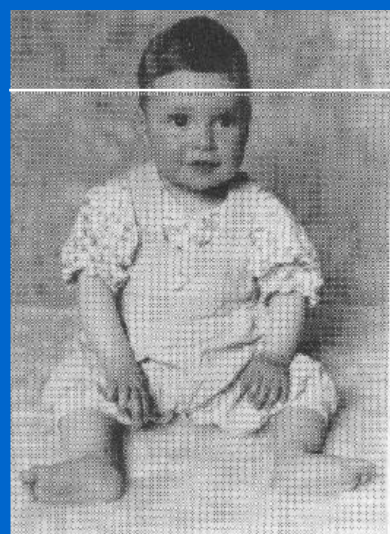
Edge Detection

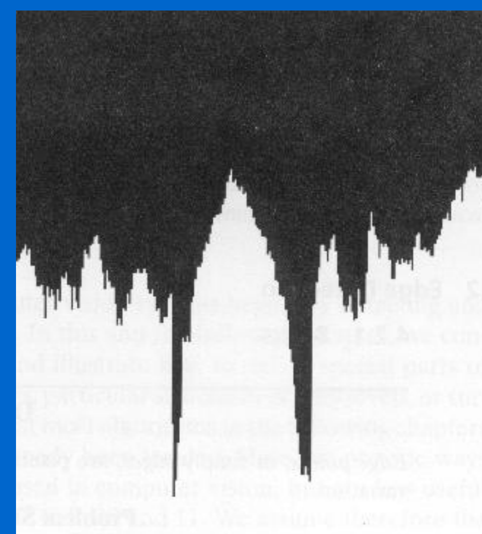
CS485/685 Computer Vision

Dr. George Bebis

Definition of Edges

- Edges are significant local changes of intensity in an image.

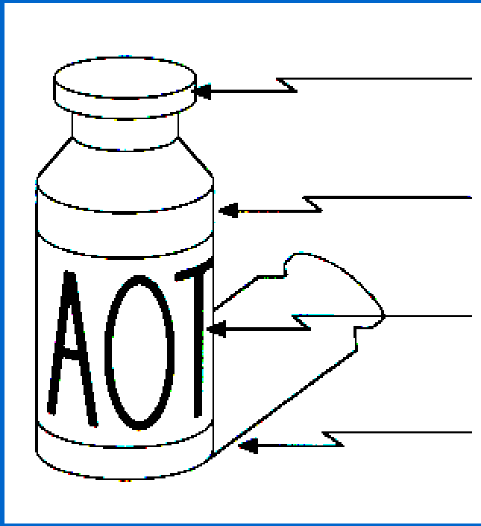




What Causes Intensity Changes?

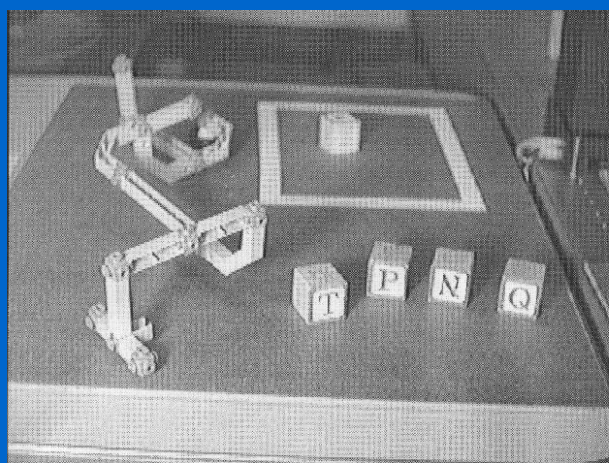
- Geometric events
  - surface orientation (boundary) discontinuities
  - depth discontinuities
  - color and texture discontinuities

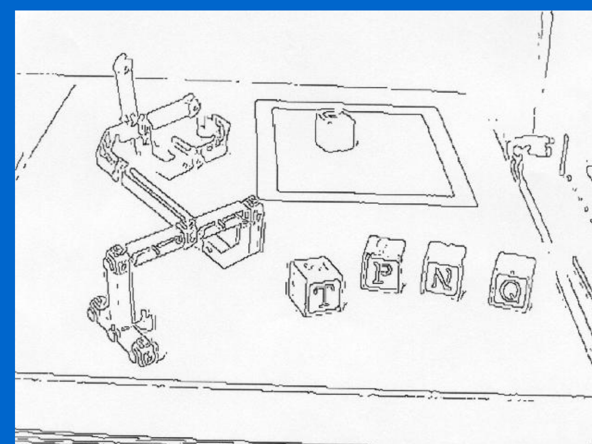
- Non-geometric events
  - illumination changes
  - specularities
  - shadows
  - inter-reflections



Goal of Edge Detection

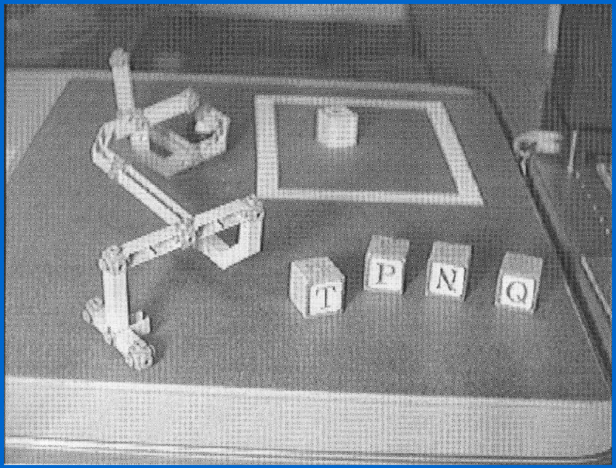
- Produce a line "drawing" of a scene from an image of that scene.

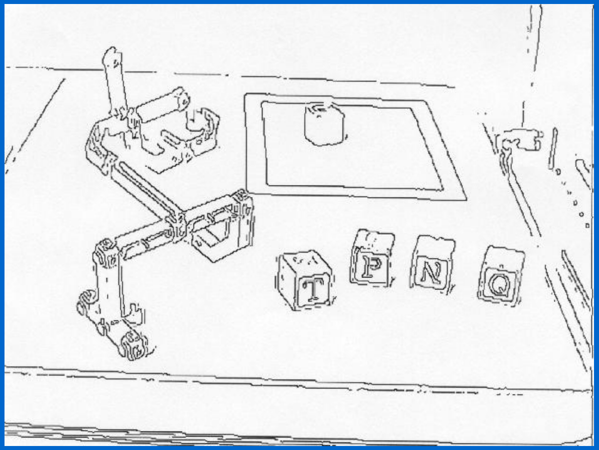




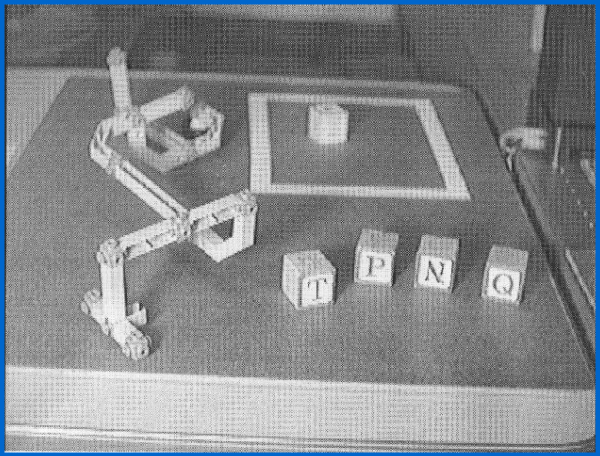
Why is Edge Detection Useful?

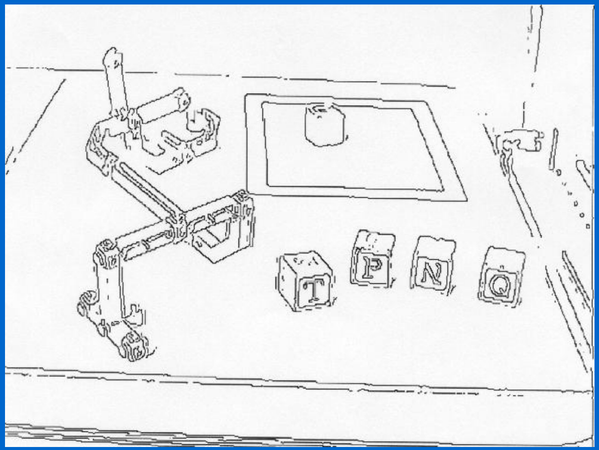
- Important features can be extracted from the edges of an image (e.g., corners, lines, curves).
- These features are used by higher-level computer vision algorithms (e.g., recognition).

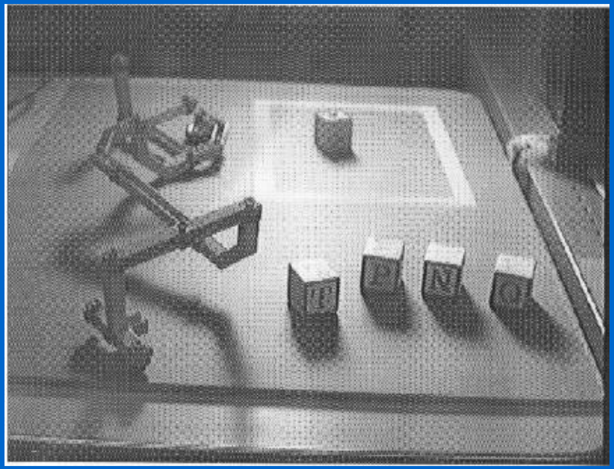


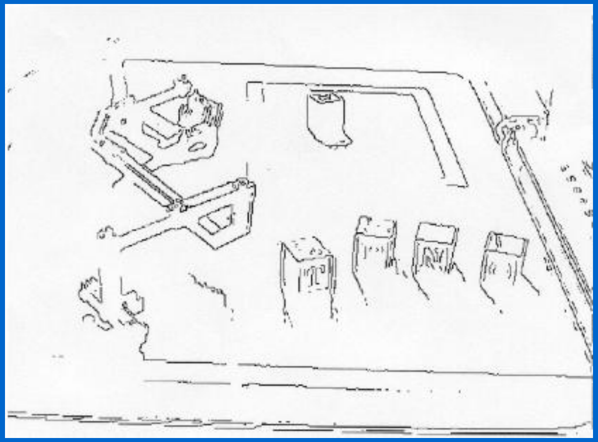


Effect of Illumination



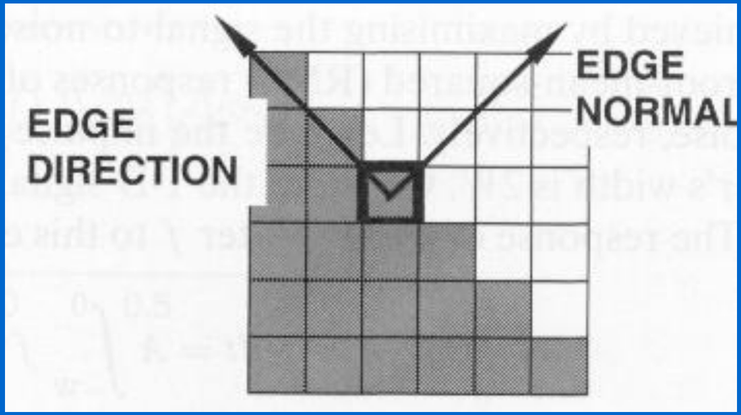






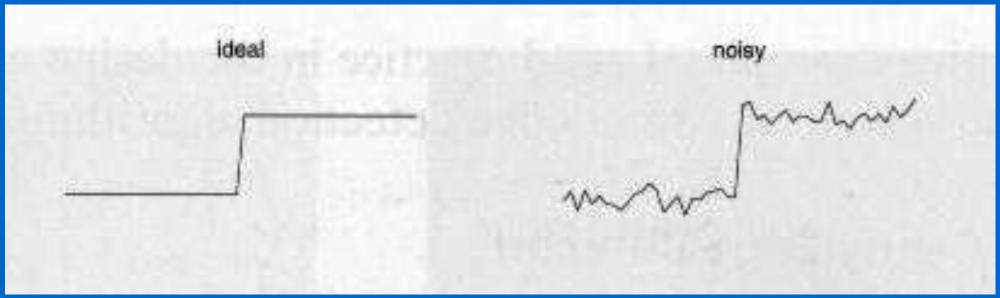
Edge Descriptors

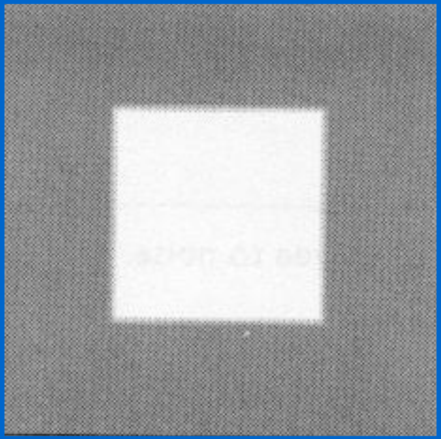
- **Edge direction:** perpendicular to the direction of maximum intensity change (i.e., edge normal)
- **Edge strength:** related to the local image contrast along the normal.
- **Edge position:** the image position at which the edge is located.



