Lecture 7:
Most Common Edge Detectors

Saad Bedros

sbedros@umn.edu

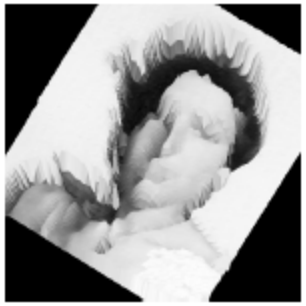
Edge Detection

- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels
- **Ideal:** artist's line drawing (but artist is also using object-level knowledge)

Source: D. Lowe

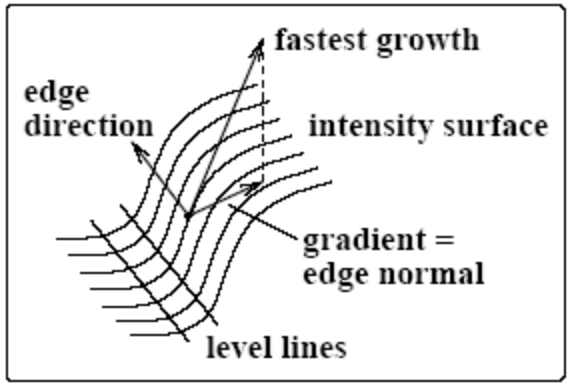
Gradient Visualization

original image



intensity surface

thresholded image



Intensity surface of edge and its gradient

Designing an edge detector

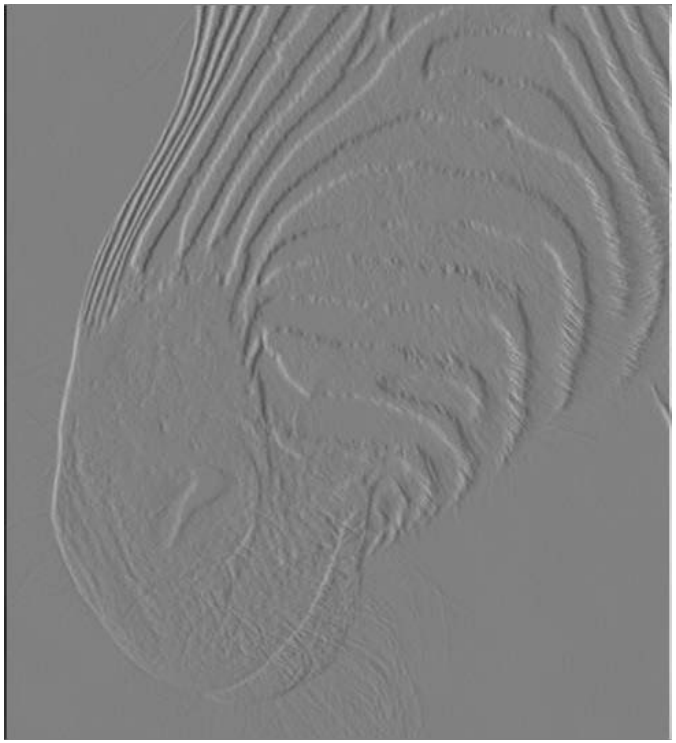
- Criteria for a good edge detector:
  - **Good detection:** the optimal detector should find all real edges, ignoring noise or other artifacts
  - **Good localization**
    - the edges detected must be as close as possible to the true edges
    - the detector must return one point only for each true edge point
- Cues of edge detection
  - Differences in color, intensity, or texture across the boundary
  - Continuity and closure
  - High-level knowledge

Source: L. Fei-Fei

Edge Detection Using Derivative

- Image first and second derivatives are a good first level Edge Detectors: Convolution …

Image I



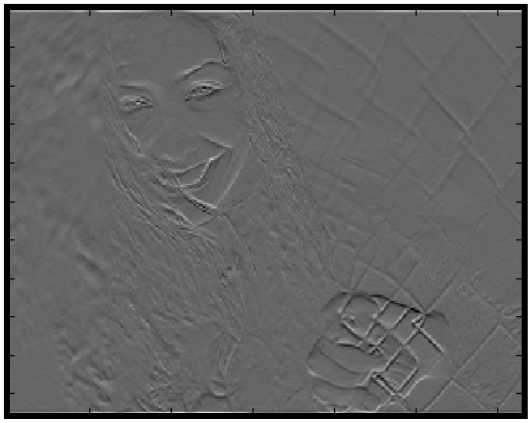
$$I_x = I * \begin{bmatrix} -1 & 1 \end{bmatrix}$$

Edge Detection Using Derivative

- Image derivatives:

Image I

$$I_x = I * \begin{bmatrix} -1 & 1 \end{bmatrix}$$

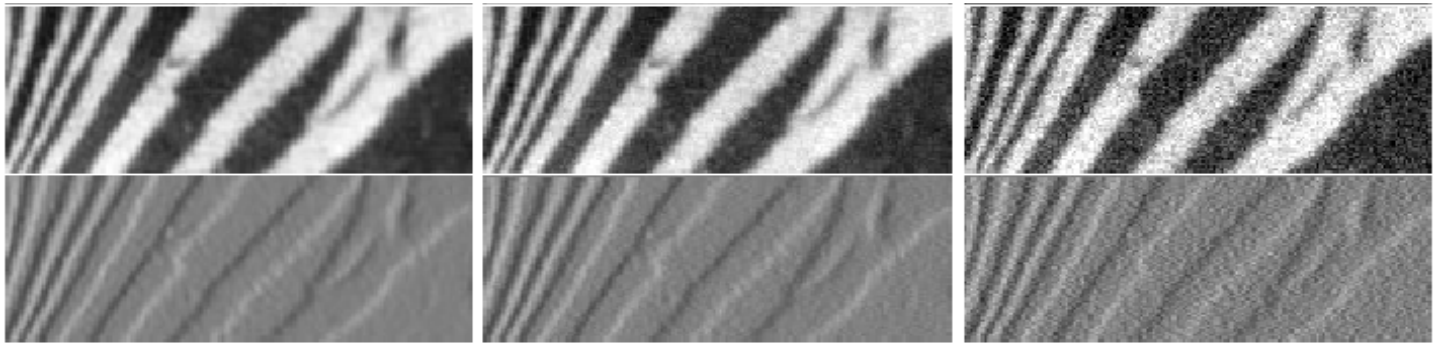


$$I_y = I * \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Derivatives and Noise

- Derivatives are strongly affected by noise
  - obvious reason: image noise results in pixels that look very different from their neighbors
  - The larger the noise - the stronger the response
- What is to be done?
  - Neighboring pixels look alike
  - Pixel along an edge look alike
  - Image smoothing should help
  - Force pixels different to their neighbors (possibly noise) to look like neighbors

Derivatives and Noise



- Need to perform image smoothing as a preliminary step
- Generally – use Gaussian smoothing

Edge Detection & Image Noise

$$I(x) = \hat{I}(x) + N(x) \qquad N(x) \sim N(0,\sigma)\ \text{i.i.d}$$

Taking differences:

$$I'(x) \cong \hat{I}(x+1) + N(x+1) \ - (\hat{I}(x-1) + N(x-1)) \ =$$

$$= \underbrace{(\hat{I}(x+1) - \hat{I}(x-1))}_{\hat{I}'(x)} + \underbrace{(N(x+1) - N(x-1))}_{N_d(x)}$$

Output noise: $E(N_d(x)) = 0$

$$E(N_d{}^2(x)) = E(N^2(x+1) + N^2(x-1) + 2N(x+1)N(x-1)) =$$

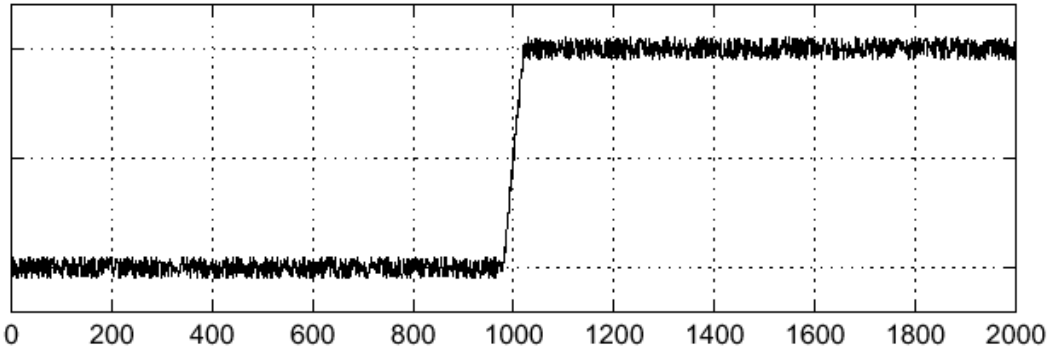
$$= \sigma^2 + \sigma^2 + 0 = 2\sigma^2$$

Increases noise !!

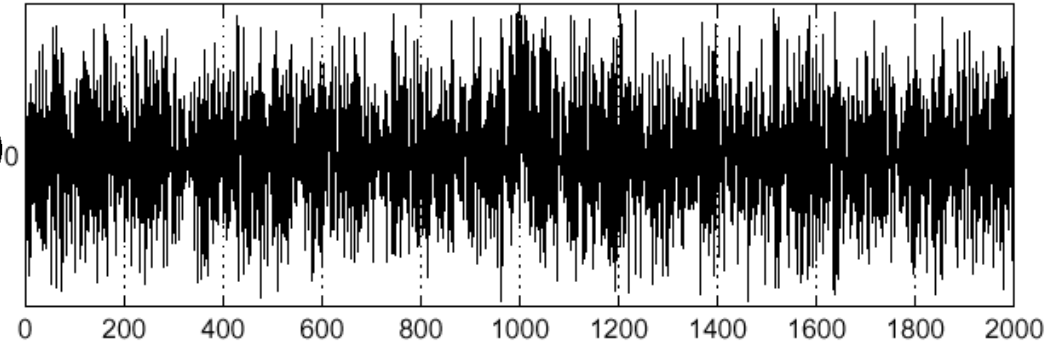
Effects of noise

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

$f(x)$



$\frac{d}{dx}f(x)$



- Where is the edge?

Solution: smooth first

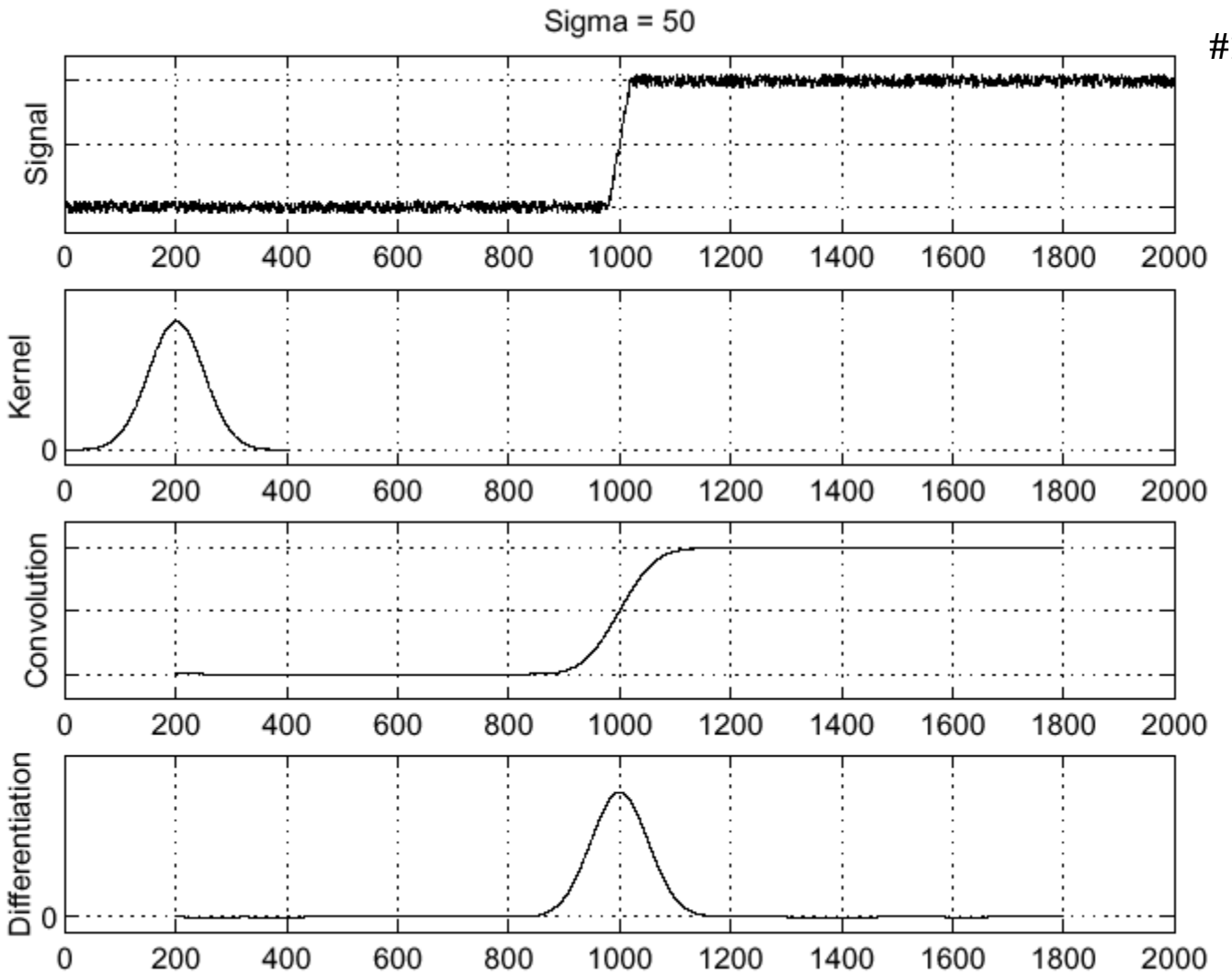
- Where is the edge?

f

h

$h \star f$

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



Derivative theorem of convolution

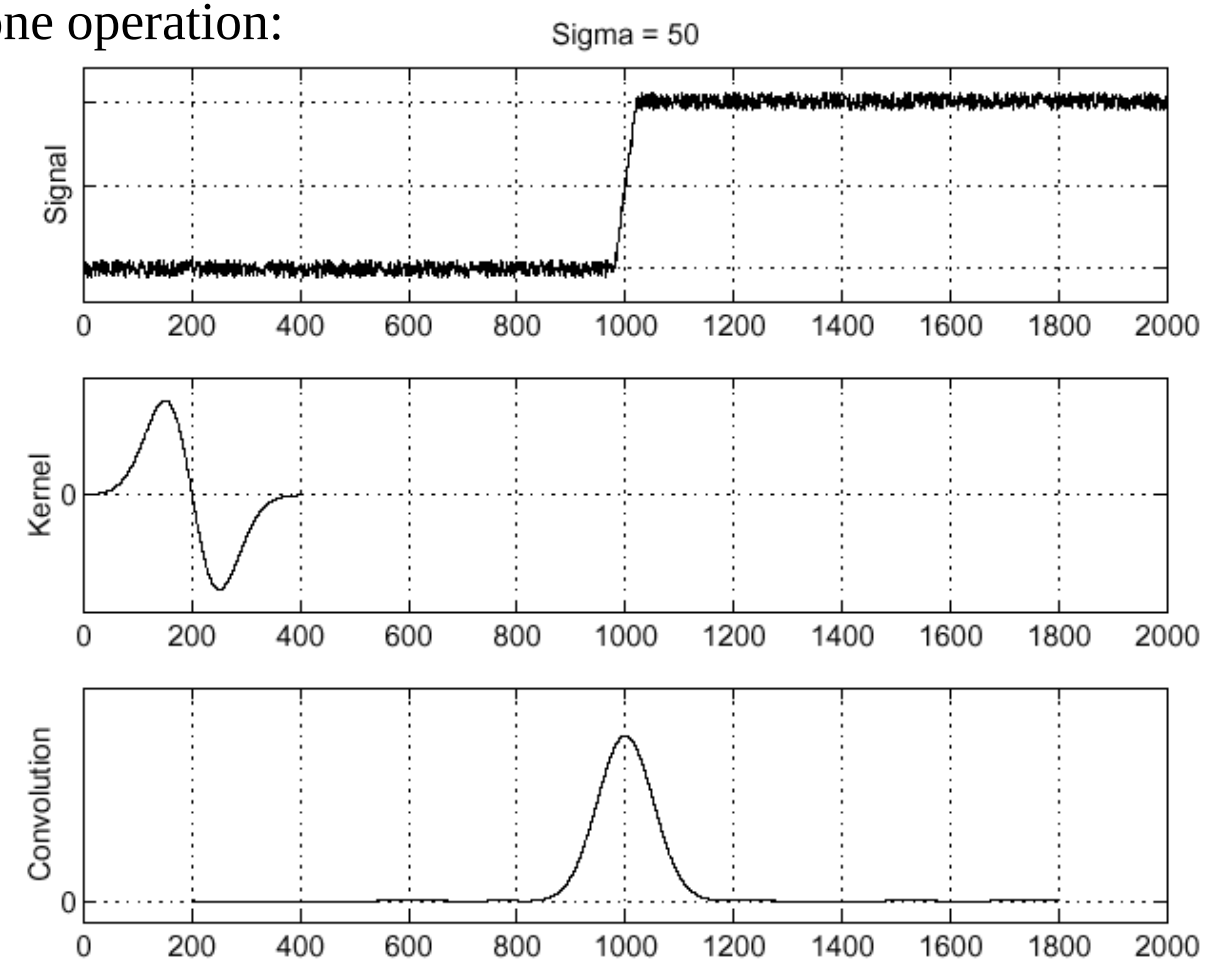
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:

f

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

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



Edge Detection Methods

Most Common Edge Detectors:

- Gradient operators
  - Roberts
  - Prewitt
  - Sobel
- Gradient of Gaussian (Canny)
- Laplacian of Gaussian (Marr-Hildreth)
- Facet Model Based Edge Detector (Haralick)

Edge Detection Using the Gradient

- Definition of the gradient:

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

$$magn(\nabla f) = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2} = \sqrt{M_x{}^2 + M_y{}^2}$$

