CSE 185 Introduction to Computer Vision

Lecture 3: Image Processing

UC MERCED

Slides credit: Yuri Boykov, Ming-Hsuan Yang, Boqing Gong, Richard Szeliski, Steve Seitz, Alyosha Efros, Fei-Fei Li, etc.

Image Processing

- An image processing operation defines a new image *g* in terms of an existing image *f*

  - *Geometric (domain) transformations:*

    $$g(x,y) = f(t_x(x,y), t_y(x,y))$$

  - *Range transformation:*

    $$g(x,y) = t(f(x,y))$$

    point processing

  - *Filtering also generates new images from an existing image*

    $$g(x,y) = \int_{\substack{|u|<\varepsilon \\ |v|<\varepsilon}} h(u,v) \cdot f(x-u, y-v) \cdot du \cdot dv$$

    neighborhood processing

Point processing

$$g(x,y) = t(f(x,y))$$

image range → image range

$$t : R \rightarrow R$$

for each original image intensity value I function $t(\cdot)$ returns a transformed intensity value $t(I)$.

$$\tilde{I} = t(I)$$

NOTE: we will often use notation I_p instead of $f(x,y)$ to denote intensity at pixel $p=(x,y)$

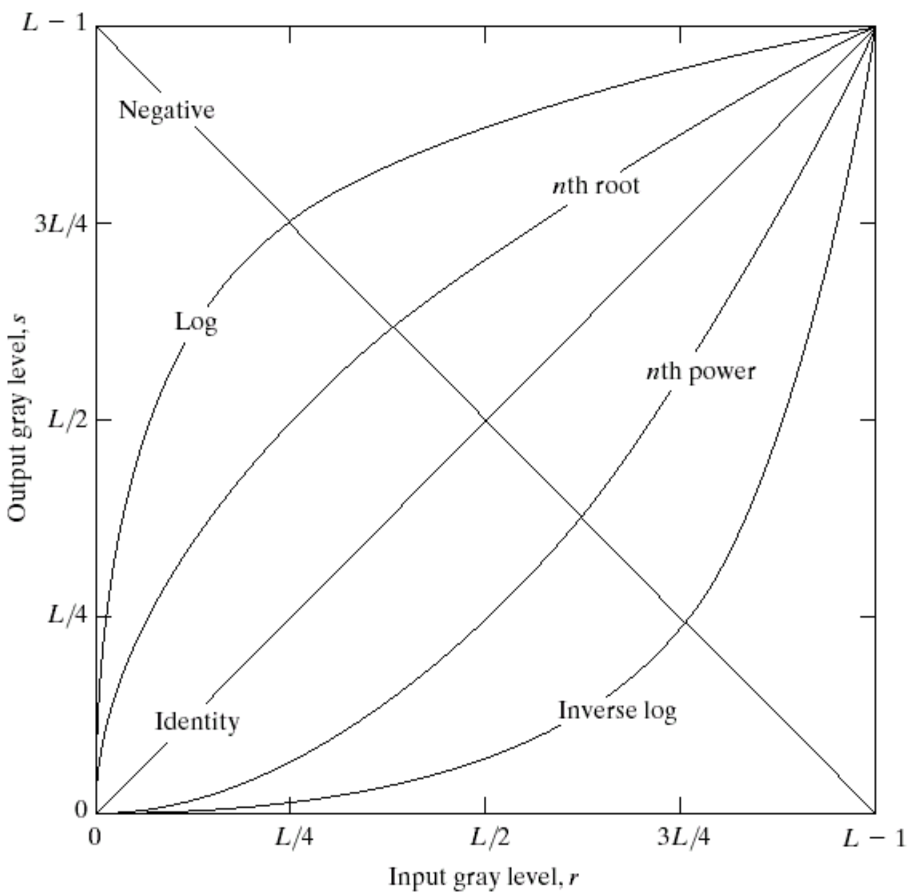
- **Important:** every pixel is for itself
  - spatial information is ignored!
- What can point processing do?

(we will focus on grey scale images, see Szeliski 3.1 for examples of point processing for color images)

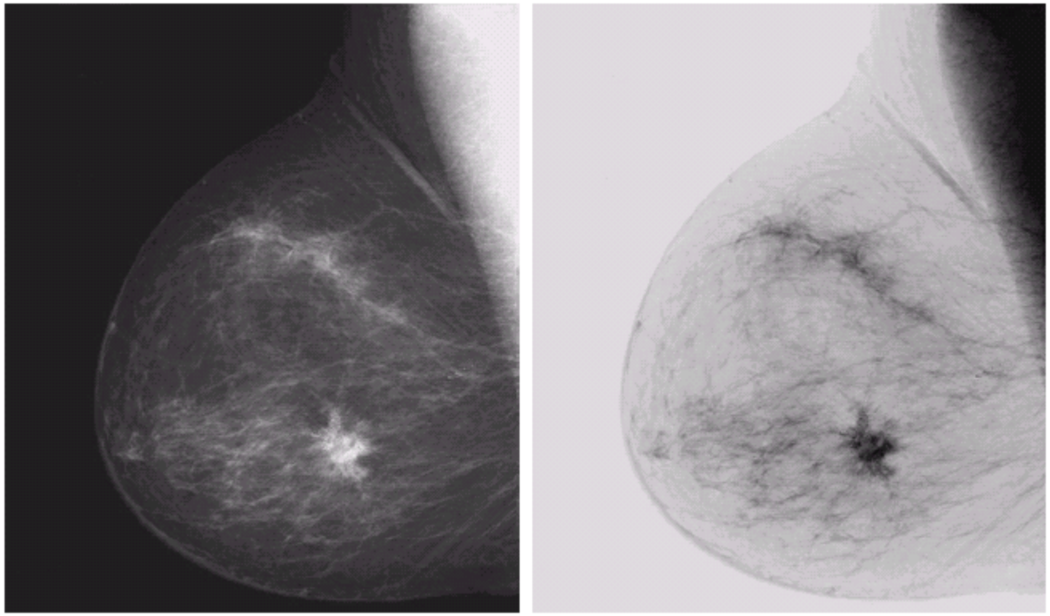
Example of gray-scale transformation *t*

$$\tilde{I} = t(I)$$

FIGURE 3.3 Some basic gray-level transformation functions used for image enhancement.



Point processing: negative



a b

FIGURE 3.4 (a) Original digital mammogram. (b) Negative image obtained using the negative transformation in Eq. (3.2-1). (Courtesy of G.E. Medical Systems.)

I_p or $f(x,y)$ $\quad\quad$ I'_p or $g(x,y)$

$$t(I) = 255 - I$$

$$g(x,y) = t(f(x,y)) = 255 - f(x,y)$$

Power-law transformations t

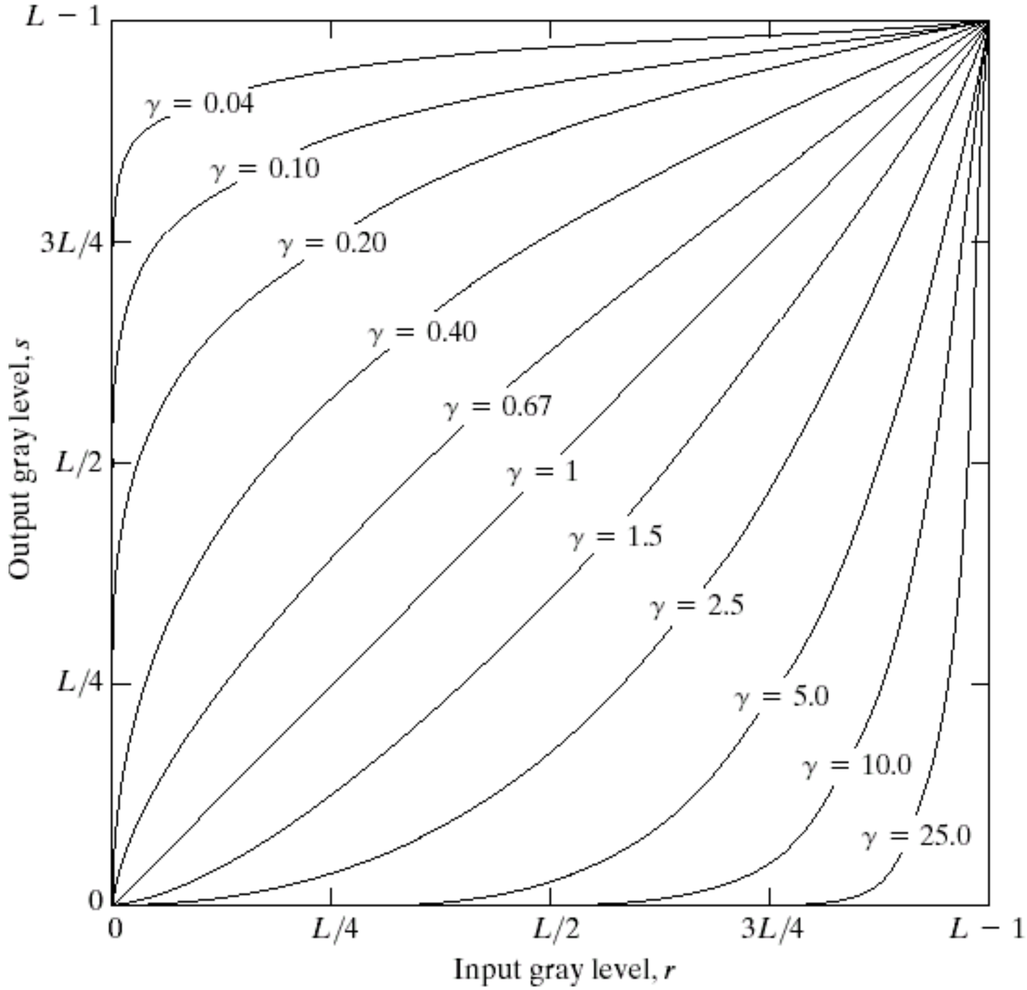


FIGURE 3.6 Plots of the equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases).

$$t(I) = I^{\gamma}$$

Image enhancement via gamma correction

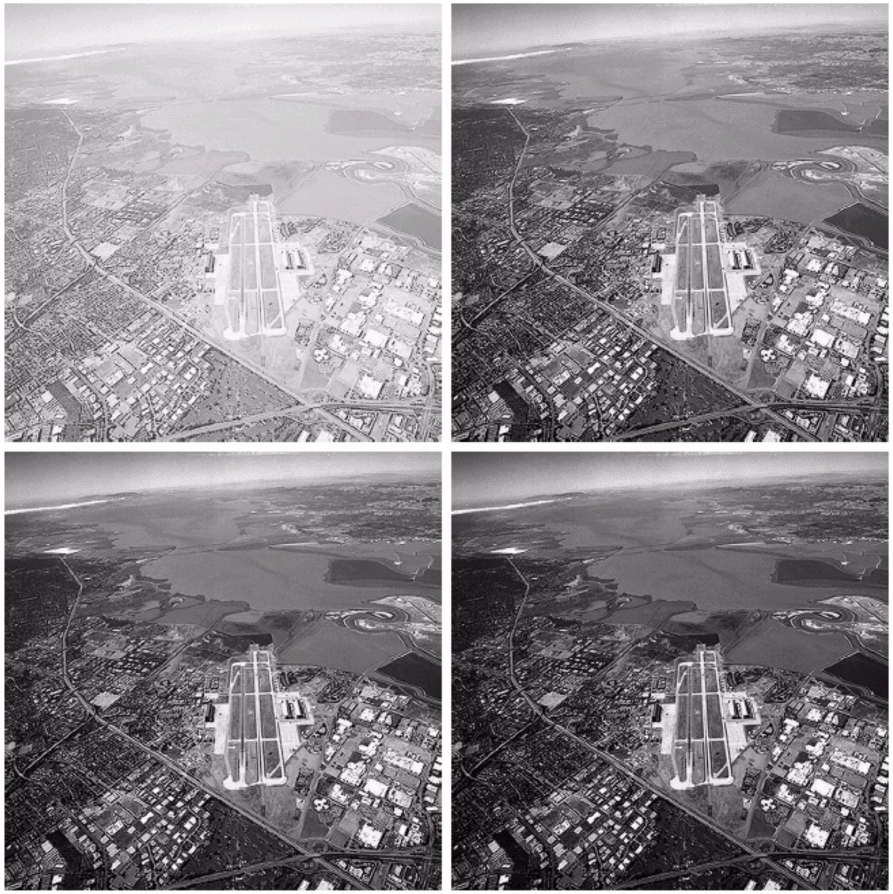


FIGURE 3.9 (a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0$, and 5.0, respectively. (Original image for this example courtesy of NASA.)

