Edge Detection

Prof. Dr. Elli Angelopoulou

Pattern Recognition Lab (Computer Science 5)

University of Erlangen-Nuremberg

Filtering

- Most of the images we capture are noisy
- Goal:

$$\boxed{\text{Noisy Image}_{in}} \longrightarrow \boxed{\text{Filter}} \longrightarrow \boxed{\text{Clean Image}_{out}}$$

- This notion of filtering is more general and can be used in a wide range of transformations that we may want to apply to images.

$$\boxed{\text{Image}_{in}} \longrightarrow \boxed{\text{Filter}} \longrightarrow \boxed{\text{Image}_{out}}$$

- Mathematically, a filter H can be treated as a function on an input image I:

$$H(I) = R$$

- Note: We use the terms *filter* and *transformation* interchangeably

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.

$$\boxed{I} \longrightarrow \boxed{\text{LSI Filter H}} \longrightarrow \boxed{R}$$

- 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:

$$R = H * I$$

Another Look at Convolution

- Filtering often involves replacing the value of a pixel in the input image *F* with the weighted sum of its neighbors.
- Represent these weights as an image, ***H***
- ***H*** is usually called the **kernel**
- The operation for computing this weighted sum is called **convolution.**

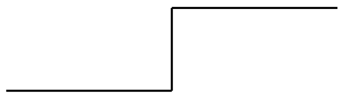
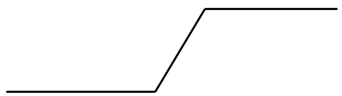
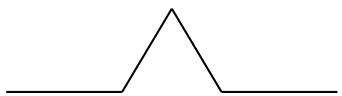
$$R = H * I$$

- Convolution is:
  - commutative, $H * I = I * H$
  - associative, $H_1 * (H_2 * I) = (H_1 * H_2) * I$
  - distributive, $(H_1 + H_2) * I = (H_1 * I) + (H_2 * I)$

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

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 Enhancement
   - 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.

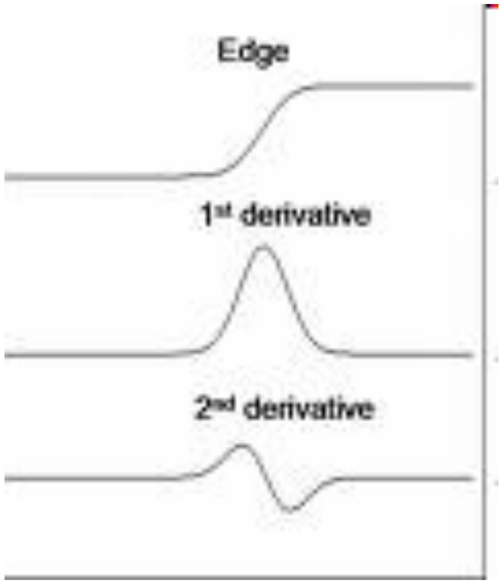
Types of Edge Detection

- 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).

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: $\|\mathbf{G}(x,y)\| = \sqrt{I_x^2(x,y) + I_y^2(x,y)}$

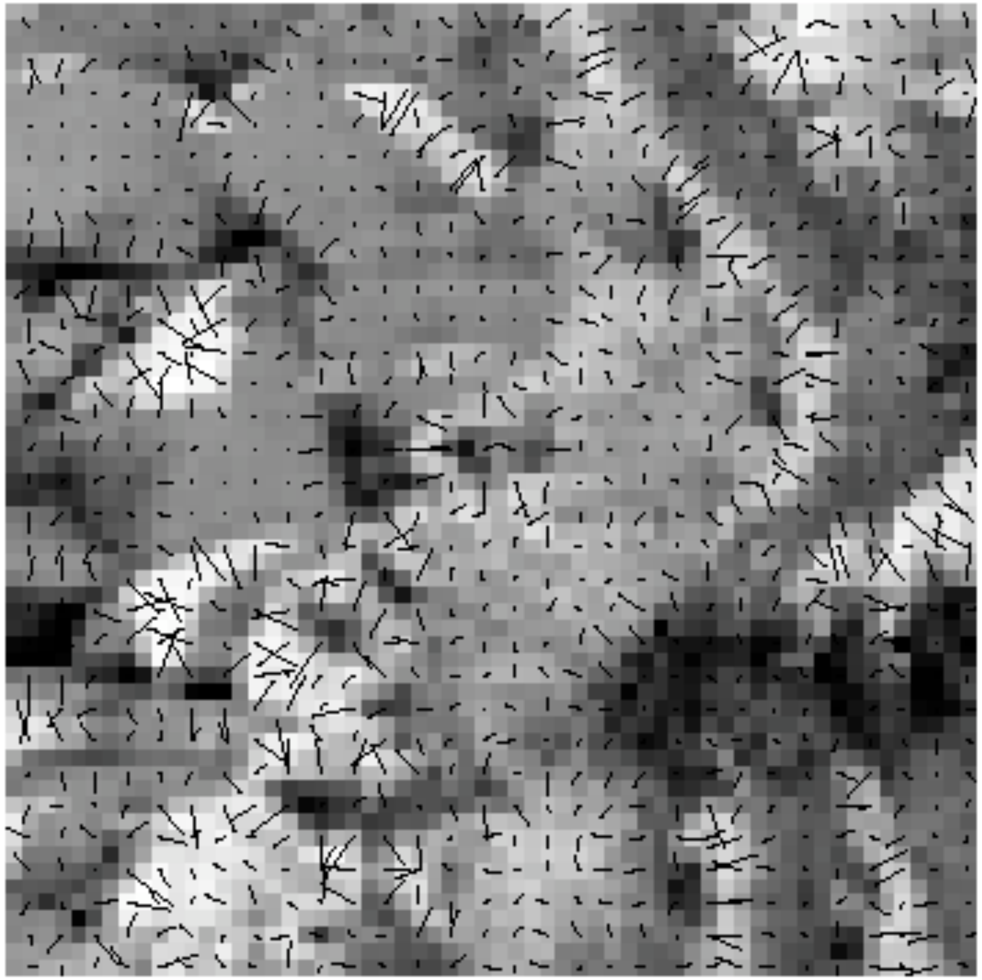
  or its approximations:

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

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

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:

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

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

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

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

Roberts Edge Detector

- To overcome this unbalanced “half-pixel” location problem, Roberts suggested two other masks (kernels) for edge detection:

$$H_{Rx} = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \qquad H_{Ry} = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

- These kernels give the maximal response to edges that run at 45° angles to the pixel grid.

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

Roberts Cross Operator

- By convolving an image with H_{Rx} and H_{Ry} one obtains estimates of the gradient:

$$I_x = H_{Rx} * I$$

$$I_y = H_{Ry} * I$$

$$\mathbf{G}(x,y) = (I_x(x,y), I_y(x,y))$$

- The edge orientation for the Roberts edge detector is given by:

$$\theta = \tan^{-1}\left(I_y(x,y) \big/ I_x(x,y) \right) + \tfrac{1}{4}\pi$$

- Its magnitude is given by:

$$\|\mathbf{G}(x,y)\| = \sqrt{I_x^2(x,y) + I_y^2(x,y)}$$

or its approximations:

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

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

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 $\|\mathbf{G}(x, y)\|$ using your favorite method
4. If $\|\mathbf{G}(x, y)\| \geq t$

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

   compute the angle θ for that pixel.

Gradient Edge Detector Example

Original image

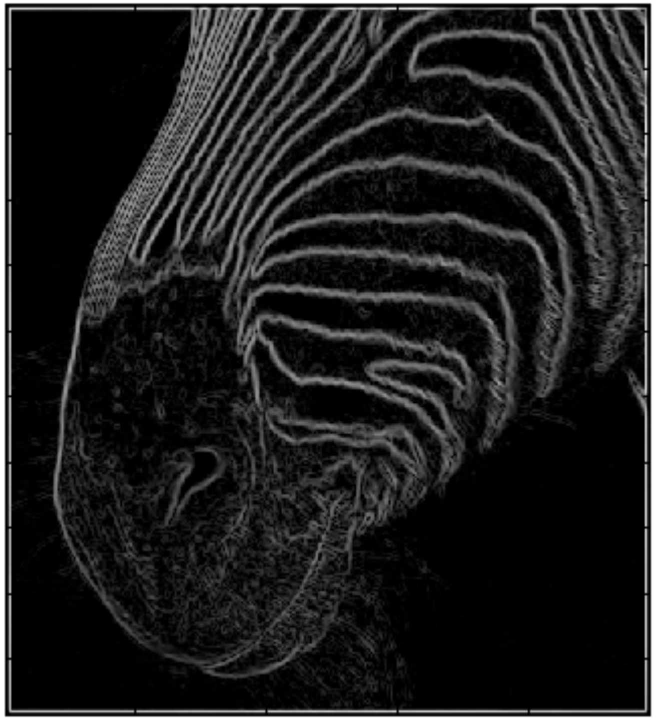


Image after edge detection

Canny Edge Detector

- It is a multi-stage (multi-pass) edge detector.
- It studies the effects of noise in a systematic way.
- Developed in 1986 by Canny as an optimal edge detector.
- The original work includes:
  - a detailed description of how and why edge detection works.
  - a proof of optimality
- It is based on gradient edge detection.

