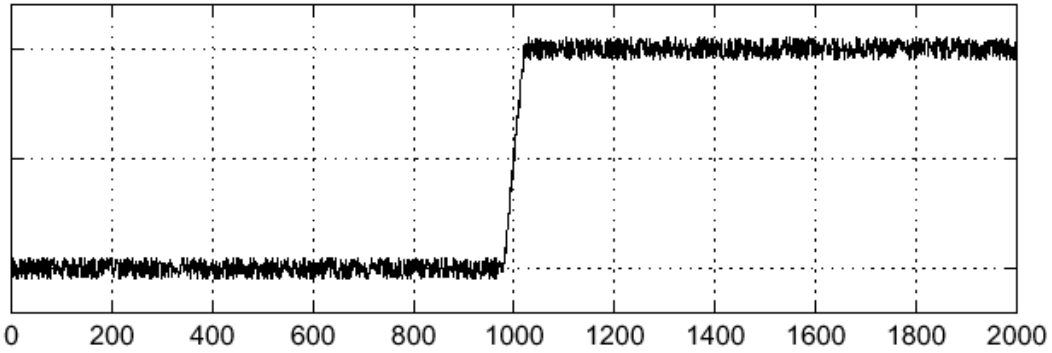
Three Classes of Image Processing Operations by Direct Manipulation of Gray Levels

- Any image-processing operation transforms the gray values of the pixels.
- Image-processing operations may be divided into three classes based on the information required to perform the transformation.
- From the simplest to the most complex, they are as follows:
  - **Point operations : chap.4**
  - **Neighborhood processing (spatial filter) : chap.5**
  - **Geometrical transforms: chap.6**

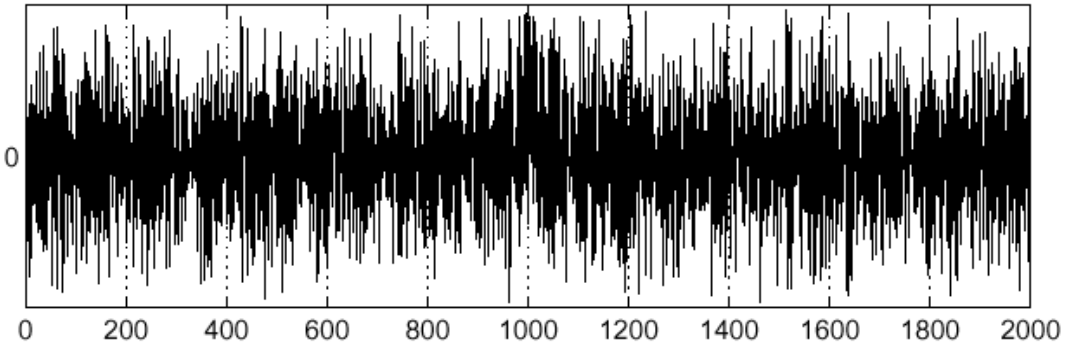
Combined Filtering: Effects of Noise

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

$f(x)$



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

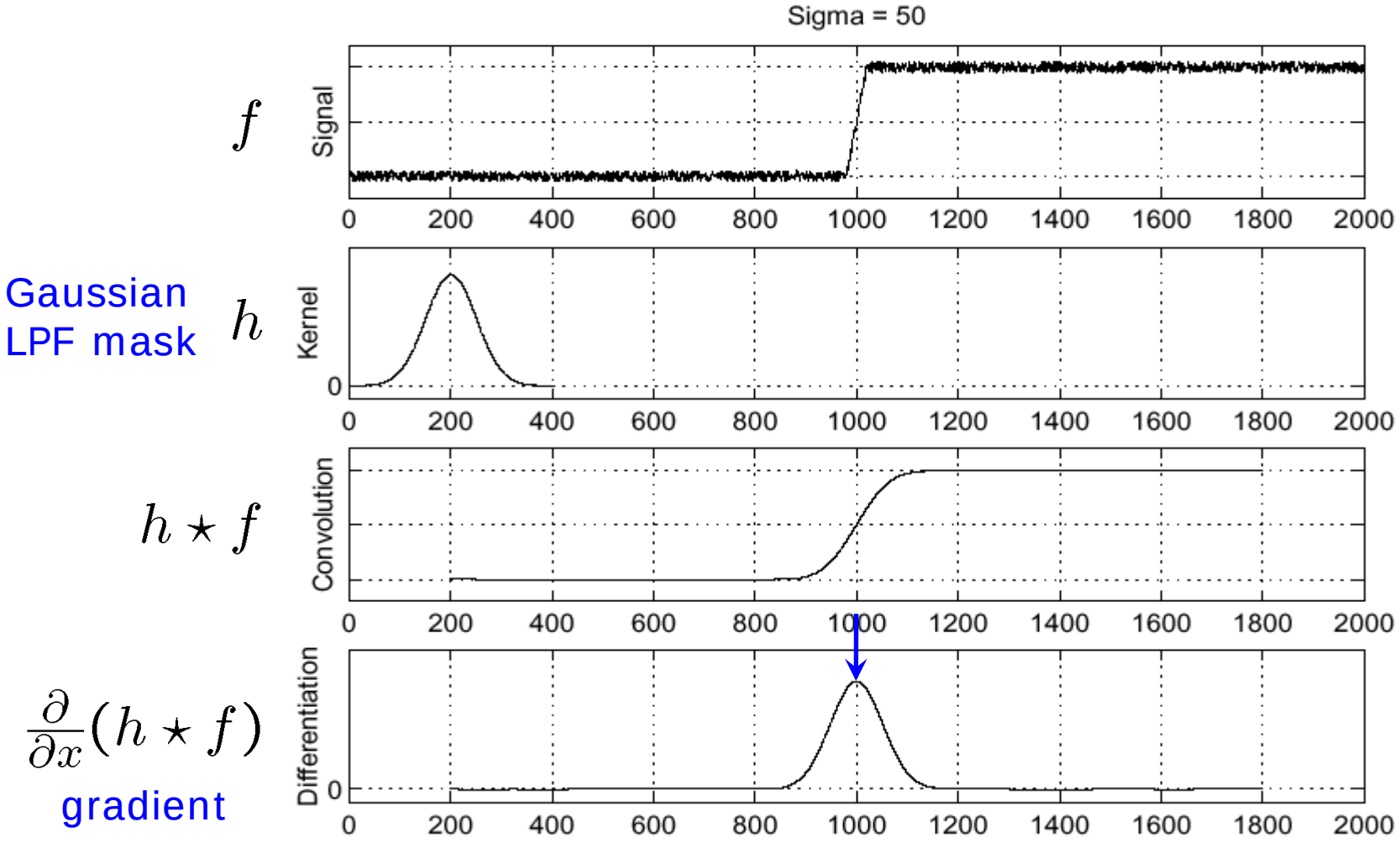


Where is the edge??