Image histogram

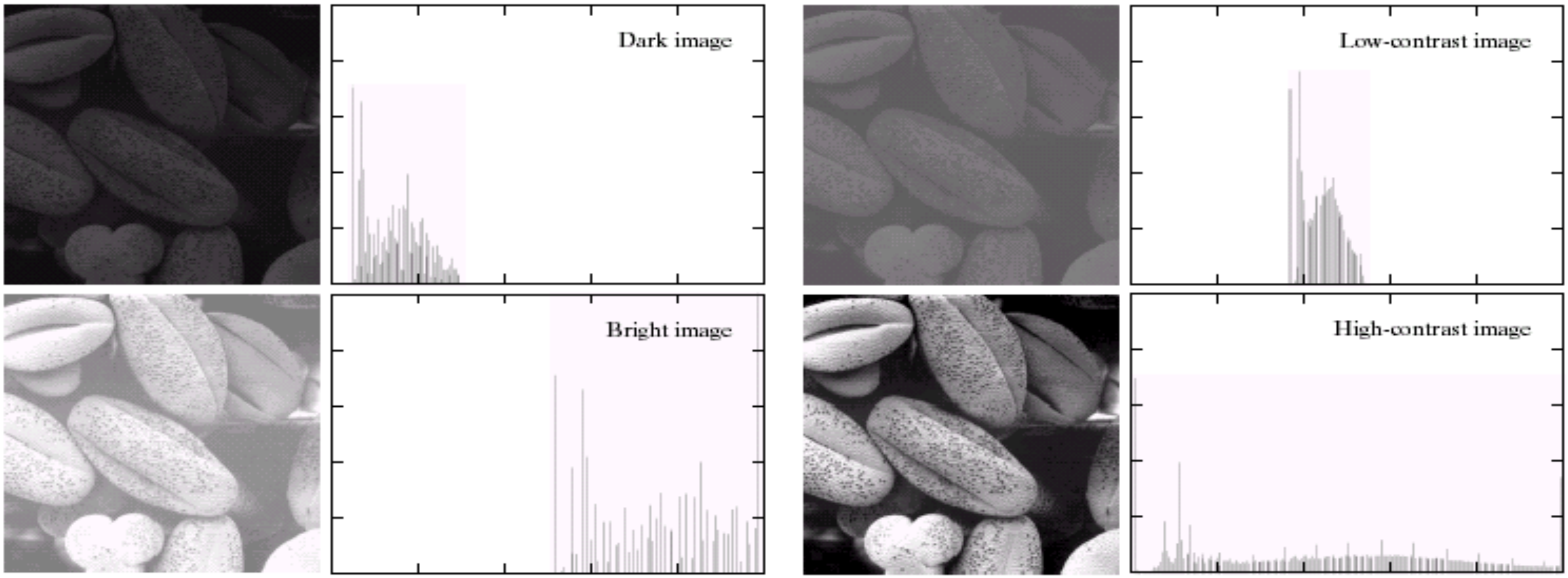


Image Brightness

Image Contrast

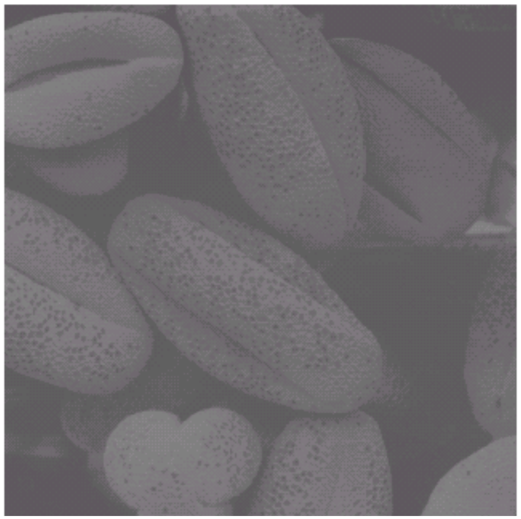
probability of intensity *i* : $p(i) = \frac{n_i}{n}$

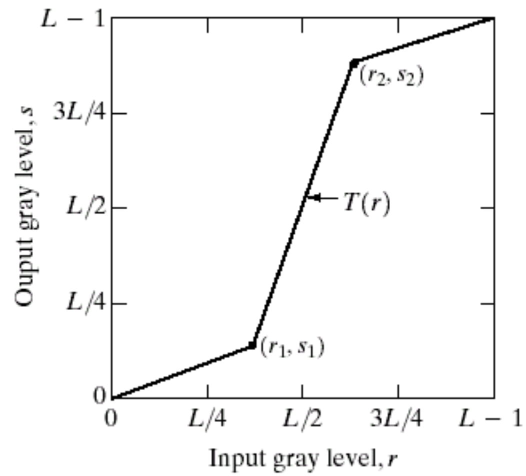
n_i ---number of pixels with intensity *i*

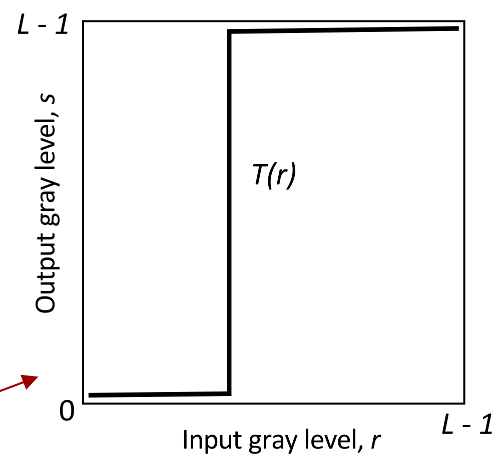
n ---total number of pixels in the image

Contrast stretching

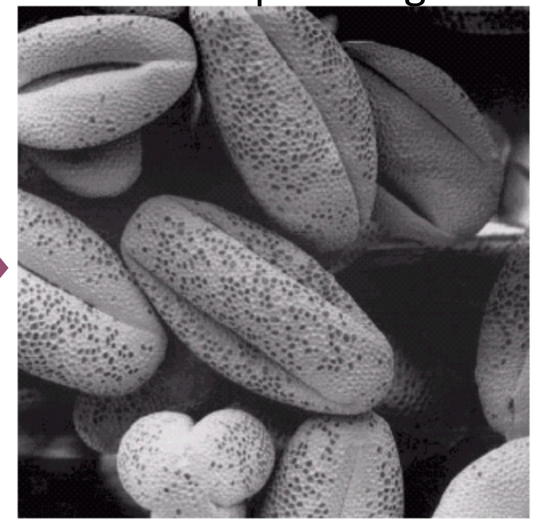
Original image







Output image



a.k.a. intensity *thresholding*

Histogram equalization

$$t(i) = \sum_{j=0}^{i} p(j) = \sum_{j=0}^{i} \frac{n_j}{n}$$

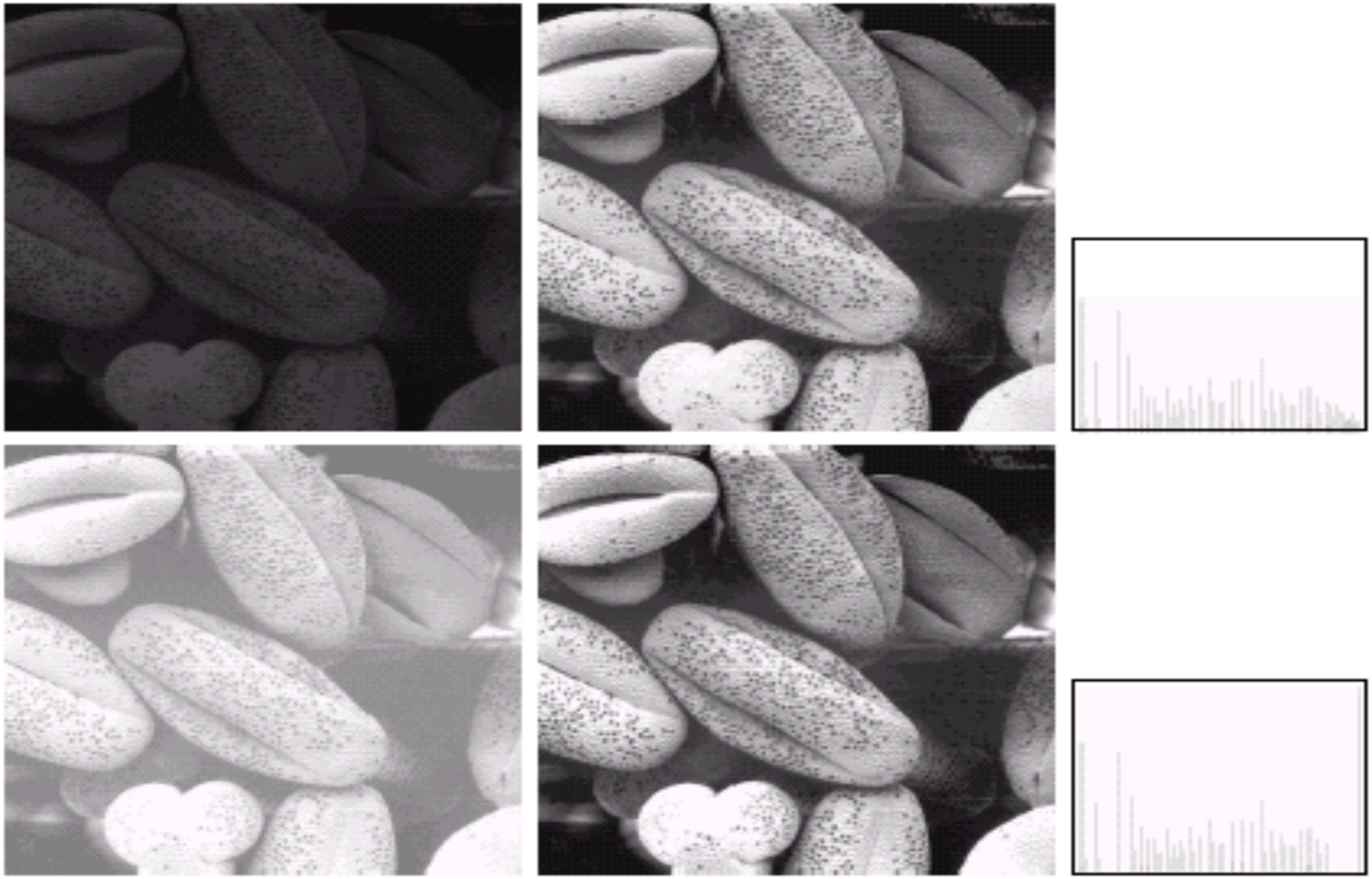
= cumulative distribution of image intensities

Histogram equalization

Original images | Histogram corrected images

1)

