C280, Computer Vision

Prof. Trevor Darrell

trevor@eecs.berkeley.edu

Lecture 4: Filtering

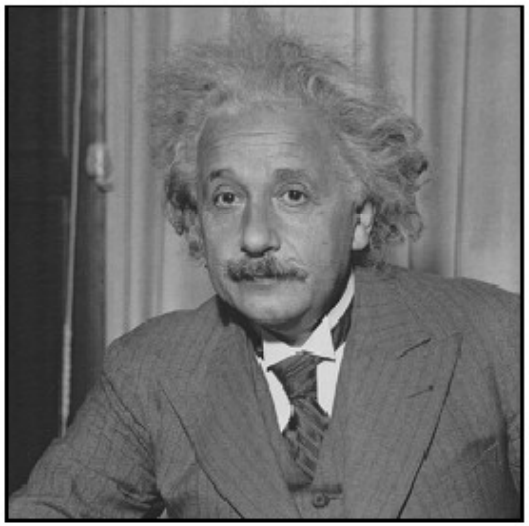
Adminstrivia

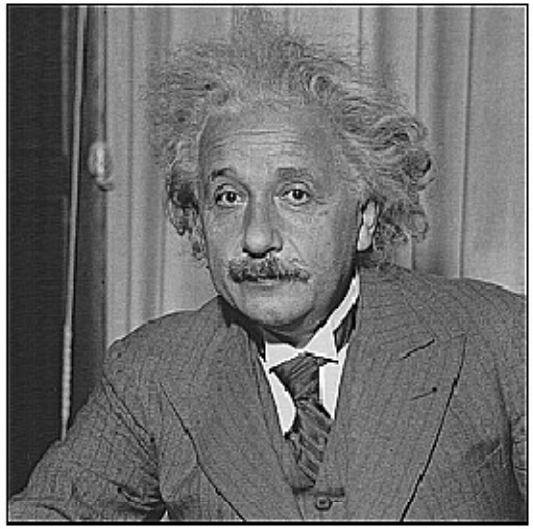
- bSpace returned…let's keep our fingers crossed…
- box.net mirror:
  - http://www.box.net/shared/9q8cdvqrco
- "external homepage"
  - http://www.eecs.berkeley.edu/~trevor/CS280.html
  - (has lecture notes, but no assignments, etc.)
- few assignments from (recently un-)WL'ed folks?
  - please do **upload via bSpace** (but keep a backup!)
- Thu 5pm-6pm office hour canceled this week.
- Pset 1 released; n.b. ***longer*** than usual, start early
- Pset 0 solutions on bSpace resource page

Last time: Color

- Measuring color
  - Spectral power distributions
  - Color mixing
  - Color matching experiments
  - Color spaces
    - Uniform color spaces
- Perception of color
  - Human photoreceptors
  - Environmental effects, adaptation
- Using color in machine vision systems

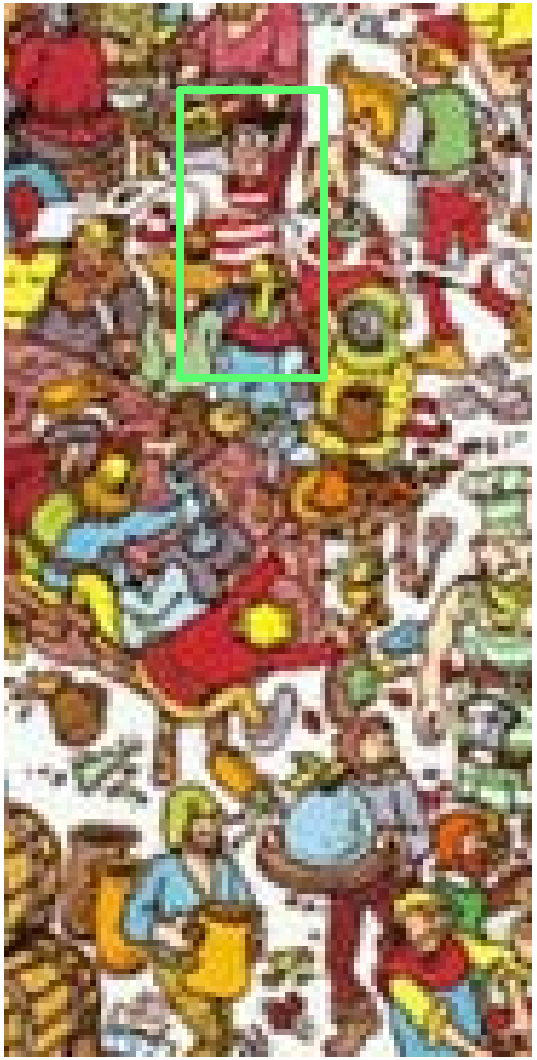
Today: Image Filters





Smooth/Sharpen Images...

Find edges...



Find waldo…

Image neighborhoods

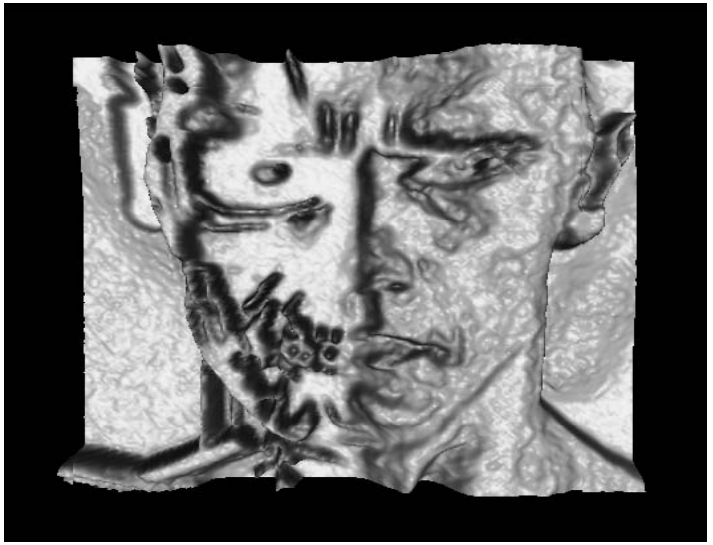
- Q: What happens if we reshuffle all pixels within the images?

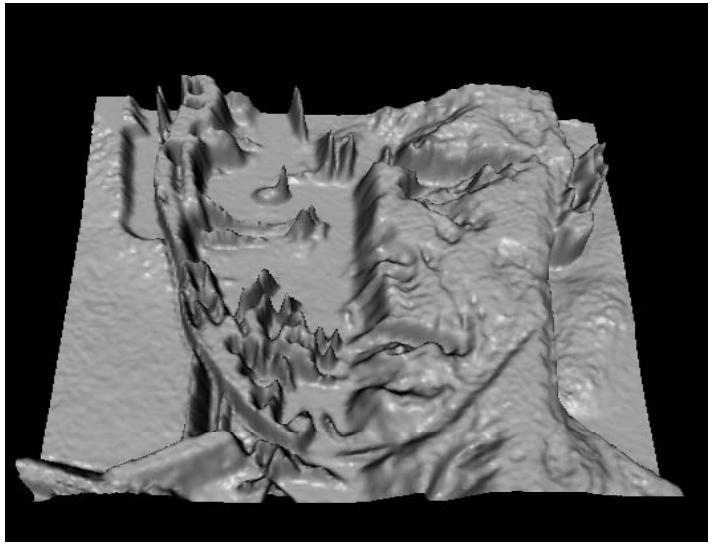


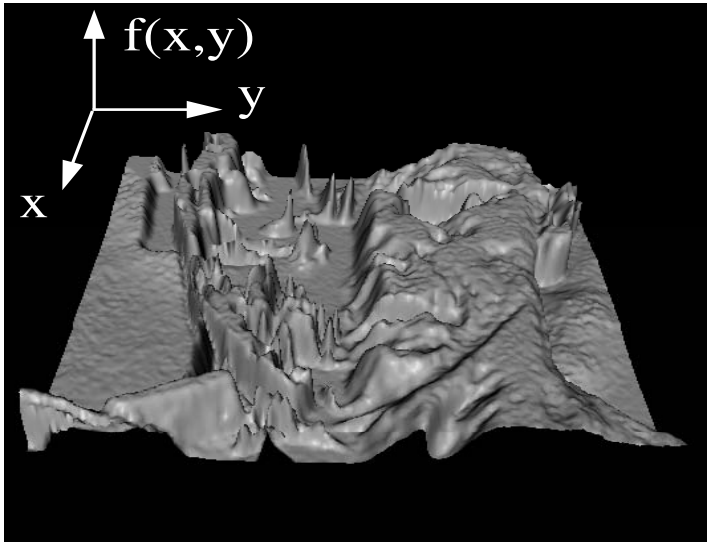

- A: Its histogram won't change.
  Point-wise processing unaffected.

- Need to measure properties relative to small *neighborhoods* of pixels

Images as functions







Source: S. Seitz

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: $[a,b] \times [c,d] \rightarrow [0, 1.0]$

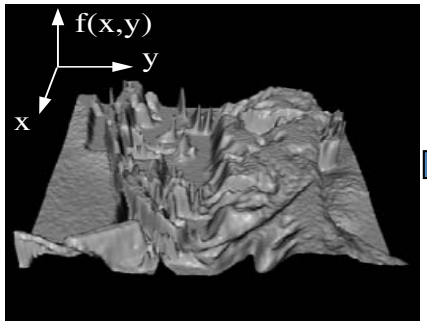
- A color image is just three functions pasted together. We can write this as a "vector-valued" function:

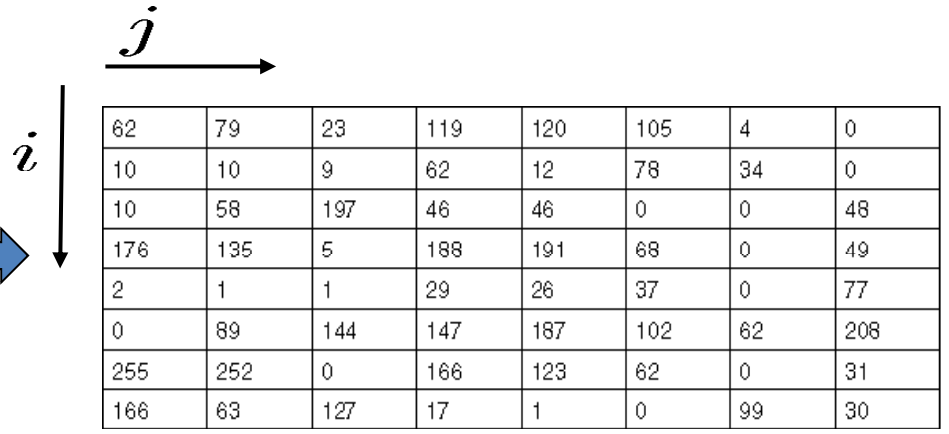
$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Source: S. Seitz

Digital images

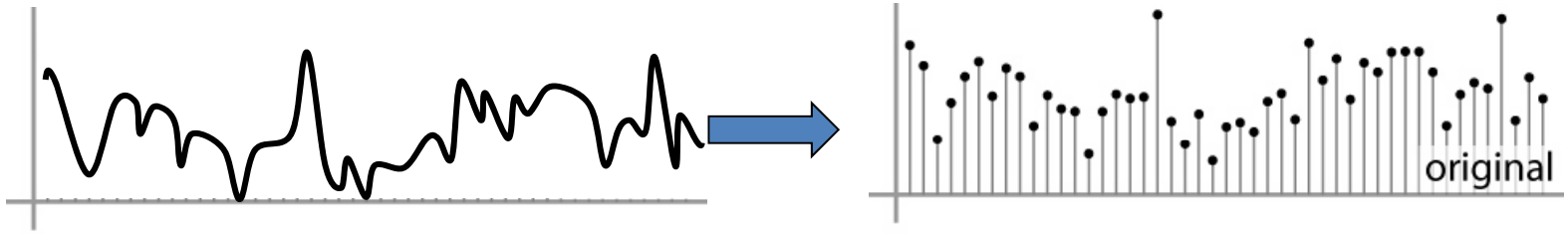
- In computer vision we operate on **digital** (**discrete**) images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to nearest integer)
- Image thus represented as a matrix of integer values.





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

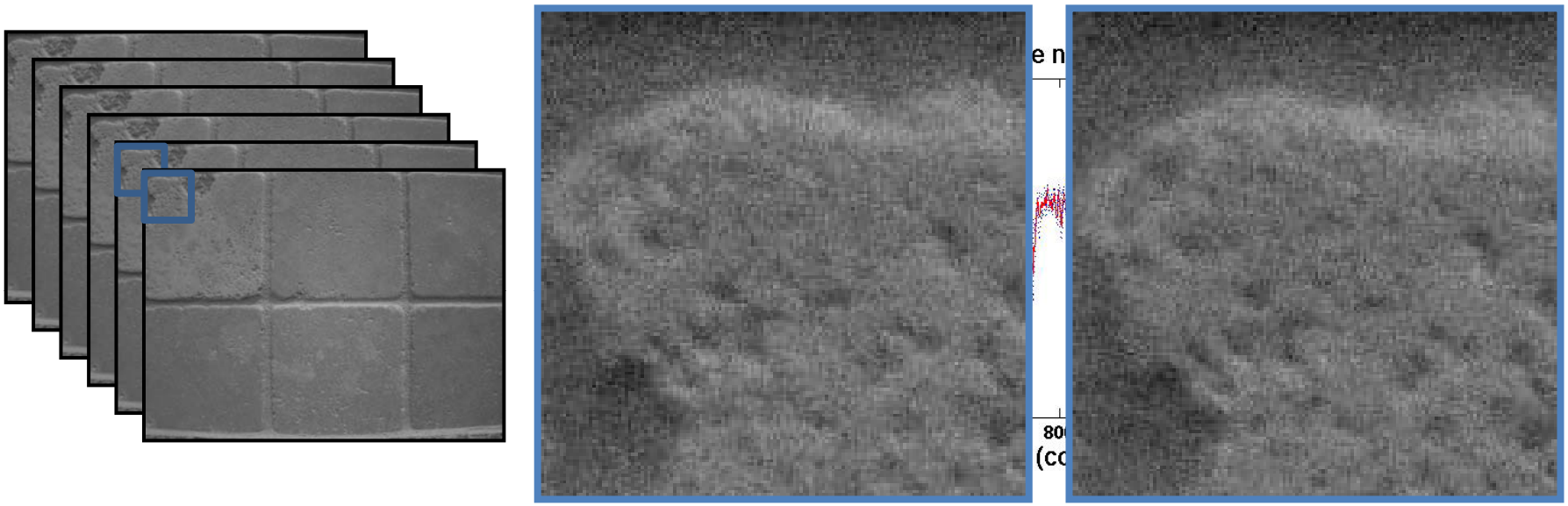
2D



1D

Adapted from S. Seitz

Motivation: noise reduction



- We can measure **noise** in multiple images of the same static scene.
- How could we reduce the noise, i.e., give an estimate of the true intensities?

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

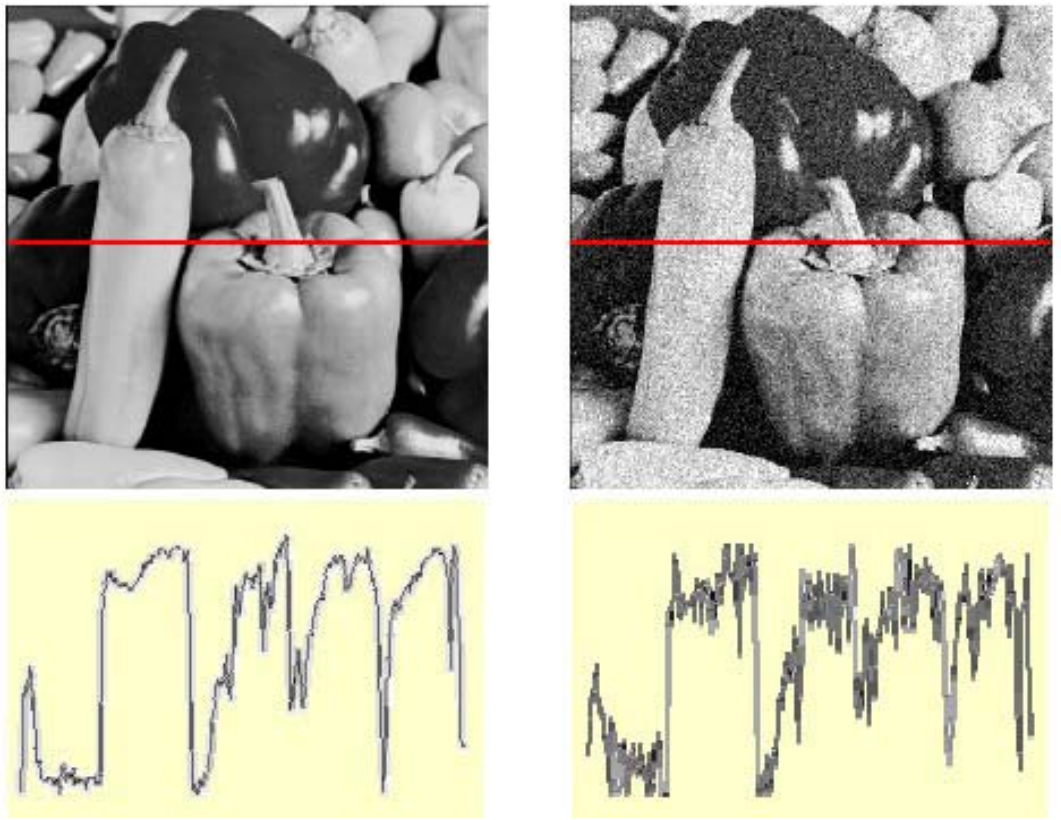
Salt and pepper noise

Impulse noise

Gaussian noise

Source: S. Seitz

Gaussian noise



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

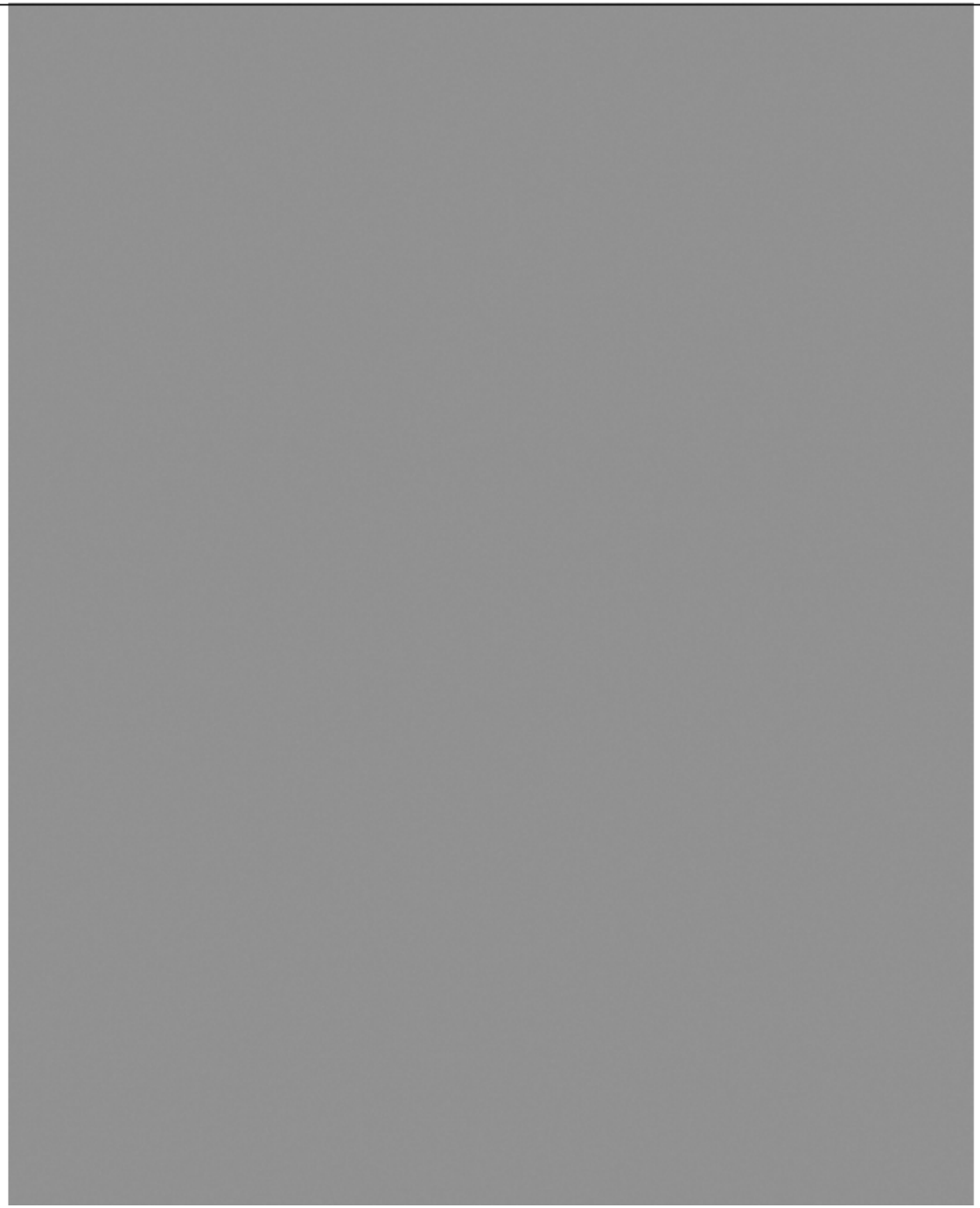
Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu, \sigma)$