Optimality Criteria

- According to Canny, an "optimal" edge detector should satisfy the following optimality criteria.

1. Good detection: find as many **real** edges as possible
2. Good localization: estimate the position of the edge as close as possible to its true location in the image.
3. Minimal (Single) response: detect each edge only once (no ghost or ringing effects).

Good Detection

- In order to find as many real edges as possible, one needs to minimize the probability of:
  - False positives (detection of spurious edges caused by noise)
  - Missed real edges
- This means that one needs to maximize the Signal to Noise Ratio (SNR).
- Let $H(x)$ be the filter, $n(x)$ be the Gaussian noise with mean n_0 and $I(x)$ be the input signal (image).
- An image with a single ideal step edge, would be:

$$I(x) = \begin{cases} 0 & x < 0 \\ A & x \geq 0 \end{cases}$$

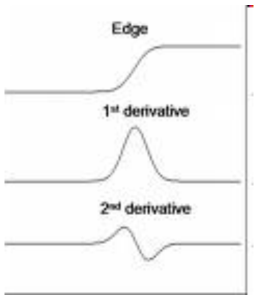
SNR

- The edge response is: $$R_e(x) = \int_{-k}^{k} I(-x)H(x)dx = A\int_{-k}^{0} H(x)dx$$
- The noise RMS response is: $$RMS_n(x) = n_0\sqrt{\int_{-k}^{k} H^2(x)dx}$$
- The corresponding SNR is: $$SNR = \frac{A\left\|\int_{-k}^{0} H(x)dx\right\|}{n_0\sqrt{\int_{-k}^{k} H^2(x)dx}}$$
- Thus, a filter which satisfies the good-detection criterion should maximize this SNR.

Good Localization

- The goal is to have the location of the detected edges as close as possible to the true edges.
- Where is the edge localized? At the maximum of the filter response.

$$R'_e(x) + R'_n(x) = 0$$



- Thus, in a similar manner to good detection, we want to maximize:

$$LOC = \frac{A\left\|H'(0)\right\|}{n_0\sqrt{\int_{-k}^{k} H'^2(x)dx}}$$

Minimal Response

- The edge detector should return only one pixel for each true edge point.
- Consider a true edge at a pixel p surrounded by noise edges.
- Idea: Discard edges that are within some small distance d from another edge.
- How can this be done in practice? Use a large edge kernel (not 3x3 but 11x11 for example).
- Ooops!!! Large kernels are bad for localization.
- Optimization: Maximize SNR and LOC subject to the single response constraint.

Non-Maximum Suppression

- A single real edge may appear as having wide ridges around.
- Non-maximum suppression thins such ridges down-to 1-pixel wide edges.
- Non-maximum suppression requires two input images:
  - the edge strength image E_s
  - the edge orientation image E_o
- Possible edge orientations are quantized to:

$$d_1 = 0^\circ \quad d_2 = 45^\circ \quad d_3 = 90^\circ \quad d_4 = 135^\circ$$

Non-Maximum Suppression Algorithm

For each pixel (i,j)

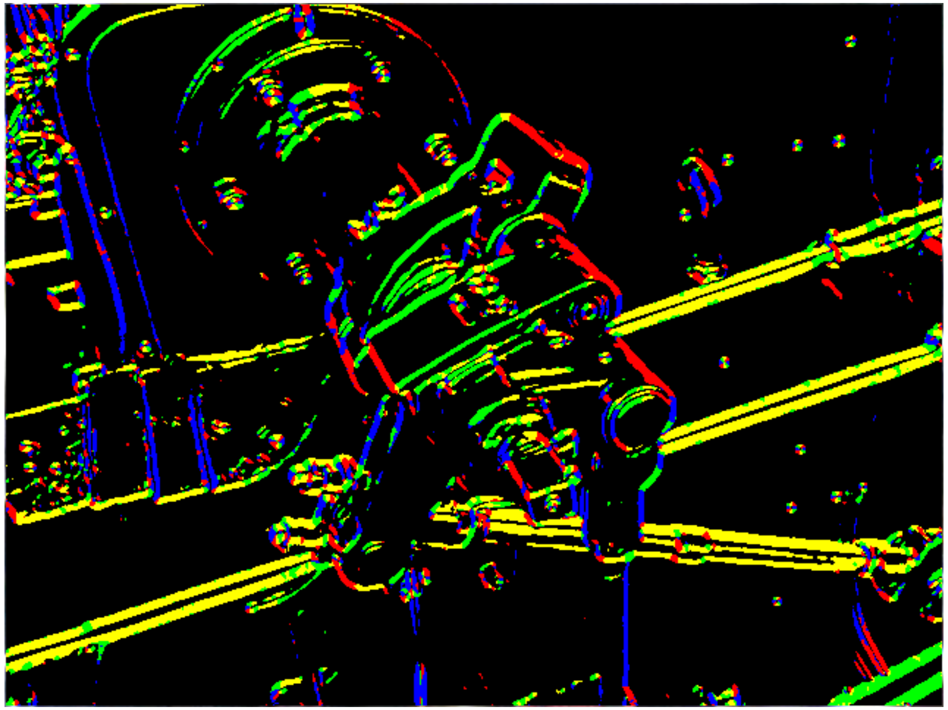
1. Find the d_k that best approximates $E_o(i,j)$
2. Examine the two neighbors $n_1(i,j)$ and $n_2(i,j)$ along the direction d_k .
3. If $(E_s(i,j) < E_s(n_1(i,j)))$ or $(E_s(i,j) < E_s(n_2(i,j)))$

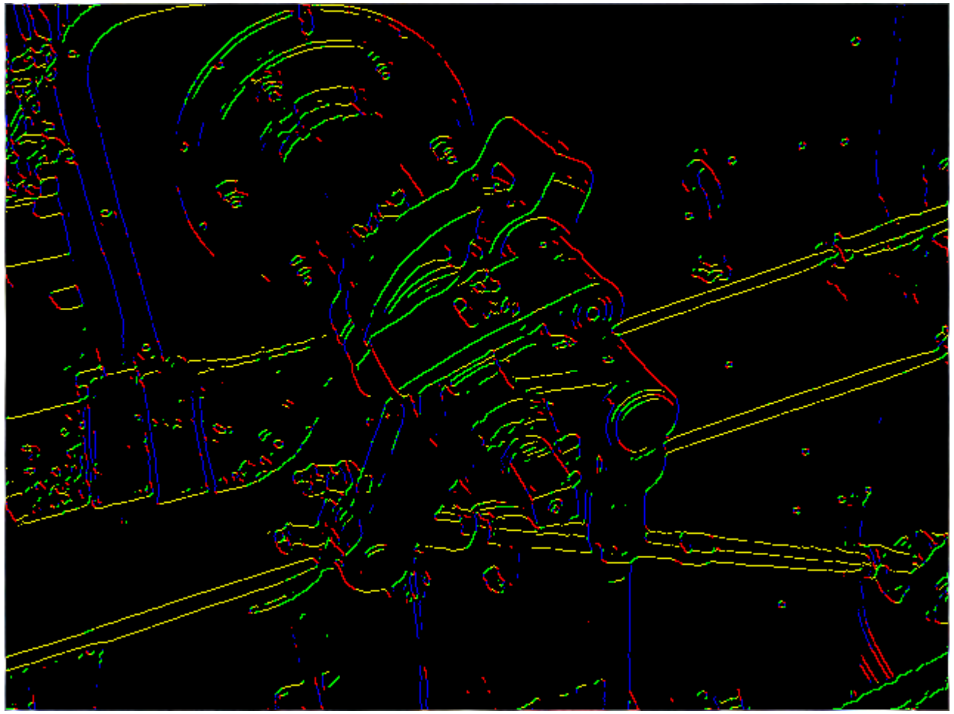
$$I_N(i,j) = 0$$

else

$$I_N(i,j) = E_s(i,j)$$

Non-Maximum Suppression Example





Edges are quantized to 0° (yellow), 45° (green), 90° (blue), 135° (red).
Non-maximum suppression addresses the minimal response criterion.