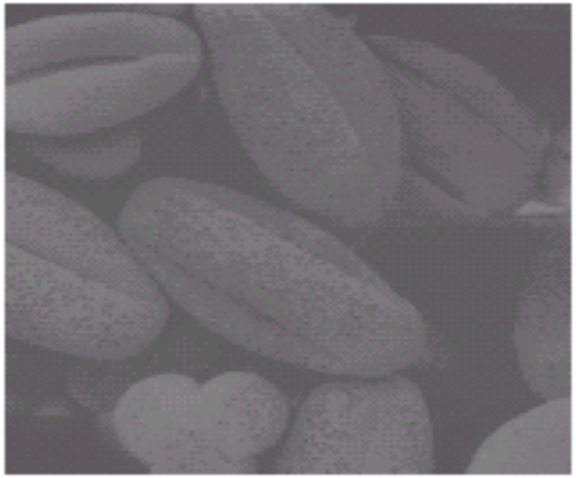
2)

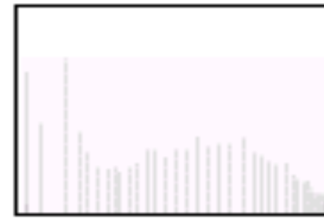


Histogram equalization

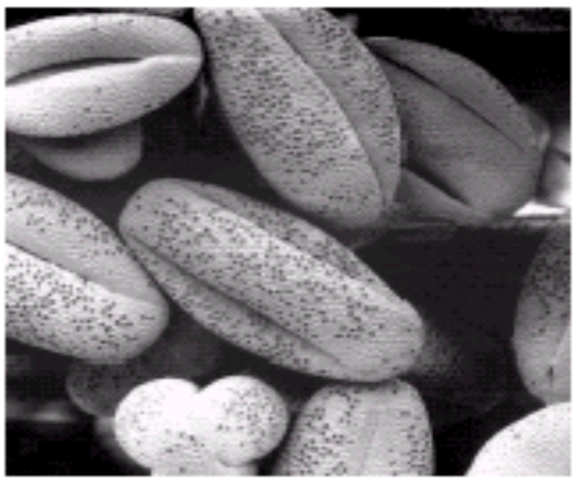
Original images | Histogram corrected images

3)



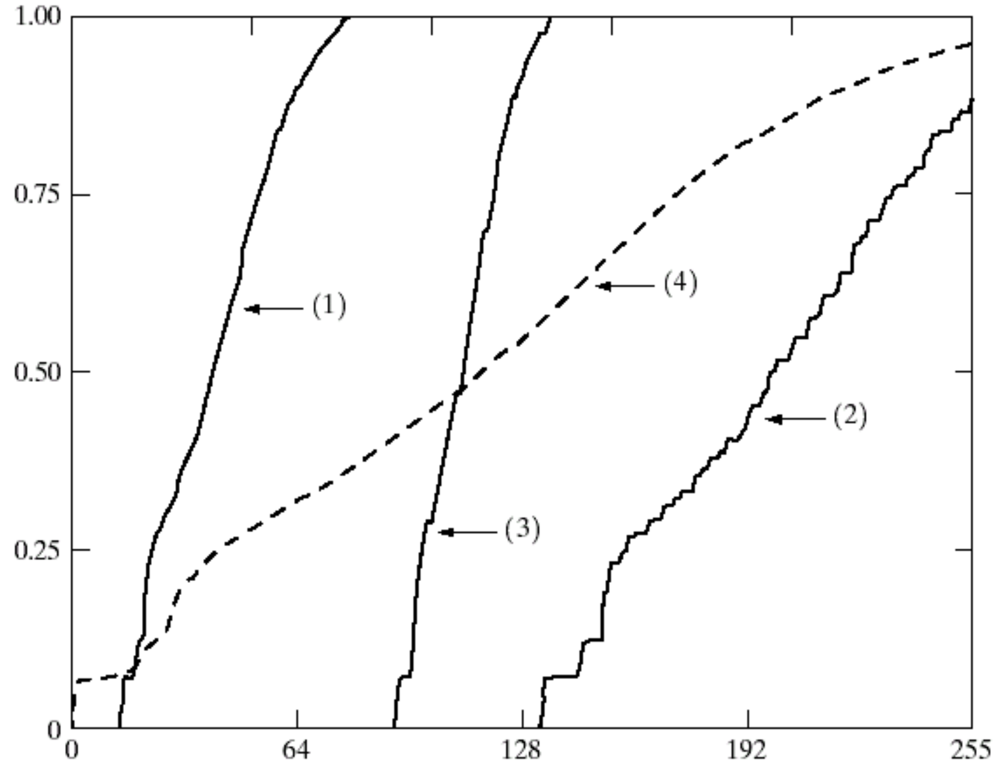


4)



Histogram equalization

FIGURE 3.18 Transformation functions (1) through (4) were obtained from the histograms of the images in Fig.3.17(a), using Eq. (3.3-8).



$$t(i) = \sum_{j=0}^{i} p(j) = \sum_{j=0}^{i} \frac{n_j}{n}$$

= cumulative distribution of image intensities

…see Gonzalez and Woods, Sec3.3.1, for more details

Histogram equalization

$$t(i) = \sum_{j=0}^{i} p(j) = \sum_{j=0}^{i} \frac{n_j}{n}$$

= cumulative distribution
of image intensities

Q: Why does that work?

Answer in probability theory:

I – random variable with *probability* distribution *p(i)* over *i* in *[0,1]*

If *t(i)* is a *cumulative* distribution of *I* then

I′=t(I) – is a random variable with *uniform* distribution over its range [0,1]

That is, transform image *I′* will have a uniformly-spread histogram (good contrast)

Histogram equalization for continuous case

- From basic probability theory

$$p_f(f) \xrightarrow{f} \boxed{T(f)} \xrightarrow{g} \qquad p_g(g) = \left[p_f(f) \frac{df}{dg} \right]_{f=T^{-1}(g)}$$

- Consider the transformation function

$$g = T(f) = \int_0^f p_f(\alpha)\, d\alpha \qquad 0 \le f \le 1$$

- Then . . .

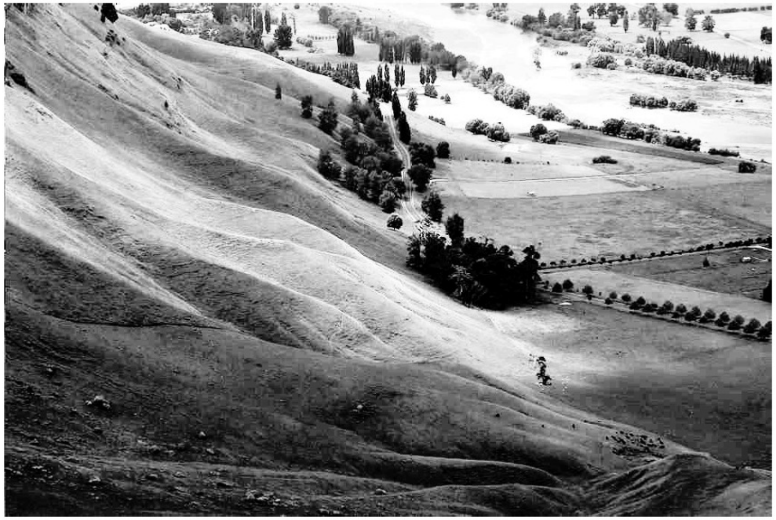
$$\frac{dg}{df} = p_f(f)$$

$$p_g(g) = \left[p_f(f) \frac{df}{dg} \right]_{f=T^{-1}(g)} = \left[p_f(f) \frac{1}{p_f(f)} \right]_{f=T^{-1}(g)} = 1 \qquad 0 \le g \le 1$$

slide from Bernd Girod

Histogram equalization example

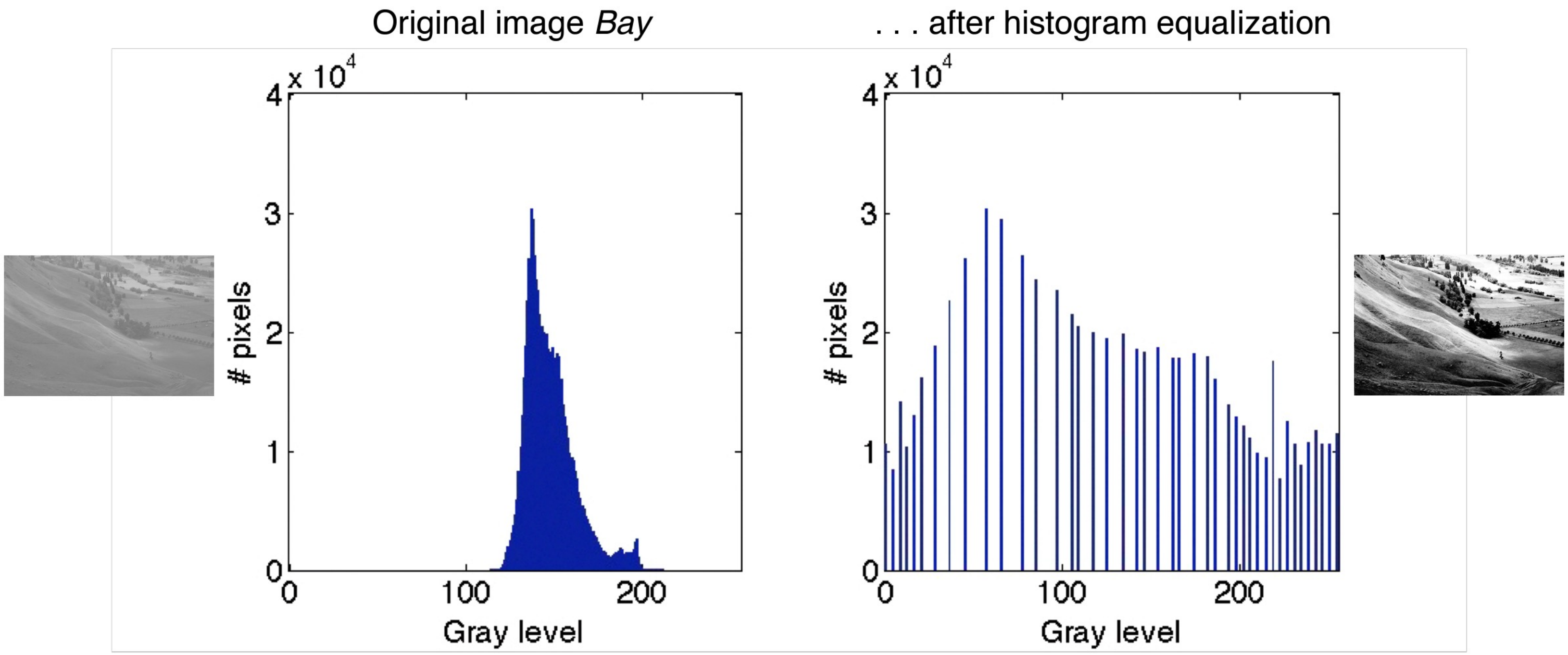
Original image *Bay*



... after histogram equalization

slide from Bernd Girod

Histogram equalization example



slide from Bernd Girod

From point to neighborhood processing

point processing:

$$g(x, y) = t(f(x, y))$$

neighborhood processing:

$$g(x, y) = \int_{\substack{|u|<\varepsilon \\ |v|<\varepsilon}} h(u, v) \cdot f(x-u, y-v) \cdot du \cdot dv$$

2D Convolution

A 2D image *f[i,j]* can be filtered by a **2D kernel** *h[u,v]* to produce an output image *g[i,j]*:

$$g[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] \cdot f[i+u, j+v]$$

This is called a **convolution** operation and written:

$$g = h \circ f$$

h is called "**kernel**" or "**mask**" or "**filter**" which representing a given "window function"

