Pre-processing

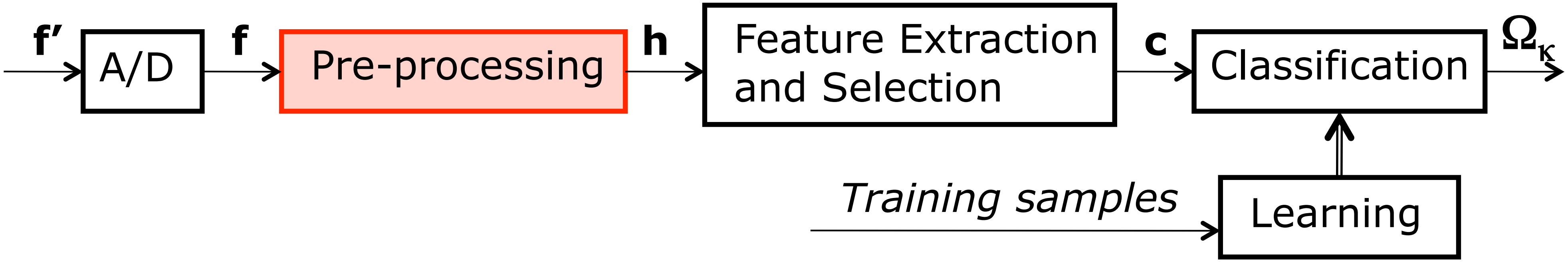
Filtering: edge detection

Dr. Elli Angelopoulou

Lehrstuhl für Mustererkennung (Informatik 5)

Friedrich-Alexander-Universität Erlangen-Nürnberg

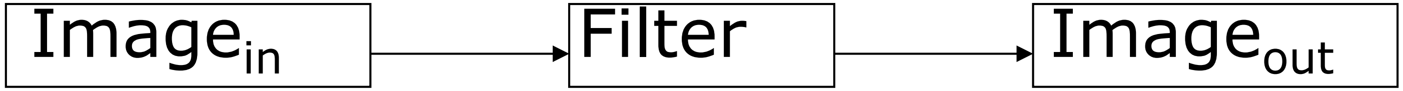
Pattern Recognition Pipeline



- The goal of pre-processing is to transform a signal f to another signal h so that the resulting signal h
  - makes subsequent processing easier
  - makes subsequent processing better (more accurate)
  - makes subsequent processing faster
- Already studied *histogram equalization, thresholding* and *smoothing.*

Filtering - revisited

- There is a family of techniques that we can apply to images, where both the input and the output to these transformations are images:

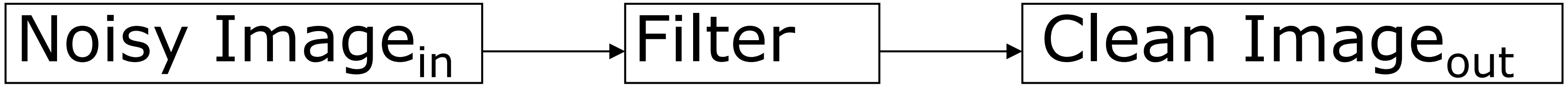


Image$_{\text{in}}$

Filter

Image$_{\text{out}}$

- We already saw one set of such filtering techniques that focus on noise reduction.



Noisy Image$_{\text{in}}$

Filter

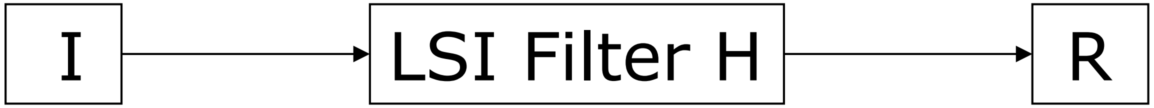
Clean Image$_{\text{out}}$

- We also said that mathematically, a filter *H* can be treated as a function on an input image I:

$$H(I) = R$$

Convolution

- If a transformation (or filter) is linear shift-invariant (LSI) then one can apply it in a systematic manner over every pixel in the image.
- **Convolution** is the process through which we apply **linear shift-invariant filters** on an image.



I

LSI Filter H

R

- Convolution is defined as:

$$R(x,y)=\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}H(x-i,y-j)I(i,j)$$

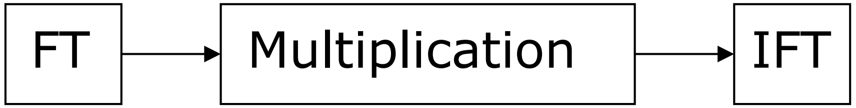
and is denoted as:

$$R=H*I$$

LSI Filtering and Convolution - Review

- We try to develop LSI filters, because we can apply them to an image through convolution.
- We have fast implementations of convolution via:
  - Its application in the frequency domain

$$F(H*I) = F(H)F(I)$$



  - Specially designed hardware that performs convolutions very fast.
- In practice, convolution can be seen as computing the weighted sum of a (2k+1)x(2k+1) neighborhood centered around pixel (x,y), where the filter H contains the applied weights.

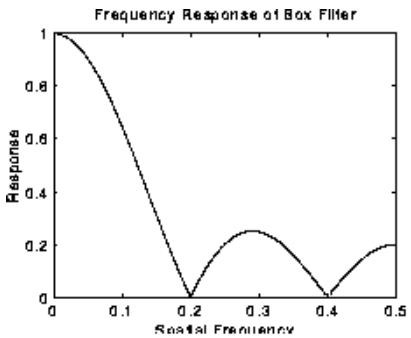
$$R(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x-i, y-j) H(i,j)$$

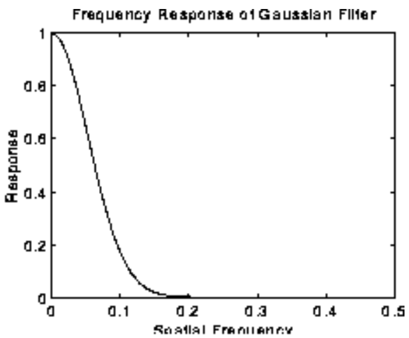
LSI Filtering and Convolution - Review

- Important Properties of Convolution:
  - commutativity, $H * I = I * H$
  - associativity, $H_1 * (H_2 * I) = (H_1 * H_2) * I$
  - distributivity, $(H_1 + H_2) * I = (H_1 * I) + (H_2 * I)$

- A very common application of filtering is for noise removal.

- Two LSI smoothing filters are:
  - Mean filter
  - Gaussian filter



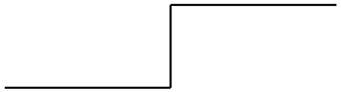


- They are also known as low-pass filters, because in the frequency domain, they allow only the transfer of the low frequency information in the output image.

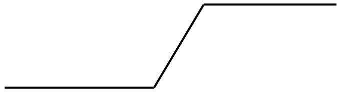
Edges

- An edge is:
  - A significant change in intensity values.
  - Related to object boundaries, patterns (brick wall), shadows, etc.
  - A property attached to each pixel.
  - Calculated using the image intensities of neighboring pixels.