Modeling Intensity Changes

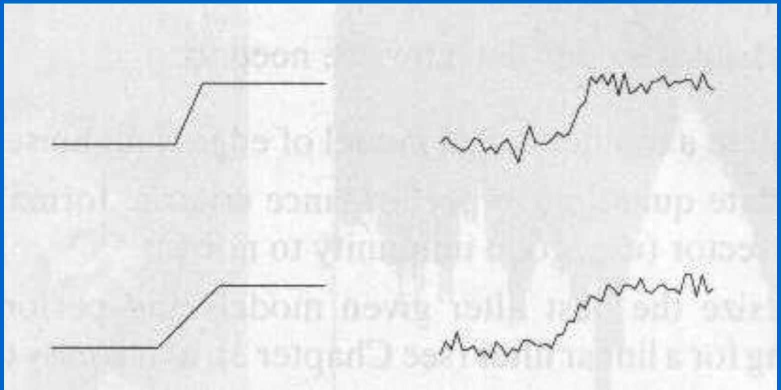
- **Step edge:** the image intensity abruptly changes from one value on one side of the discontinuity to a different value on the opposite side.

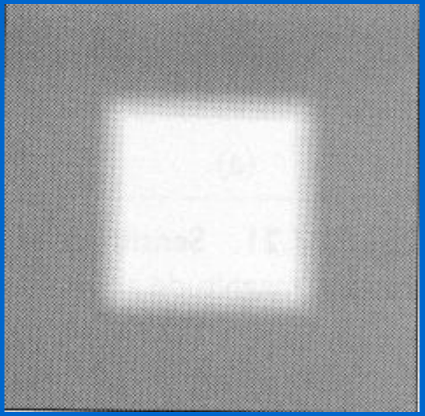




Modeling Intensity Changes (cont'd)

- **Ramp edge:** a step edge where the intensity change is not instantaneous but occur over a finite distance.



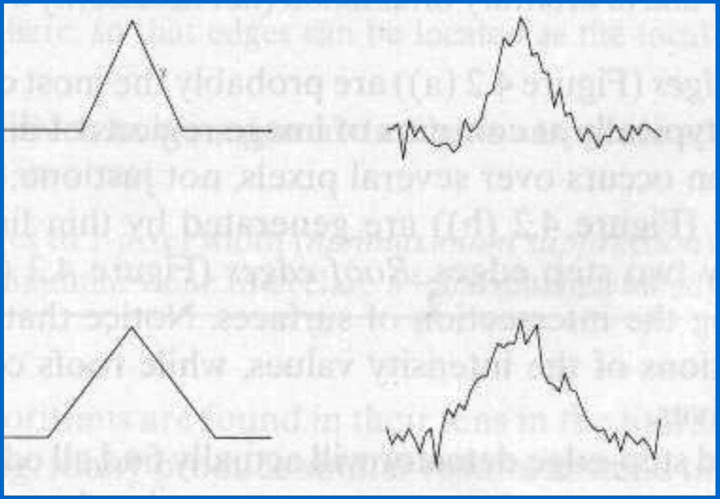


Modeling Intensity Changes (cont'd)

- **Ridge edge:** the image intensity abruptly changes value but then returns to the starting value within some short distance (i.e., usually generated by lines).

Modeling Intensity Changes (cont'd)

- **Roof edge: a** ridge edge where the intensity change is not instantaneous but occur over a finite distance (i.e., usually generated by the intersection of two surfaces).



Main Steps in Edge Detection

(1) Smoothing: suppress as much noise as possible, without destroying true edges.

(2) Enhancement: apply differentiation to enhance the quality of edges (i.e., sharpening).

Main Steps in Edge Detection (cont'd)

(3) Thresholding: determine which edge pixels should be discarded as noise and which should be retained (i.e., threshold edge magnitude).

(4) Localization: determine the exact edge location.

sub-pixel resolution might be required for some applications to estimate the location of an edge to better than the spacing between pixels.

Edge Detection Using Derivatives

- Often, points that lie on an edge are detected by:

  (1) Detecting the local **maxima** or **minima** of the first derivative.

  (2) Detecting the **zero-crossings** of the second derivative.

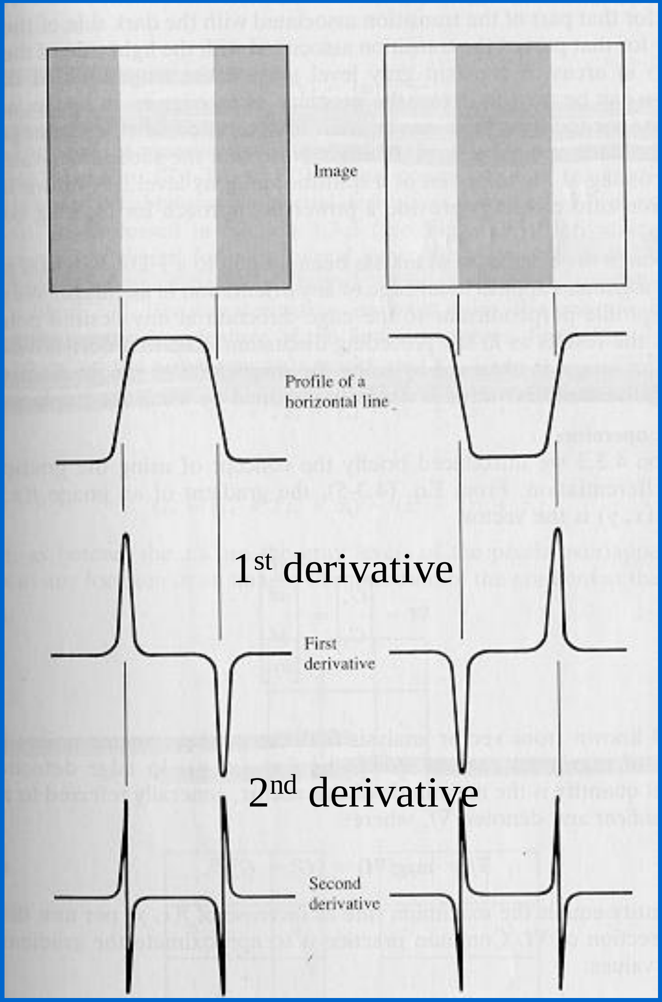


Image Derivatives

- How can we differentiate a *digital* image?
  - **Option 1:** reconstruct a continuous image, *f(x,y)*, then compute the derivative.
  - **Option 2:** take discrete derivative (i.e., finite differences)

Consider this case first!

Edge Detection Using First Derivative

1D functions

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx f(x+1) - f(x) \quad (h=1)$$

(not centered at x)

mask: $[-1 \quad 1]$

$$\text{mask } \mathbf{M} = [-1, 0, 1]$$

(centered at x)

(upward) step edge

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	0	0	0	0	12	12	0	0	0	0

(downward) step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	$\otimes$	M	0	0	0	0	-12	-12	0	0	0	0

ramp edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	$\otimes$	M	0	0	0	3	6	6	6	3	0	0

roof edge

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	0	0	0	12	0	-12	0	0	0	0

Edge Detection Using Second Derivative

- Approximate finding maxima/minima of gradient magnitude by finding places where:

$$\frac{df^2}{dx^2}(x) = 0$$

- Can't always find discrete pixels where the second derivative is zero – look for zero-crossing instead.

Edge Detection Using Second Derivative (cont'd)

1D functions:

$$f''(x) = \lim_{h \to 0} \frac{f'(x+h) - f'(x)}{h} \approx f'(x+1) - f'(x) =$$

$$f(x+2) - 2f(x+1) + f(x) \quad (h=1)$$

(centered at x+1)

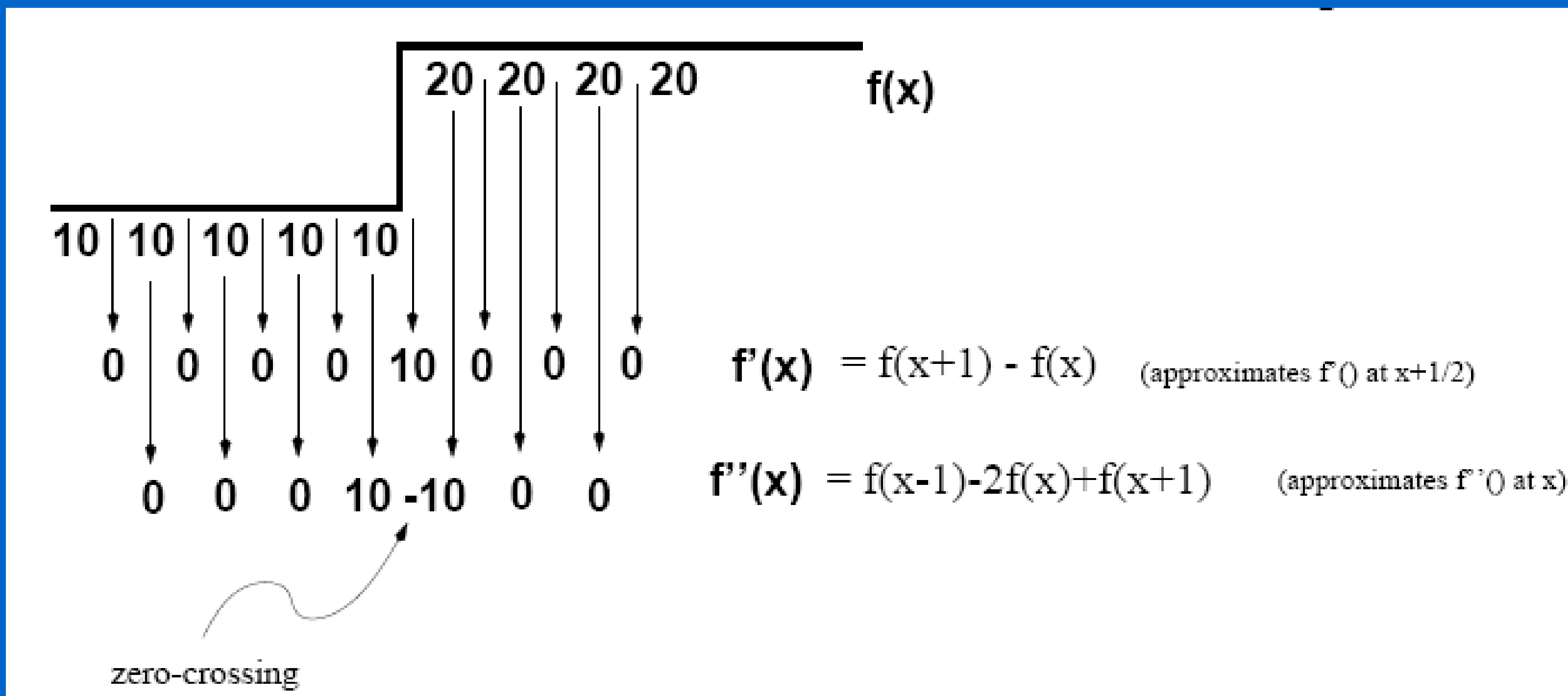
Replace x+1 with x (i.e., centered at x):

$$f''(x) \approx f(x+1) - 2f(x) + f(x-1)$$

$$\Downarrow$$

mask: $[1 \quad -2 \quad 1]$

Edge Detection Using Second Derivative (cont’d)



Edge Detection Using <u>Second</u> Derivative (cont'd)

(upward) step edge

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	0	0	0	0	-12	12	0	0	0	0

(downward) step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	$\otimes$	M	0	0	0	0	12	-12	0	0	0	0

ramp edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	$\otimes$	M	0	0	0	-3	0	0	0	3	0	0

roof edge

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	0	0	0	-12	24	-12	0	0	0	0

Edge Detection Using Second Derivative (cont'd)

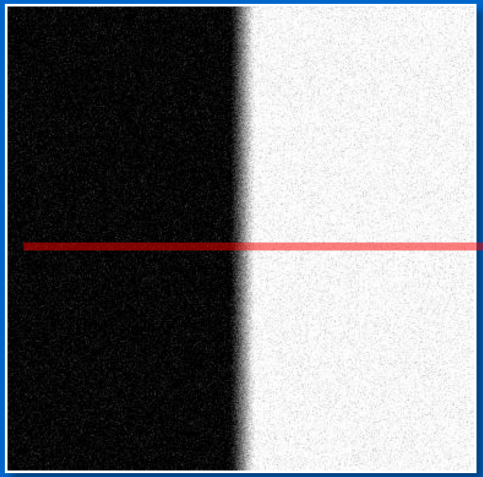
- Four cases of zero-crossings:

$$\{+,-\}, \{+,0,-\},\{-,+\}, \{-,0,+\}$$

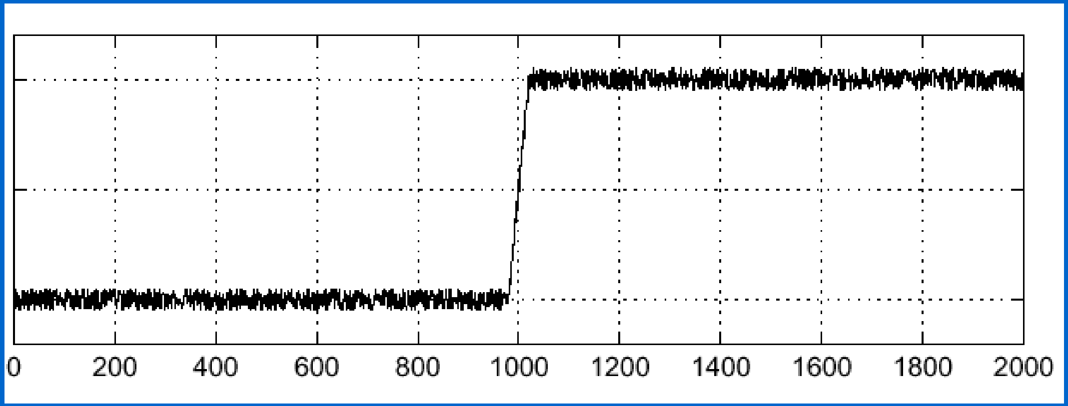
- **Slope** of zero-crossing $\{a, -b\}$ is: $|a+b|$.

