CS4495/6495
Introduction to Computer Vision

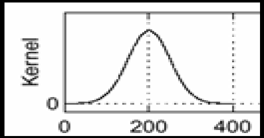
2A-L6 *Edge detection:*
2D operators

Derivative theorem of convolution - 1D

- This saves us one operation:

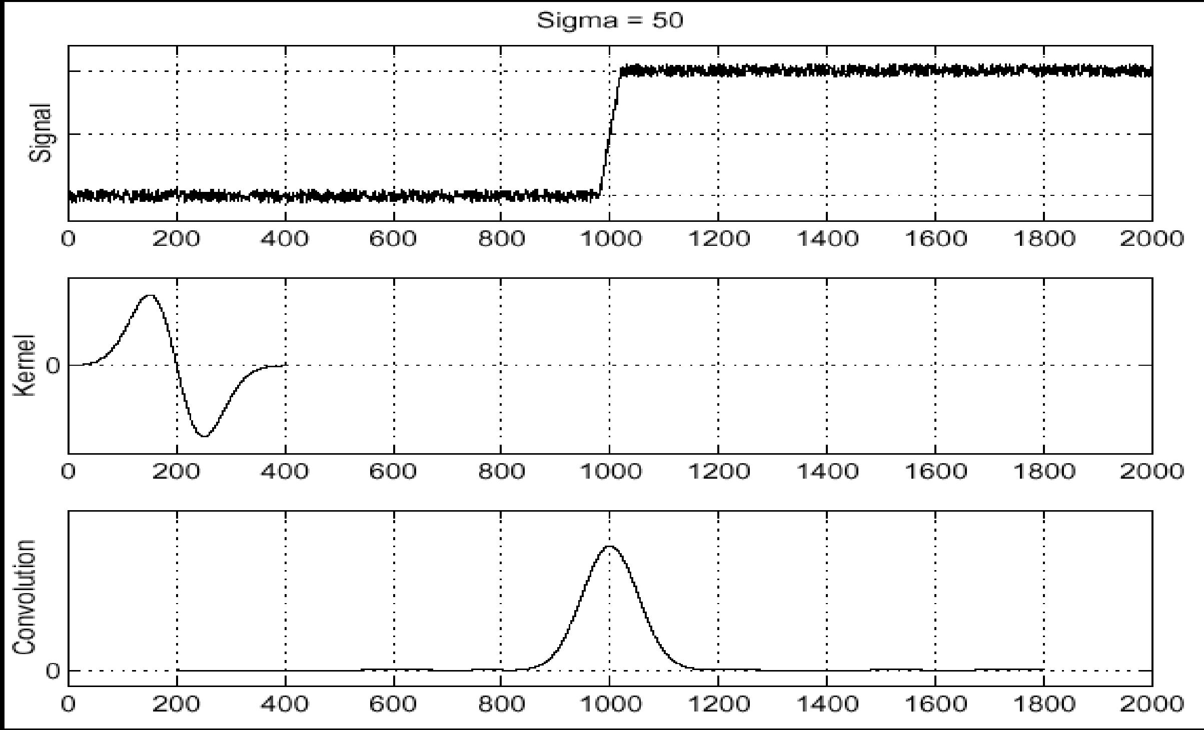
f

h



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

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

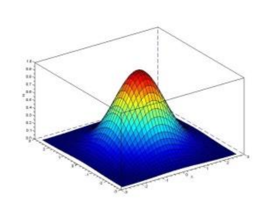


Derivative of Gaussian filter – 2D

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

Derivative of Gaussian filter – 2D

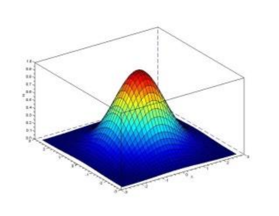
$$(I \otimes g) \otimes h_x = I \otimes (g \otimes h_x)$$