2D filtering for noise reduction

- Common types of noise:
  - **Salt and pepper noise**: random occurrences of black and white pixels
  - **Impulse noise:** random occurrences of white pixels
  - **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Original

Salt and pepper noise

Impulse noise

Gaussian noise

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x,y]$

		10							

$g[x,y]$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x,y]$

				80					
		10							

$g[x,y]$

Mean filtering

side effect of mean filtering: **blurring**

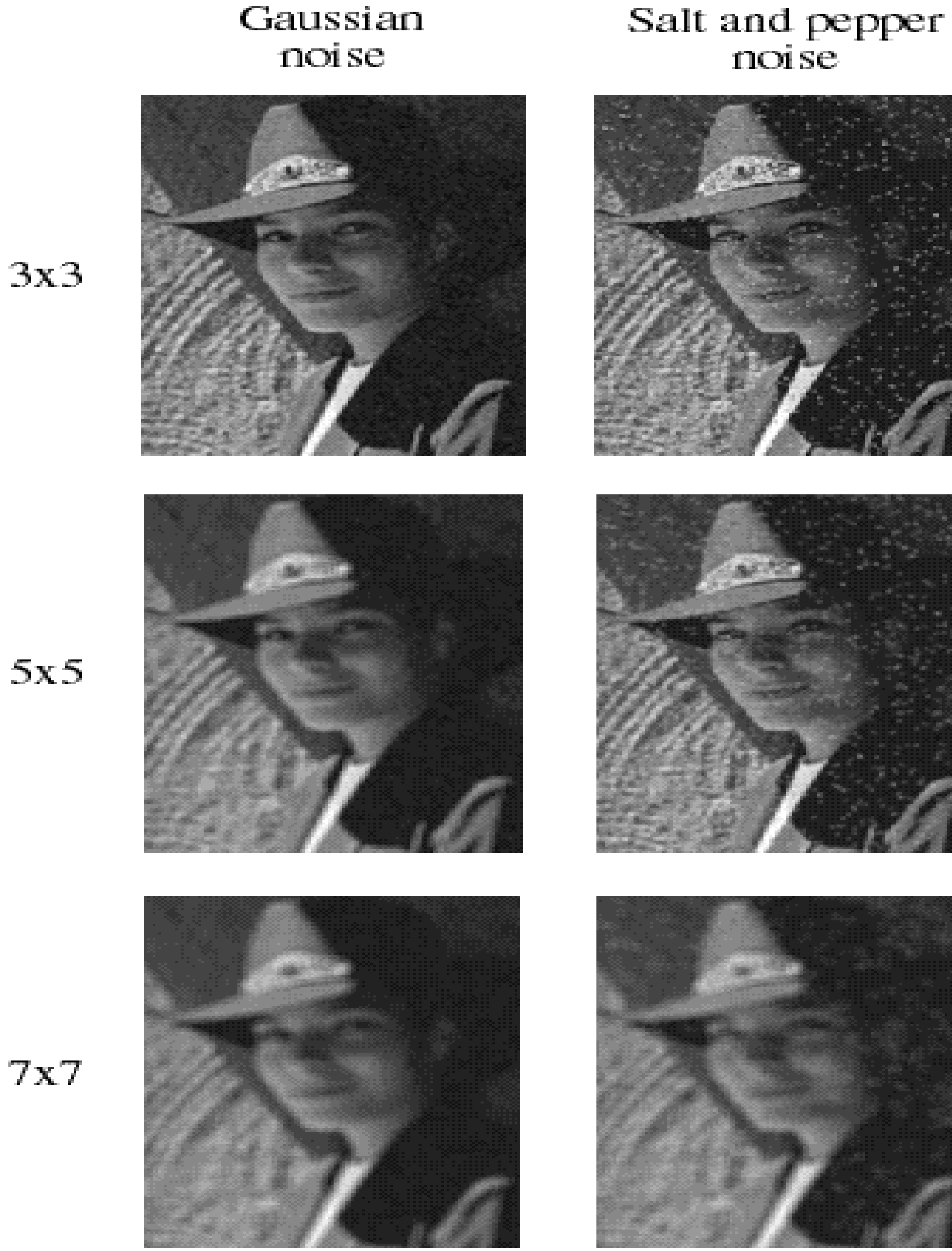
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$g[x, y]$

Mean filtering



Mean kernel

- What's the kernel for a 3x3 mean filter?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$\frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

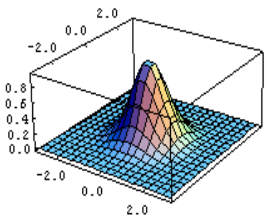
Gaussian filtering

- A Gaussian kernel gives less weight to pixels further from the center

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$\frac{1}{16} \cdot$

1	2	1
2	4	2
1	2	1



discrete approximation of
a Gaussian (density) function

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

- NOTE: ***Gaussian*** distribution is a synonym for ***Normal*** distribution!

Gaussian filtering

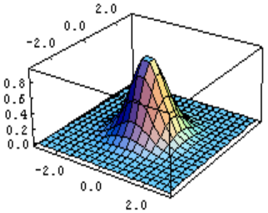
- A Gaussian kernel gives less weight to pixels further from the center

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$\frac{1}{16} \cdot$

1	2	1
2	4	2
1	2	1

$\mathbf{G}_\sigma$

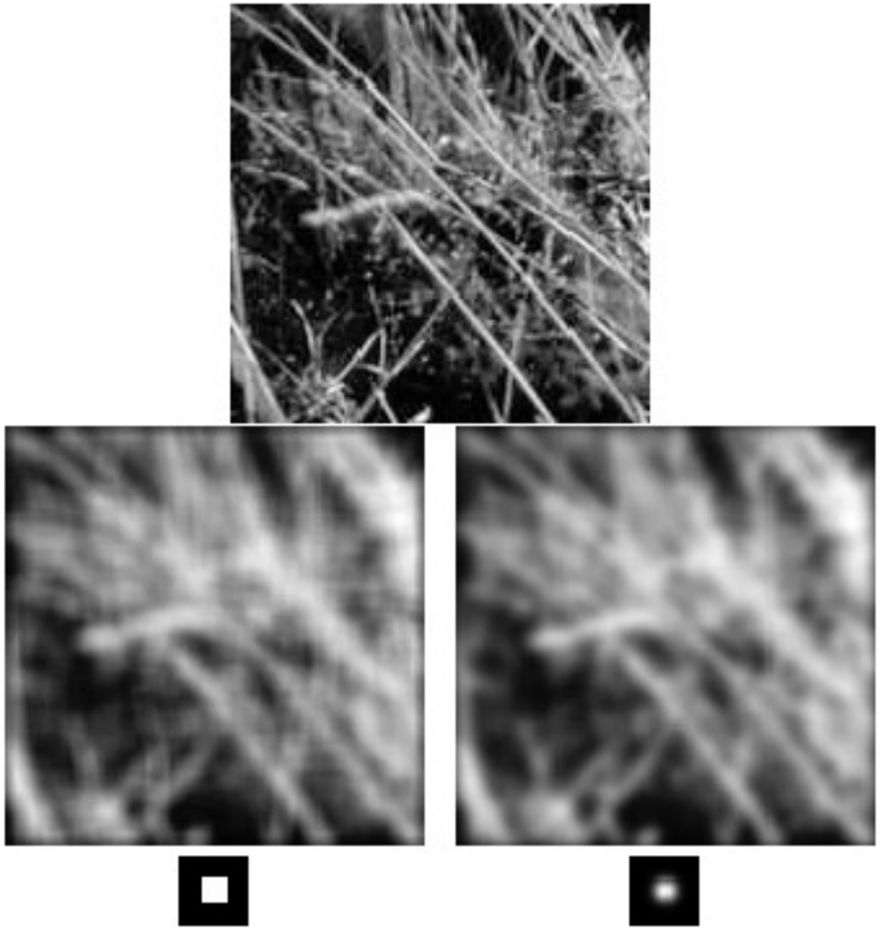


discrete approximation of a Gaussian (density) function

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

We denote such Gaussian kernels by ***G*** or $\boldsymbol{G_\sigma}$

Mean vs Gaussian filtering



no rotational invariance

Median filter

- A **Median Filter** operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?
- Is a median filter a kind of convolution?
    - \- No, median filter is non-linear

Comparison: **salt and pepper** noise

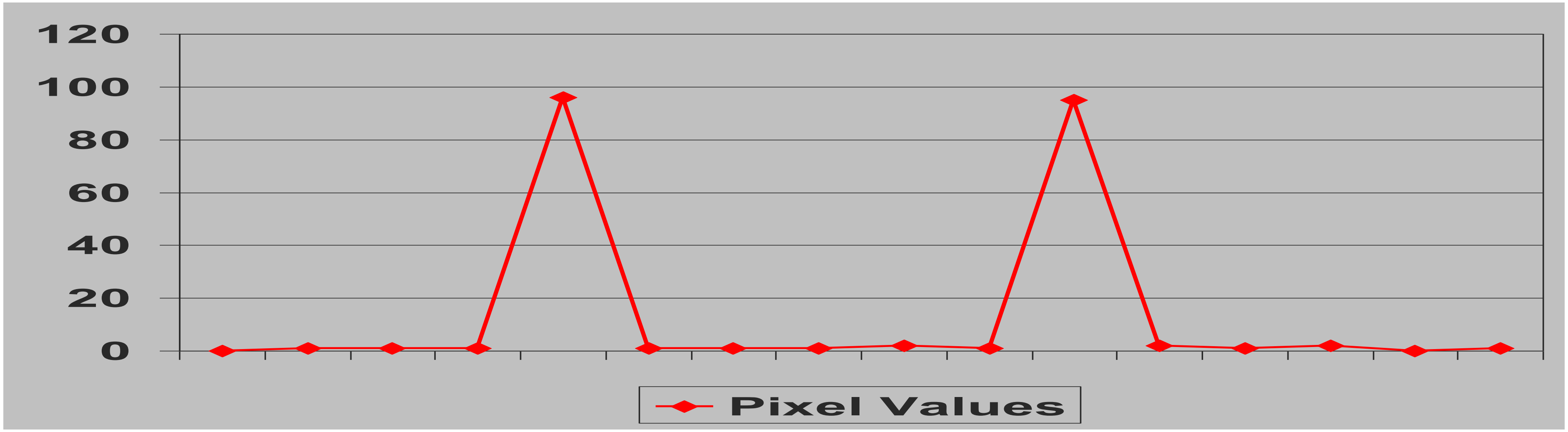
Edge detection

- The purpose of Edge Detection is to find jumps in the brightness function (of an image) and mark them.

Edge = abrupt change in pixel intensity

1 2 1 0 98 99 98 97 99 98 1 2 1 2

Look at: abs(jumps in value sideways)



Edge detection

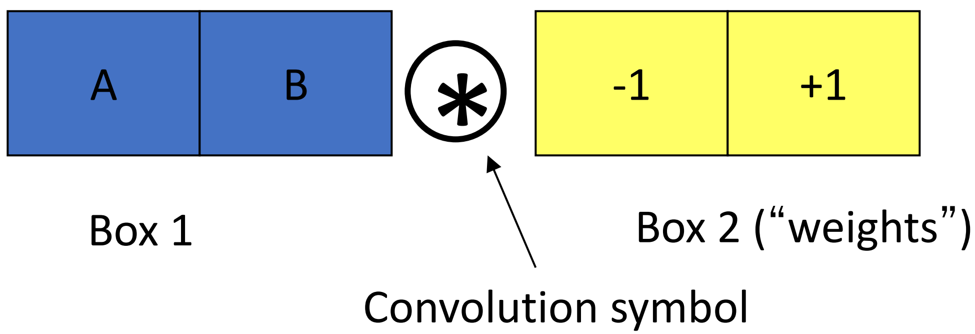
- Create the algorithm in pseudocode:

```
while row not ended  // keep scanning until end of row
select the next A and B pair, which are
neighboring pixels.
    diff = B – A                //formula to show math
    if abs(diff) > Threshold    //(THR)
        mark as edge
```

Above is a simple solution to detecting the differences in pixel values that are side by side.