- To detect "strong" zero-crossing, threshold the slope.

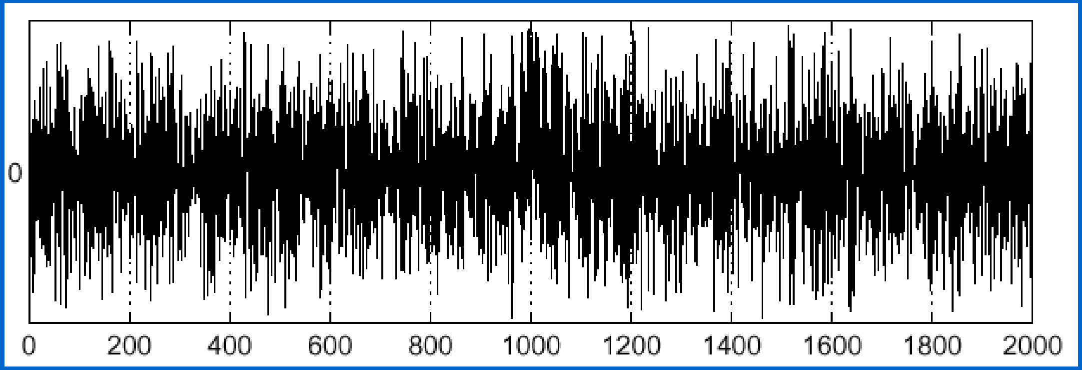
Effect Smoothing on Derivates



$f(x)$



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



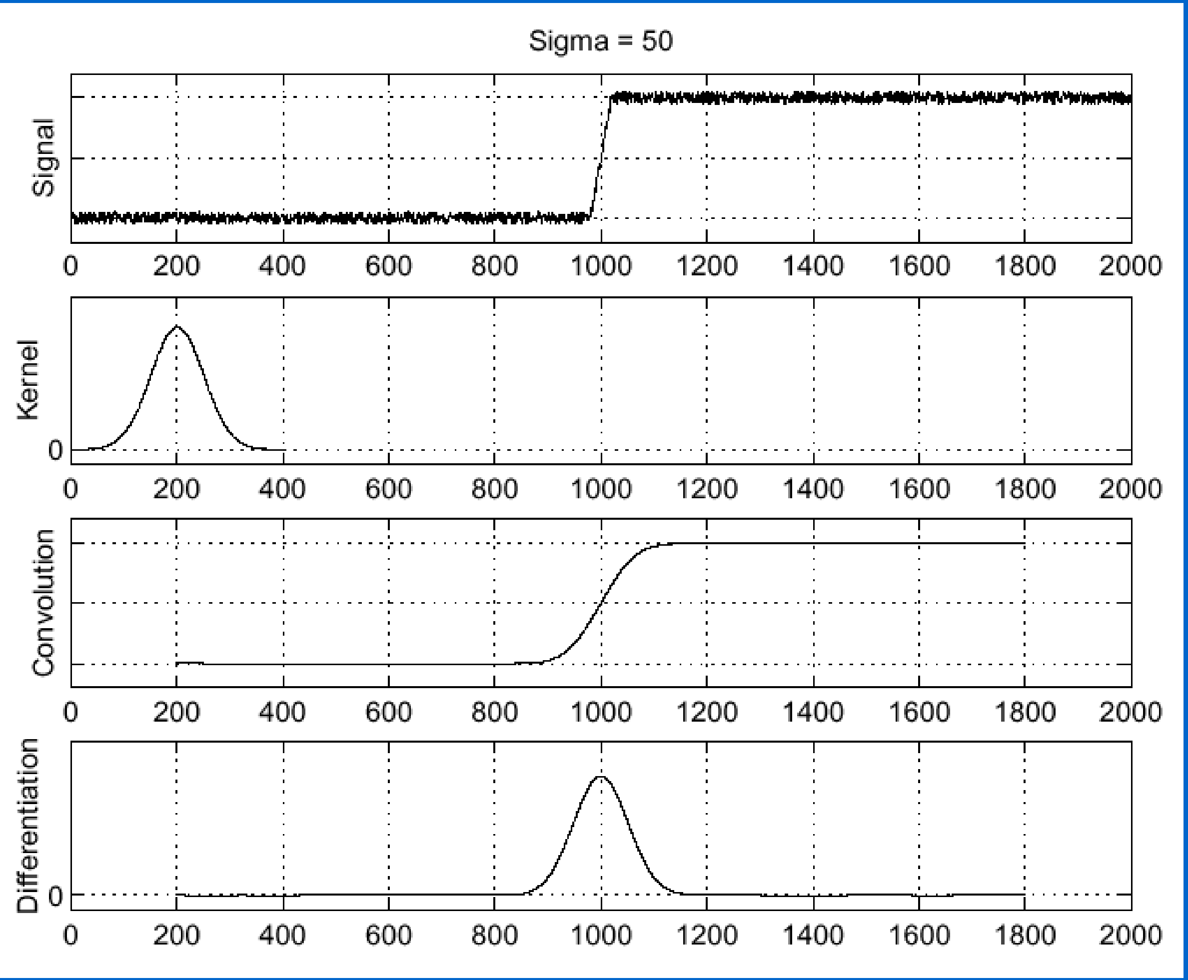
Effect of Smoothing on Derivatives (cont'd)

f

h

$h \star f$

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



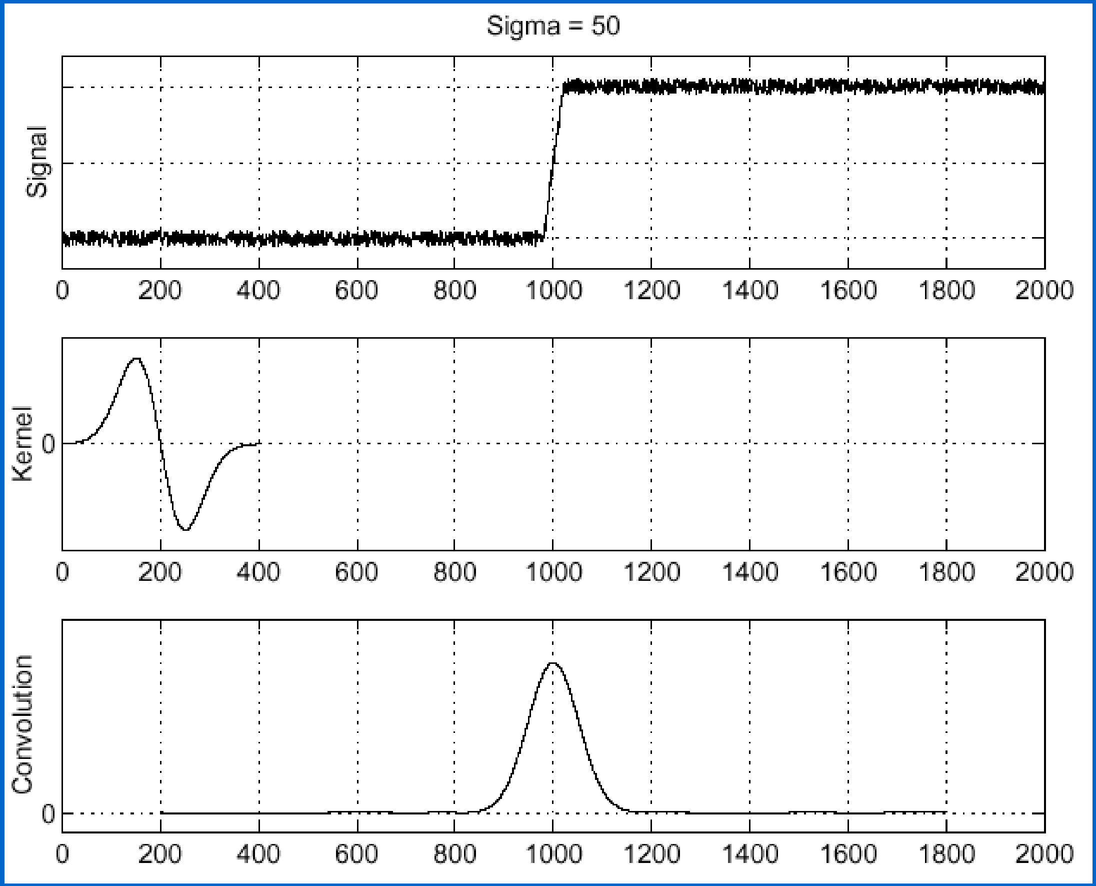
Combine Smoothing with Differentiation

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$ (i.e., saves one operation)

f

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

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



Mathematical Interpretation of combining smoothing with differentiation

- Numerical differentiation is an ill-posed problem.
  - i.e., solution does not exist or it is not unique or it does not depend continuously on initial data)
- Ill-posed problems can be solved using "regularization"
  - i.e., impose additional constraints
- Smoothing performs image interpolation.

Edge Detection Using First Derivative (Gradient)

2D functions:

- The first derivate of an image can be computed using the gradient:

$$\nabla f$$

$$grad(f) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

Gradient Representation

- The gradient is a vector which has magnitude and direction:

$$magnitude(grad(f)) = \sqrt{\frac{\partial f}{\partial x}^2 + \frac{\partial f}{\partial y}^2}$$

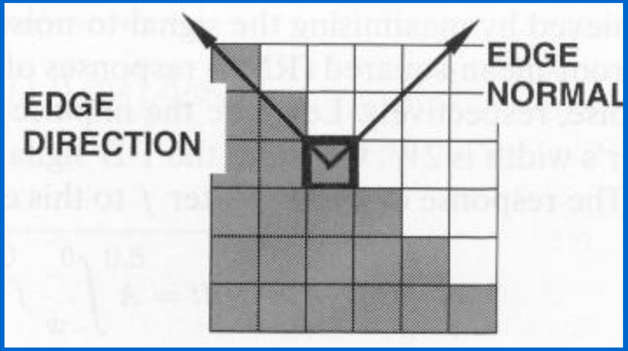
$$direction(grad(f)) = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

or

(approximation)

$$\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$$

- **Magnitude:** indicates edge strength.
- **Direction:** indicates edge direction.
  - i.e., perpendicular to edge direction



Approximate Gradient

- Approximate gradient using finite differences:

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

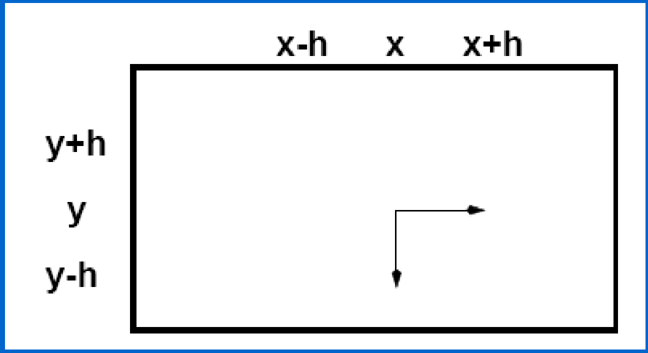
$$\frac{\partial f}{\partial y} = \lim_{h->0} \frac{f(x,y+h)-f(x,y)}{h}$$

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

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

Approximate Gradient (cont'd)

- Cartesian vs pixel-coordinates:
  - j corresponds to x direction
  - i to $-y$ direction



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

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

Approximate Gradient (cont'd)

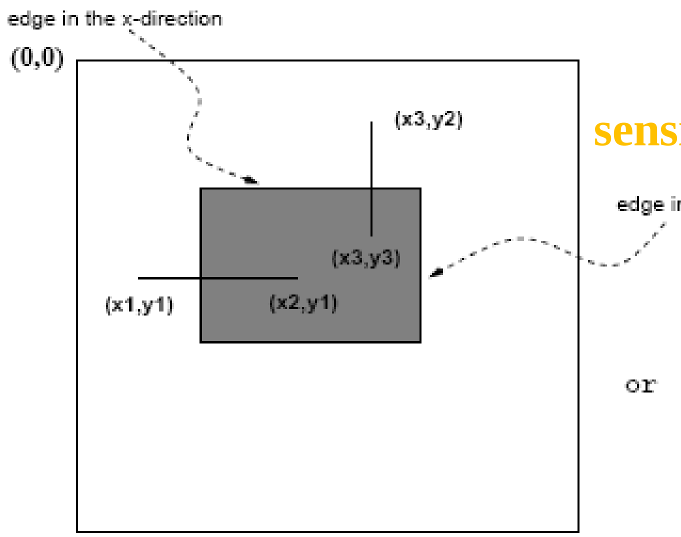
sensitive to horizontal edges!

