ECE 1390/2390 Image Processing and Computer Vision – Fall 2021

Lecture 4: Linear Filtering – Edge detection

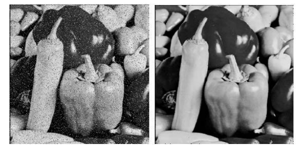
Ahmed Dallal
Assistant Professor of ECE
University of Pittsburgh

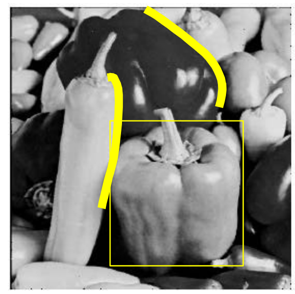
Reading

• FP 5.1, 5.2

Filters for features

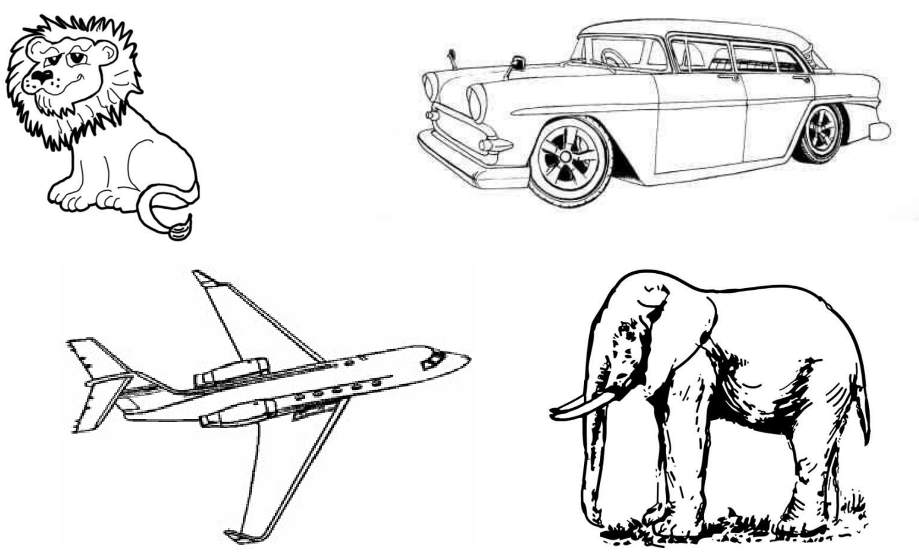
- Previously, thinking of filtering as a way to remove or reduce **noise**
- Now, consider how filters will allow us to abstract higher- level "**features**".
  - Map raw pixels to an intermediate representation that will be used for subsequent processing
  - Goal: reduce amount of data, discard redundancy,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





