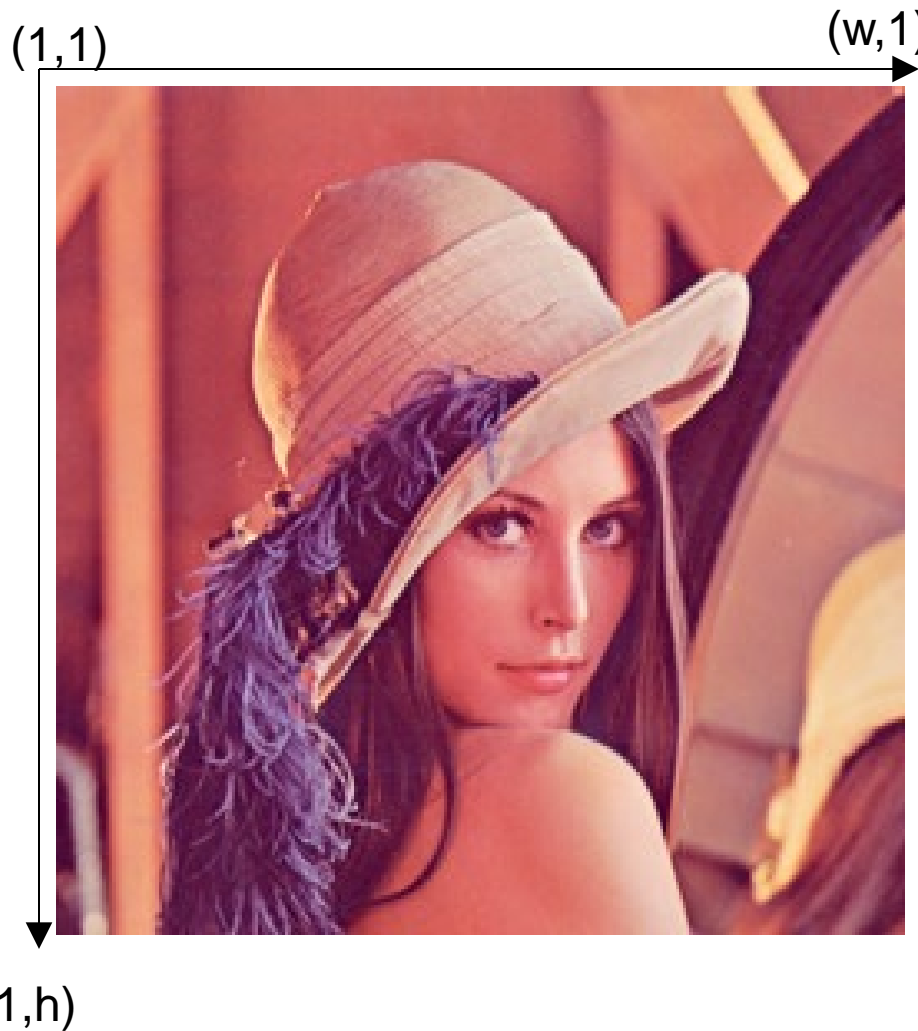


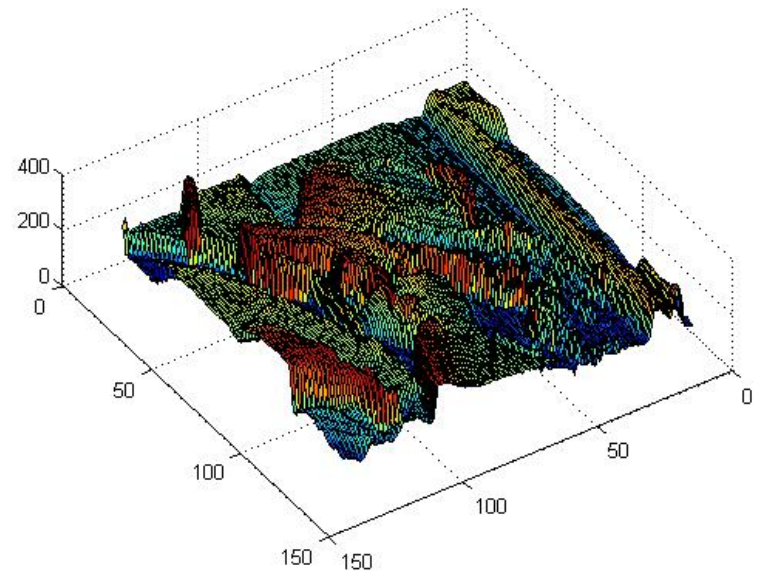
(Compressed) Intro to Digital Image Processing

Slides adapted from D.A. Forsyth, Zhou Wang, Lana Lazebnik, Li Fei-Fei, Kristen Grauman, D. Lowe, and Steve Seitz

Digital Images

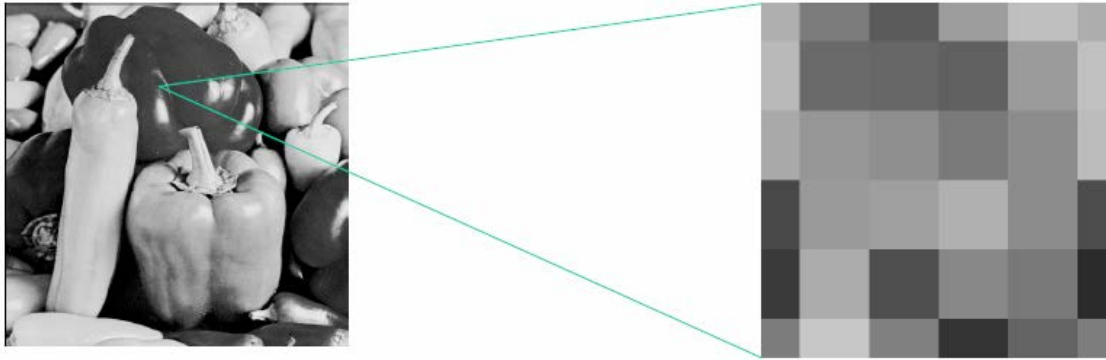


- 2 models of digital images:
 - 2D matrix of intensity values
 - Function
 - Input: 2D location
 - Output: Intensity value



Imaging Noise

- Cameras are not perfect sensors
- Scenes are not represented in the way you may expect



some noise





more noise

i.i.d. Noise

- Simplest noise model
 - independent stationary additive Gaussian noise
 - the noise value at each pixel is given by an independent draw from the same normal probability distribution
- Issues
 - this model allows noise values that could be greater than maximum camera output or less than zero
 - for small standard deviations, this isn't too much of a problem - it's a fairly good model
 - independence may not be justified (e.g. damage to lens)
 - may not be stationary (e.g. thermal gradients in the ccd)