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

>> output = im + noise;
```

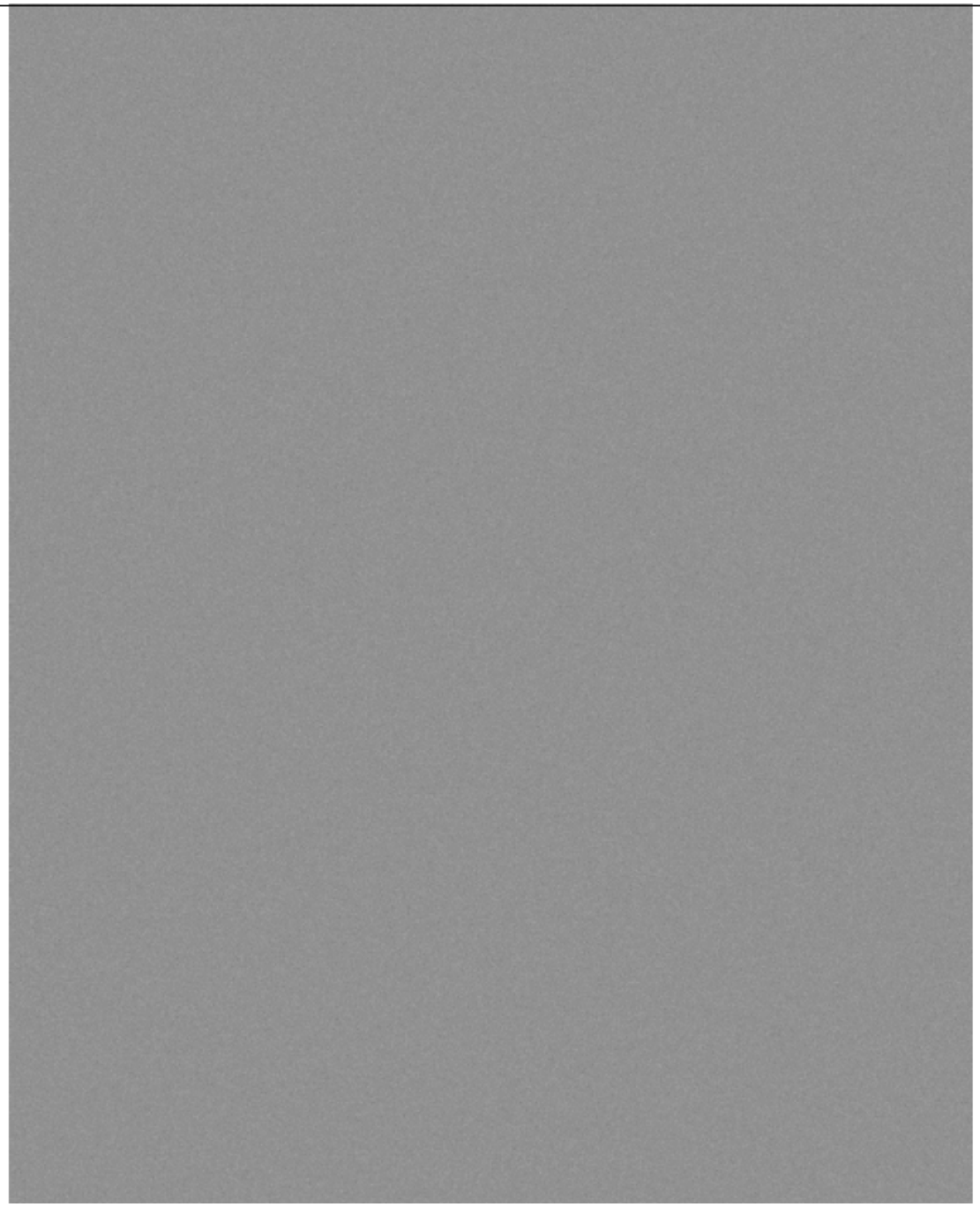

Fig: M. Hebert

sigma=1



Effect of sigma on Gaussian noise:

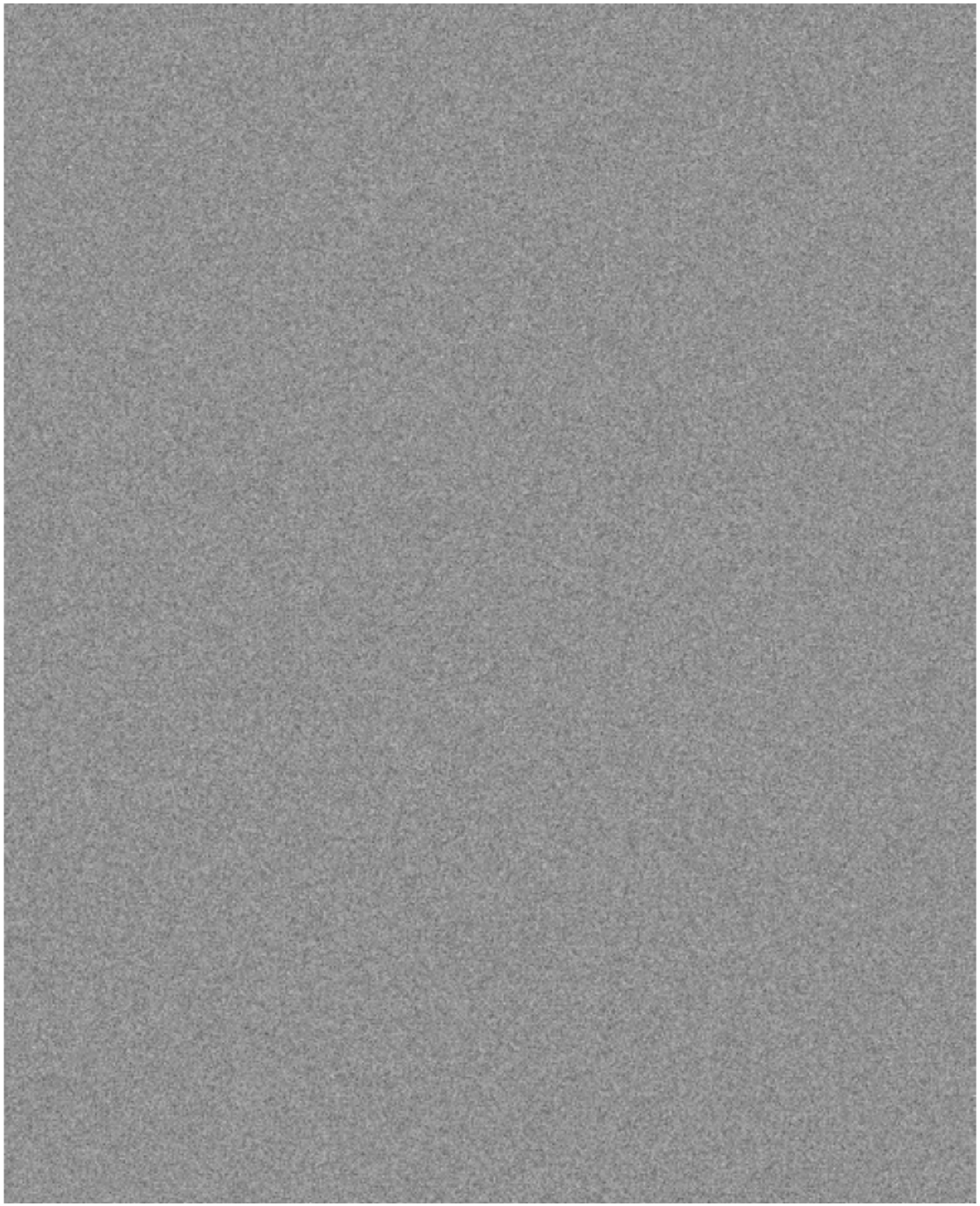
Image shows the noise values themselves.



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= 16

sigma=1

Effect of sigma on Gaussian noise:

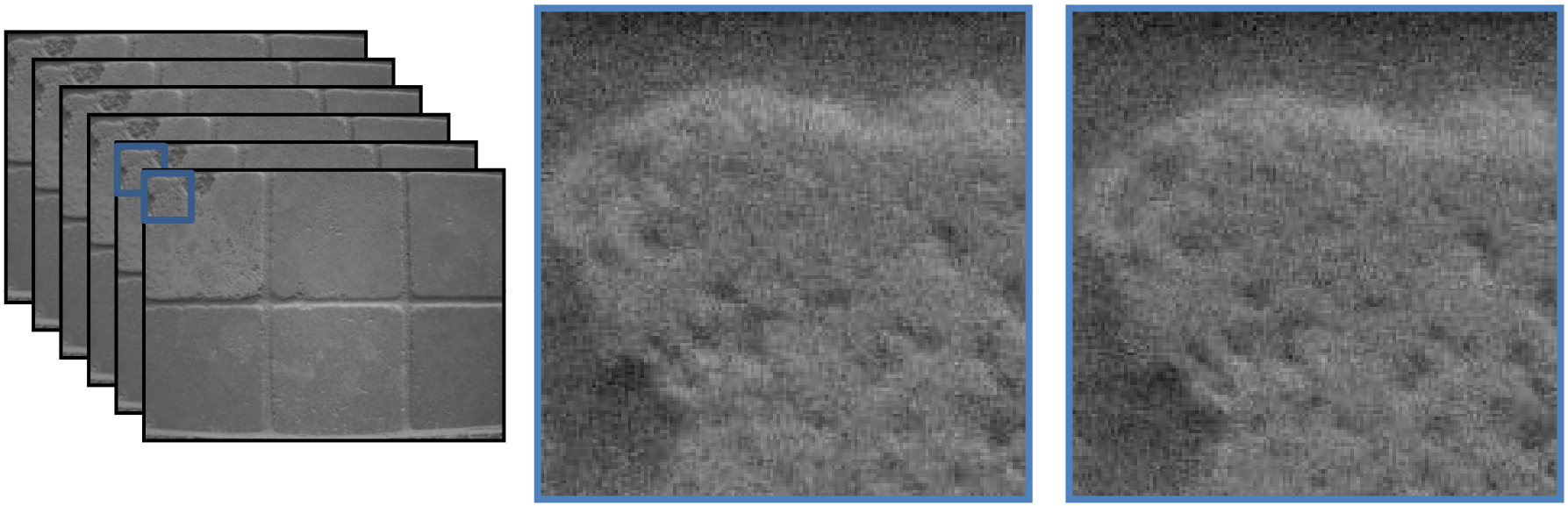
This shows the noise values added to the raw intensities of an image.

sigma=16

Effect of sigma on Gaussian noise

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

Motivation: noise reduction



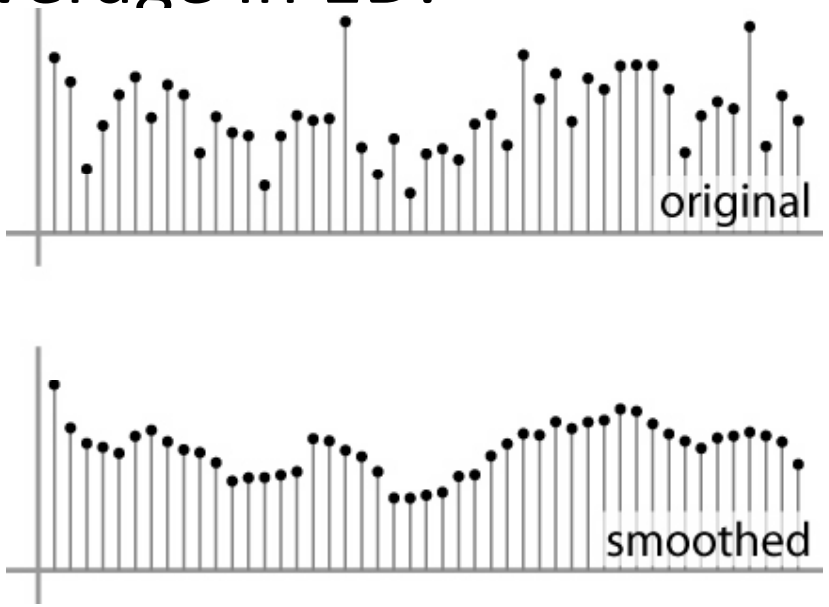
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- **What if there's only one image?**

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

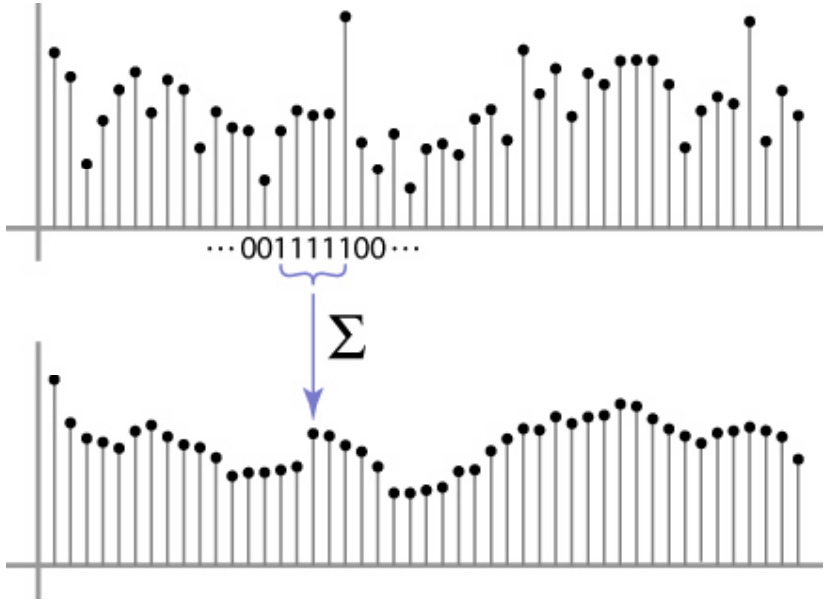
- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



Source: S. Marschner

Weighted Moving Average

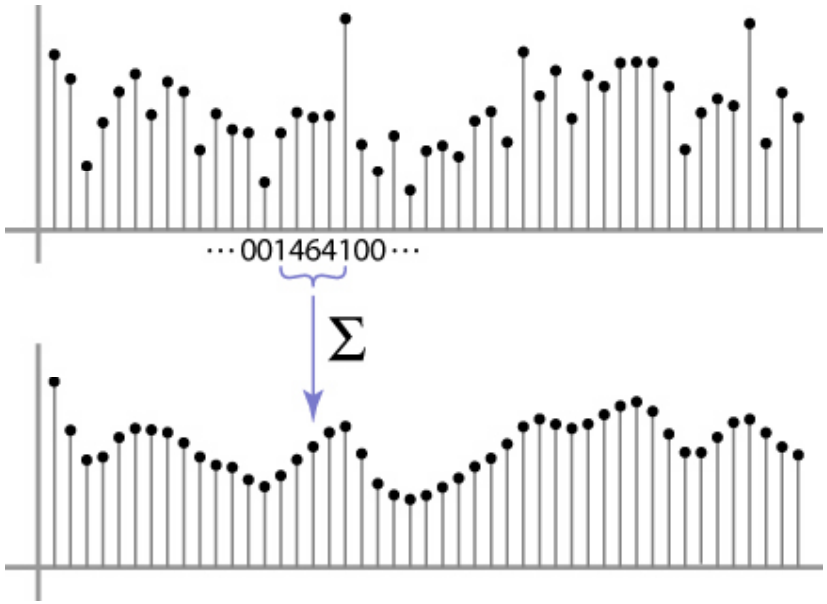
- Can add weights to our moving average
- *Weights* [1, 1, 1, 1, 1] / 5



Source: S. Marschner

Weighted Moving Average

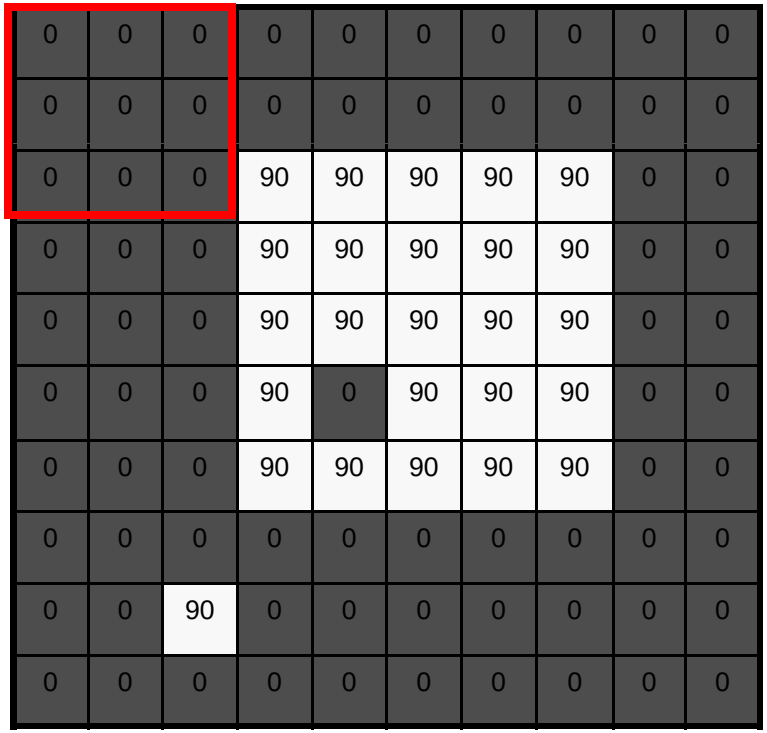
- Non-uniform weights [1, 4, 6, 4, 1] / 16



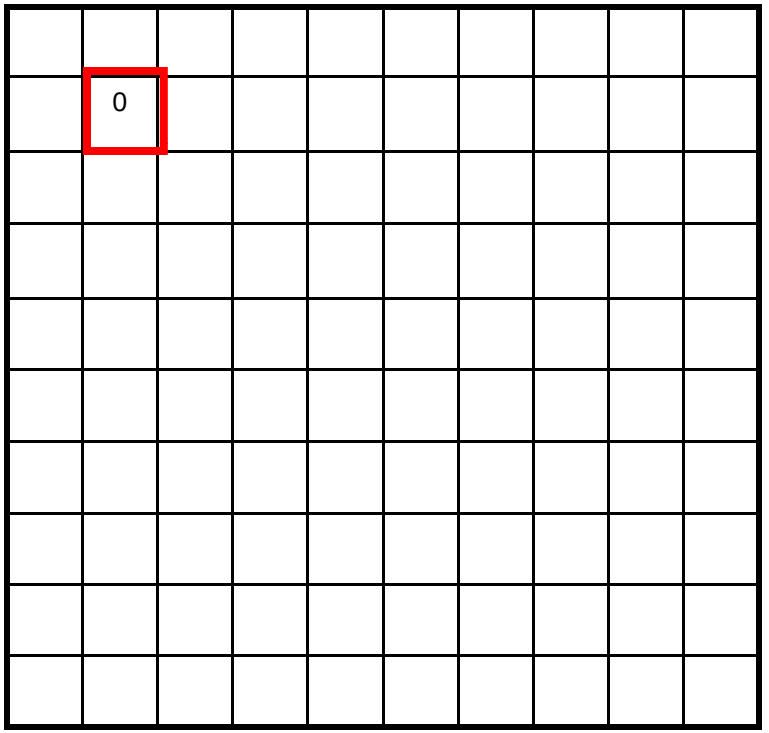
Source: S. Marschner

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:

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

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.

Averaging filter

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

$F[x, y]$

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

$\otimes$

$H[u, v]$

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

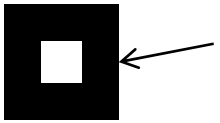
"box filter"

$G[x, y]$

	0	10	20	30	30				

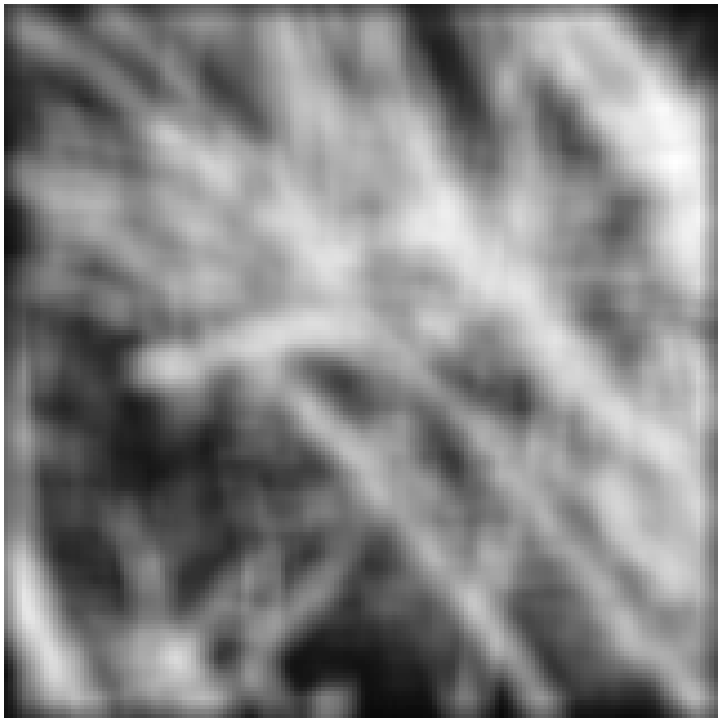
$$G = H \otimes F$$

Smoothing by averaging



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

original



filtered

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

$F[x, y]$

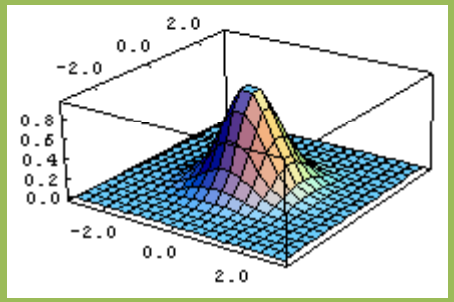
$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$H[u, v]$

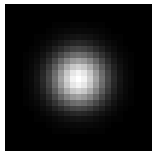
This kernel is an approximation of a Gaussian function:

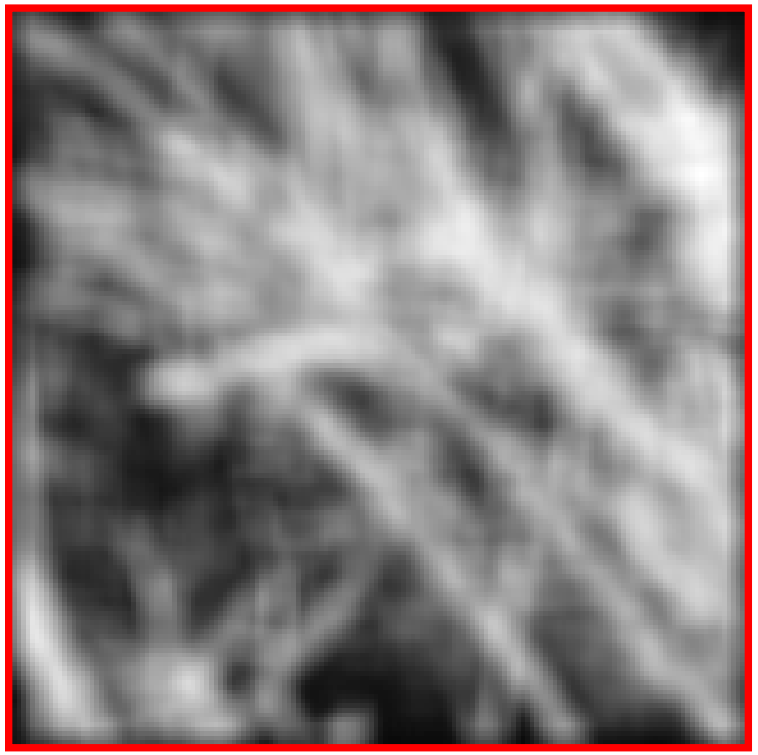
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Source: S. Seitz

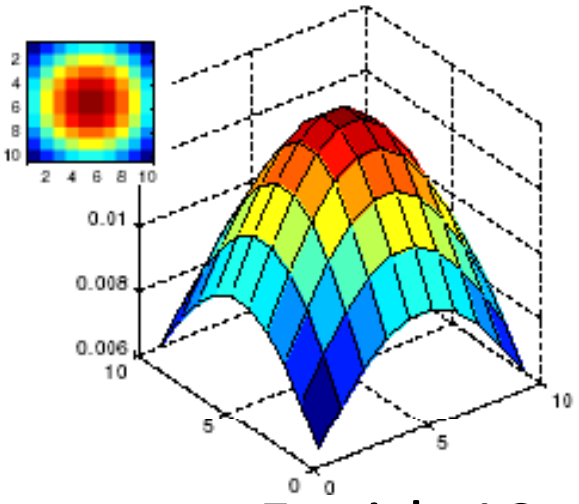
Smoothing with a Gaussian



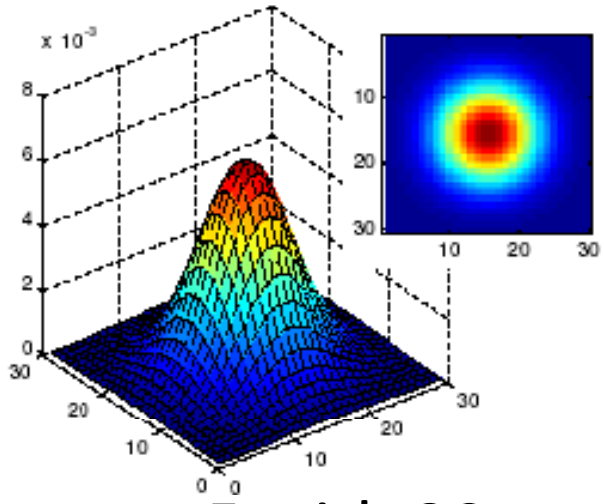


Gaussian filters

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



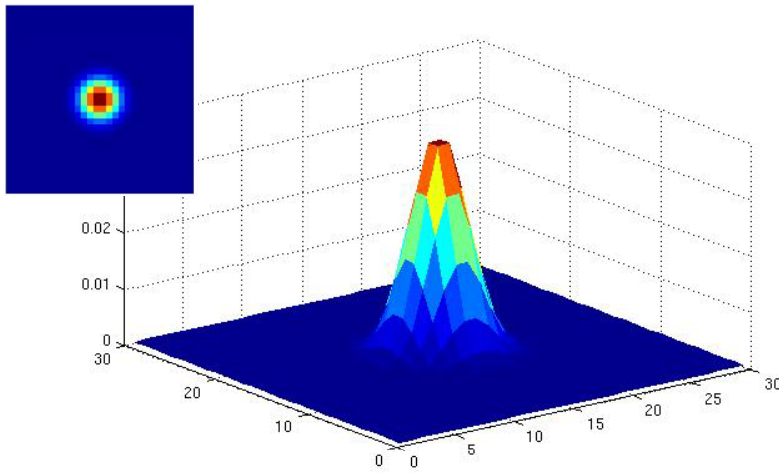
σ = 5 with 10 x 10 kernel



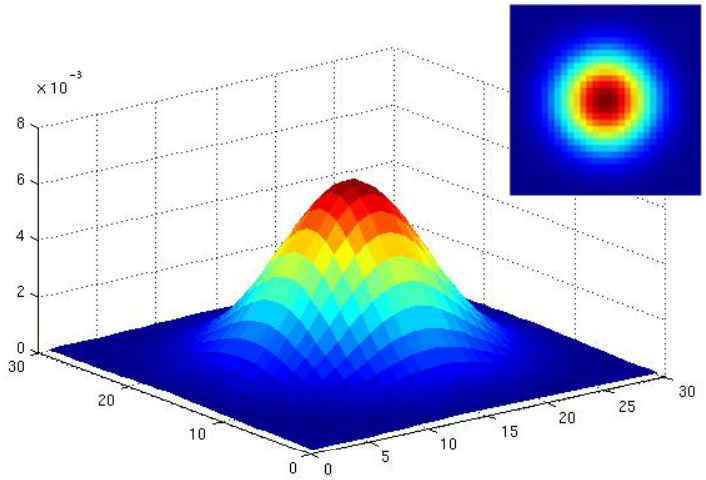
σ = 5 with 30 x 30 kernel

Gaussian filters

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



σ = 2 with 30 x 30 kernel

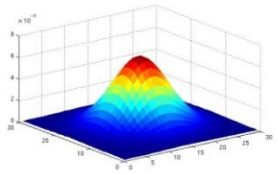


σ = 5 with 30 x 30 kernel

Matlab

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

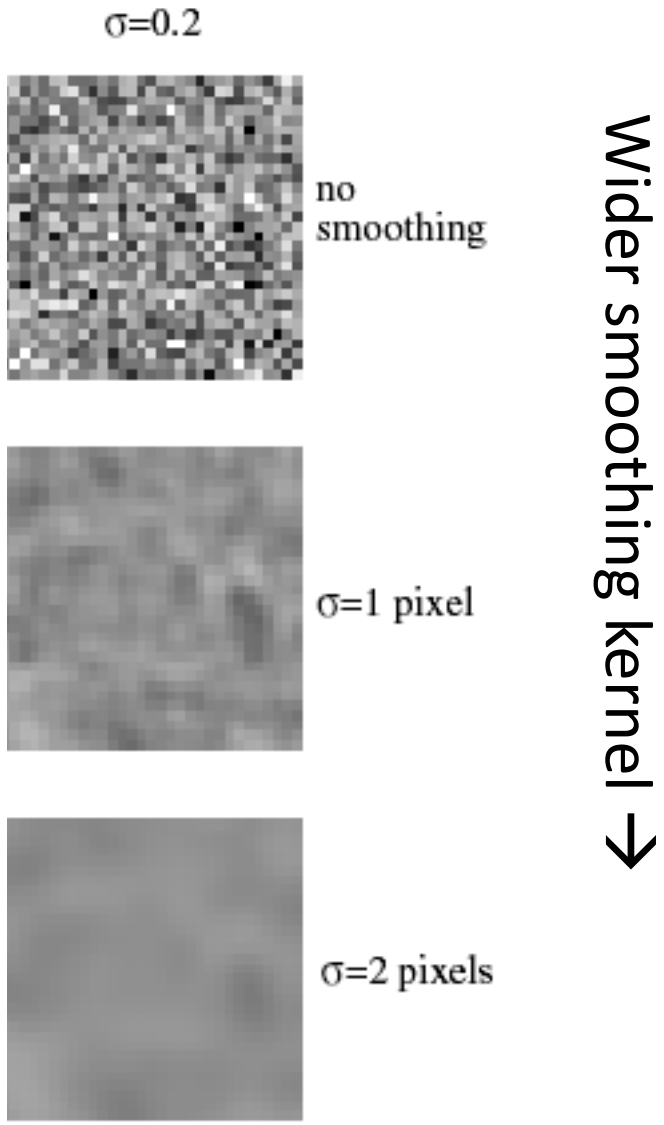
```
>> mesh(h);
```



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

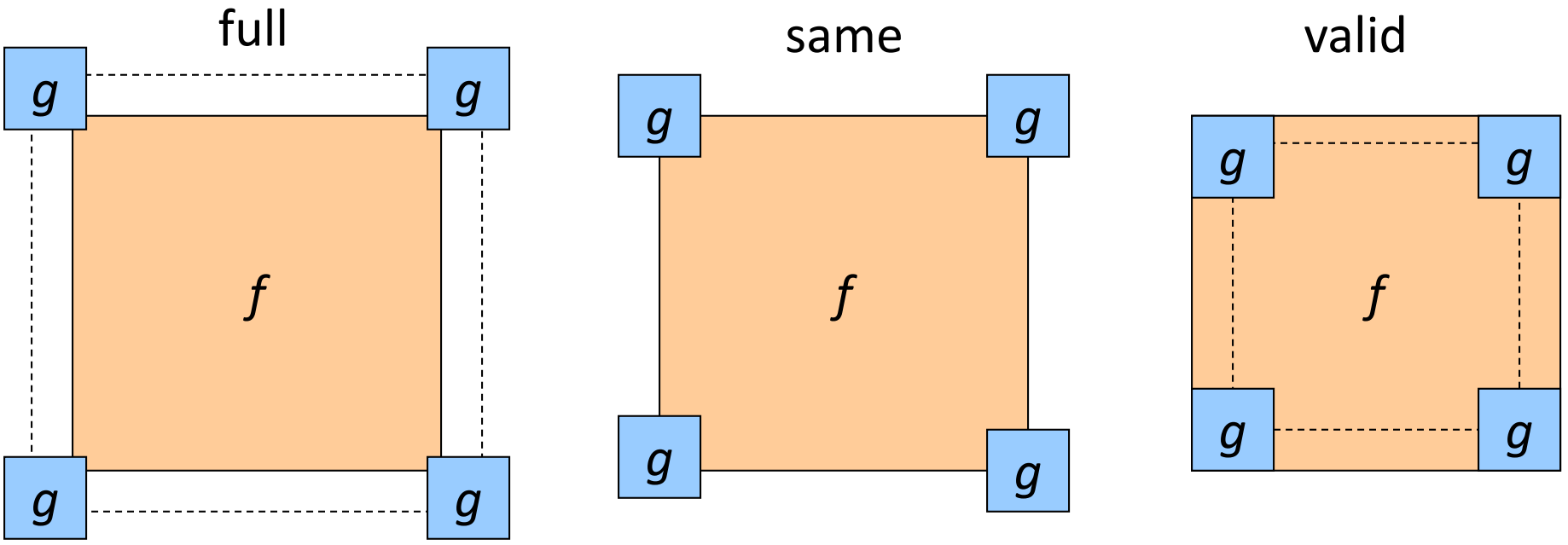
```
>> outim = imfilter(im, h);
>> imshow(outim);
```

outim


σ=0.2
no smoothing
σ=1 pixel
σ=2 pixels
Wider smoothing kernel →

Boundary issues

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



Source: S. Lazebnik

Boundary issues

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

Source: S. Marschner

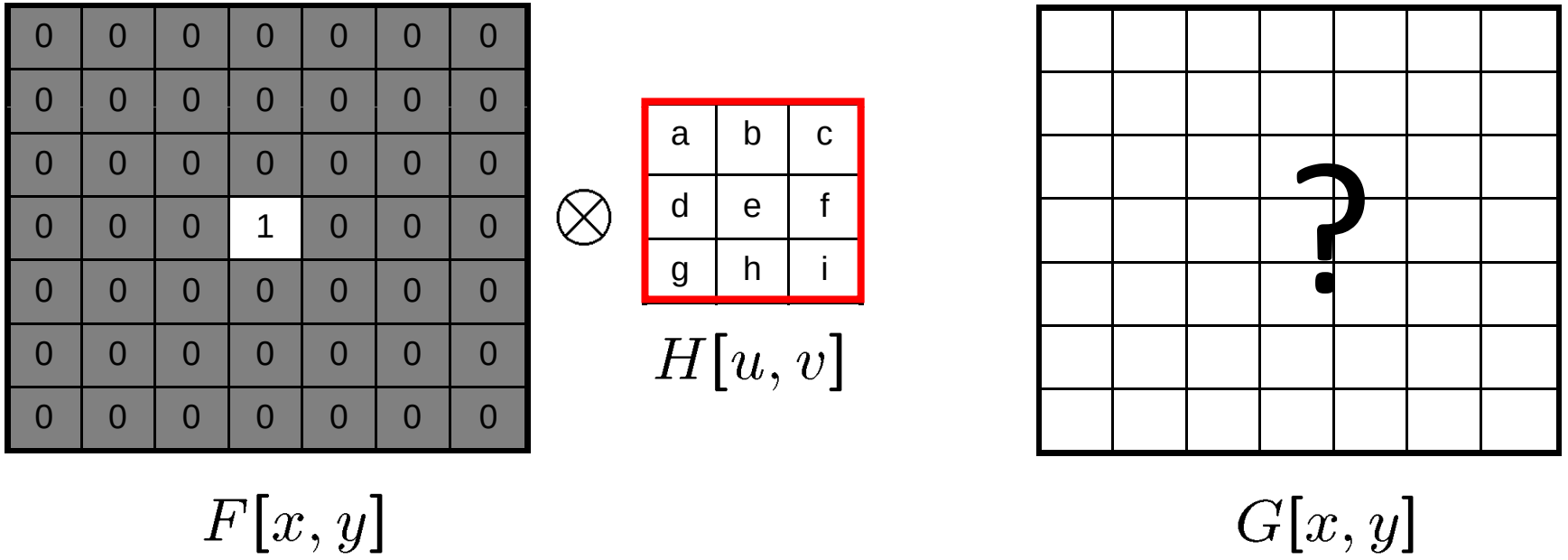
Boundary issues

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods (MATLAB):
    - clip filter (black): imfilter(f, g, 0)
    - wrap around: imfilter(f, g, ‘circular’)
    - copy edge: imfilter(f, g, ‘replicate’)
    - reflect across edge: imfilter(f, g, ‘symmetric’)

Source: S. Marschner

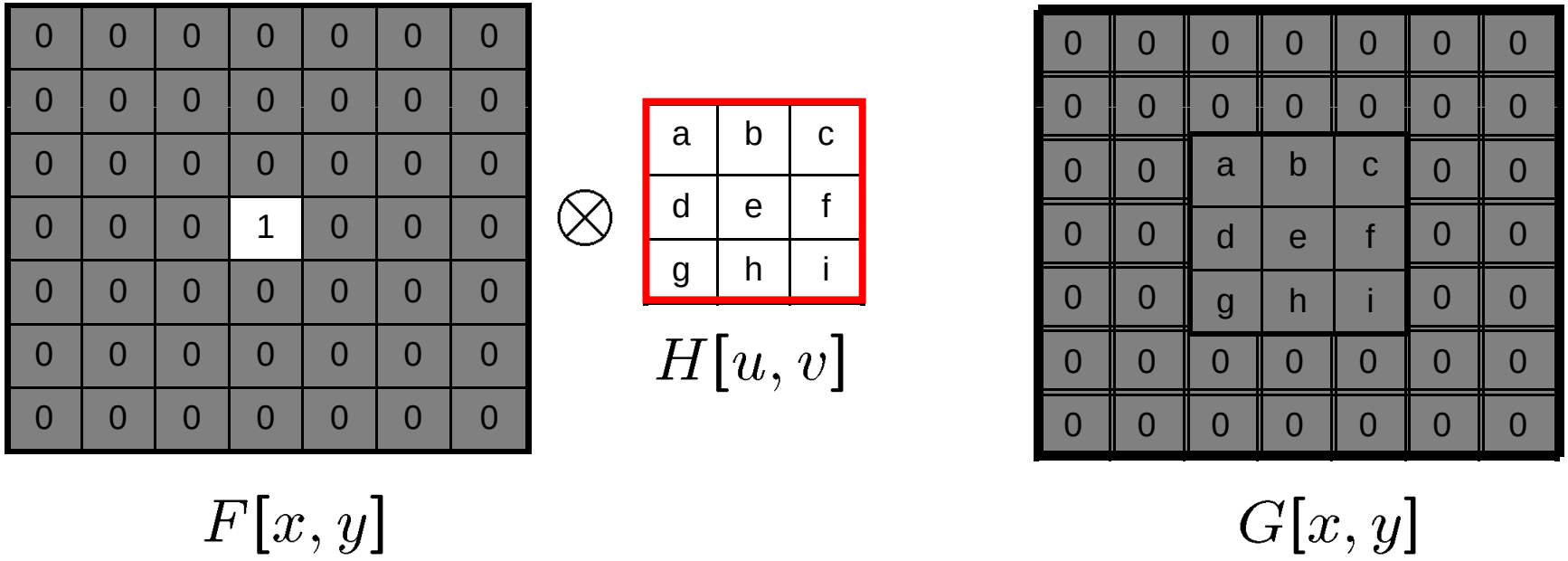
Filtering an impulse signal

What is the result of filtering the impulse signal (image) *F* with the arbitrary kernel *H*?



Filtering an impulse signal

What is the result of filtering the impulse signal (image) *F* with the arbitrary kernel *H*?



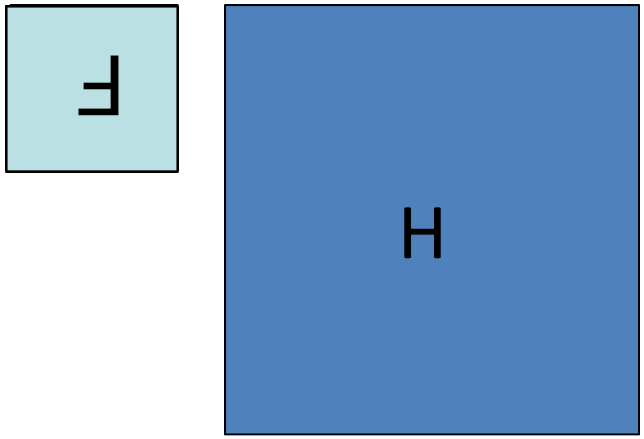
Convolution

- 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 vs. correlation

Convolution

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

$$G = H \star F$$

Cross-correlation

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

$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

If the input is an impulse signal, how will the outputs differ?

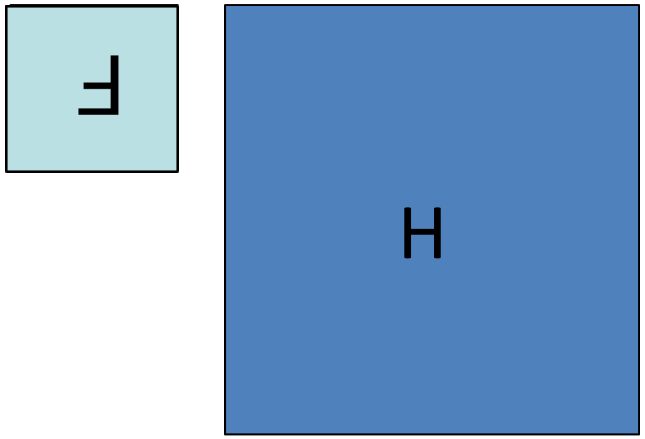
Convolution

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

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

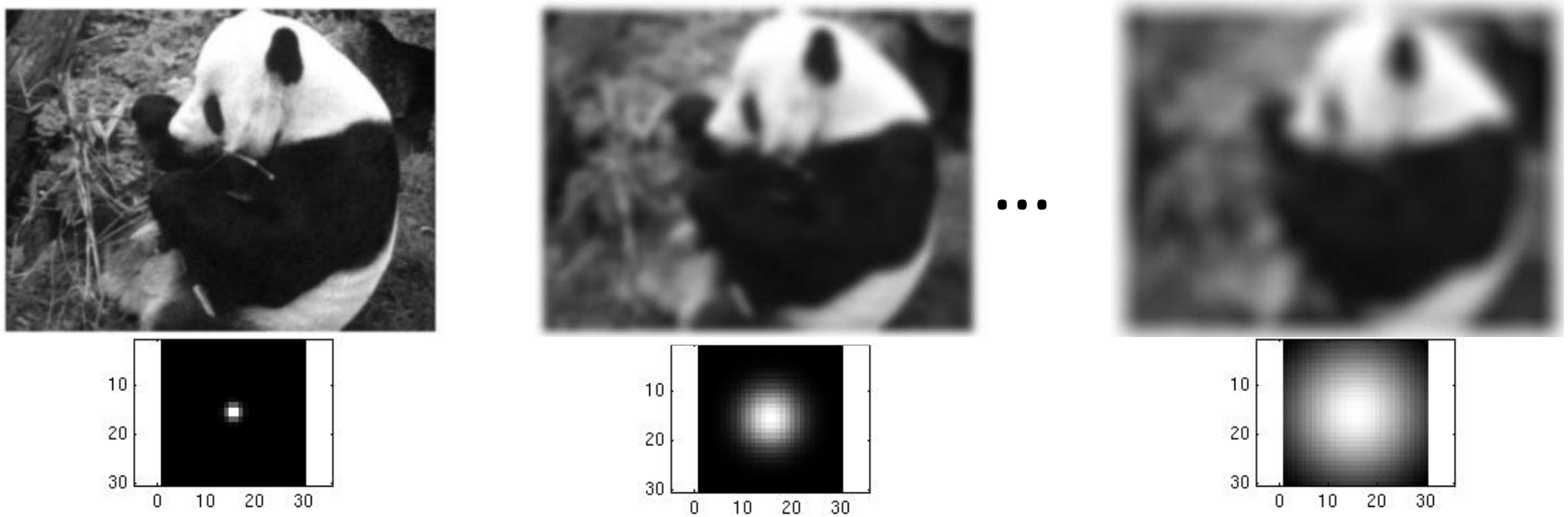
$$G = H \star F$$

Notation for convolution operator



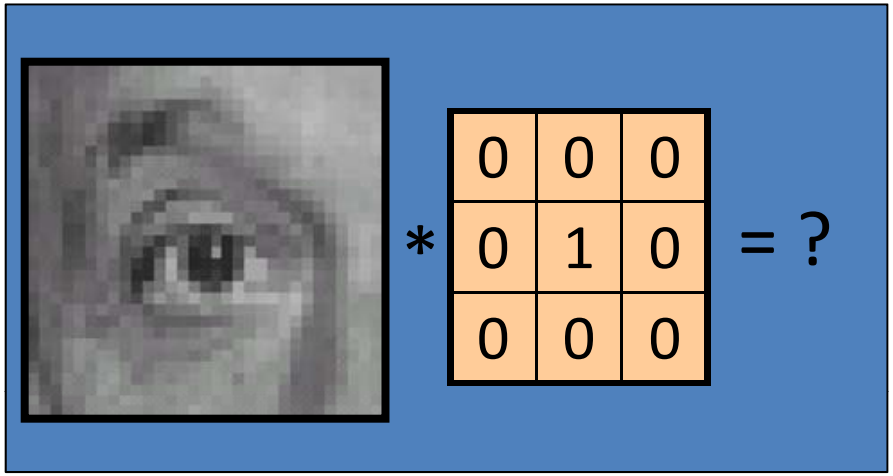
Smoothing with a Gaussian

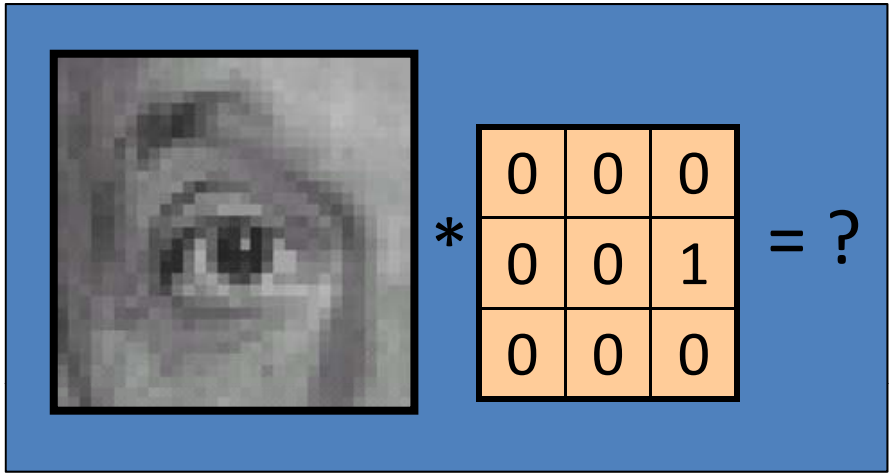
Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

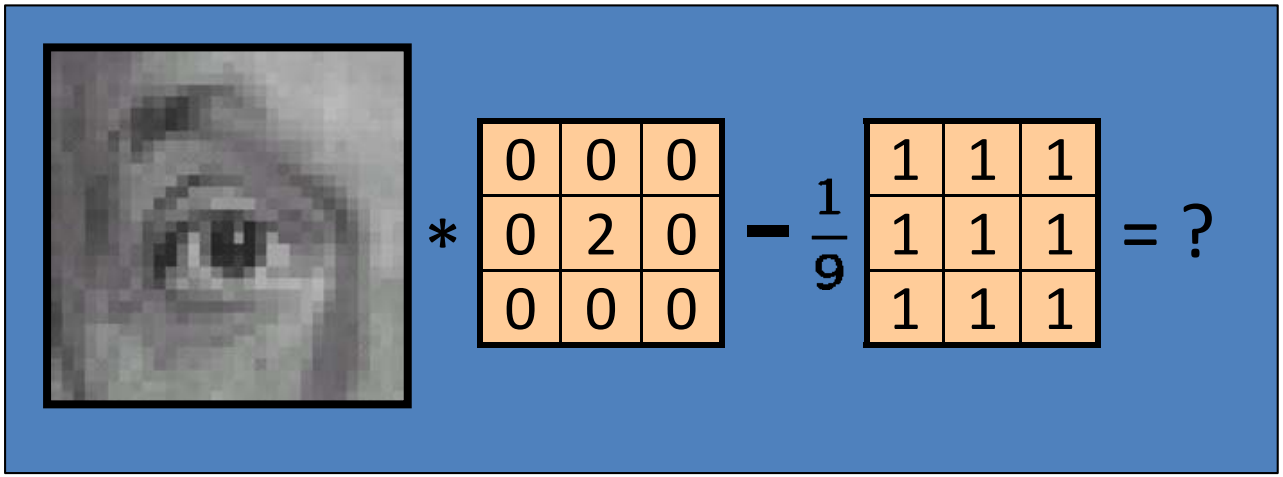