K. Grauman

Reduced images

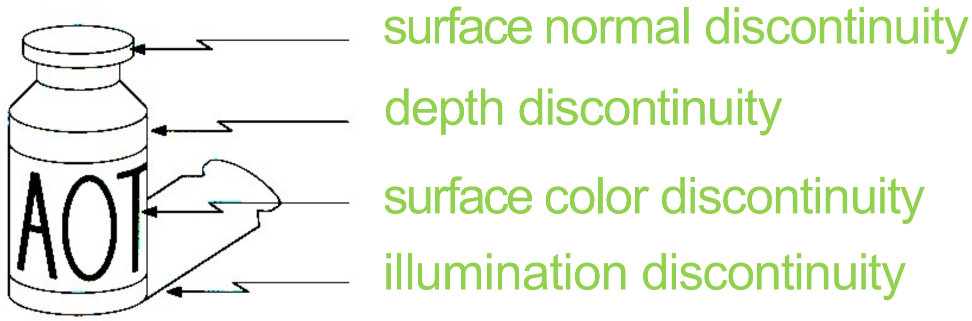


Reduced images

Edges seem to be important…

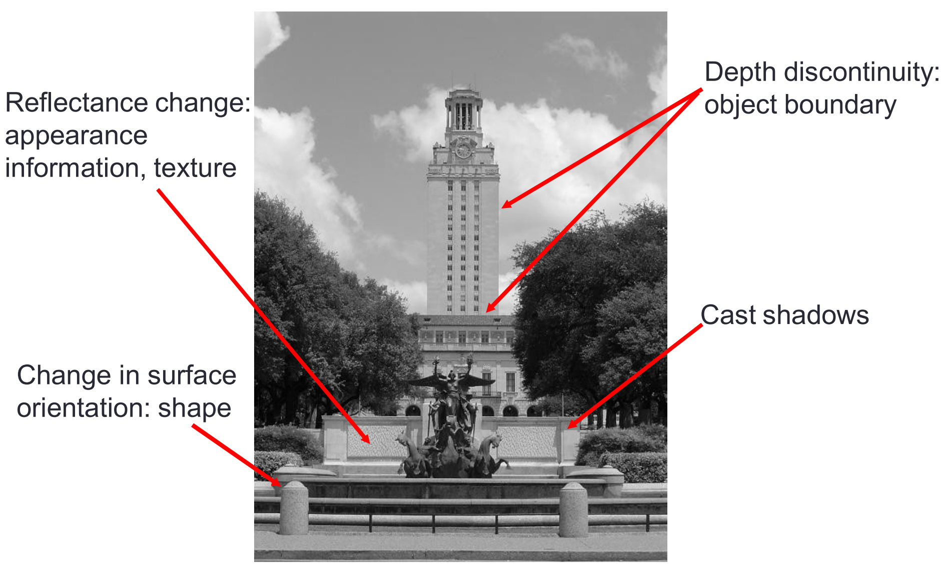
Edge detection: Gradients

Origin of Edges

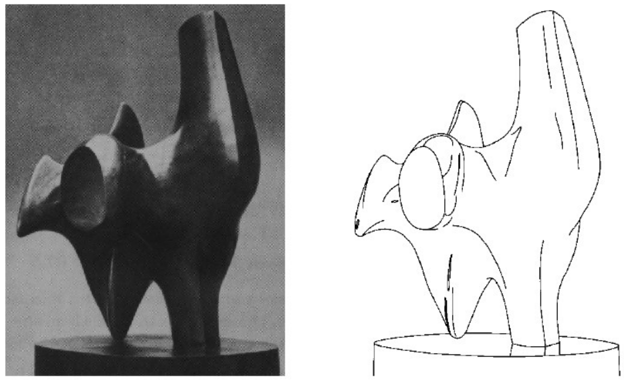


- Edges are caused by a variety of factors
- Information theory view: edges encode change, change is what is hard to predict, therefore edges efficiently encode an image

In a real image

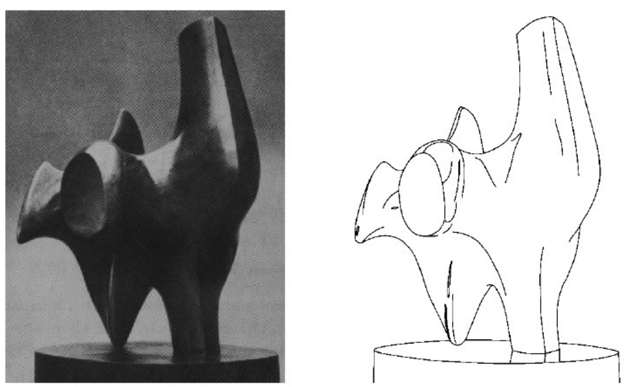


Edge detection



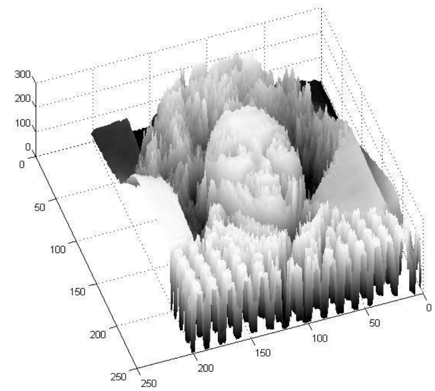
- Convert a 2D image into a set of curves
  - Extracts salient features of the scene
  - More compact than pixels

Edge detection



- How can you tell that a pixel is on an edge?

Recall images as functions…



Edges look like steep cliffs → very large change in gray level

Edge Detection

Basic idea: look for a neighborhood with strong signs of change.

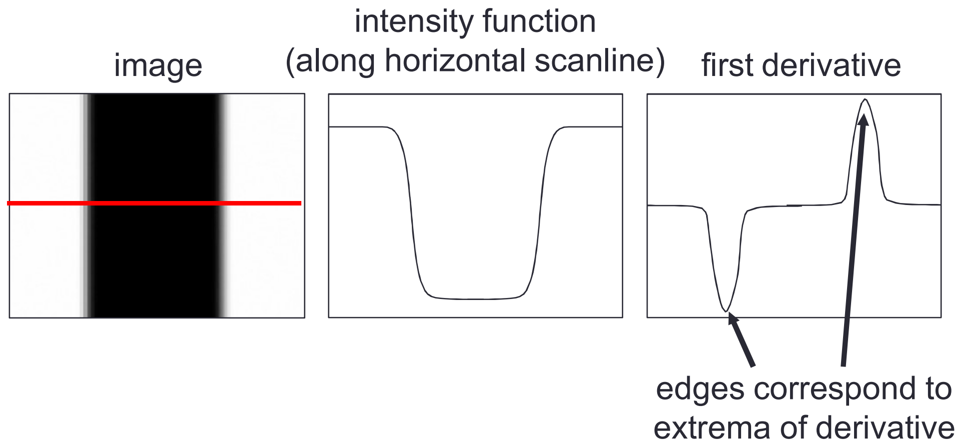
Problems:

- neighborhood size
- how to detect change

81	82	26	24
82	33	25	25
81	82	26	24

Derivatives and edges

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



Source: L. Lazebnik

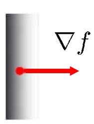
Differential Operators

- Differential operators –when applied to the image returns some derivatives.
- Model these "operators" as masks/kernels that compute the image gradient function.
- Threshold the this gradient function to select the edge pixels.
- Which brings us to the question:

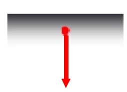
What's a gradient?