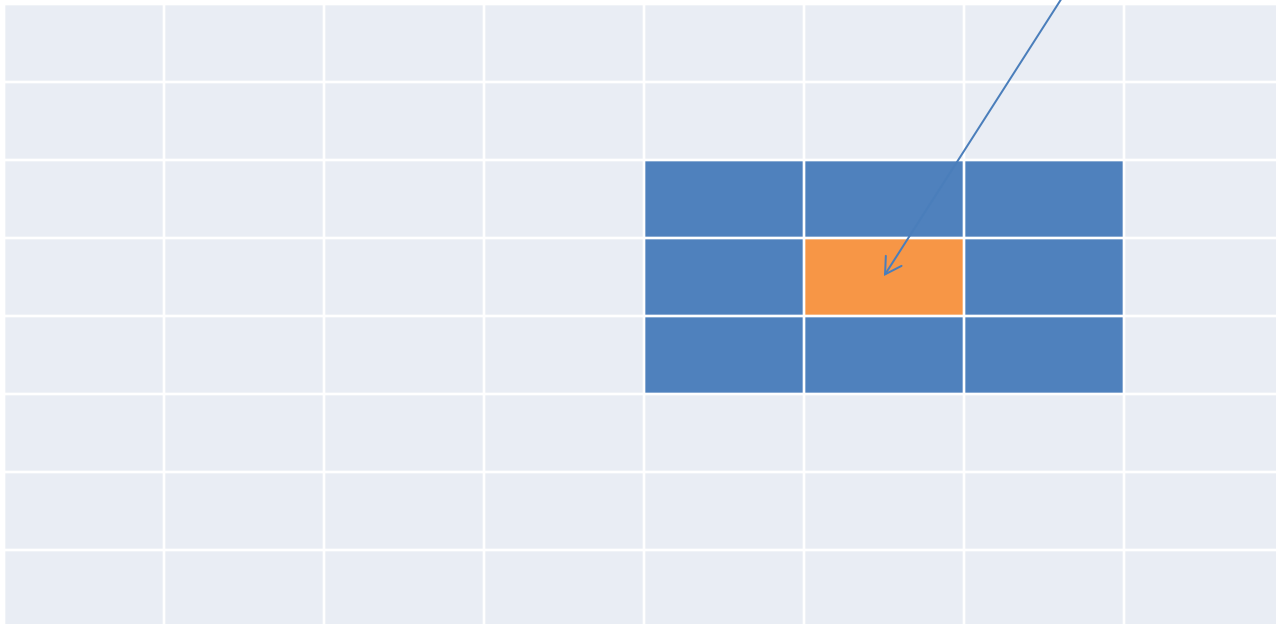
Smoothing (Averaging) reduces noise

- Generally expect pixels to “be like” their neighbors
 - surfaces turn slowly
 - relatively few reflectance changes
- Generally expect noise processes to be independent from pixel to pixel
- Implies that smoothing suppresses noise, for appropriate noise models
- Scale
 - as this parameter goes up, more pixels are involved in the average
 - and the image gets more blurred
 - and noise is more effectively suppressed

Averaging Pixel Values

$$I'(i, j) = 1/9 \sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} I(i', j')$$

pixel (i,j)



Linear Filters

- General process:
 - Form new image whose pixels are a weighted sum of original pixel values, using the same weights at each point.
- Properties
 - Output is a linear function of the input
 - Output is a shift-invariant function of the input (i.e. shift the input image two pixels to the left, the output is shifted two pixels to the left)
- Examples:
 - Smoothing by averaging
 - form the average of pixels in a neighborhood
 - Smoothing with a Gaussian
 - form a weighted average of pixels in a neighborhood

Convolution

- Represent these weights as an image, H
- H is usually called the **kernel, filter, or point spread function**
- Operation is called **convolution**
 - it's associative, so it doesn't really matter which is the *image* and which is *kernel*.

■ Result is:

$$R_{ij} = \sum_{u,v} H_{i-u, j-v} F_{uv}$$

Convolution on images

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Input

1	2	0	1	3	
2	1	4	2	2	
1	0	1	0	1	
1	2	1	0	2	
2	5	3	1	2	

Output

	$\frac{12}{9}$				

Convolution on images

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Input

1	2	0	1	3	
2	1	4	2	2	
1	0	1	0	1	
1	2	1	0	2	
2	5	3	1	2	

Output

		$\frac{12}{9}$	$\frac{11}{9}$		

Convolution on images

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Input

1	2	0	1	3	
2	1	4	2	2	
1	0	1	0	1	
1	2	1	0	2	
2	5	3	1	2	

Output

	$\frac{12}{9}$	$\frac{11}{9}$	$\frac{14}{9}$		

Convolution on images

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Input

1	2	0	1	3	
2	1	4	2	2	
1	0	1	0	1	
1	2	1	0	2	
2	5	3	1	2	

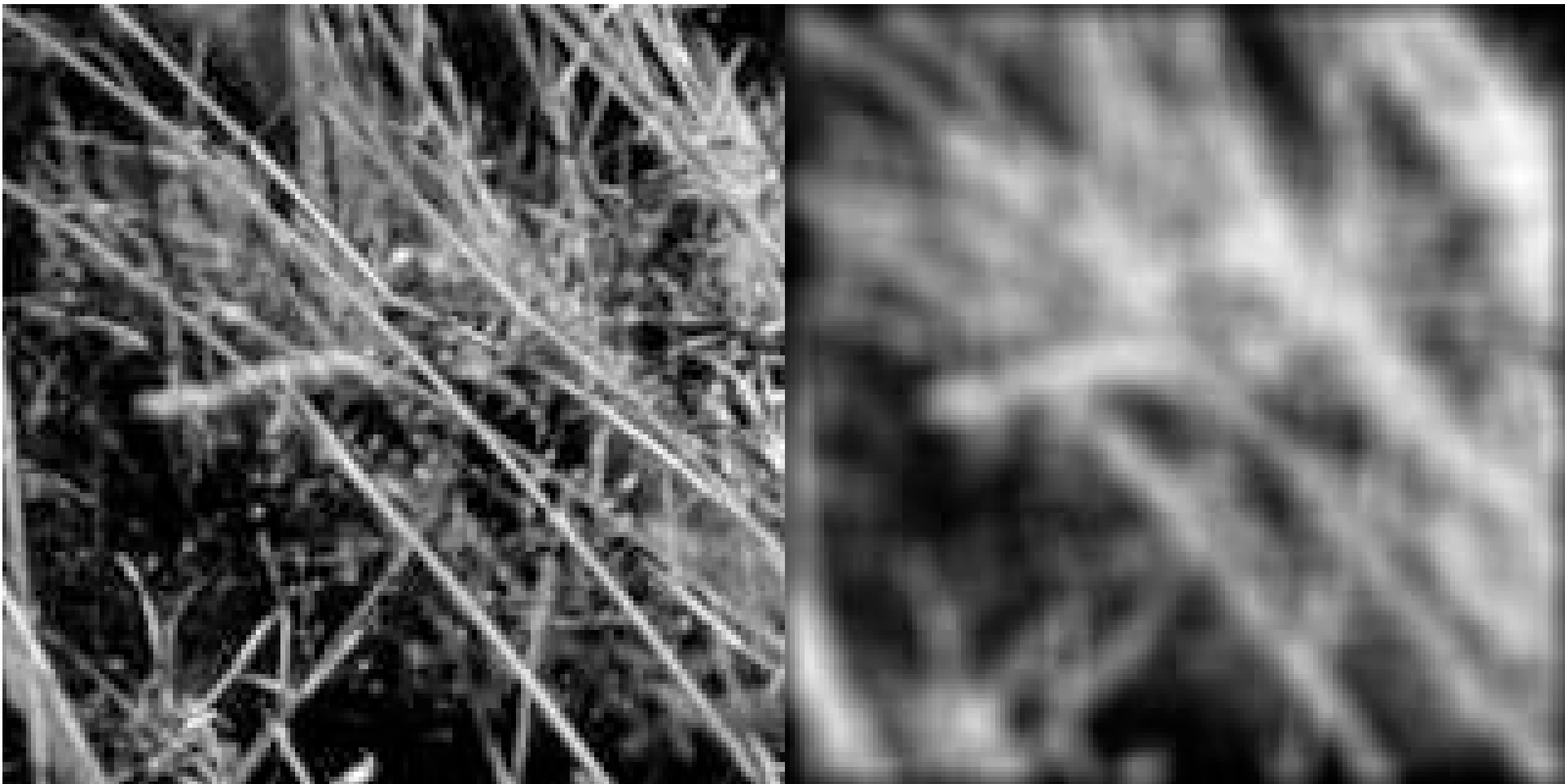
Output

	$\frac{12}{9}$	$\frac{11}{9}$	$\frac{14}{9}$		
	$\frac{13}{9}$	$\frac{11}{9}$	$\frac{13}{9}$		
	$\frac{16}{9}$	$\frac{13}{9}$	$\frac{11}{9}$		