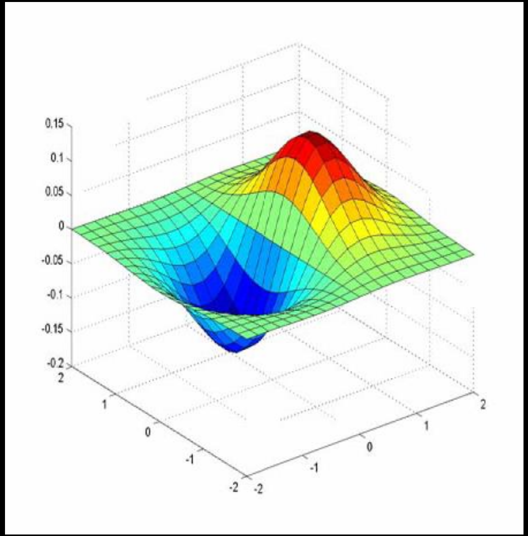
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \otimes \begin{bmatrix} -1 & 1 \end{bmatrix}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

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?

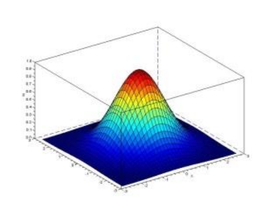
Quiz

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

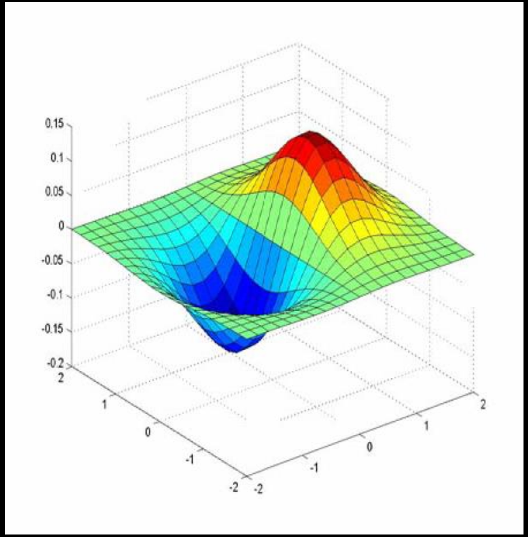
a) It's not – they are mathematically equivalent.
b) Since *h* is typically smaller we take fewer derivatives so it's faster.
c) The smoothed derivative operator is computed once and you have it to use repeatedly.
d) B & C

Derivative of Gaussian filter – 2D

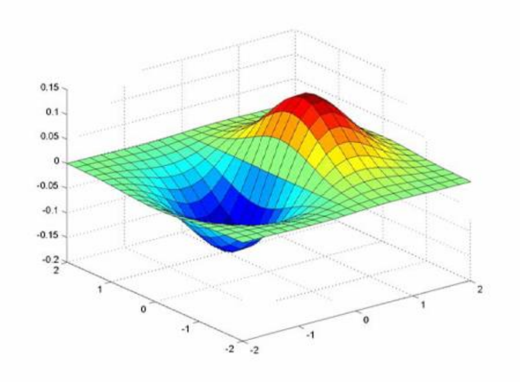
$$(I \otimes g) \otimes h = I \otimes (g \otimes h)$$



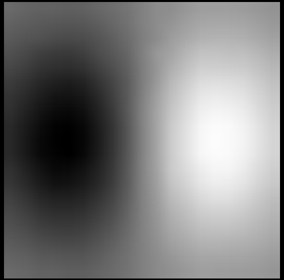
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \otimes \begin{bmatrix} -1 & 1 \end{bmatrix}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