Image gradient

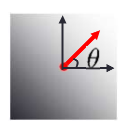
The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



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



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



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

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

The gradient direction is given by:

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

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

The *edge strength* is given by the gradient magnitude $\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

Discrete gradient

- 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:

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

$$\approx f(x+1,y) - f(x,y) \qquad \textit{“right derivative”}$$

Finite differences

Source: D.A. Forsyth

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:

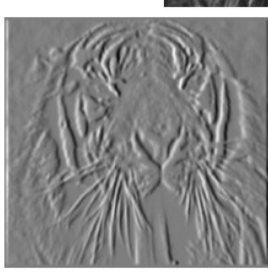
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$$

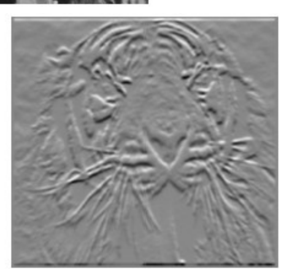
- To implement above as correlation, what would be the associated filter?

Partial derivatives of an image

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

-1	1





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

-1
1

? or

1
-1

Which shows changes with respect to x?

(showing correlation filters)

The discrete gradient

- We want an "operator" (mask/kernel) that we can apply to the image that implements:

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

How would you implement this as a cross-correlation?
(not flipped)

0	0
-1	+1
0	0

H

Not symmetric around image point; which is "middle" pixel?

0	0	0
-1/2	0	+1/2
0	0	0

H

Average of "left" and "right" derivative. See?

Example: Sobel operator

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

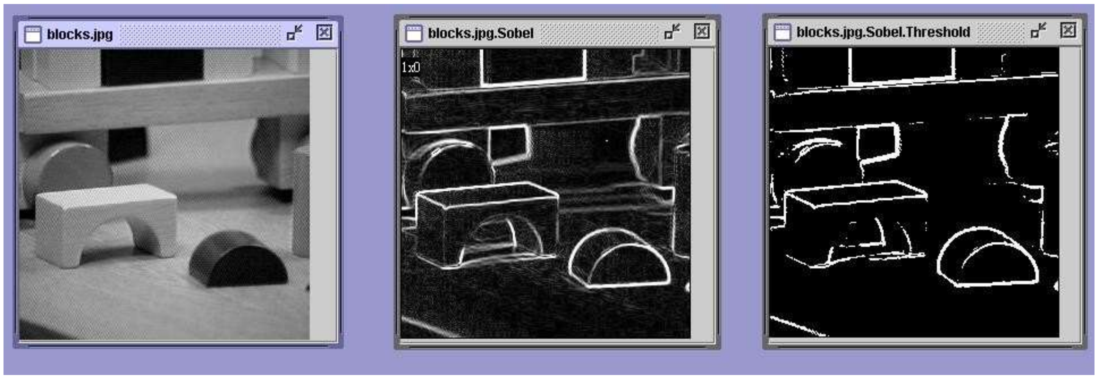
s_x $\qquad$ s_y

On a pixel of the image I
•Let g_x be the response to mask S_x (sometimes * 1/8)
•Let g_y be the response to mask S_y
What is the gradient?

(Sobel) Gradient is $\nabla \mathbf{I} = [\mathbf{g_x} \ \ \mathbf{g_y}]^T$

$g = (g_x^2 + g_y^2)^{1/2}$ is the gradient magnitude.
$\theta = \mathrm{atan2}(g_y, g_x)$ is the gradient direction.

Sobel Operator on Blocks Image



original image

gradient magnitude

thresholded gradient magnitude

Some Well-Known Gradients Masks

- Sobel:

Sx

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

Sy

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

- Prewitt:

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

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

- Roberts

0	1
-1	0

1	0
0	-1

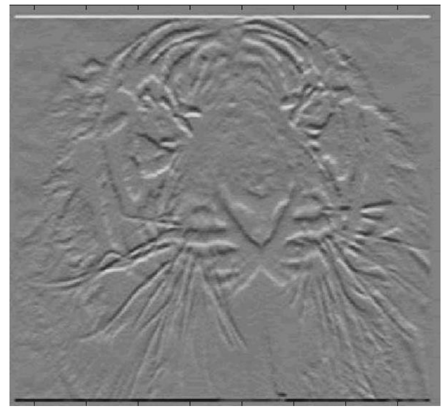
MATLAB does edges

```
filt = fspecial('sobel')
```

```
filt =

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