Notes on Image Convolution

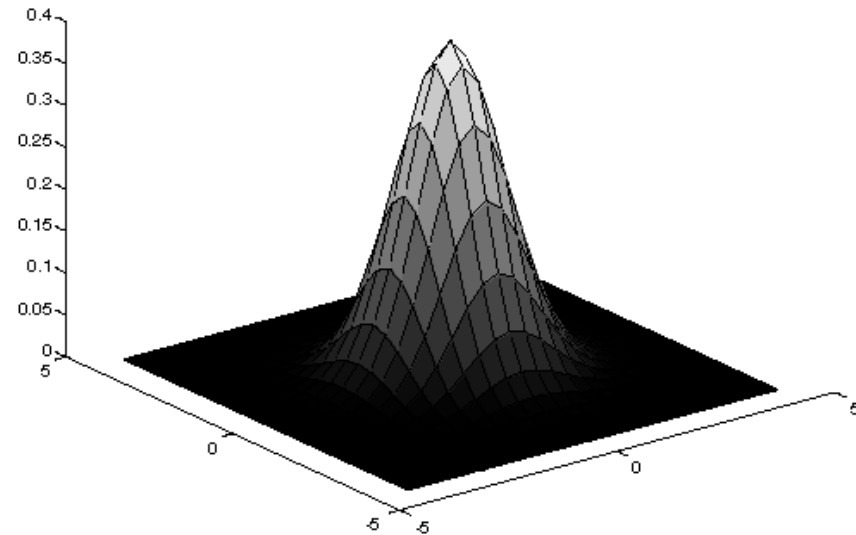
- Handling Borders
 - The filter can “fall off” the edge of the image
 - 3 main approaches:
 - Pad the image w/ extra 0's so that result is same size as image
 - Pad the image w/ border values so that result is same size as image
 - Don't pad the image, result will be smaller than original image
- Time complexity?
 - Image is $n * n$, Kernel is $m * m$
 - Time complexity is $O(n^2m^2)$

Example: Smoothing by Averaging



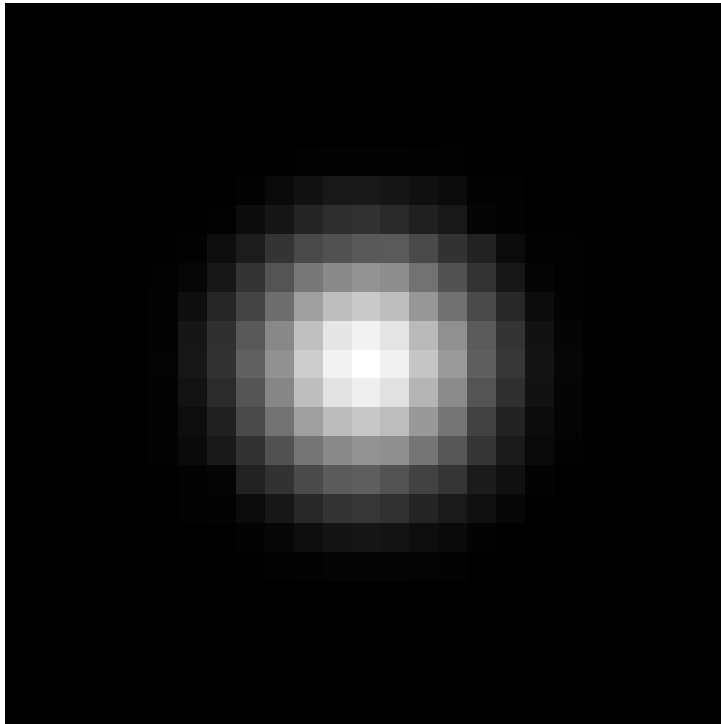
Smoothing with a Gaussian

- Smoothing with an average actually doesn't compare at all well with a defocussed lens
 - Most obvious difference is that a single point of light viewed in a defocussed lens looks like a fuzzy blob; but the averaging process would give a little square.