```
for sigma=1:3:10
   h = fspecial('gaussian‘, fsize, sigma);
   out = imfilter(im, h);
   imshow(out);
   pause;
end
```

Predict the filtered outputs







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

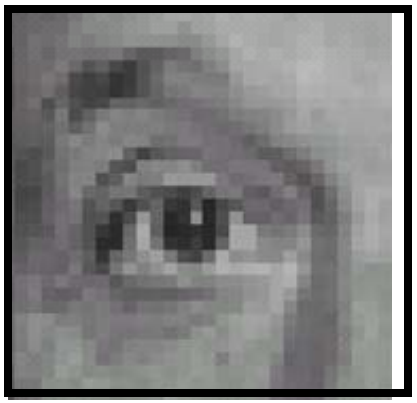
?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted left by 1 pixel with correlation

Source: D. Lowe

Practice with linear filters

Original

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

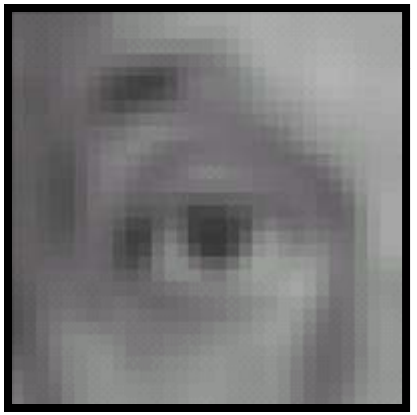
?

Source: D. Lowe

Practice with linear filters

Original

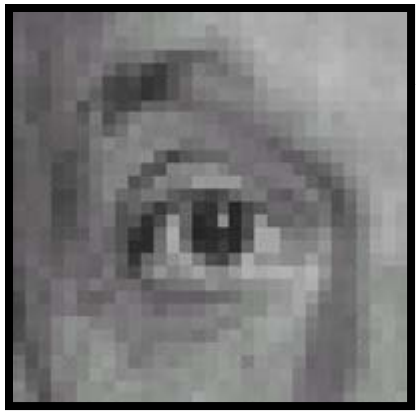
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Blur (with a box filter)

Source: D. Lowe

Practice with linear filters

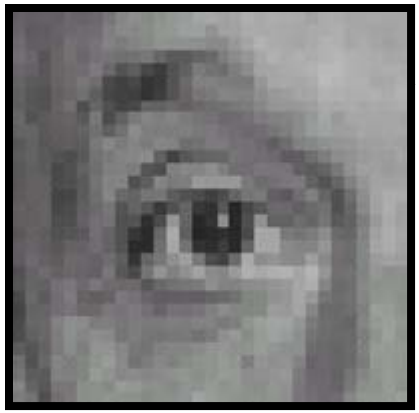


Original

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

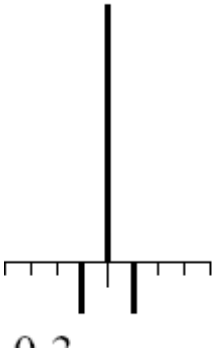
Source: D. Lowe

Practice with linear filters



Original

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

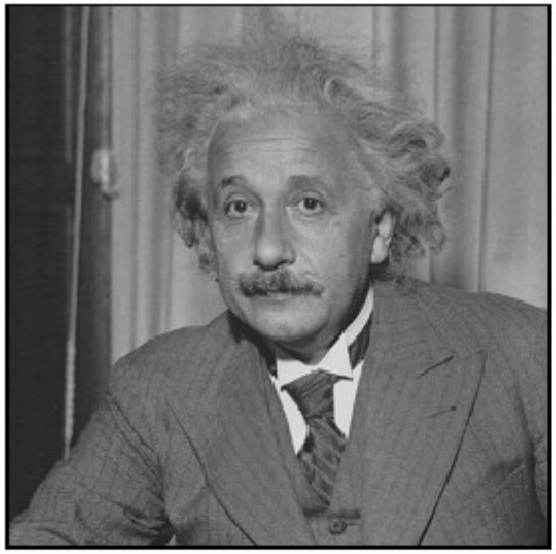


Sharpening filter

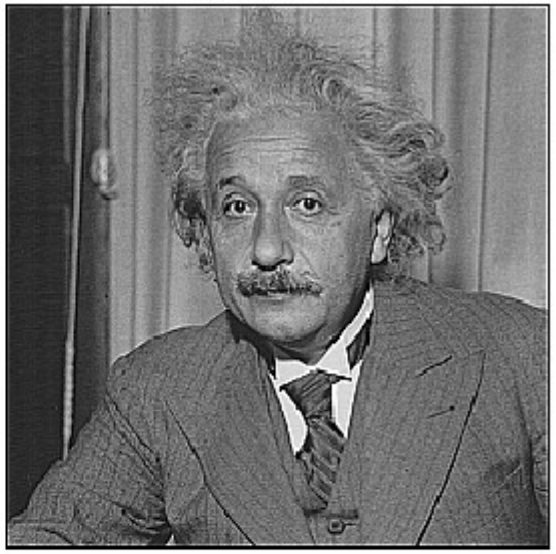
- Accentuates differences with local average

Source: D. Lowe

Filtering examples: sharpening



before



after

Shift invariant linear system

- **Shift invariant:**
  - Operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.
- **Linear:**
  - Superposition: h * (f1 + f2) = (h * f1) + (h * f2)
  - Scaling: h * (k f) = k (h * f)

Properties of convolution

- Linear & shift invariant
- Commutative:

  f * g = g * f
- Associative

  (f * g) * h = f * (g * h)
- Identity:

  unit impulse e = […, 0, 0, 1, 0, 0, …]. f * e = f
- Differentiation:

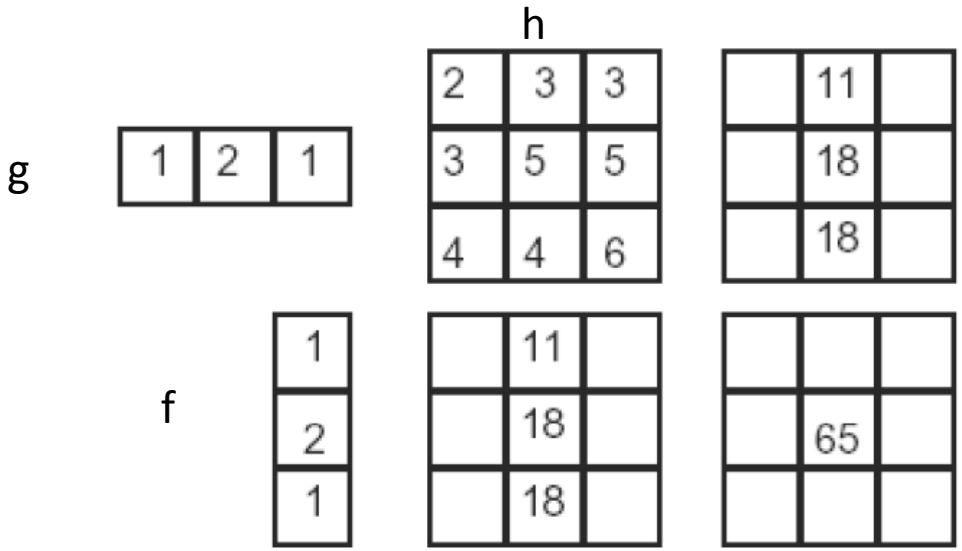
$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

Separability

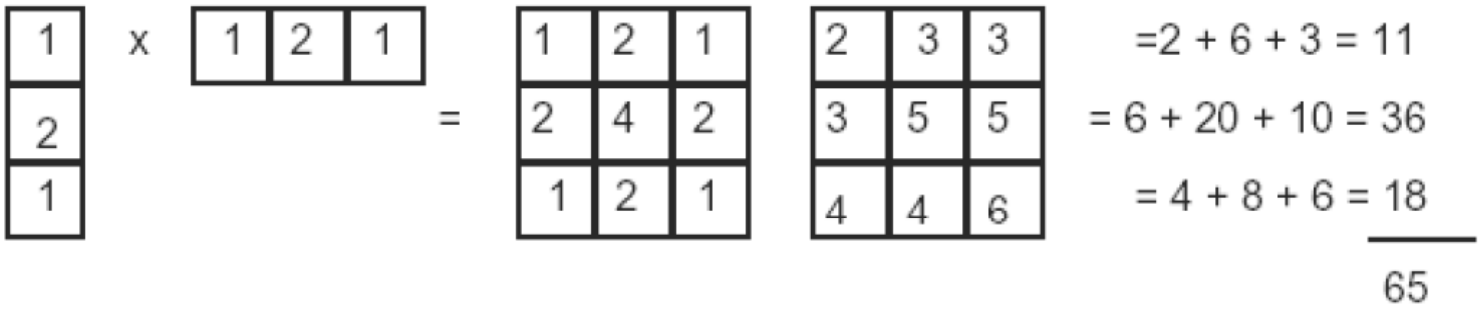
- In some cases, filter is separable, and we can factor into two steps:
  - Convolve all rows
  - Convolve all columns

Separability

- In some cases, filter is separable, and we can factor into two steps: e.g.,



What is the computational complexity advantage for a separable filter of size k x k, in terms of number of operations per output pixel?



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

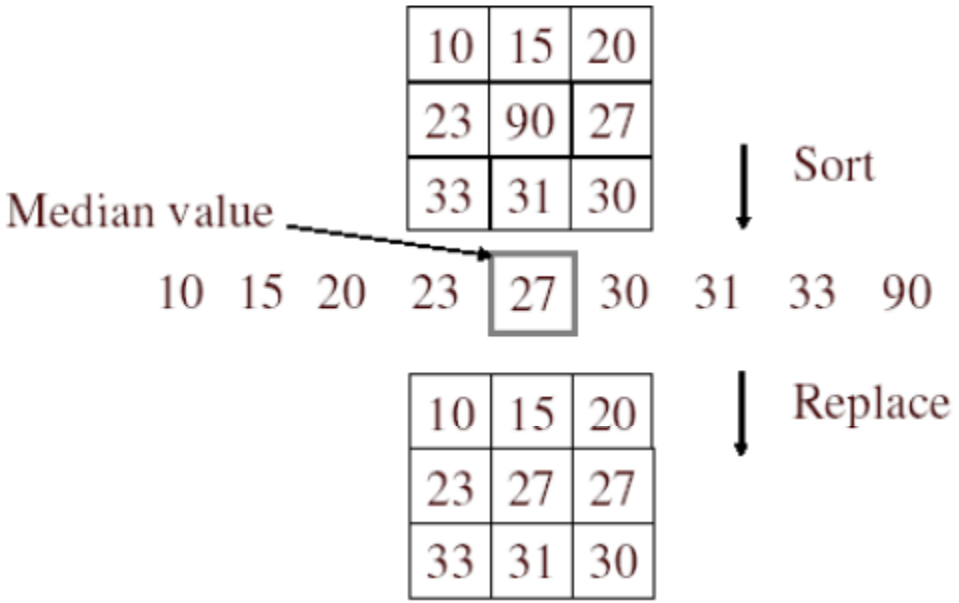
Effect of smoothing filters

5x5

Additive Gaussian noise

Salt and pepper noise

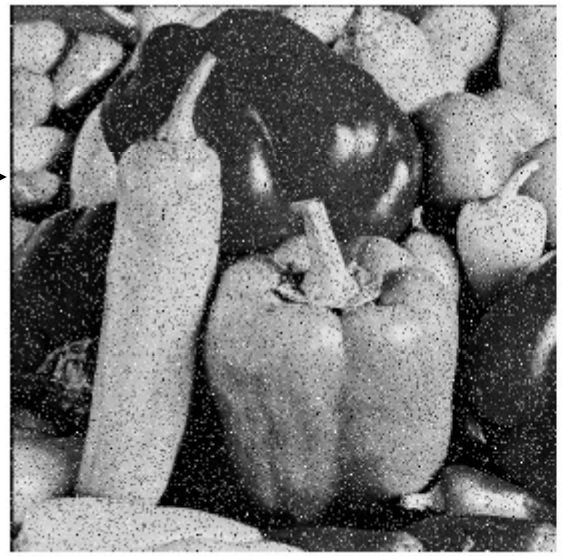
Median filter



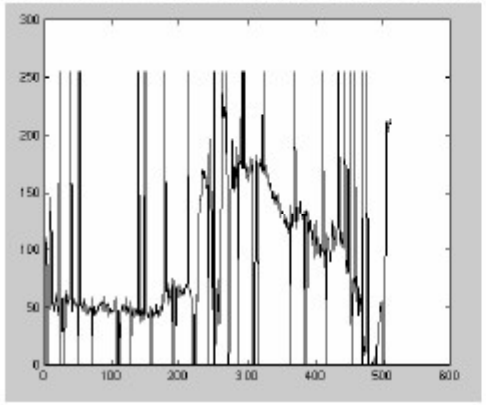
- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise

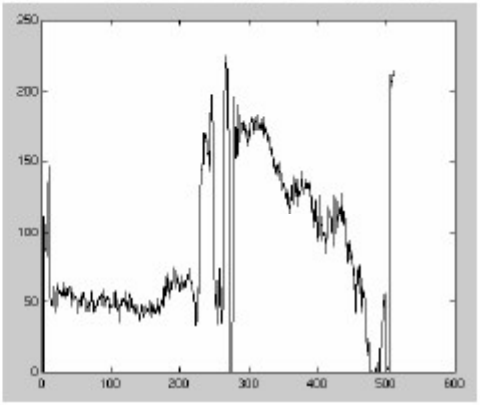
Median filter

Salt and pepper noise



Median filtered



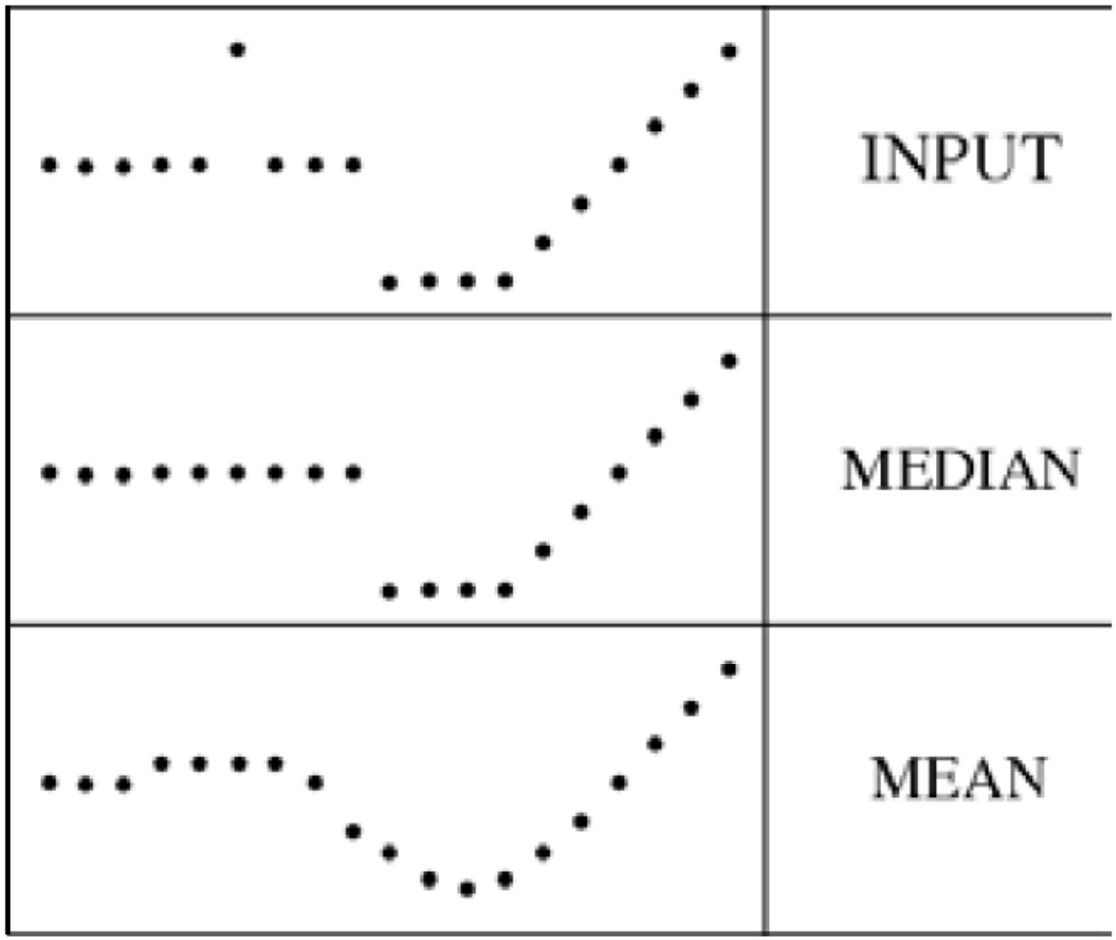


Plots of a row of the image

Source: M. Hebert

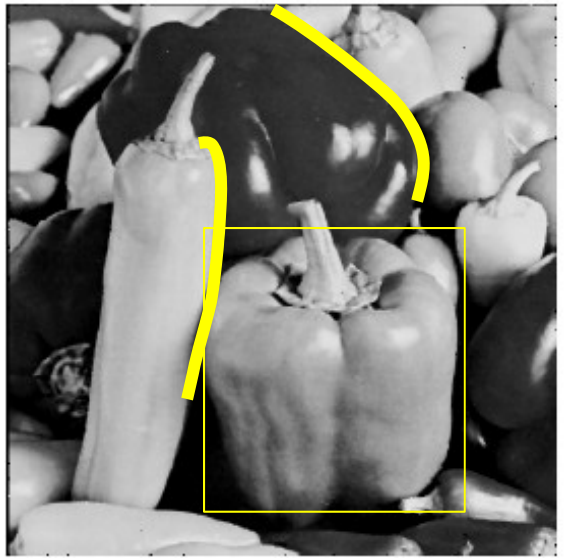
Median filter

- Median filter is edge preserving



Filters for features

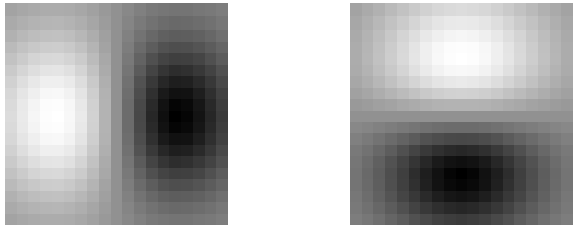
- Previously, thinking of filtering as a way to remove or reduce **noise**