```
outim = imfilter(double(im),filt);
imagesc(outim);
colormap gray;
```



Quiz

It is better to compute gradients using:

a) ***Convolution*** since that's the right way to model filtering so you don't get flipped results.

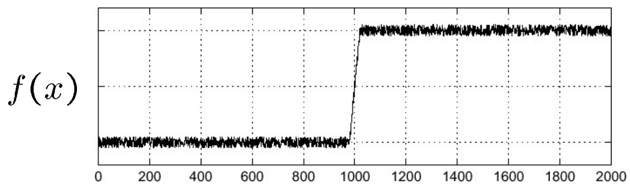
b) ***Correlation*** because it's easier to know which way the derivatives are being computed.

c) Doesn't matter.

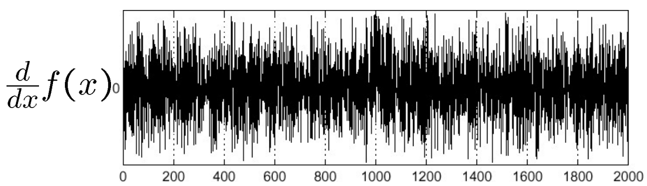
d) Neither since I can just write a for-loop to computer the derivatives.

But…

- Consider a single row or column of the image
  - Plotting intensity as a function of *x*

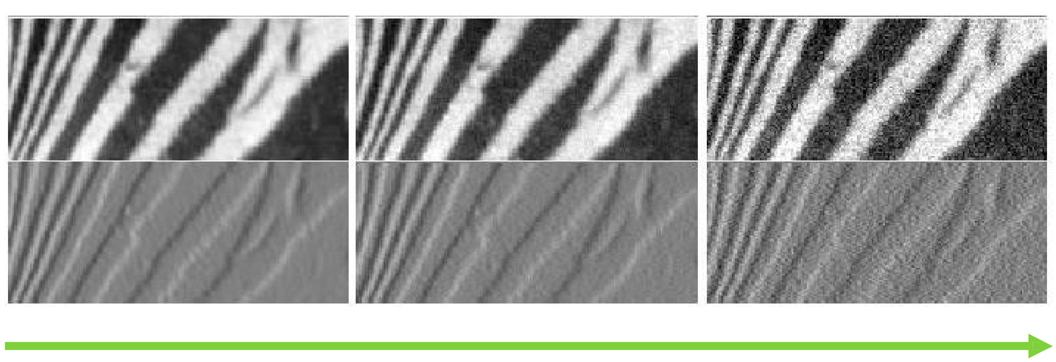


  - Apply derivative operator….



Uh, where's the edge?

Finite differences responding to noise

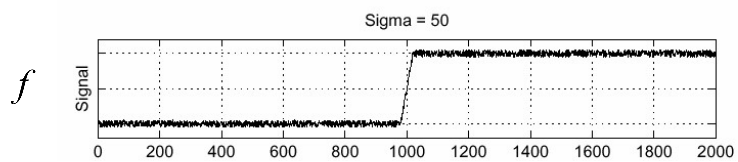


Increasing noise

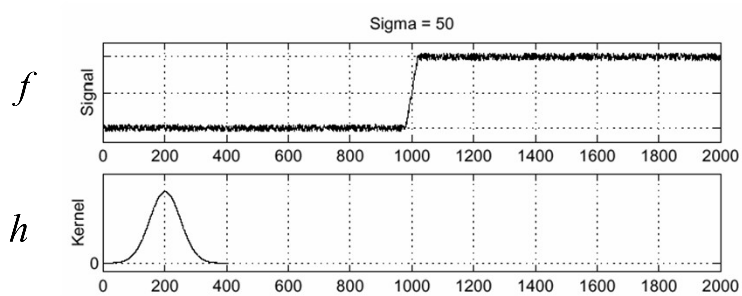
(this is zero mean additive Gaussian noise)

Source: D. Forsyth

Solution: smooth first



Solution: smooth first

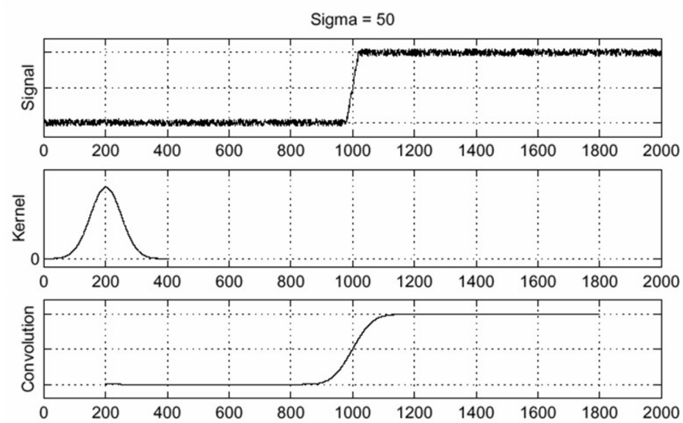


Solution: smooth first

f

h

$h * f$



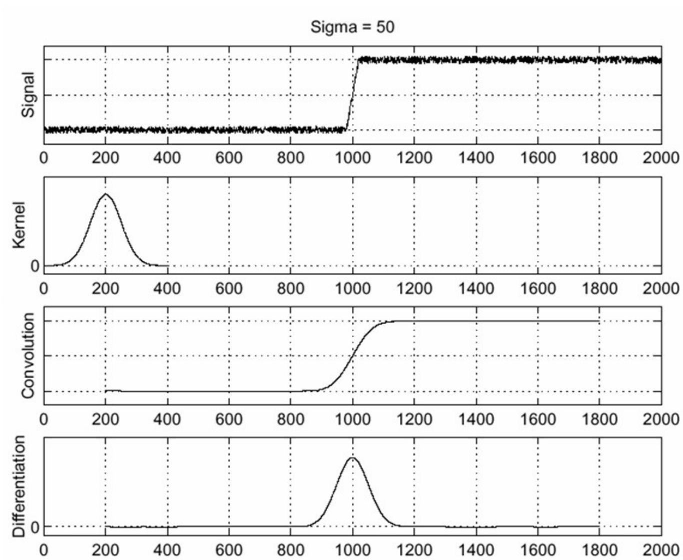
Solution: smooth first

f

h

$h * f$

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



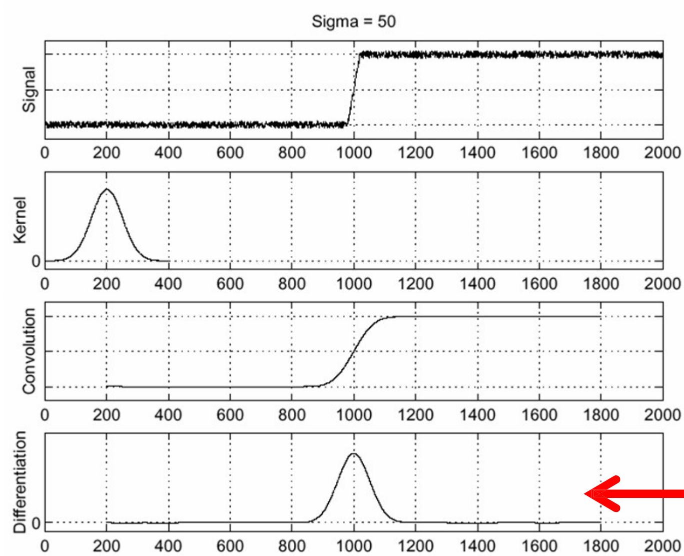
Solution: smooth first

f

h

$h * f$

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



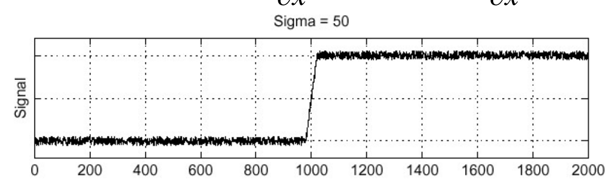
Where is the edge?

Look for peaks

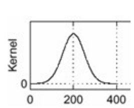
Derivative theorem of convolution

- This saves us one operation: $\frac{\partial}{\partial x}(h * f) = (\frac{\partial}{\partial x}h) * f$

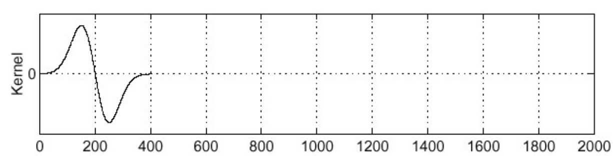
f



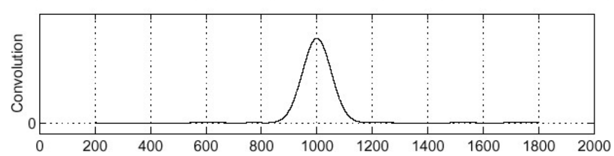