Hysteresis Thresholding

- Non-maximum suppression examines parallel edges in small neighborhood and eliminates the ones with the smaller (not max.) gradient magnitude.
- Even after non-maximum suppression, I_N still contains edges that are just responses to noise.
- Use thresholding to eliminate noisy responses.
- What threshold value?
  - too low: not all the noise is eliminated
  - too high: real edge are removed
- Canny’s solution: Use 2 thresholds

Main Idea of Hysteresis Thresholding

- Assumption: Important edges should form continuous curves in the image.
- Idea: Follow a faint direction of a given line and discard a few noisy pixels that do not constitute a line but have produced large gradients.
- Do this by using a **high** threshold.
- After the high thresholding we are left with edges which are most probably real edges.
- Do a 2^{nd} pass tracing (following) the curves. During the tracing use the **lower** threshold. If an edge strength is larger than the lower threshold it is a real edge.

Hysteresis Thresholding Algorithm

Let t_l and t_h be the low threshold and high threshold values, respectively.

For each non-zero pixel (i,j) in I_N and scanning in a fixed order (i.e. follow a contour in a clockwise manner)

1. Locate the next unvisited pixel $I_N(i,j)$ such that $I_N(i,j) > t_h$
2. Start from $I_N(i,j)$.

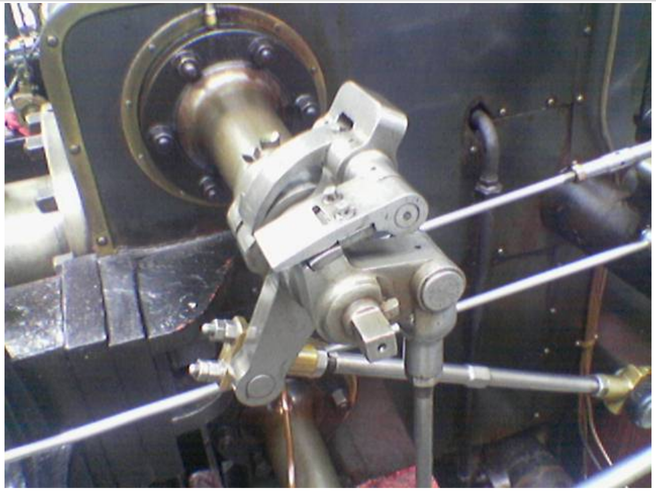
   Follow the chains of connected I_N pixels in both directions perpendicular to the edge gradient, as long as $I_N(i,j) > t_l$.

   Mark each such I_N as visited.

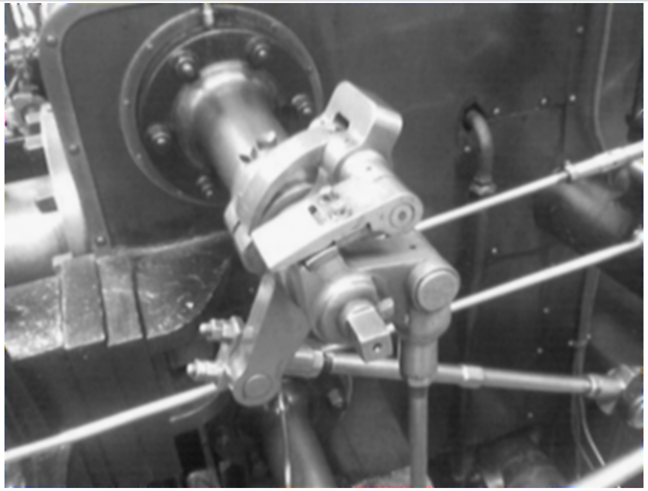
   Save all such visited pixel locations in a list that represents a connected contour.
3. Create a new output image I_H. Set all pixels to 0.
4. Traverse each contour list.

   As you traverse the contour list, set each pixel location (i,j) on that list to 1, i.e. $I_H(i,j) = 0$, or to the edge strength $I_H(i,j) = E_S(i,j)$

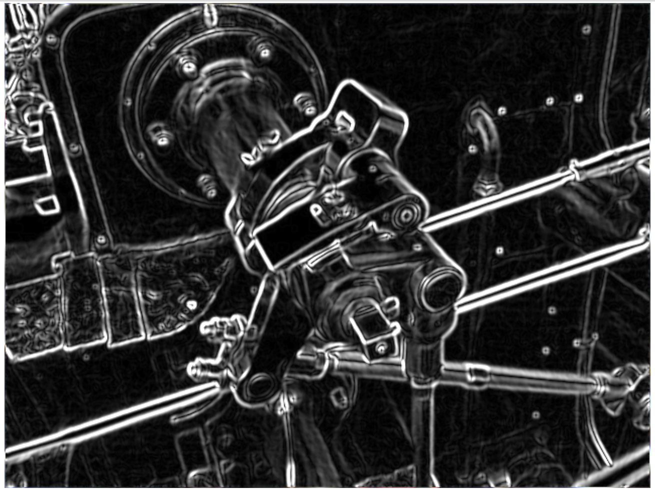
Canny Example



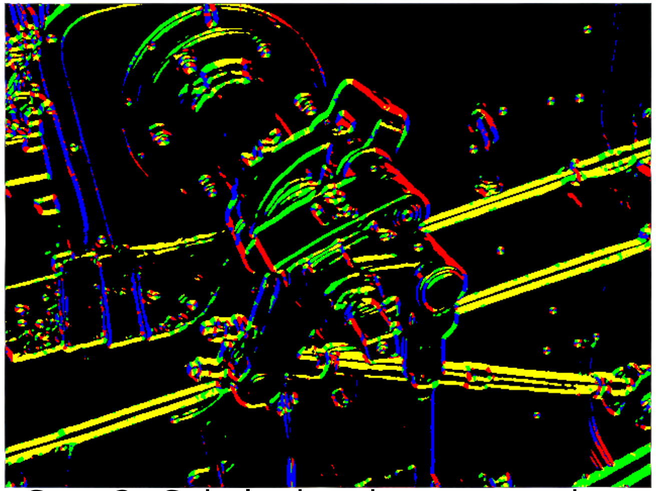
Original image



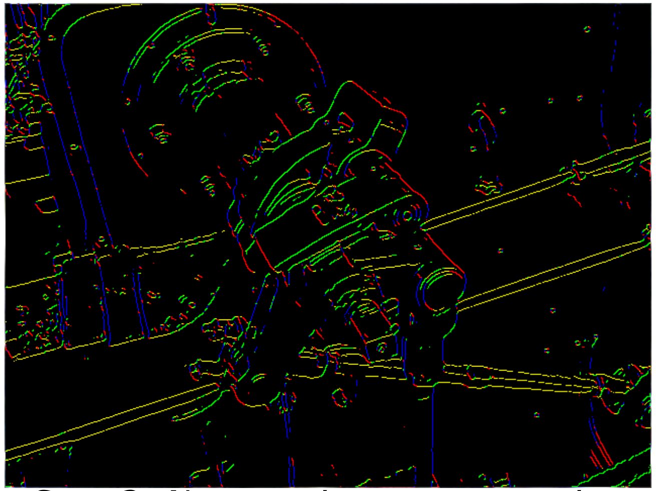
Step 1: Conversion to grayscale and smoothing with 5x5 Gaussian



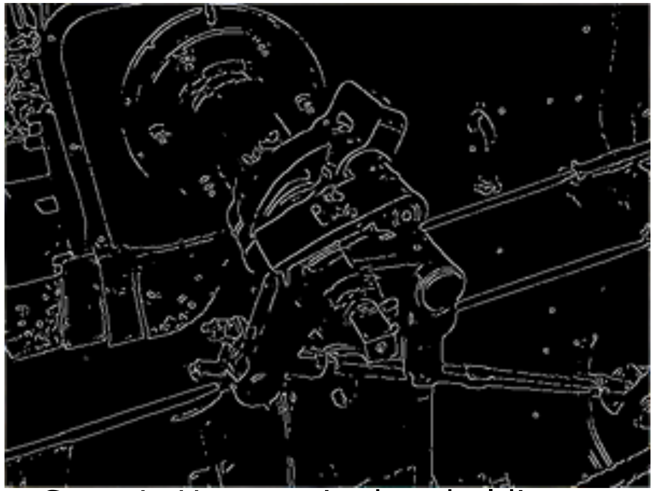
Step 2: Sobel edge detector – edge magnitude image