- A Gaussian gives a good model of a fuzzy blob

An Isotropic Gaussian

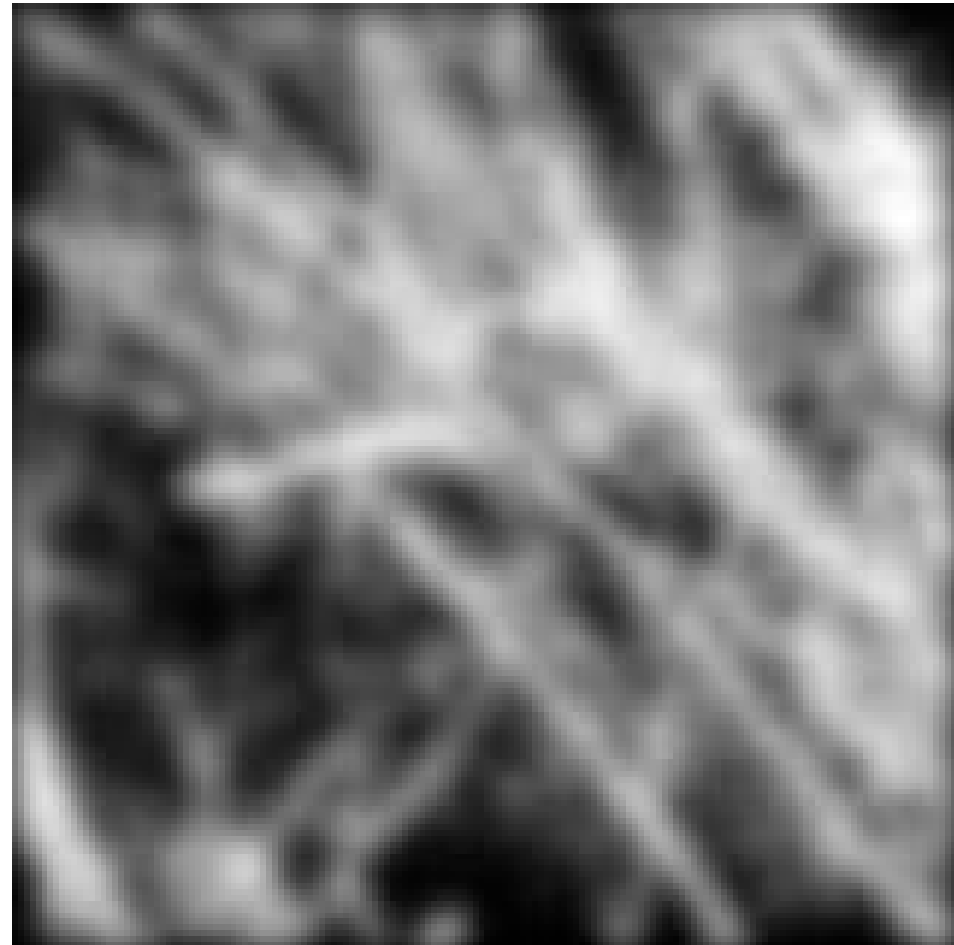


- The picture shows a smoothing kernel proportional to

$$\exp\left(-\left(\frac{x^2 + y^2}{2\sigma^2}\right)\right)$$

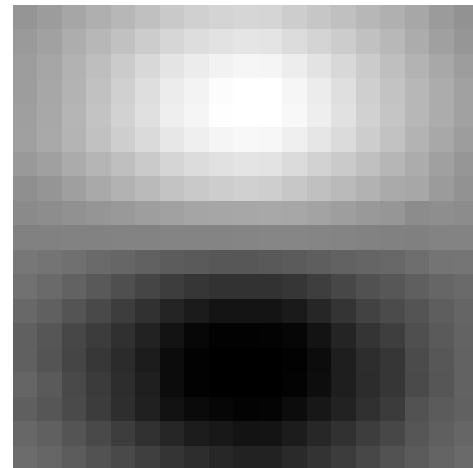
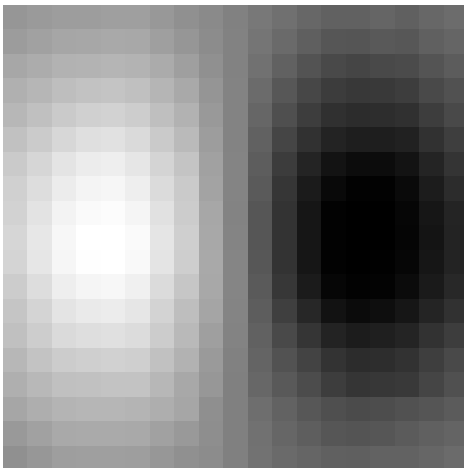
(which is a reasonable model of a circularly symmetric fuzzy blob)

Smoothing with a Gaussian



Linear Filters are Templates

- Applying a filter at some point can be seen as taking a dot-product between the image and some vector
- Filtering the image is a set of dot products
- Insight
 - filters look like the effects they are intended to find
 - filters find effects they look like



Linear Filters Summary

- A linear filter is a kernel (which can be represented as an image) which can be used for convolution
- Result is a new image where each pixel is a weighted combination of the original pixels

$$p'_i = c_1 * p_1 + c_2 * p_2 + \dots$$

- Anything else is a *nonlinear filter*

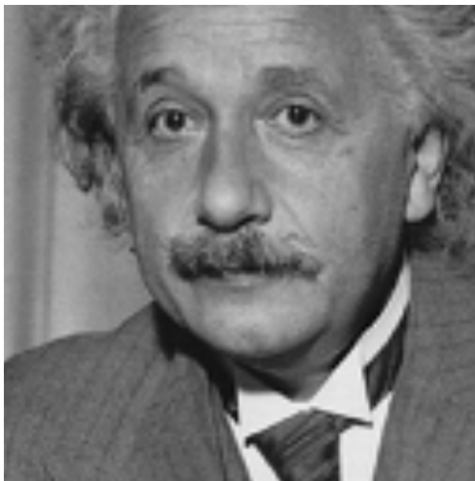
Salt & Pepper Noise

- **Definition**

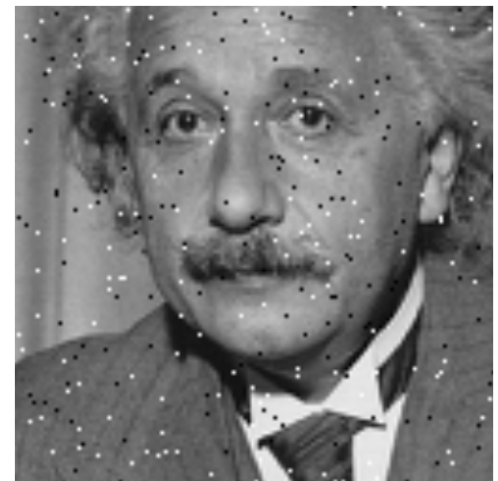
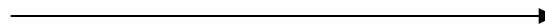
- Each pixel in an image has a probability p_a or p_b of being contaminated by a white dot (salt) or a black dot (pepper)

X : noise-free image, Y : noisy image

with probability p_a ————— noisy pixels
with probability p_b —————
with probability $1 - p_a - p_b$ — clean pixels



add salt & pepper noise



Median Filter

- Order Statistics (OS)
 - Given a set of numbers

Denote the OS as

$$\mathbf{x} = \{x_1, x_2, \dots, x_{2M+1}\}$$

$$\mathbf{x}_{OS} = \{x_{(1)}, x_{(2)}, \dots, x_{(2M+1)}\}$$

such that

$$x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(M+1)} \leq \dots \leq x_{(2M+1)}$$

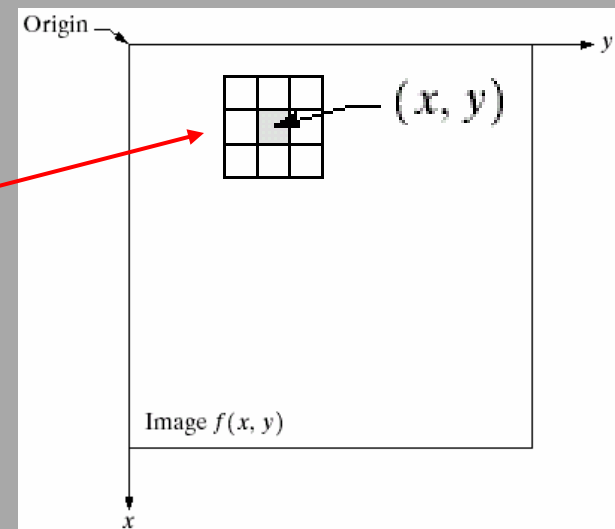
min value

max
value

middle value

$$\text{Median}\{x_1, x_2, \dots, x_{2M+1}\} = x_{(M+1)}$$

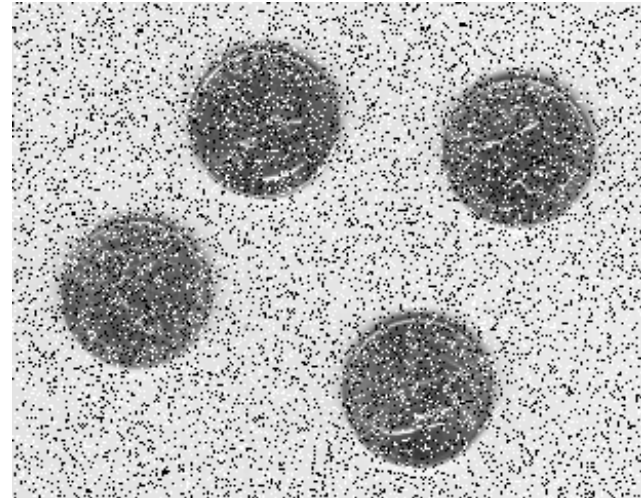
- Applying Median Filters to Images
 - Use sliding windows (similar to spatial linear filters)
 - Typical windows: 3x3, 5x5, 7x7, other shapes