Edge detection as convolution

- diff = B – A is the same as:

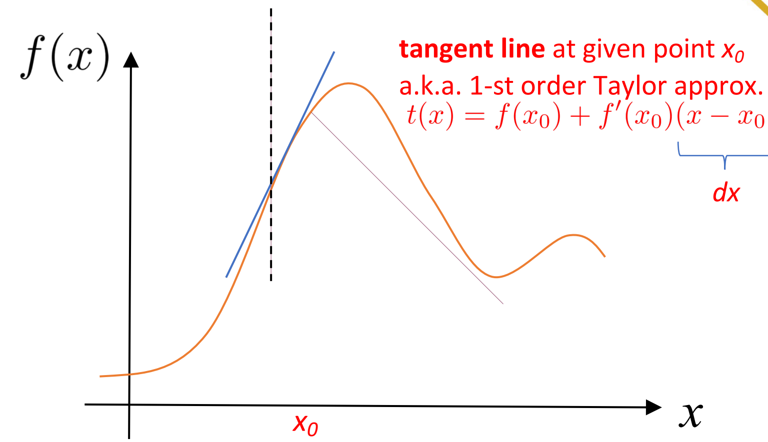


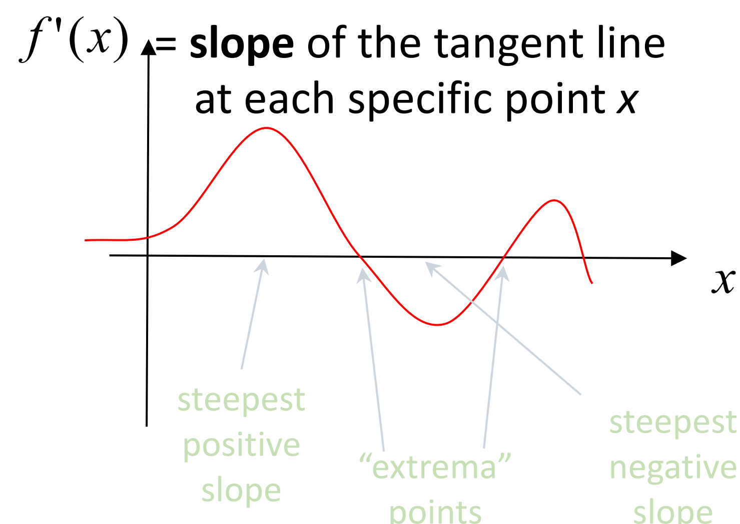
Place box 2 on top of box 1, multiply.
-1 * A and +1 * B
Result is –A + B which is the same as
B – A

Differentiation and convolution

- ❑ Recall for $f(x)$

$$f'(x) = \lim_{\varepsilon \to 0}\left(\frac{f(x+\varepsilon) - f(x)}{\varepsilon}\right)$$

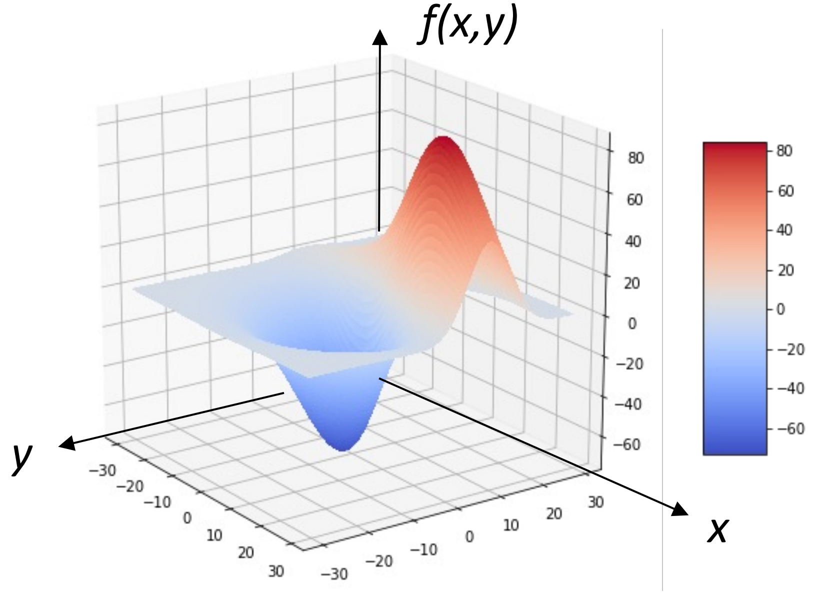




- ❑ Useful for analyzing *f(x)*
- ❑ How to extend differentiation to multivariate functions like $f(x,y)$ or $f(x,y,z)$?

Differentiation and convolution

$$f(x, y)$$



3D plot of *f*

What is "**slope**" of *f(x,y)* at a given point (*x,y*)?

Some intuition first:

- For functions *f(x,y)* think about the **slope** of a tangent plane for its 3D plot at point (x,y).

- Such a slope could be characterized by direction and magnitude - attributes of a **vector** (?)

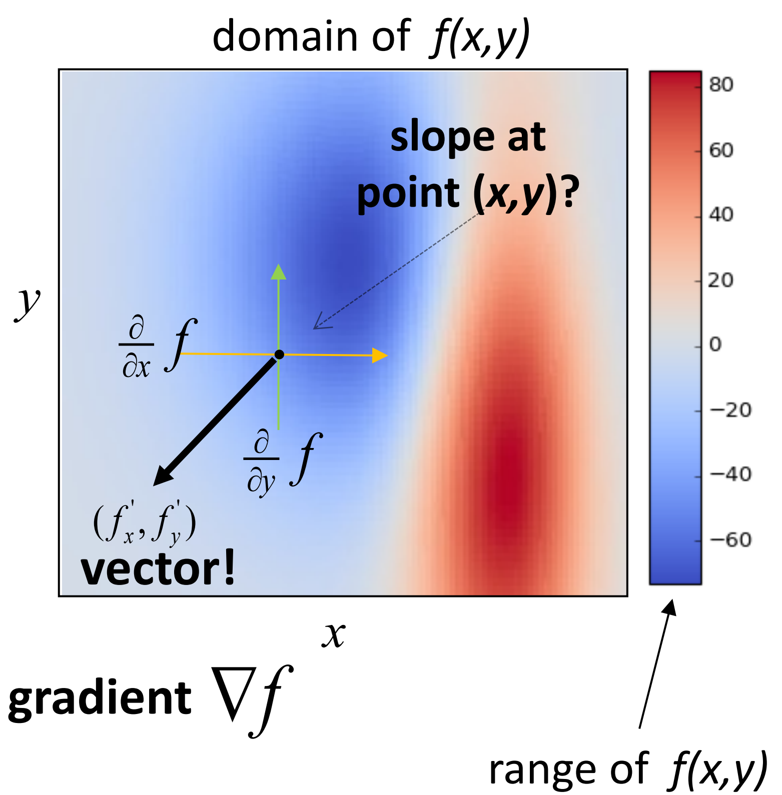
Differentiation and convolution

- For $f(x, y)$ use fixed directions

(e.g. "partial" derivatives)

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

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



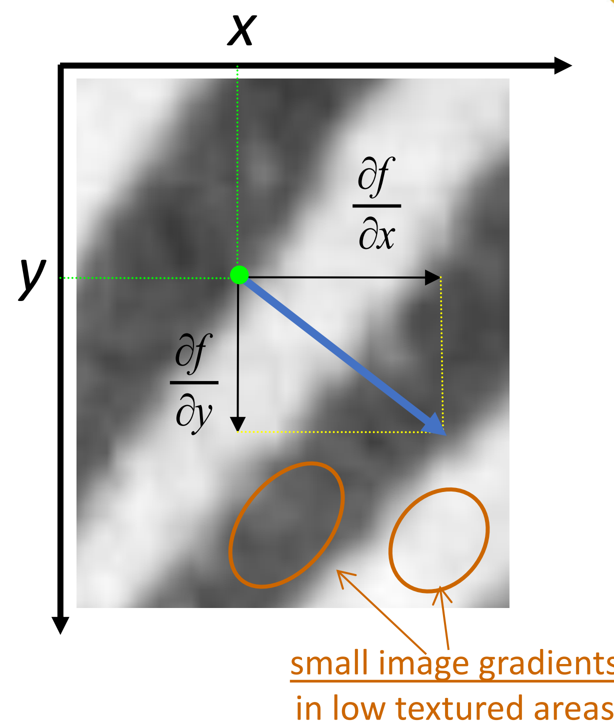
Gradients for function *f(x, y)*

- For a function of two (or more) variables

$$f(x, y)$$

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

two (or more) dimensional vector



- Gradient's absolute value $|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$ describes "steepness" of the "slope"
  - large at contrast edges, small in inform color regions

- Gradient's direction corresponds to the **steepest ascend** direction of the "slope"
  - gradient is orthogonal to image object boundaries

Partial derivative for image

partial derivative with respect to *x*

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

At given point (x_i, y_i)

one can approximate this as

$$\frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y_i) - f(x_{i-1}, y_i)}{2 \cdot \Delta x}$$

$$= \nabla_x * f$$

convolution with kernel

$$\nabla_x$$

$$\frac{1}{2\Delta x} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

Partial derivative for image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	40	60	60	60	40	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	40	60	60	60	40	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

At given point (x_i, y_i)

one can approximate this as

$$\frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y_i) - f(x_{i-1}, y_i)}{2 \cdot \Delta x}$$

$$= \nabla_x * f$$

convolution with kernel ∇_x

$\frac{1}{2\Delta x} \cdot$

0	0	0
1	0	-1
0	0	0

1/2*(90-0) = 45

Partial derivative for image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	40	60	60	60	40	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	40	60	60	60	40	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

At given point (x_i, y_i)

one can approximate this as

$$\frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y_i) - f(x_{i-1}, y_i)}{2 \cdot \Delta x}$$

$$= \nabla_x * f$$

convolution with kernel

∇_x

$$\frac{1}{2\Delta x} \cdot$$

0	0	0
1	0	-1
0	0	0

1/2*(0-90) = -45

Partial derivative for image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	40	60	60	60	40	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	60	90	90	90	60	0	0
0	0	0	40	60	60	60	40	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