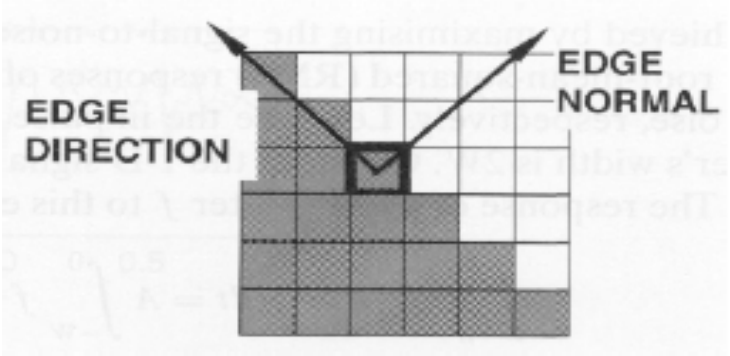
$$dir(\nabla f) = \tan^{-1}(M_y / M_x)$$

- To save computations, the magnitude of gradient is usually approximated by:

$$magn(\nabla f) \approx |M_x| + |M_y|$$

Edge Detection Using the Gradient

- Properties of the gradient:
  - The magnitude of gradient provides information about the strength of the edge
  - The direction of gradient is always perpendicular to the direction of the edge



- Main idea:
  - Compute derivatives in x and y directions
  - Find gradient magnitude
  - Threshold gradient magnitude

Edge Detection Using the Gradient

- Estimating the gradient with finite differences

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

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

  - Approximation by *finite differences*:

$$\frac{\partial f}{\partial x} = \frac{f(x+h_x, y) - f(x, y)}{h_y} = f(x+1, y) - f(x, y), \quad (h_x = 1)$$

$$\frac{\partial f}{\partial y} = \frac{f(x, y+h_y) - f(x, y)}{h_y} = f(x, y+1) - f(x, y), \quad (h_y = 1)$$

Edge Detection Using the Gradient

- Example:
  - Suppose we want to approximate the gradient magnitude at z_5

Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

$$\frac{\partial I}{\partial x} = z_6 - z_5, \qquad \frac{\partial I}{\partial y} = z_5 - z_8$$

$$magn(\nabla I) = \sqrt{(z_6 - z_5)^2 + (z_5 - z_8)^2}$$

- We can implement $\partial I/\partial x$ and $\partial I/\partial$y using the following masks:

-1	1

1
-1

Note: M_x is the approximation at $(i, j + 1/2)$ and M_y is the approximation at $(i + 1/2, j)$

Edge Detection Steps Using Gradient

(1) Smooth the input image ($\hat{f}(x,y) = f(x,y) * G(x,y)$)

(2) $\hat{f}_x = \hat{f}(x,y) * M_x(x,y) \longrightarrow \frac{\partial f}{\partial x}$

(3) $\hat{f}_y = \hat{f}(x,y) * M_y(x,y) \longrightarrow \frac{\partial f}{\partial y}$

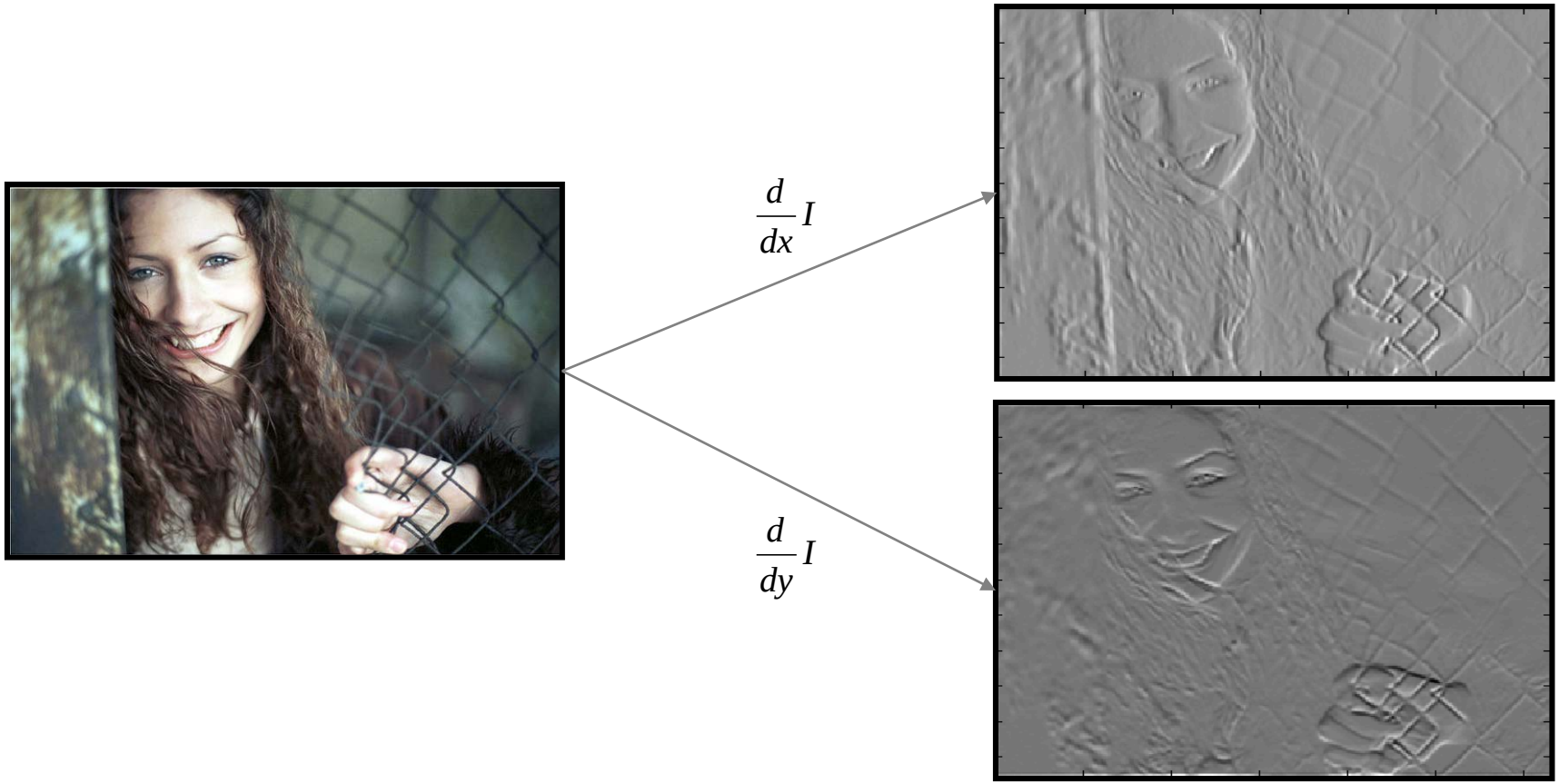
(4) $magn(x,y) = |\hat{f}_x| + |\hat{f}_y|$ (i.e., sqrt is costly!)

(5) $dir(x,y) = \tan^{-1}(\hat{f}_y / \hat{f}_x)$

(6) If $magn(x,y) > T$, then possible edge point

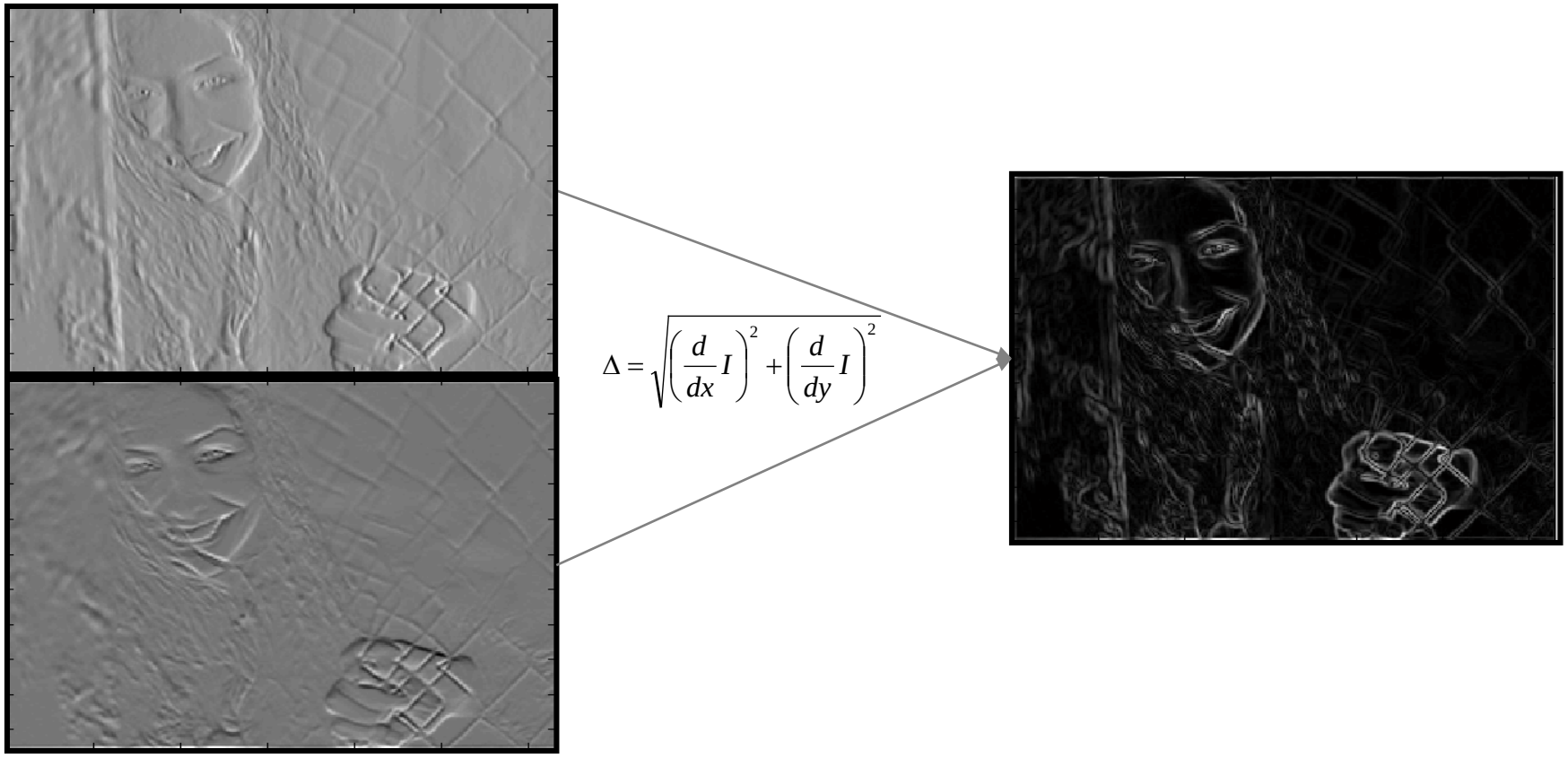
Edge Detection Using the Gradient

- Example:



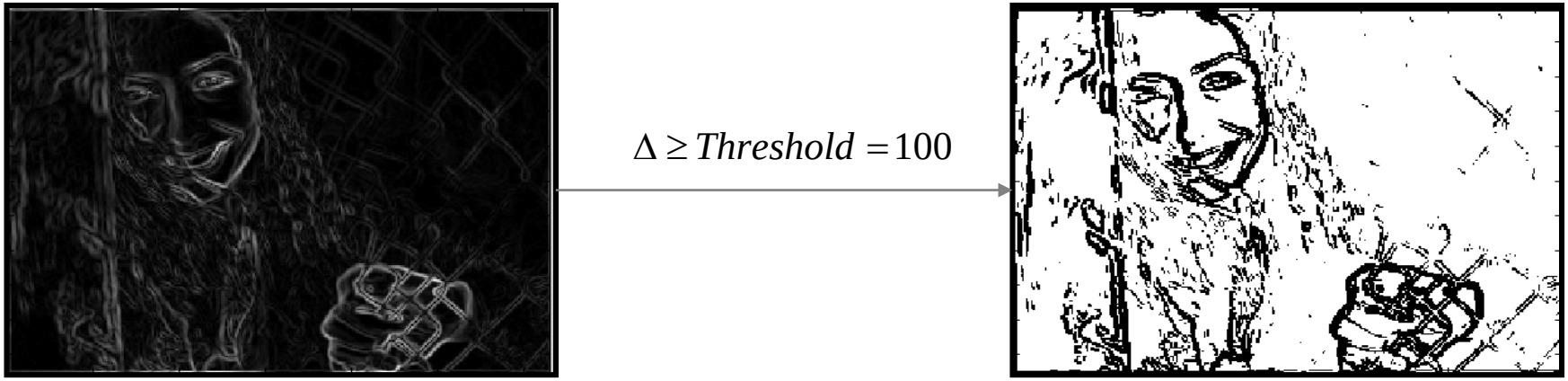
Edge Detection Using the Gradient

- Example – cont.:



Edge Detection Using the Gradient

- Example – cont.:



Roberts Edge Detector

- There are two forms of the Roberts Edge Detector:

1. $$\sqrt{[I(r,c)-I(r-1,c-1)]^2+[I(r,c-1)-I(r-1,c)]^2}$$

2. $$|I(r,c)-I(r-1,c-1)|+|I(r,c-1)-I(r-1,c)|$$

  - The second form of the equation is often used in practice due to its computational efficiency

Robert Edge Detection

- The Roberts edge detector

$$\frac{\partial f}{\partial x} = f(i, j) - f(i+1, j+1)$$

$$\frac{\partial f}{\partial y} = f(i+1, j) - f(i, j+1)$$

- This approximation can be implemented by the following masks:

$$M_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad M_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Note: M_x and M_y are approximations at $(i + 1/2, j + 1/2)$

Roberts Edge Detector

- A simple approximation to the first derivative
- Marks edge points only; it does not return any information about the edge orientation
- Simplest of the edge detection operators and will work best with binary images

Approximation of First Derivative

- Consider the arrangement of pixels about the pixel (i, j):

3 x 3 neighborhood:

$$\begin{matrix} a_0 & a_1 & a_2 \\ a_7 & [i,j] & a_3 \\ a_6 & a_5 & a_4 \end{matrix}$$

- The partial derivatives $\frac{\partial f}{\partial x}$ $\frac{\partial f}{\partial y}$ can be computed by:

$$M_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6)$$
$$M_y = (a_6 + ca_5 + a_4) - (a_0 + ca_1 + a_2)$$

- The constant c implies the emphasis given to pixels closer to the center of the mask.

Prewitt Operator

- Setting $c = 1$, we get the Prewitt operator:

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

M_x and M_y are approximations at (i, j)

Sobel Operator

- Setting $c = 2$, we get the Sobel operator:

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

M_x and M_y are approximations at (i, j)

Sobel and Prewitt Detectors

- Approximates the gradient by using a row and a column mask, which approximates the first derivative in each direction

- The Prewitt and Sobel edge detection masks find edges in both the horizontal and vertical directions, and then combine this information into a single metric

Sobel vs Prewitt

- The Prewitt is simpler to calculate than the Sobel, since the only coefficients are 1's, which makes it easier to implement in hardware

- However, the Sobel is defined to place emphasis on the pixels closer to the mask center, which may be desirable for some applications

Edge Detection Using the Gradient

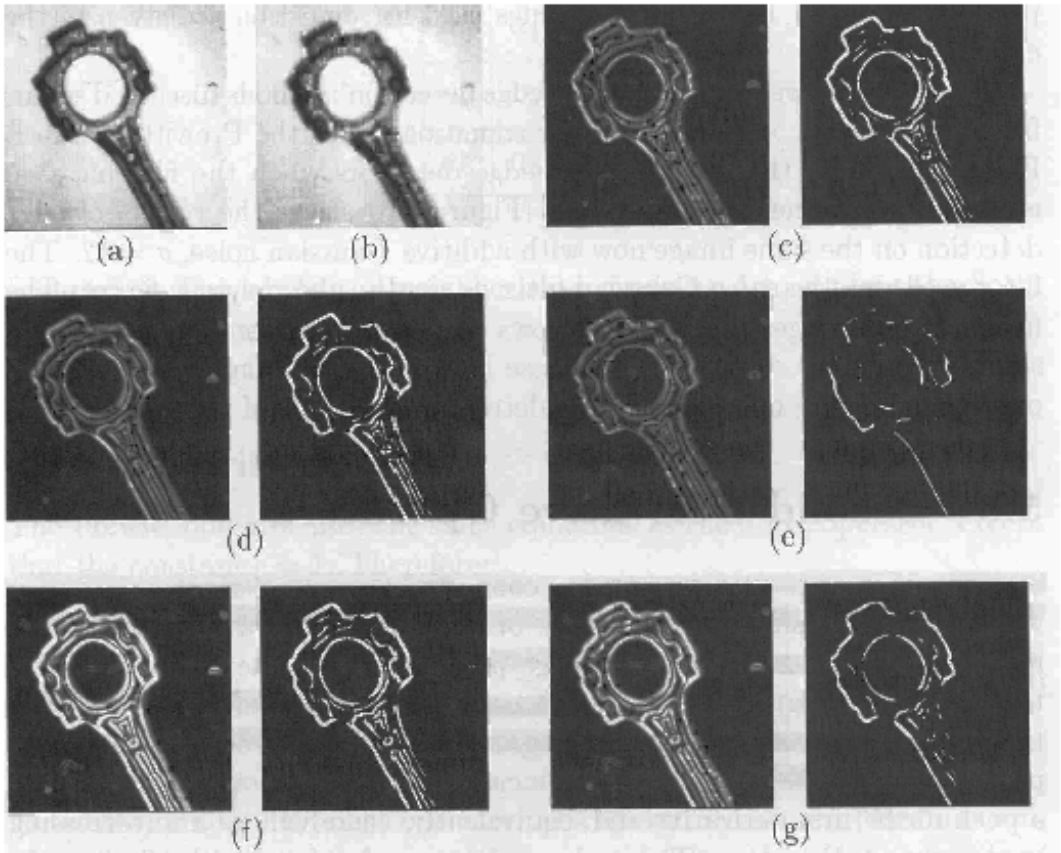


Figure 5.4: A comparison of various edge detectors. (a) Original image. (b) Filtered image. (c) Simple gradient using 1×2 and 2×1 masks, $T = 32$. (d) Gradient using 2×2 masks, $T = 64$. (e) Roberts cross operator, $T = 64$. (f) Sobel operator, $T = 225$. (g) Prewitt operator, $T = 225$.