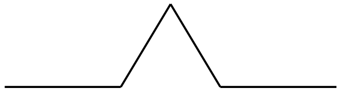
- Examples of 1D Edges



50 50 50 50 100 100 100 100 step edge



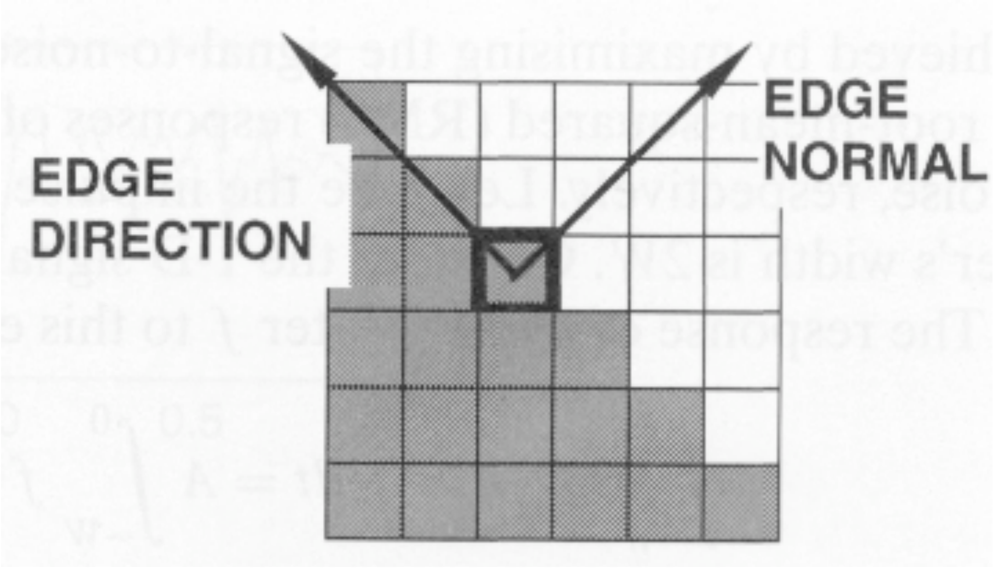
50 50 60 70 80 90 100 100 ramp edge



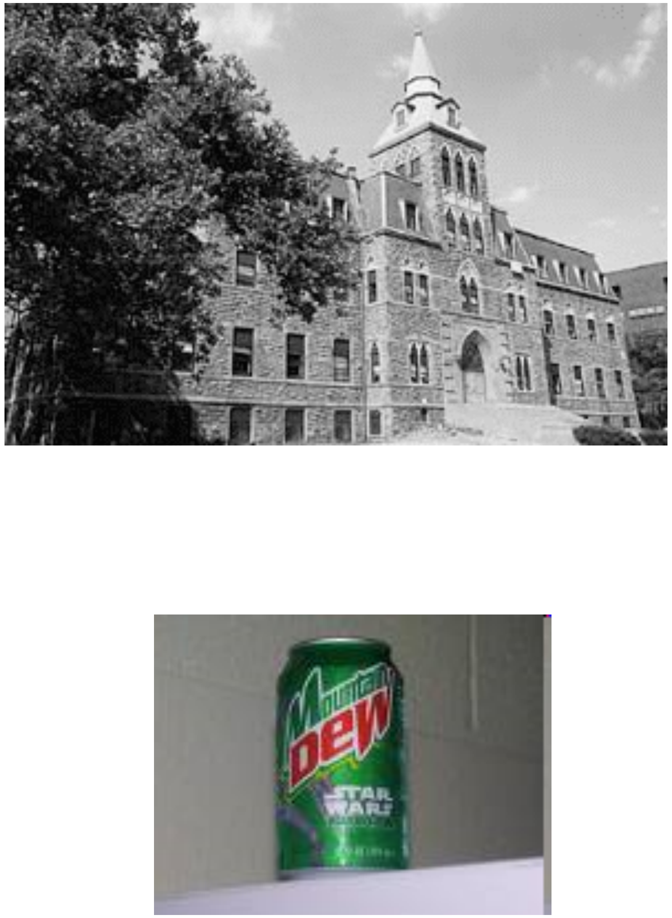
50 50 50 50 100 100 50 50 roof edge

Edges

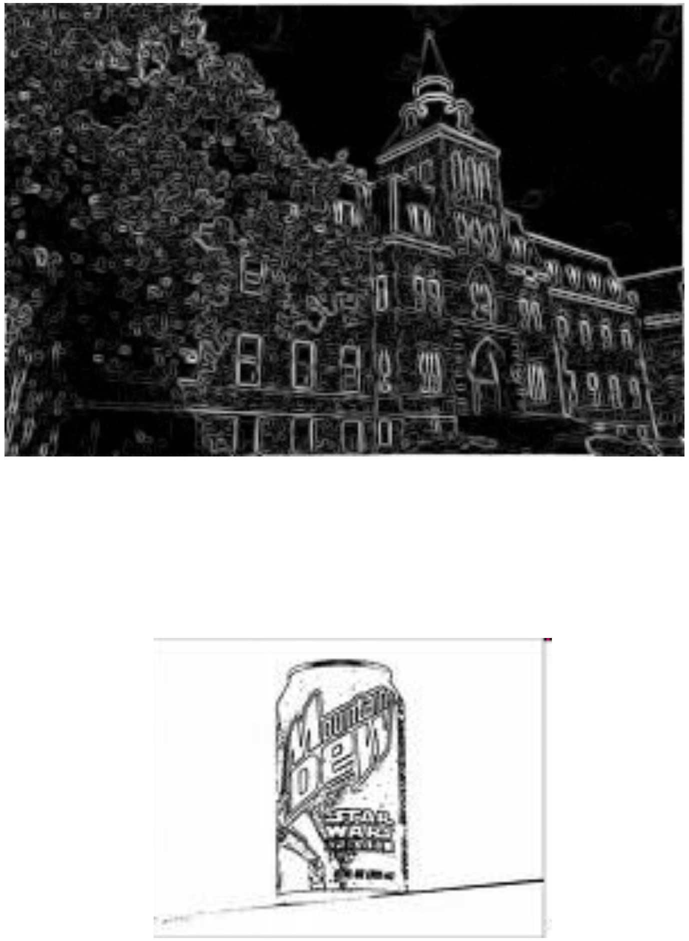
- A 2D example of an edge.



Edge Detection Example



Original images



Images after edge detection

Edge Detection Steps

1. Noise Smoothing
   - Suppress as much noise as possible without destroying edge information.
2. Edge Detection
   - Design a filter that gives high responses at edges and low response at non-edge pixels.
3. Edge Localization
   - Decide which high responses of the edge filter are responses to true edges and which ones are caused by noise or other artifacts.

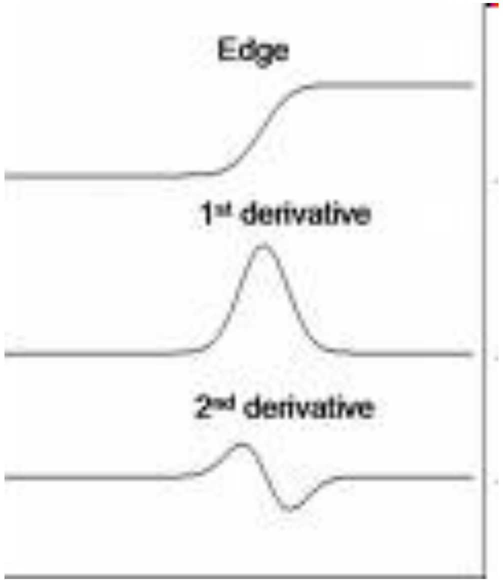
Types of Edge Detection

- Detecting edges is equivalent to detecting changes in intensity values.
- How do we detect change?

  Differentiation
- Image is a 2D function

  => partial derivative in x

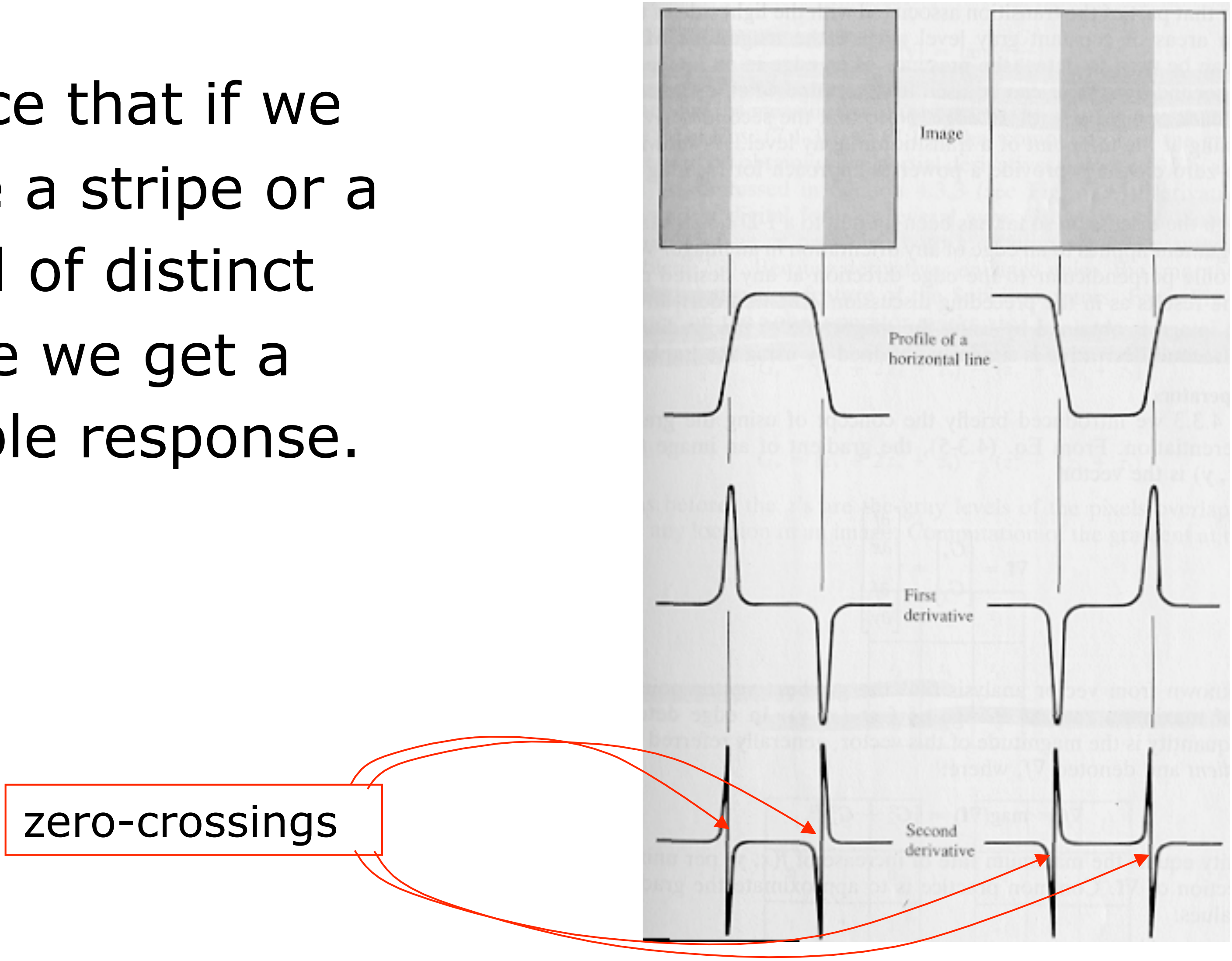
  & partial derivative in y



- If we take the 1st derivative we have **Gradient-based** edge detectors.
- If we take the 2nd derivative we have **Laplacian** edge detectors (look for zero-crossings).

Stripes and Edges

- Notice that if we have a stripe or a band of distinct value we get a double response.