Median Filtering



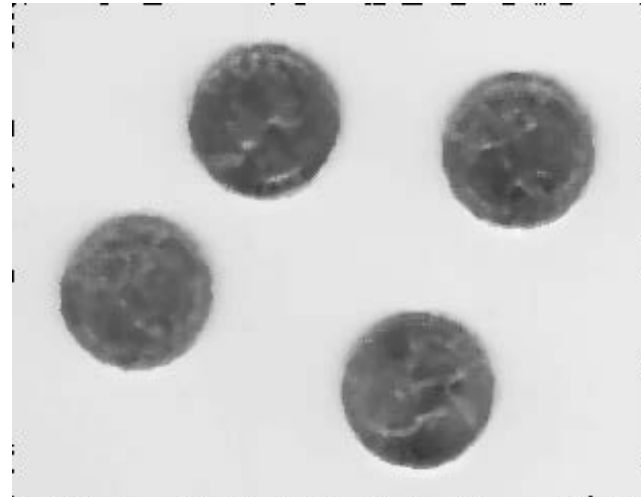
original



noisy ($p_a = p_b = 0.1$)



median filtered 3x3 window



median filtered 5x5 window

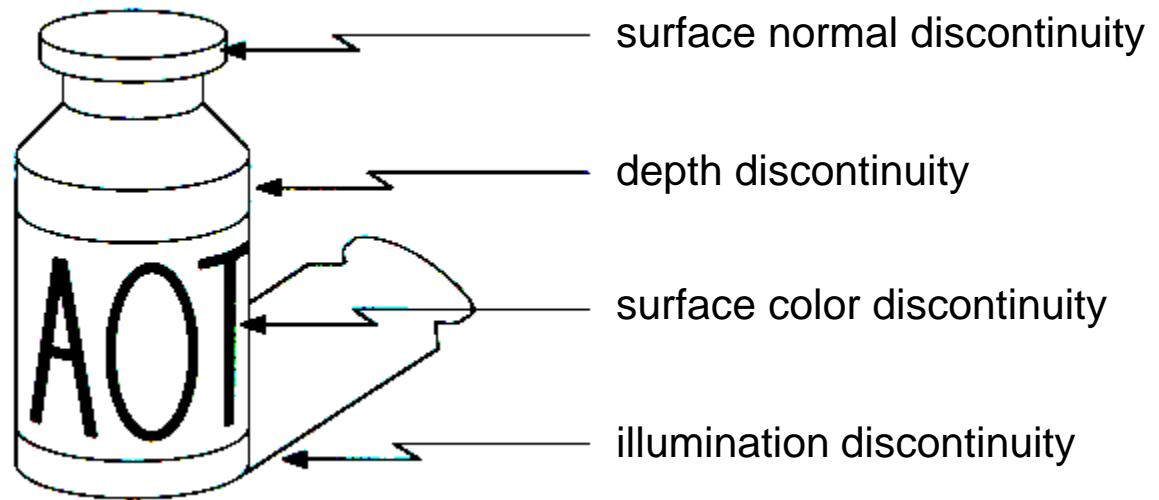
From MATLAB sample images

Filtering

- More in-depth treatment in Image Processing courses
 - In addition to spatial filtering, you can also filter in the frequency domain
- In computer vision, it is quite common to (Gaussian) blur any image prior to most processing tasks.

Edges and More Filters

Why are edges important?



- Edges are caused by a variety of factors
 - Something “interesting” is usually happening

Characterizing Edges

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

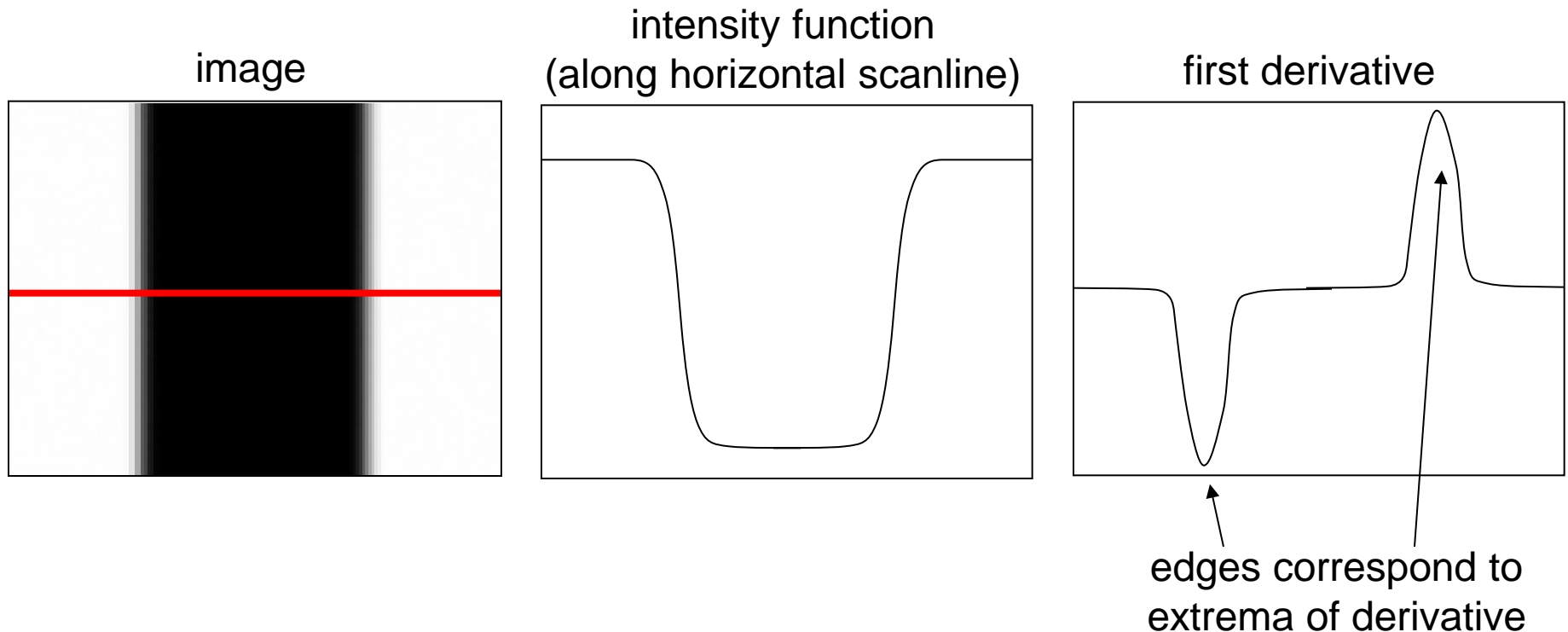
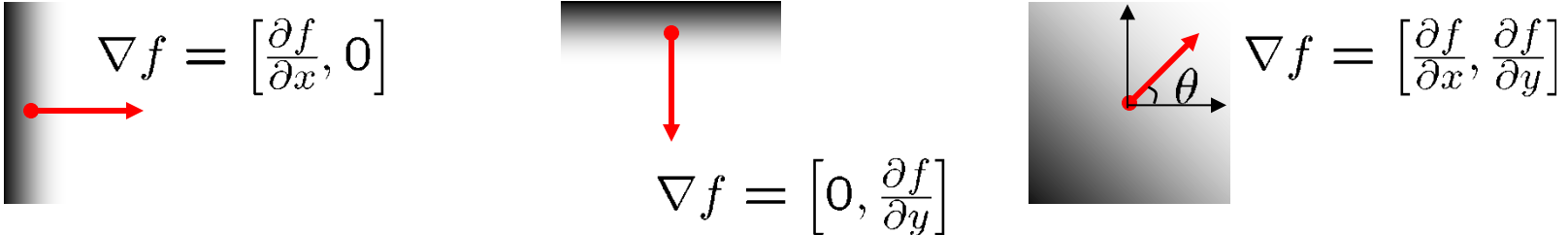


Image Gradient

- 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 increase in intensity

The gradient direction is given by $\theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$

Which is orthogonal to the direction of the edge in the image

The *edge strength* is given by the gradient magnitude

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

Differentiation and convolution

- Recall, for a 2D function, $f(x,y)$:

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

- We could approximate this as

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

- And design an linear image filter for differentiation

-1	1
----	---

Other Edge Filters

Prewitt: $M_x =$

-1	0	1
-1	0	1
-1	0	1

 ; $M_y =$

1	1	1
0	0	0
-1	-1	-1

Sobel: $M_x =$

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

 ; $M_y =$

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

Roberts: $M_x =$

0	1
-1	0

 ; $M_y =$

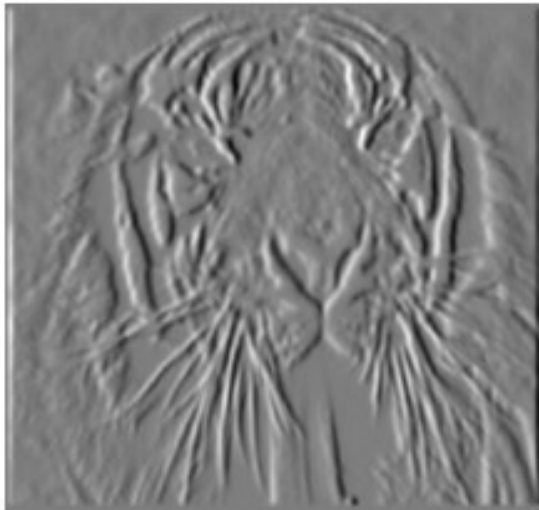
1	0
0	-1

Edge Filtering Example

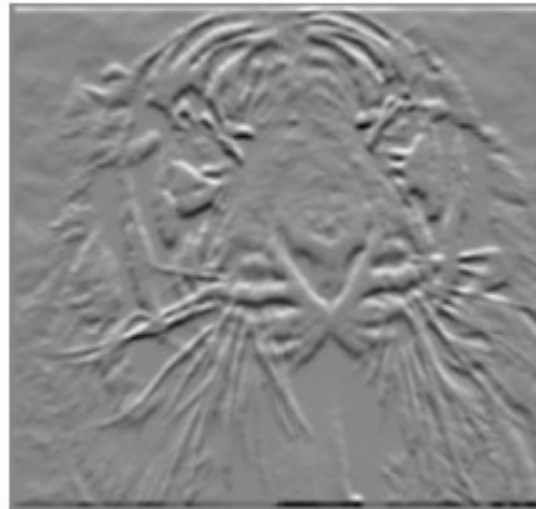


Gradient
(Magnitude)
Image

x-derivative
Image

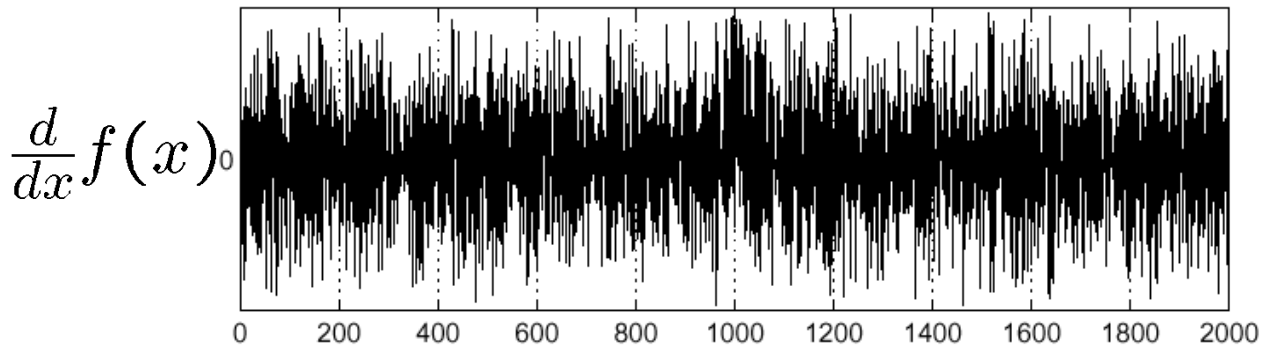
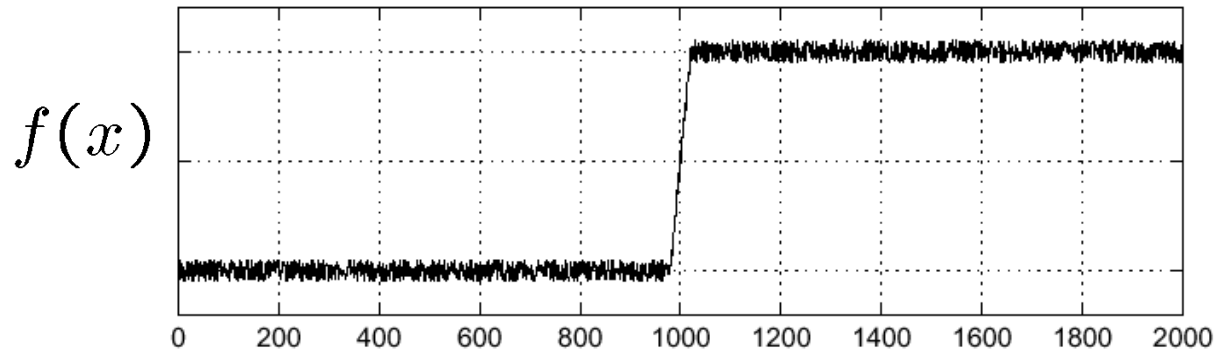


y-derivative
Image



Effects of noise

- Consider a single row or column of the image
 - Plotting intensity as a function of position gives a signal

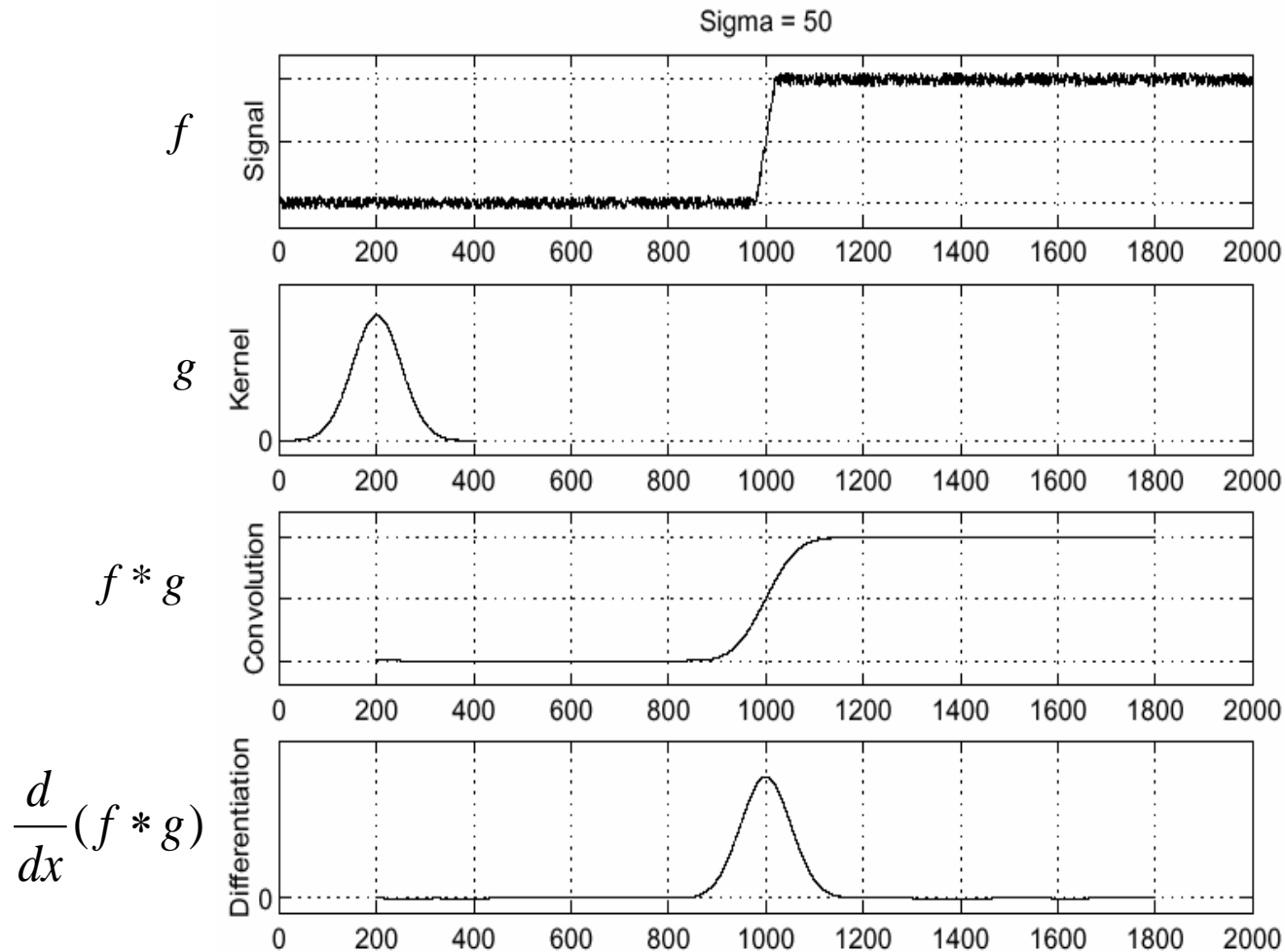


Where is the edge?