- Now, consider how filters will allow us to abstract higher-level "**features**".
  - Map raw pixels to an intermediate representation that will be used for subsequent processing
  - Goal: reduce amount of data, discard redundancy, preserve what's useful



Template matching

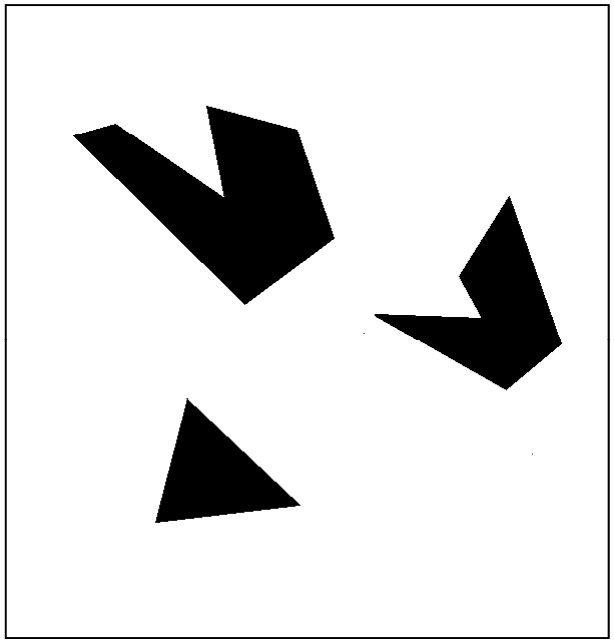
- Filters as **templates**:

  Note that filters look like the effects they are intended to find --- “matched filters”

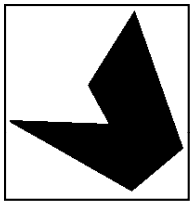


- Use normalized cross-correlation score to find a given pattern (template) in the image.
  - Szeliski Eq. 8.11
- Normalization needed to control for relative brightnesses.

Template matching



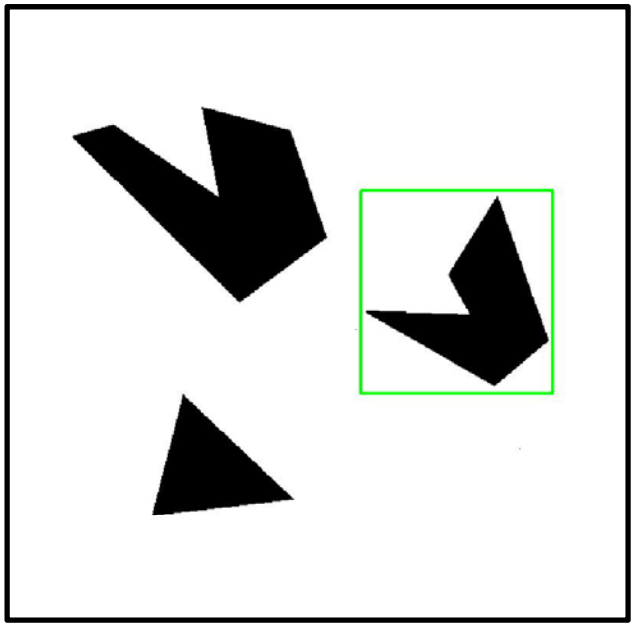
Scene



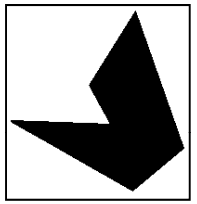
Template (mask)

A toy example

Template matching

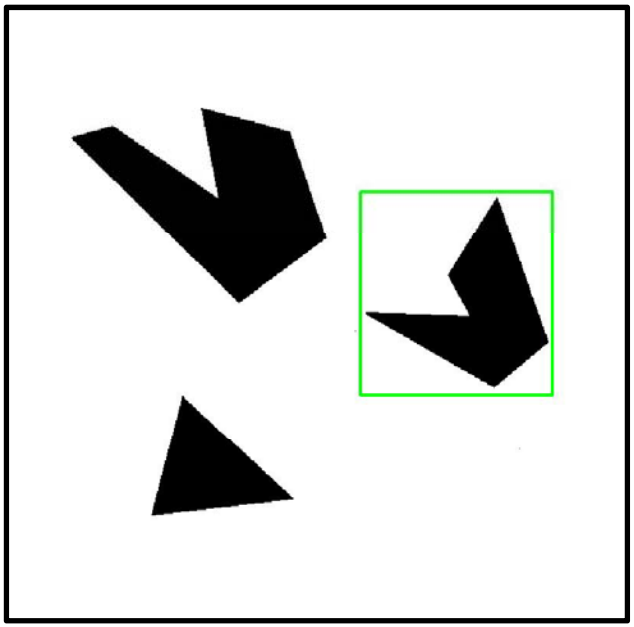


Detected template

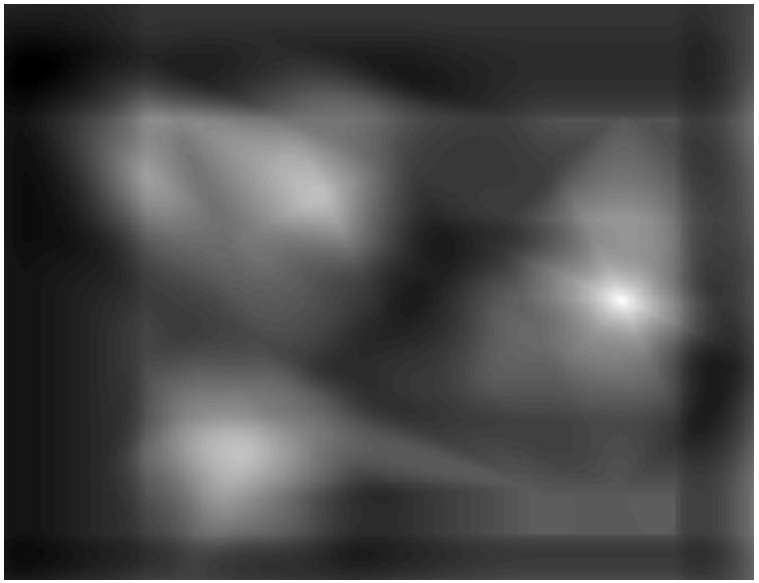


Template

Template matching

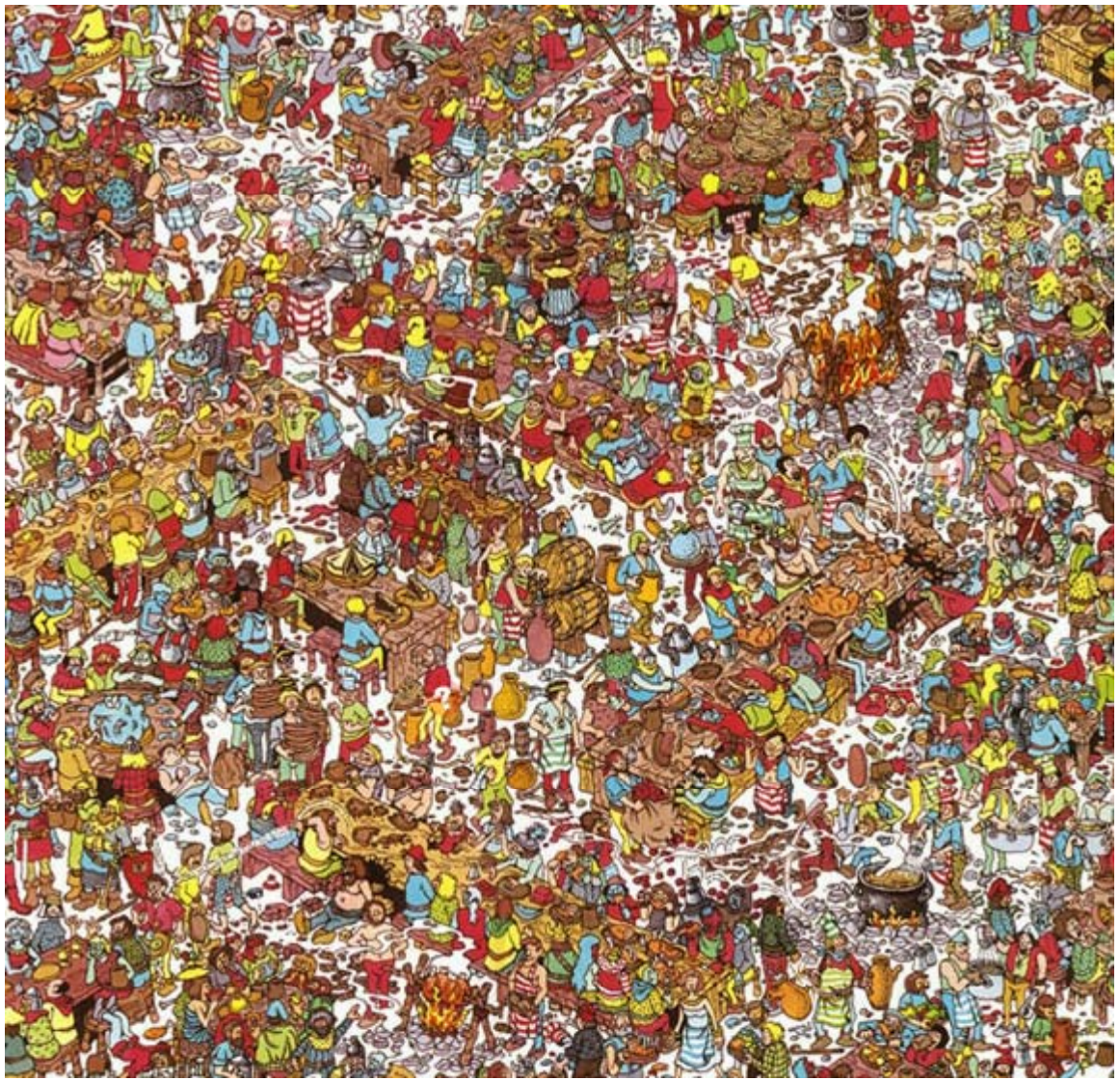


Detected template



Correlation map

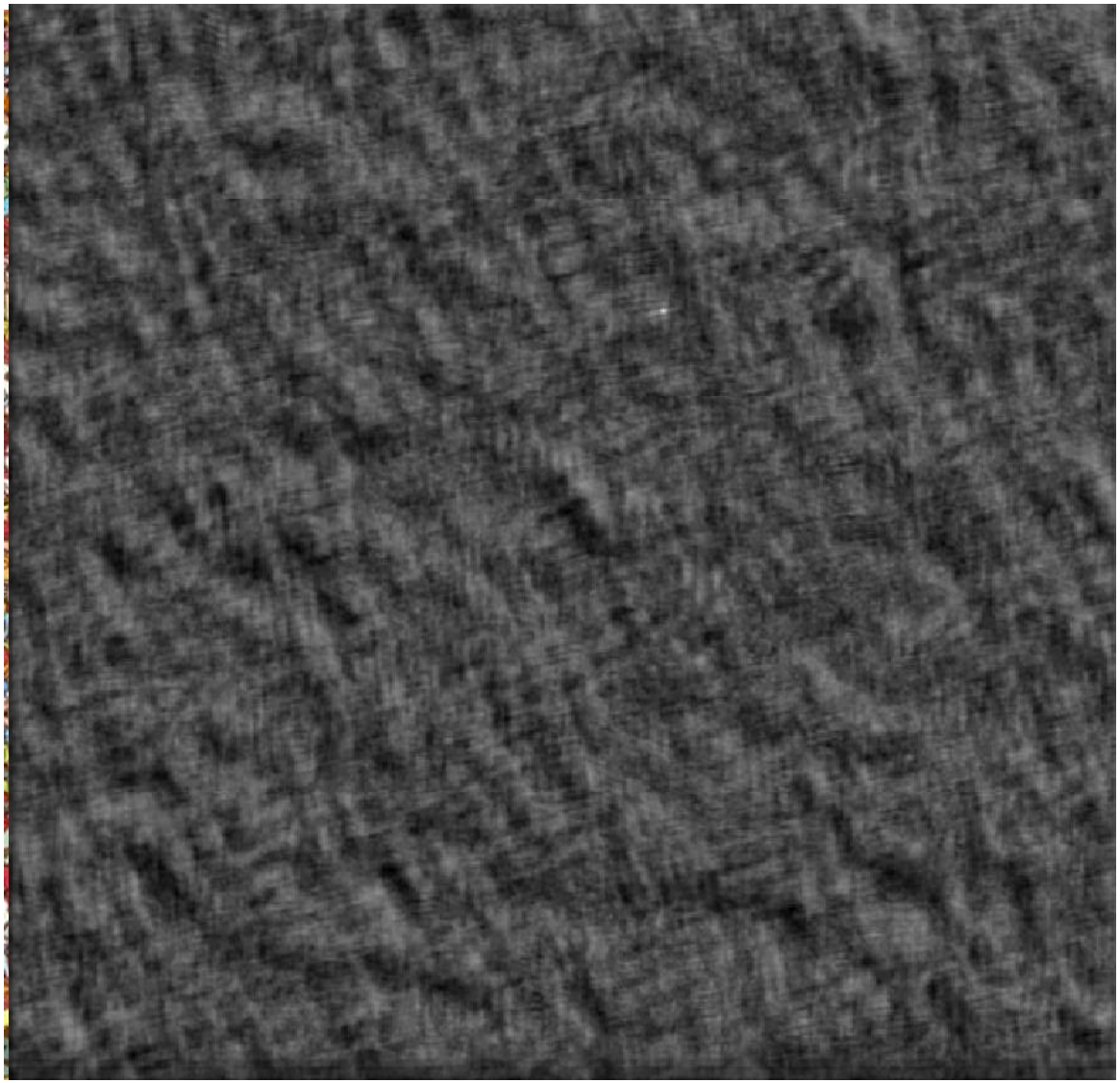
Where's Waldo?



Scene

Template

Where's Waldo?

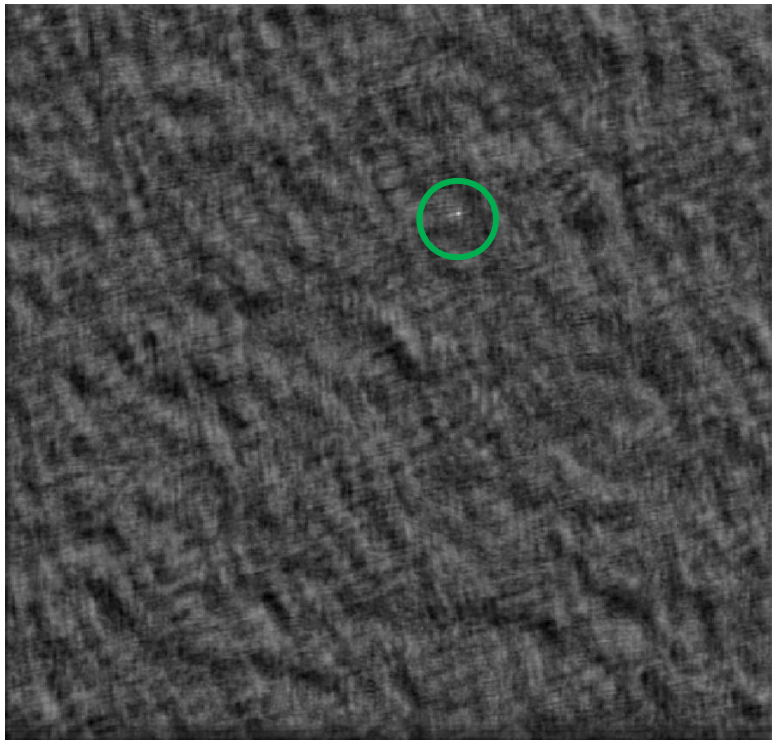


Scene

Template

Where's Waldo?

Detected template



Correlation map

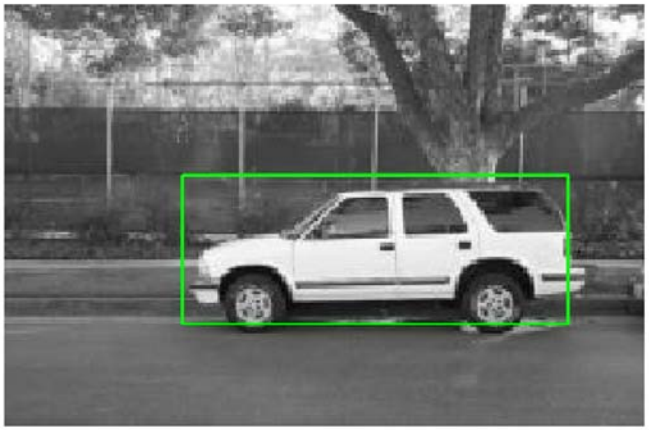
Template matching

Scene

Template

What if the template is not identical to some subimage in the scene?

Template matching



Detected template

Template

Match can be meaningful, if scale, orientation, and general appearance is right.

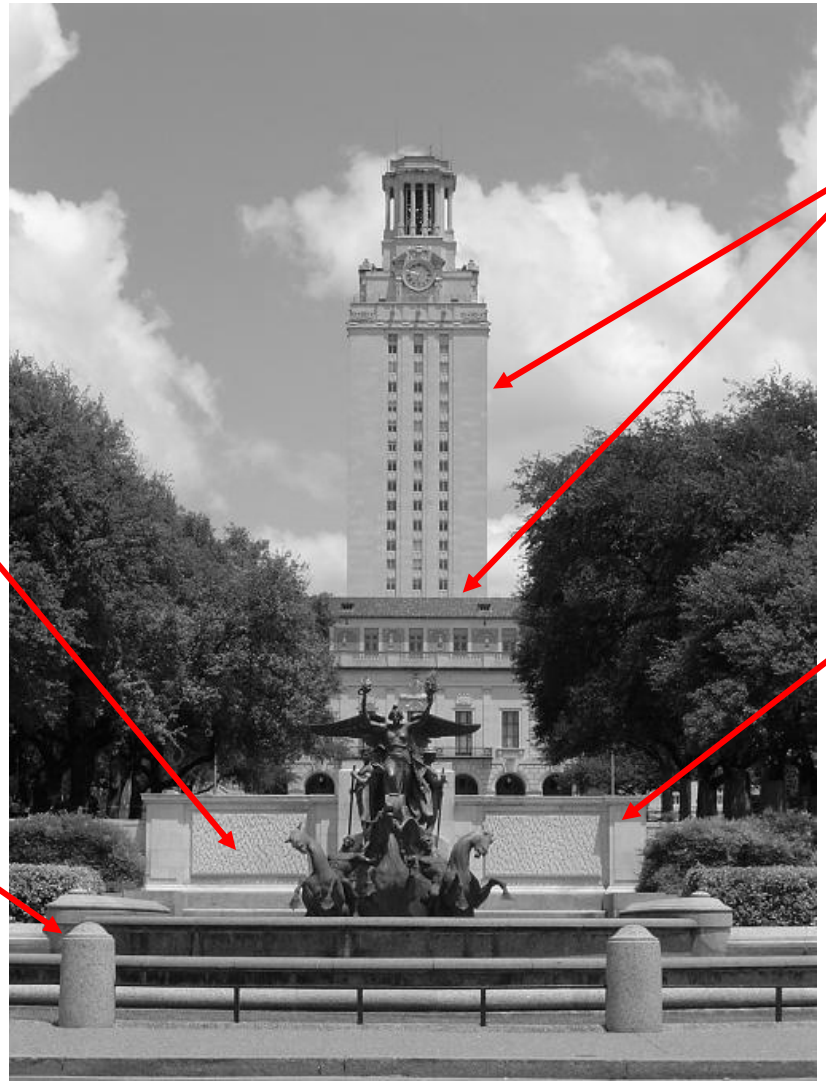
Edge detection

- **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 can cause an edge?



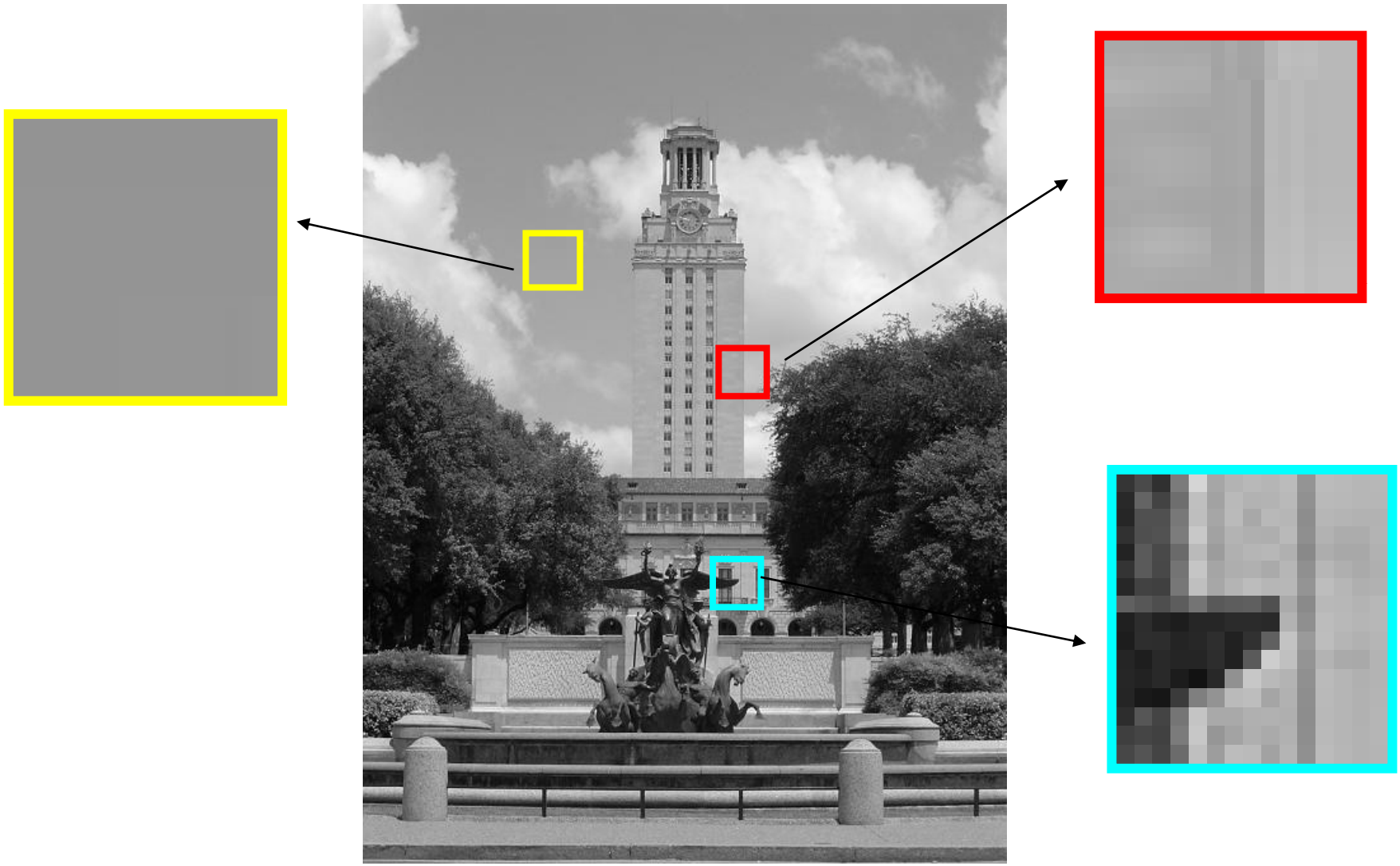
Reflectance change: appearance information, texture

Depth discontinuity: object boundary

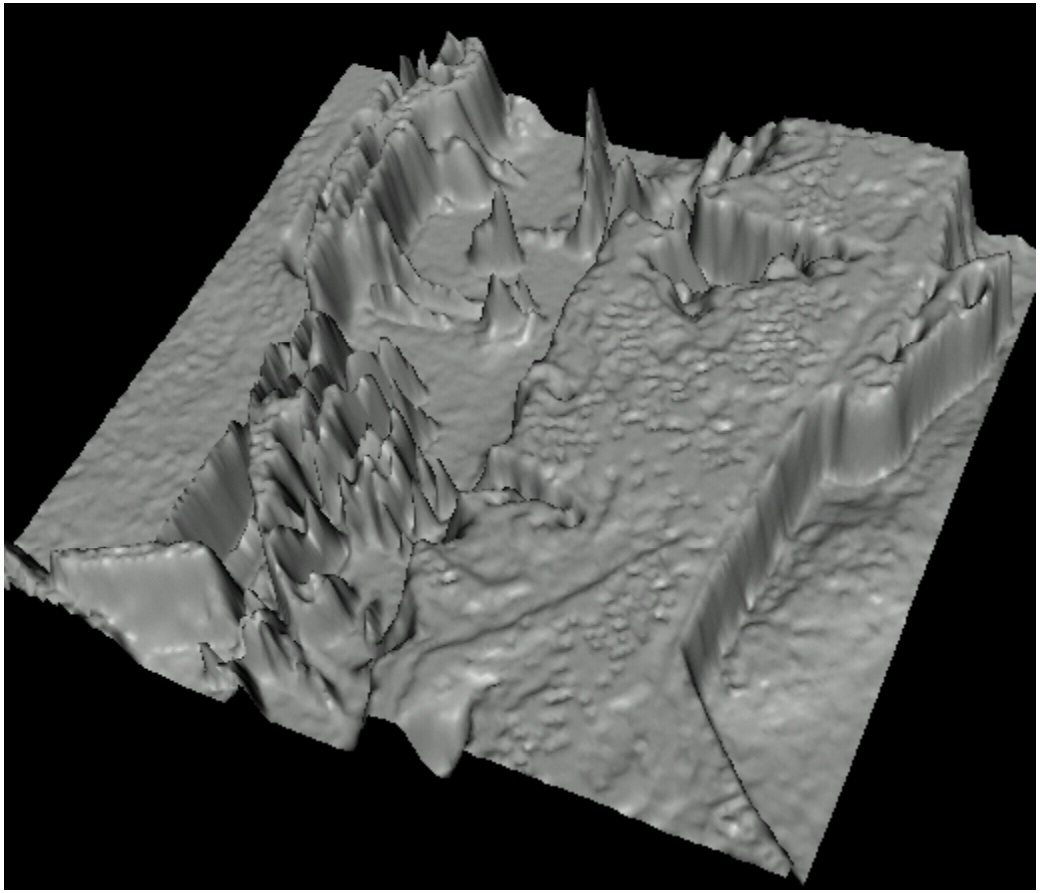
Cast shadows

Change in surface orientation: shape

Contrast and invariance



Recall : Images as functions

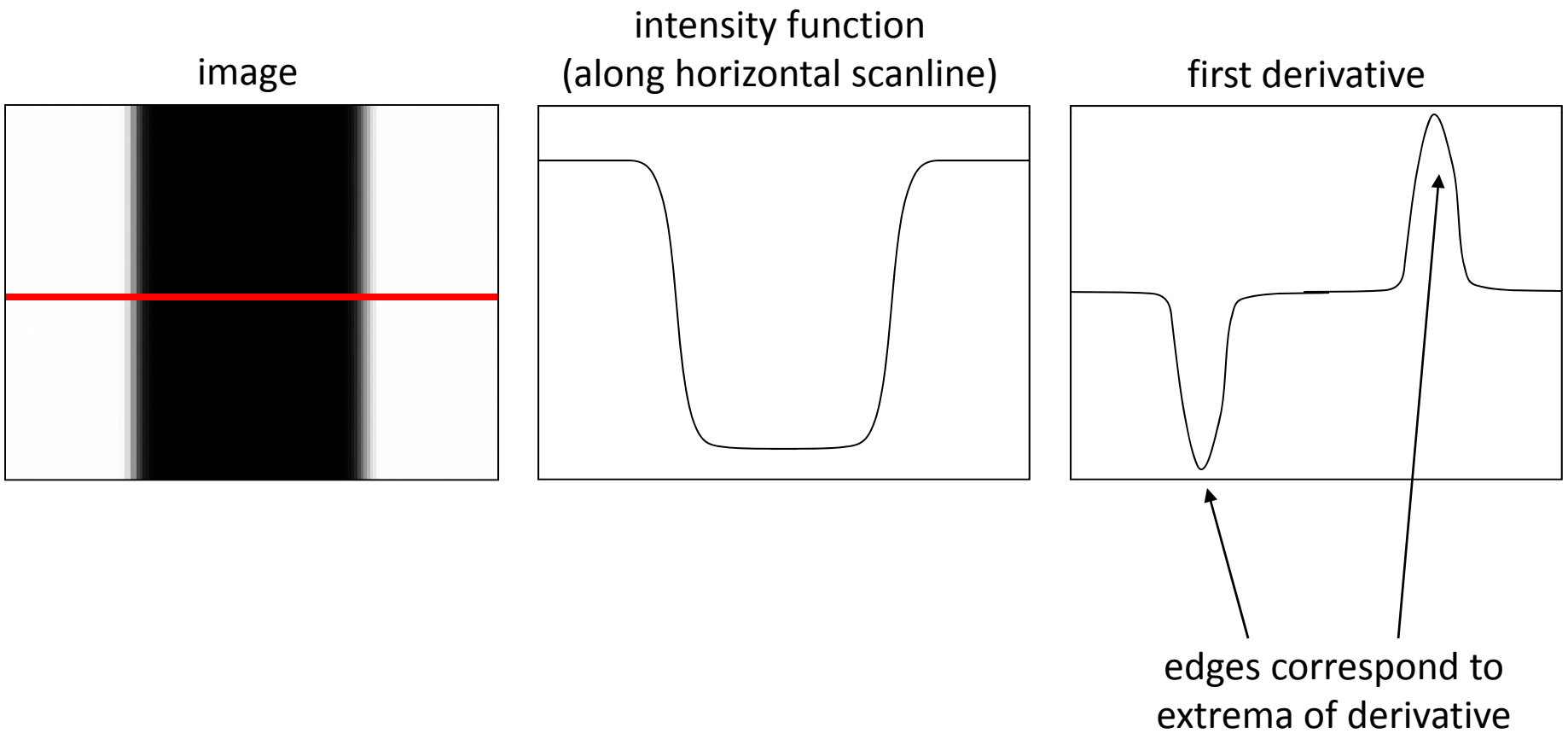


- Edges look like steep cliffs

Source: S. Seitz

Derivatives and edges

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



Source: L. Lazebnik

Differentiation and convolution

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

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

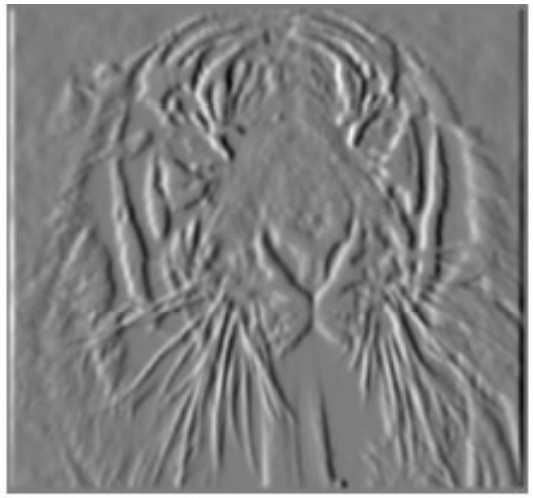
For discrete data, we can approximate using finite differences:

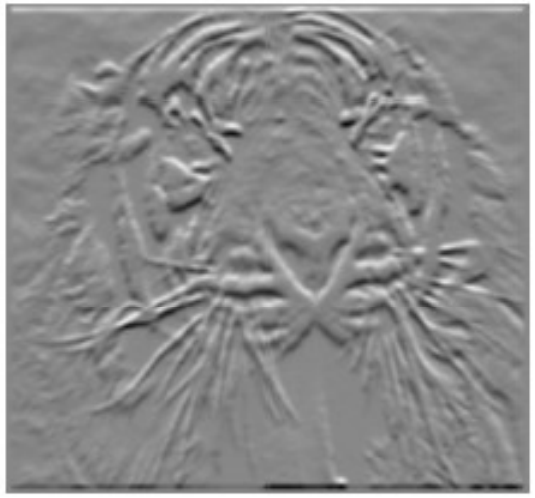
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

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

Partial derivatives of an image

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





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

-1	1

-1
1

?

or

1
-1

Which shows changes with respect to x?

(showing flipped filters)

Assorted finite difference filters

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;
```