Gradient-Based Edge Detection

- The gradient vector **G**(x,y), at an image pixel *I(x,y)* is:

$$\mathbf{G}(x,y)=\left(\frac{\partial I(x,y)}{\partial x},\frac{\partial I(x,y)}{\partial y}\right)=(I_x(x,y),I_y(x,y))$$

- The gradient vector points in the direction of maximum change.
- Its orientation (its angle with the x-axis) is given by:

$$\theta=\tan^{-1}\left({I_y(x,y)}\big/{I_x(x,y)}\right)$$

- Its magnitude is given by: $\left\|\mathbf{G}(x,y)\right\|=\sqrt{I_x^2(x,y)+I_y^2(x,y)}$

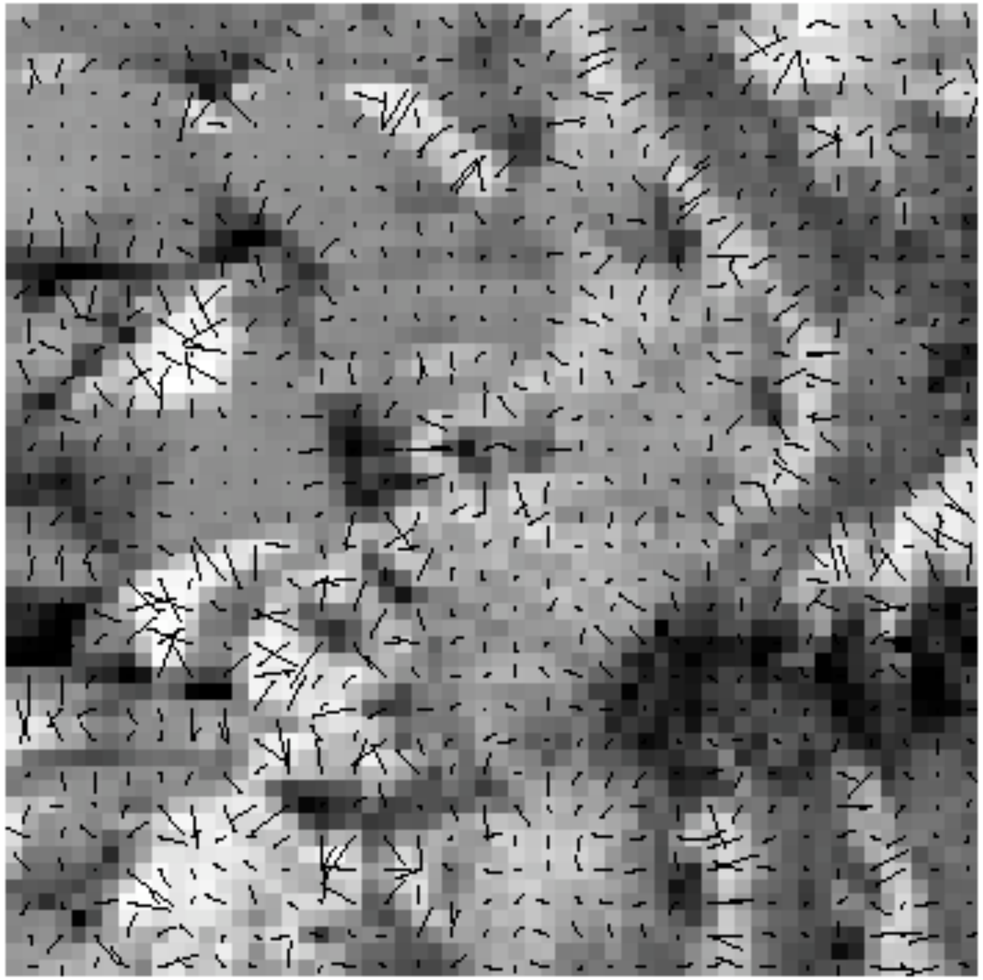
  or its approximations:

$$\left\|\mathbf{G}(x,y)\right\|\approx\left|I_x(x,y)\right|+\left|I_y(x,y)\right|$$

$$\left\|\mathbf{G}(x,y)\right\|\approx\max(I_x(x,y),I_y(x,y))$$

Gradient Vector Image

- An image showing the gradient vectors themselves.
- The length of the gradient vector corresponds to its magnitude.



Implementation

- By definition:

$$\partial I(x,y)/\partial x = \lim_{\varepsilon \to 0}\left(\frac{I(x,y)}{\varepsilon} - \frac{I(x-\varepsilon,y)}{\varepsilon}\right)$$

- In the discrete world differentiation is approximated by finite differencing:

$$I_x(x,y) = \partial I(x,y)/\partial x \approx \frac{I[x,y]-I[x-\Delta x,y]}{\Delta x}$$

- But since our smallest step is $\Delta x = 1$:

$$I_x(x,y) = \partial I(x,y)/\partial x = I[x,y]-I[x-1,y]$$
$$I_y(x,y) = \partial I(x,y)/\partial y = I[x,y]-I[x,y-1]$$

Implementation (continued)

- We can express this operation in a kernel form:

$$H_x = I_x = \begin{bmatrix} -1 & +1 \end{bmatrix} \qquad H_y = I_y = \begin{bmatrix} -1 \\ +1 \end{bmatrix}$$

- To make it less susceptible to noise we use the values of two consecutive rows or columns.

$$H_x = I_x = \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix} \qquad H_y = I_y = \begin{bmatrix} -1 & -1 \\ +1 & +1 \end{bmatrix}$$

- These kernels, however, evaluate an approximation of the derivative at half-pixel locations, $I_x[x-1/2, y]$ and $I_y[x, y-1/2]$

Common Edge Masks

- Prewitt edge detection masks

$$P_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} \qquad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

- Sobel edge detection masks

$$S_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Gradient Edge Detection Process

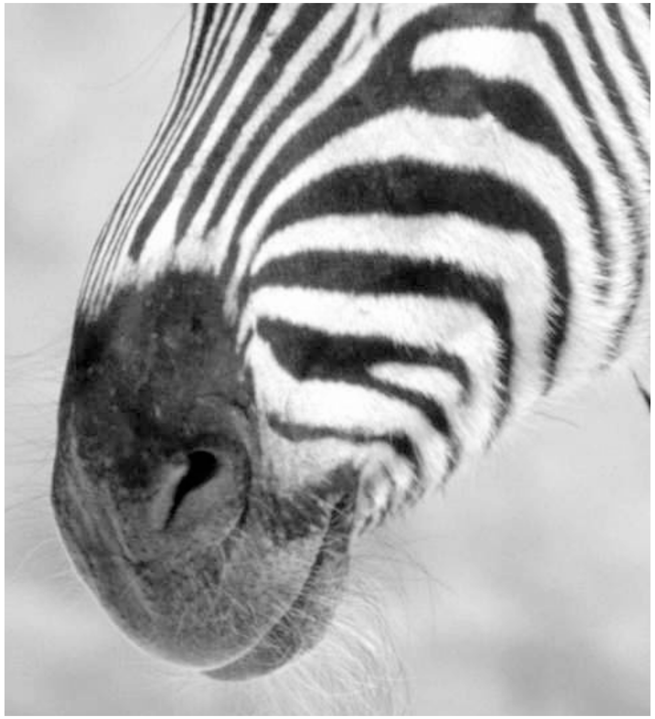
- Given an input image I, the gradient-based edges are computed as follows:

1. Compute $I_x = H_x * I$
2. Compute $I_y = H_y * I$
3. Compute $\left\| \mathbf{G}(x,y) \right\|$ using your favorite method
4. If $\left\| \mathbf{G}(x,y) \right\| \geq t$

   then pixel (x,y) is an edge-pixel (*edgel*)

   compute the angle θ for that pixel.

Gradient Edge Detector Example



Original image

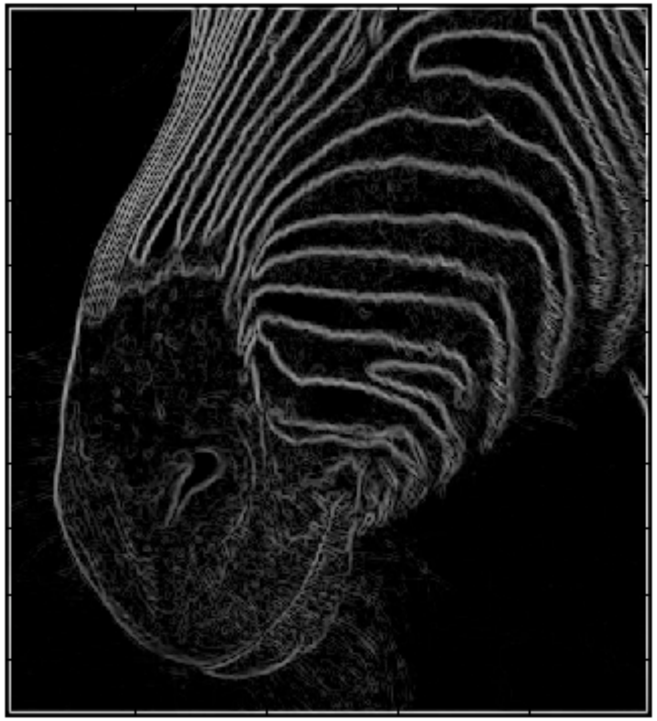


Image after edge detection

Canny Edge Detector

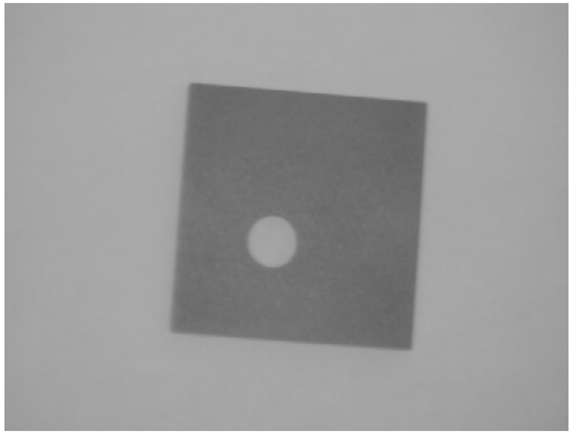
- After a a gradient-based edge image is created, the Canny method uses optimization to systematically clean noise effects. It uses two separate optimization processes:
  1. Non-maximum suppression

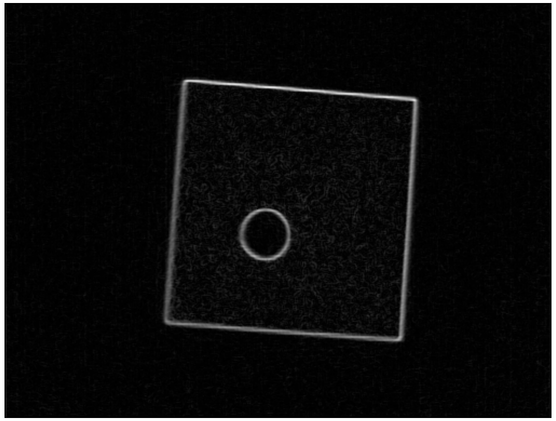
     A single real edge may appear as having wide ridges around it.

