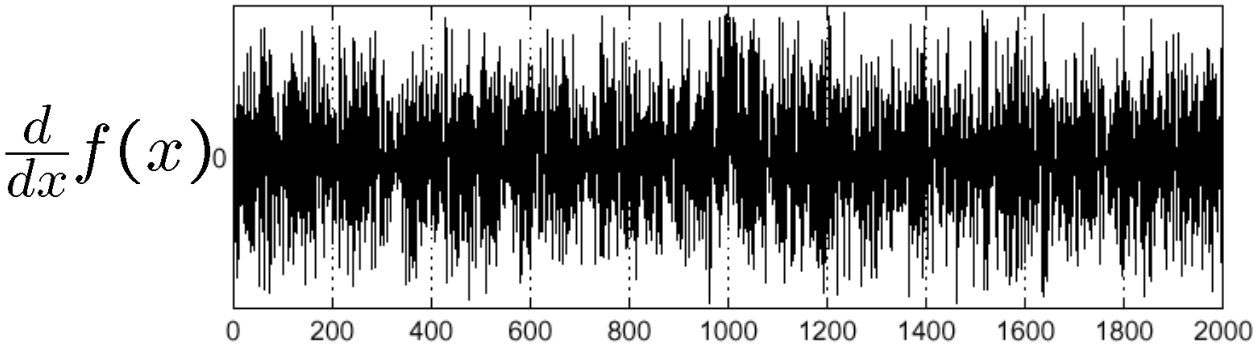
Slide credit: Steve Seitz

Effects of noise

Consider a single row or column of the image

– Plotting intensity as a function of position gives a signal





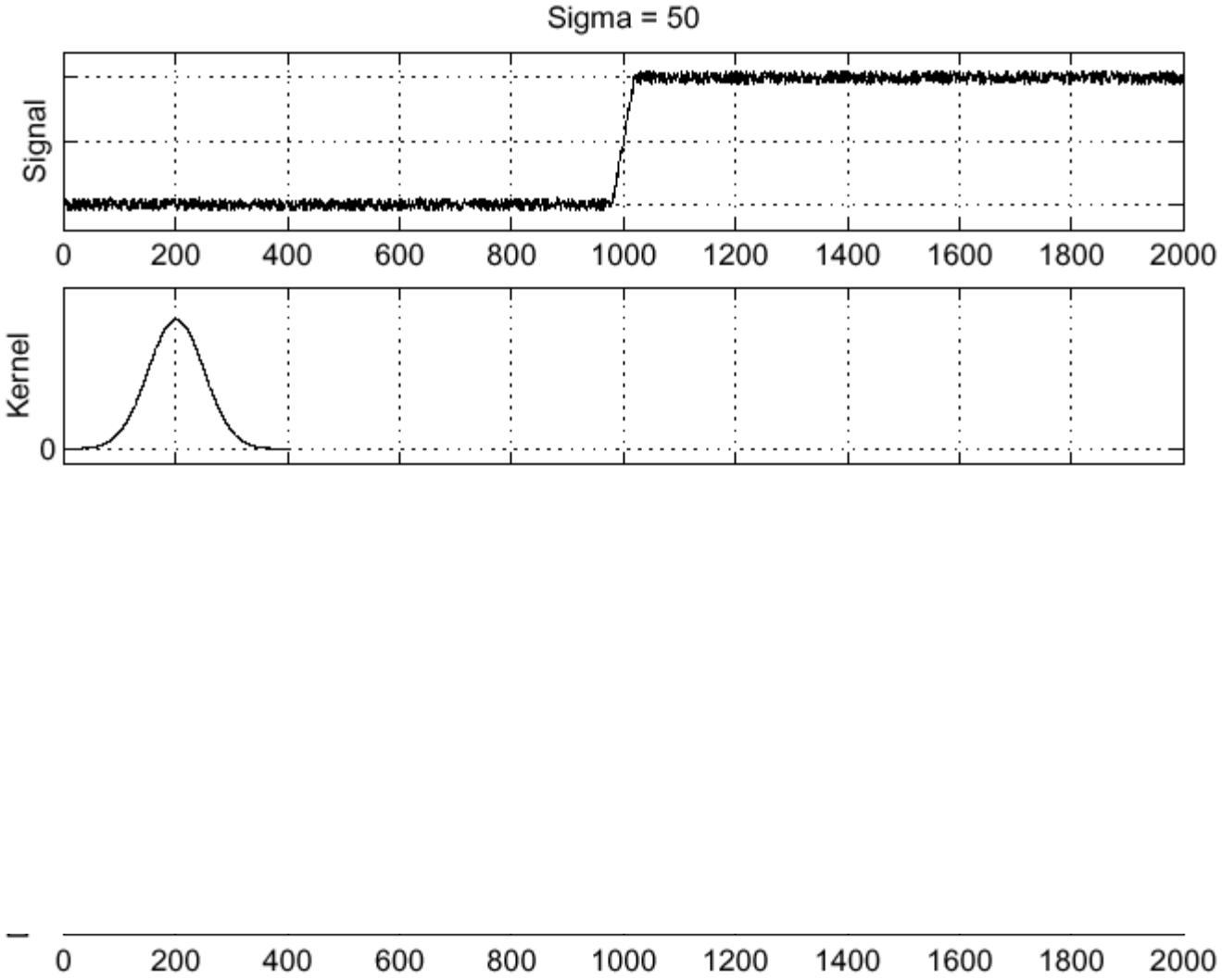
- Where is the edge?