Step 2: Sobel edge detector – edge orientation image

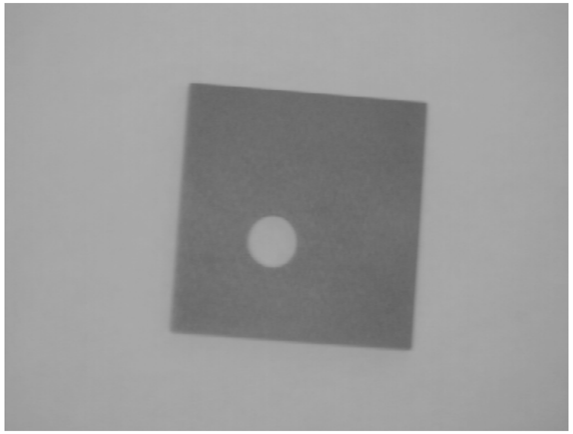


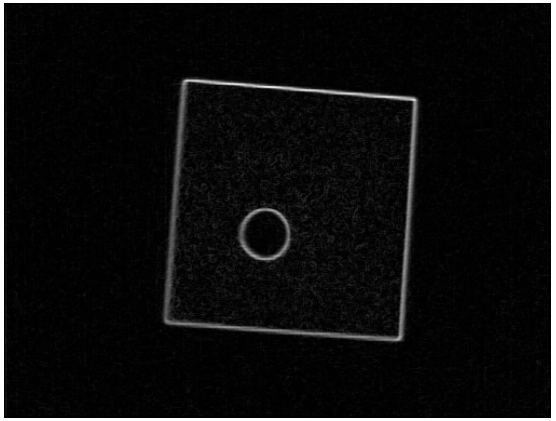
Step 3: Non-maximum suppression



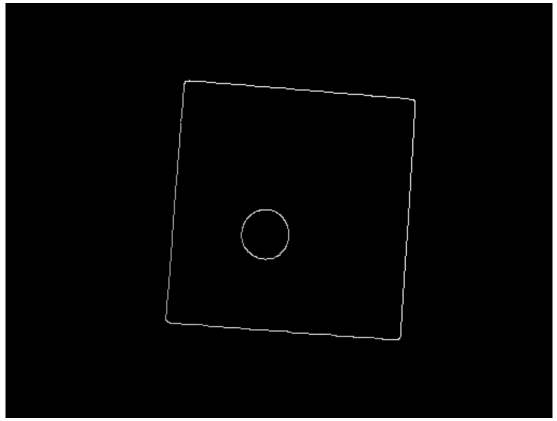
Step 4: Hysteresis thresholding final results

Sobel vs. Canny



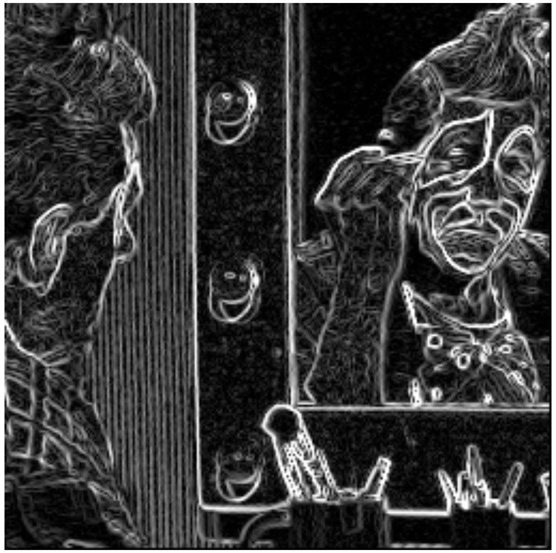


Sobel

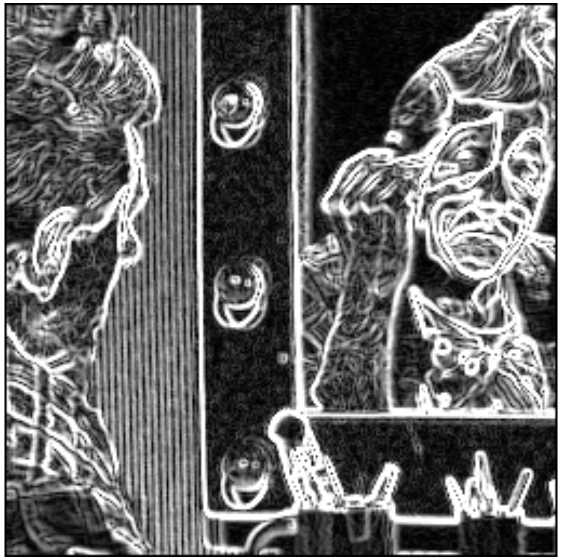


Canny

Roberts vs. Sobel

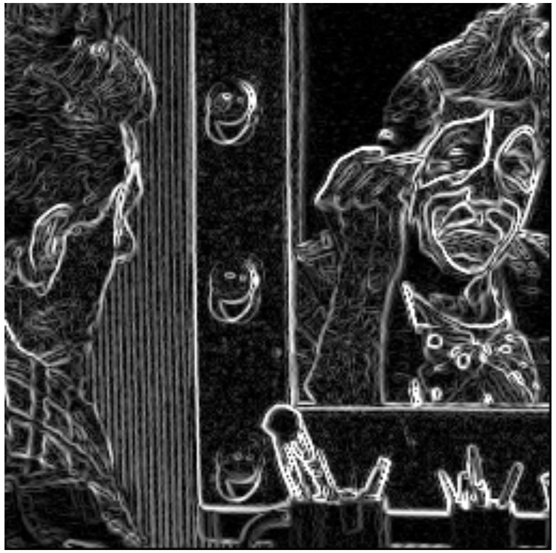


Roberts



Sobel

Roberts vs. Canny



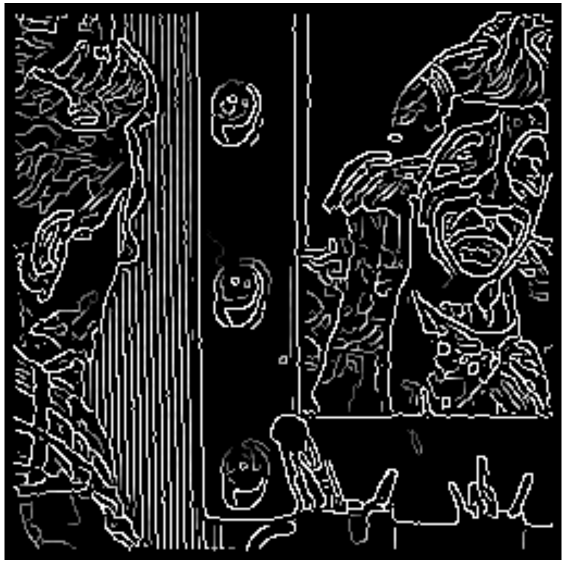
Roberts

Canny

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

Canny Edge Detector

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



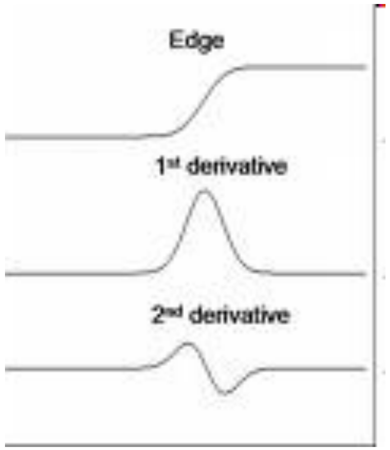
Canny
$\sigma = 1, t_l=1 \;, t_h= 128$

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

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:

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

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

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

- 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.