Laplace operator may detect edges as well as noise.

courtesy of S. Narasimhan at CMU

Solution: Smooth(LPF) First



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

courtesy of S. Narasimhan at CMU

Derivative of Gaussian (DoG)

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

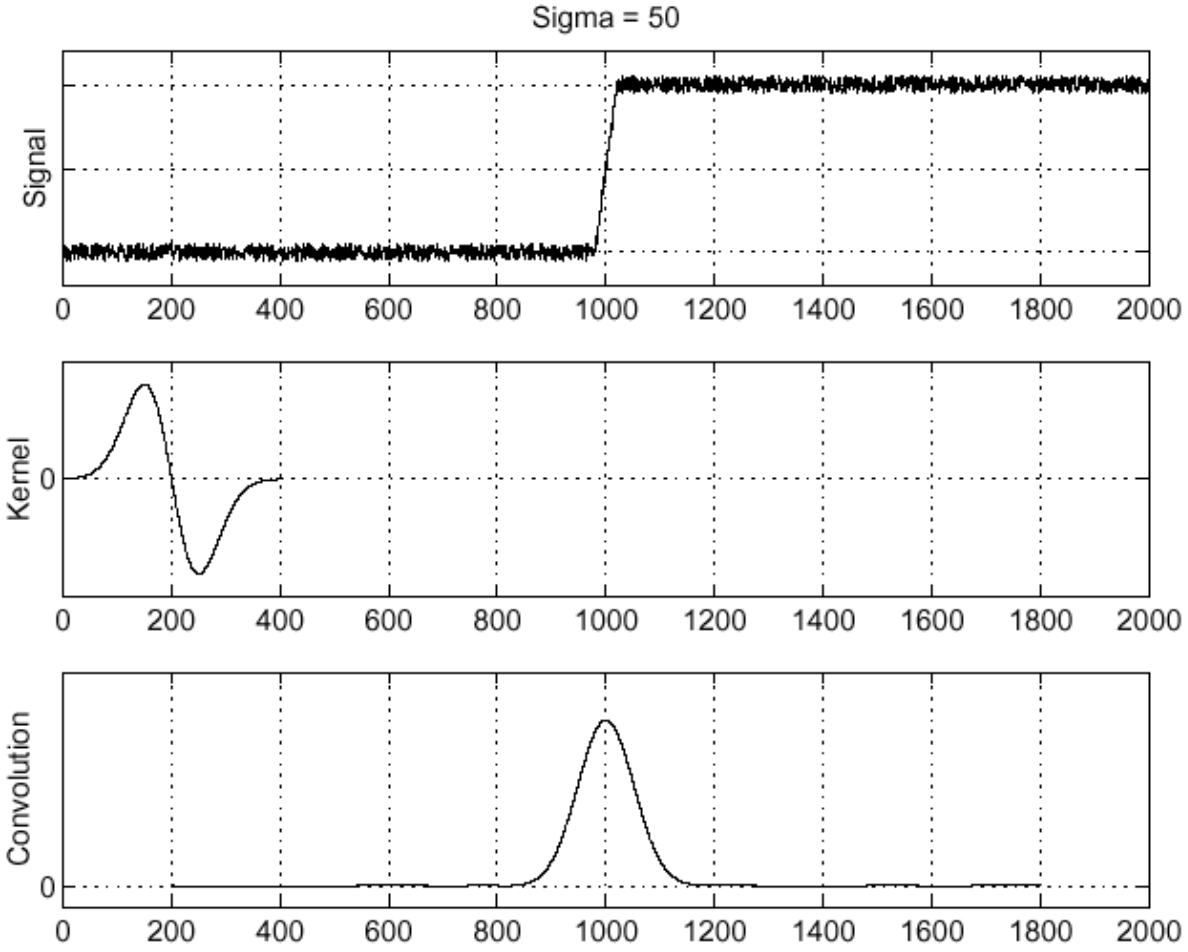
…saves us one operation.

f

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

Derivative of Gaussian

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



courtesy of S. Narasimhan at CMU

Laplacian of Gaussian (LoG)

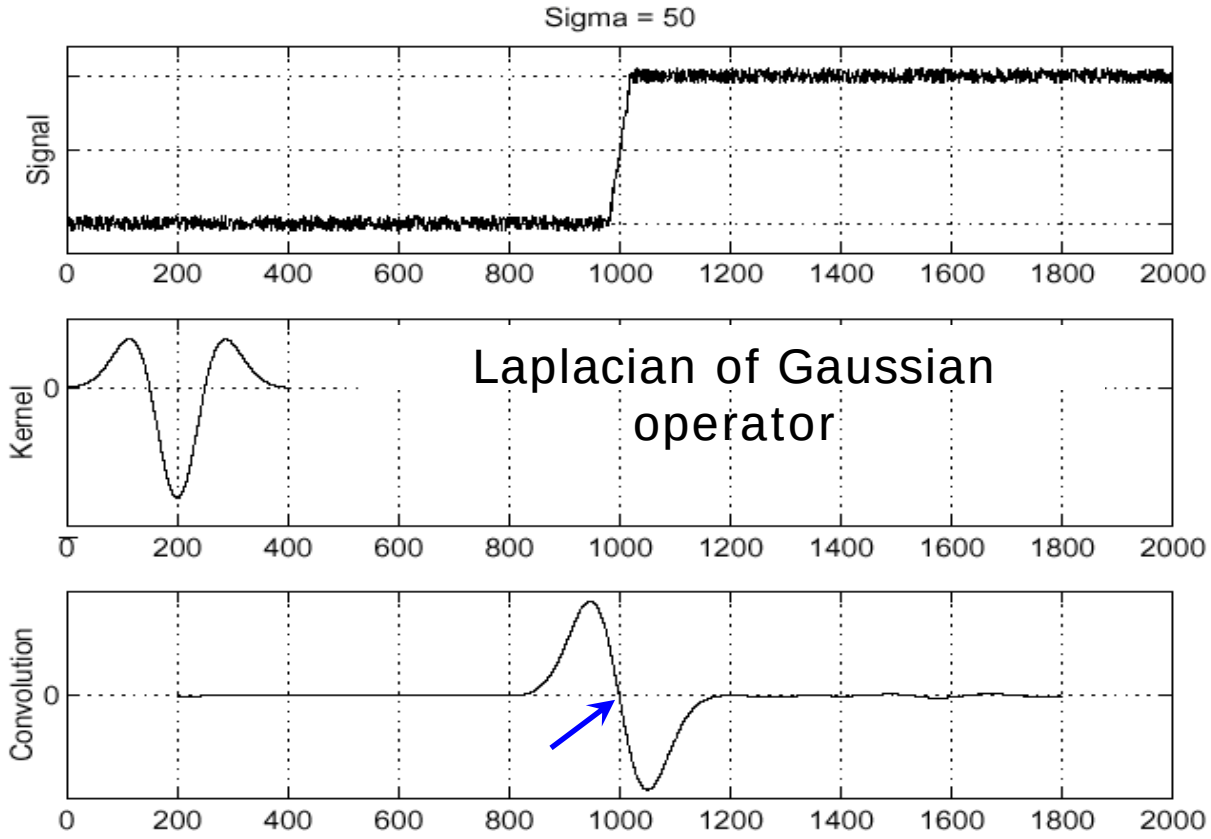
$$\frac{\partial^2}{\partial x^2}(h * f) = \left(\frac{\partial^2}{\partial x^2} h\right) * f$$