At given point (x_i, y_i)

one can approximate this as

$$\frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y_i) - f(x_{i-1}, y_i)}{2 \cdot \Delta x}$$

$$= \nabla_x * f$$

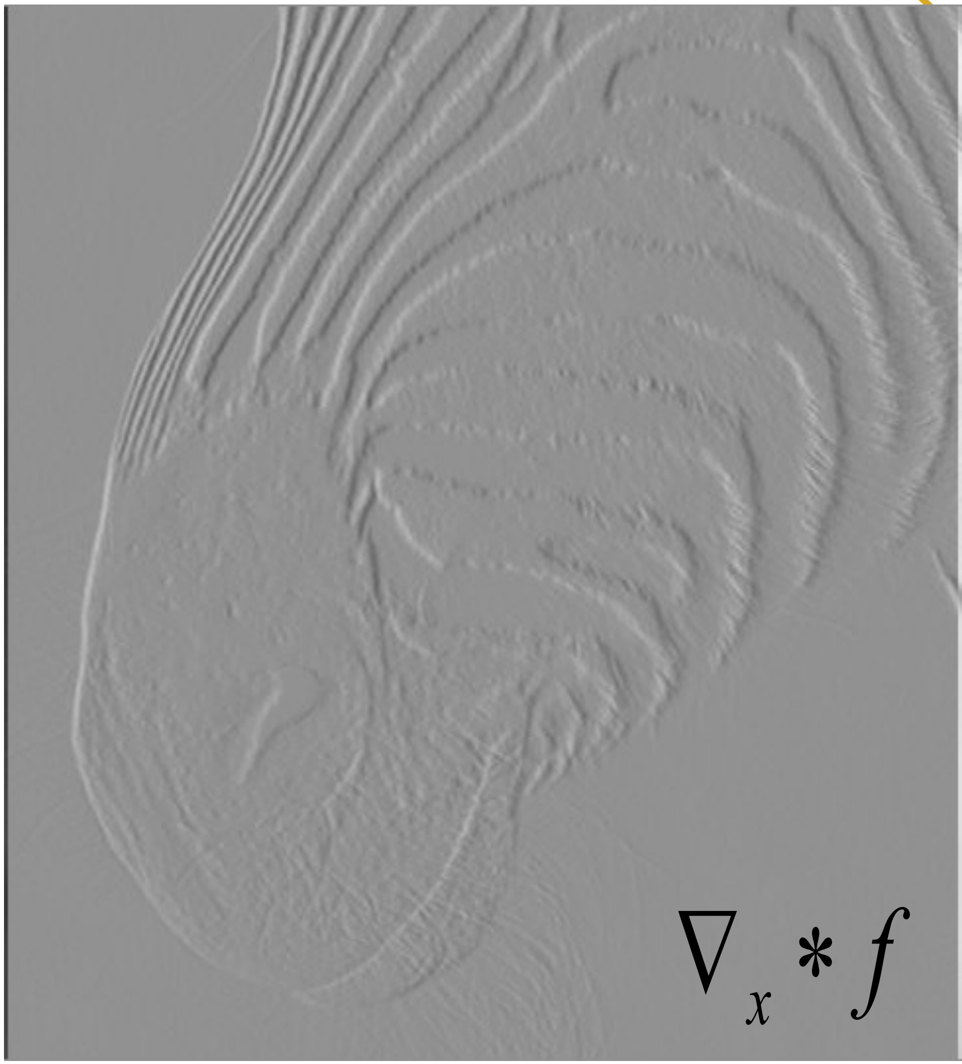
convolution with kernel ∇_x

$\frac{1}{2\Delta x} \cdot$

0	0	0
1	0	-1
0	0	0

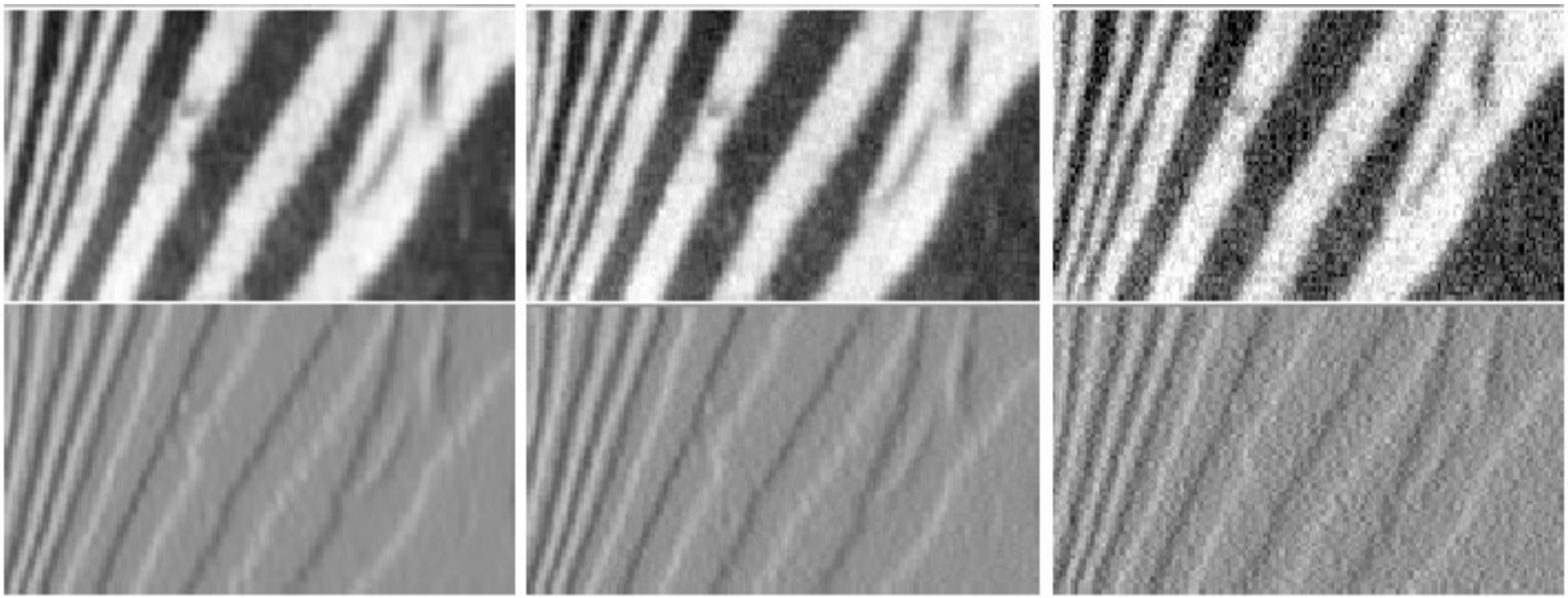
1/2*(60-60) = 0

Image gradient



$\nabla_x * f$

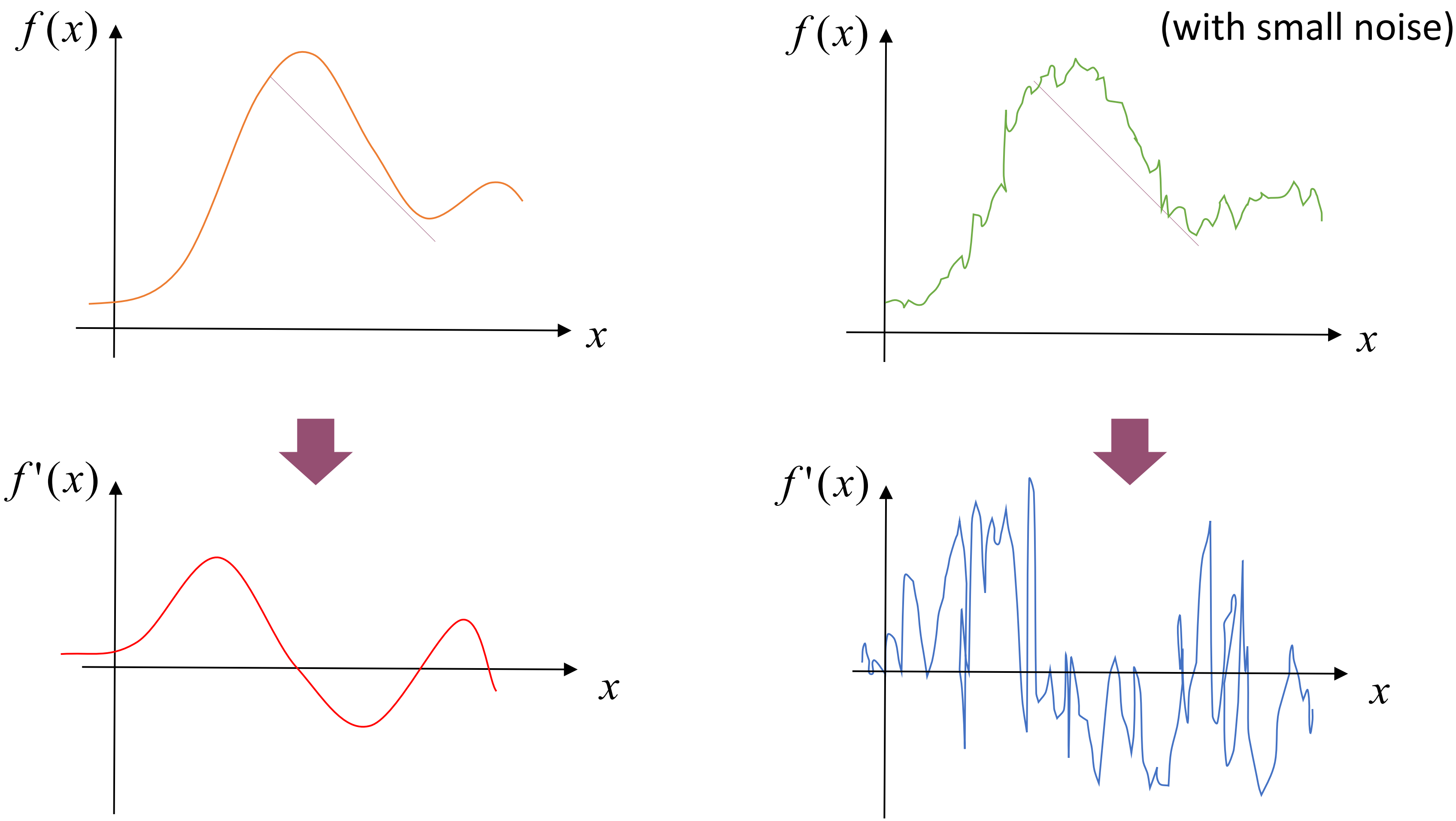
Image gradient responding to noise



increasing noise ->

(this is zero mean additive Gaussian noise)

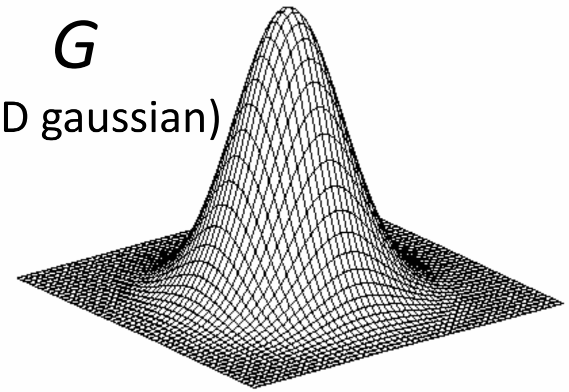
Gradient responding to noise

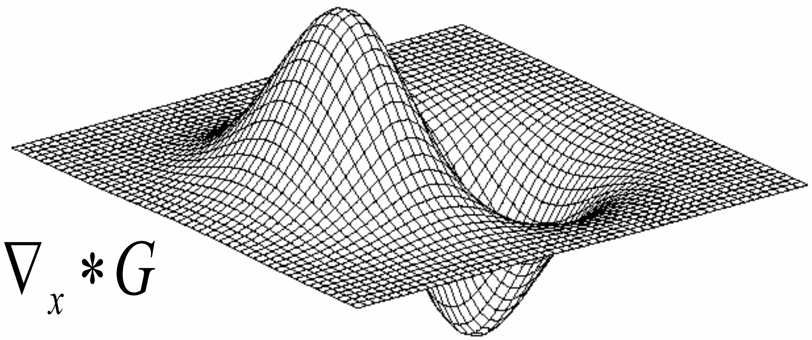


Smoothing and differentiation

- Issue: noise
  - smooth before differentiation
  - two convolutions: smooth, and then differentiate?
  - actually, no - we can use a derivative of Gaussian filter
    - because differentiation is convolution, and convolution is associative

$$\nabla_x * (G * f) = (\nabla_x * G) * f$$



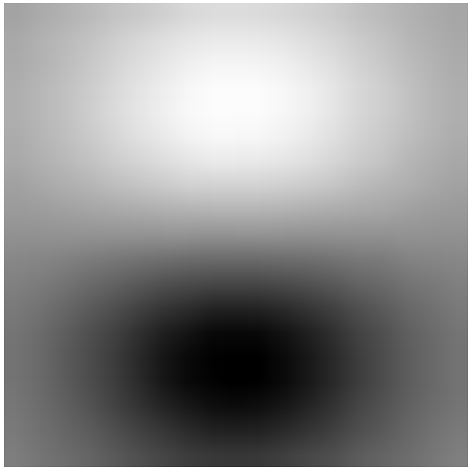


(*x*-derivative of 2D gaussian)