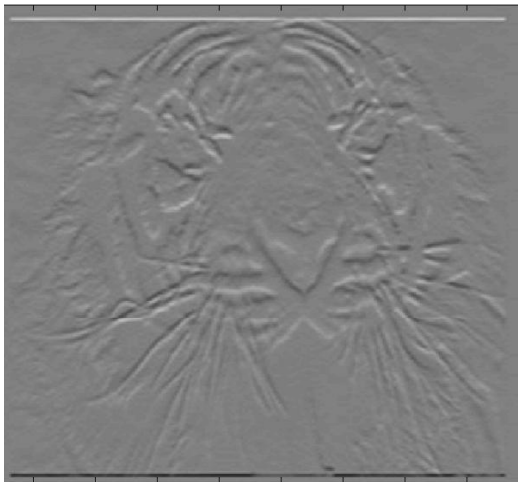
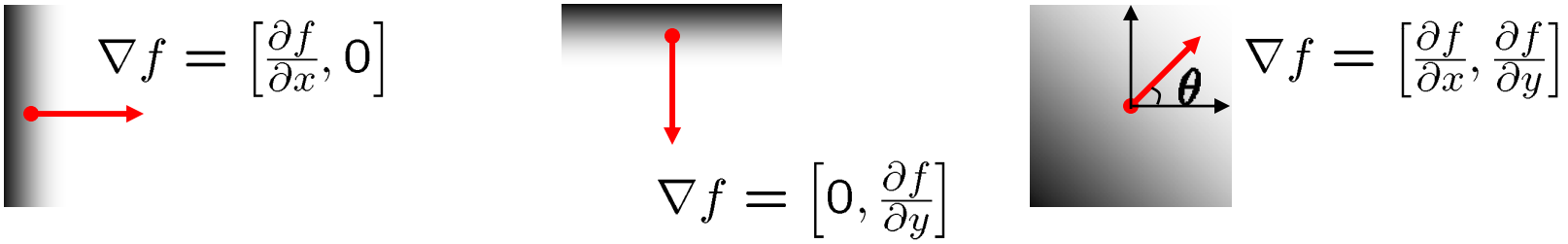