h



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



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



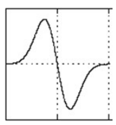
How can we find (local) maxima of a function?

2nd derivative of Gaussian

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

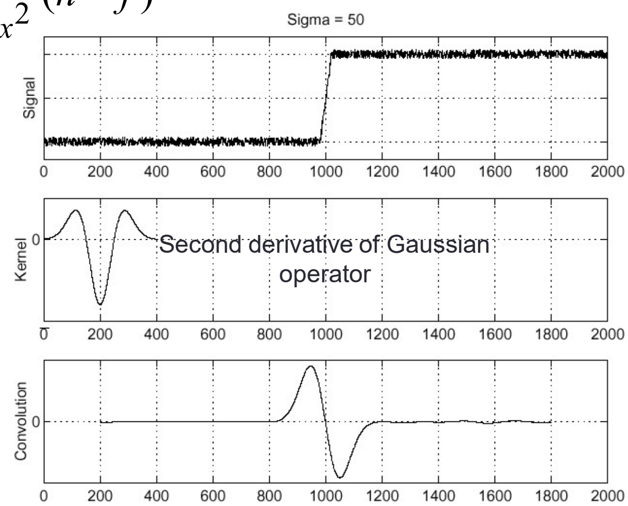
f

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



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

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



Where is the edge? Zero-crossings of bottom graph

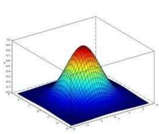
Edge detection: 2D operators

Derivative of Gaussian filter – 2D

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

Derivative of Gaussian filter – 2D

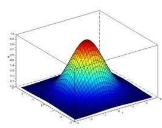
$$(I \otimes g) \otimes h_x = I \otimes (g \otimes h_x)$$



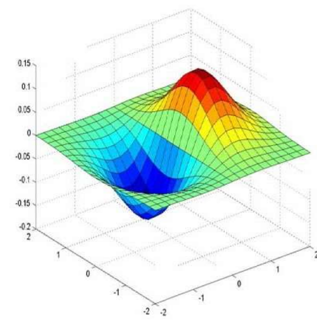
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \otimes \begin{bmatrix} -1 & 1 \end{bmatrix}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

Derivative of Gaussian filter – 2D

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



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



Is this preferable?

Quiz

Why is it preferable to apply *h* to the smoothing function *g* and apply the result to the Image.

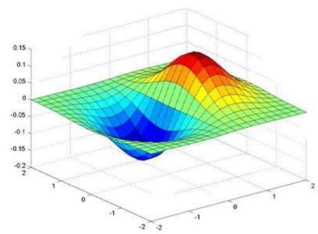
a) It's not – they are mathematically equivalent.

b) Since *h* is typically smaller we take fewer derivatives so it's faster.

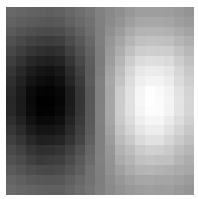
c) The smoothed derivative operator is computed once and you have it to use repeatedly.