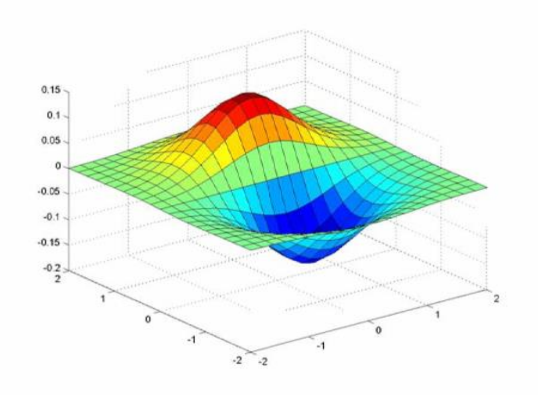


Derivative of Gaussian filter

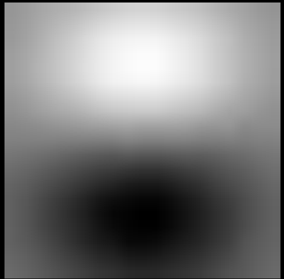


x-direction





y-direction

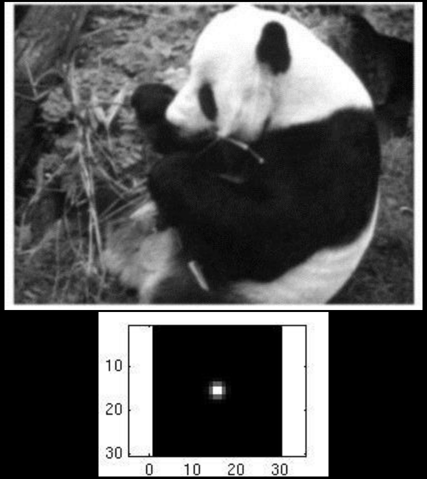


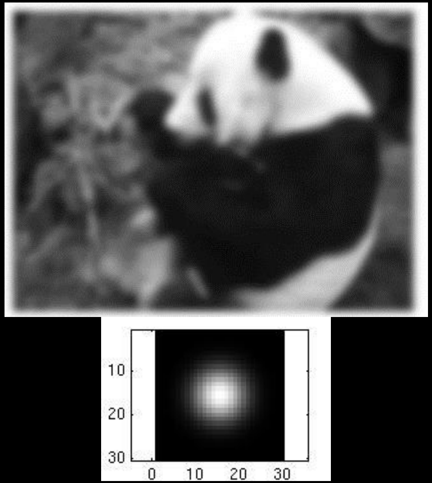
Correlation or convolution?

And for y it's always a problem!

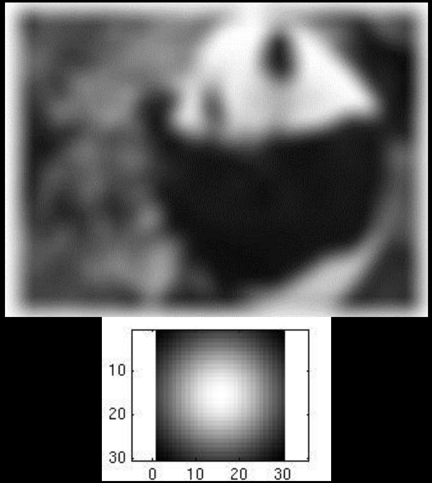
Source: S. Lazebnik

Smoothing with a Gaussian





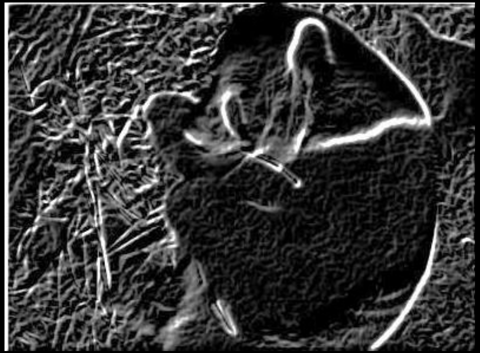
...




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

Effect of σ on derivatives

Effect of σ on derivatives



σ = 1 pixel

σ = 3 pixels

Smaller values: finer features detected

Larger values: larger scale edges detected

Gradients -> edges

Primary edge detection steps:

1. Smoothing derivatives to suppress noise and compute gradient.

2. Threshold to find regions of “significant” gradient.

3. “Thin” to get localized edge pixels

4. And link or connect edge pixels.

Canny edge detector

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

Source: D. Lowe, L. Fei-Fei

Canny edge detector

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

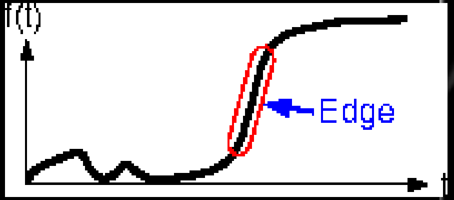
The Canny edge detector

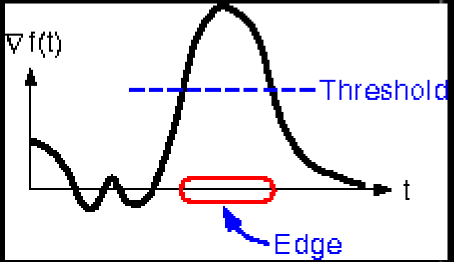
thresholding

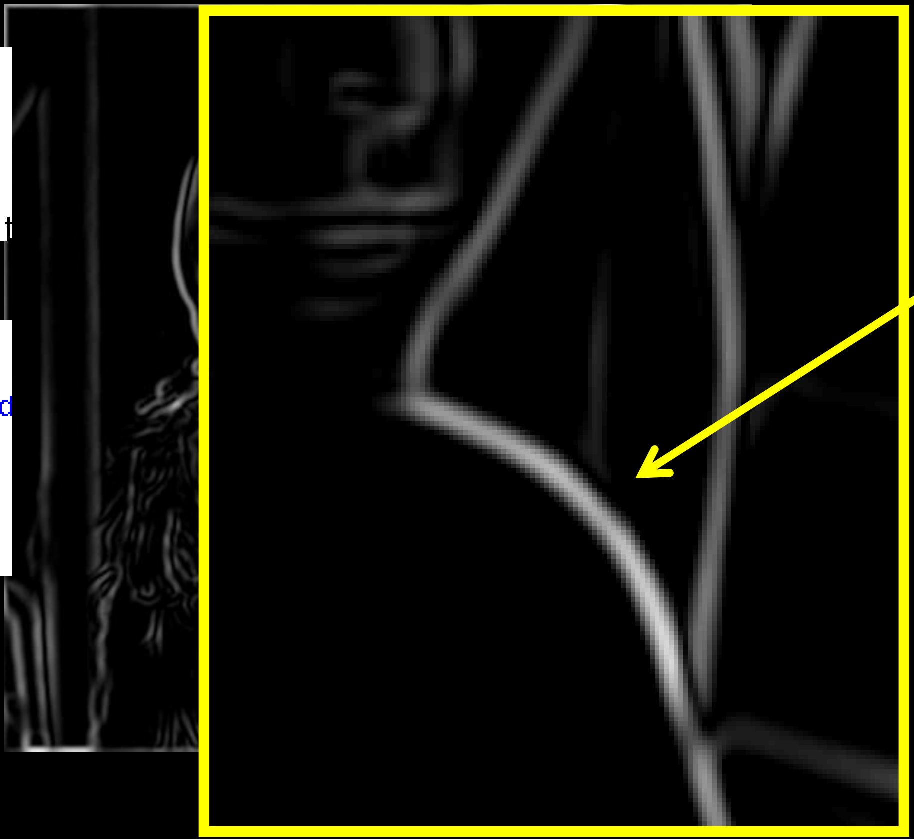
The Canny edge detector

thinning
(non-maximum suppression)