     Non-maximum suppression thins such ridges downto 1-pixel wide edges.
  2. Hysteresis thresholding

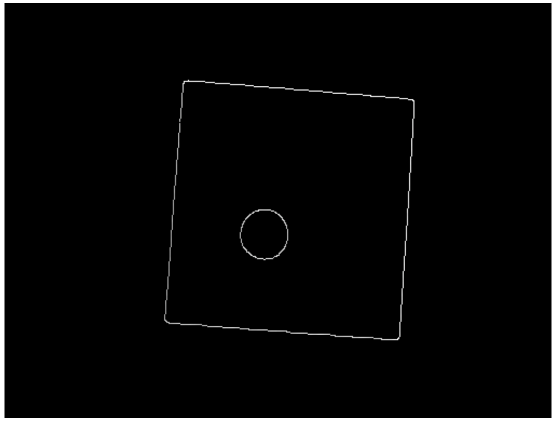
     Use a pair of threshold values. The high threshold is used as a first rough screening. For the edge pixels that survive this first screening, follow chains (contours) of edges. Use those edgels on the chain which are above the second, lower, threshold.
- Canny proved that this is the optimal edge detection method.
- Due to the optimization post-processing, it is slower than the basic gradient-based edge detectors.

Sobel versus Canny



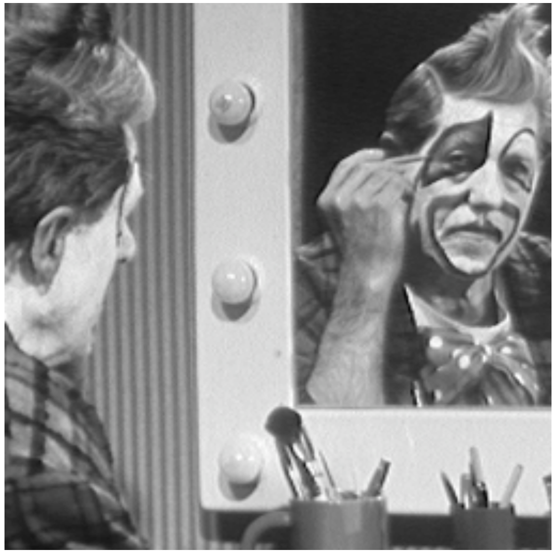


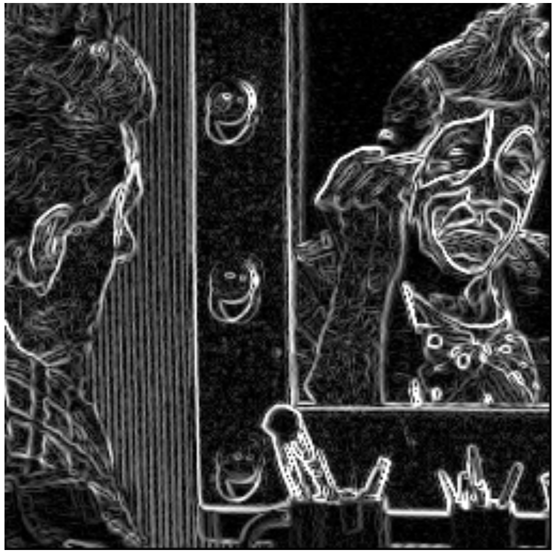
Sobel



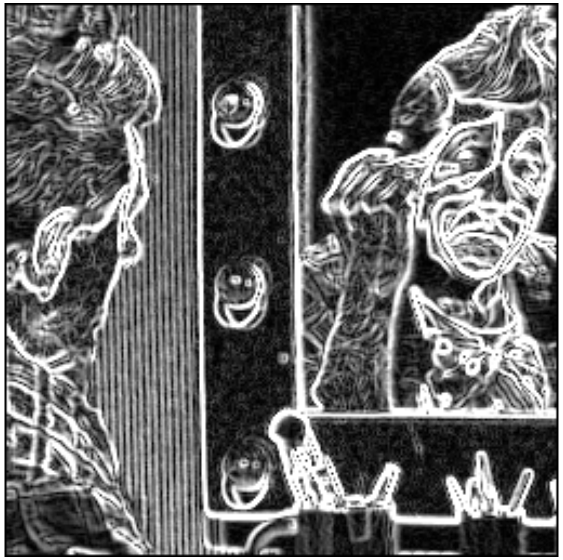
Canny

Roberts vs. Sobel



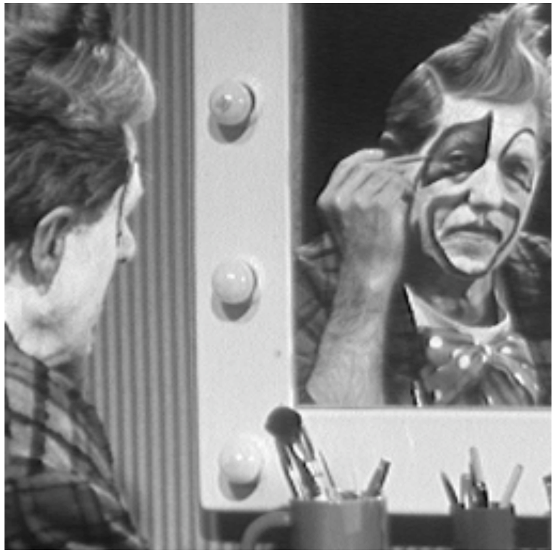


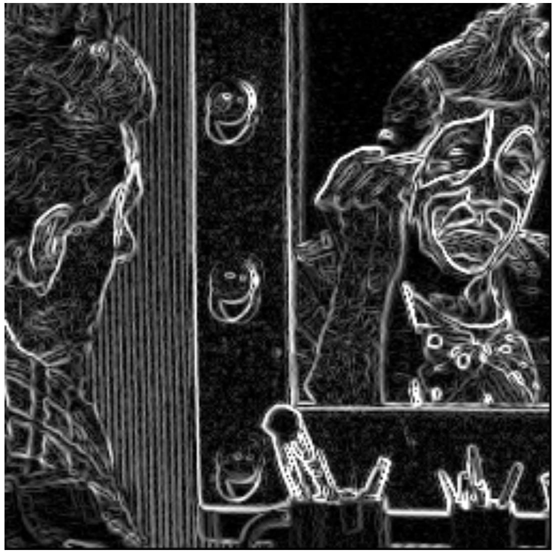
Roberts



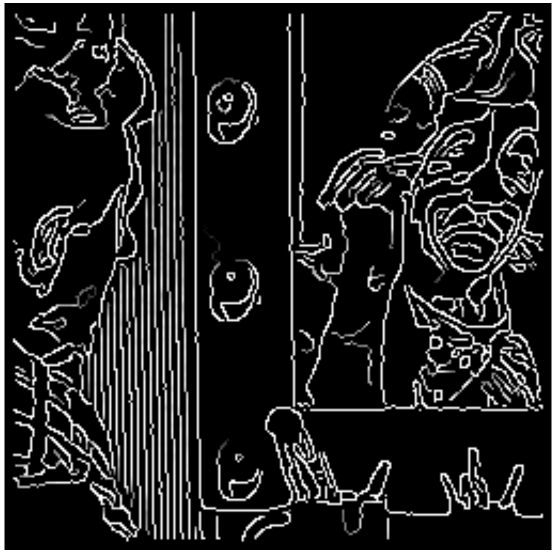
Sobel

Roberts vs. Canny





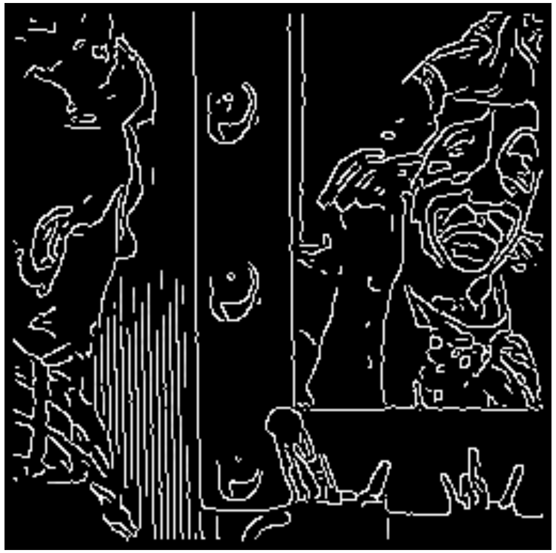
Roberts



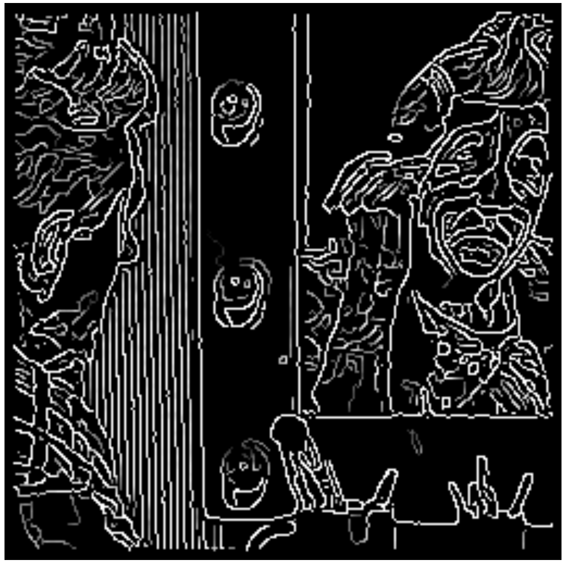
Canny

$\sigma = 1, t_l=1 , t_h= 255$

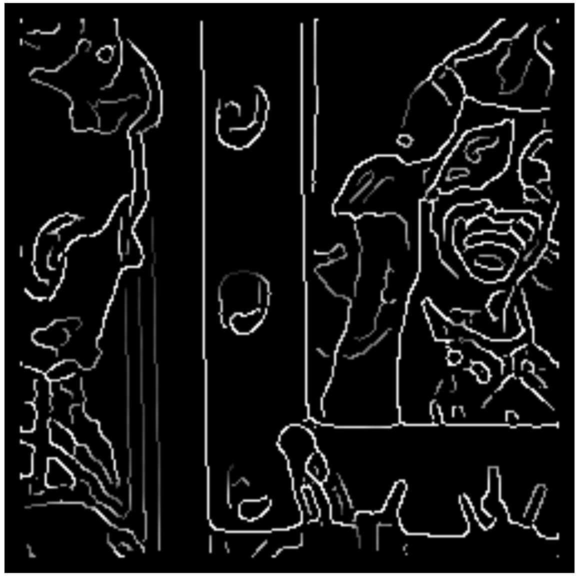
Canny Edge Detector



Canny
$\sigma = 1, t_l=220\ , t_h= 255$

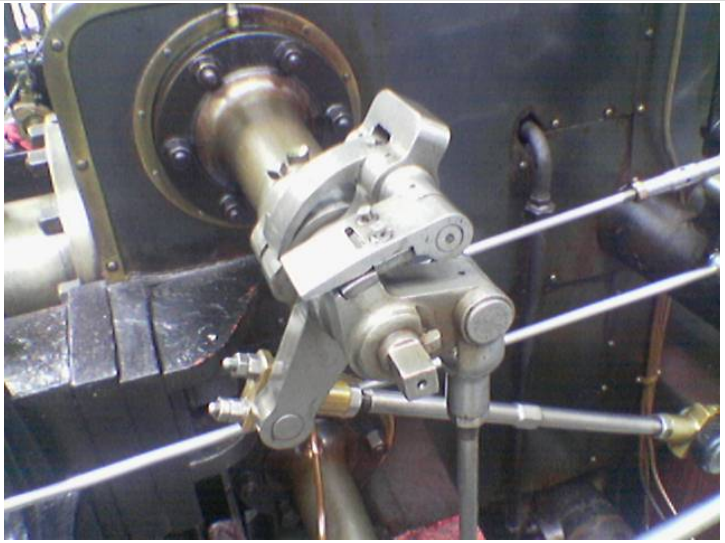


Canny
$\sigma = 1, t_l=1\ , t_h= 128$

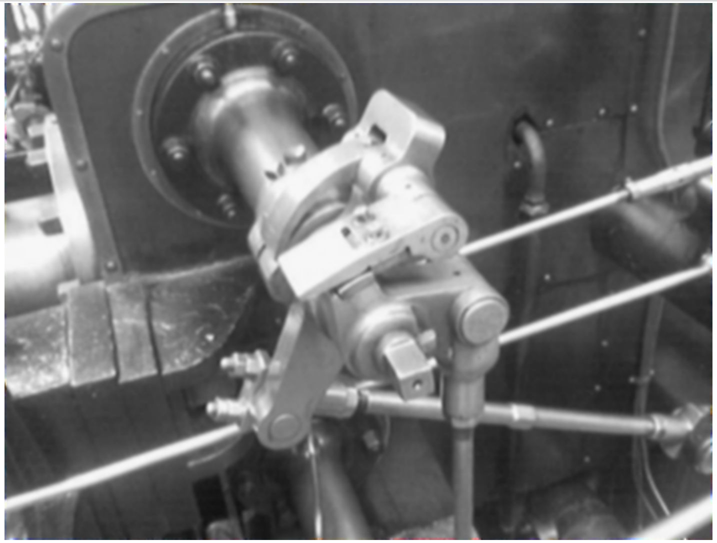


Canny
$\sigma = 2, t_l=1\ , t_h= 128$

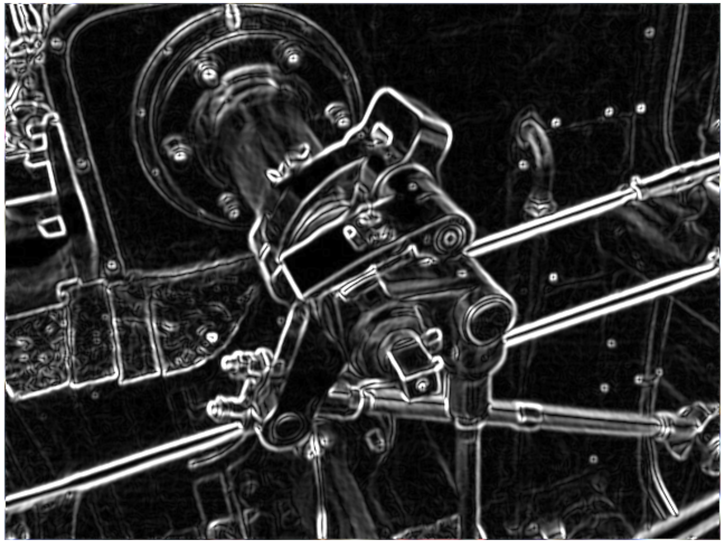
Gradient-Based Edge Detector Example



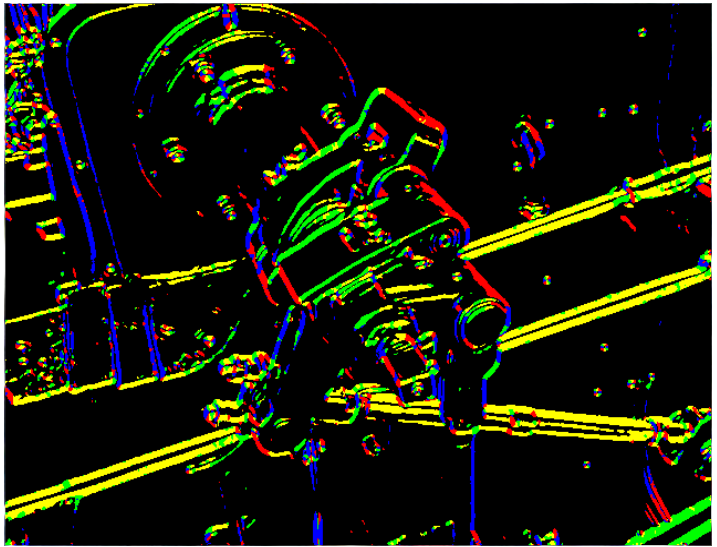
Original image



Step 1: Conversion to grayscale and smoothing with 5x5 Gaussian



Step 2: Sobel edge detector – edge magnitude image



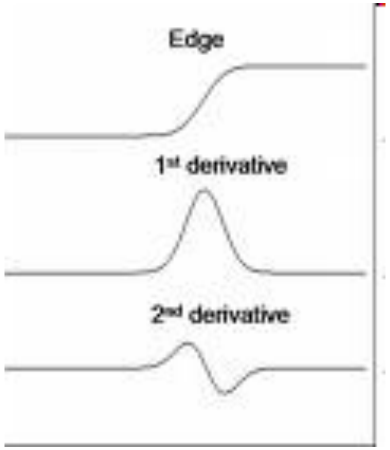