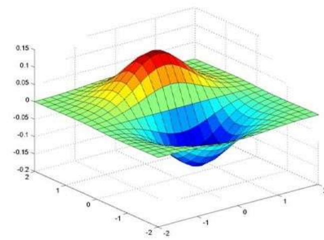
d) B & C

Derivative of Gaussian filter

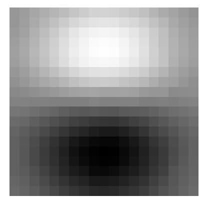


x-direction





y-direction

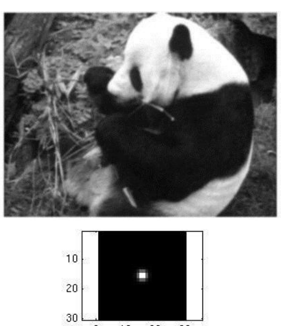


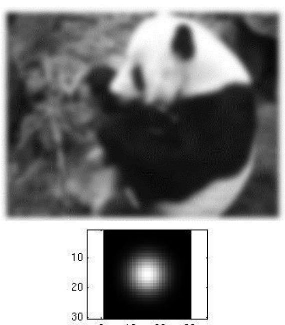
Correlation or convolution?

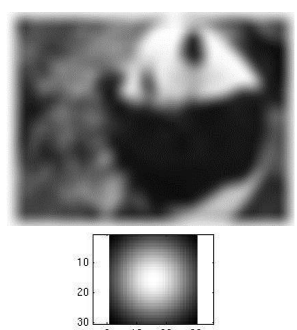
And for y it's always a problem!

Source: S. Lazebnik

Smoothing with a Gaussian

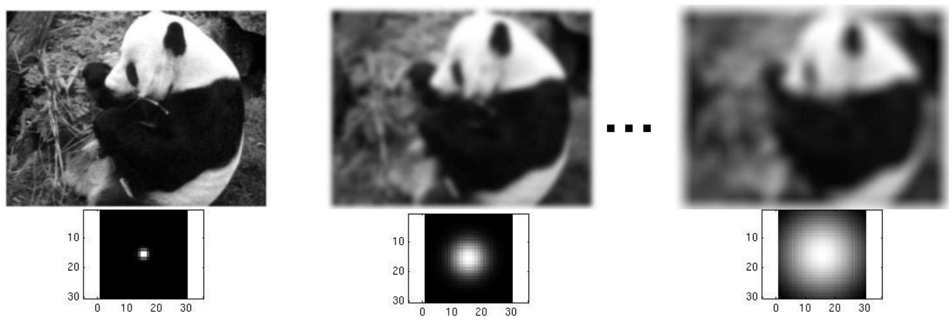






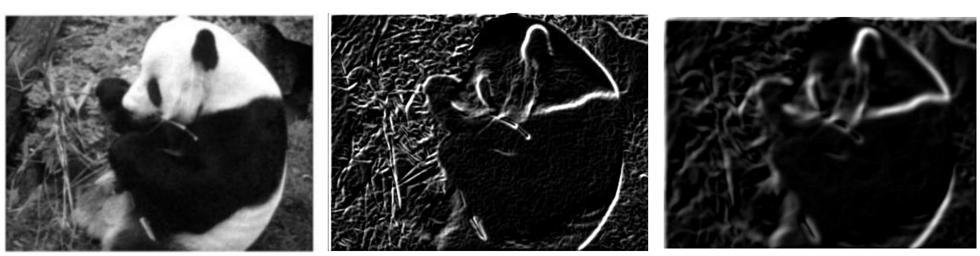
```
for sigma=1:3:10
 h = fspecial('gaussian', fsize, sigma);
 out = imfilter(im, h);
 imshow(out);
 pause;
end
```

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**

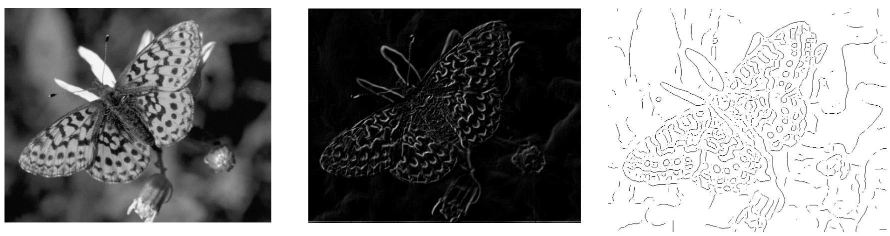
The apparent structures differ depending on Gaussian’s scale parameter.

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

Gradients -> edges

Primary edge detection steps:

- 1. Smoothing: suppress noise
- 2. Edge "enhancement": filter for contrast
- 3. Edge localization ("Thin")
  - Determine which local maxima from filter output are actually edges vs. noise
    - Threshold, Thin → get a single contour
- We may need linking to connect edge pixels.



Canny edgedetector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:

   Thin multi-pixel wide "ridges" down to single pixel width

Canny edgedetector

4. 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');
>>doc edge *(or help edge* if doc is not supported)

Source: D. Lowe, L. Fei-Fei

The Canny edge detector

original image (Lena)

The Canny edge detector

magnitude of the gradient

The Canny edge detector

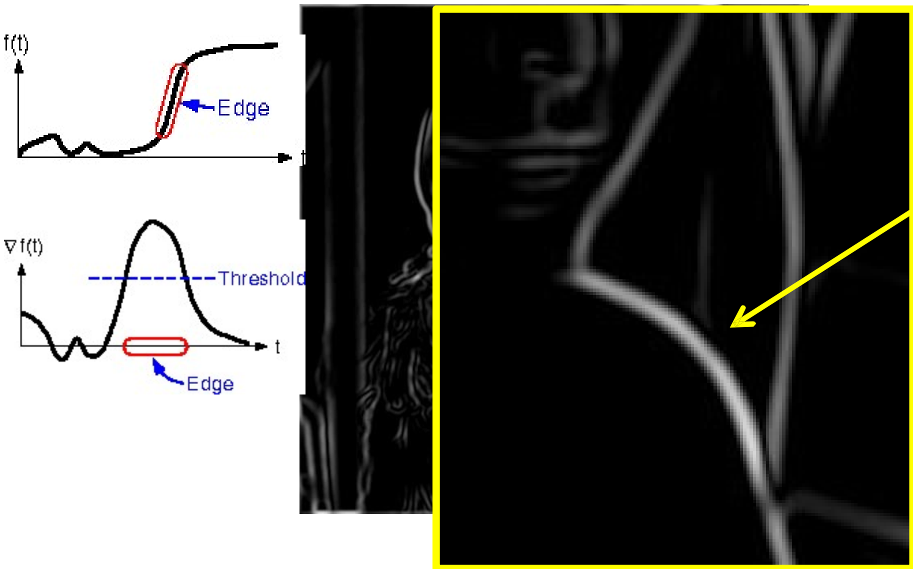
thresholding

The Canny edge detector