Laplacian of Gaussian

f

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

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

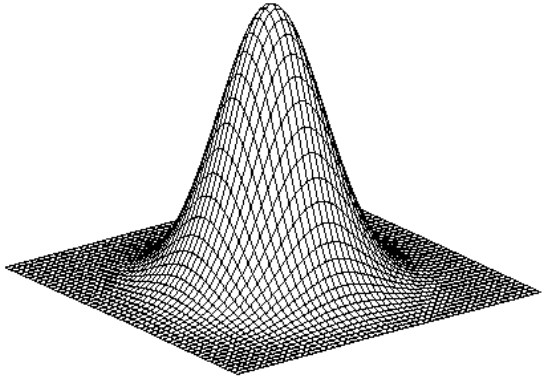


Where is the edge?

Zero-crossings of bottom graph !

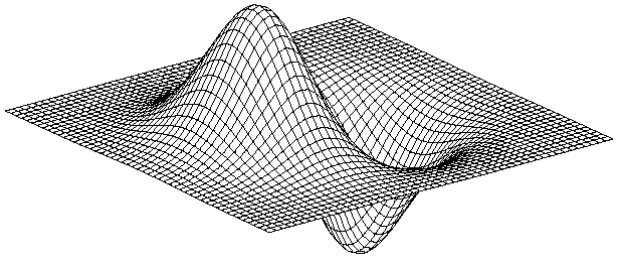
courtesy of S. Narasimhan at CMU

2D Gaussian Edge Operators



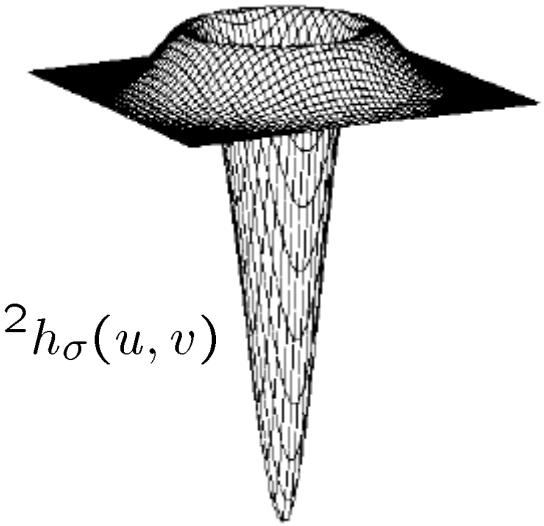
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

Gaussian



$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Derivative of Gaussian (DoG)



$$\nabla^2 h_\sigma(u,v)$$

Laplacian of Gaussian

Mexican Hat (Sombrero)

- ∇^2 is the **Laplacian** operator:

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

High-Pass Filters in Matlab

```
>> f=fspecial('laplacian')

f =

    0.1667    0.6667    0.1667
    0.6667   -3.3333    0.6667
    0.1667    0.6667    0.1667

>> cf=filter2(f,c);
>> imshow(cf/100)
>> f1=fspecial('log')

f1 =

    0.0448    0.0468    0.0564    0.0468    0.0448
    0.0468    0.3167    0.7146    0.3167    0.0468
    0.0564    0.7146   -4.9048    0.7146    0.0564
    0.0468    0.3167    0.7146    0.3167    0.0468
    0.0448    0.0468    0.0564    0.0468    0.0448

>> cf1=filter2(f1,c);
>> figure,imshow(cf1/100)
```