Image gradient

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

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

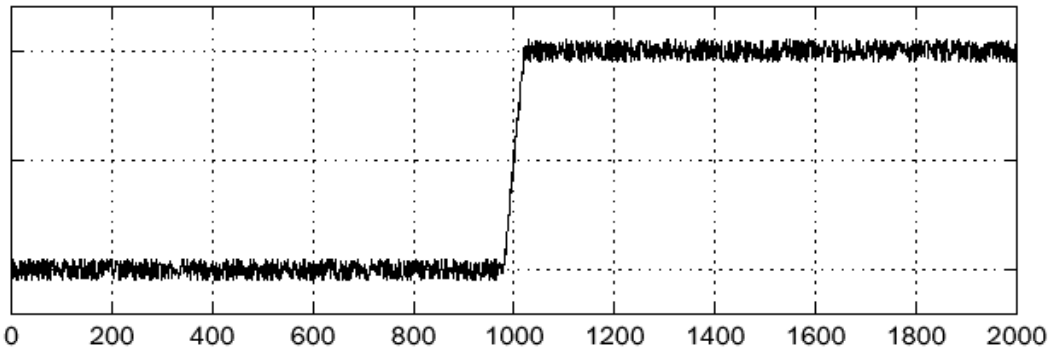
Slide credit S. Seitz

Effects of noise

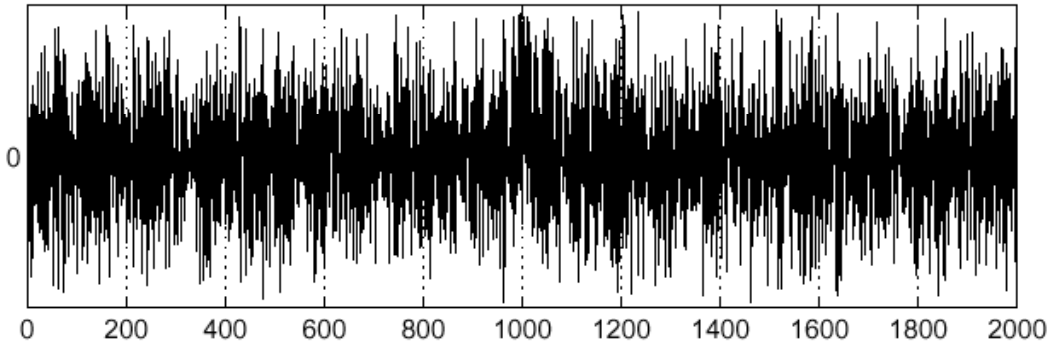
Consider a single row or column of the image

– Plotting intensity as a function of position gives a signal

$f(x)$



$\frac{d}{dx}f(x)$



Where is the edge?