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

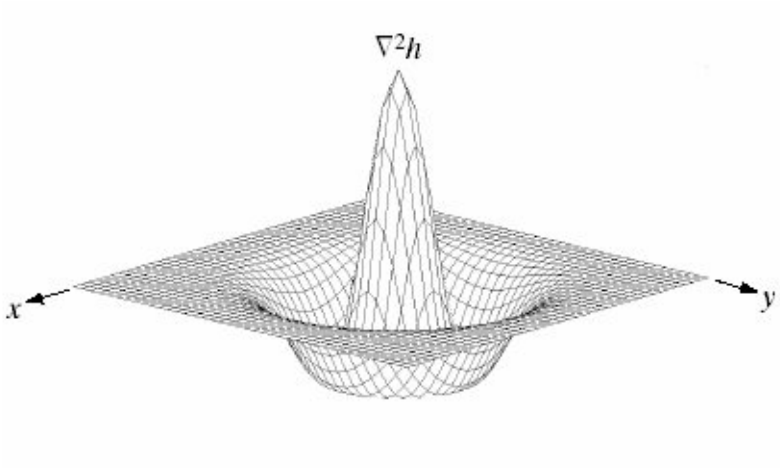
- The combined filter (H_{Lap} * H_{Gauss}) is nothing more than computing the Laplacian of the Gaussian (LoG):

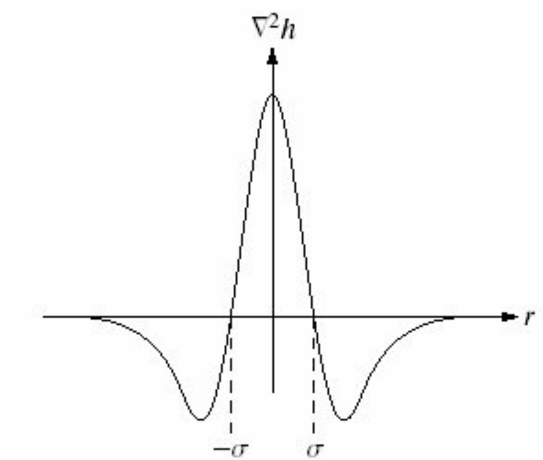
$$\nabla^2(G_{auss}(x, y)) = \nabla^2(\mathrm{e}^{(-(x^2+y^2)/2\sigma^2})$$

$$= \frac{(x^2 + y^2 - \sigma^2)}{\sigma^4}(\mathrm{e}^{(-(x^2+y^2)/2\sigma^2})$$

LoG Kernel

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

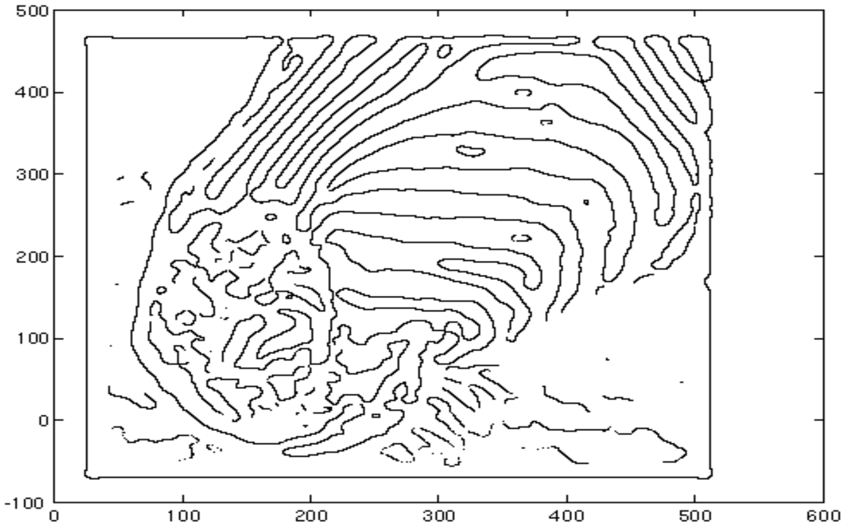




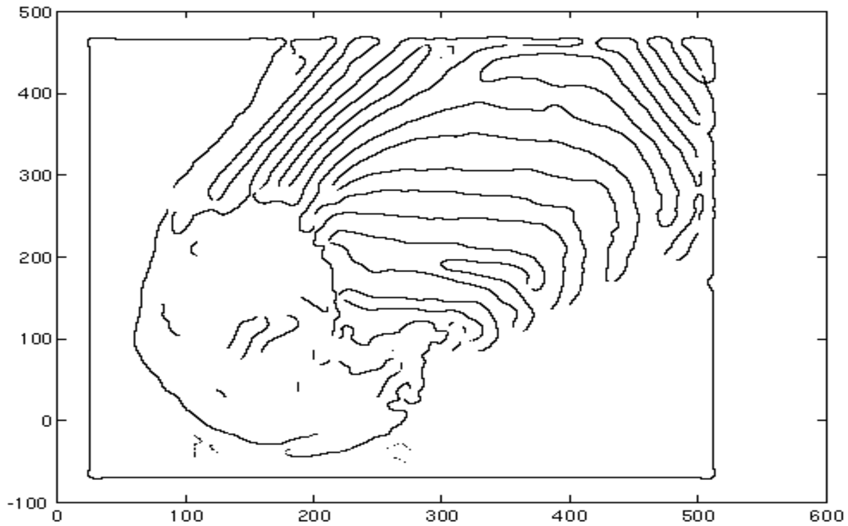
- $\nabla^2(G_{auss}(x,y))$ can also be approximated by a convolution kernel:

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

Examples of LoG Zero Crossings

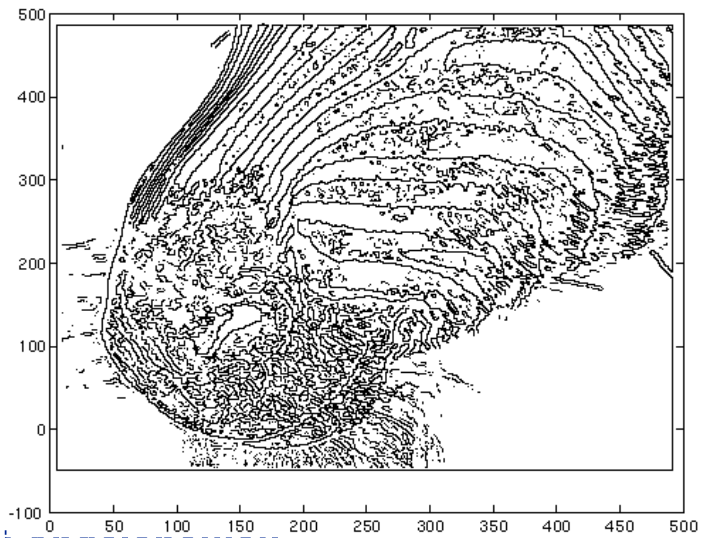


$\sigma = 4$

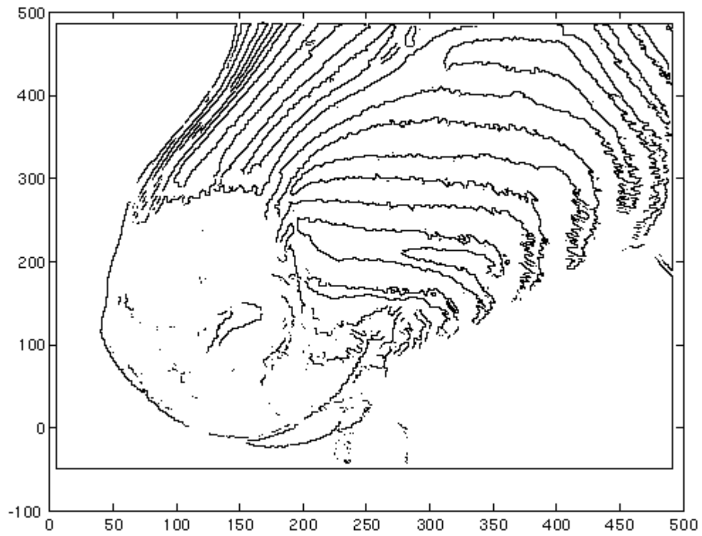


contrast=1

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.

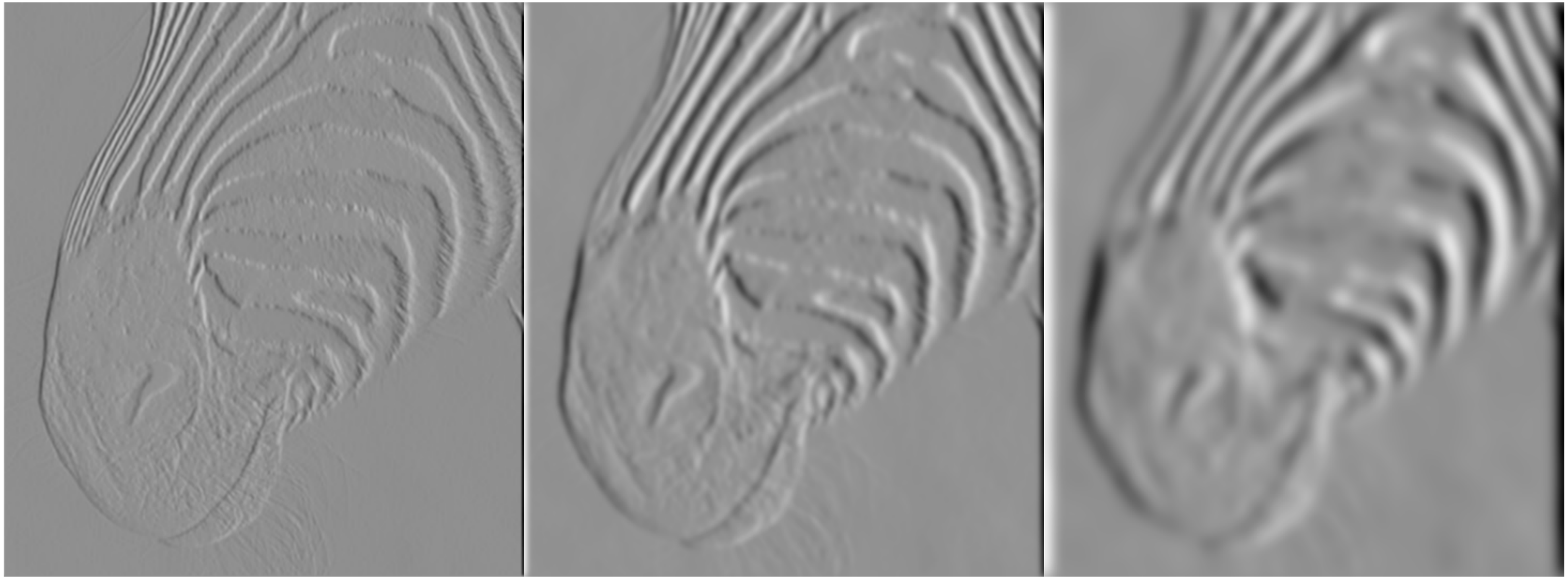
$$R = H_{edge} * (H_{smooth} * I) = (H_{edge} * H_{smooth}) * I = H * I$$