Smoothing and differentiation

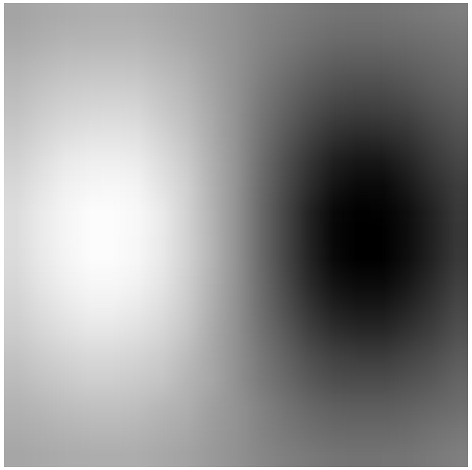
- Issue: noise
  - smooth before differentiation
  - two convolutions: smooth, and then differentiate?
  - actually, no - we can use a derivative of Gaussian filter
    - because differentiation is convolution, and convolution is associative

$\nabla_y * G$



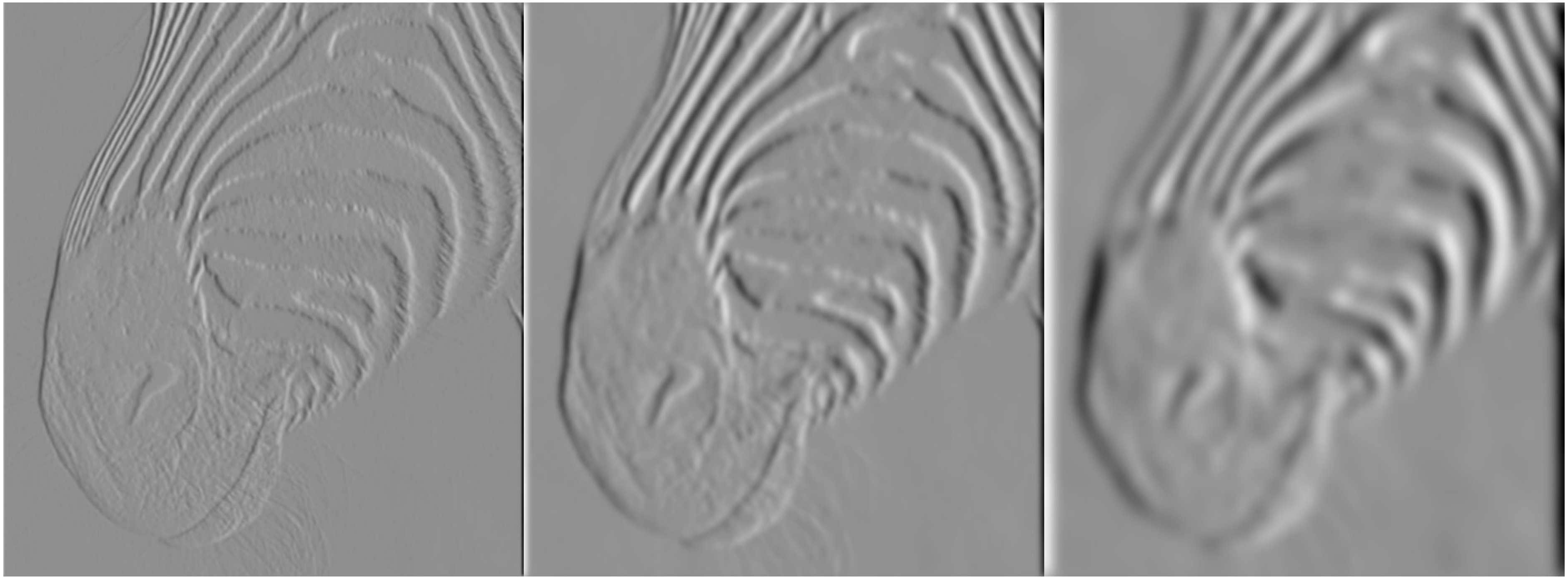
(*y*-derivative of 2D gaussian)

$\nabla_x * G$



(*x*-derivative of 2D gaussian)

$$(\nabla_x * G) * f$$



1 pixel 3 pixels 7 pixels

The scale of the smoothing filter (e.g. "bandwidth" σ of a Gaussian kernel) affects derivative estimates, and also the semantics of the edges recovered.

Image gradient and edges

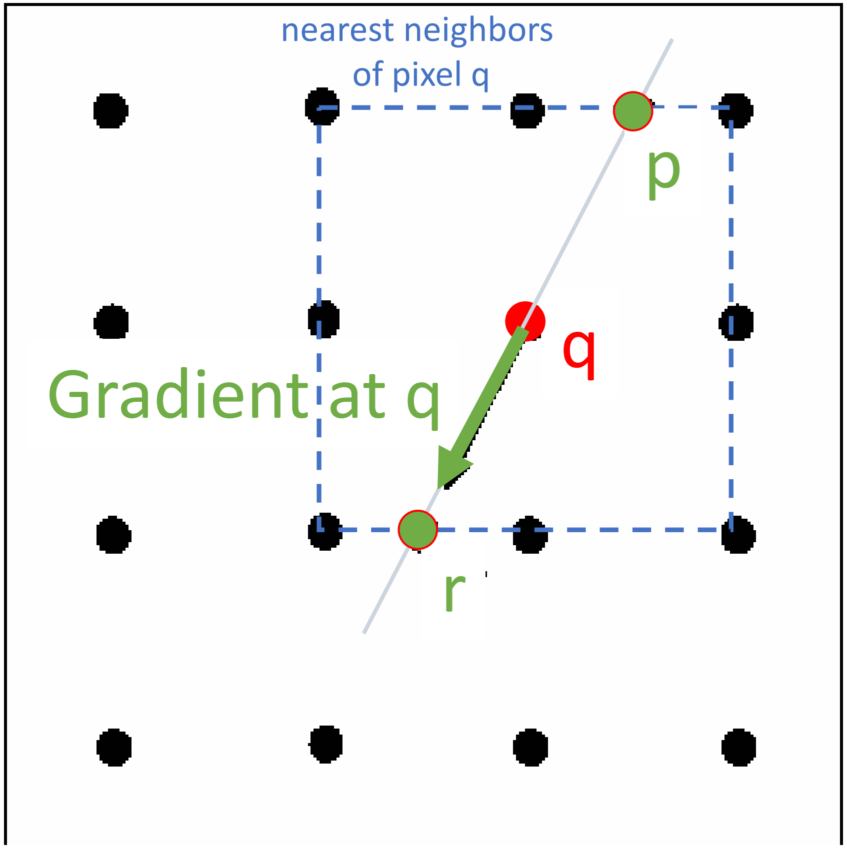
- Typical application where image gradients are used is *image edge* detection
  - find points with large image gradients

“Lena’s image”

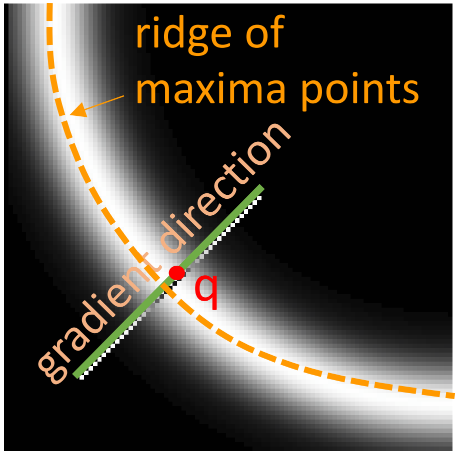
gradient magnitudes
$||\nabla f||$

thresholded
gradient magnitudes

Edge *thinning* via *non-maximum suppression*



At any given point q we have a maximum if the value $\|\nabla f\|$ at q is larger than those at both p and at r. (interpolate to get these values).



gradient magnitudes

Image gradient and edges

- Typical application where image gradients are used is *image edge* detection
  - find points with large image gradients

"Lena's image"

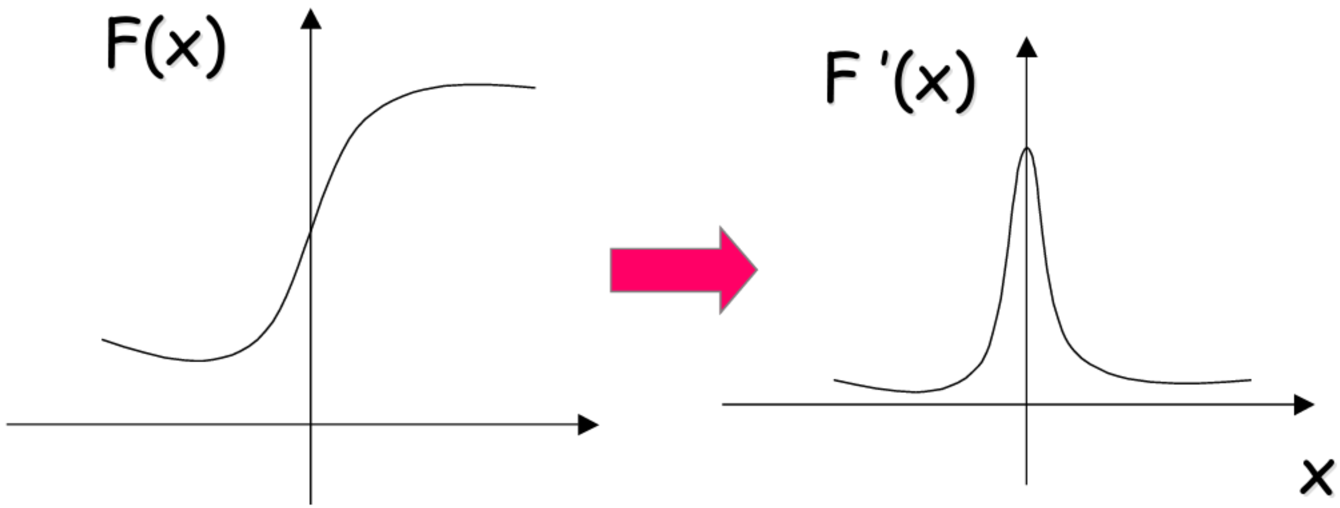
gradient magnitudes
$||\nabla f||$

"edge features"

Canny edge detector
(non-maxima suppression + adaptive thresholding)