$$\frac{\partial f}{\partial y}$$

$$\frac{f(x3,y2) - f(x3,y3)}{y2 - y3}$$

$$\text{or} \quad \frac{f(x,y+Dy) - f(x,y)}{-Dy} = \boxed{f(x,y) - f(x,y+1)} \quad \textbf{(grad in the y-direction)}$$

$(y3=y2+Dy,\ y2=y,\ x3=x,\ Dy=1)$



sensitive to vertical edges!

$$\frac{\partial f}{\partial y}$$

$$\frac{f(x2,y1) - f(x1,y1)}{x2 - x1}$$

$$\text{or} \quad \frac{f(x+Dx,y) - f(x,y)}{Dx} =$$

$$\boxed{f(x+1,y) - f(x,y)}$$

(grad in the x-direction)

$(x2=x+Dx,\ x1=x,\ y1=y2=y,\ Dx=1)$

Approximating Gradient (cont'd)

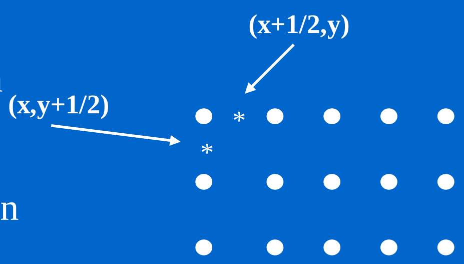
- We can implement $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ using the following masks:

-1	1

$\frac{\partial f}{\partial x}$ good approximation at (x+1/2,y)

1
-1

$\frac{\partial f}{\partial y}$ good approximation at (x,y+1/2)



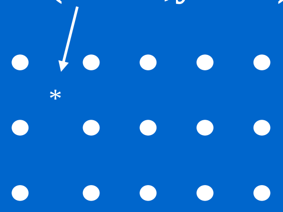
Approximating Gradient (cont'd)

- A different approximation of the gradient:

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

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

good approximation
(x+1/2,y+1/2)



- $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ can be implemented using the following masks:

1	0
0	−1

0	1
−1	0

(b) Roberts

Another Approximation

- 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)

Edge Detection Steps Using Gradient

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

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

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

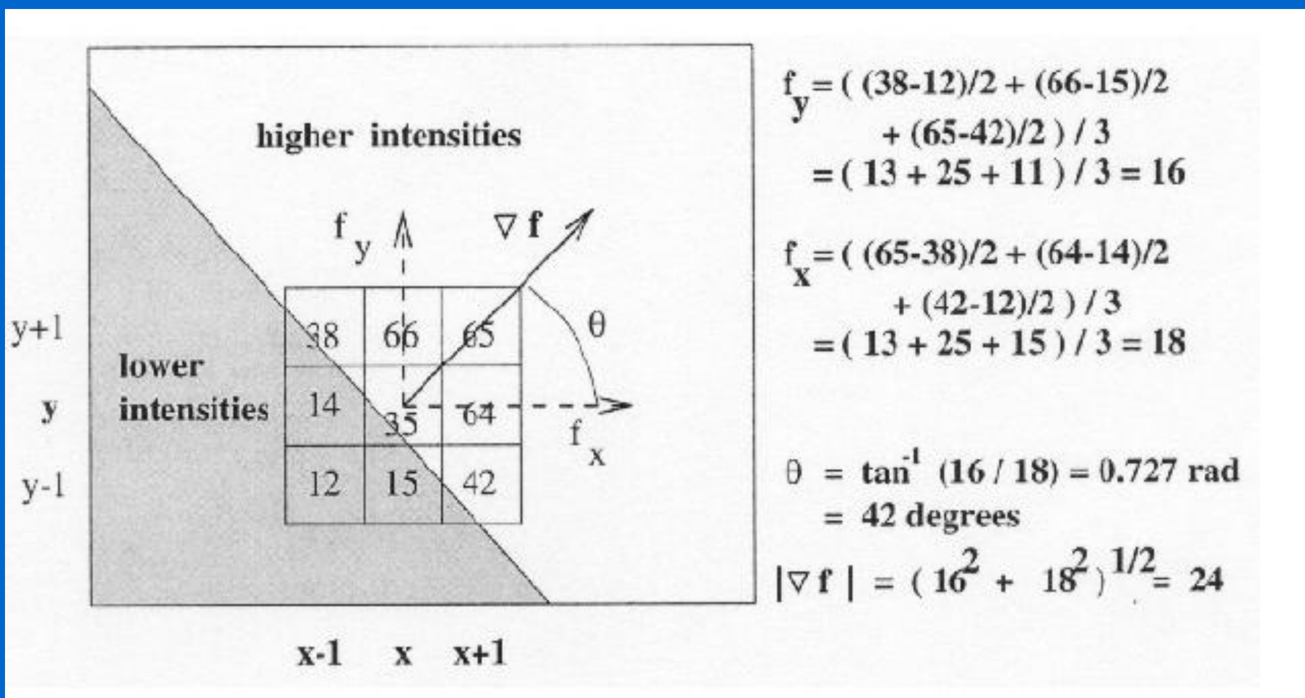
(4) $magn(x, y) = |\hat{f}_x| + |\hat{f}_y|$ (i.e., sqrt is costly!)

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

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

Example (using Prewitt operator)

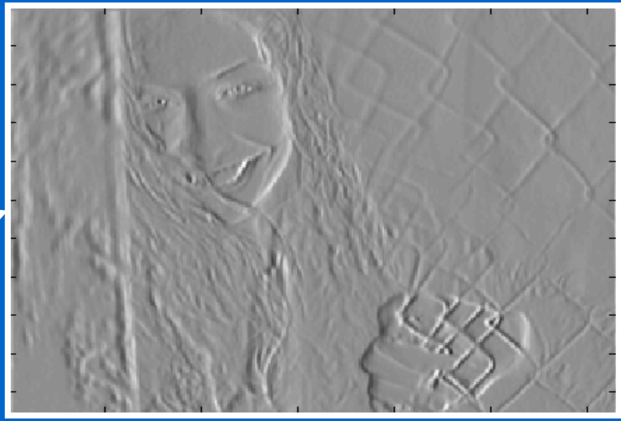
$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$



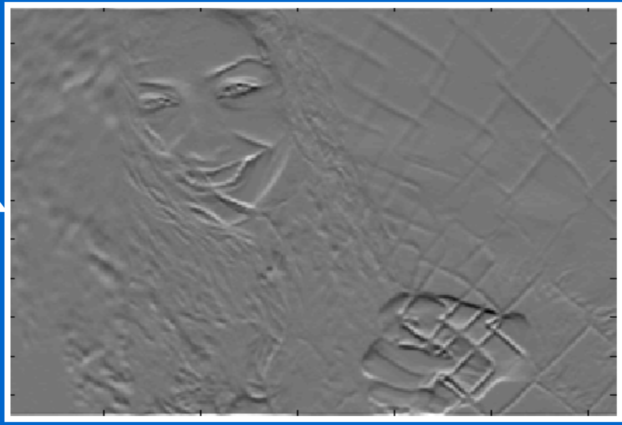
<u>Note</u>: in this example, the divisions by 2 and 3 in the computation of f_x and f_y are done for normalization purposes only

Another Example

$$\frac{d}{dx}I$$

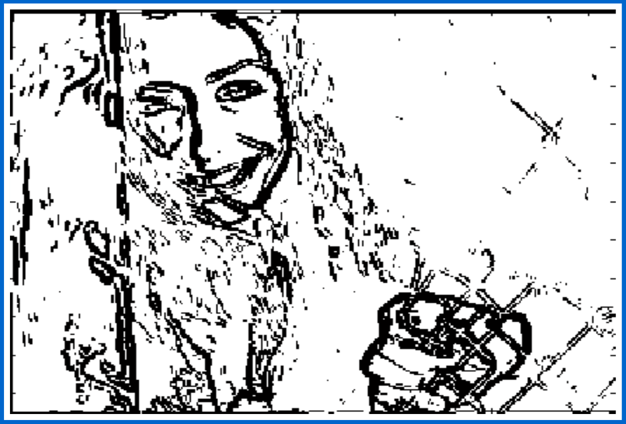


$$\frac{d}{dy}I$$



Another Example (cont'd)

$$\nabla = \sqrt{\left(\frac{d}{dx}I\right)^2 + \left(\frac{d}{dy}I\right)^2}$$



$$\nabla \geq Threshold = 100$$

Isotropic property of gradient magnitude

- The magnitude of the gradient detects edges in all directions.

$$\frac{d}{dx} I$$

$$\frac{d}{dy} I$$

$$\nabla = \sqrt{\left(\frac{d}{dx} I\right)^2 + \left(\frac{d}{dy} I\right)^2}$$

Practical Issues

- 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

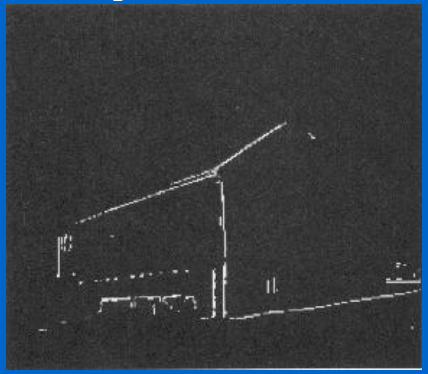
Practical Issues (cont'd)

- Choice of threshold.

gradient magnitude

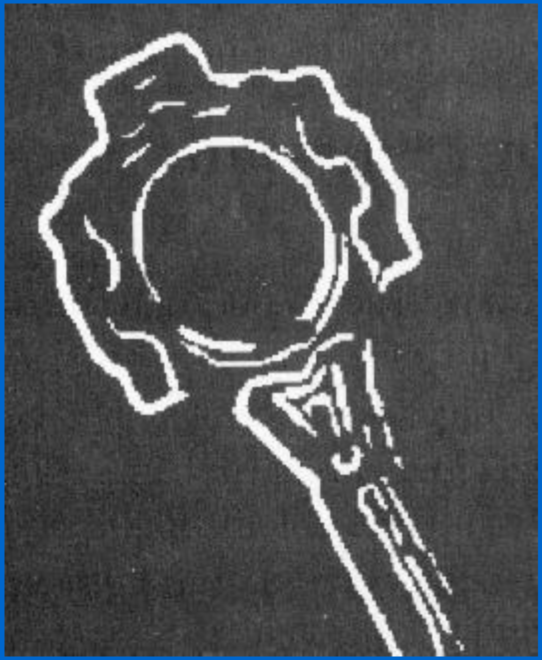
low threshold

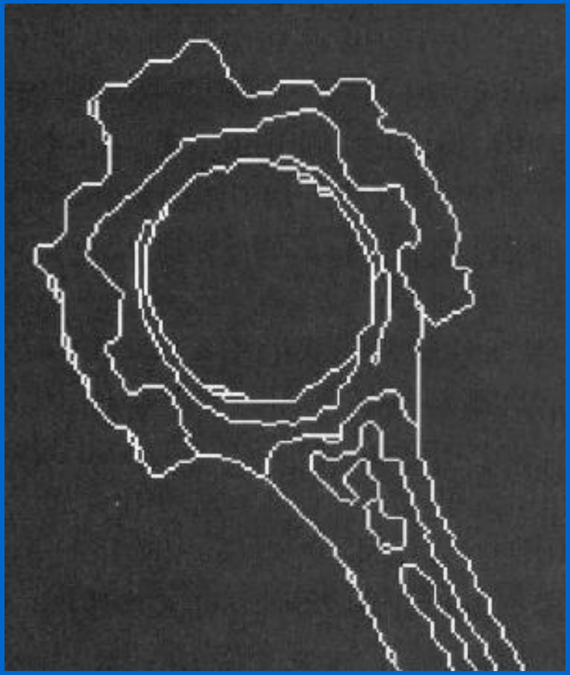
high threshold



Practical Issues (cont'd)

- Edge thinning and linking.





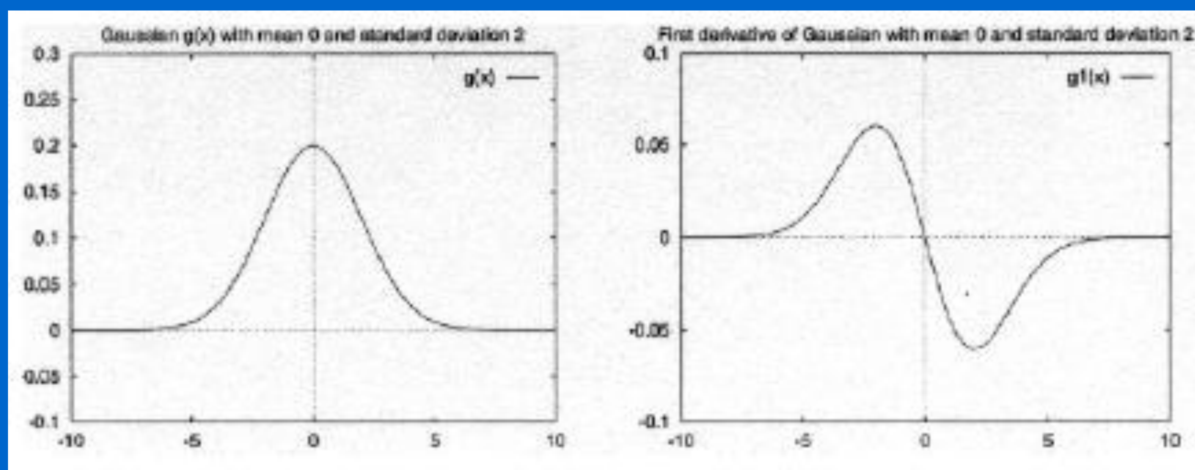
Criteria for Optimal Edge Detection

- **(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.