Step 2: Sobel edge detector – edge orientation image

Second Order Derivative

- Another way to detect an extremal first derivative is to look for a zero-valued 2nd derivative.

- A popular calculus tool that gives the magnitude of change in a bivariate function without direction information is the Laplacian.

$$\nabla^2(I(x,y)) = \left(\frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} \right)$$



- Note that the result of the Laplacian is a scalar.

Laplacian Implementation

- Again differentiation is approximated by finite differencing.

$$\begin{aligned}
\partial I^2(x,y)/\partial x^2 &= \partial(I_x(x,y))/\partial x \\
&= \partial(I[x,y]-I[x-1,y])/\partial x \\
&= \partial(I[x,y])/\partial x - \partial(I[x-1,y])/\partial x \\
&= (I[x+1,y]-I[x,y]) - (I[x,y]-I[x-1,y]) \\
&= I[x+1,y]-2I[x,y]+I[x-1,y]
\end{aligned}$$

- Written as a mask, we get:

$$H_x = {}^2 I_x = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Laplacian Implementation

- Similarly, for the 2nd partial derivative with respect to y, we get:

$$H_y = {}^2I_y = \begin{bmatrix} 0 & +1 & 0 \\ 0 & -2 & 0 \\ 0 & +1 & 0 \end{bmatrix}$$

- By adding the two together, we get the Laplacian mask:

$$H_{Lap} = {}^2I_x + {}^2I_y = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- If we want to use all 8 neighbors, we can use:

$$H_{Lap} = \begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

Simple Laplacian Example

- When we convolve an image that contains a significant change in values (i.e. edge) with a Laplacian kernel, we get a new image with negative values on one side of the edge and positive values on the other side of the edge.