FIGURE 5.5

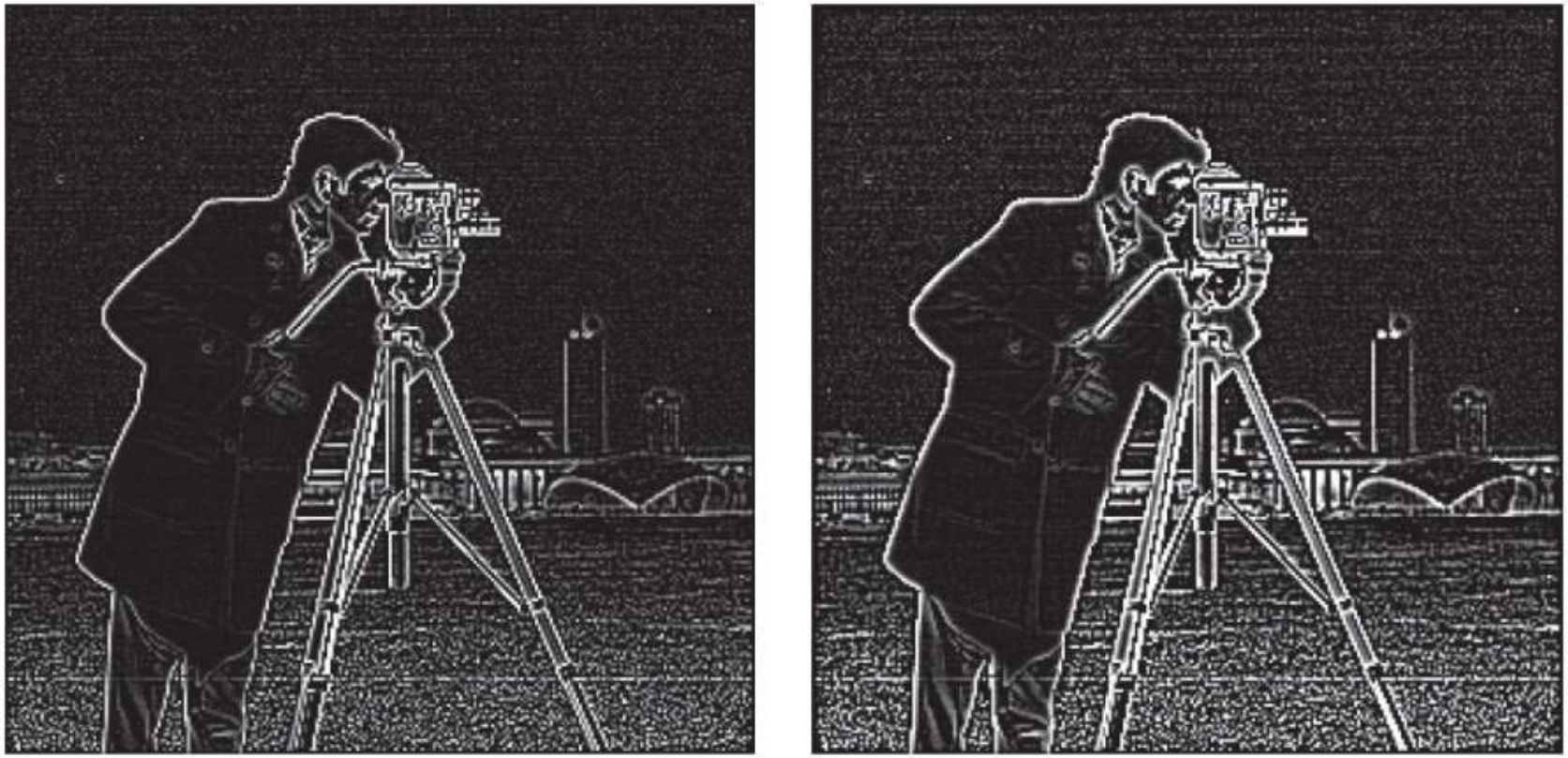


FIGURE 5.5 *High-pass filtering. (a) Laplacian filter. (b) Laplacian of Gaussian (log) filtering.*

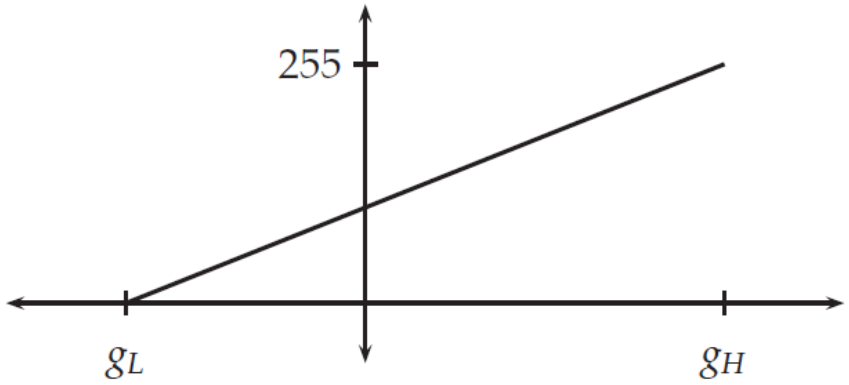
VALUES OUTSIDE THE RANGE 0–255

✓ **Make negative values positive**

method 1. Clip values

$$y = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \le x \le 255 \\ 255 & \text{if } x > 255 \end{cases}$$

method 2. 0-255 Scaling transformation (uint8)



$$y = 255\frac{x - g_L}{g_H - g_L}$$

VALUES OUTSIDE THE RANGE 0–255

```
>> f2=[1 -2 1;-2 4 -2;1 -2 1];
>> cf2=filter2(f2,c);
```

```
>> figure,imshow(mat2gray(cf2));
```

0-1 Scaling transformation (double)

```
>> maxcf2=max(cf2(:));
>> mincf2=min(cf2(:));
>> cf2g=(cf2-mincf2)/(maxcf2-mncf2);
```

```
>> figure,imshow(cf2/60)
```