Canny edge detector

- 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.

  (i.e., analysis based on "step-edges" corrupted by "Gaussian noise“)



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

Steps of 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: $G_x(x, y) = \dfrac{-x}{\sigma^2} G(x, y)$

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

Steps of Canny edge detector (cont'd)

2. Compute the gradient magnitude (and direction)

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

3. Apply non-maxima suppression.

4. Apply hysteresis thresholding/edge linking.

Canny edge detector - example

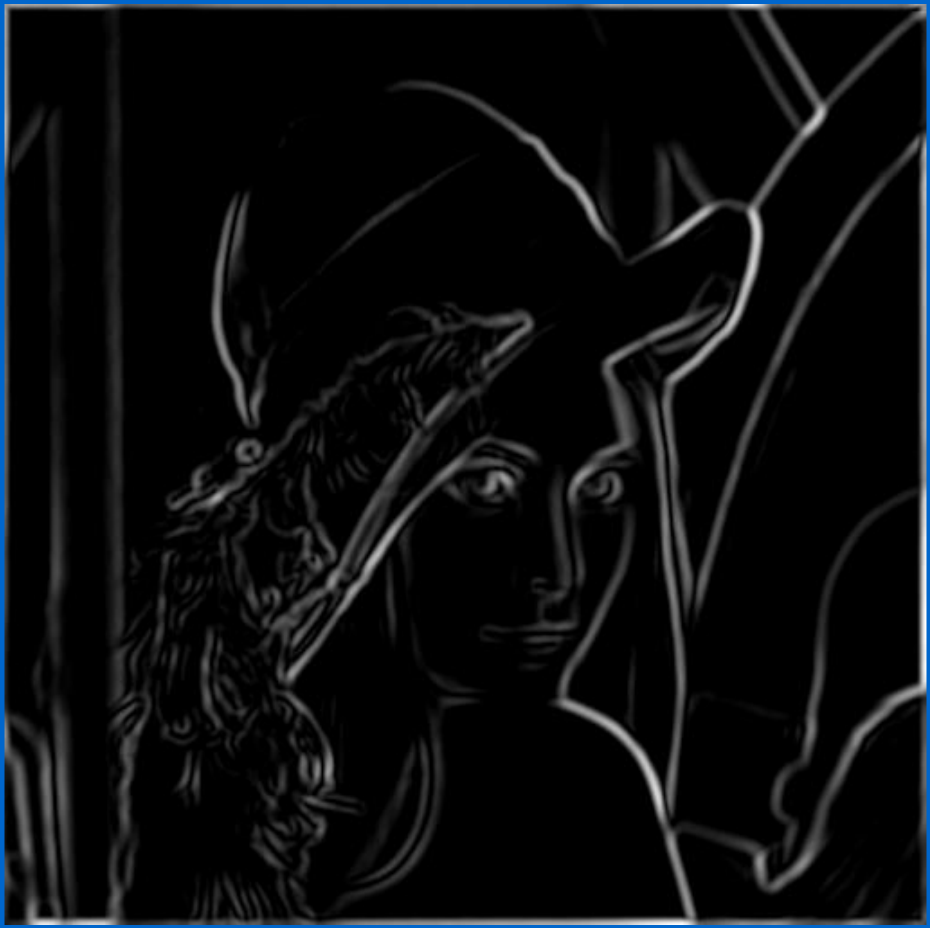
original image

Canny edge detector – example (cont'd)

Gradient magnitude

Canny edge detector – example (cont'd)

Thresholded gradient magnitude

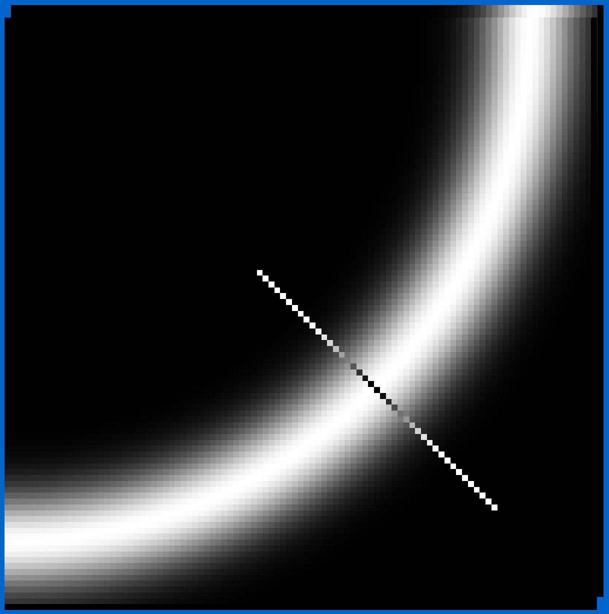


Canny edge detector – example (cont'd)

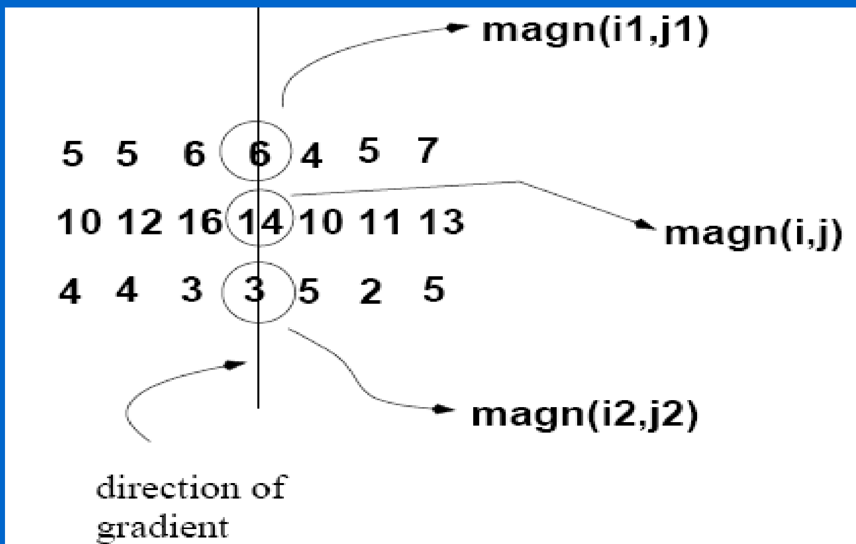
Thinning (non-maxima suppression)

Non-maxima suppression

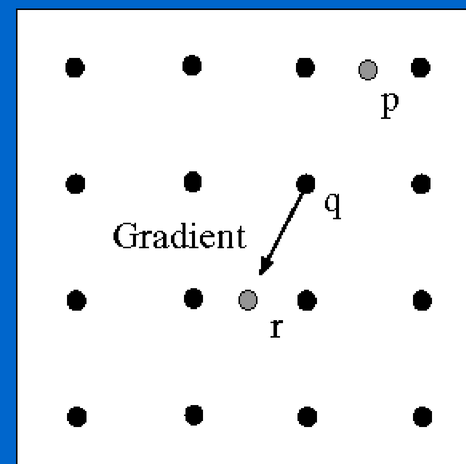
- Check if gradient magnitude at pixel location (i,j) is local maximum along gradient direction



Non-maxima suppression (cont'd)



Warning: requires checking interpolated pixels p and r



Algorithm

For each pixel (i,j) 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 $I_N(i, j) = magn(i, j)$

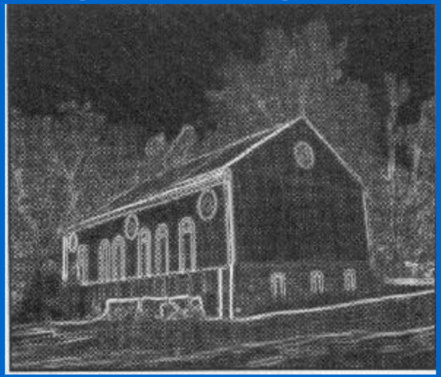
Hysteresis thresholding

- Standard thresholding:

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

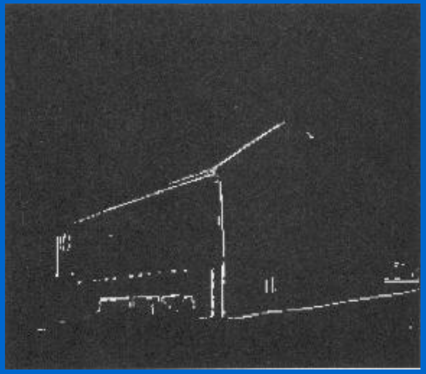
  - Can only select “strong” edges.
  - Does not guarantee “continuity”.

gradient magnitude



low threshold

high threshold



Hysteresis thresholding (cont'd)

- Hysteresis thresholding uses two thresholds:

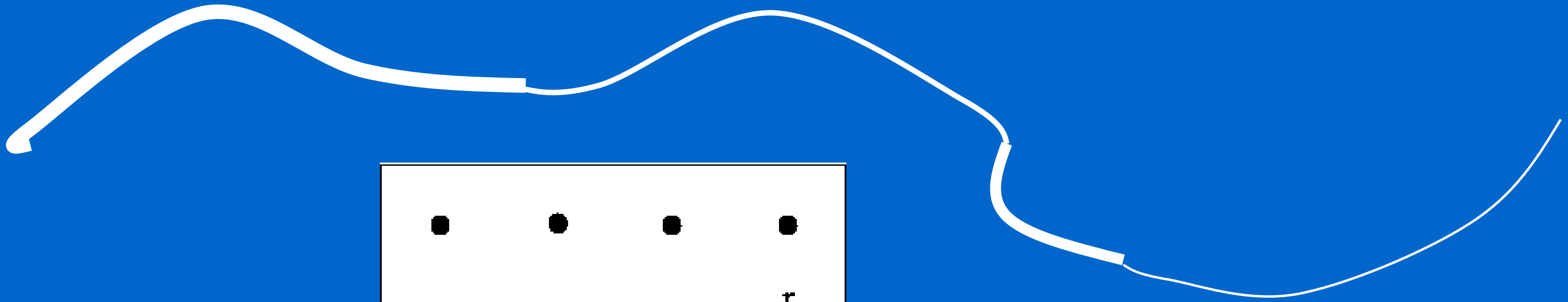
  - low threshold t_l
  - high threshold t_h (usually, $t_h = 2t_l$)

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

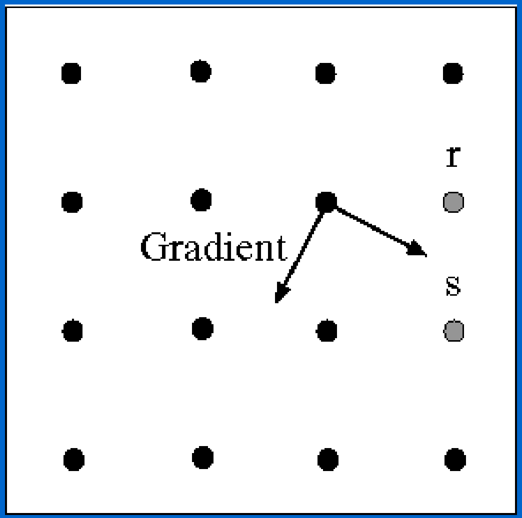
- For "maybe" edges, decide on the edge if <u>neighboring</u> pixel is a strong edge.

Hysteresis thresholding/Edge Linking

Idea: use a **high** threshold to start edge curves and a **low** threshold to continue them.



Use edge “direction” for linking edges



Hysteresis Thresholding/Edge Linking (cont'd)

Algorithm

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

(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 $I_2(i, j)$.

Note: large gaps are still difficult to bridge.
(i.e., more sophisticated algorithms are required)

Second Derivative in 2D: Laplacian

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

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

Second Derivative in 2D: Laplacian (cont'd)

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

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

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

0	0	0
1	-2	1
0	0	0

+

0	1	0
0	-2	0
0	1	0

=

0	1	0
1	-4	1
0	1	0

Variations of Laplacian

$$\begin{array}{|c|c|c|}\hline 0.5 & 0.0 & 0.5 \\ \hline 1.0 & -4.0 & 1.0 \\ \hline 0.5 & 0.0 & 0.5 \\ \hline\end{array} + \begin{array}{|c|c|c|}\hline 0.5 & 1.0 & 0.5 \\ \hline 0.0 & -4.0 & 0.0 \\ \hline 0.5 & 1.0 & 0.5 \\ \hline\end{array} = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & -8 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

$$\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline 1 & -2 & 1 \\ \hline 1 & -2 & 1 \\ \hline\end{array} + \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline -2 & -2 & -2 \\ \hline 1 & 1 & 1 \\ \hline\end{array} = \begin{array}{|c|c|c|}\hline 2 & -1 & 2 \\ \hline -1 & -4 & -1 \\ \hline 2 & -1 & 2 \\ \hline\end{array}$$

Laplacian - 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

detect zero-crossings

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

Properties of Laplacian

- It is an isotropic operator.
- It is cheaper to implement than the gradient (i.e., one mask only).
- It does not provide information about edge direction.
- It is more sensitive to noise (i.e., differentiates twice).