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

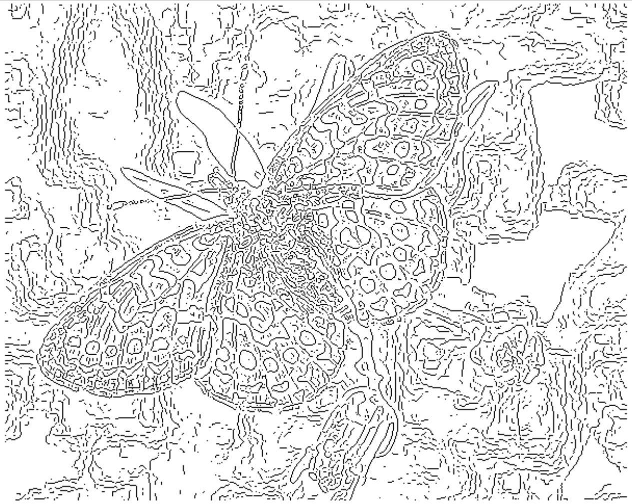
- The scale of the smoothing filter affects the derivative estimates as well as the semantics of the recovered edges



No smoothing | 3x3 filter | 7x7 filter

Different Scales

Original image



Fine scale, high threshold

Coarse scale, low threshold

Coarse scale, high threshold

Gaussian Pyramid

- Gaussian Pyramid (also known as a lowpass pyramid) is a hierarchy of low pass filtered versions of the original image.
- Successive layers correspond to lower frequencies (larger σ).
- Each successive layer is also a sub-sampled version of the previous level. Sub-sampling is typically by a factor of 2 in each coordinate direction.
- It allows us to analyze the image at different spatial and frequency resolutions.
- It is a form of multi-resolution analysis.

Gaussian Pyramid Example

Construction of a Gaussian Pyramid

- Let l be the level of the Gaussian pyramid. Then:

$$G_l(x,y) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} H_{Gauss}(m,n) G_{l-1}(2x+m, 2y+n)$$

  where k is typically set to 2.

- This function is also known as the REDUCE operation:

$$G_l = \mathrm{REDUCE}(G_{l-1})$$

- A Gaussian pyramid is then recursively constructed:

$$G_0 = I$$

$$G_{l+1} = \mathrm{REDUCE}(G_l)$$

Gaussian Pyramid Facts

- Each pixel at level l contains the local weighted average of the neighborhood of the corresponding pixel at previous level $l-1$ of the Gaussian pyramid.
- In such a pyramid, a coarse level, l , representation predicts the appearance of the immediate finer level, $l-1$.
- One can use an **upsampling** (EXPAND) operation to reconstruct an approximation of the immediate finer level $l-1$ from the coarse level l.

Expand Operation

- The following operation produces an *approximation (a smoothed version)* of level l using the information stored in a coarser level of the Gaussian pyramid:

$$G_l(x,y) = 4 \sum_{m=-k}^{k} \sum_{n=-k}^{k} H_{Gauss}(m,n) G_{l+1}\left(\frac{x-m}{2}, \frac{y-n}{2}\right)$$

where k is typically set to 2.

- This function is also known as the EXPAND operation:

$$G_l = \mathrm{EXPAND}(G_{l+1})$$

Gaussian Pyramid Example

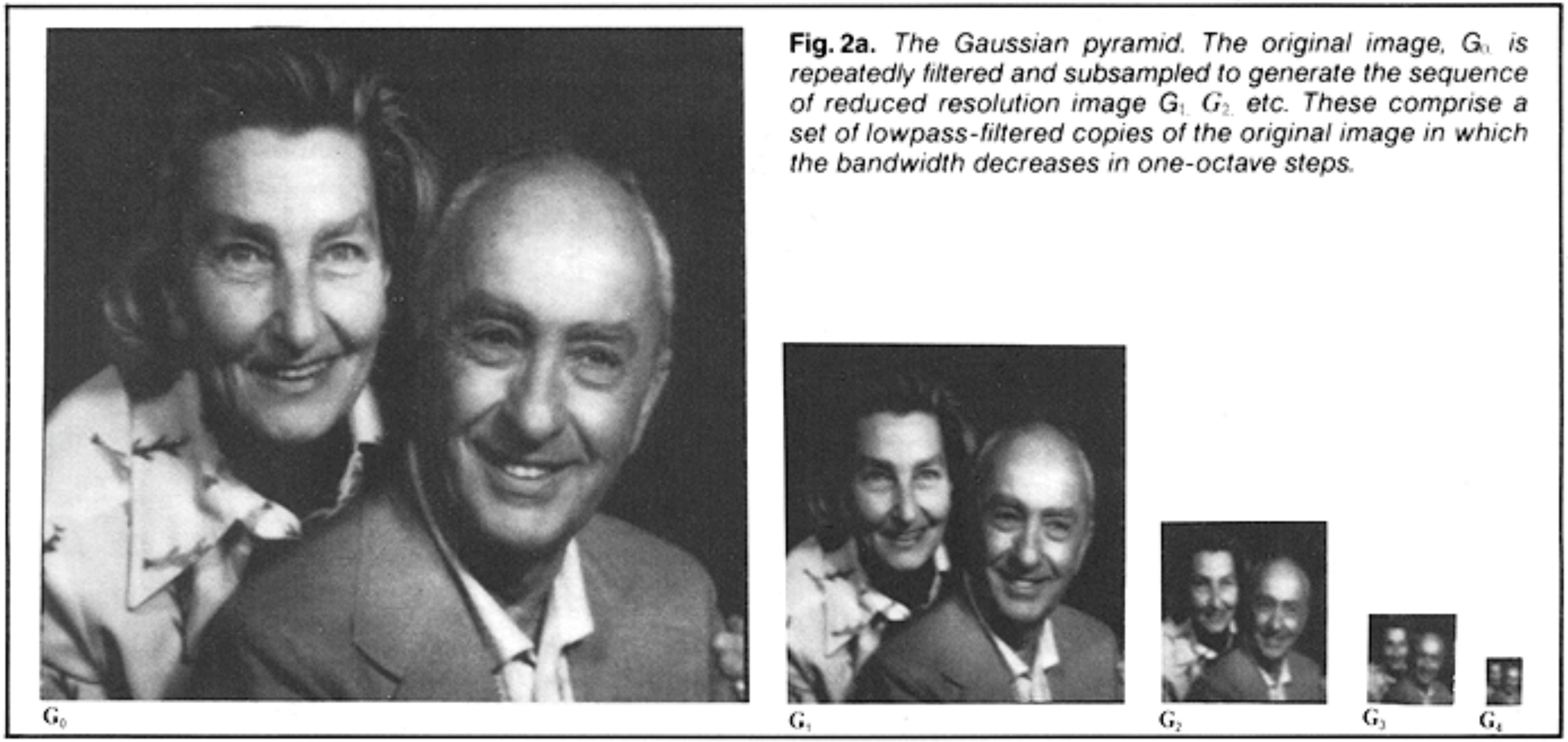


Fig. 2a. *The Gaussian pyramid. The original image, G_0, is repeatedly filtered and subsampled to generate the sequence of reduced resolution image G_1, G_2, etc. These comprise a set of lowpass-filtered copies of the original image in which the bandwidth decreases in one-octave steps.*

Figure from the original paper: E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden, “Pyramid Methods in Image Processing,” RCA Engineer, Nov/Dec 1984, pp. 33-41.