FIGURE 5.6

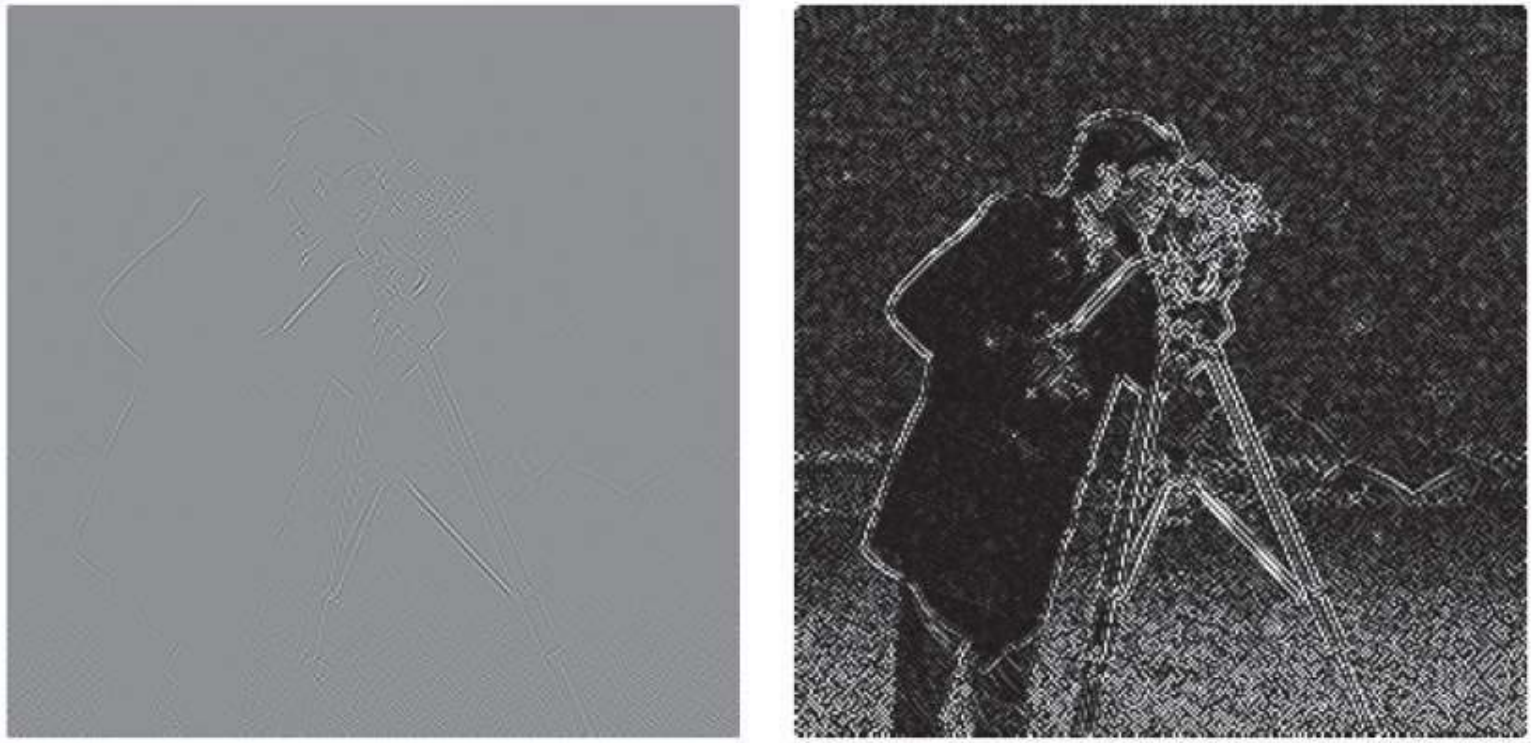
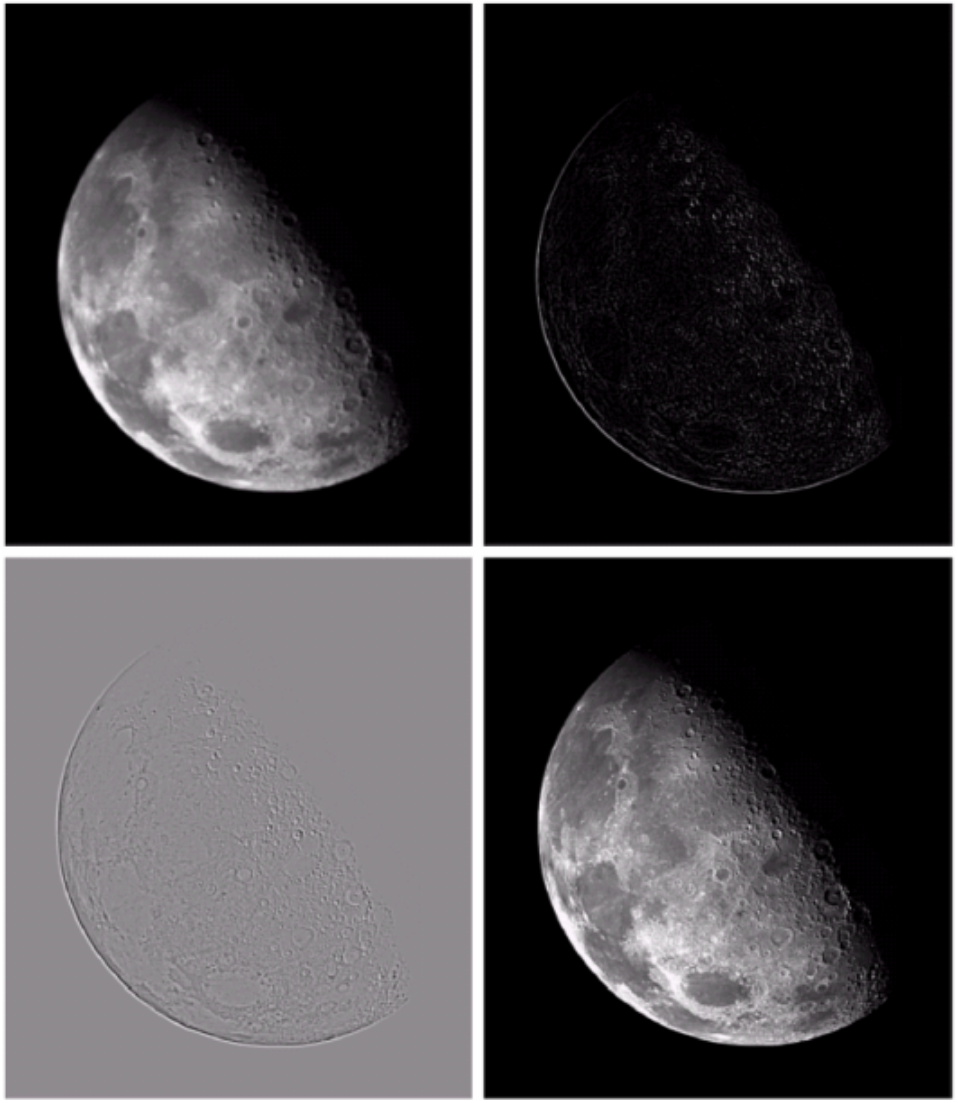


FIGURE 5.6 *Using a high-pass filter and displaying the result. (a) Using* `mat2gray`*. (b) Dividing by a constant.*

HPF for Image Enhancement

(a) Moon



(b) After Laplacian HPF (2^{nd} derivative) : Neg. value was set to 0.