•**Slide credit Steve Seitz**

Solution: smooth first

f

h



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

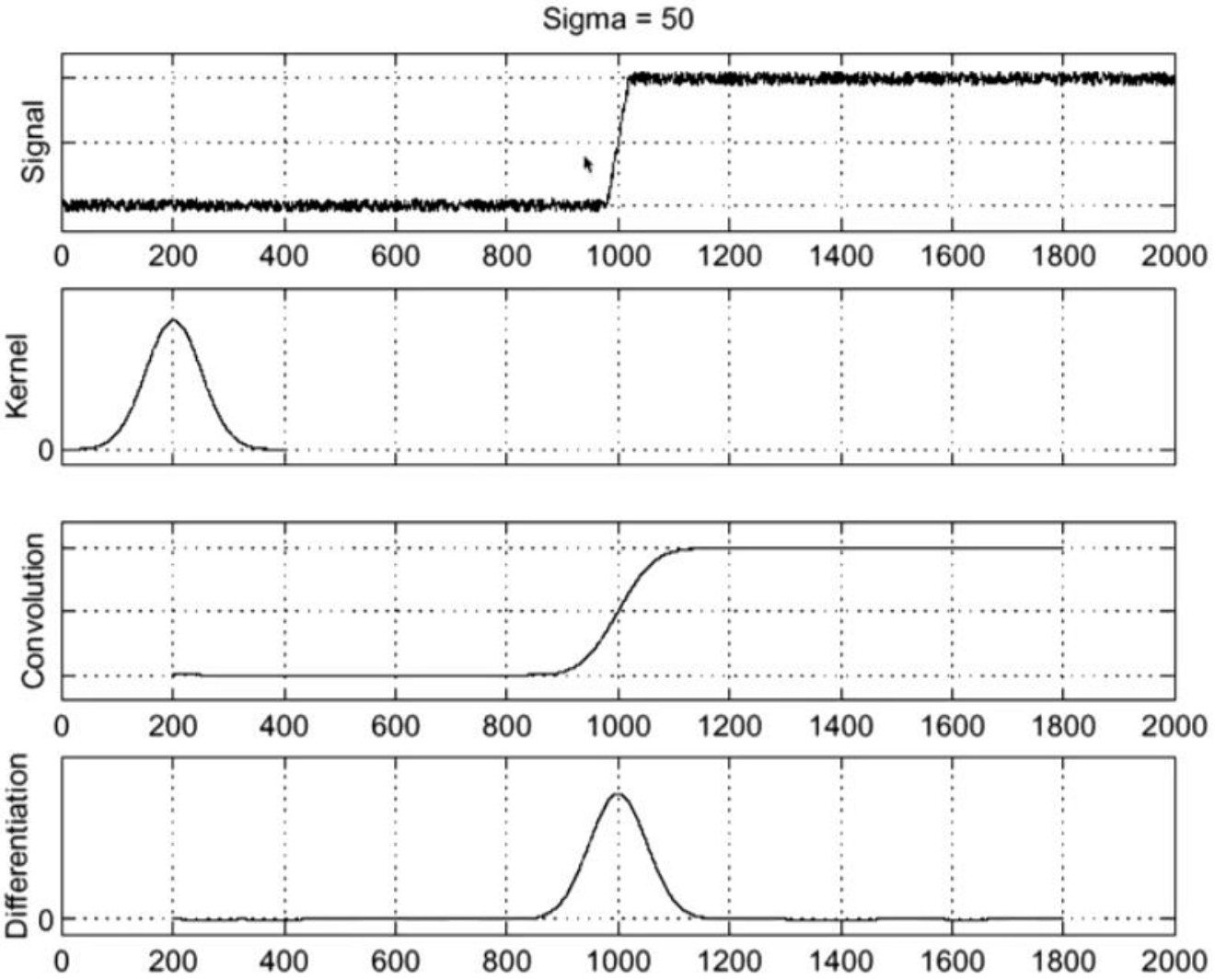
Solution: smooth first

f

h

$h \star f$

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



- Where is the edge?

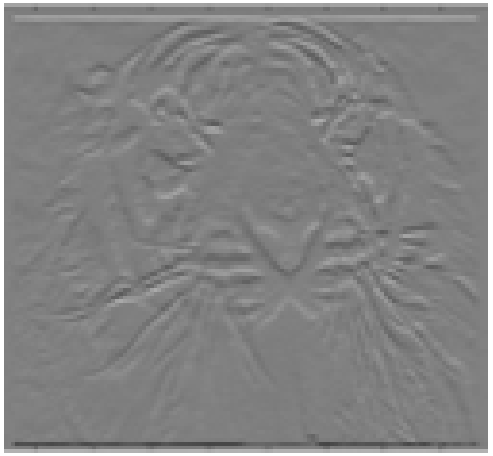
Gradient Edge Detectors

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

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```



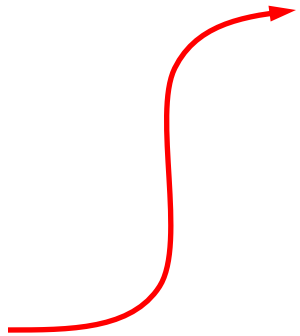
Eg: Convolute an image with a Sobel kernel