Effects of noise

- Derivative filters respond strongly to noise
 - Image noise results in pixels that look very different from their neighbors
 - Generally, the larger the noise the stronger the response
- What is to be done?
 - Smoothing the image should help, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors

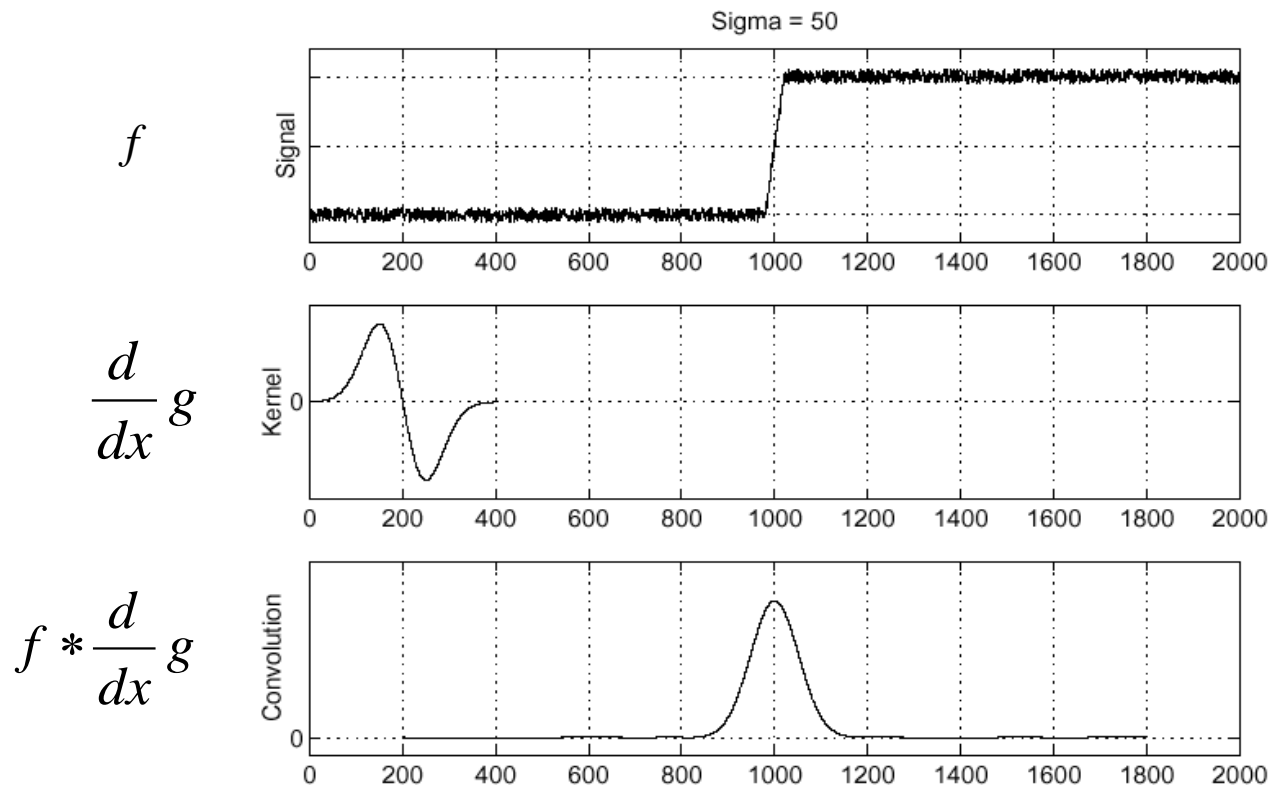
Solution: smooth first



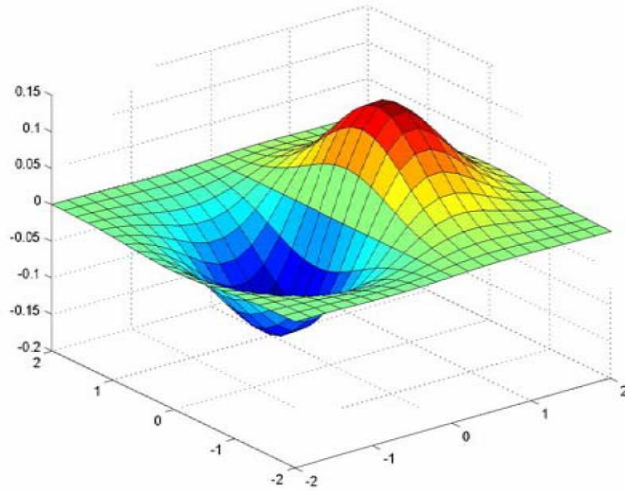
- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

Derivative theorem of convolution

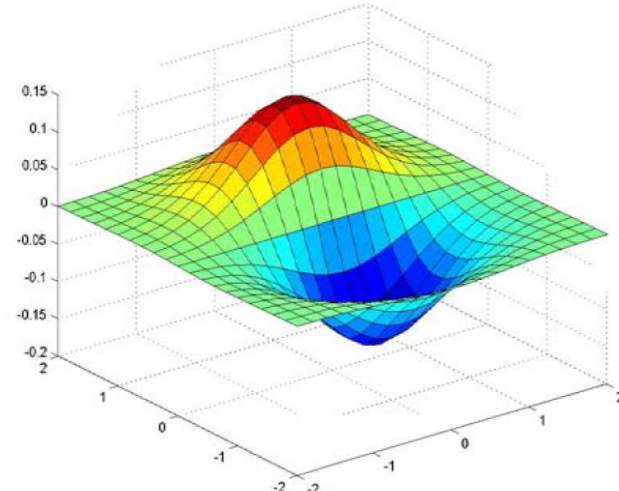
- Differentiation is convolution, and convolution is associative: $\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$
- This saves us one operation:



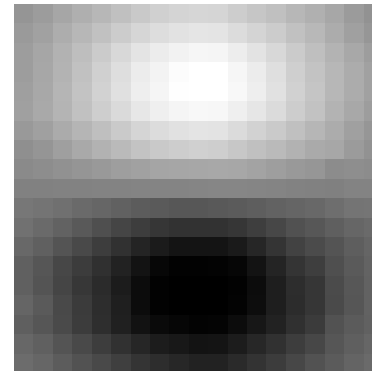
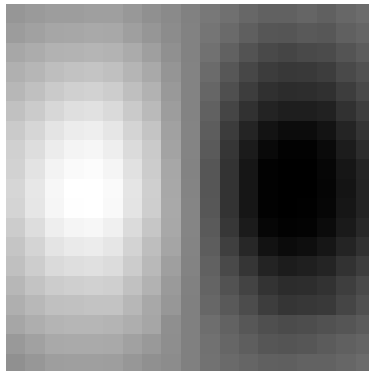
Derivative of Gaussian filter