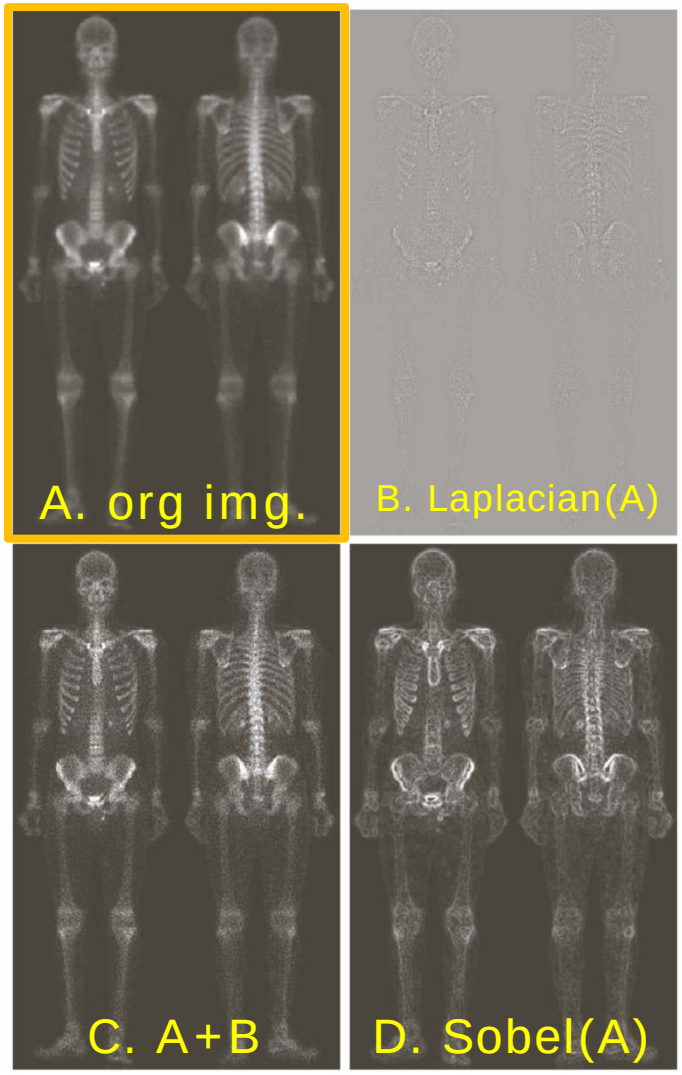
(c) Intensity scaling of (b)

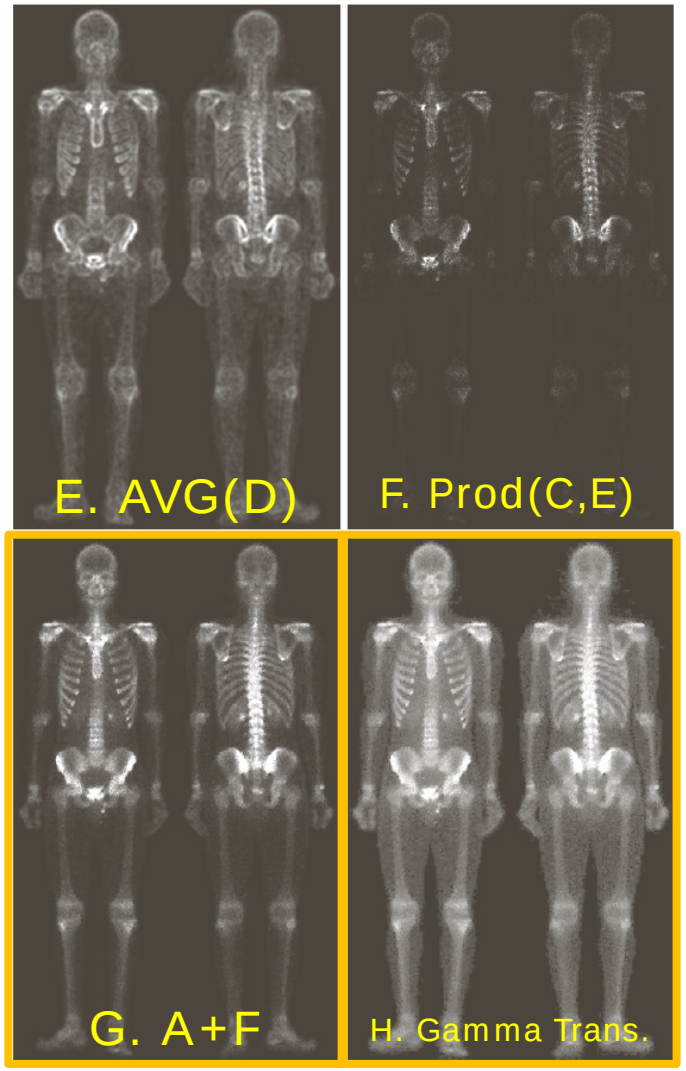
(d) = (a)-(b) with negative center in the mask

Combined Spatial Enhancement : nuclear whole body bone scan

a b
c d

FIGURE 3.43
(a) Image of whole body bone scan.
(b) Laplacian of (a). (c) Sharpened image obtained by adding (a) and (b). (d) Sobel gradient of (a).