The Canny edge detector

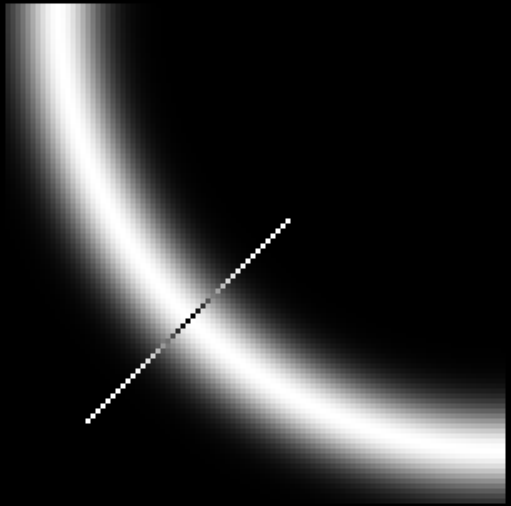


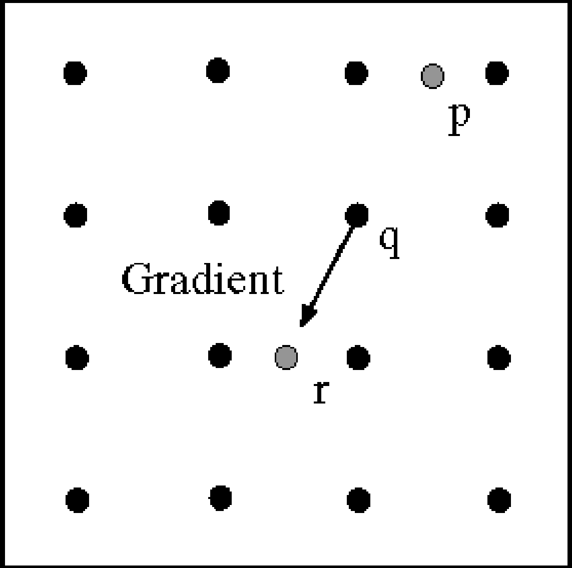




How to turn these thick regions of the gradient into curves?

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?

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

Too coarse of a scale…the branches are gone.

Quiz

The Canny edge operator is probably quite sensitive to noise.

a) True – derivatives accentuate noise

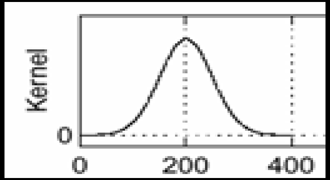
b) False – the gradient is computed using a derivative of Gaussian operator which removes noise.

c) Mostly false – it depends upon the σ chose.