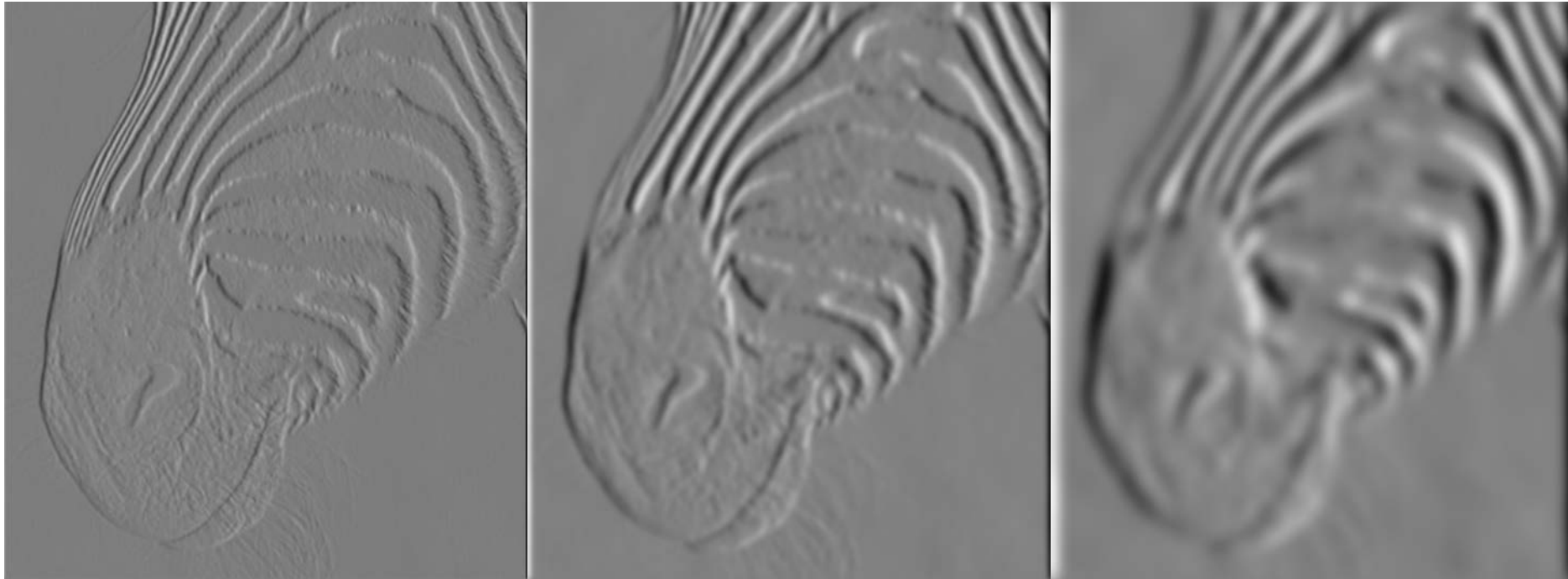
x-direction



y-direction



Tradeoff between smoothing and localization



1 pixel

3 pixels

7 pixels

- Smoothed derivative removes noise, but blurs edge. Also finds edges at different “scales”.

Summary: Filter Properties

- Filters act as templates
 - Highest response for regions that “look the most like the filter”
- Smoothing masks
 - Values positive
 - Sum to 1 \rightarrow constant regions are unchanged
 - Amount of smoothing proportional to mask size
- Derivative masks
 - Opposite signs used to get high response in regions of high contrast
 - Sum to 0 \rightarrow no response in constant regions
 - High absolute value at points of high contrast

Combining Filters

- Combining the edge filter with the Gaussian blur filter reduces the number of convolutions with the full-sized image from 2 to 1
- With certain filters, it is possible to get savings by *increasing* the number of convolutions
 - However, using smaller filters

Separable Filters

- Recall the Sobel edge filter
- A 2D filter is *separable* if it can be expressed as the outer product of two vectors

$$M_x = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Separable Filters (cont'd)

- Recall
 - Convolution is associative $f * (v * h) = (f * v) * h$
 - Filtering an $M*N$ image with a $P*Q$ filter requires roughly $MNPQ$ computations
- Using separable filters, you can filter in two steps
 - $MNP + MNQ = MN(P + Q)$
- The computational advantage is $PQ/(P + Q)$
 - For a 9-by-9 filter, that is 4.5x speedup

Image Filtering in Matlab

- *imfilter* is a built-in Matlab function from the Image Processing Toolbox
- *imfilter* takes in an image and a filter and returns the filtered image
 - type 'help imfilter' for additional options
- *fspecial* creates predefined 2D filters

fspecial filters

'average' averaging filter

'disk' circular averaging filter

'gaussian' Gaussian lowpass filter

'laplacian' filter approximating the 2D Laplacian

'log' Laplacian of Gaussian filter

'motion' motion filter

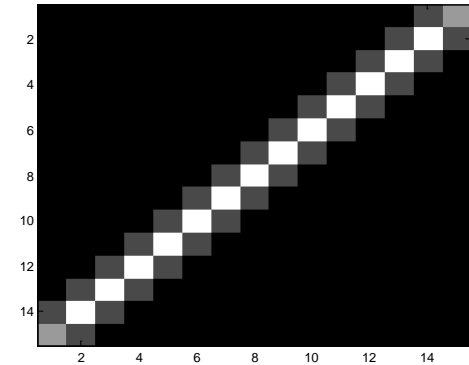
'prewitt' Prewitt horizontal edge-emphasizing filter

'sobel' Sobel horizontal edge-emphasizing filter

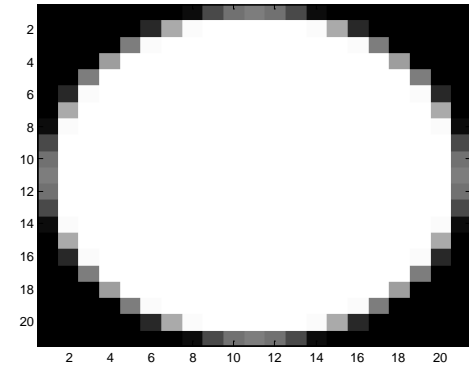
'unsharp' unsharp contrast enhancement filter

Built-in Filters

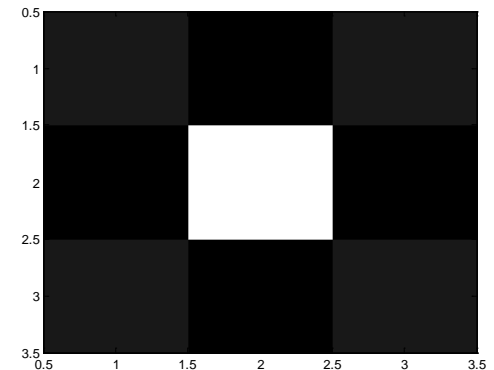
```
H = fspecial('motion',20,45);
```



```
H = fspecial('disk',10);
```



```
H = fspecial('unsharp');
```



Filtering in Matlab

```
I = imread('cameraman.tif');
```

```
H = fspecial('motion',20,45);
```

```
MotionBlur = imfilter(I,H,'replicate');
```

```
H = fspecial('disk',10);
```

```
blurred = imfilter(I,H,'replicate');
```

```
H = fspecial('unsharp');
```

```
sharpened = imfilter(I,H,'replicate');
```

*type **help fspecial** to see
full example code*

Original Image



Motion Blurred Image



Blurred Image



Sharpened Image