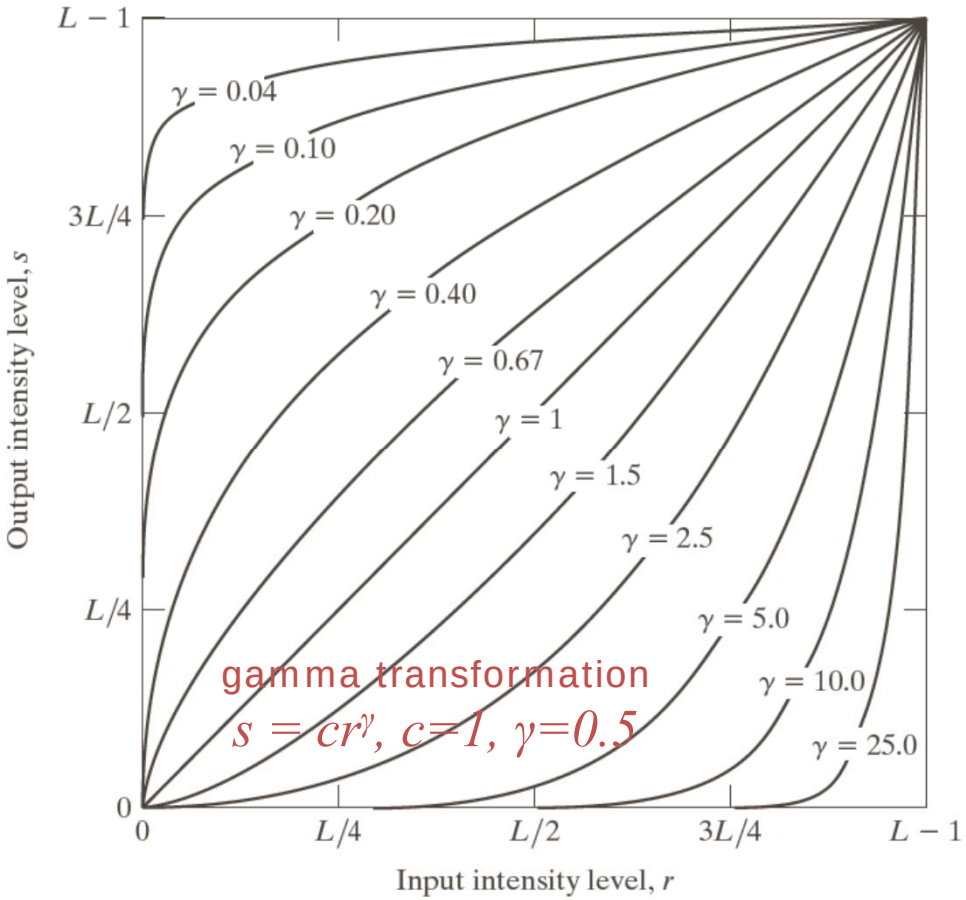
e f
g h

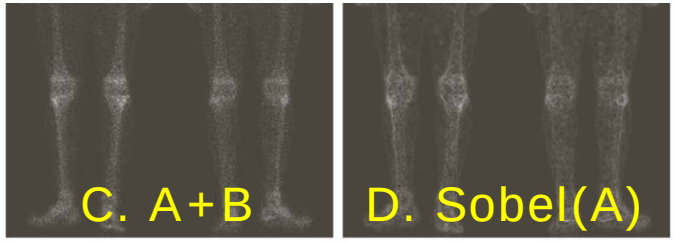
FIGURE 3.43
(Continued)
(e) Sobel image smoothed with a 5×5 averaging filter. (f) Mask image formed by the product of (c) and (e).
(g) Sharpened image obtained by the sum of (a) and (f). (h) Final result obtained by applying a power-law transformation to (g). Compare (g) and (h) with (a). (Original image courtesy of G.E. Medical Systems.)

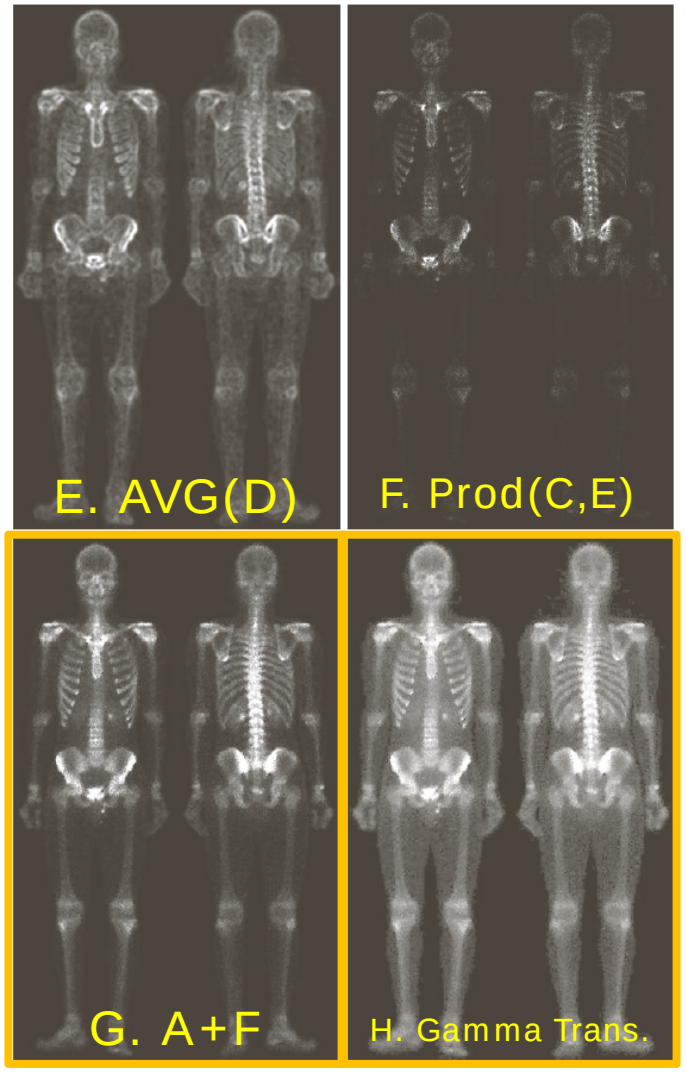
Sharpen the image and bring out more of the skeletal detail.

1) Laplacian to highlight fine detail 2) gradient to enhance the prominent edges 3) increase the dynamic range of intensity (3.2.3 gamma trans.)

Combined Spatial Enhancement
body bone scan







e f
g h

FIGURE 3.43 *(Continued)* (e) Sobel image smoothed with a 5 × 5 averaging filter. (f) Mask image formed by the product of (c) and (e). (g) Sharpened image obtained by the sum of (a) and (f). (h) Final result obtained by applying a power-law transformation to (g). Compare (g) and (h) with (a). (Original image courtesy of G.E. Medical Systems.)

Sharpen the image and bring out more of the skeletal detail.

1) Laplacian to highlight fine detail 2) gradient to enhance the prominent edges 3) increase the dynamic range of intensity (3.2.3 gamma trans.)

5.6 Edge Sharpening

- **Unsharp Masking**

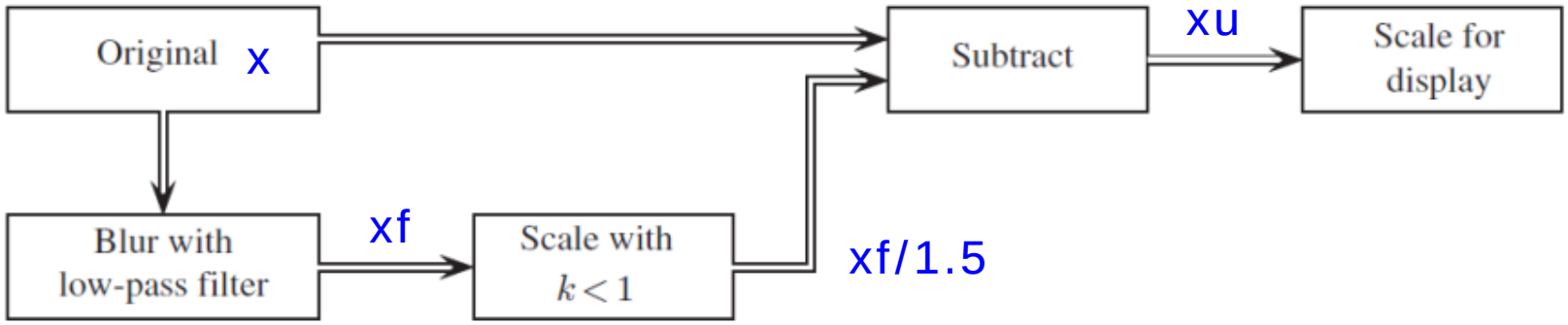


FIGURE 5.10 *Schema for unsharp masking.*

```
>> f=fspecial('average');
>> xf=filter2(f,x);
>> xu=double(x)-xf/1.5
>> imshow(xu/70)
```