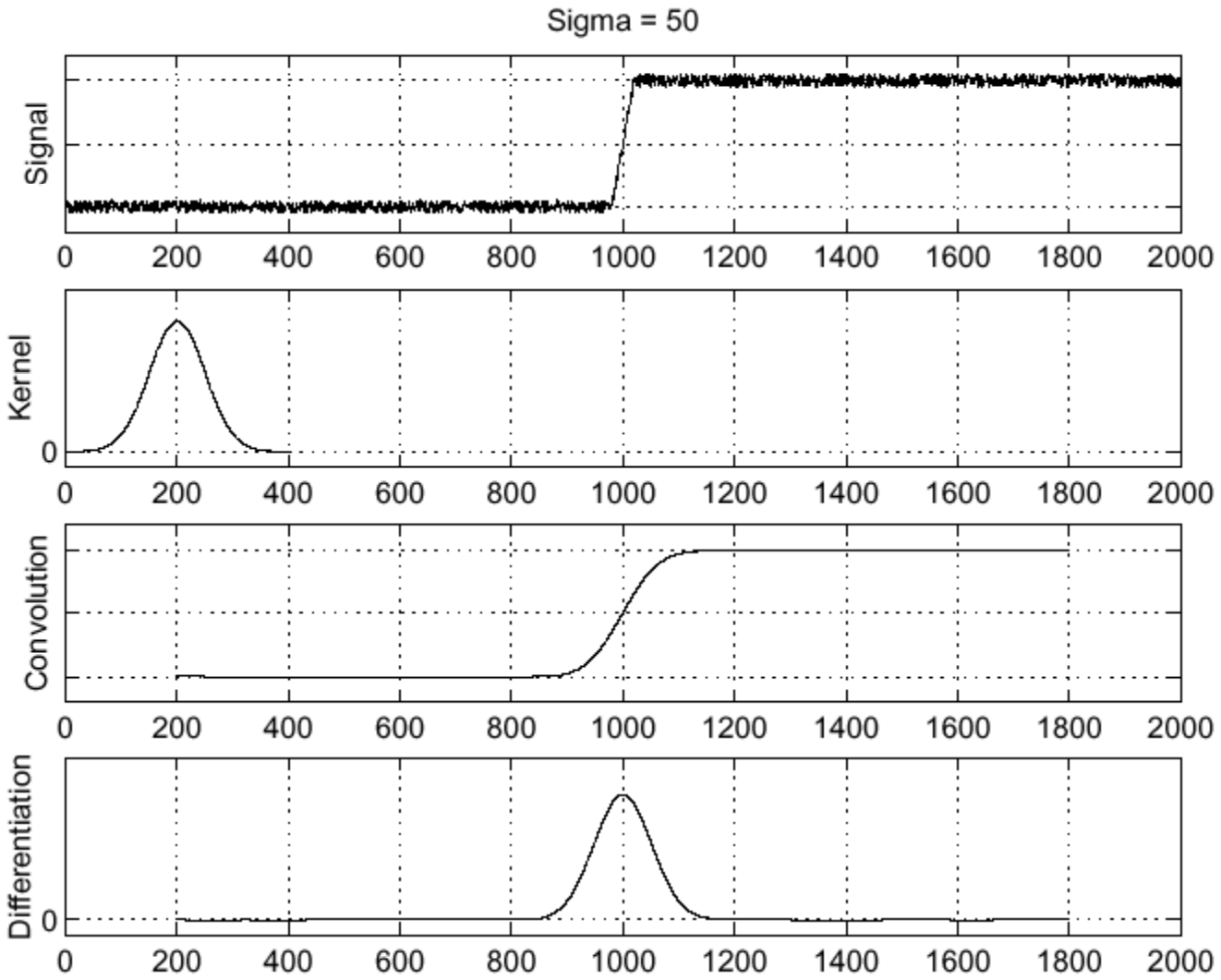
Solution: smooth first

f

h

$h \star f$

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



Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

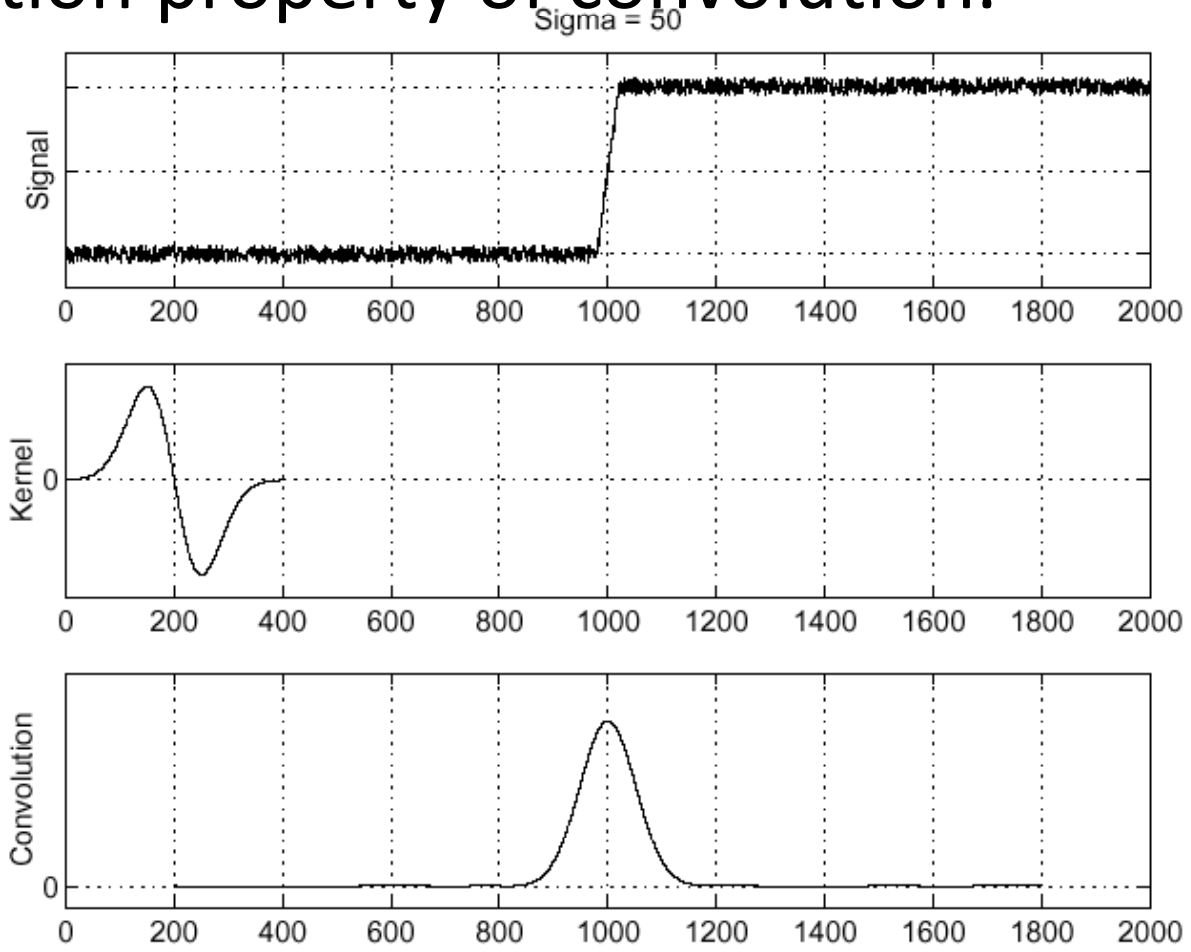
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Differentiation property of convolution.

f

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

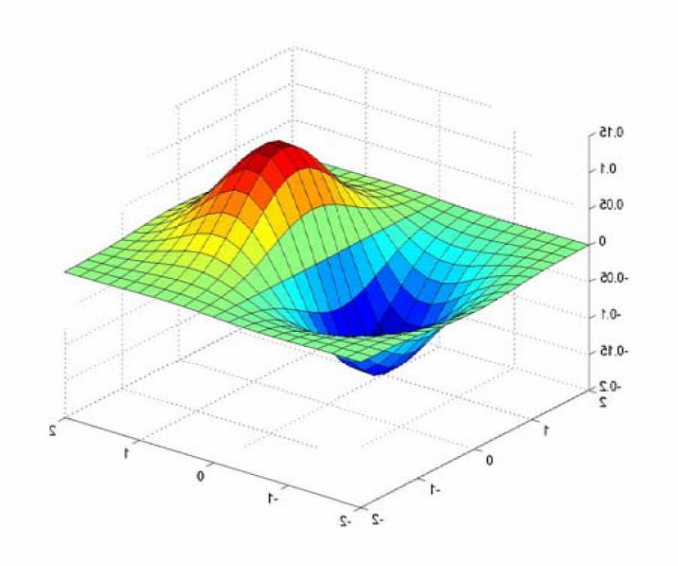
$(\frac{\partial}{\partial x}h) \star f$



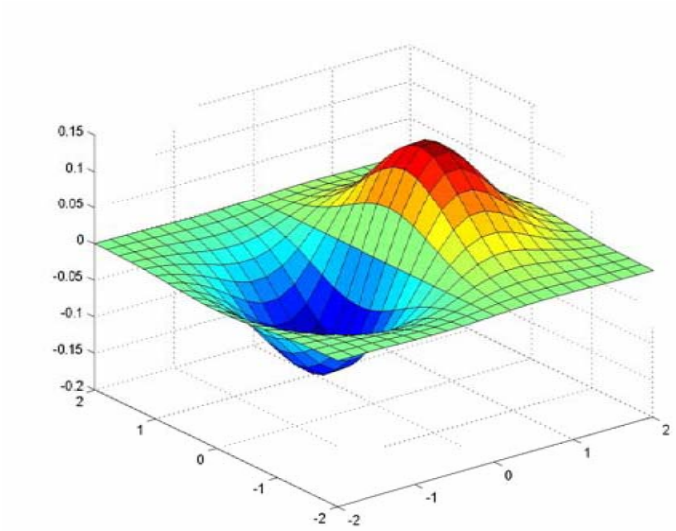
Derivative of Gaussian filter

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

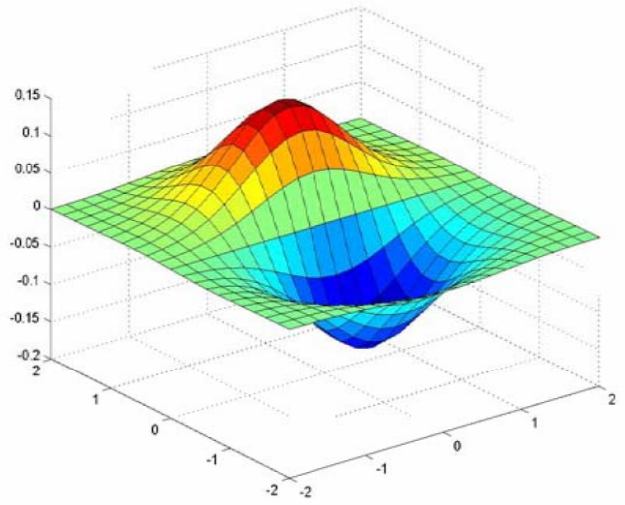
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \end{bmatrix}$$



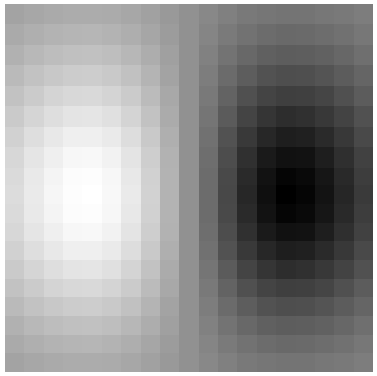
Derivative of Gaussian filters

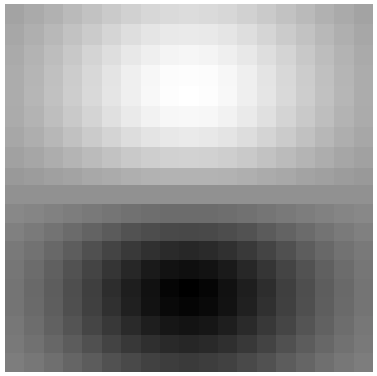


x-direction



y-direction



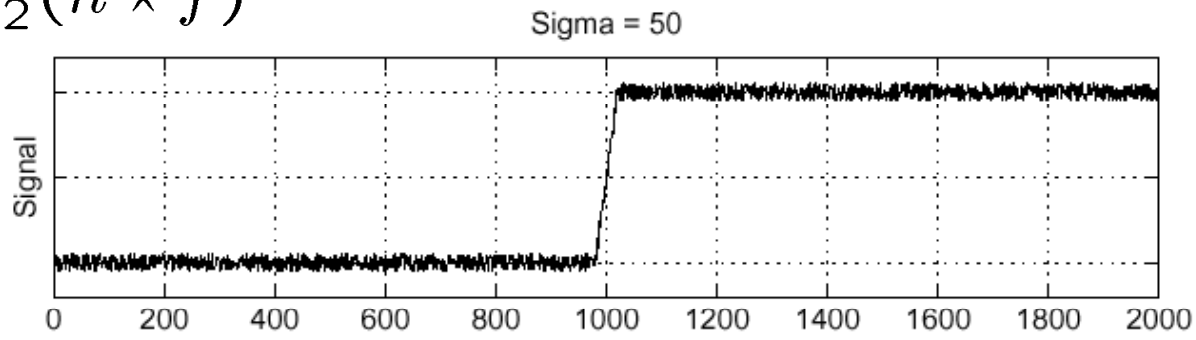


Source: L. Lazebnik

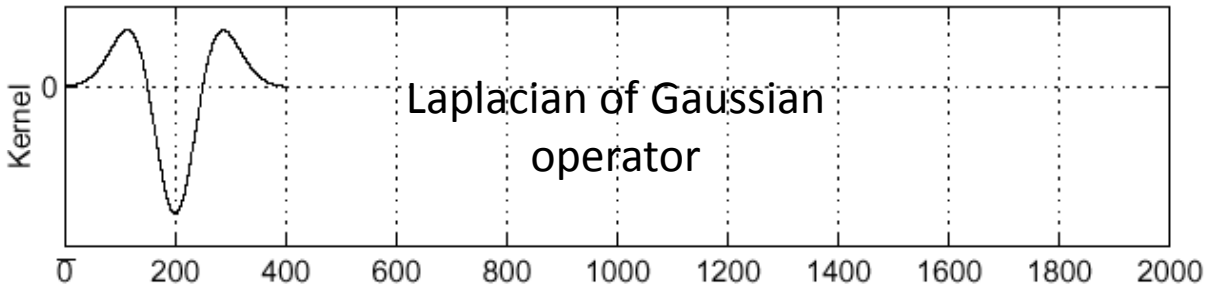
Laplacian of Gaussian

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

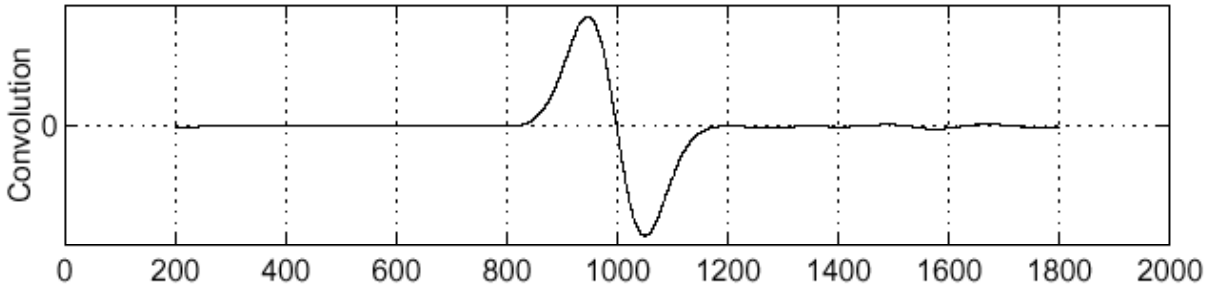
f



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



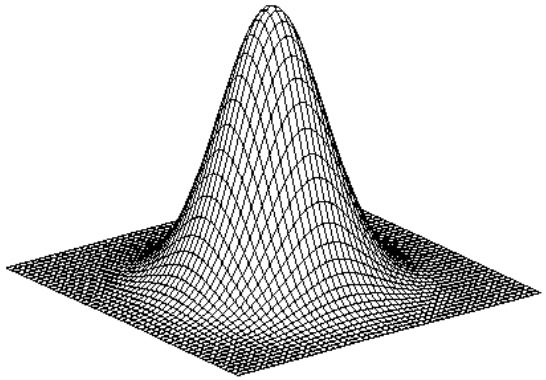
$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge?