(with noise filtering)

Edge Detection Using the Gradient

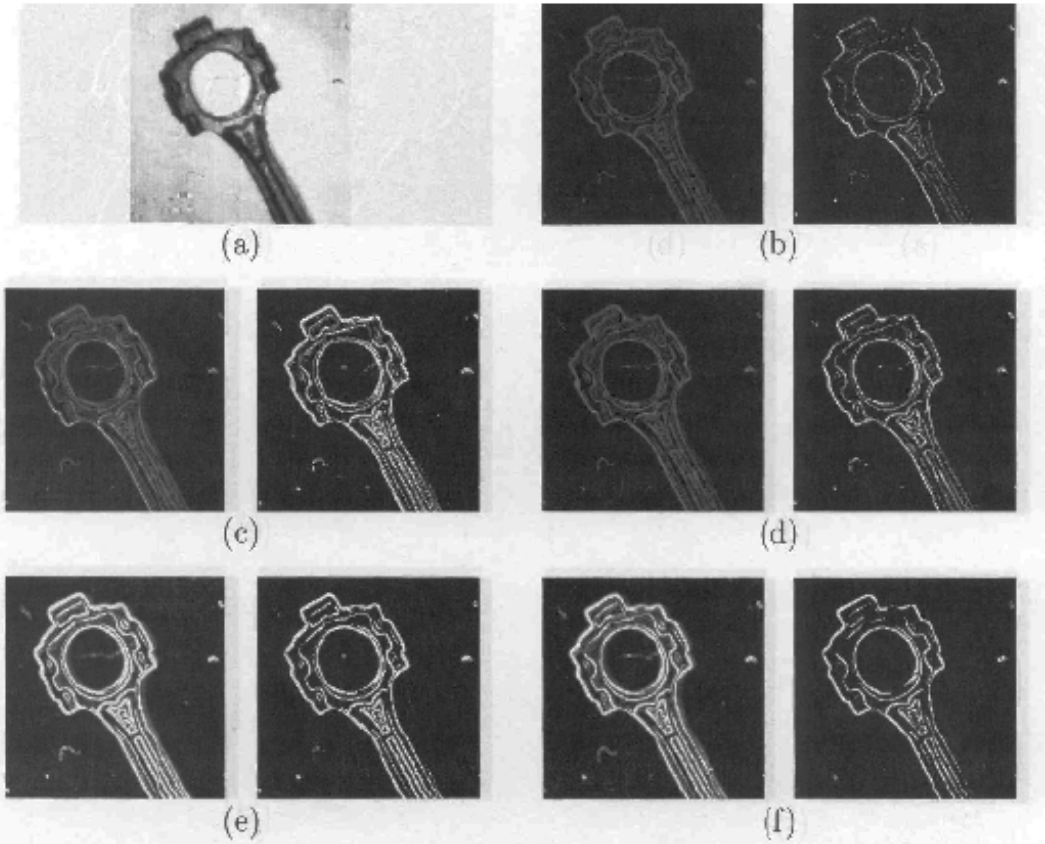
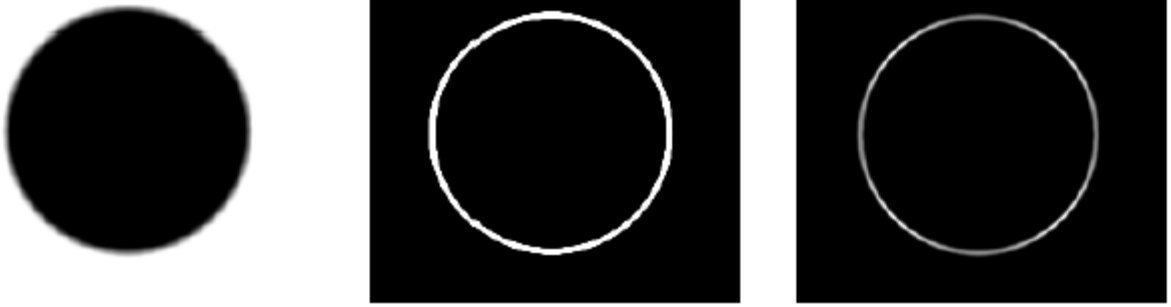


Figure 5.5: A comparison of various edge detectors without filtering. (a) Original image. (b) Simple gradient using 1×2 and 2×1 masks, $T = 64$. (c) Gradient using 2×2 masks, $T = 64$. (d) Roberts cross operator, $T = 64$. (e) Sobel operator, $T = 225$. (f) Prewitt operator, $T = 225$.

(without noise filtering)

Isotropy



original image | isotropic filter | anisotropic filter

- **Isotropic** filter: uniform edge magnitude for all directions
- **Anisotropic** edge filter: non-uniform magnitude
- In this illustration, response depends on edge orientation
  - directions $45° \cdot k$ are amplified
  - directions $90° \cdot k$ are suppressed

Edge Detection with Gradient Filters

Assume intensity function $f(x,y)$ is sufficiently smooth.

- Intensity **gradient** is **vector**

$$\nabla f(x,y) \doteq \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (f_x, f_y)$$

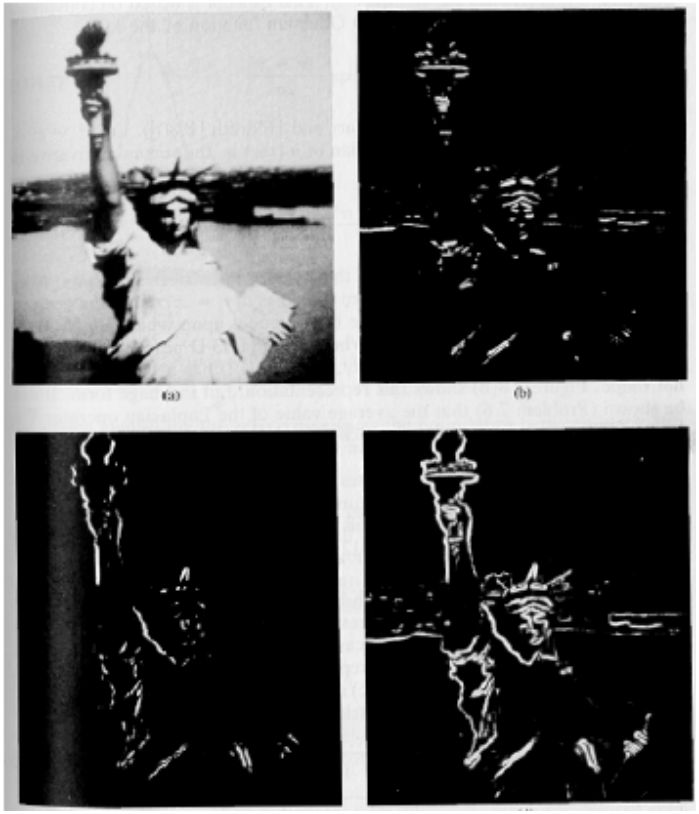
- **Magnitude** $M(x,y)$ and **orientation** $\Theta(x,y)$ of gradient are

$$\begin{aligned} M(x,y) = \|\nabla f(x,y)\| &= \sqrt{f_x^2 + f_y^2} \\ \Theta(x,y) &= \arctan\frac{f_x}{f_y} \end{aligned}$$

- Gradient vector gives direction and magnitude of *fastest growth of intensity*

Edge Detection Using the Gradient

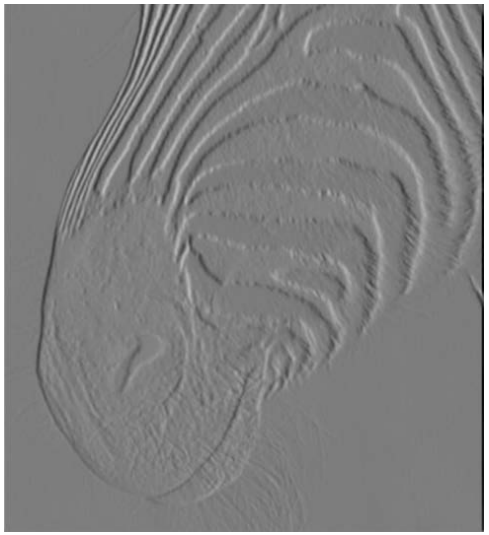
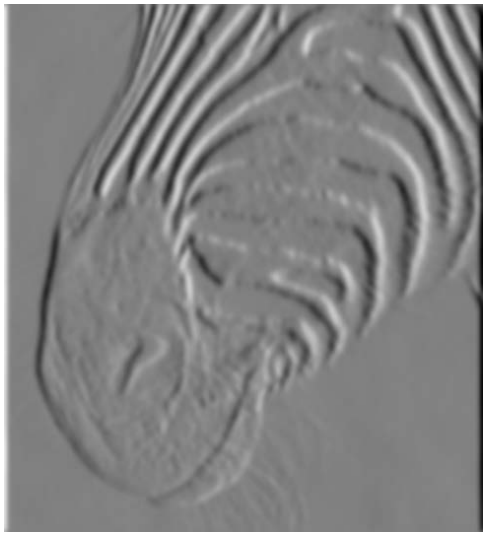
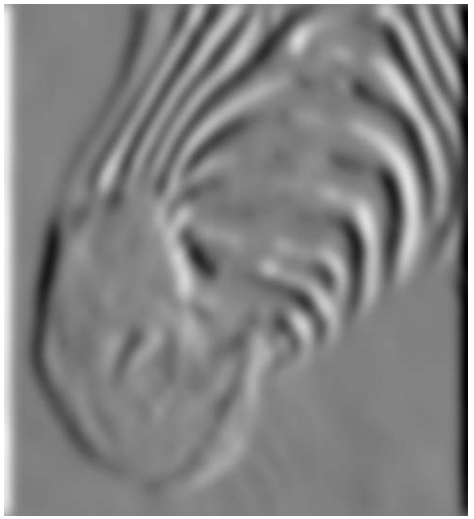
- Isotropic property of gradient magnitude:
  - The magnitude of gradient is an *isotropic* operator (it detects edges in any direction !!)



Other Steps in Edge Detection

- Practical issues:
  - Differential masks act as high-pass filters – tend to amplify noise.
  - Reduce the effects of noise - first smooth with a low-pass filter.

1) <u>The noise suppression-localization tradeoff</u>
   - a larger filter reduces noise, but worsens localization (i.e., it adds uncertainty to the location of the edge) and vice-versa.

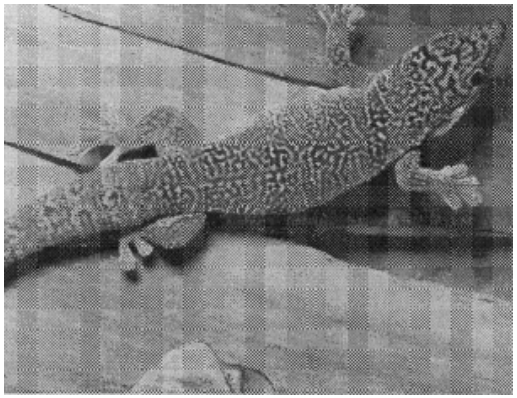




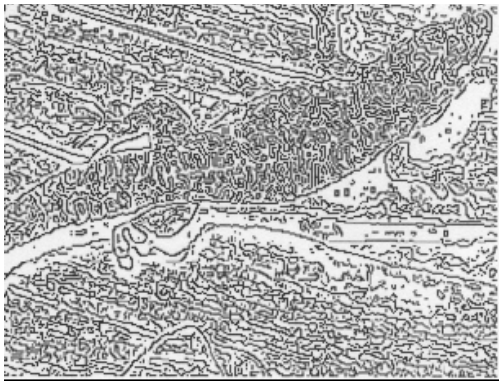
Noise Suppression vs Localization

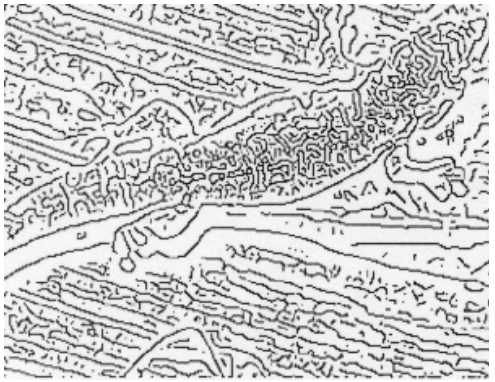
- Noise suppression-localization tradeoff.
  - Smoothing depends on mask size (e.g., depends on σ for Gaussian filters).
  - Larger mask sizes reduce noise, but worsen localization (i.e., add uncertainty to the location of the edge) and vice versa.

smaller mask

larger mask

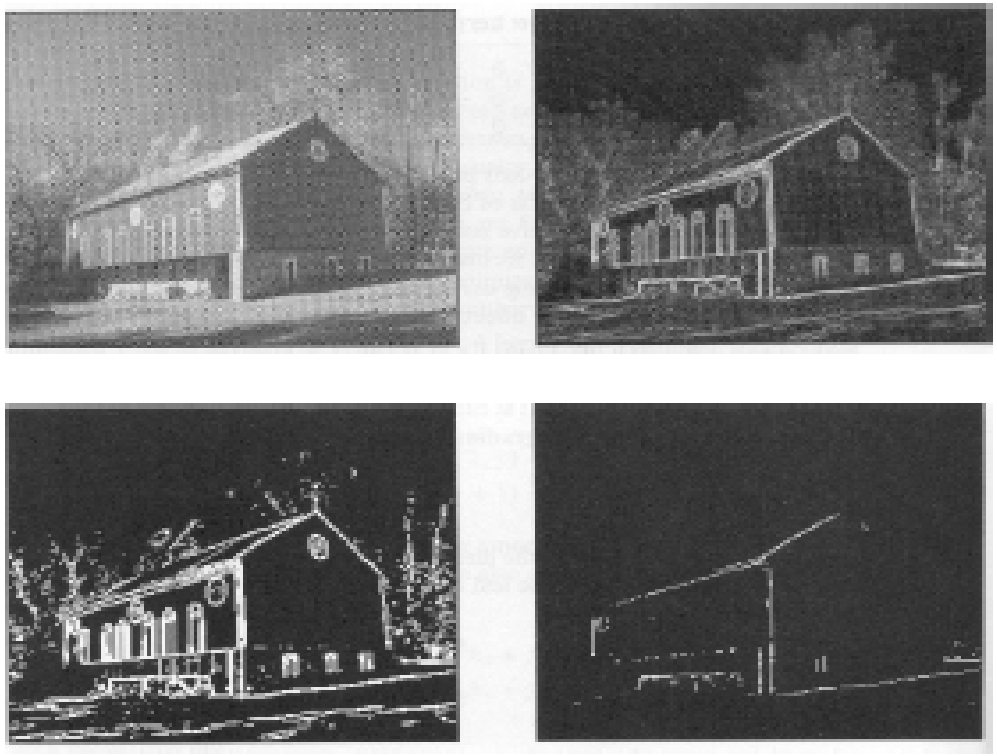






Other Steps in Edge Detection

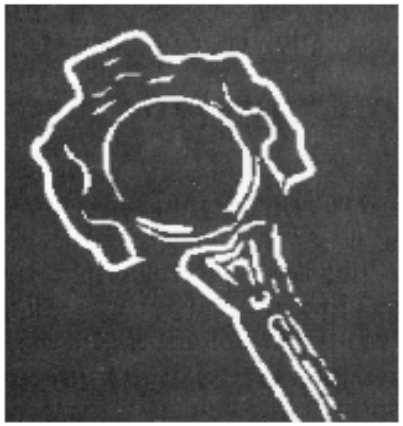
2) How should we choose the threshold?