- For example:

Input image

$$\begin{matrix} 2 & 2 & 2 & 2 & 2 & 2 & 8 & 8 & 8 & 8 \\ 2 & 2 & 2 & 2 & 2 & 2 & 8 & 8 & 8 & 8 \\ 2 & 2 & 2 & 2 & 2 & 2 & 8 & 8 & 8 & 8 \\ 2 & 2 & 2 & 2 & 2 & 2 & 8 & 8 & 8 & 8 \\ 2 & 2 & 2 & 2 & 2 & 2 & 8 & 8 & 8 & 8 \\ 2 & 2 & 2 & 2 & 2 & 2 & 8 & 8 & 8 & 8 \end{matrix}$$

Image after the Laplacian

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 6 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & -6 & 0 & 0 & 0 \end{matrix}$$

zero crossing

Laplacian of Gaussian

- The computation of 2nd order derivatives is very sensitive to noise.
- Solution: Smooth first the image I with a Gaussian H_{Gauss} and then apply the Laplacian H_{Lap} on the image.

$$R_{LapEdge} = H_{Lap} * (H_{Gauss} * I)$$

- Convolution is associative.

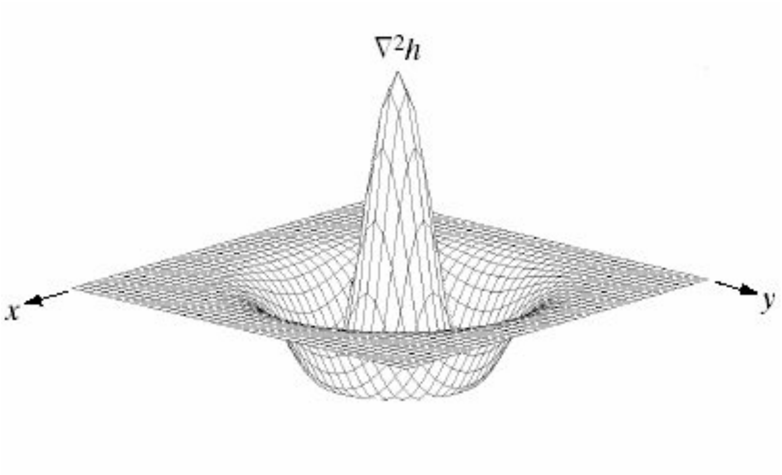
$$R_{LapEdge} = (H_{Lap} * H_{Gauss}) * I$$

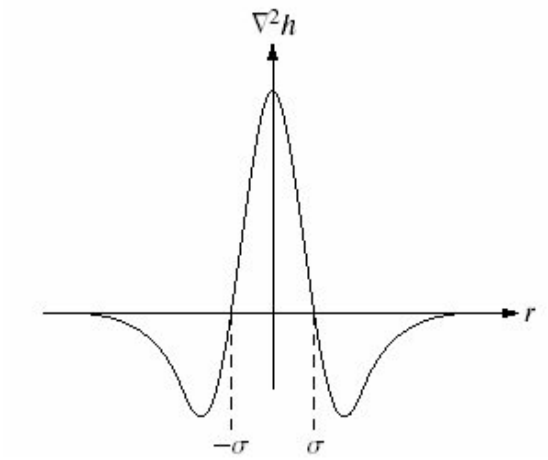
- The combined filter (H_{Lap} * H_{Gauss}) is nothing more than computing the Laplacian of the Gaussian (LoG):

$$\begin{aligned} \nabla^2(G_{auss}(x,y)) &= \nabla^2(\mathrm{e}^{(-(x^2+y^2)/2\sigma^2})) \\ &= \frac{(x^2+y^2-\sigma^2)}{\sigma^4}(\mathrm{e}^{(-(x^2+y^2)/2\sigma^2})) \end{aligned}$$

LoG Kernel

- The LoG function, $\nabla^2(G_{auss}(x,y))$ looks like a “mexican hat”.

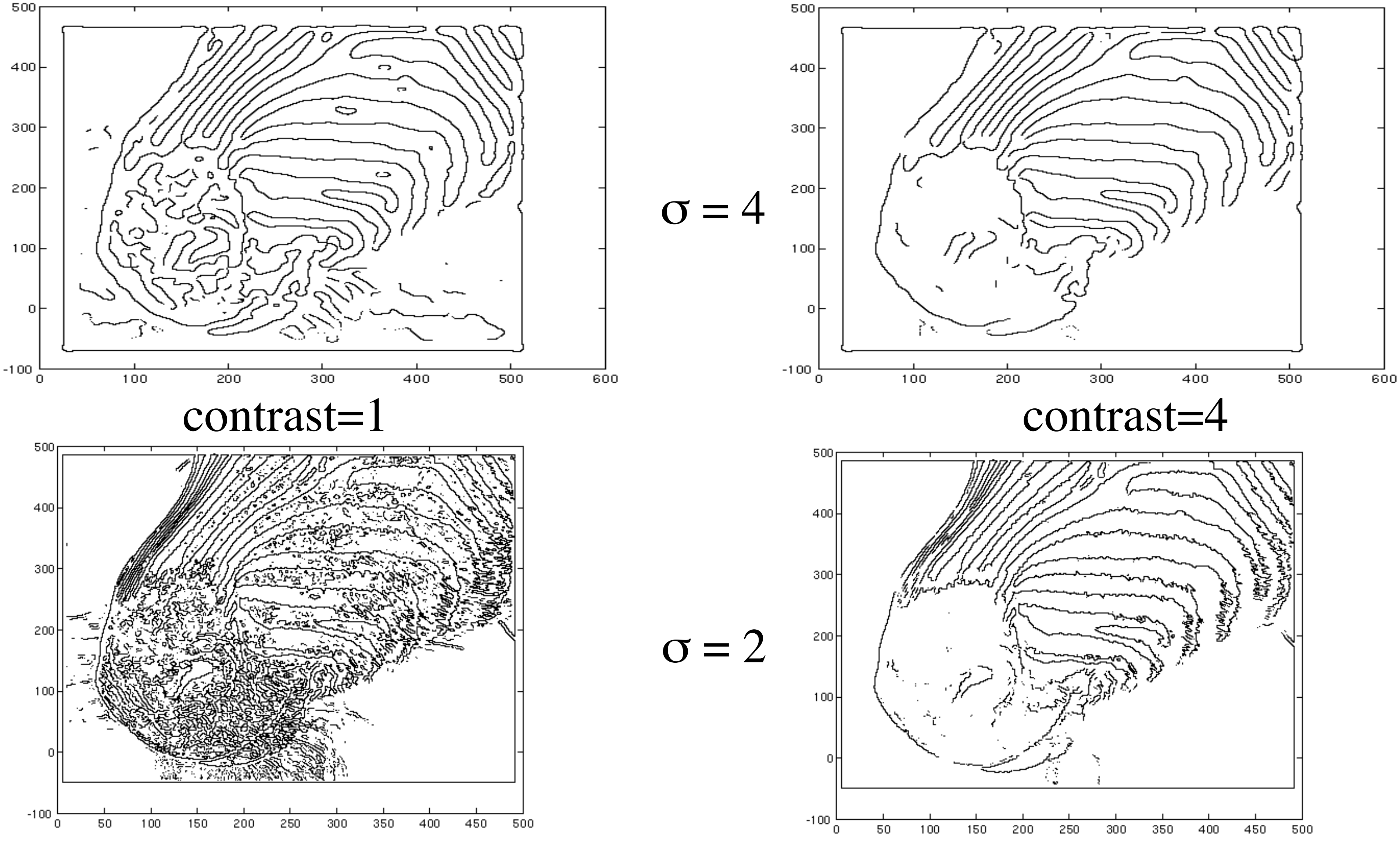




- $\nabla^2(G_{auss}(x,y))$ can also be approximated by a convolution kernel:

$$H_{LoG} = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Examples of LoG Zero Crossings



$\sigma = 4$

contrast=1

contrast=4

$\sigma = 2$

Smoothing and Differentiation

- The concepts of first smoothing and then differentiating generalizes to all edge detection methods (both 1st and 2nd order derivative methods).

- Convolution is associative, so we can always create a combined filter and convolve (filter) the image only once.

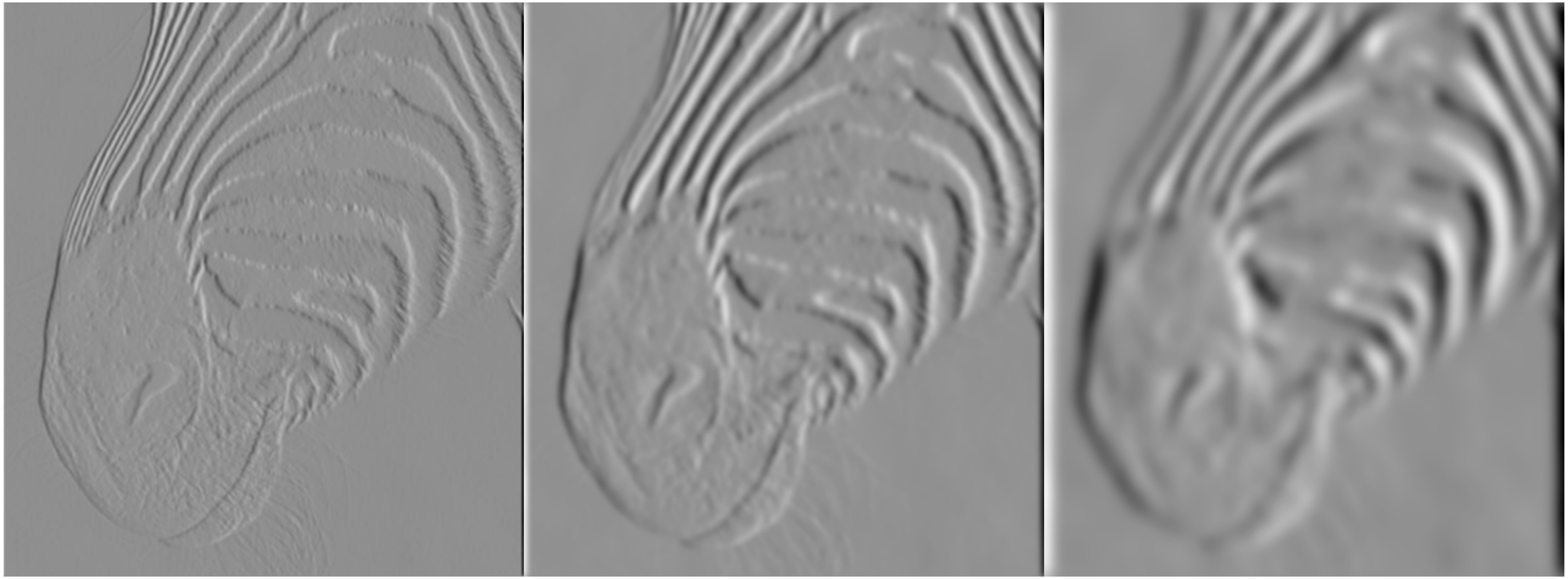
$$R = H_{edge} * (H_{smooth} * I) = (H_{edge} * H_{smooth}) * I = H * I$$

where $H = H_{edge} * H_{smooth}$

- By using different degrees of smoothing (Gaussian with different σ values or mean filters of different sizes, i.e. 3x3, 5x5, 7x7, etc.) we can obtain a hierarchy, a pyramid, of images with different levels of detail.