Recall 1D 2nd derivative of Gaussian

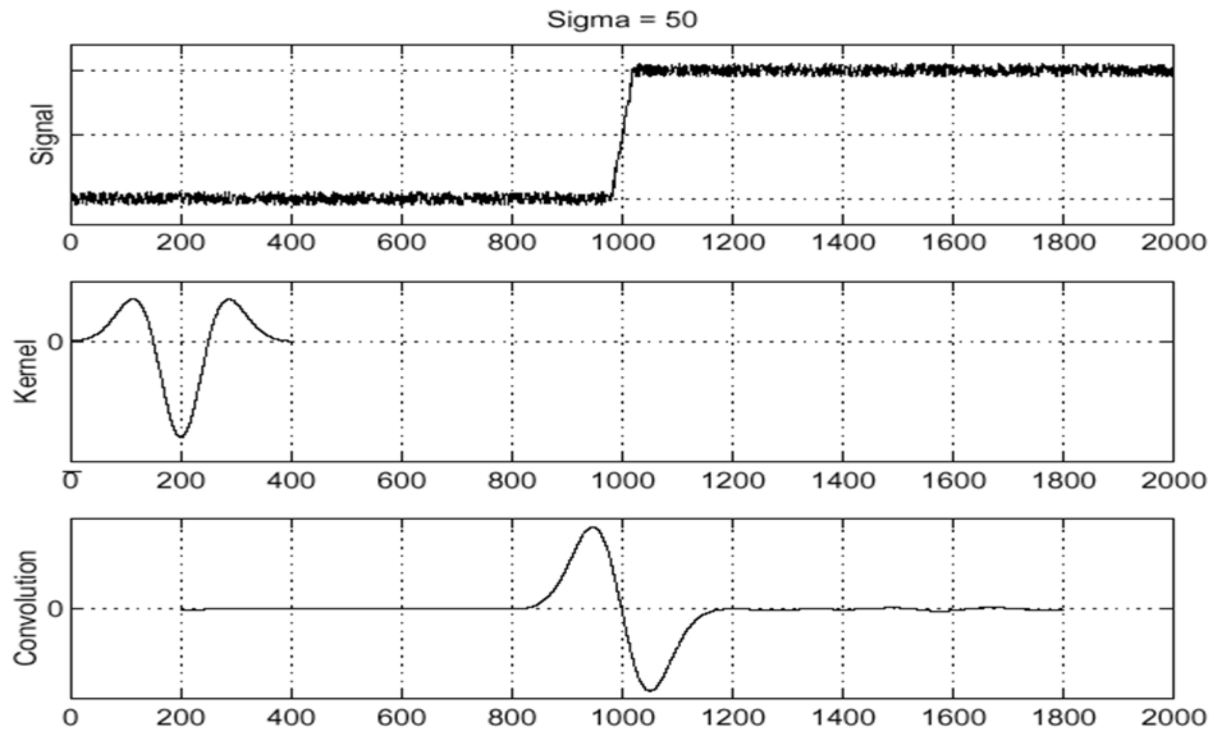
f

h



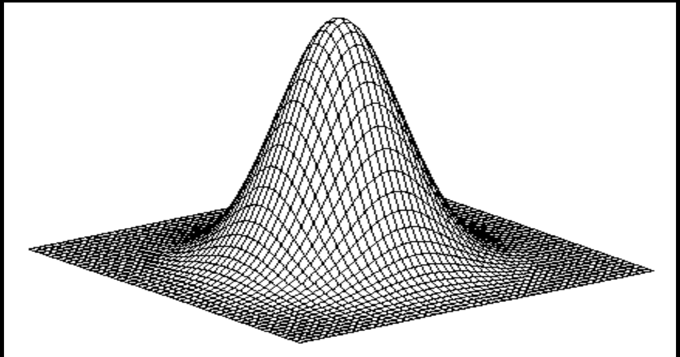
$$\frac{\partial^2}{\partial x^2} h$$

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

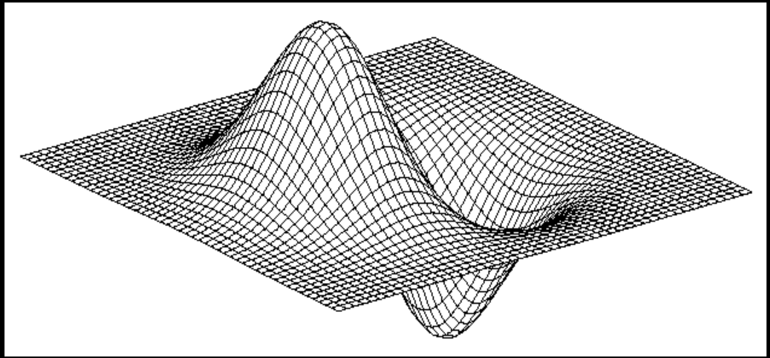


Zero-crossings of bottom graph are edges

Single 2D edge detection filter



Gaussian



derivative of Gaussian

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

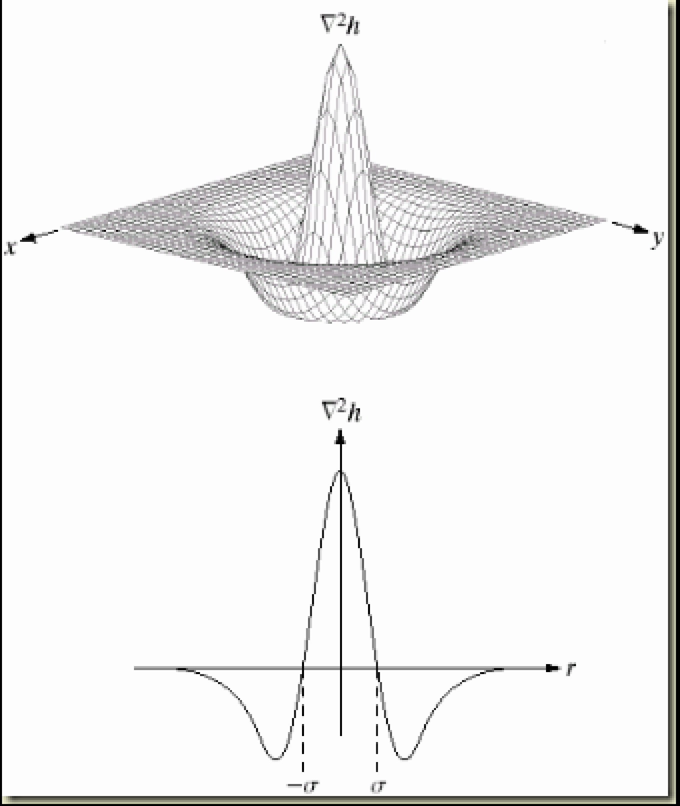
$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Single 2D edge detection filter

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

∇^2 is the **Laplacian** operator,

And the zero-crossings are the edges.



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');

%% 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 as templates – something we'll use again later.
- Next we'll take a detour and do some "real" computer vision where we fid structures in images. It will make use of the edges we discussed today.