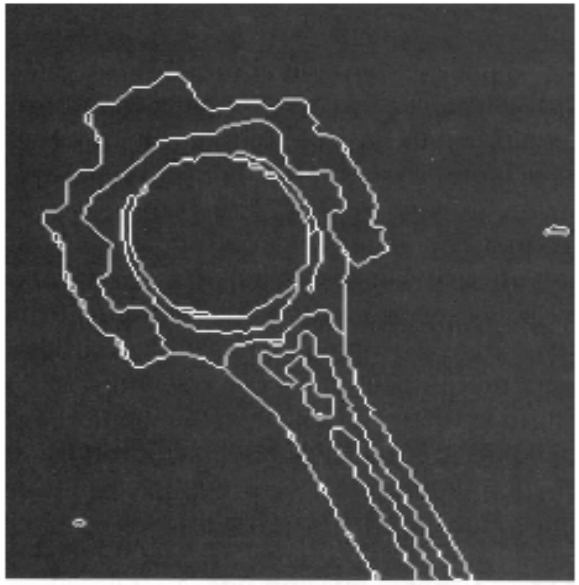


Other Steps in Edge Detection

3) Edge thinning and linking

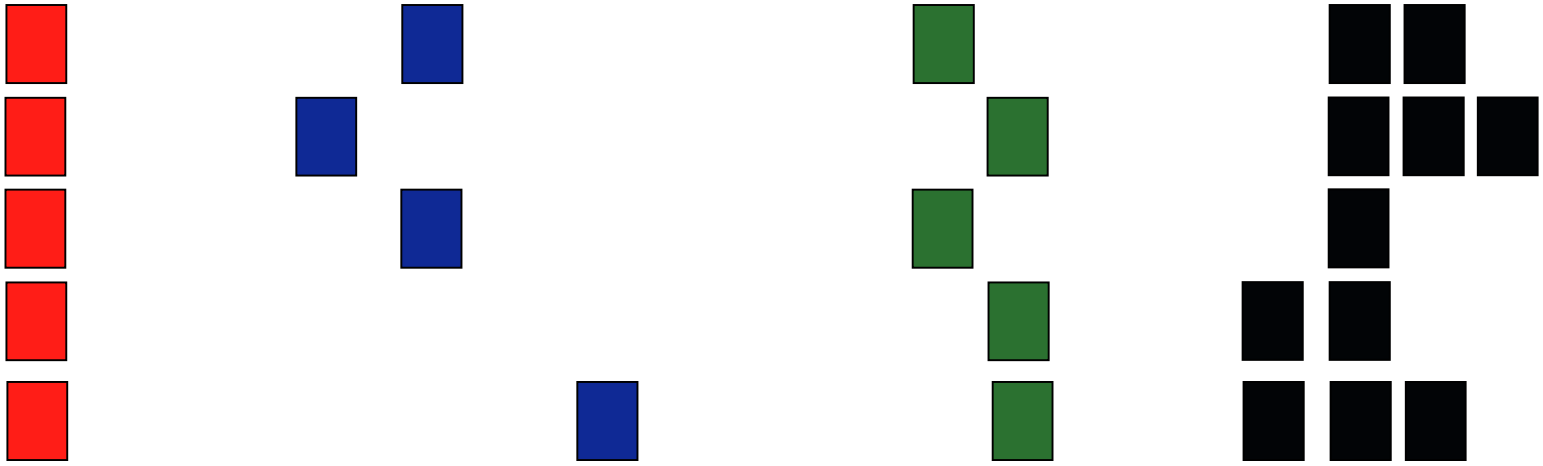
– required to obtain good contours





Factors in Edge Detection

- Examples:



True edge

Poor robustness to noise

Poor localization

Too many responses

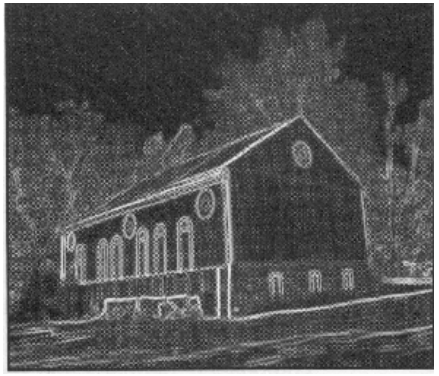
Criteria for Optimal Edge Detection

- **(1) Good detection**
  - Minimize the probability of false positives (i.e., spurious edges).
  - Minimize the probability of false negatives (i.e., missing real edges).
- **(2) Good localization**
  - Detected edges must be as close as possible to the true edges.
- **(3) Single response**
  - Minimize the number of local maxima around the true edge. (one point for the true edge)

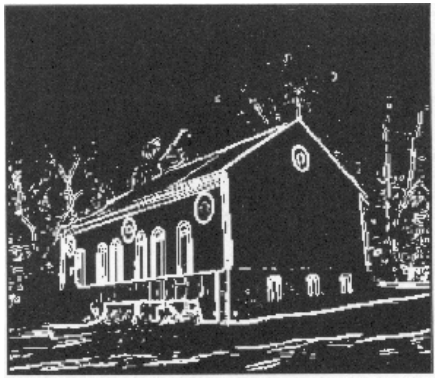
Choice of Threshold

- Trade off of threshold value.

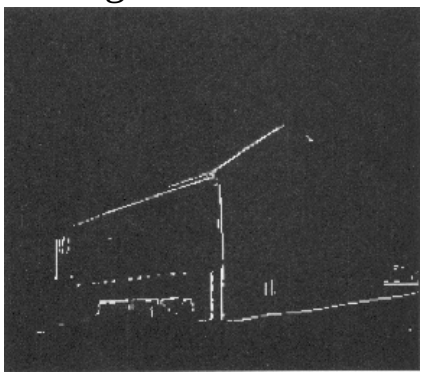
gradient magnitude



low threshold



high threshold



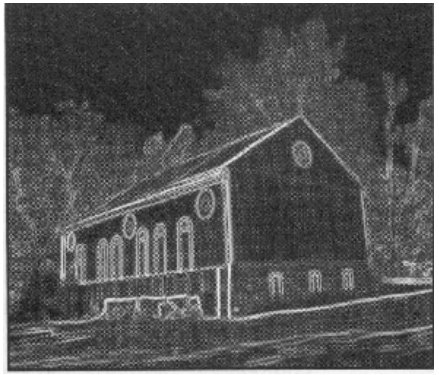
Standard thresholding

- Standard thresholding:

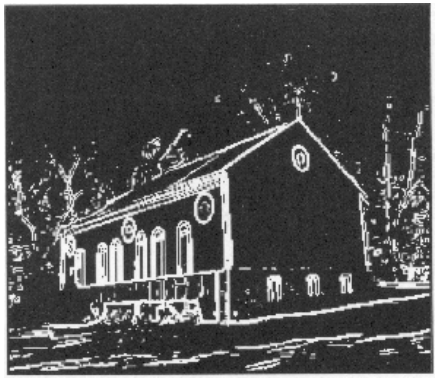
$$E(x,y) = \begin{cases} 1 & \text{if } \|\nabla f(x,y)\| > T \text{ for some threshold } T \\ 0 & \text{otherwise} \end{cases}$$

- Can only select “strong” edges.
- Does not guarantee “continuity”.

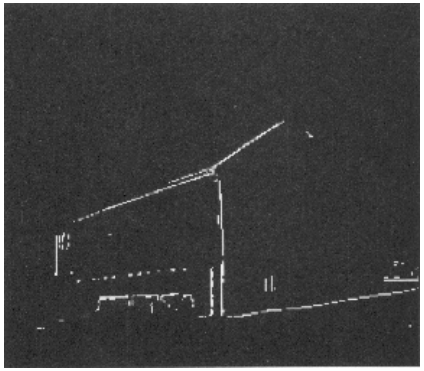
gradient magnitude



low threshold



high threshold



Hysteresis thresholding

- Hysteresis thresholding uses two thresholds:

  - low threshold t_l
  - high threshold t_h (usually, $t_h = 2t_l$)

$$
\begin{array}{rll}
 & \|\nabla f(x,y)\| \geq t_h & \text{definitely an edge} \\
t_l \geq & \|\nabla f(x,y)\| < t_h & \text{maybe an edge, depends on context} \\
 & \|\nabla f(x,y)\| < t_l & \text{definitely not an edge}
\end{array}
$$

- For "maybe" edges, decide on the edge if neighboring pixel is a strong edge.

Hysteresis Thresholding Example

original image

edge magnitude

$T = \{5, 20\}$

$T = \{20, 40\}$

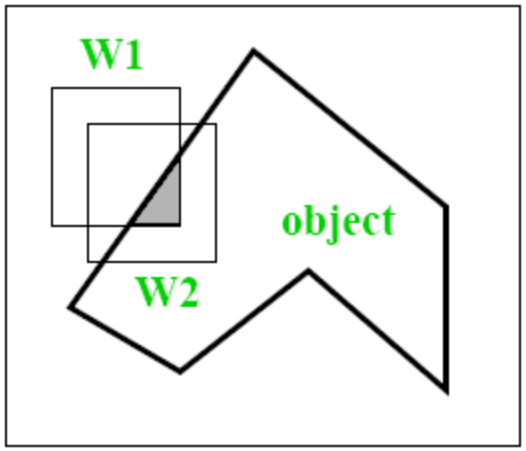
- Higher thresholds result in less edges
- In both cases, contour connectivity is preserved
  - well, almost... :-)

Edge Localization

- **Input**
  - edge magnitude (strength) $M(x, y)$
  - edge orientation $\Theta(x, y)$
- **Output**
  - binary *edge map*

  $\Rightarrow$ 1 indicates edge, 0 no edge
- Selects maxima of $M(x, y)$ that are true edge pixels
- Usable with any filter that gives magnitude and orientation
  - gradient: Canny, Prewitt
  - non-gradient: Mérő & Vassy
- Includes two basic operations
  - **non-maxima suppression** to remove 'phantom' edges
  - **hysteresis thresholding** to remove noisy maxima

Single Response



- Same piece of contour detected in windows W1 and W2
  - ⇒ 'phantom' edges parallel to 'true' edges, thick contours
- Response depends on overlap between window and contour
- Multiple response typical for *all* window-based detection tasks

Canny edge detector

- This is probably the most widely used edge detector in computer vision
- Theoretical model: step-edges corrupted by additive Gaussian noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise ratio* and localization

http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html

J. Canny, *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Source: L. Fei-Fei

Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” down to single pixel width
4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

Source: D. Lowe, L. Fei-Fei

The Canny Edge Detector

Algorithm

1. Compute f_x and f_y

$$f_x = \frac{\partial}{\partial x}(f * G) = f * \frac{\partial}{\partial x} G = f * G_x$$

$$f_y = \frac{\partial}{\partial y}(f * G) = f * \frac{\partial}{\partial y} G = f * G_y$$

$G(x, y)$ is the Gaussian function

$G_x(x, y)$ is the derivate of $G(x, y)$ with respect to x: $G_x(x, y) = \frac{-x}{\sigma^2} G(x, y)$

$G_y(x, y)$ is the derivate of $G(x, y)$ with respect to y: $G_y(x, y) = \frac{-y}{\sigma^2} G(x, y)$

2. Compute the gradient magnitude

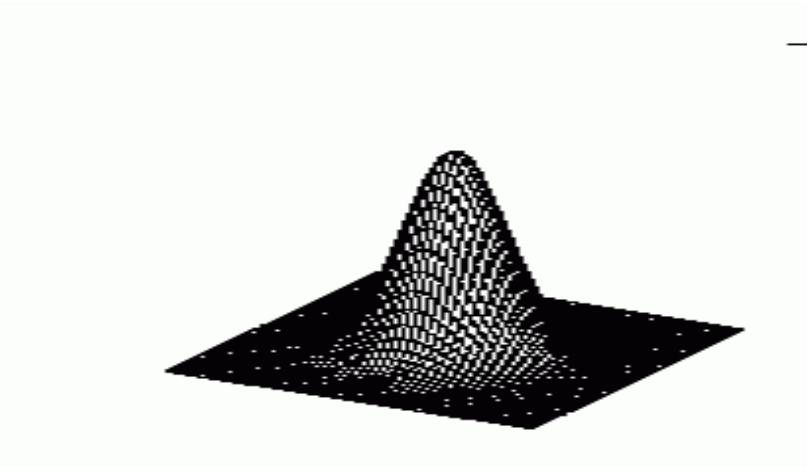
$$magn(i, j) = \sqrt{f_x^2 + f_y^2}$$

3. Apply non-maxima suppression.

4. Apply hysteresis thresholding/edge linking.

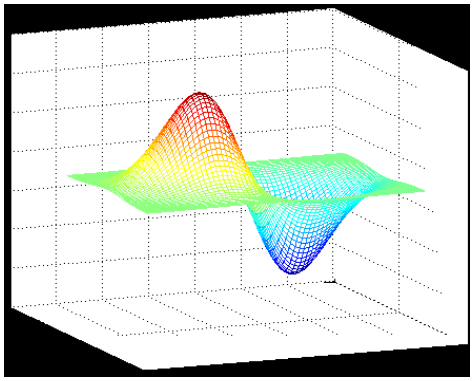
The Canny Edge Detector

- The derivative of the Gaussian:

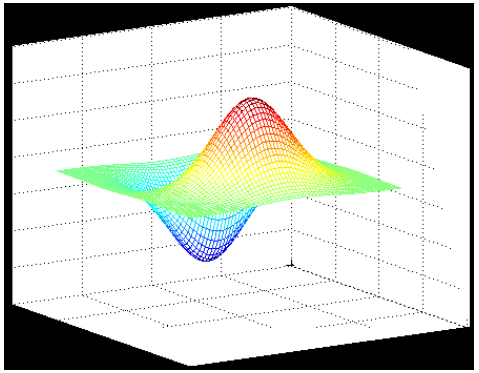


$g(x, y)$

$g_x(x, y)$

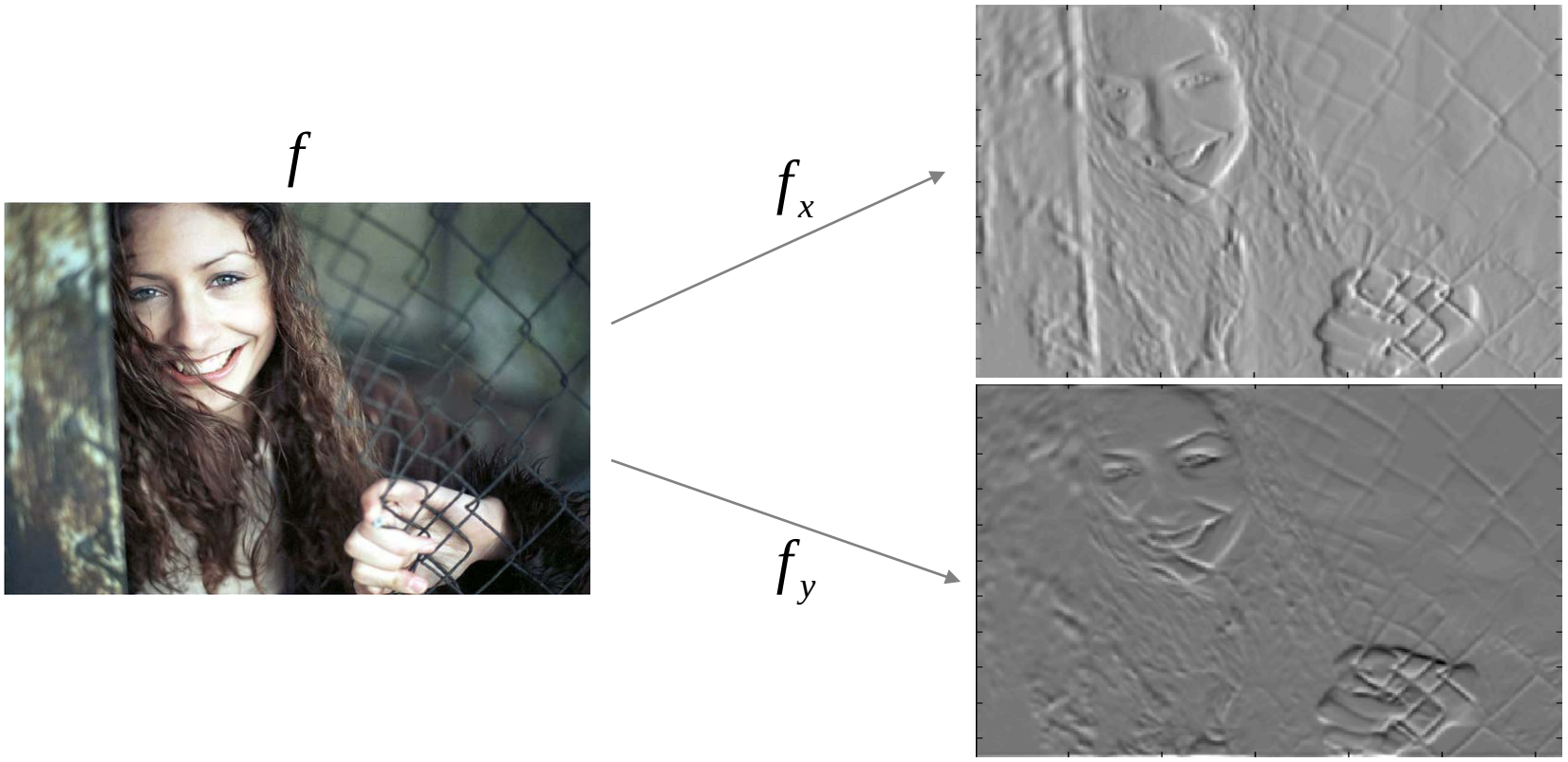


$g_y(x, y)$



The Canny Edge Detector

- Canny – smoothing and derivatives:



The Canny Edge Detector

- Canny – gradient magnitude:

image

gradient magnitude

Edge Detection

- Canny – Non-maxima suppression:

gradient magnitude

thinned

Edge Detection

- Canny – Hysteresis thresholding / Edge linking

thinned

regular

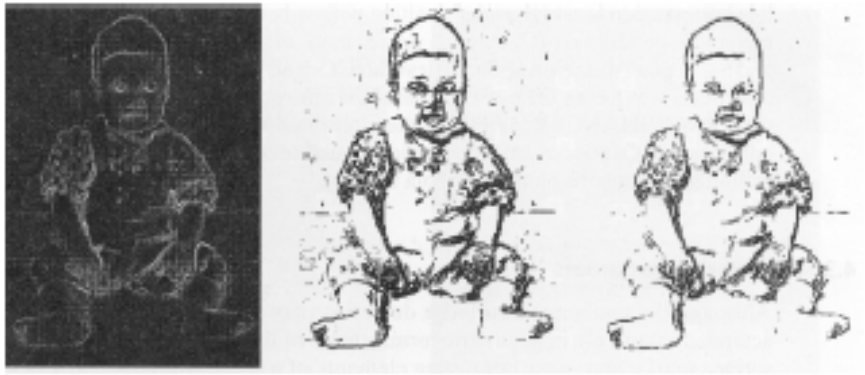
$t = 25$

Hysteresis

$High = 35$

$Low = 15$

Edge Detection



(left:Sobel, middle: thresh=35, right: thersh=50)

(Canny - left:σ=1, middle: σ=2, right: σ=3)

Edge Detection

- Hysteresis thresholding / Edge linking – cont.