Different Scales

- The scale of the smoothing filter affects the derivative estimates as well as the semantics of the recovered edges

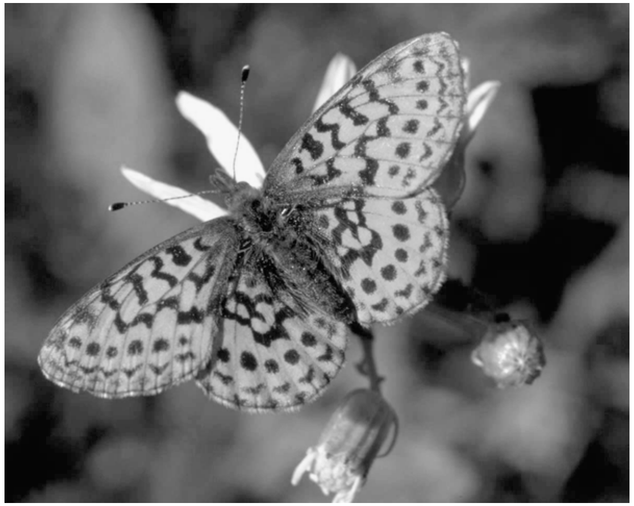


No smoothing

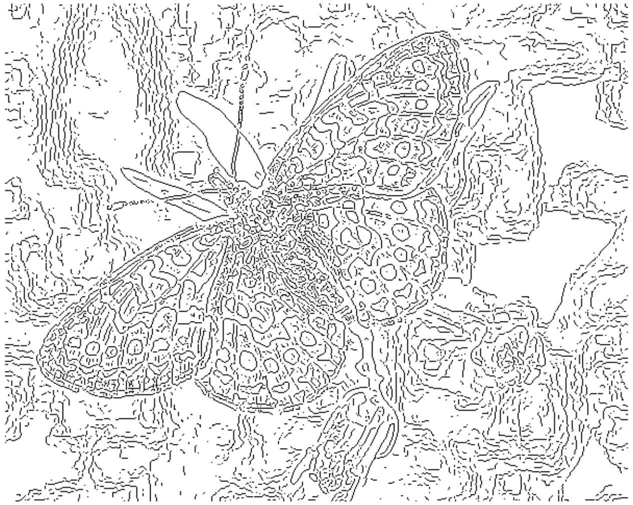
3x3 filter

7x7 filter

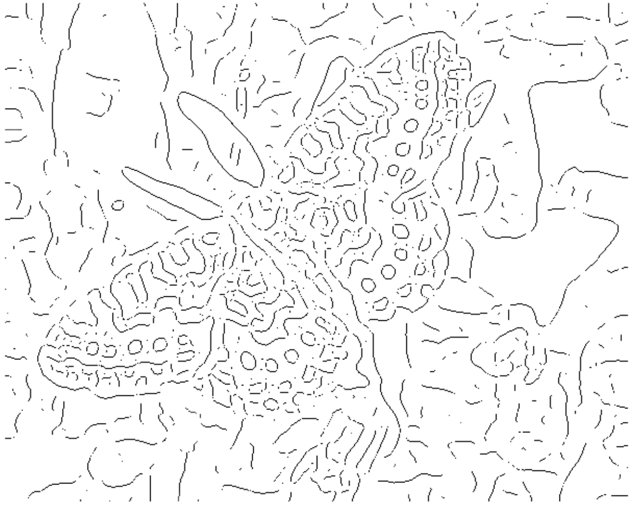
Different Scales



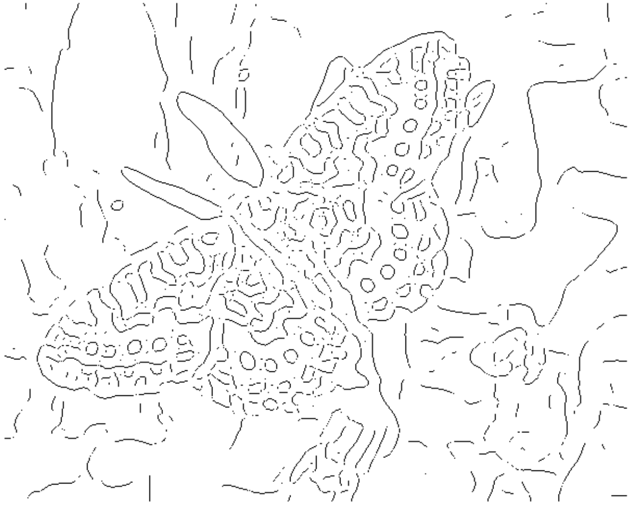
Original image



Fine scale, high threshold



Coarse scale, low threshold



Coarse scale, high threshold

Comments on Filtering

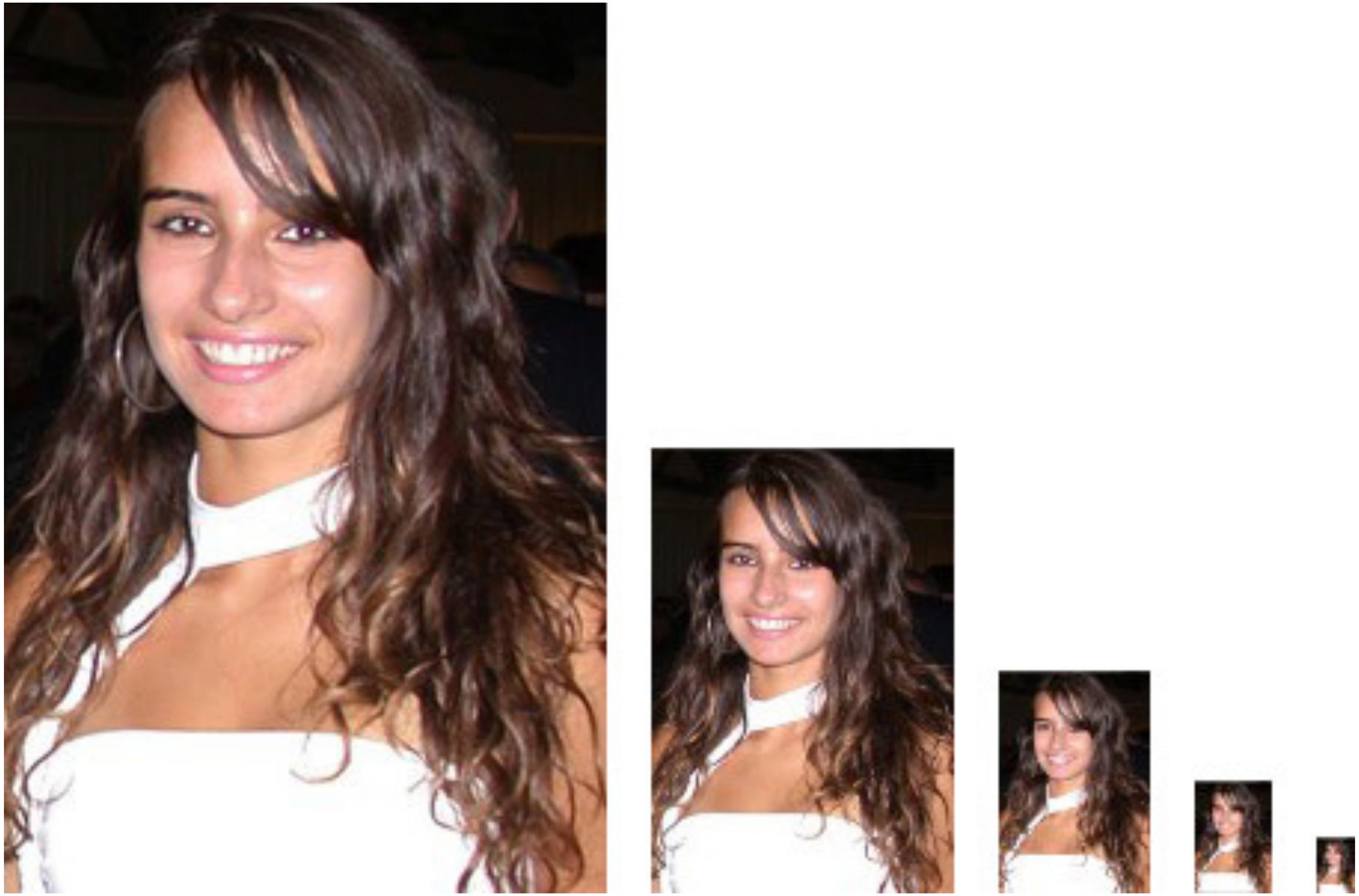
- Design Decisions:
  - Size of filter. There is no single good size. It depends on he size of the objects in the image.
  - Speed versus accuracy: (Gaussian vs. Median, Gradient-based vs. Laplacian-based, Canny vs. Sobel)
- Systematic approach: try different resolutions
  - Either create a formal model for each resolution and study the change of the model at different resolutions.
  - Or maintain a tree (pyramid) of images at different resolutions.
- Multi-resolution example:

Apply an edge detector at different resolutions of Gaussians.