K

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

•Rotate

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



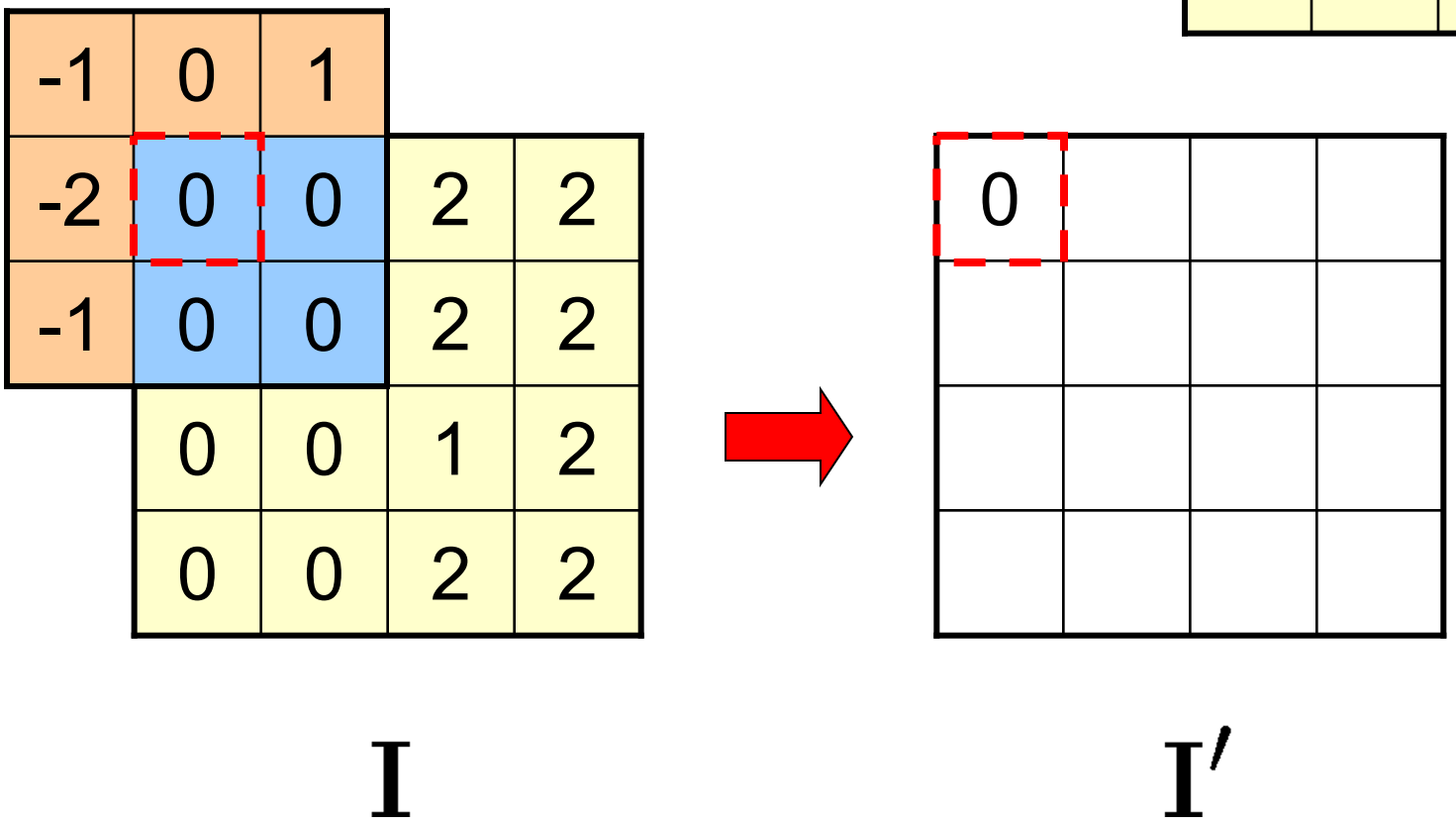
I

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

Step 1

-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



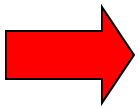
I I'

Step 2

-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

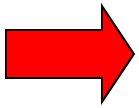
I'

Step 3

-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	

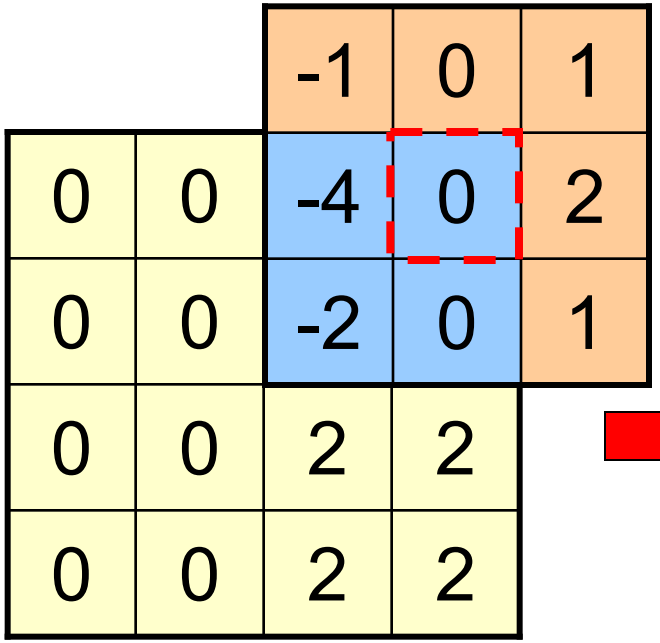
I

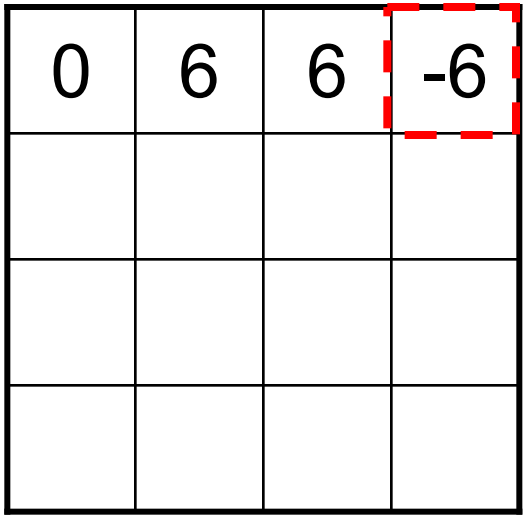
I'

Step 4

-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

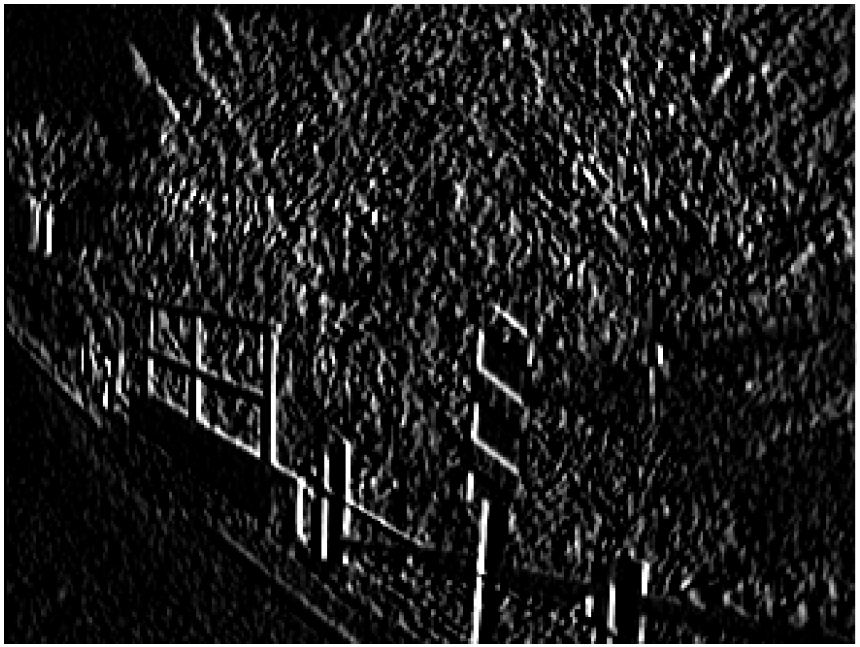


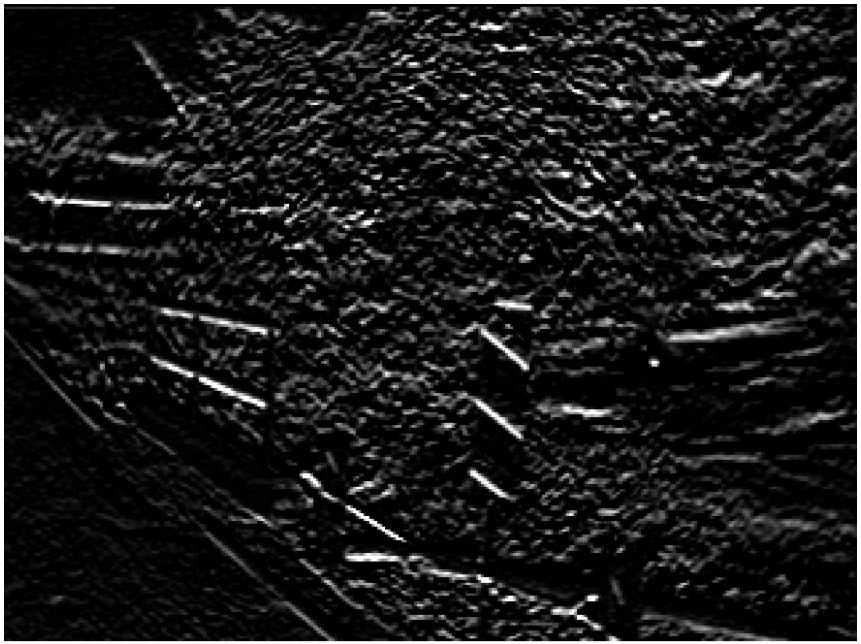


border artifect

I I'