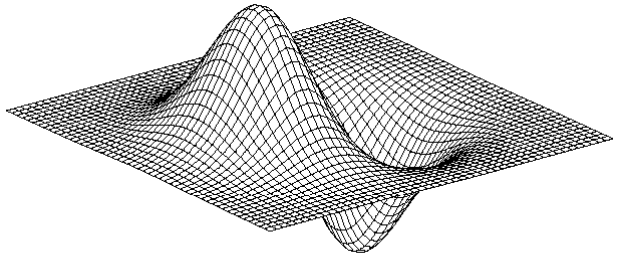
Zero-crossings of bottom graph

2D edge detection filters



Gaussian

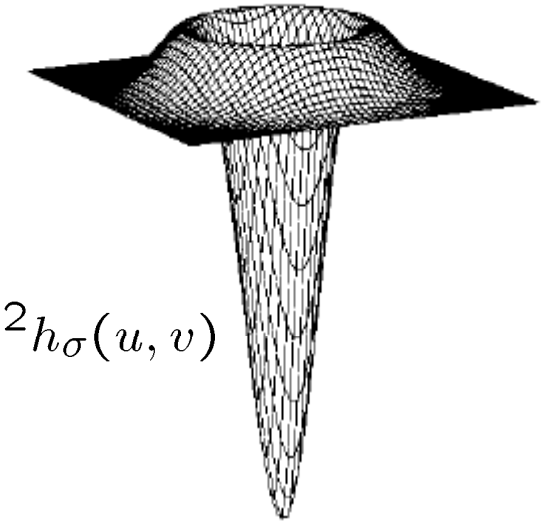
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



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

- $\nabla^2$is the Laplacian operator:

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

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
- Filters act as templates
  - Highest response for regions that “look the most like the filter”
  - Dot product as correlation

Gradients -> edges

Primary edge detection steps:

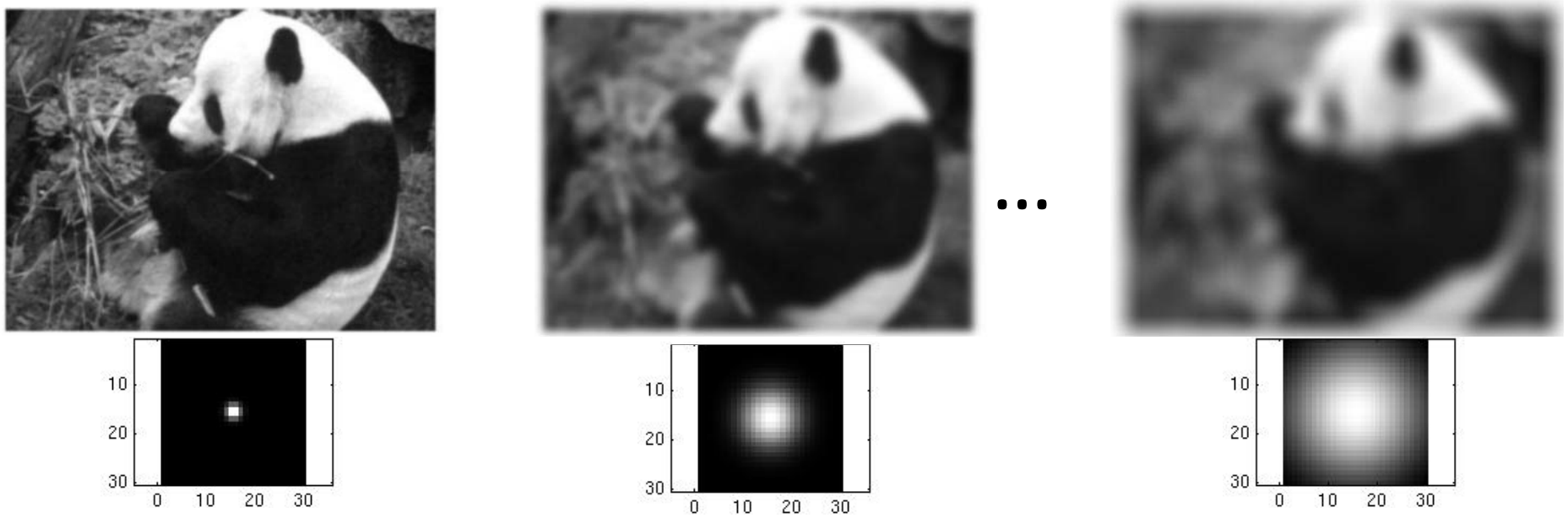
1. Smoothing: suppress noise
2. Edge enhancement: filter for contrast
3. Edge localization

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

   - Threshold, Thin

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

σ = 1 pixel

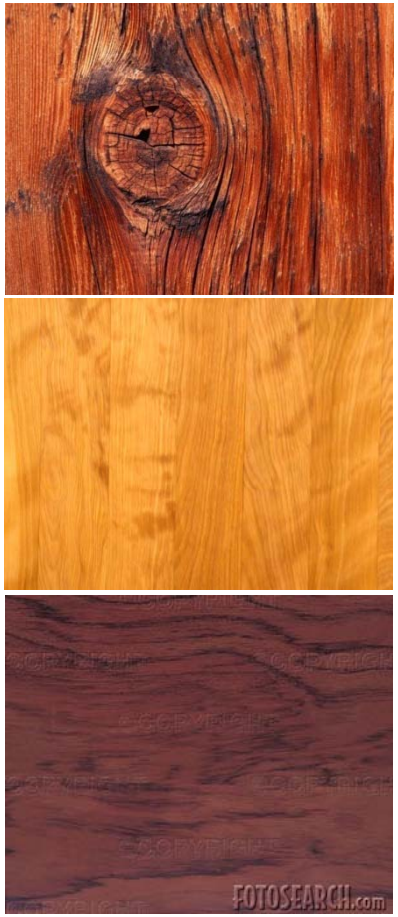
σ = 3 pixels

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

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

So, what scale to choose?

It depends what we're looking for.



Too fine of a scale…can't see the forest for the trees.

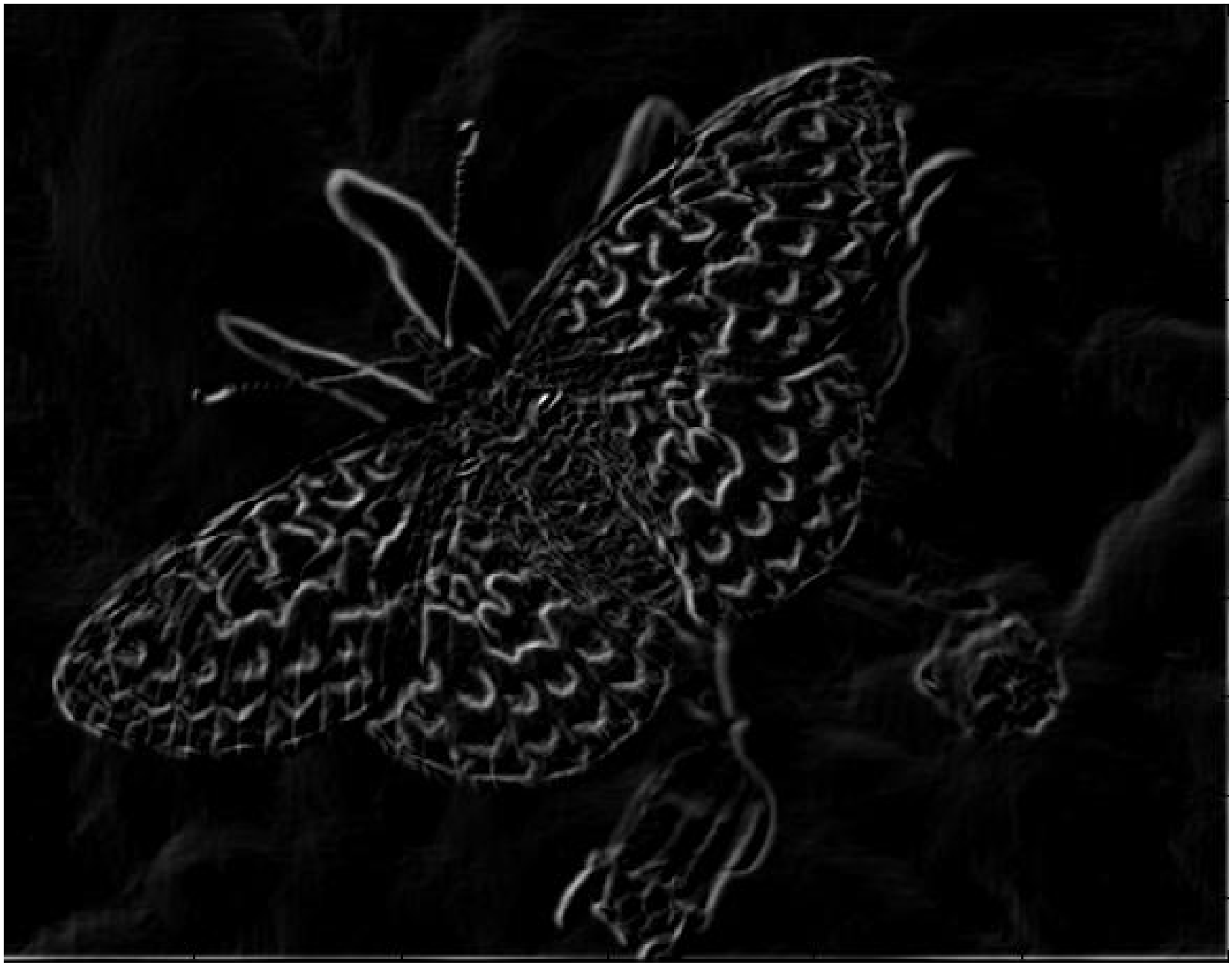
Too coarse of a scale…can't tell the maple grain from the cherry.

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 multi-pixel 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)

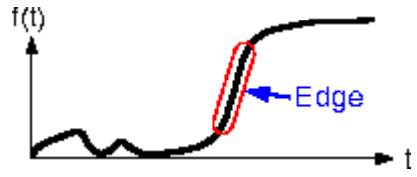
The Canny edge detector

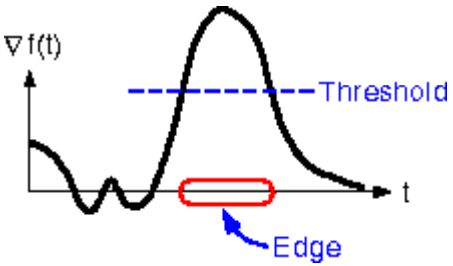
norm of the gradient

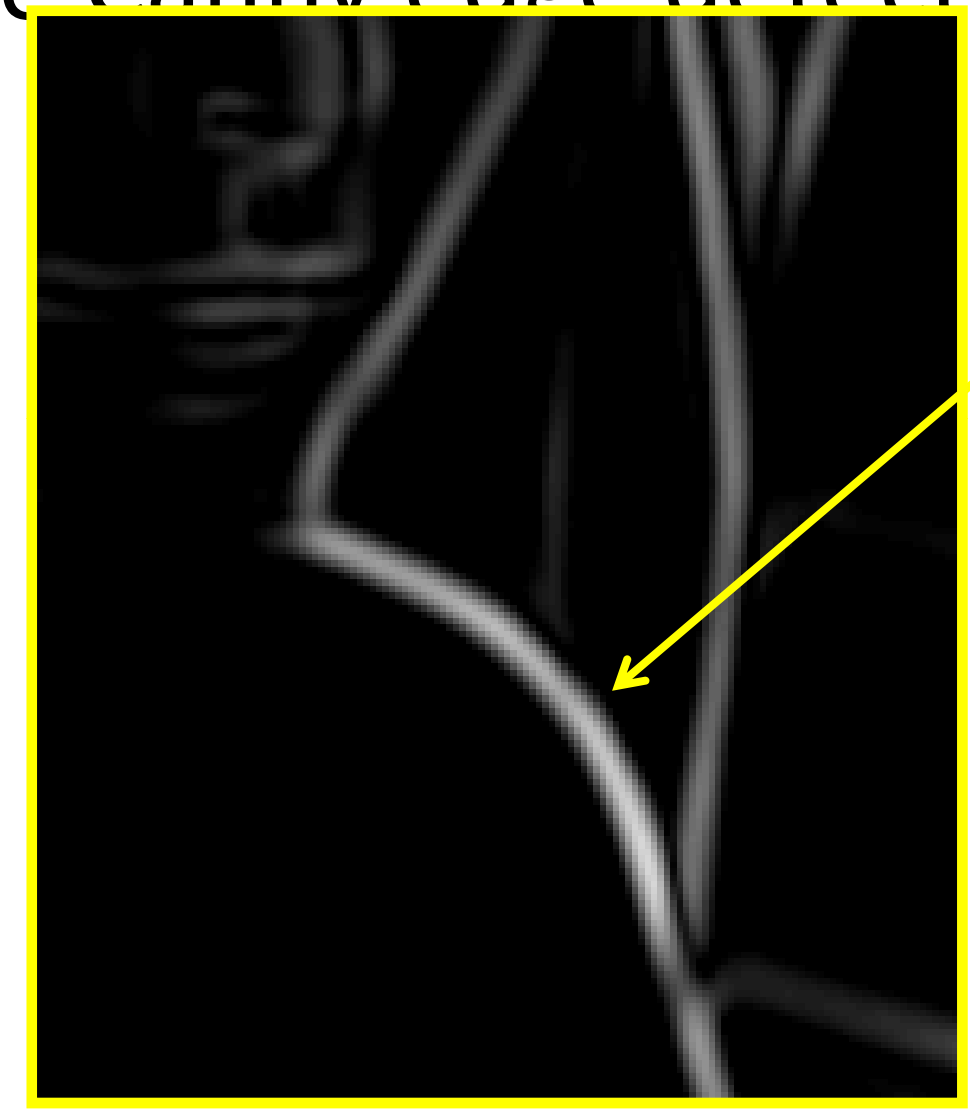
The Canny edge detector

thresholding

The Canny edge detector

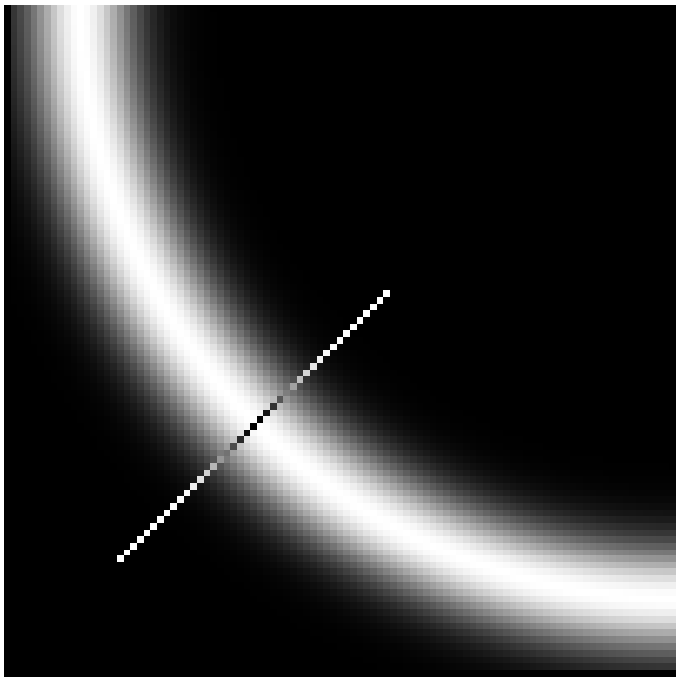


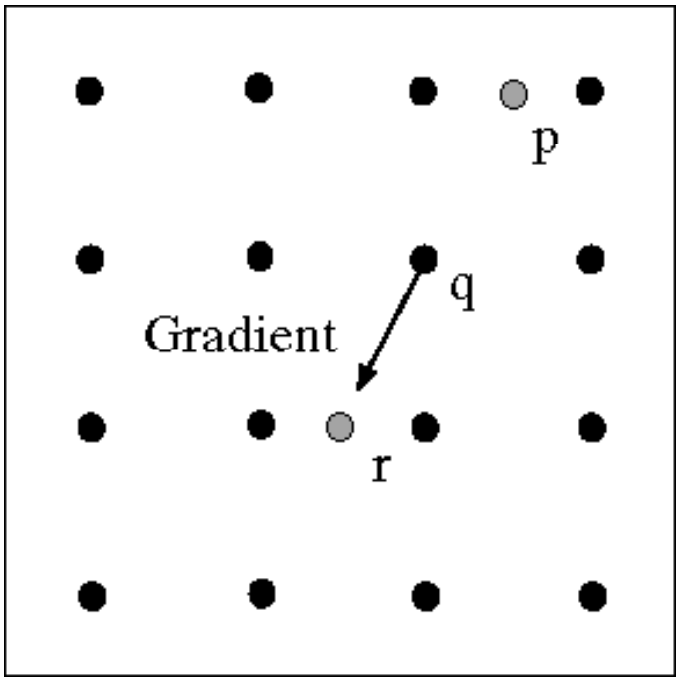




How to turn these thick regions of the gradient into curves?

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

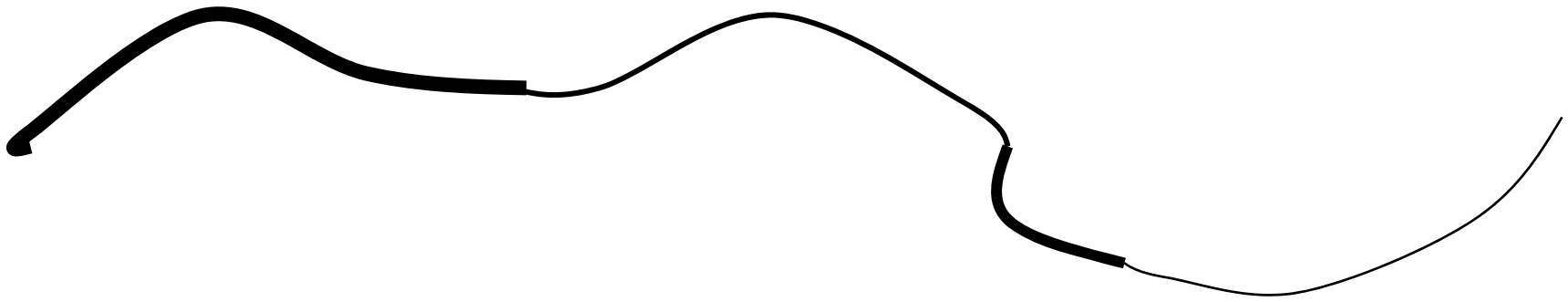
Problem: pixels along this edge didn't survive the thresholding

thinning

(non-maximum suppression)

Hysteresis thresholding

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



Source: S. Seitz

Hysteresis thresholding

original image

high threshold
(strong edges)

low threshold
(weak edges)

hysteresis threshold

Source: L. Fei-Fei

Object boundaries vs. edges

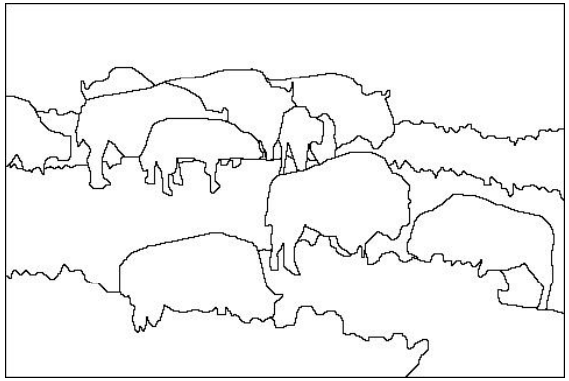
Background

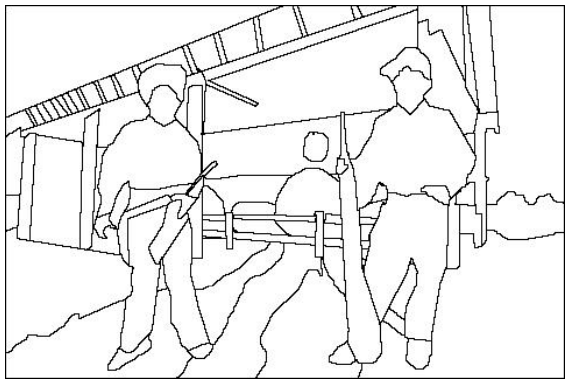
Texture

Shadows

Edge detection is just the beginning…

image | human segmentation | gradient magnitude




Berkeley segmentation database:
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

Much more on segmentation later in term…

Source: L. Lazebnik

Slide Credits

- Kristen Grauman: all
- and Seitz, Marschner, Lazebnik and others, as noted…

Summary

- Filters allow local image neighborhood to influence our description and features
  - Smoothing to reduce noise
  - Derivatives to locate contrast, gradient
- Filters have highest response on neighborhoods that “look like” it; can be thought of as template matching.
- Convolution properties will influence the efficiency with which we can process images.
  - Associative
  - Filter separability
- Edge detection processes the image gradient to find curves, or chains of edgels.

Next time

- Sampling
- 2D Fourier Transform
- Pyramids…

Adminstrivia, revisited…

- bSpace returned…let's keep our fingers crossed…
- box.net mirror:
  - http://www.box.net/shared/9q8cdvqrco
- "external homepage"
  - http://www.eecs.berkeley.edu/~trevor/CS280.html
  - (has lecture notes, but no assignments, etc.)
- few assignments from (recently un-)WL'ed folks?
  - please do **upload via bSpace** (but keep a backup!)
- Thu 5pm-6pm office hour canceled this week.
- Pset 1 released; n.b. ***longer*** than usual, start early
- Pset 0 solutions on bSpace resource page