Algorithm

1. Produce two thresholded images $I_1(i, j)$ and $I_2(i, j)$.

(note: since $I_2(i, j)$ was formed with a high threshold, it will contain fewer false edges but there might be gaps in the contours)

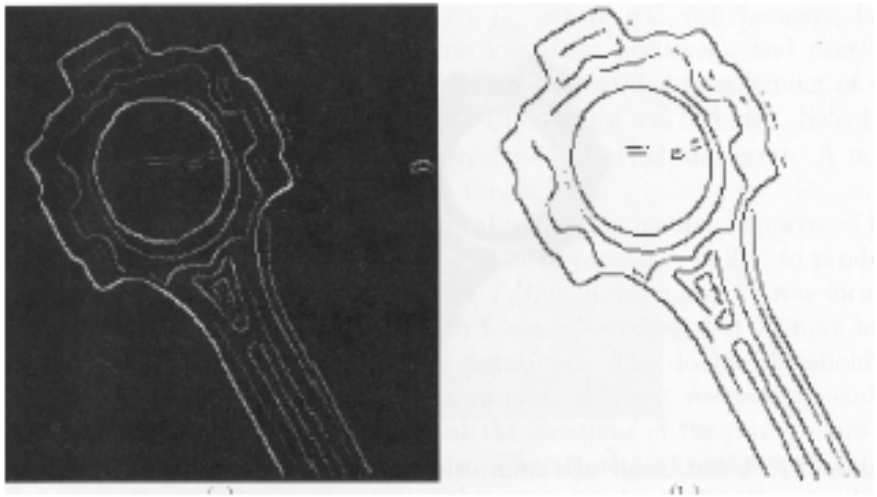
2. Link the edges in $I_2(i, j)$ into contours

   2.1 Look in $I_1(i, j)$ when a gap is found.

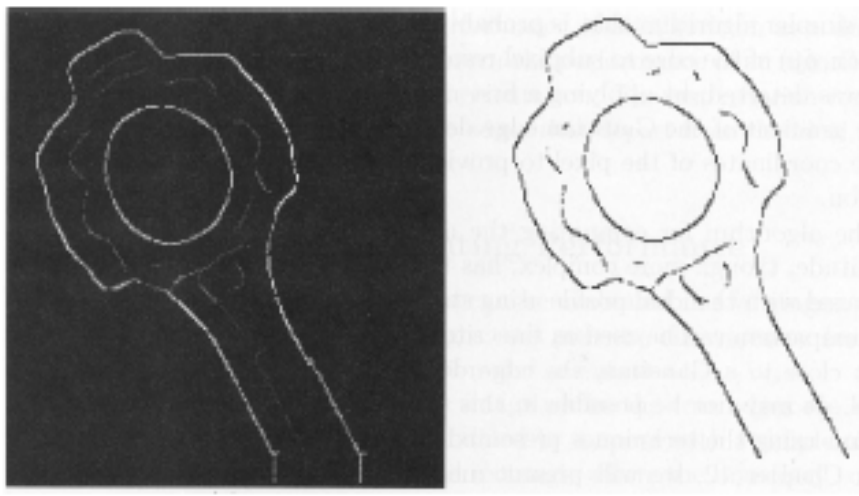
   2.2 By examining the 8 neighbors in $I_1(i, j)$, gather edge points from $I_1(i, j)$ until the gap has been bridged to an edge in $I_2(i, j)$.

  – The algorithm performs edge linking as a by-product of double-thresholding !!

Edge Detection



(Canny - 7x7 Gaussian, more details)



(Canny - 31x31 Gaussian, less details)

Properties of Canny Edge Detection

- Canny edge detector is **optimal** under assumptions:
  - noisy step edge considered
  - image noise is additive, uncorrelated and Gaussian
  - edge filter is linear
- Optimality criterion combines
  - **good detection** and
  - **good localisation**
- To satisfy **single response** criterion, two post-processing operations are used
  - *non-maxima suppression*
  - *hysteresis thresholding*

The Canny edge detector

- original image (Lena)

Compute Gradients

X-Derivative of Gaussian

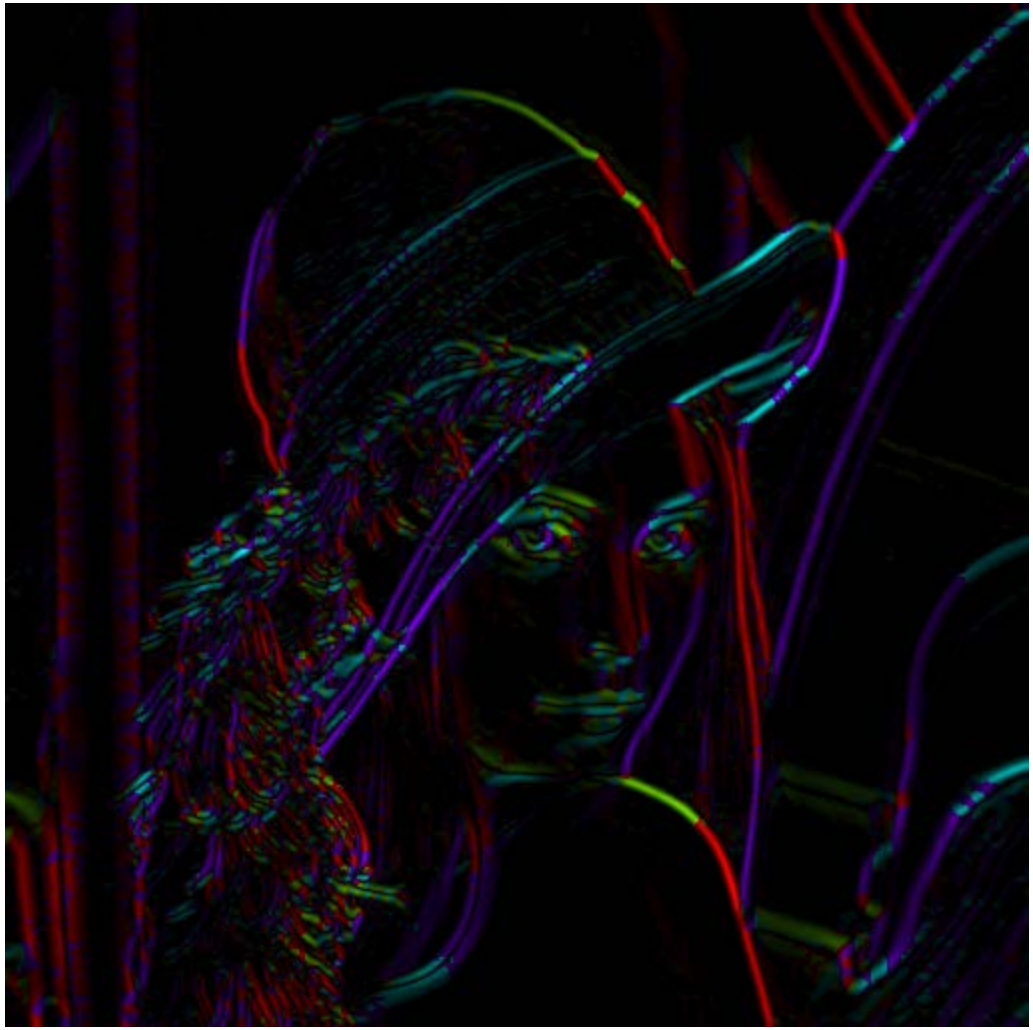
Y-Derivative of Gaussian

Gradient Magnitude

The Canny edge detector

thresholding

Get Orientation at Each Pixel



theta = atan2(-gy, gx)

The Canny edge detector

Canny Edges

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

An edge is not a line...

How can we detect ***lines*** ? Next Lectures

Canny Edge Detection: Approximation

- Original Canny filter is quite complicated
- Simple practical approximation
  - apply **Gaussian filter** to smooth image:
    $g(x,y) = f(x,y) * w_G(x,y;\sigma)$
  - $\Rightarrow$ parameter σ determines *size of filter*
  - apply **gradient operator** $\nabla g(x,y)$ to obtain edge magnitude and orientation
- Scale parameter σ is selected based on
  - desired level of detail: fine edges, global edges
  - noise level
  - localisation-detection trade-off
  - $\Rightarrow$ see template matching

Fast Approximation of Canny Edge

- Use *associativity* of linear filters

$$\nabla\big(f(x,y) * w_G(x,y)\big) = f(x,y) * \big(\nabla w_G(x,y)\big)$$

- Use **separability** of Gaussian $w_G(x,y) = w_G(x) \cdot w_G(y)$
- Obtain resulting vector filter (C is normaliser)

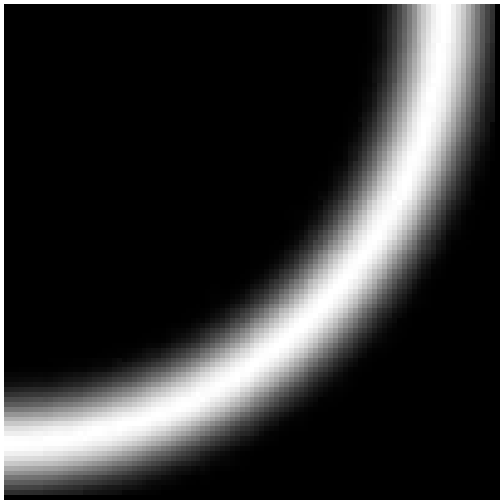
$$\nabla w_G(x,y) = \big(w_G(y) \cdot w'_G(x), w_G(x) \cdot w'_G(y)\big)$$

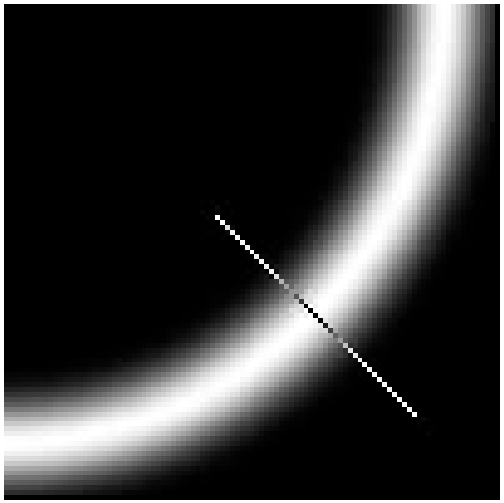
$$w'_G(x) \doteq \frac{\partial w_G(x)}{\partial x} = C \cdot x \exp\left\{-\frac{x^2}{2\sigma^2}\right\}$$

$\Rightarrow$ Filter is implemented as sequence of **1D masks**

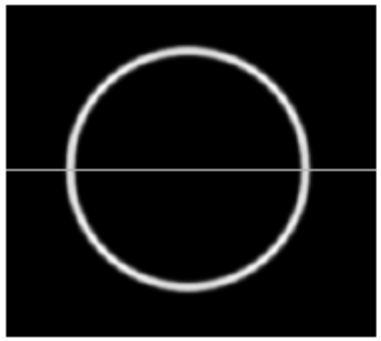
Non-Maxima Suppression

- Non-maxima suppression
  - To find the edge points, we need to find the local maxima of the gradient magnitude.
  - Broad ridges must be thinned so that only the magnitudes at the points of greatest local change remain.
  - All values along the direction of the gradient that are not peak values of a ridge are suppressed.



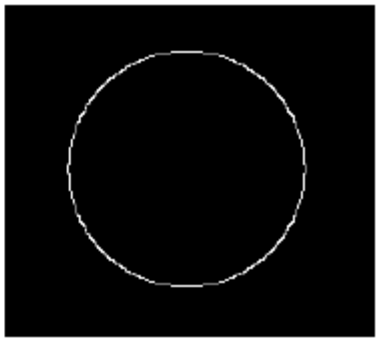


No-maxima Suppression : Example



edge magnitude

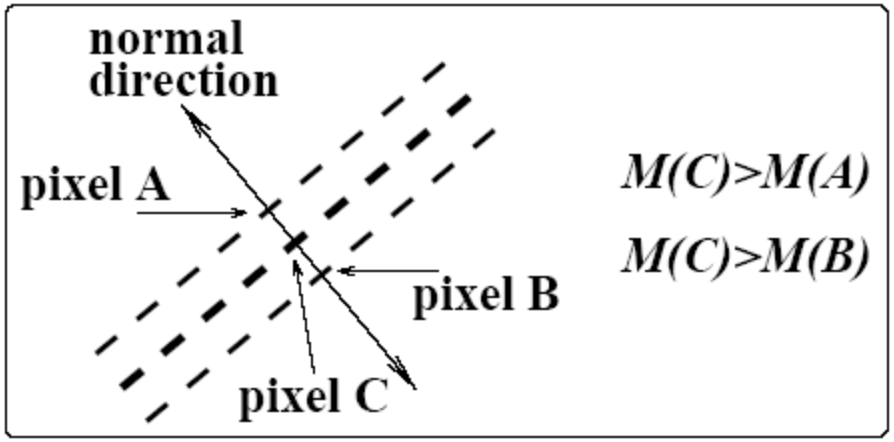
intensity profile along line



result of NMS

- Non-maxima suppression thins wide contour in edge magnitude image
- Intensity profile along indicated line is shown resized for better visibility

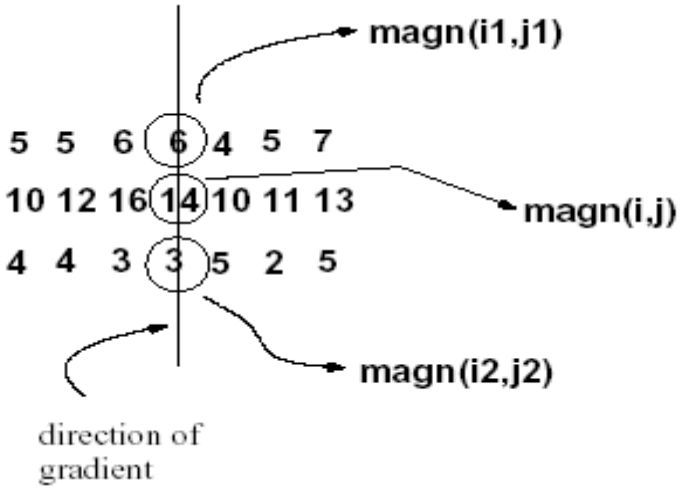
No-Maxima Suppression Explanation



- Pixel C is not deleted
  - $M(C) > M(A)$ and $M(C) > M(B)$
- Pixels A, B will be deleted
  - $M(A) < M(C)$, $M(B) < M(C)$

Edge Detection

- Non-maxima suppression – cont.



Algorithm

For each pixel (x,y) do:

if $magn(i,j) < magn(i_1,j_1)$ or $magn(i,j) < magn(i_2,j_2)$
  then $I_N(i,j) = 0$
else $I_N(i,j) = magn(i,j)$

Edge Detection

- Non-maxima suppression – cont.
  - What are the neighbors?
    - Look along gradient normal
  - Quantization of normal directions:

$\tan\theta = \textit{gradient direction}$

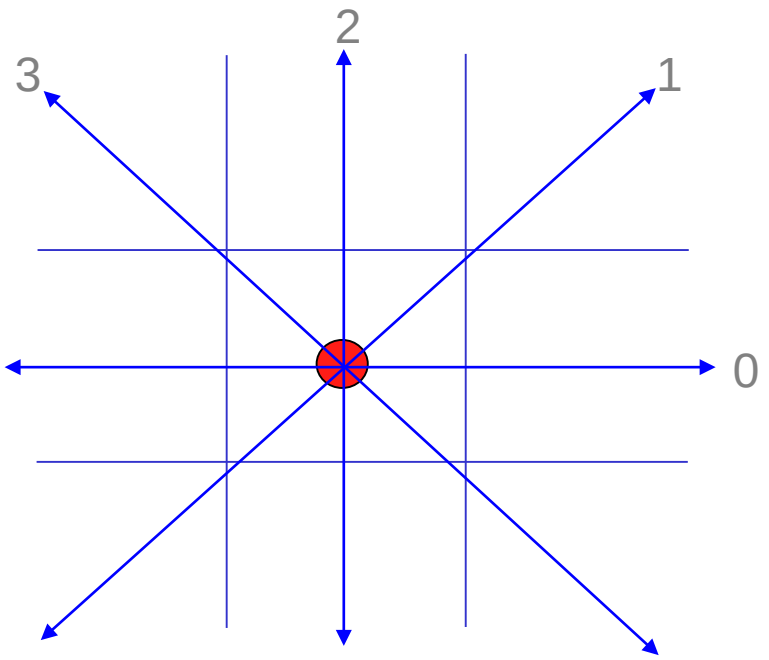
Quantization :

0 : $-0.4142 < \tan\theta \leq 0.4142$

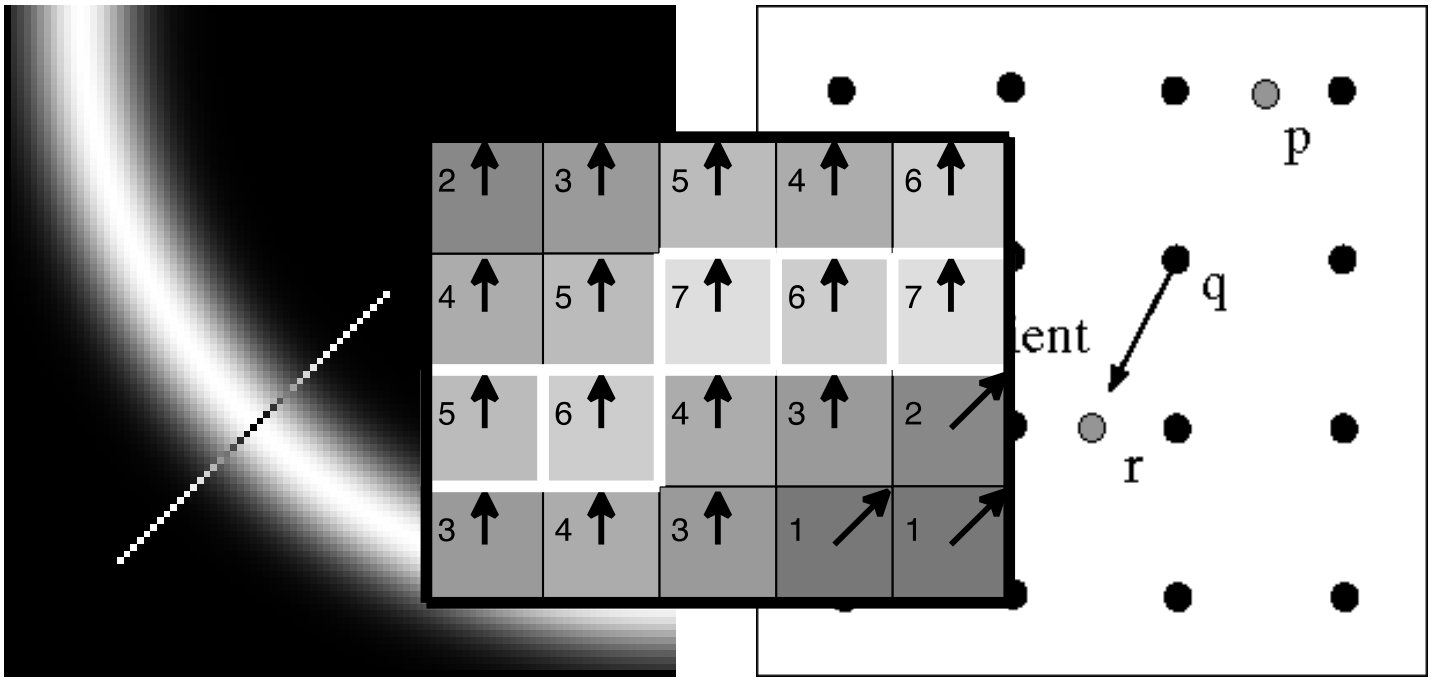
1 : $0.4142 < \tan\theta < 2.4142$

2 : $|\tan\theta| \geq 2.4142$

3 : $-2.4142 < \tan\theta \leq -0.4142$



Non-maximum suppression



- Check if pixel is local maximum along gradient direction

No-Maxima Suppression

- Due to multiple response, edge magnitude $M(x, y)$ may contain wide ridges around local maxima
- Non-maxima suppression removes non-maxima pixels **preserving connectivity** of contours