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

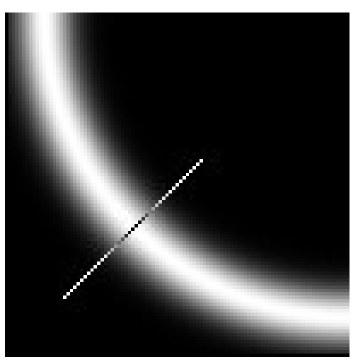
The Canny edge detector

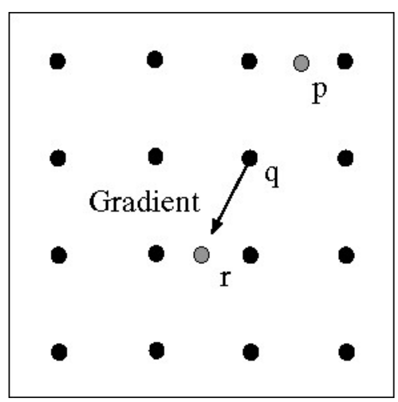
The Canny edge detector



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

Canny: Non-maximal suppression





- Check if pixel is local maximum along gradient direction
    - can require checking interpolated pixels p and r

The Canny edge detector

thinning
(non-maximum suppression)

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

Canny threshold hysteresis

1. Apply a high threshold to detect strong edge pixels.
2. Link those strong edge pixels to form strong edges.
3. Apply a low threshold to find weak but plausible edge pixels.
4. Extend the strong edges to follow weak edge pixels.

Result of Canny

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

- Large σ detects large scale edges
- Small σ detects fine features

The choice of σ depends on desired behavior

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.

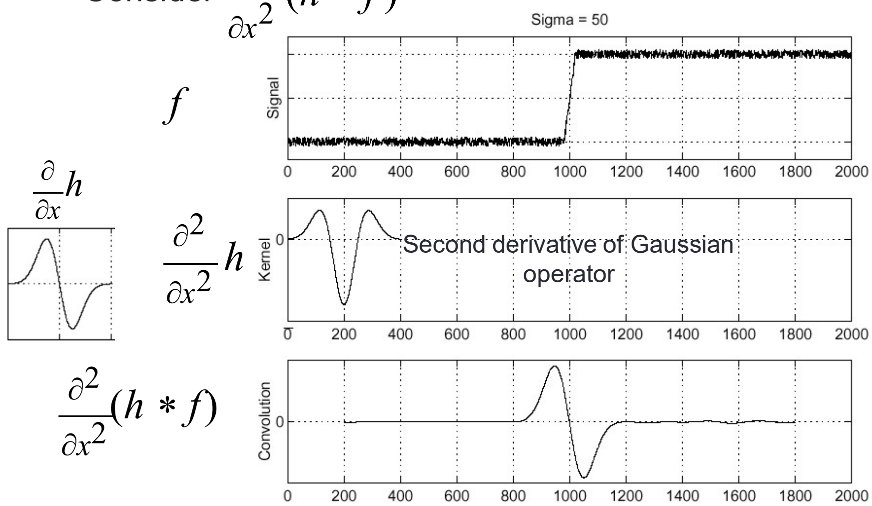
Quiz

The Canny edge operator is probably quite sensitive to noise.

a) True – derivatives accentuate noise
b) False – the gradient is computed using a derivative of Gaussian operator which removes noise.
c) Mostly false – it depends upon the σ chose.

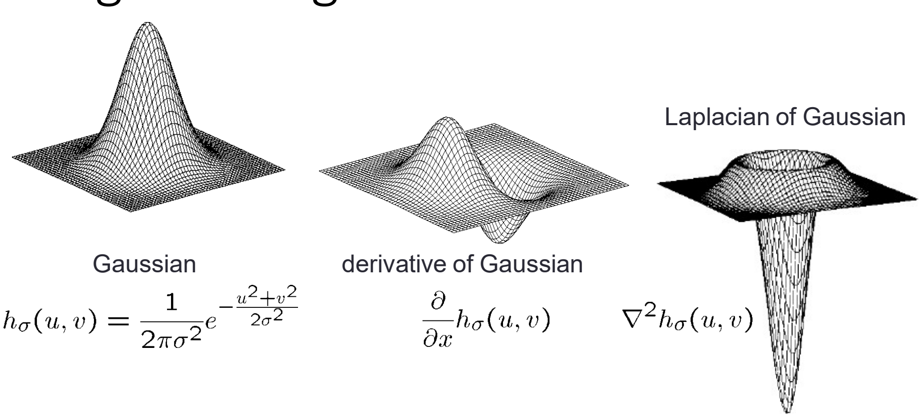
Recall: 2nd derivative of Gaussian (1-D)

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



Zero-crossings of bottom graph are edges

Single 2D edge detection filter



∇^2 is the **Laplacian** operator. Edges at zero crossings:

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

Edge demo