Sobel edge detection: example





- Vertical edges
- Horizontal edges

Derivative theorem of convolution

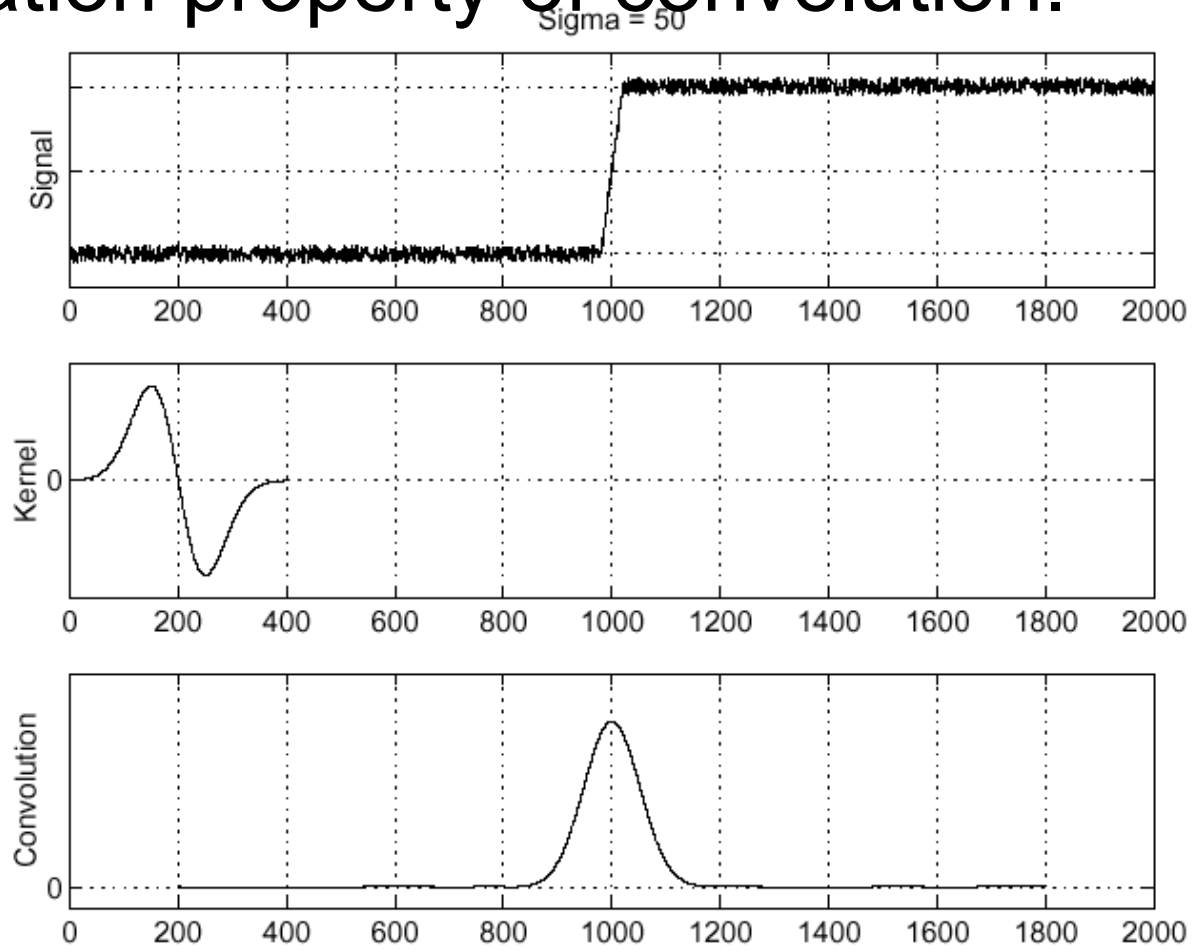
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Differentiation property of convolution.

f

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

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

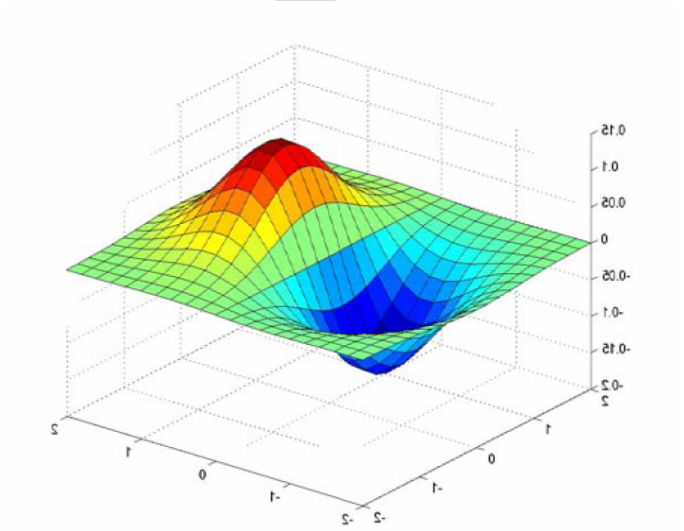


•**Slide credit Steve Seitz**

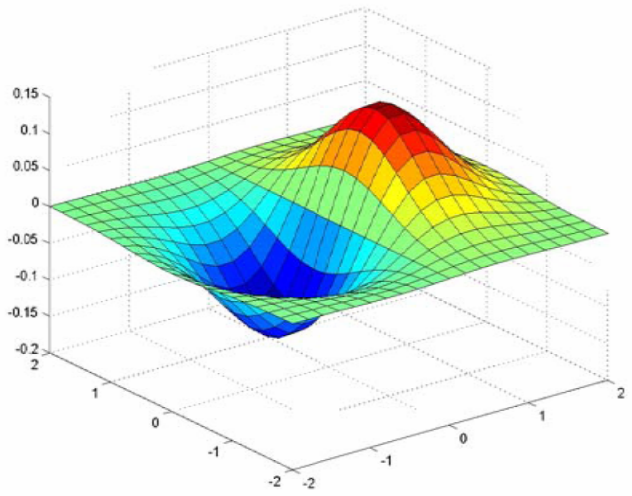
Derivative of Gaussian filters

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

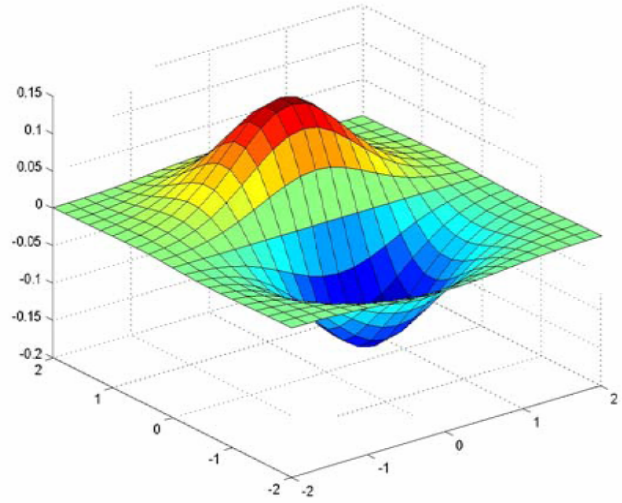
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \end{bmatrix}$$