Algorithm 1: Non-maxima suppression

1. From each position (x, y), step in the two directions **perpendicular** to edge orientation $\Theta(x, y)$
2. Denote inital pixel (x, y) by C, the two neighbouring pixels in perpendicular directions by A and B
3. If $M(A) > M(C)$ or $M(B) > M(C)$, discard pixel (x, y) by setting $M(x, y) = 0$

Before Non-max Suppression

After non-max suppression

Hysteresis Thresholding

- Output of non-maxima suppression still contains noisy local maxima
- Contrast (edge strength) may vary along contour
  - $\Rightarrow$ careful thresholding of $M(x, y)$ needed to remove weak edges while preserving connectivity of contours
- Input of hysteresis thresholding
  - output of non-maxima suppression, $M_{NMS}(x, y)$
- Algorithm uses **2 thresholds**, T_{low} and T_{high}
  - edge pixel (x, y) is **strong** if $M_{NMS}(x, y) > T_{high}$
  - edge pixel (x, y) is **weak** if $M_{NMS}(x, y) \leq T_{low}$
  - all other pixels are **candidates**

Edge Detection

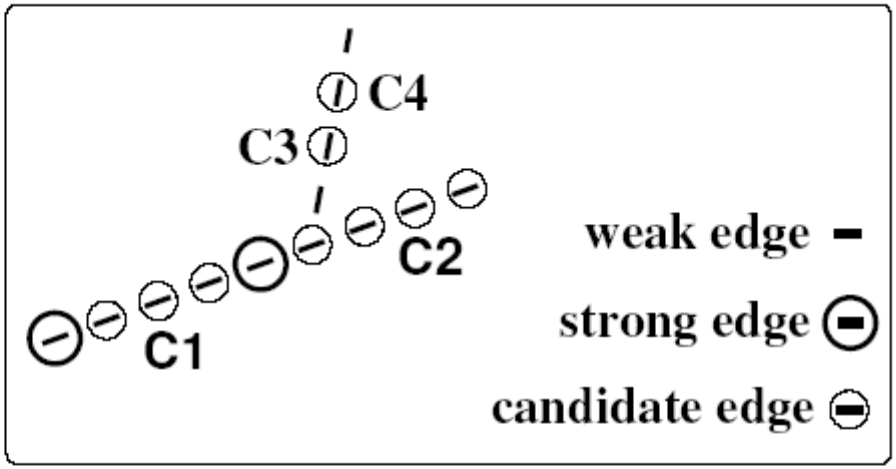
- Hysteresis thresholding / Edge linking
  - The output of non-maxima suppression still contains the local maxima created by noise.
  - Can we get rid of them just by using a single threshold?
    - if we set a low threshold, some noisy maxima will be accepted too.
    - if we set a high threshold, true maxima might be missed (the value of true maxima will fluctuate above and below the threshold, fragmenting the edge).
  - A more effective scheme is to use two thresholds:
    - a low threshold t_l
    - a high threshold t_h
    - usually, $t_h \approx 2t_l$

Hysteresis Thresholding: Algorithm

Algorithm 2: Hysteresis thresholding

1. In each position of (x, y)
   - discard edge pixel (x, y) if it is **weak**
   - output edge pixel (x, y) if it is **strong**
2. If edge pixel is **candidate**,
   - follow chain of connected local maxima in both directions **along the edge**
   - stop when $M_{NMS}(x, y) \leq T_{low}$
3. Make decision upon the starting candidate pixel
   - output it if it is **connected to a strong pixel**
   - otherwise, do not output it

Principle of Hysteresis Thresholding



- Candidate edges C1 and C2 are output
- Candidate edges C3 and C4 are not

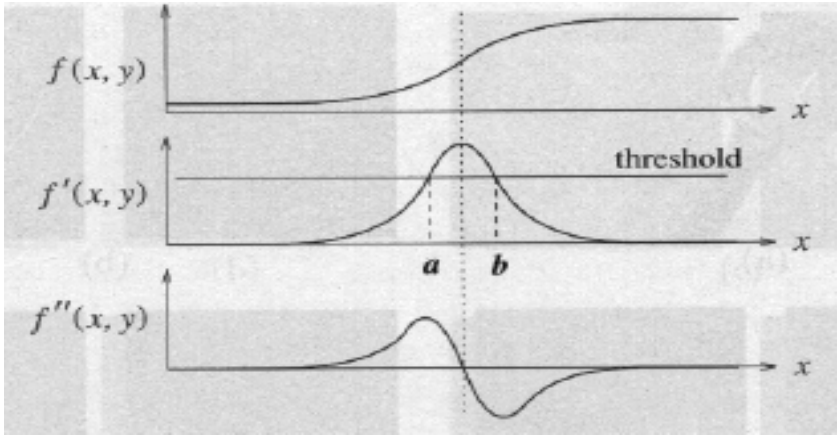
Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels

Final Canny Edges

Edge Detection Using the 2nd Derivative

- Edge points can be detected by finding the zero-crossings of the second derivative.



- There are two operators in 2D that correspond to the second derivative:
  - Laplacian
  - Second directional derivative

Edge Detection Using the 2nd Derivative

The *Laplacian* is defined mathematically as

$$\nabla^2 = \nabla \cdot \nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

When we apply it to an image, we get

$$\nabla^2 f = \left(\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \right) I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Edge Detection Using the 2nd Derivative

- The Laplacian:

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

Approximating $\nabla^2 f$:

$$\frac{\partial^2 f}{\partial x^2} = f(i, j+1) - 2f(i, j) + f(i, j-1)$$

$$\frac{\partial^2 f}{\partial y^2} = f(i+1, j) - 2f(i, j) + f(i-1, j)$$

$$\nabla^2 f = -4f(i, j) + f(i, j+1) + f(i, j-1) + f(i+1, j) + f(i-1, j)$$

Edge Detection Using the 2nd Derivative

- Example:

Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

$$\nabla^2 f = -4z_5 + (z_2 + z_4 + z_6 + z_8)$$

- The Laplacian can be implemented using the mask:

0	1	0
1	-4	1
0	1	0

- Example:

5	5	5	5	5	5
5	5	5	5	5	5
5	5	10	10	10	10
5	5	10	10	10	10
5	5	5	10	10	10
5	5	5	5	10	10

-	-	-	-	-	-
-	0	-5	-5	-5	-
-	-5	10	5	5	-
-	-5	10	0	0	-
-	0	-10	10	0	-
-	-	-	-	-	-

Variations of Laplacian

0.5	0.0	0.5
1.0	-4.0	1.0
0.5	0.0	0.5

+

0.5	1.0	0.5
0.0	-4.0	0.0
0.5	1.0	0.5

=

1	1	1
1	-8	1
1	1	1

1	-2	1
1	-2	1
1	-2	1

+

1	1	1
-2	-2	-2
1	1	1

=

2	-1	2
-1	-4	-1
2	-1	2

Edge Detection Using the 2nd Derivative

- Properties of the Laplacian
  - It is an isotropic operator.
  - It is cheaper to implement (one mask only).
  - It does not provide information about edge direction.
  - It is more sensitive to noise (differentiates twice).
- Find zero crossings
  - Scan along each row, record an edge point at the location of zero-crossing.
  - Repeat above step along each column

Edge Detection Using the 2nd Derivative

- How do we estimate the edge strength?
- Four cases of zero-crossings :
  - {+,-}
  - {+,0,-}
  - {-,+}
  - {-,0,+}
- Slope of zero-crossing {a, -b} is |a+b|.
- To mark an edge:
  - compute slope of zero-crossing
  - apply a threshold to slope

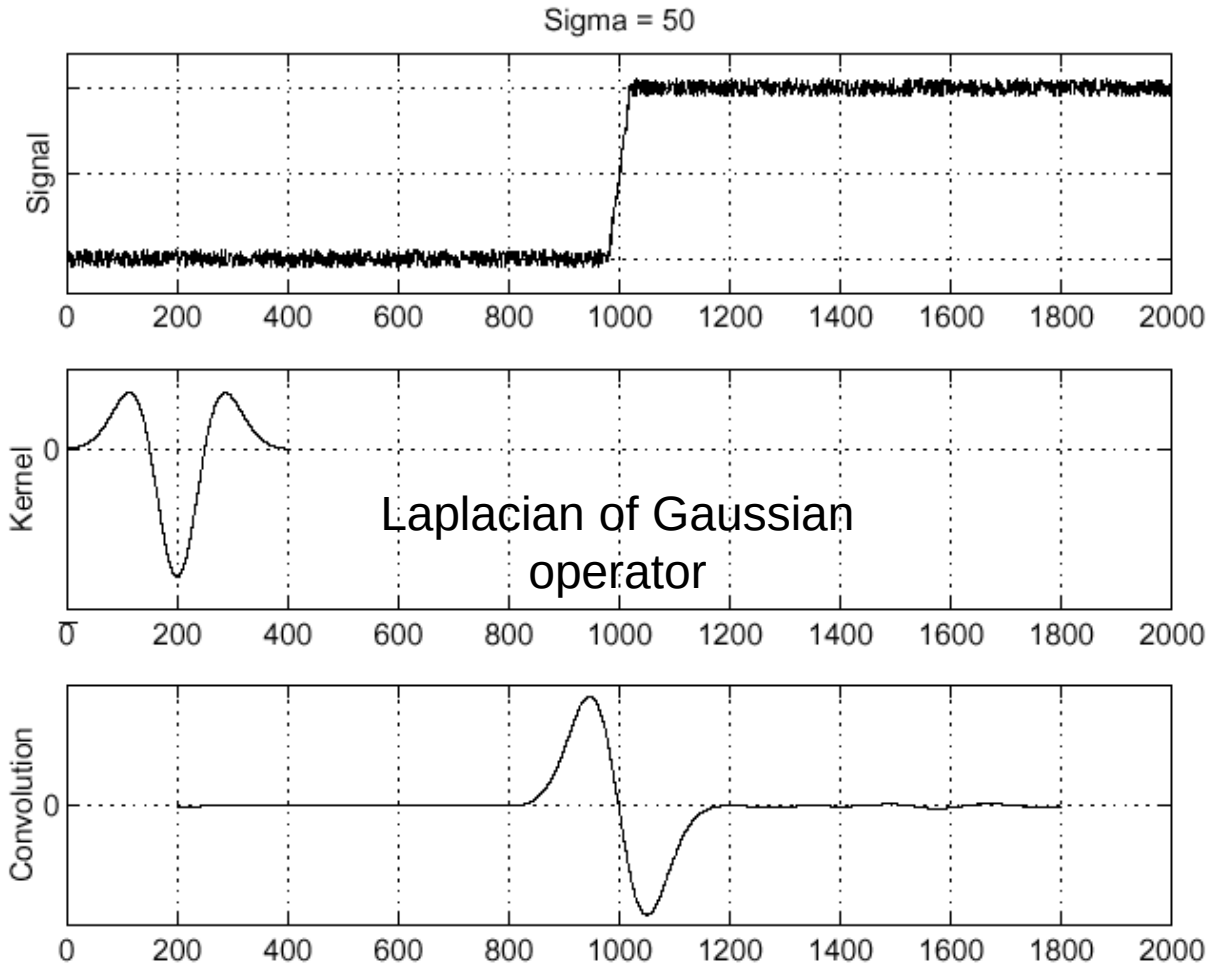
Laplacian of Gaussian (LoG)

- Look for zero-crossings of: $\frac{\partial^2}{\partial x^2}(h \star f)$

f

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

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



Laplacian of Gaussian (LoG) (Marr-Hildreth operator)

- To reduce the noise effect, the image is first smoothed.
- When the filter chosen is a Gaussian, we call it the LoG edge detector.

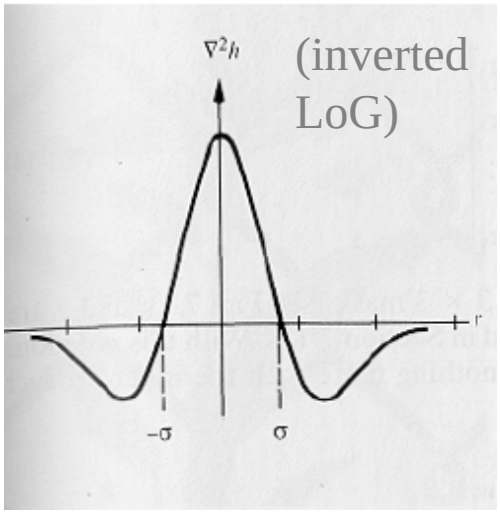
$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

σ controls smoothing

- It can be shown that:

$$\nabla^2[f(x,y) * G(x,y)] = \nabla^2 G(x,y) * f(x,y)$$

$$\nabla^2 G(x,y) = \left(\frac{r^2 - 2\sigma^2}{\sigma^4}\right) e^{-r^2/2\sigma^2}, \quad (r^2 = x^2 + y^2)$$

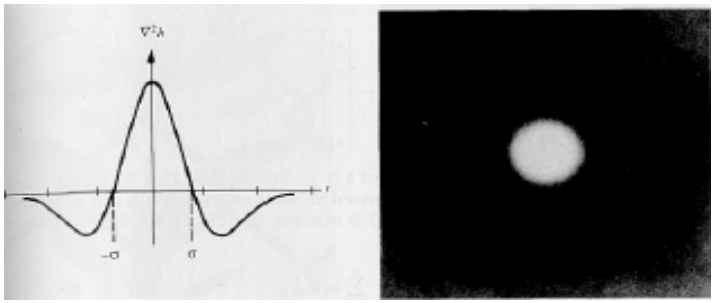


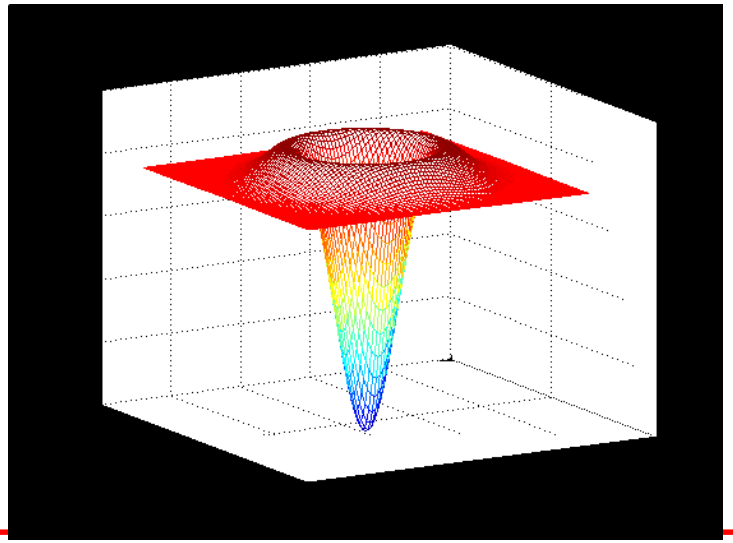
Edge Detection Using the 2nd Derivative

- The Laplacian-of-Gaussian (LOG) – cont.
  - It can be shown that:

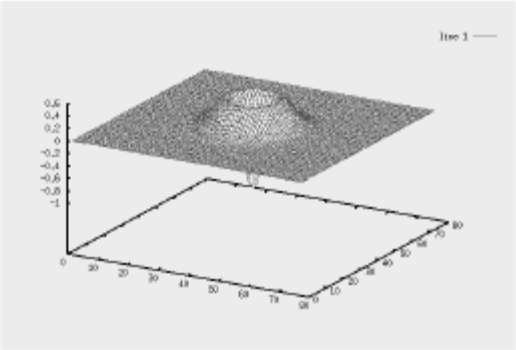
$$\nabla^2[f(x,y) * G(x,y)] = \nabla^2 G(x,y) * f(x,y)$$

$$\nabla^2 G(x,y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

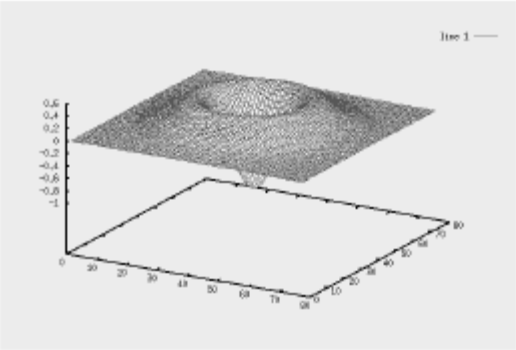




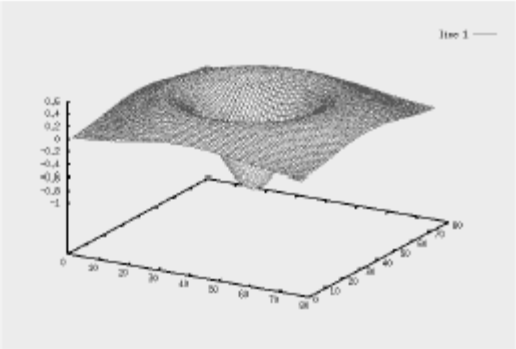
LoG



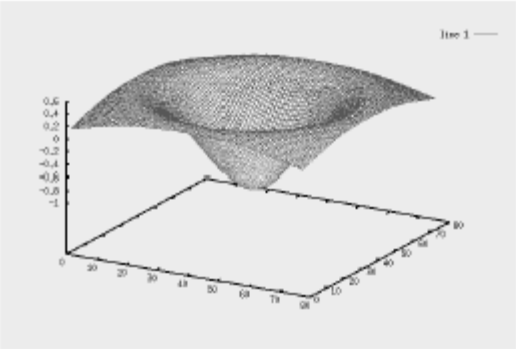
$\sigma = 5$



$\sigma = 10$



$\sigma = 15$



$\sigma = 20$

Laplacian of Gaussian (LoG) - Example

(inverted LoG)