Perform numerical optimization to find the best response for the particular image.

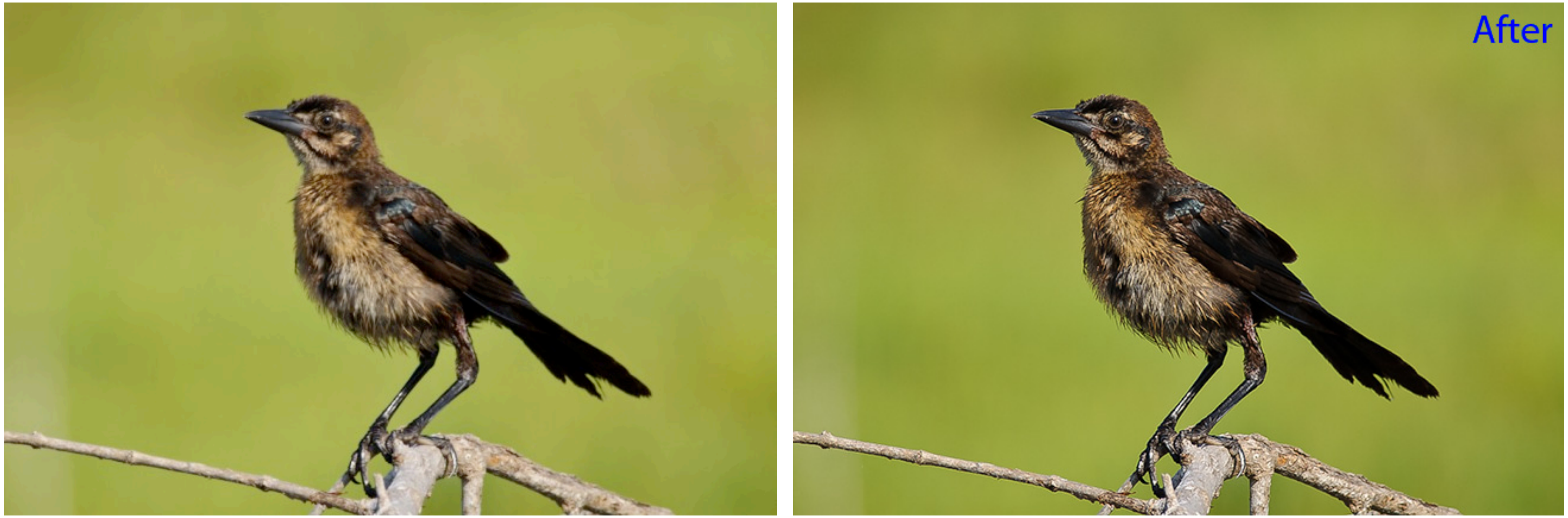
Optimal for edges corrupted by white noise.

Gaussian Pyramid Example



Sharpening

- A very common filtering operation for contrast enhancement in images is *image sharpening*.
- The goal of image sharpening is to produce a more visually pleasing image:
  - Texture and finer details are made more prominent
  - The image looks sharper, crisper.



Sharpening - continued

- Image sharpening almost always involves improving the parts of the image where a sudden change in intensity or color occur, since this is where inaccuracies are introduced by the digital data capturing process.
- What filtering operation do we know that gives a high response at sudden changes in intensity or color?
- Edge Detector, H_{edge}
- A simple way to achieve sharpening is to superimpose the original image with the magnitude of the edge image.

$$R = I + c(I * H_{edge})$$

UnSharp Mask

- Most image processing software packets perform sharpening using the UnSharp Mask (USM).
- It is based on an old photographic film technique.
- It is called unsharp masking, because it first blurs the image (unsharpens it)

$$R_1 = I * H_{smooth}$$

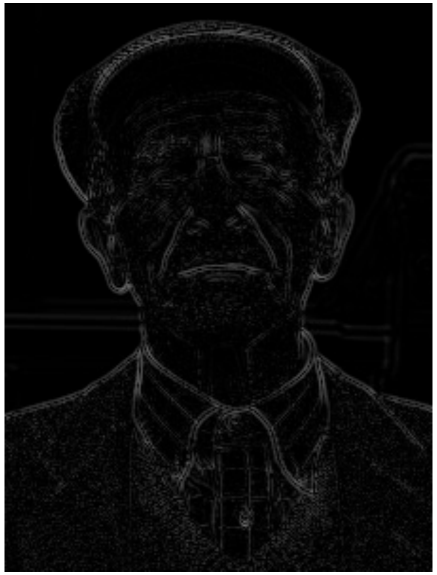
- An unsharp mask, UM, for the entire image is created by thresholding the absolute difference of the original and the blurred image.

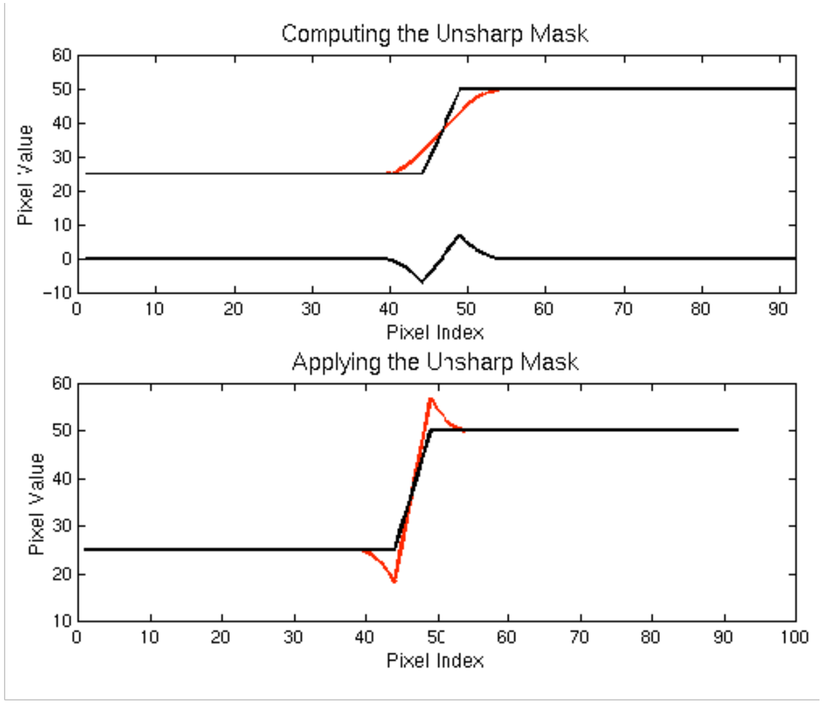
$$UM(x,y) = \begin{cases} 1 & \text{if } \left| I(x,y) - R_1(x,y) \right| > \theta \\ 0 & \text{otherwise} \end{cases}$$

UnSharp Mask - continued

- The unsharp mask is then scaled (to achieve the desired visual effect) and added to the original image. The scaling factor *c* is often called *amount*.

$$R_2 = I + cUM$$





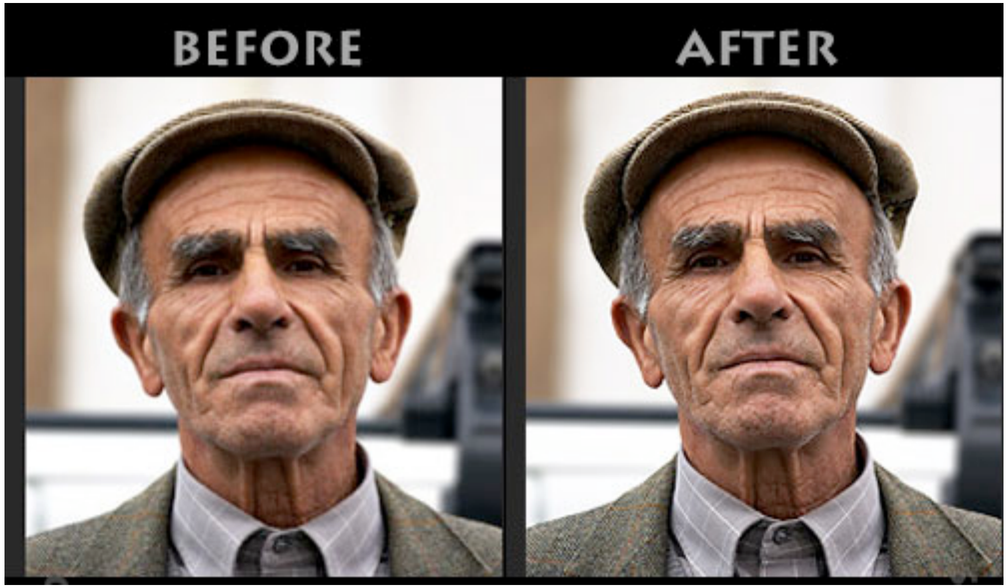


Image Sources

1. “Image with salt & pepper noise”, Marko Meza.
2. “Set of images of Roberts vs. Canny vs. Sobel”, Hypermedia Image Processing Reference at the University of Edinburgh.
3. “LoG plots”, Simon Yu Ming, http://hi.baidu.com/simonyuee/blog/item/446a911bf43cc91c8618bf8f.html
4. Many of the smoothing and edge detection images are from the slides by D.A. Forsyth, University of California at Urbana-Champaign.
5. The bird sharpening example was done using Adobe Photoshop Lightroom, http://mansurovs.com/how-to-properly-sharpen-images-in-lightroom.
6. The unsharp mask example is copyrighted by Sean T. McHugh, http://www.cambridgeincolour.com/tutorials/unsharp-mask.htm