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.

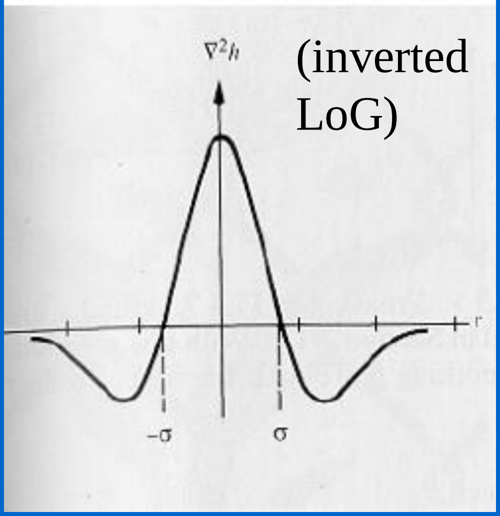
$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

σ controls smoothing

- It can be shown that:

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

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



Laplacian of Gaussian (LoG) - Example

(inverted LoG)

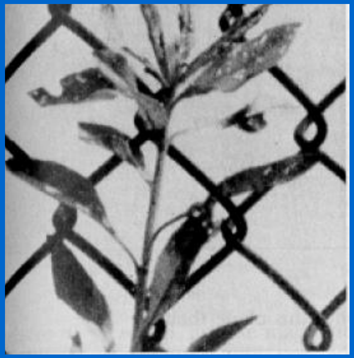
5 × 5 Laplacian of Gaussian mask

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

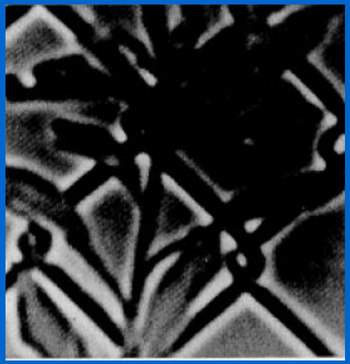
(inverted LoG)

17 × 17 Laplacian of Gaussian mask

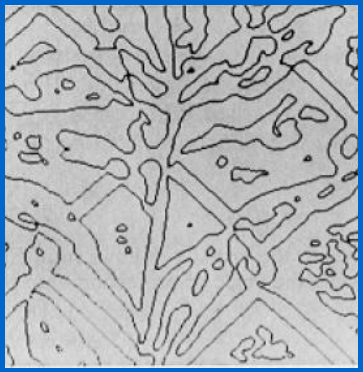
0	0	0	0	0	0	−1	−1	−1	−1	−1	0	0	0	0	0	0
0	0	0	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	0	0	0
0	0	−1	−1	−1	−2	−3	−3	−3	−3	−3	−2	−1	−1	−1	0	0
0	0	−1	−1	−2	−3	−3	−3	−3	−3	−3	−3	−2	−1	−1	0	0
0	−1	−1	−2	−3	−3	−3	−2	−3	−2	−3	−3	−3	−2	−1	−1	0
0	−1	−2	−3	−3	−3	0	2	4	2	0	−3	−3	−3	−2	−1	0
−1	−1	−3	−3	−3	0	4	10	12	10	4	0	−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

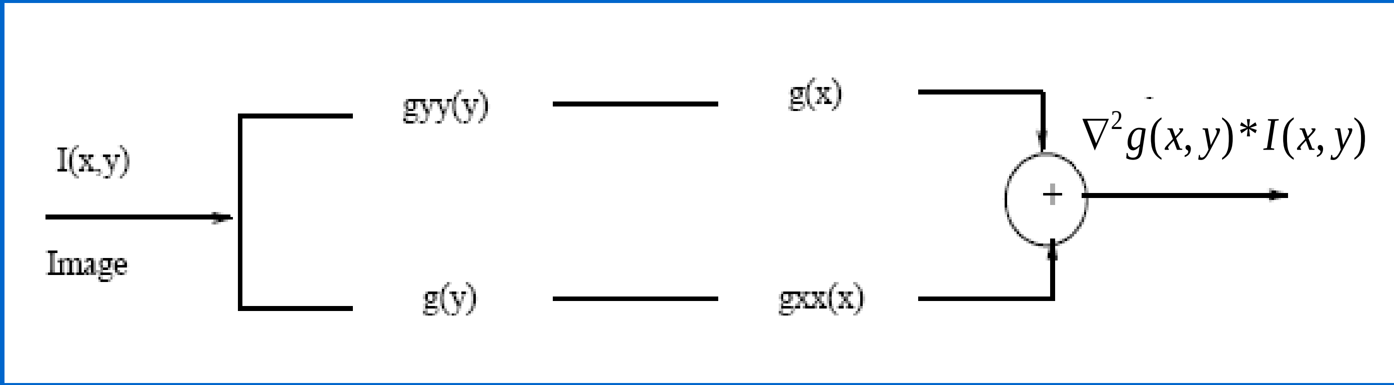


Decomposition of LoG

- It can be shown than LoG can be written as follows:

$$\nabla^2 g(x,y) = \frac{\partial}{\partial y^2} g(y) * g(x) + g(y) * \frac{\partial}{\partial x^2} g(x).$$

- 2D LoG convolution can be implemented using 4, 1D convolutions.



Decomposition of LoG (cont'd)

Steps

1. Convolve the image with a second derivative of Gaussian mask ($g_{yy}(y)$) along each column.

2. Convolve the resultant image from step (1) by a Gaussian mask ($g(x)$) along each row. Call the resultant image I^x.

3. Convolve the original image with a Gaussian mask ($g(y)$) along each column.

4. Convolve the resultant image from step (3) by a second derivative of Gaussian mask ($g_{xx}(x)$) along each row. Call the resultant image I^y.

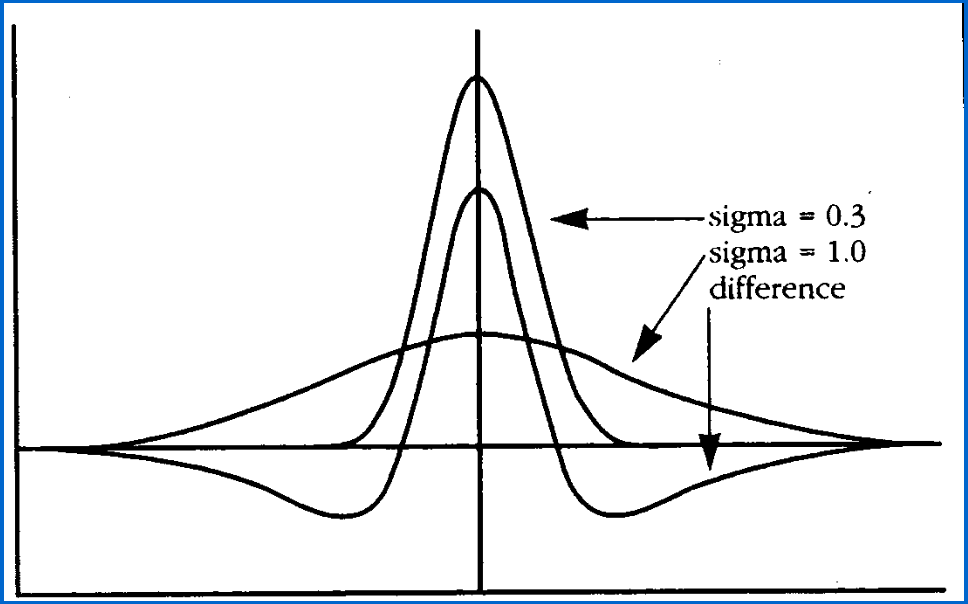
5. Add I^x and I^y.

Difference of Gaussians (DoG)

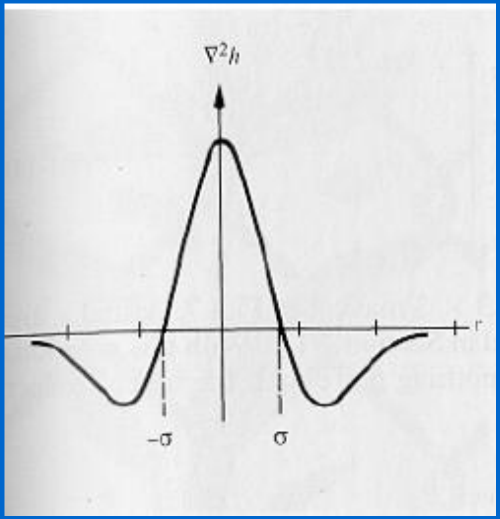
- The Laplacian of Gaussian can be approximated by the difference between two Gaussian functions:

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

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 $\sqrt{2}$.)

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

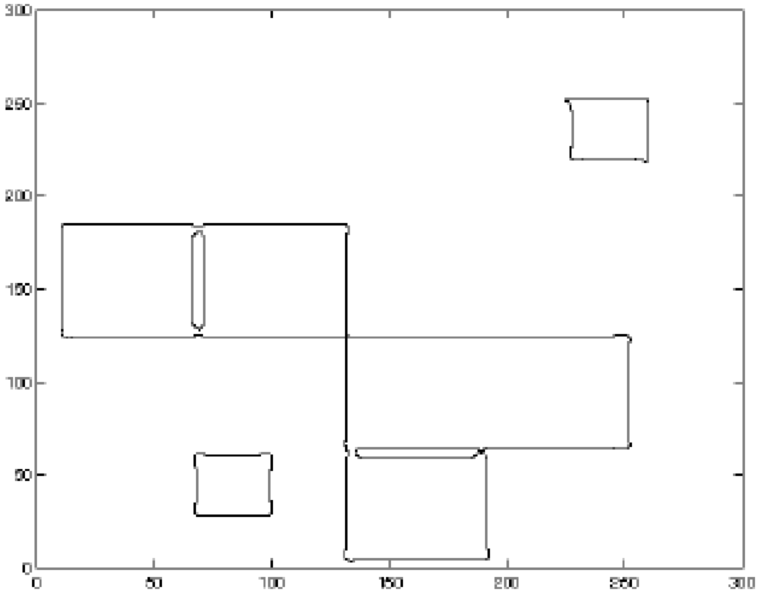
ramp edge

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

Gradient vs LoG (cont'd)

LoG behaves poorly at corners



Directional Derivative

$$\nabla f \quad grad(f) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

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

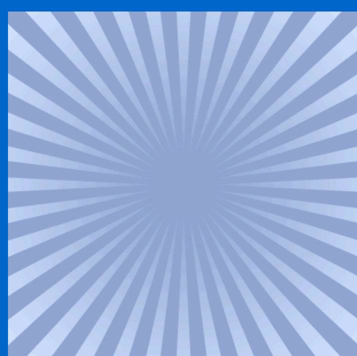
- The partial derivatives of f(x,y) will give the slope $\partial f/\partial x$ in the positive x direction and the slope $\partial f/\partial y$ in the positive y direction.

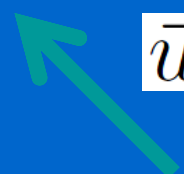
- We can generalize the partial derivatives to calculate the slope in any direction (i.e., *directional derivative*).

Directional Derivative (cont'd)

- Directional derivative computes intensity changes in a specified direction.

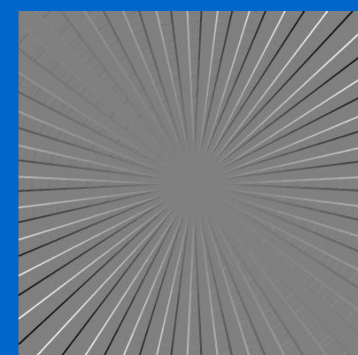
f





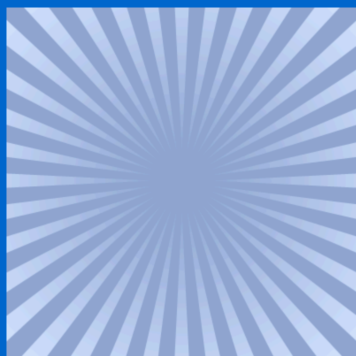
$\vec{u}$

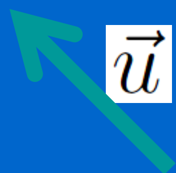
Compute
derivative
in direction u



Directional Derivative (cont'd)

f



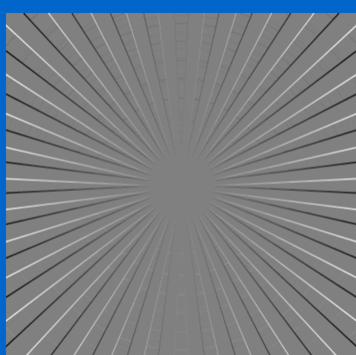


$\vec{u}$

Directional derivative is a linear combination of partial derivatives.