```
% Edge Demo
pkg load image; % Octave only

%% Read Lena image
lena = imread('lena.png');
figure, imshow(lena), title('Original image, color');

%% Convert to monochrome (grayscale) using rgb2gray
lenaMono = rgb2gray(lena);
figure, imshow(lenaMono), title('Original image, monochrome');

%% Make a blurred/smoothed version
h = fspecial('gaussian', [11 11], 4);
figure, surf(h);
lenaSmooth = imfilter(lenaMono, h);
figure, imshow(lenaSmooth), title('Smoothed image');
```

Edge demo (contd.)

```
%% Method 1: Shift left and right, and show diff image
lenaL = lenaSmooth;
lenaL(:, [1:(end - 1)]) = lenaL(:, [2:end]);
lenaR = lenaSmooth;
lenaR(:, [2:(end)]) = lenaR(:, [1:(end - 1)]);
lenaDiff = double(lenaR) - double(lenaL);
figure, imshow(lenaDiff, []), title('Difference between right and left shifted images');

%% Method 2: Canny edge detector
cannyEdges = edge(lenaMono, 'canny'); % on original mono image
figure, imshow(cannyEdges), title('Original edges');
cannyEdges = edge(lenaSmooth, 'canny'); % on smoothed image
figure, imshow(cannyEdges), title('Edges of smoothed image');
% notice how a lot of the detail features are now gone

%% Method 3: Laplacian of Gaussian
logEdges = edge(lenaMono, 'log');
figure, imshow(logEdges), title('Laplacian of Gaussian');
```

Summary

- Hopefully you've learned filtering by convolution and correlation, taking derivatives by operators,computing gradients and using these for edge detection.
- We've also discussed filters astemplates – something we'll use again later.
- Next we'll take adetour and do some "real" computer vision where we fid structuresin images. It will make use of the edges we discussed today.