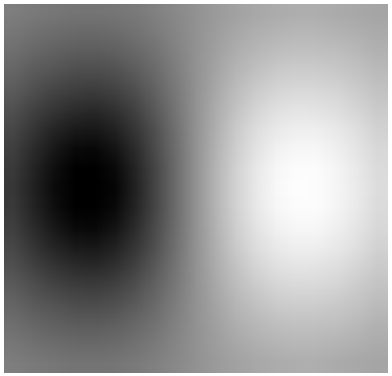
Derivative of Gaussian filters

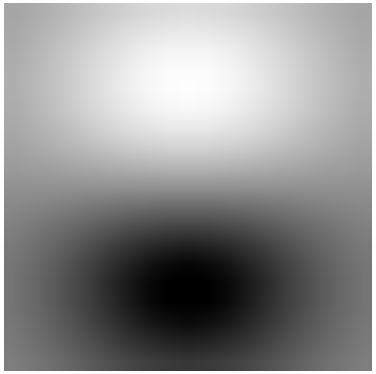


•***x*-direction**



•***y*-direction**





•**Source: L. Lazebnik**

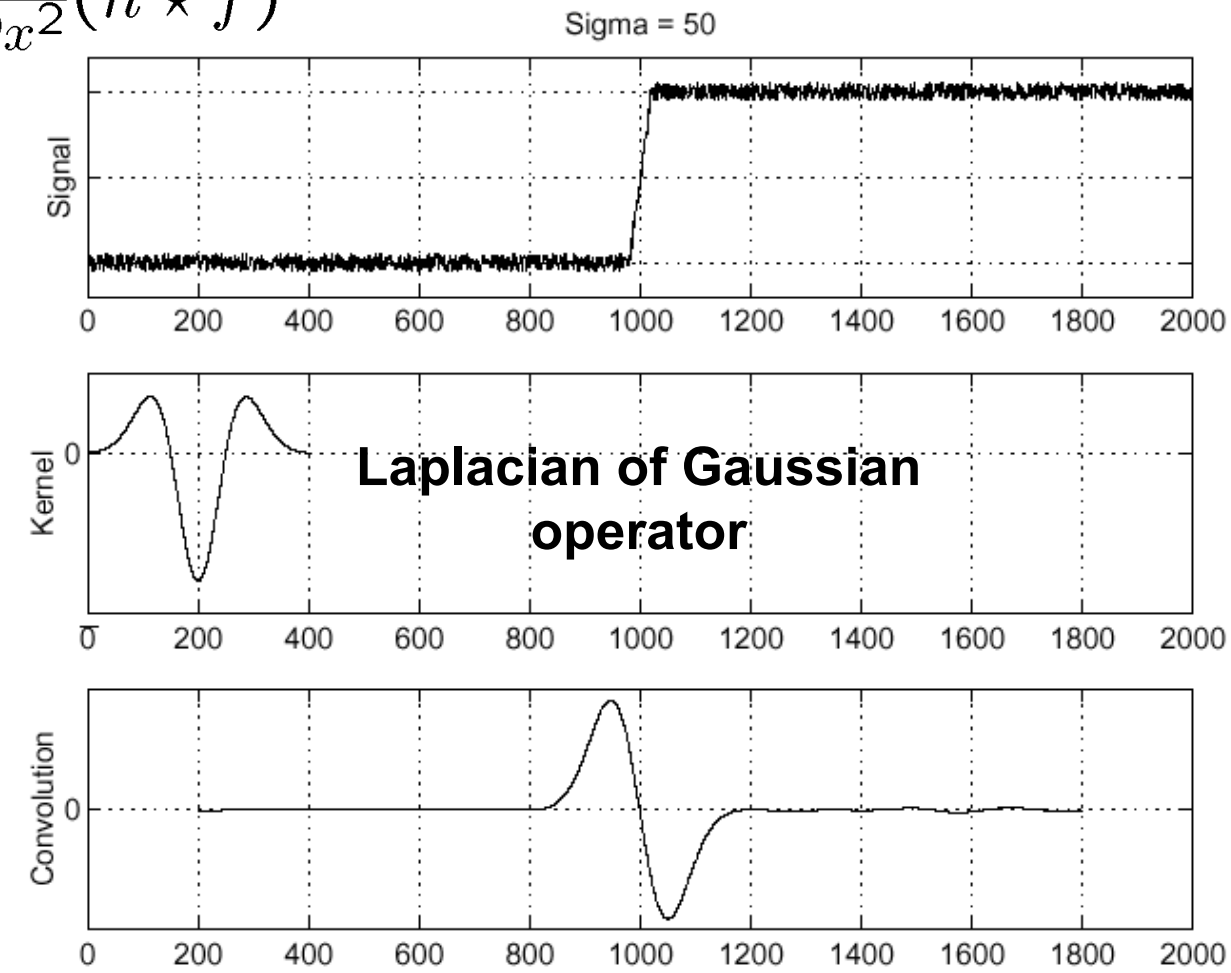
Laplacian of Gaussian

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

f

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

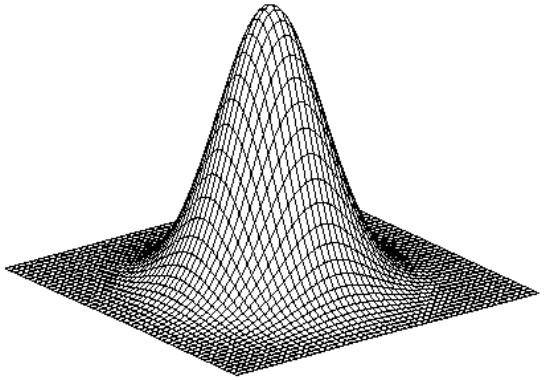
$(\frac{\partial^2}{\partial x^2} h) \star f$