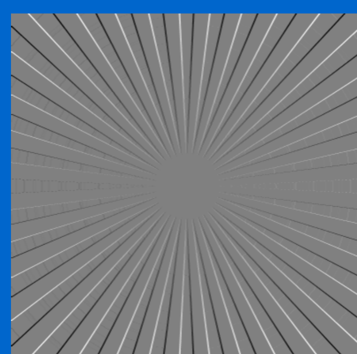
(From vector calculus)

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



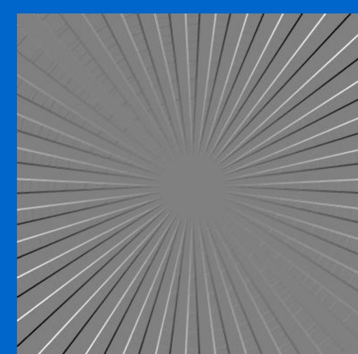
$\frac{\partial f}{\partial x} \cdot u_x$

+



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

=



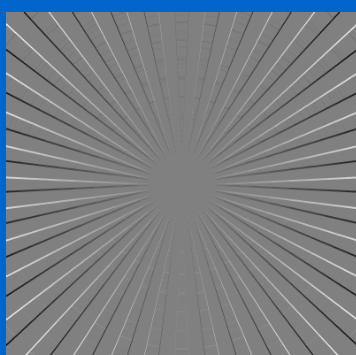
$\nabla_{\vec{u}} f$

Directional Derivative (cont'd)

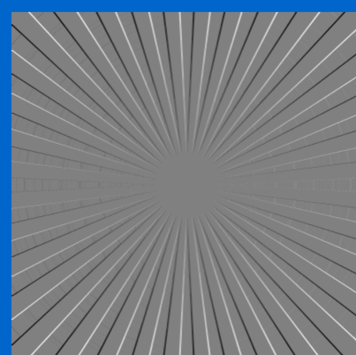
$$\cos\theta = \frac{u_x}{u},\ \sin\theta = \frac{u_y}{u}$$

$\|u\|=1$ ⇒

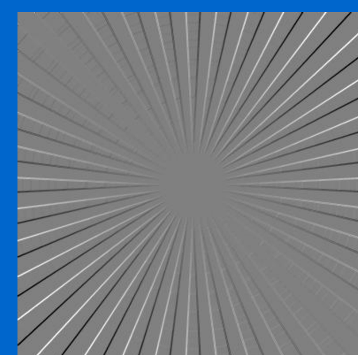
$$u_x = \cos\theta,\ u_y = \sin\theta$$



+



=



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

Higher Order Directional Derivatives

$$f_\theta^{'}(x,y) = \frac{\partial f}{\partial x}\cos\theta + \frac{\partial f}{\partial y}\sin\theta$$

$$f_\theta^{''}(x,y) = \frac{\partial^2 f}{\partial x^2}\cos^2\theta + 2\frac{\partial^2 f}{\partial x \partial y}\cos\theta\sin\theta + \frac{\partial^2 f}{\partial y^2}\sin^2\theta$$

$$f_\theta^{'''}(x,y) = \frac{\partial^3 f}{\partial x^3}\cos^3\theta + 3\frac{\partial^3 f}{\partial x^2 \partial y}\cos^2\theta\sin\theta + 3\frac{\partial^3 f}{\partial x \partial y^2}\cos\theta\sin^2\theta + \frac{\partial^3 f}{\partial y^3}\sin^3\theta$$

Edge Detection Using Directional Derivative

- What direction would you use for edge detection?

Direction of gradient:

$$\theta = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

Second Directional Derivative
(along gradient direction)

$$f_{\theta}''(x,y) = \frac{\partial^2 f}{\partial x^2}\cos^2\theta + 2\frac{\partial^2 f}{\partial x \partial y}\cos\theta \sin\theta + \frac{\partial^2 f}{\partial y^2}\sin^2\theta$$

$$\theta = \begin{pmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

$$\frac{\partial^2 f}{\partial n^2} \equiv \frac{f_x{}^2 f_{xx} + 2 f_x f_y f_{xy} + f_y{}^2 f_{yy}}{f_x{}^2 + f_y{}^2}$$

Edge Detection Using Second Derivative

Laplacian: $$\nabla^2 f(x,y) \equiv \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

or $$\nabla^2 f \equiv f_{xx} + f_{yy}$$

Second directional derivative along the gradient:

$$\frac{\partial^2 f}{\partial n^2} \equiv \frac{{f_x}^2 f_{xx} + 2 f_x f_y f_{xy} + {f_y}^2 f_{yy}}{{f_x}^2 + {f_y}^2}$$

(i) the second directional derivative is equal to zero and
(ii) the third directional derivative is negative.

Properties of Second Directional Derivative (along gradient direction)

Mathematical:

1. $\frac{\partial^2}{\partial n^2}$ is non-linear
2. $\frac{\partial^2}{\partial n^2}$ neither commutes nor associates with convolution

$$\frac{\partial^2}{\partial n^2}(g*f) \neq \left(\frac{\partial^2 g}{\partial n^2}\right)*f$$
$$\left(\frac{\partial^2 g}{\partial n^2}\right)*f \neq g*\left(\frac{\partial^2 f}{\partial n^2}\right)$$

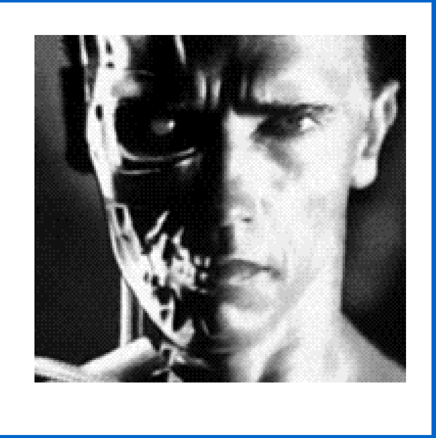
3. $\frac{\partial^2}{\partial n^2}$ is not everywhere defined (i.e., require ${f_x}^2 + {f_y}^2 \neq 0$)

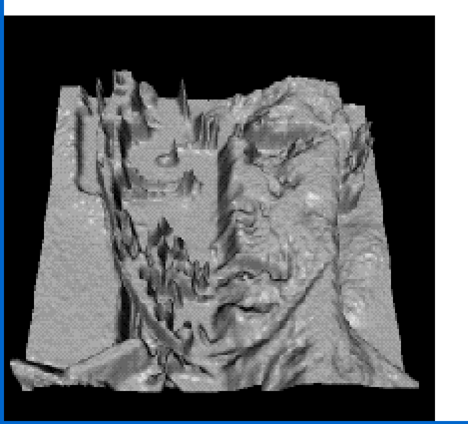
Experimental:

4. $\frac{\partial^2}{\partial n^2}$ provides better localization, especially at corners

Facet Model

- 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).

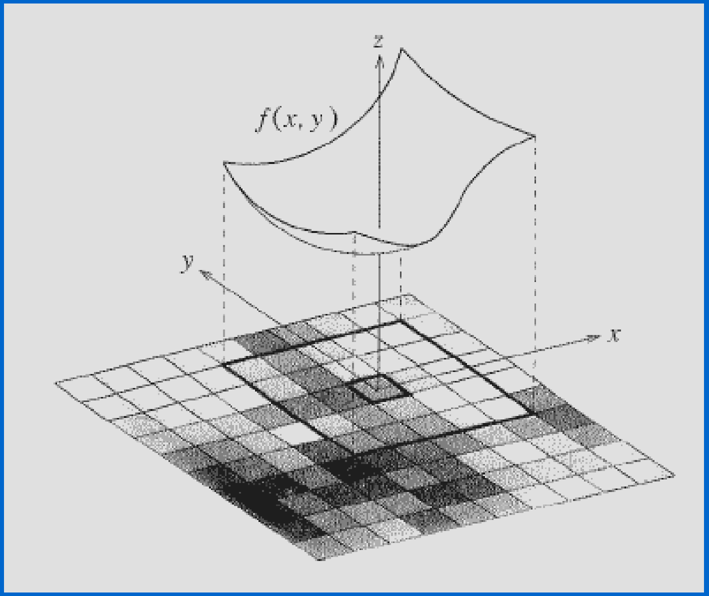




z=f(x,y)

Facet Model (cont'd)

- 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 <u>each</u> 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 (cont'd)

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

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

- 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

$\frac{1}{420}$

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

$\frac{1}{70}$

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

$\frac{1}{140}$

-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

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

- Using polar coordinates $x = \rho \sin\theta,\ y = \rho\cos\theta$

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

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.

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

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:

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

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.

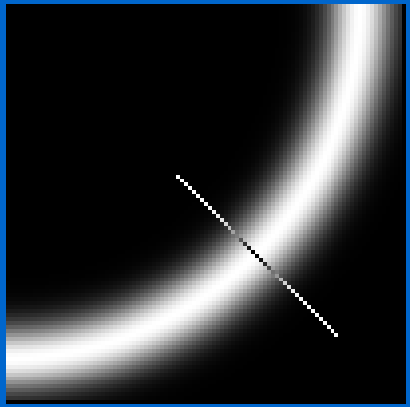
Figure 2.9: The steps in Haralick's Edge Detector.

Anisotropic Filtering
(i.e., edge preserving smoothing)

- Symmetric Gaussian smoothing tends to blur out edges rather aggressively.

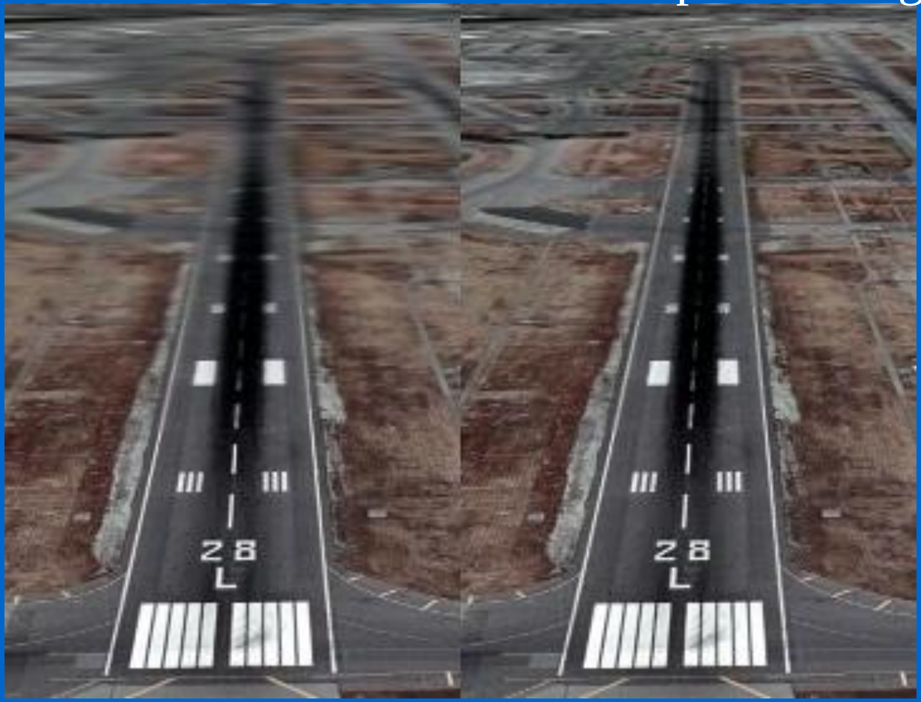
- An “oriented” smoothing operator would work better:
  (i) Smooth aggressively perpendicular to the gradient
  (ii) Smooth little along the gradient

- Mathematically formulated using diffusion equation.



Anisotropic filtering - Example

result using
anisotropic filtering



Effect of scale (i.e., σ)

original

$\sigma = 1$

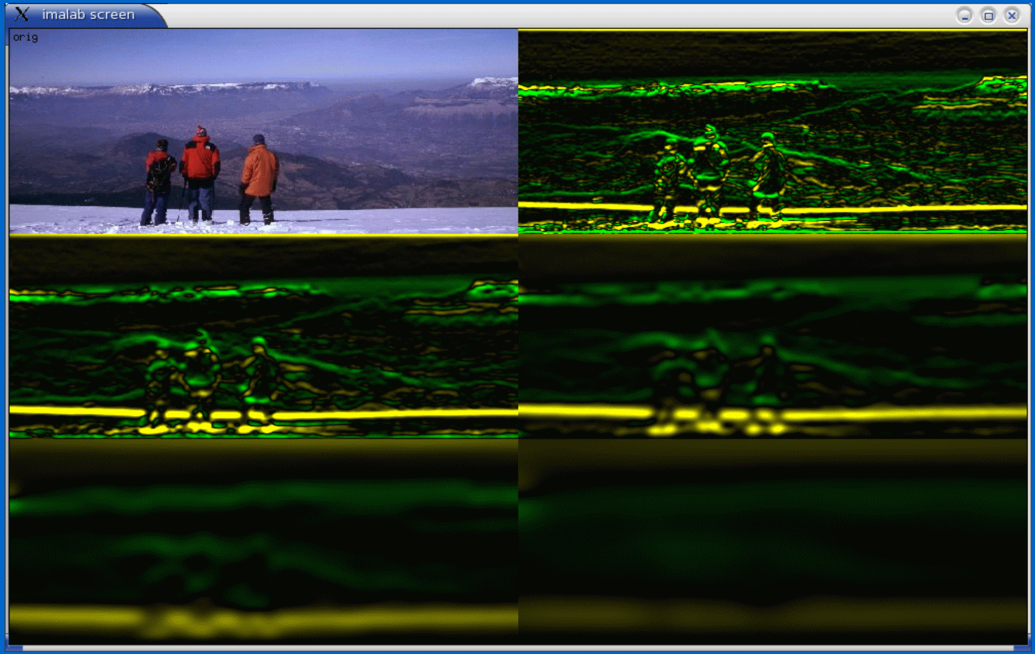
$\sigma = 2$

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

Multi-scale Processing

- A formal theory for handling image structures at different scales.
- Process images **multiple scales**.
- Determine which structures (e.g., edges) are most significant by considering **the range of scales** over which they occur.