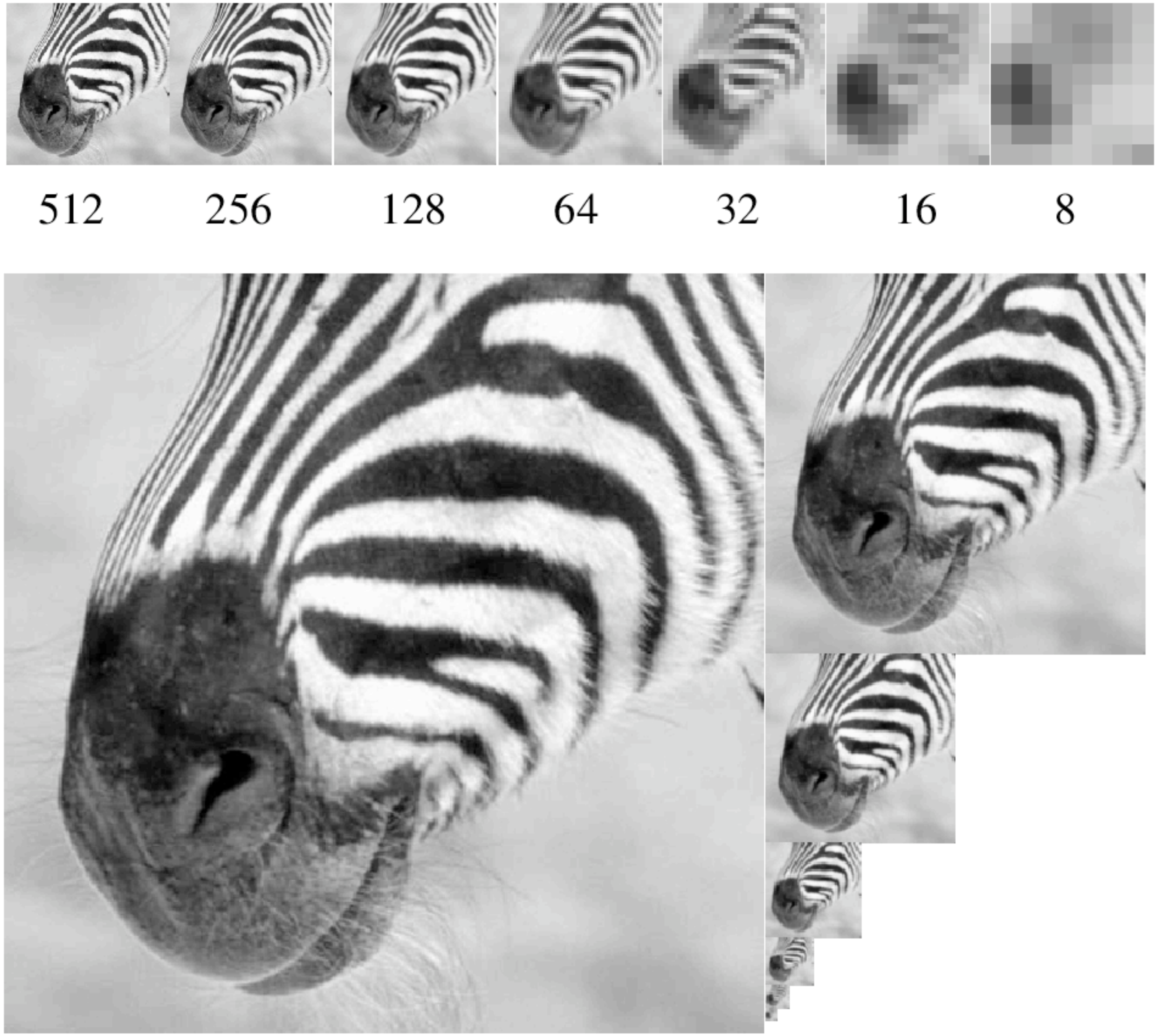
Successive Smoothing

$G_{0,0}$ $G_{1,1}$ $G_{2,2}$

Fig. 2b. Levels of the Gaussian pyramid expanded to the size of the original image. The effects of lowpass filtering are now clearly apparent.

Figure from the original paper: E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden, "Pyramid Methods in Image Processing," RCA Engineer, Nov/Dec 1984, pp. 33-41.

Gaussian Pyramid Example



Laplacian Pyramid

- Given such an EXPAND operation, one can reproduce the immediate finer level l from the coarse level $l+1$ by just **storing the differences** between two successive levels.

$$\Delta = L_l(x,y) = G_l(x,y) - \mathrm{EXPAND}(G_{l+1}(x,y))$$

$$G_l(x,y) = \Delta + \mathrm{EXPAND}(G_{l+1}(x,y))$$

- One can then build another type of pyramid based on these difference images, called the **Laplacian pyramid**.

Laplacian Pyramid Example

Construction of the Laplacian Pyramid

- The Laplacian Pyramid is also known as a bandpass pyramid.

$$L_l(x,y) = G_l(x,y) - \mathrm{EXPAND}(G_{l+1}(x,y))$$

$$L_N(x,y) = G_N(x,y)$$

where N is the highest level of the pyramid.

- It is a complete image representation. The steps used to construct the pyramid can be reversed to recover the original image exactly. Thus, the Laplacian pyramid can be used for image compression.

$$G_0 = \sum_l L_l$$

Laplacian Pyramid Example

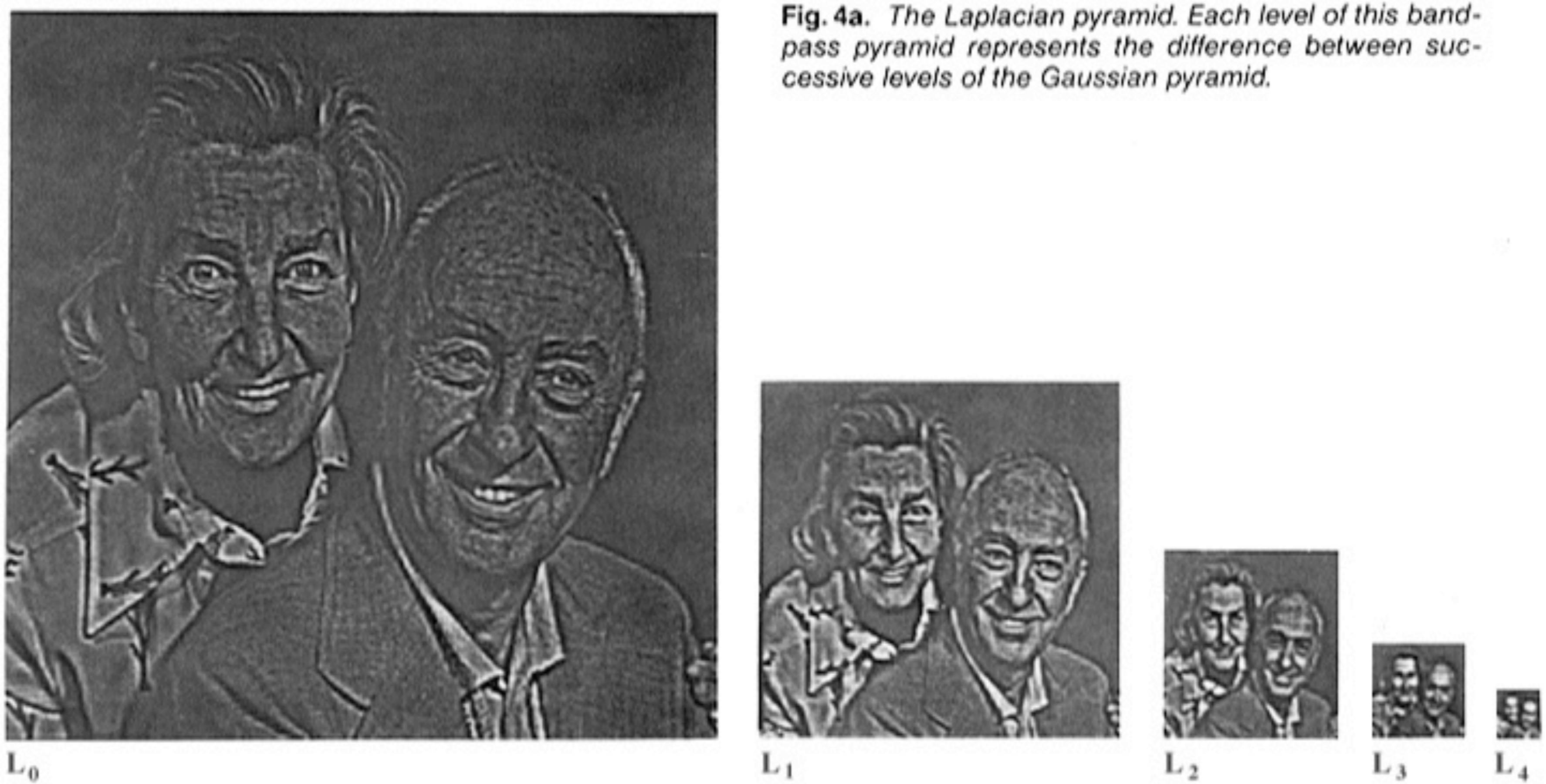
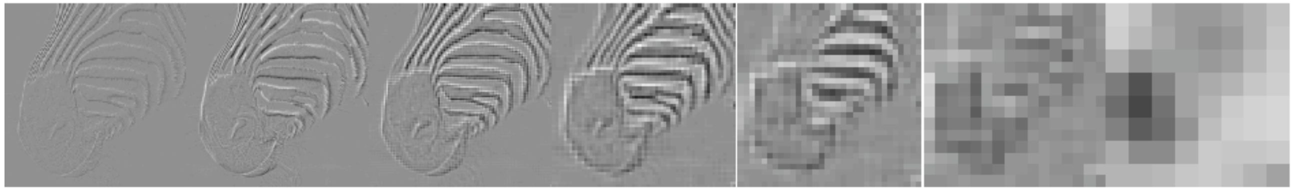