- Where is the edge?
- Zero-crossings of bottom graph

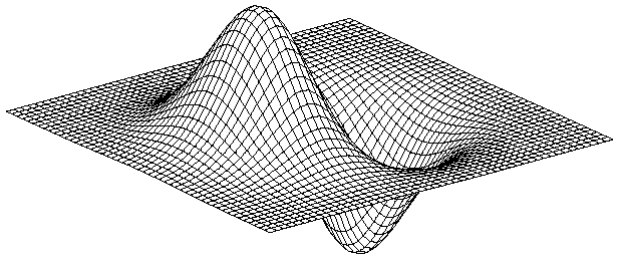
•Slide credit: Steve Seitz

2D edge detection filters



Gaussian

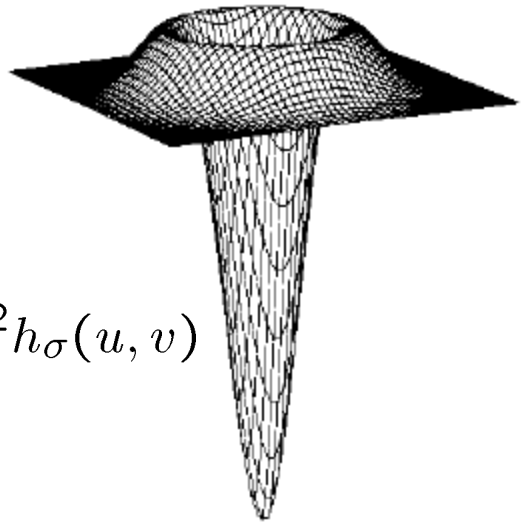
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



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

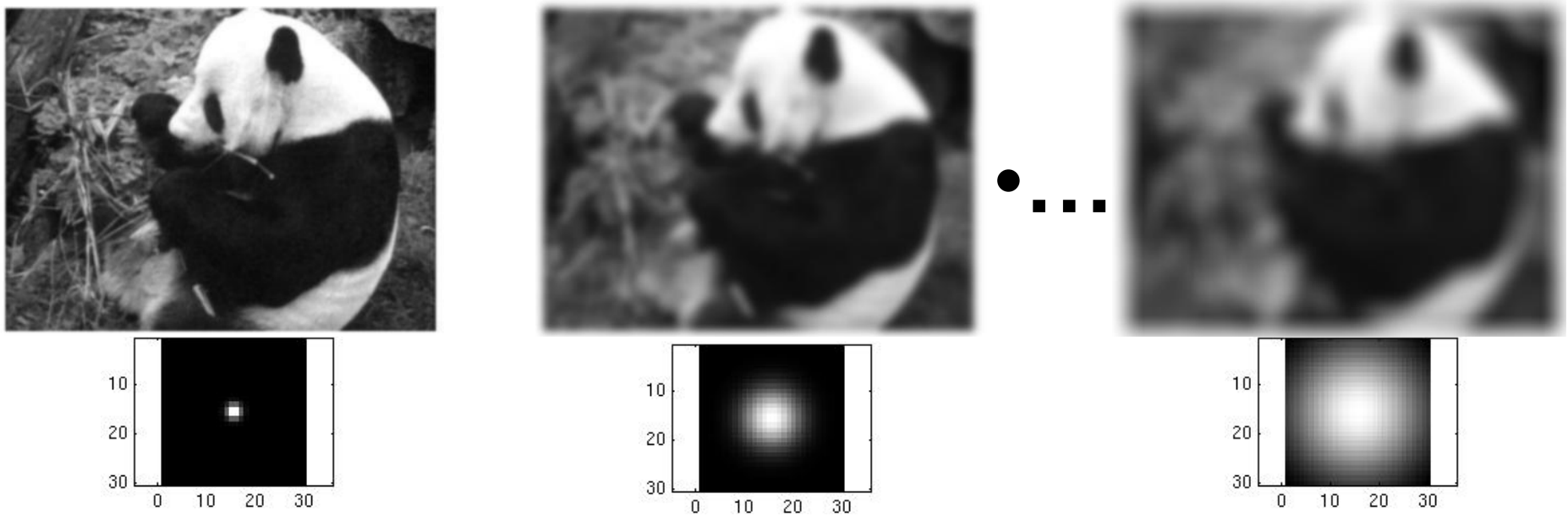
- ∇^2 is the Laplacian operator:

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

•Slide credit: Steve Seitz

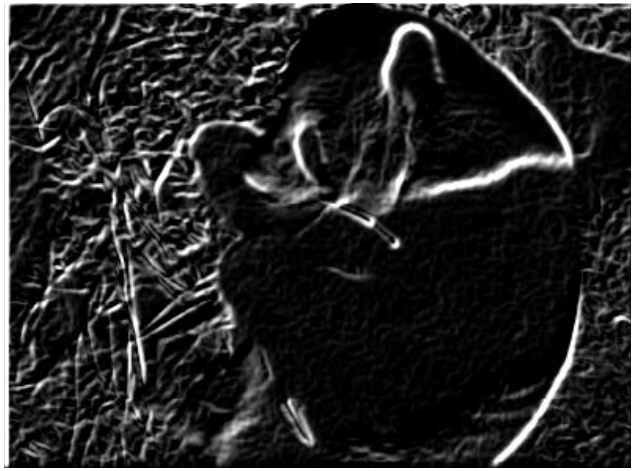
Smoothing with a Gaussian

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



•Slide credit: Kristen Grauman

Effect of σ on derivatives



σ = 1 pixel **σ = 3 pixels**

- The apparent structures differ depending on Gaussian's scale parameter.

- Larger values: larger scale edges detected
- Smaller values: finer features detected

- Slide credit: Kristen Grauman

Gradients -> edges

Primary edge detection steps:

1. Smoothing: suppress noise
2. Edge enhancement: filter for contrast
3. Edge localization

   Determine which local maxima from filter output are actually edges vs. noise

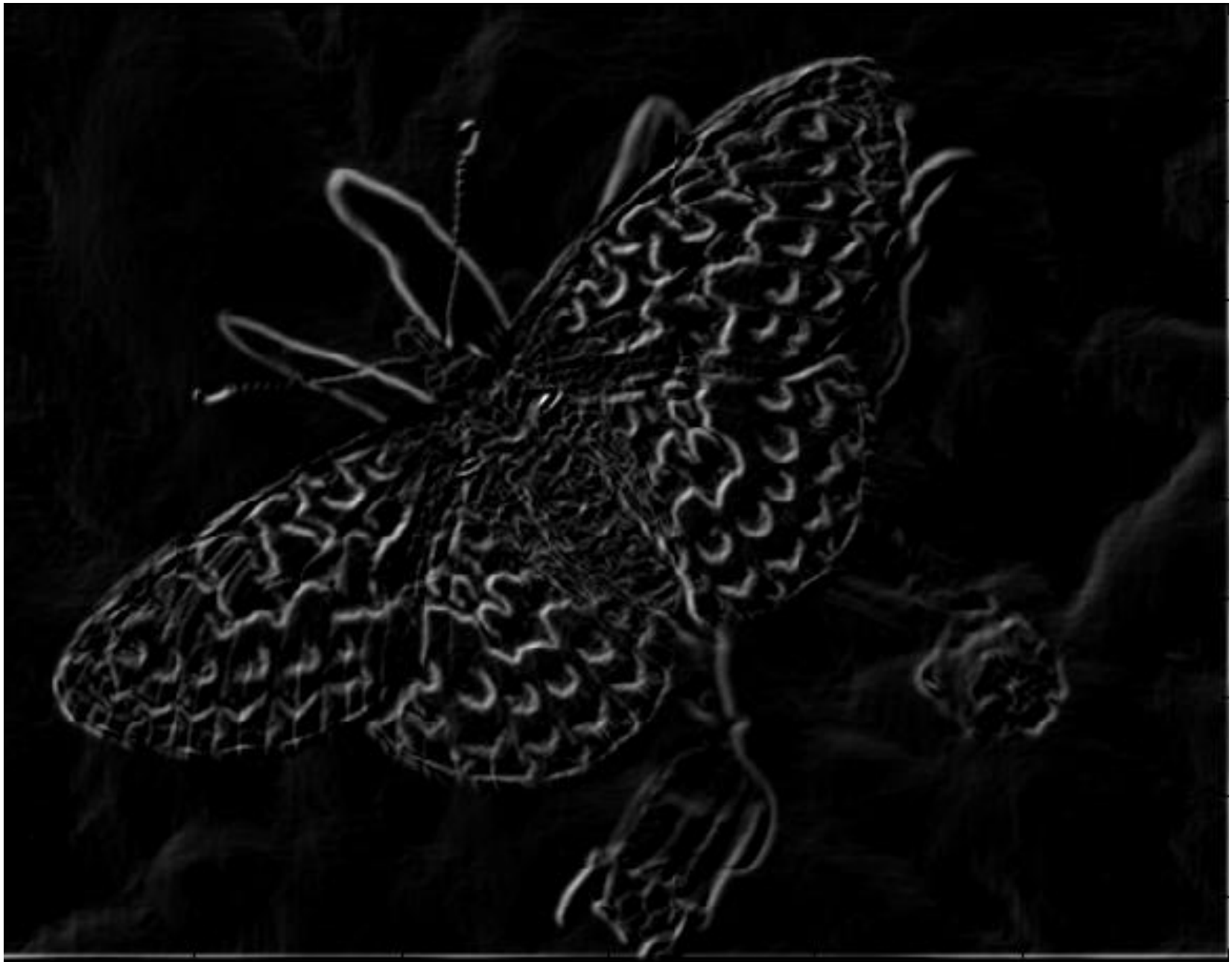
   - Threshold, Thin

Thresholding

- Choose a threshold value t
- Set any pixels less than t to zero (off)
- Set any pixels greater than or equal to t to one (on)

Original image

Gradient magnitude image



Thresholding gradient with a lower threshold

Thresholding gradient with a higher threshold

Canny edge detector

- Filter image with derivative of Gaussian
- Find magnitude and orientation of gradient
- **Non-maximum suppression**:
  - Thin wide “ridges” down to single pixel width
- **Linking and thresholding** (**hysteresis**):
  - Define two thresholds: low and high
  - Use the high threshold to start edge curves and the low threshold to continue them

- MATLAB: `edge(image, 'canny');`
- `>>help edge`

•Source: D. Lowe, L. Fei-Fei

The Canny edge detector

original image (Lena)