Multi-scale Processing (cont'd)



σ=1

σ=2

σ=4

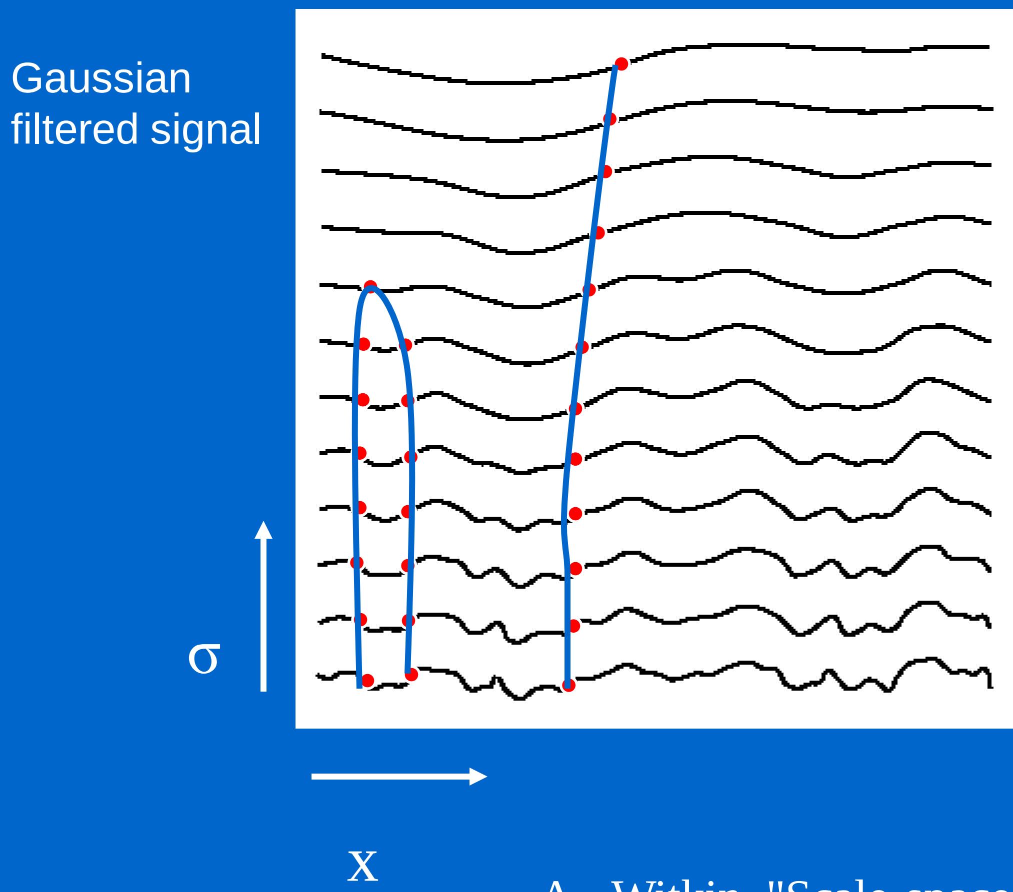
σ=8

σ=16

•**Interesting scales**: scales at which important structures are present.

e.g., in the image above, people can be detected at scales [1.0 - 4.0]

Scale Space (Witkin 1983)

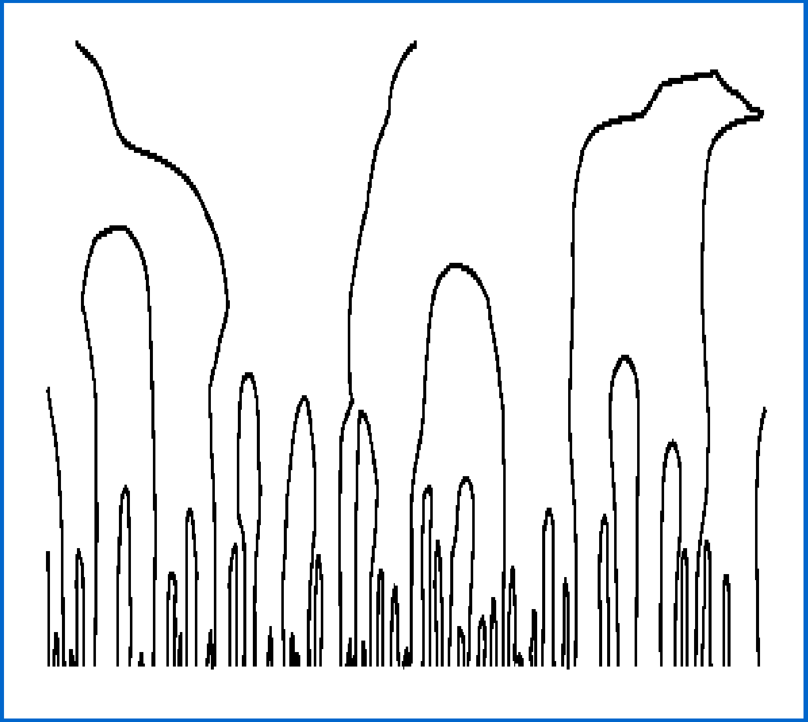


- Detect and plot the zero-crossing of a 1D function over a continuum of scales σ.

- Instead of treating zero-crossings at a single scale as a single point, we can now treat them at multiple scales as contours.

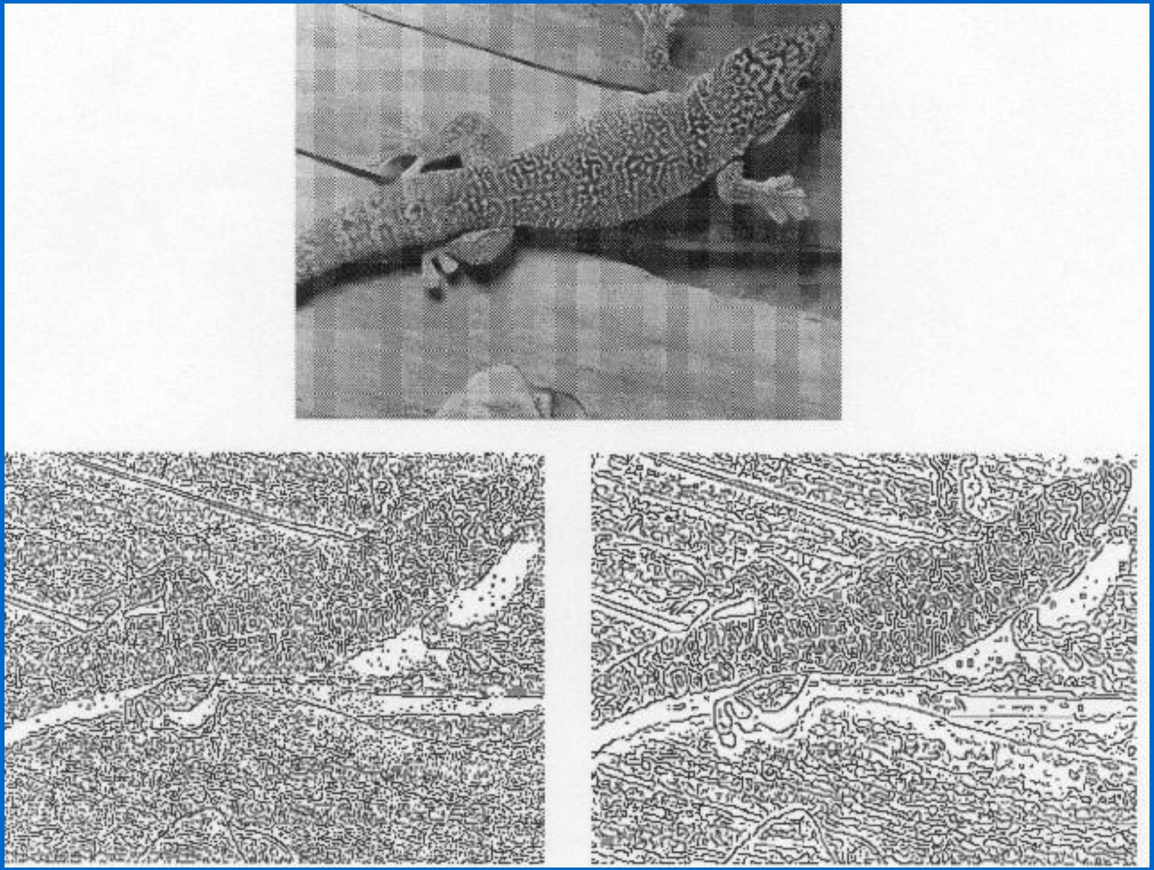
A. Witkin, "Scale-space filtering", 8th Int. Joint Conf. Art. Intell., Karlsruhe, Germany,1019–1022, 1983

Scale Space (cont'd)



- Properties of scale space (assuming Gaussian smoothing):
  - Zero-crossings may shift with increasing scale (σ).
  - Two zero-crossing may merge with increasing scale.
  - A contour may ***not*** split into two with increasing scale.

Multi-scale processing (cont'd)



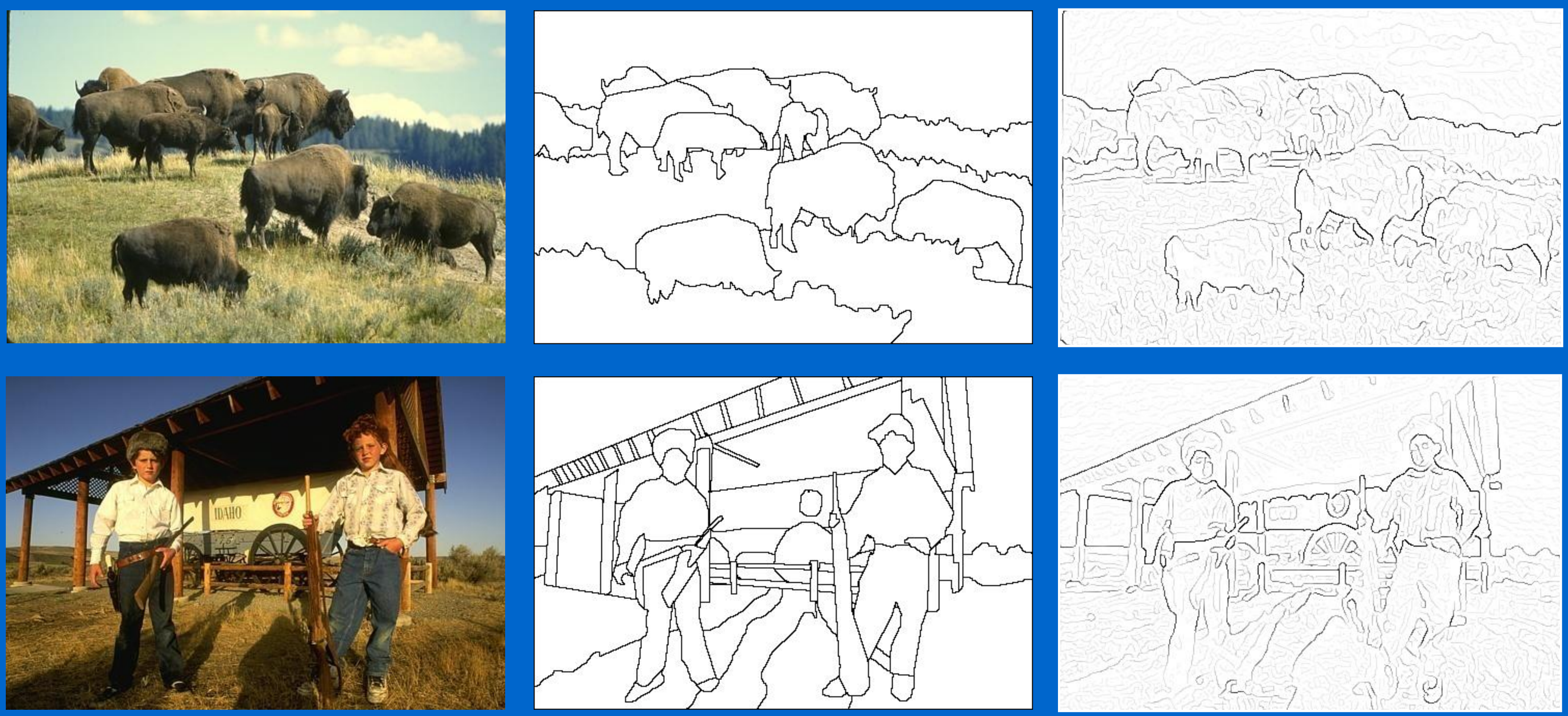
(Canny edges at multiple scales of smoothing, σ=0.5, 1,

Multi-scale processing (cont'd)

(Canny edges at multiple scales of smoothing, σ= , 2, 4, 8, 16)

Edge detection is just the beginning…

image | human segmentation | gradient magnitude



- Berkeley segmentation database:
  http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/