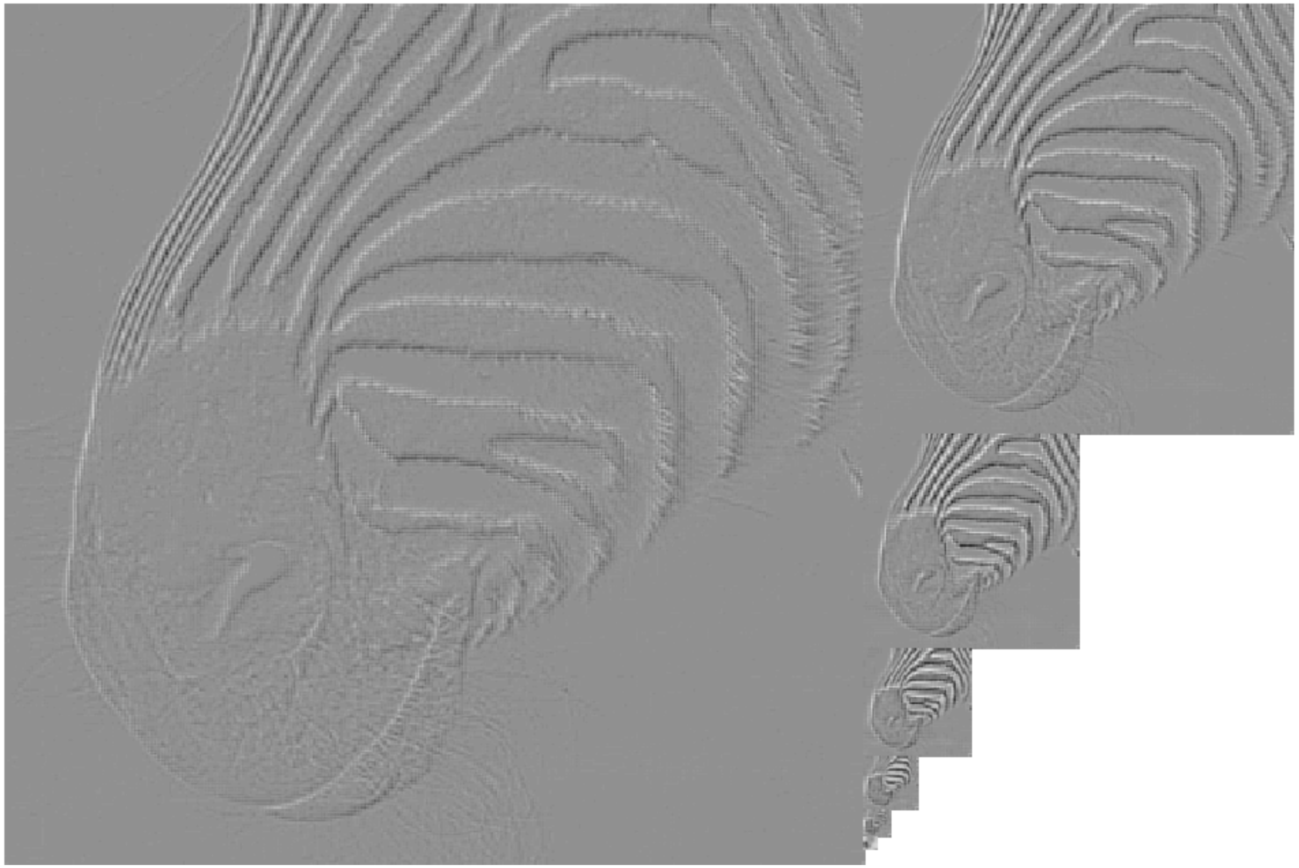
Fig. 4a. *The Laplacian pyramid. Each level of this band-pass pyramid represents the difference between successive levels of the Gaussian pyramid.*

Figure from the original paper: E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden, "Pyramid Methods in Image Processing," RCA Engineer, Nov/Dec 1984, pp. 33-41.

Laplacian Pyramid Example



512 256 128 64 32 16 8



Laplacian Pyramid Example

(a) (b) (c)

Fig. 5. Pyramid data compression. The original image represented at 8 bits per pixel is shown in (a). The node values of tbe Laplacian pyramid representation of this image were quantitized to obtain effective data rates of 1 b/p and 1/2 b/p. Reconstructed images (b) and (c) show relatively little degradation.

Figure from the original paper: E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden, "Pyramid Methods in Image Processing," RCA Engineer, Nov/Dec 1984, pp. 33-41.

Laplacian vs. Gaussian Pyramid

- At each level, a Laplacian filtering and a Gaussian filtering is performed respectively. The standard deviation σ increases at each level.

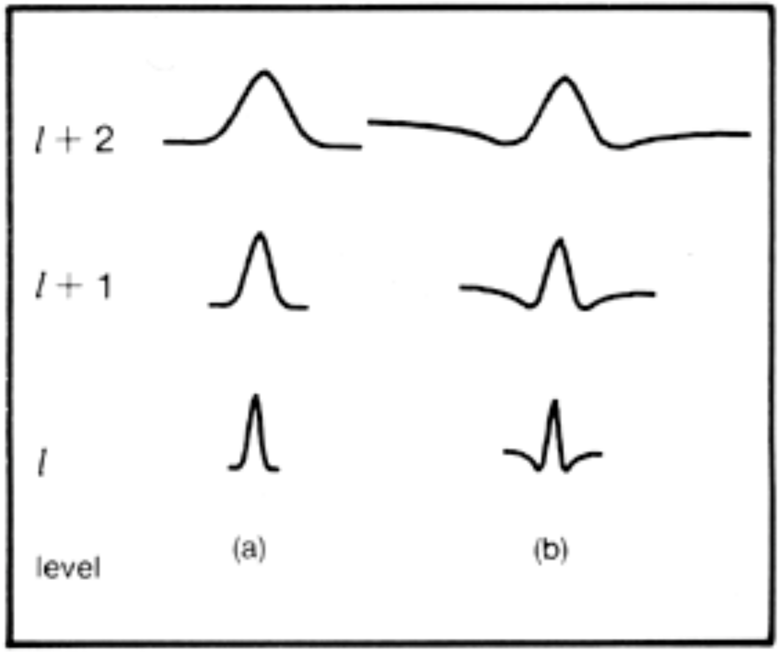


Figure from the original paper: E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden, “Pyramid Methods in Image Processing,” RCA Engineer, Nov/Dec 1984, pp. 33-41.

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. "Nonmaximum suppresion" from Wikipedia Commons.
4. "LoG plots", Simon Yu Ming, http://hi.baidu.com/simonyuee/blog/item/446a911bf43cc91c8618bf8f.html
5. Many of the edge detection and the pyramid images are from the slides by D.A. Forsyth, University of California at Urbana-Champaign.
6. The pyramid images of the female face are from Technion – Israel Institute of Technology, http://www.cs.technion.ac.il/~ronrubin/Projects/fusion/index.html