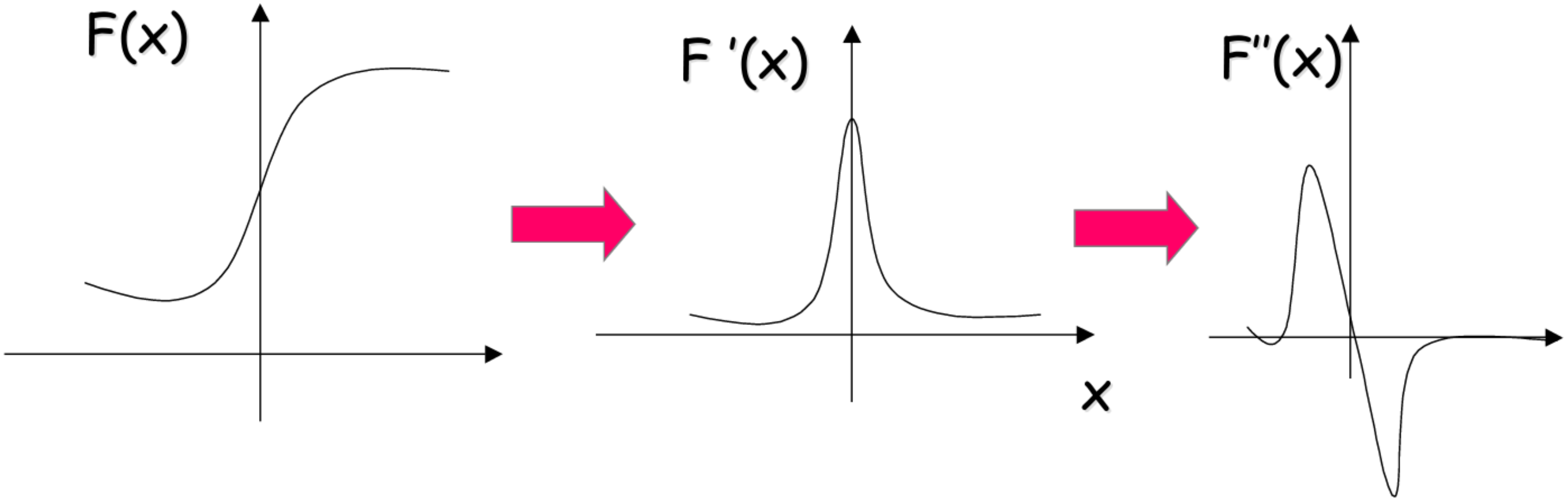
First Derivative Filter

- Sharp changes in gray level of the input image correspond to “peaks or valleys” of the first-derivative of the input signal.



Second Derivative Filter

- Peaks or valleys of the first-derivative of the input signal, correspond to “zero-crossings” of the second-derivative of the input signal.



Numerical Derivative

- Taylor series expansion

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + O(h^4)$$

add

$$+ \left[f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + O(h^4) \right]$$

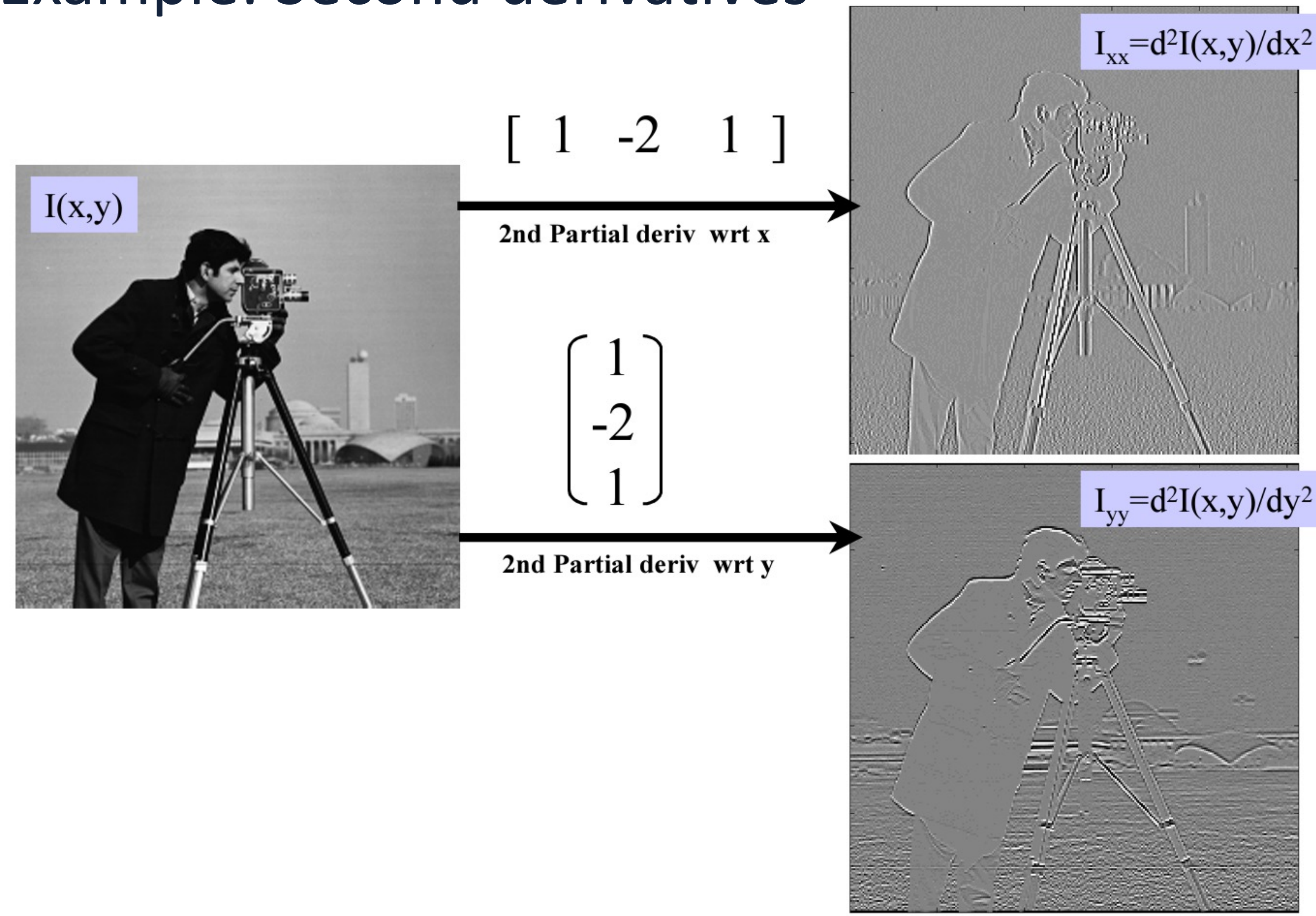
$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4)$$

$$\frac{f(x-h) - 2f(x) + f(x+h)}{h^2} = f''(x) + O(h^2)$$

1	-2	1

Central difference approx to second derivative

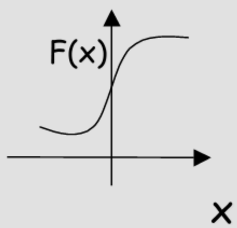
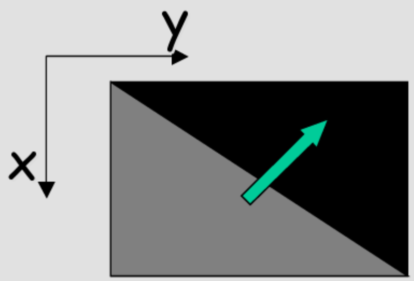
Example: Second derivatives



Finding zero-crossings

- An alternative approx to finding edges as peaks in first deriv is to find zero-crossings in second deriv.
- In 1D, convolve with [1 -2 1] and look for pixels where response is (nearly) zero?
- Problem: when first deriv is zero, so is second. i.e. the filter [1 -2   1] also produces zero when convolved with regions of constant intensity.
- So, in 1D, convolve with [1 -2 1] and look for pixels where response is nearly zero AND magnitude of first derivative is "large enough".

Edge detection summary

	1D	2D
step edge	$I(x)$ 	$I(x,y)$ 
1st deriv	$\left\lvert \frac{dI(x)}{dx} \right\rvert > Th$	$\lvert\nabla I(x,y)\rvert = (I_x{}^2(x,y) + I_y{}^2(x,y))^{1/2} > Th$ $\tan\theta = I_x(x,y) / I_y(x,y)$
2nd deriv	$\frac{d^2I(x)}{dx^2} = 0$	$\nabla^2 I(x,y) = I_{xx}(x,y) + I_{yy}(x,y) = 0$ Laplacian

Laplacian filter

$$I_{xx} + I_{yy} = \left(\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \right) * I$$

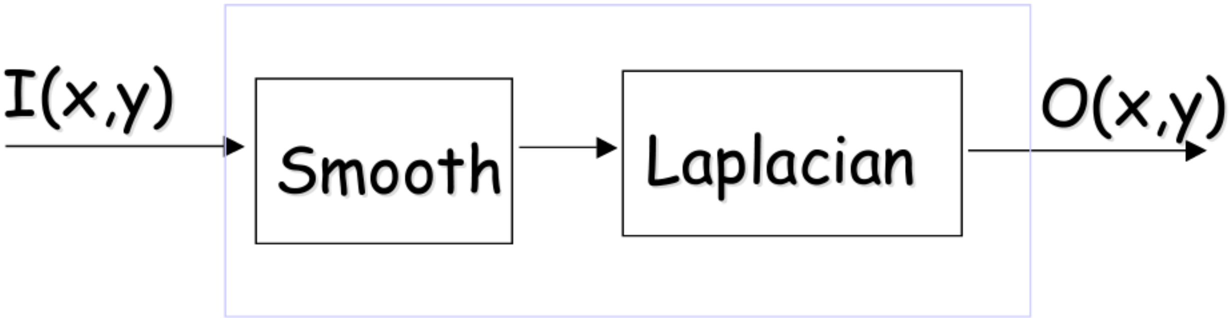
$$= \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\text{Laplacian filter } \nabla^2 I(x,y)} * I$$

Example: Laplacian filter

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * I$$

More about Laplacian filter

- Sum of second-order derivatives
- Very sensitive to noise
- It is always combined with a smoothing operation



Laplacian of Gaussian (LoG) filter

- First, smooth image (Gaussian filtering)
- Second, find zero-crossings (Laplacian operator)

$$\underbrace{\nabla^2 (f(x,y) \otimes G(x,y))}_{\text{Laplacian of Gaussian-filtered image}} = \underbrace{\nabla^2 G(x,y) \otimes f(x,y)}_{\text{Laplacian of Gaussian (LoG)-filtered image}}$$

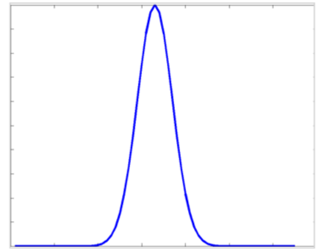
Laplacian of Gaussian-filtered image

Laplacian of Gaussian (LoG) -filtered image

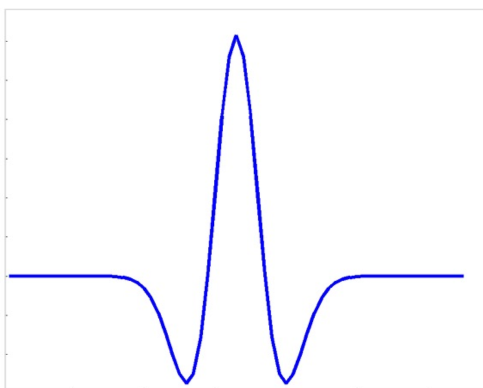
Do you see the distinction?

Second derivatives of Gaussian

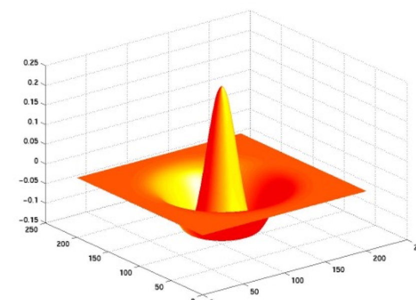
$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$



$$g''(x) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right)e^{-\frac{x^2}{2\sigma^2}}$$



2D analog



LoG "Mexican Hat"