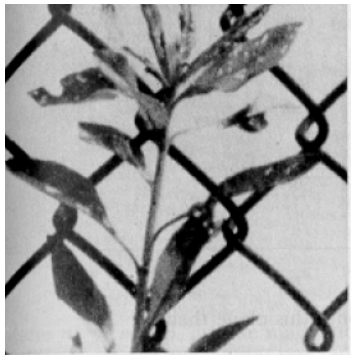
5 × 5 Laplacian of Gaussian mask

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

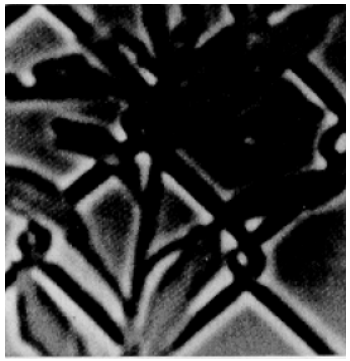
(inverted LoG)

17 × 17 Laplacian of Gaussian mask

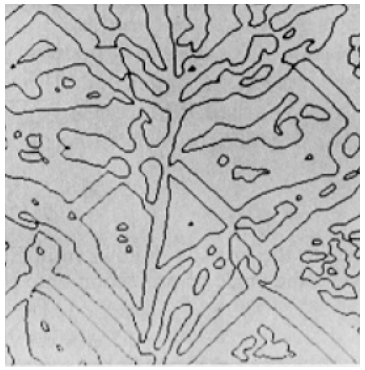
0	0	0	0	0	0	−1	−1	−1	−1	−1	0	0	0	0	0	0
0	0	0	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	0	0	0
0	0	−1	−1	−1	−2	−3	−3	−3	−3	−3	−2	−1	−1	−1	0	0
0	0	−1	−1	−2	−3	−3	−3	−3	−3	−3	−3	−2	−1	−1	0	0
0	−1	−1	−2	−3	−3	−3	−2	−3	−2	−3	−3	−3	−2	−1	−1	0
0	−1	−2	−3	−3	−3	0	2	4	2	0	−3	−3	−3	−2	−1	0
−1	−1	−3	−3	−3	0	4	10	12	10	4	0	−3	−3	−3	−1	−1
−1	−1	−3	−3	−2	2	10	18	21	18	10	2	−2	−3	−3	−1	−1
−1	−1	−3	−3	−3	4	12	21	24	21	12	4	−3	−3	−3	−1	−1
−1	−1	−3	−3	−2	2	10	18	21	18	10	2	−2	−3	−3	−1	−1
−1	−1	−3	−3	−3	0	4	10	12	10	4	0	−3	−3	−3	−1	−1
0	−1	−2	−3	−3	−3	0	2	4	2	0	−3	−3	−3	−2	−1	0
0	−1	−1	−2	−3	−3	−3	−2	−3	−2	−3	−3	−3	−2	−1	−1	0
0	0	−1	−1	−2	−3	−3	−3	−3	−3	−3	−3	−2	−1	−1	0	0
0	0	−1	−1	−1	−2	−3	−3	−3	−3	−3	−2	−1	−1	−1	0	0
0	0	0	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	0	0	0
0	0	0	0	0	0	−1	−1	−1	−1	−1	0	0	0	0	0	0



filtering



zero-crossings



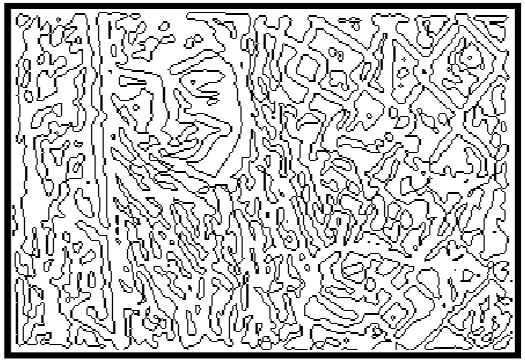
Edge Detection Using the 2nd Derivative

- The Laplacian-of-Gaussian (LOG) – cont.

I

$I * (\Delta^2 G)$

Zero crossings of $I * (\Delta^2 G)$



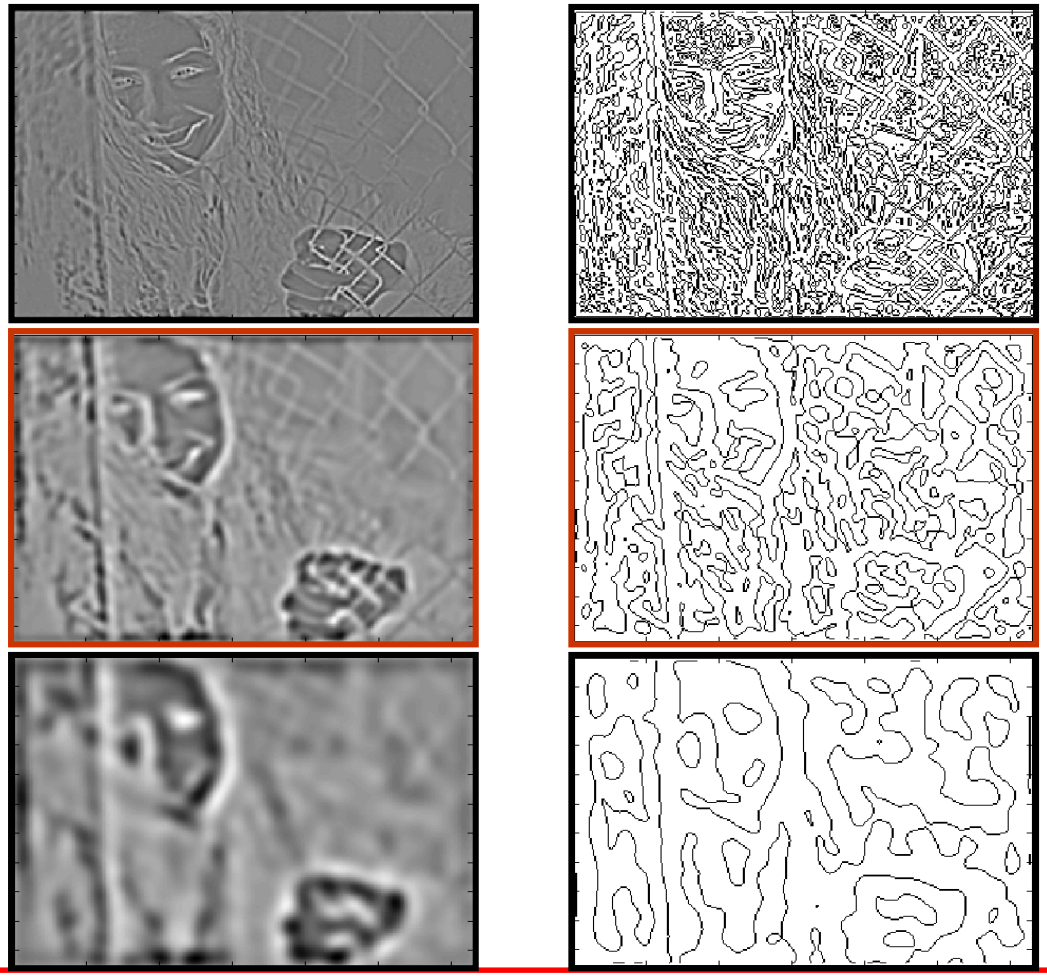
Edge Detection Using the 2nd Derivative

- The Laplacian-of-Gaussian (LOG) – cont.

$\sigma = 1$

$\sigma = 3$

$\sigma = 6$



Gradient vs LoG

- Gradient works well when the image contains sharp intensity transitions and low noise.
- Zero-crossings of LOG offer better localization, especially when the edges are not very sharp.

step edge

2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8

0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0

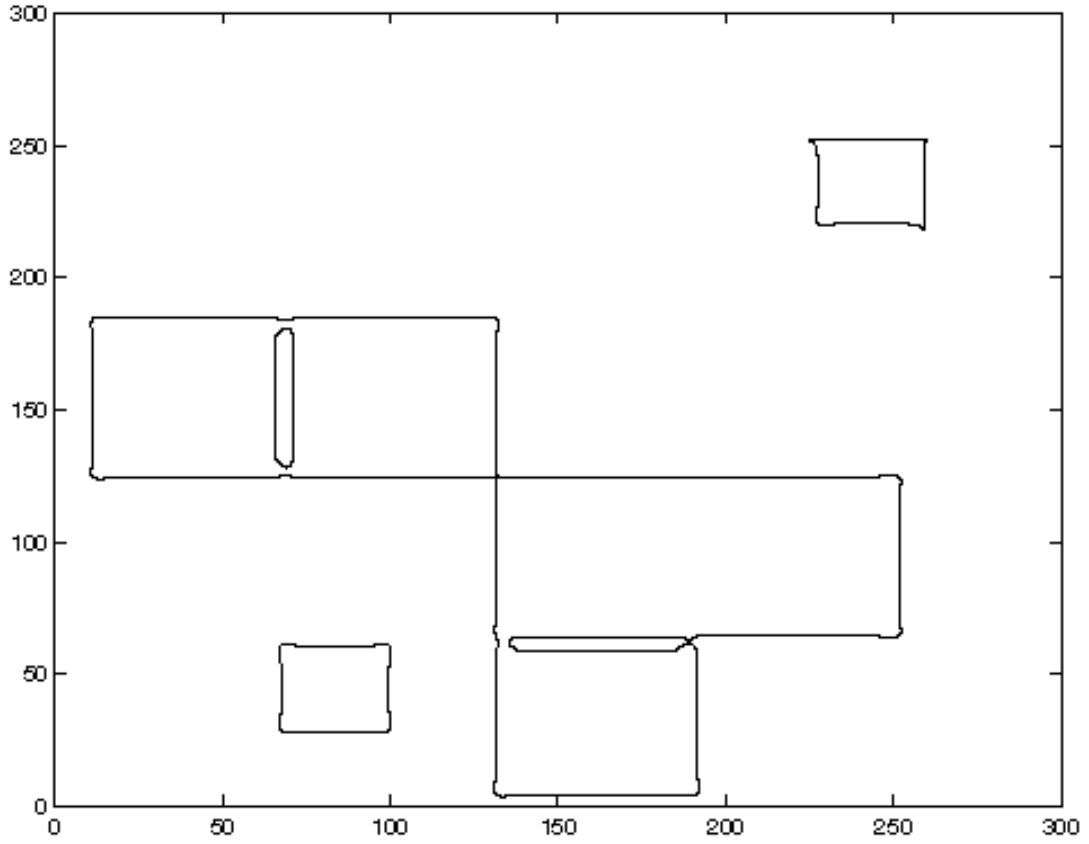
ramp edge

2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8

0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0

Edge Detection Using the 2nd Derivative

- Disadvantage of LOG edge detection:
  - Does not handle corners well

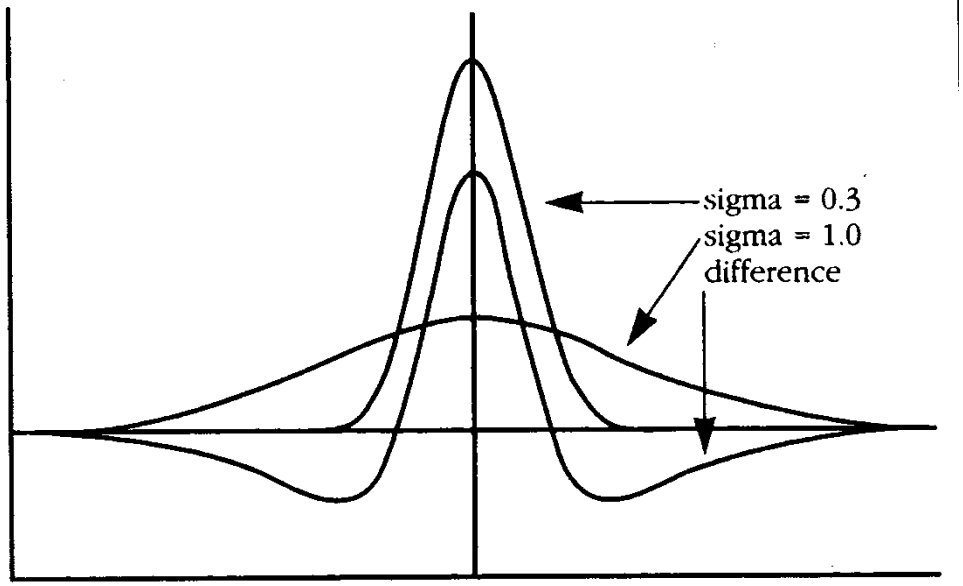


Difference of Gaussians (DoG)

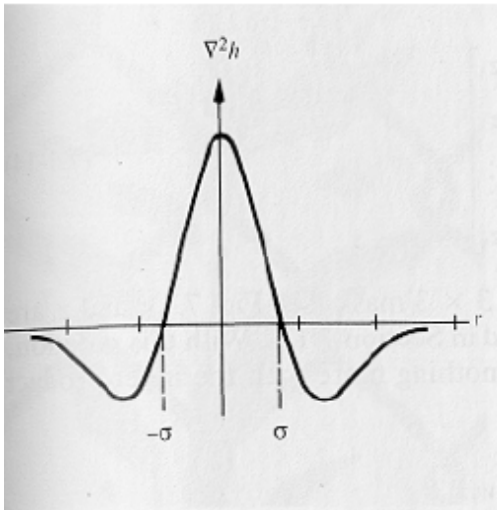
- The Laplacian of Gaussian can be approximated by the difference between two Gaussian functions:

$$\nabla^2 G \approx G(x, y; \sigma_1) - G(x, y; \sigma_2)$$

approximation



actual LoG



Difference of Gaussians (DoG) (cont'd)

$$\nabla^2 G \approx G(x, y; \sigma_1) - G(x, y; \sigma_2)$$

$\sigma = 1$

$\sigma = 2$

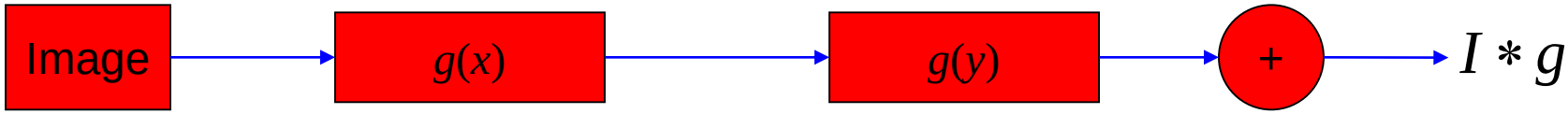
(b)-(a)

Ratio (σ_1/σ_2) for best approximation is about 1.6.
(Some people like $\sqrt{2}$.)

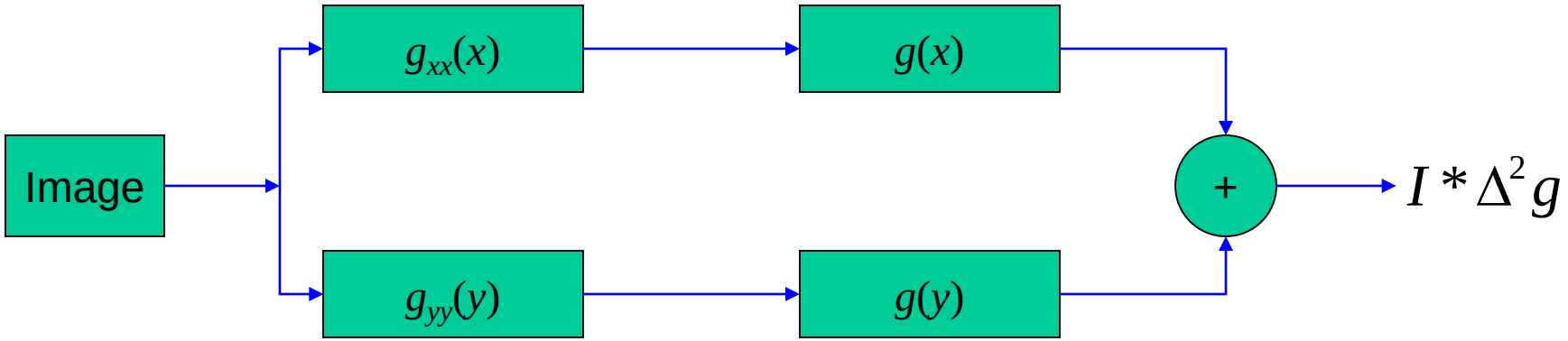
Edge Detection Using the 2nd Derivative

- Separability

Gaussian Filtering



Laplacian-of-Gaussian Filtering



Edge Detection Using the 2nd Derivative

- Separability
  - Gaussian:
    - A 2-D Gaussian can be separated into two 1-D Gaussians
    - Perform 2 convolutions with 1-D Gaussians

$$h(x,y) = I(x,y) * g(x,y)$$ n^2 multiplications per pixel

$$h(x,y) = \left(I(x,y) * g_1(x)\right) * g_2(y)$$ $2n$ multiplications per pixel

$$g_1 = \begin{bmatrix} .011 & .13 & .6 & 1 & .6 & .13 & .011 \end{bmatrix} \qquad g_2 = \begin{bmatrix} .011 \\ .13 \\ .6 \\ 1 \\ .6 \\ .13 \\ .011 \end{bmatrix}$$

Edge Detection Using the 2nd Derivative

- Separability
  - Laplacian-of-Gaussian:

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I = I * (\Delta^2 g)$$

Requires n^2 multiplications per pixel

$$\Delta^2 S = (I * g_{xx}(x)) * g(x) + (I * g_{yy}(y)) * g(y)$$

Requires $4n$ multiplications per pixel

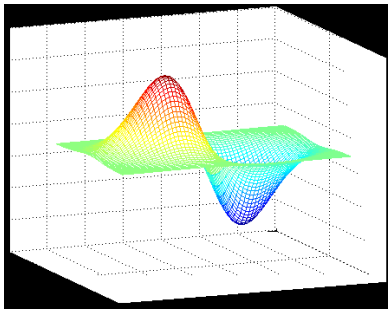
Edge Detection Using the 2nd Derivative

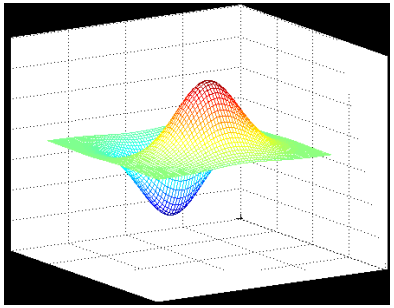
- Marr-Hildteth (LOG) Algorithm:
  - Compute LoG
    - Use one 2D filter: $\Delta^2 g(x,y)$
    - Use four 1D filters: $g(x),\ g_{xx}(x),\ g(y),\ g_{yy}(y)$
  - Find zero-crossings from each row and column
  - Find slope of zero-crossings
  - Apply threshold to slope and mark edges

Edge Detection Using the 2nd Derivative

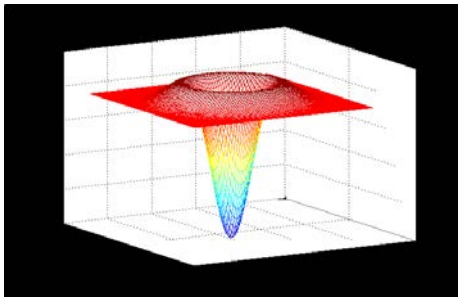
- Disadvantage of LOG edge detection:
  - Does not handle corners well
  - Why?