Slide credit: Steve Seitz

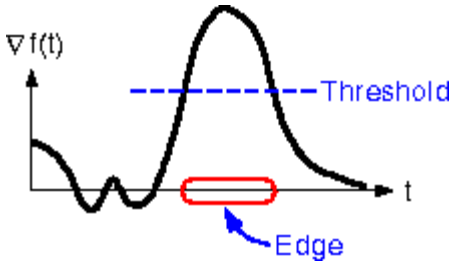
The Canny edge detector

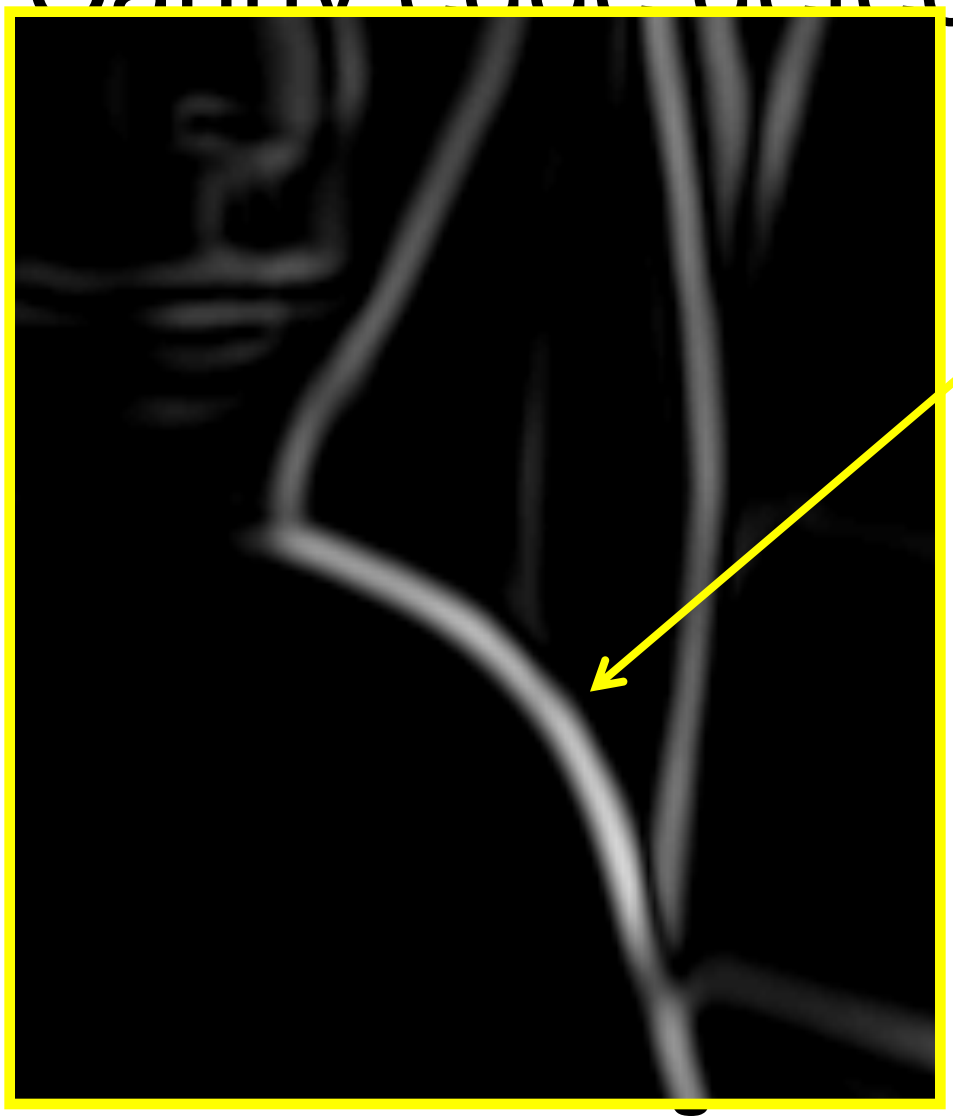
- norm of the gradient

The Canny edge detector

- thresholding

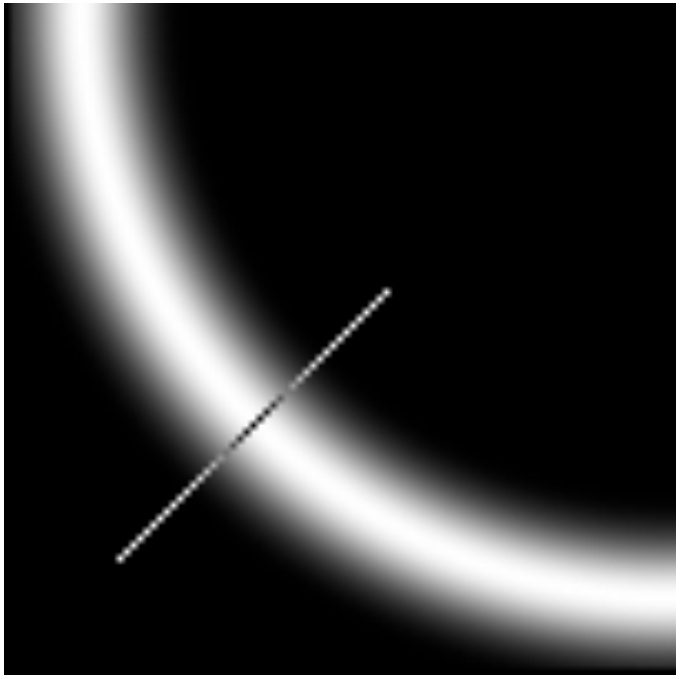
The Canny edge detector

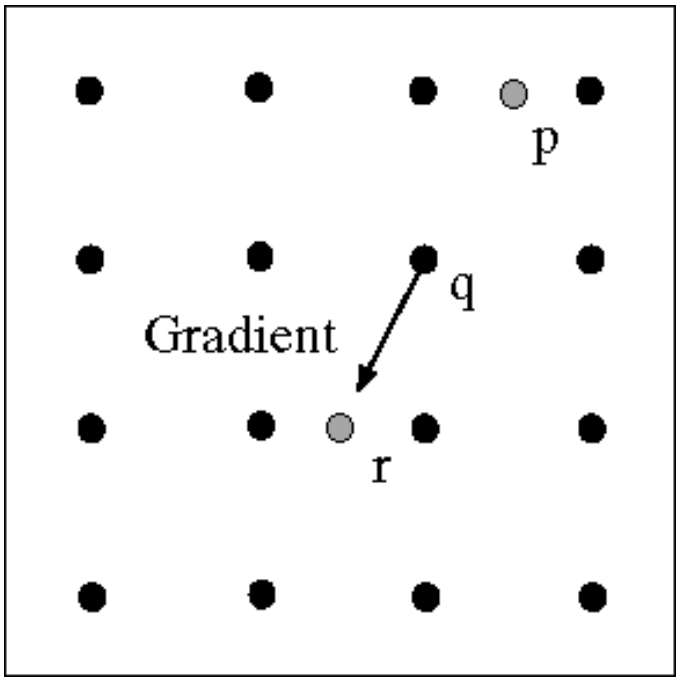




- How to turn these thick regions of the gradient into curves?

Non-maximum suppression





Check if pixel is local maximum along gradient direction, select single max across width of the edge

- requires checking interpolated pixels p and r

The Canny edge detector

•Problem: pixels along this edge didn't survive the thresholding

- thinning
- (non-maximum suppression)

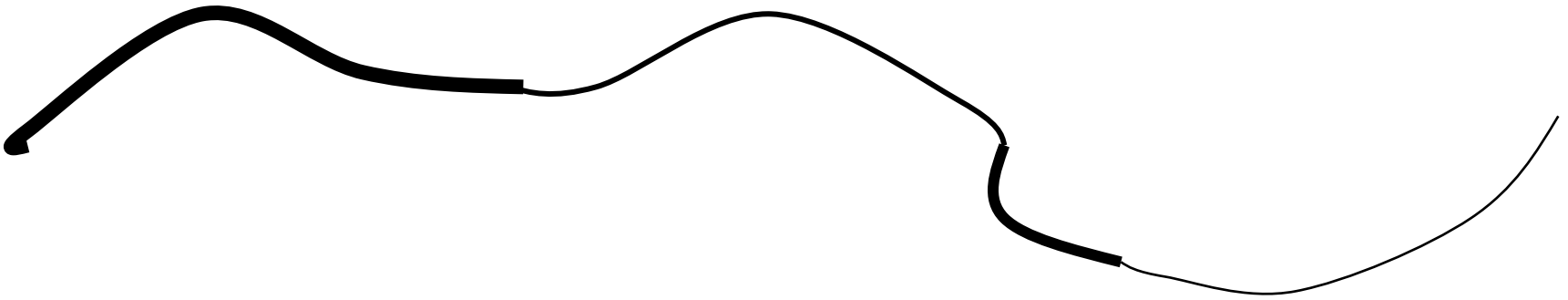
Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels

•Credit: James Hay

Hysteresis thresholding

- Use a high threshold to start edge curves, and a low threshold to continue them.



•Source: Steve Seitz

Final Canny Edges

•Credit: Jame

Recap: Canny edge detector

- Filter image with derivative of Gaussian
- Find magnitude and orientation of gradient
- **Non-maximum suppression**:
  - Thin wide “ridges” down to single pixel width
- **Linking and thresholding** (**hysteresis**):
  - Define two thresholds: low and high
  - Use the high threshold to start edge curves and the low threshold to continue them

- MATLAB: `edge(image, 'canny');`
- `>>help edge`

•Source: D. Lowe, L. Fei-Fei

Mask properties

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter
- Derivatives
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 → no response in constant regions
  - High absolute value at points of high contrast

•Slide credit: Kristen Grauman