The derivative of the Gaussian:





The Laplacian of the Gaussian:
(unoriented)



OTHER EDGE DETECTORS

Directional Derivative

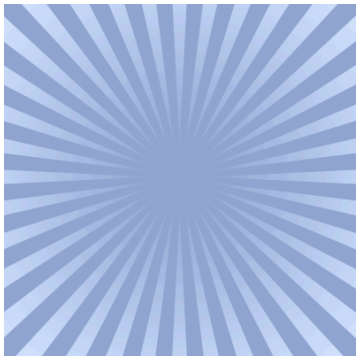
$$\nabla f \quad grad(f) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \qquad \nabla^2 f = \left(\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \right) I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

- The partial derivatives of f(x,y) will give the slope $\partial f/\partial x$ in the positive x direction and the slope $\partial f/\partial y$ in the positive y direction.

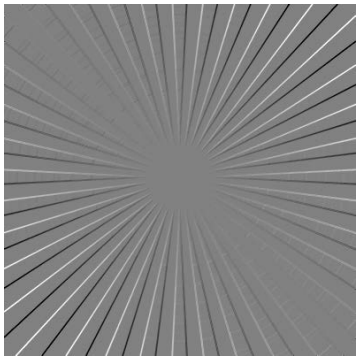
- We can generalize the partial derivatives to calculate the slope in any direction (i.e., *directional derivative*).

Directional Derivative (cont'd)

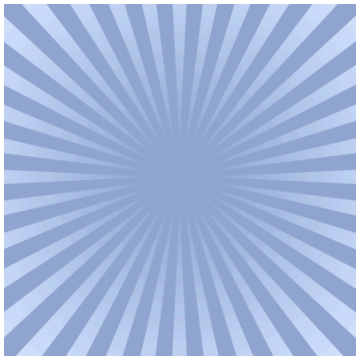
- Directional derivative computes intensity changes in a specified direction.

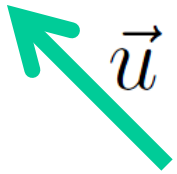
f 

Compute
derivative
in direction u



Directional Derivative (cont'd)

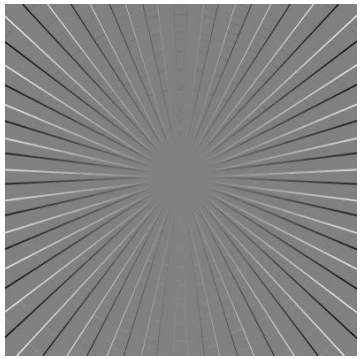
f 



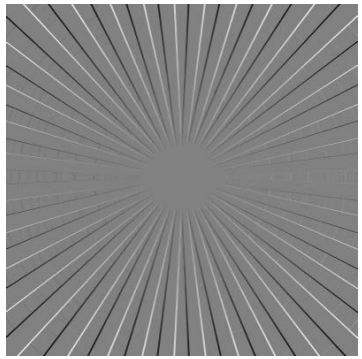
(From vector calculus)

$$\nabla_{\vec{u}} f(\vec{x}) = \nabla f(\vec{x}) \cdot \vec{u}$$

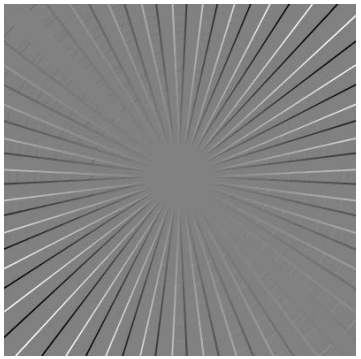
Directional derivative is a linear combination of partial derivatives.



$\frac{\partial f}{\partial x} \cdot u_x$ +



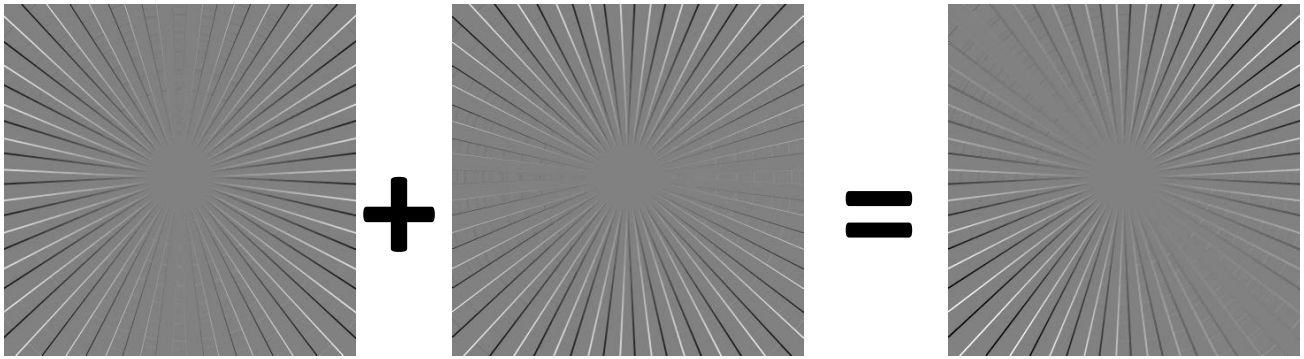
$\frac{\partial f}{\partial y} \cdot u_y$ =



$\nabla_{\vec{u}} f$

Directional Derivative (cont'd)

$$\cos\theta = \frac{u_x}{u},\ \sin\theta = \frac{u_y}{u} \quad \xrightarrow{\|u\|=1} \quad u_x = \cos\theta,\ u_y = \sin\theta$$



$\frac{\partial f}{\partial x} \cdot \cos\theta$ $\frac{\partial f}{\partial y} \cdot \sin\theta$ $\nabla_{\vec{u}} f$

- ***Compass Masks:***

  - The *Kirsch* and *Robinson* edge detection masks are called compass masks since they are defined by taking a single mask and rotating it to the eight major compass orientations: North, Northwest, West, Southwest, South, Southeast, East, and Northeast

- The edge magnitude is defined as the maximum value found by the convolution of each of the masks with the image

- The edge direction is defined by the mask that produces the maximum magnitude

- Any of the edge detection masks can be extended by rotating them in a manner like the compass masks, which allows us to extract explicit information about edges in any direction

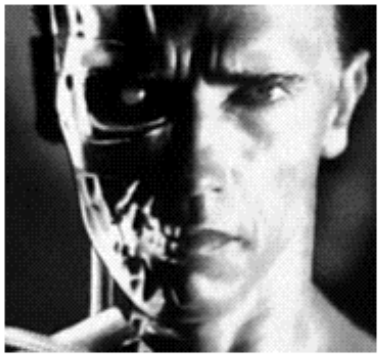
- The Kirsch-Robinson masks are as follows:

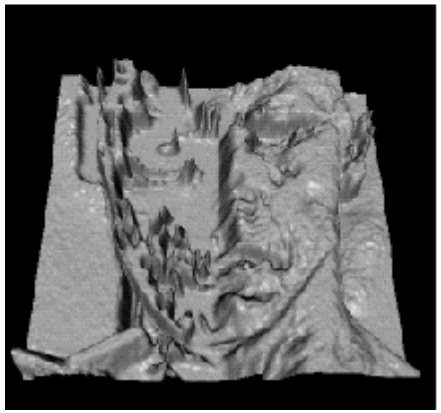
$$r_0\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}\quad r_1\begin{bmatrix}0 & 1 & 2\\ -1 & 0 & 1\\ -2 & -1 & 0\end{bmatrix}\quad r_2\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}\quad r_3\begin{bmatrix}2 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -2\end{bmatrix}$$

$$r_4\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}\quad r_5\begin{bmatrix}0 & -1 & -2\\ 1 & 0 & -1\\ 2 & 1 & 0\end{bmatrix}\quad r_6\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}\quad r_7\begin{bmatrix}-2 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 2\end{bmatrix}$$

Facet Model Based Edge Detector (Haralick)

- Assumes that an image is an array of samples of a continuous function f(x,y).
- Reconstructs f(x,y) from sampled pixel values.
- Uses directional derivatives which are computed **analytically** (i.e., without using discrete approximations).

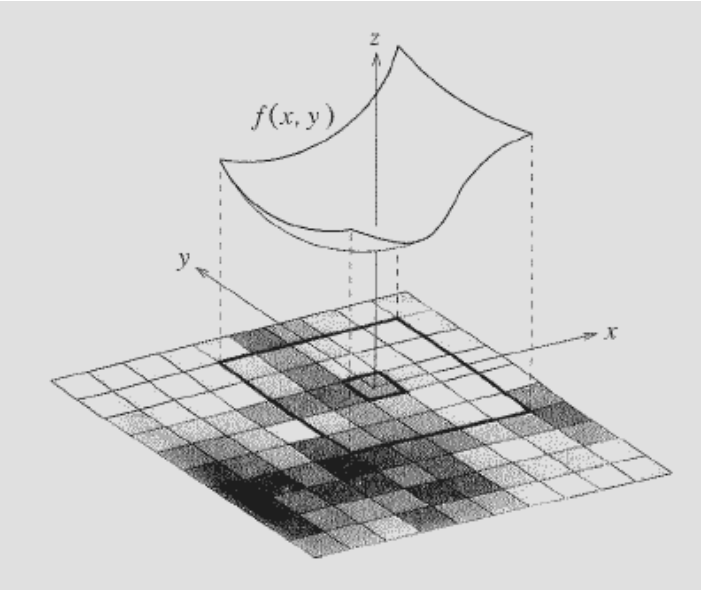




z=f(x,y)

Facet Model

- For complex images, f(x,y) could contain extremely high powers of x and y.
- **Idea:** model f(x,y) as a piece-wise function.
- Approximate each pixel value by fitting a bi-cubic polynomial in a small neighborhood around the pixel (facet).



$$f(x,y) = k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 + k_7x^3 + k_8x^2y + k_9xy^2 + k_{10}y^3.$$

Facet Model

Steps

(1) Fit a bi-cubic polynomial to a small neighborhood of each pixel (this step provides smoothing too).

(2) Compute **(analytically)** the second and third directional derivatives in the direction of gradient.

(3) Find points where (i) the second derivative is equal to zero and (ii) the third derivative is negative.

Fitting bi-cubic polynomial

$$f(x,y) = k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 xy + k_6 y^2 + k_7 x^3 + k_8 x^2 y + k_9 x y^2 + k_{10} y^3.$$

- If a 5 x 5 neighborhood is used, the masks below can be used to compute the coefficients.
  - Equivalent to least-squares (e.g., SVD)

$\frac{1}{175}$

-13	2	7	2	-13
2	17	22	17	2
7	22	27	22	7
2	17	22	17	2
-13	2	7	2	-13

k_1

$\frac{1}{420}$

31	-5	-17	-5	31
-44	-62	-68	-62	-44
0	0	0	0	0
44	62	68	62	44
-31	5	17	5	-31

k_2

$\frac{1}{70}$

2	-1	-2	-1	2
2	-1	-2	-1	2
2	-1	-2	-1	2
2	-1	-2	-1	2
2	-1	-2	-1	2

k_6

$\frac{1}{60}$

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

k_7

$\frac{1}{140}$

-4	-2	0	2	4
2	1	0	-1	-2
4	2	0	-2	-4
2	1	0	-1	-2
-4	-2	0	2	4

k_8

$\frac{1}{420}$

31	-44	0	44	-31
-5	-62	0	62	5
-17	-68	0	68	17
-5	-62	0	62	5
31	-44	0	44	-31

k_3

$\frac{1}{70}$

2	2	2	2	2
-1	-1	-1	-1	-1
-2	-2	-2	-2	-2
-1	-1	-1	-1	-1
2	2	2	2	2

k_4

$\frac{1}{100}$

4	2	0	-2	-4
2	1	0	-1	-2
0	0	0	0	0
-2	-1	0	1	2
-4	-2	0	2	4

k_5

$\frac{1}{140}$

-4	2	4	2	-4
-2	1	2	1	-2
0	0	0	0	0
2	-1	-2	-1	2
4	-2	-4	-2	4

k_9

$\frac{1}{60}$

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

k_{10}

Analytic computations of second and third directional derivatives

$$f(x,y) = k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 xy + k_6 y^2 + k_7 x^3 + k_8 x^2 y + k_9 xy^2 + k_{10} y^3.$$

$$x = \rho \sin\theta,\ y = \rho \cos\theta$$

- Using polar coordinates

$$f_\theta(\rho) = C_0 + C_1 \rho + C_2 \rho^2 + C_3 \rho^3,$$

where

$$\begin{aligned} C_0 &= k_1, \\ C_1 &= k_2 \sin\theta + k_3 \cos\theta, \\ C_2 &= k_4 \sin^2\theta + k_5 \sin\theta \cos\theta + k_6 \cos^2\theta, \\ C_3 &= k_7 \sin^3\theta + k_8 \sin^2\theta \cos\theta + k_9 \sin\theta \cos^2\theta + k_{10} \cos^3\theta. \end{aligned}$$

Compute analytically second and third directional derivatives

- Gradient angle θ (with positive y-axis at (0,0)):

$$\sin\theta = \frac{k_2}{\sqrt{{k_2}^2 + {k_3}^2}},$$
$$\cos\theta = \frac{k_3}{\sqrt{{k_2}^2 + {k_3}^2}}.$$

Locally approximate surface by a plane and use the normal to the plane to approximate the gradient.

$$f(x,y) = k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 xy + k_6 y^2 + k_7 x^3 + k_8 x^2 y + k_9 xy^2 + k_{10} y^3.$$

Computing directional derivatives (cont'd)

- The derivatives can be computed as follows:

$$f_\theta(\rho) = C_0 + C_1\rho + C_2\rho^2 + C_3\rho^3,$$

$$\begin{aligned} f_\theta'(\rho) &= C_1 + 2C_2\rho + 3C_3\rho^2, \\ f_\theta''(\rho) &= 2C_2 + 6C_3\rho, \\ f_\theta'''(\rho) &= 6C_3. \end{aligned}$$

Second derivative equal to zero implies:

$$f_\theta''(\rho) = 2C_2 + 6C_3\rho = 0, \text{ we get } |\tfrac{C_2}{3C_3}| < \rho_0$$

Third derivative negative implies:

$$f_\theta'''(\rho) < 0, \text{ we get } 6C_3 < 0, \text{ or } C_3 < 0,$$

Edge Detection Using Facet Model (cont'd)

Steps

1. Find $k_1, k_2, k_3, \ldots, k_{10}$ using least square fit, or masks given in Figure 2.8.
2. Compute θ, $\sin\theta$, $\cos\theta$.
3. Compute C_2, C_3.
4. If $C_3 < 0$ and $|\frac{C_2}{3C_3}| < \rho_0$ then that point is an edge point.

Edge Detection Review

- Edge detection operators are based on the idea that edge information in an image is found by looking at the relationship a pixel has with its neighbors
- If a pixel's gray level value is similar to those around it, there is probably not an edge at that point

Edge Detection Review

- Edge detection operators are often implemented with convolution masks and most are based on discrete approximations to differential operators

- Differential operations measure the rate of change in a function, in this case, the image brightness function

Edge Detection Review

- Preprocessing of image is required to eliminate or at least minimize noise effects
- There is tradeoff between sensitivity and accuracy in edge detection
- The parameters that we can set so that edge detector is sensitive include the size of the edge detection mask and the value of the gray level threshold
- A larger mask or a higher gray level threshold will tend to reduce noise effects, but may result in a loss of valid edge points

Summary of Edge Detectors

- 3×3 gradient operators (Prewitt, Sobel) are **simple and fast**. Used when
  - fine edges are only needed
  - noise level is low
- By varying σ parameter, **Canny operator** can be used
  - to detect fine as well as rough edges
  - at different noise levels
- All **gradient operators**
  - provide edge orientation
  - need localisation: non-maxima suppression, hysteresis threshoding
- **Zero-crossing** edge detector
  - is supported by neurophysiological experiments
  - was popular in the 1980's
  - today, **less frequently used** in practice