Unsharp Masking

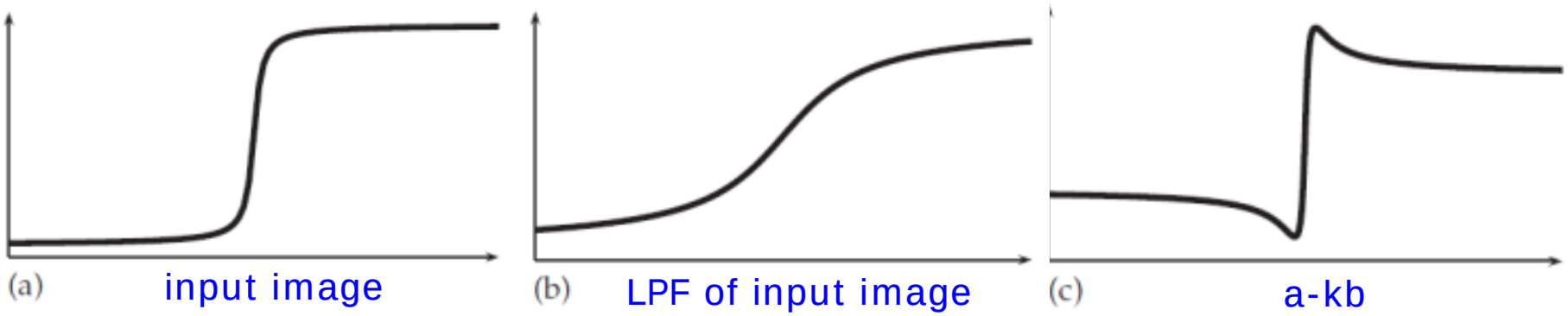


FIGURE 5.12 *Unsharp masking. (a) Pixel values over an edge. (b) The edge blurred. (c) (a) – k(b).*

input image

a-kb

Unsharp Masking

- The `unsharp` option of `fspecial` produces such filters

$$\frac{1}{\alpha+1}\begin{bmatrix} -\alpha & \alpha-1 & -\alpha \\ \alpha-1 & \alpha+5 & \alpha-1 \\ -\alpha & \alpha-1 & -\alpha \end{bmatrix}$$

```
>> p=imread('pelicans.tif');
>> u=fspecial('unsharp',0.5);   α = 0.5
>> pu=filter2(u,p);
>> imshow(p),figure,imshow(pu/255)
```

(a)

(b)

High-Boost Filtering

- Allied to unsharp masking filters are the **high-boost** filters

$$\text{high boost} = A(\text{original}) - (\text{low pass})$$

✓ where A is an amplification factor.

✓ If $A = 1$, then the high-boost filter becomes an ordinary high-pass filter.

High-Boost Filtering

```
>> id=[0 0 0;0 1 0;0 0 0];
>> f=fspecial('average');
>> hb1=3*id-2*f

hb1 =

   -0.2222   -0.2222   -0.2222
   -0.2222    2.7778   -0.2222
   -0.2222   -0.2222   -0.2222

>> hb2=1.25*id-0.25*f

hb2 =

   -0.0278   -0.0278   -0.0278
   -0.0278    1.2222   -0.0278
   -0.0278   -0.0278   -0.0278
```

```
>> x1=filter2(hb1, x);
>> imshow(x1/255)
```

```
>> x2=filter2(hb2, x);
>> imshow(x2/255)
```

Nonlinear Filters

- Maximum and minimum filter

```
>> cmax=nlfilter(c,[3,3],'max(x(:))');
```

```
>> cmin=nlfilter(c,[3,3],'min(x(:))');
```

FIGURE 5.15 *Using nonlinear filters. (a) Using a maximum filter. (b) Using a minimum filter.*

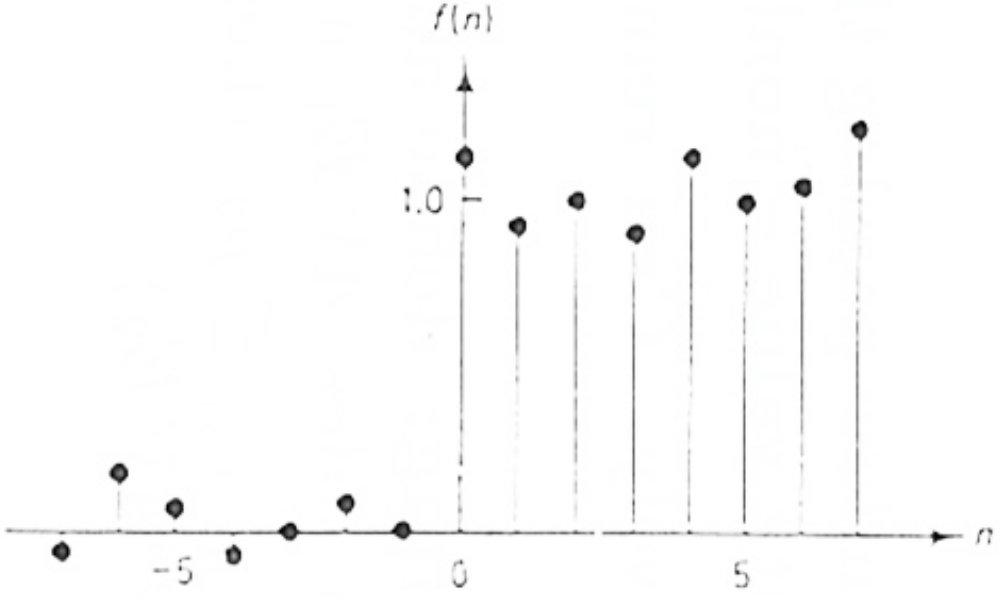
Non-linear LPF: Median Filter

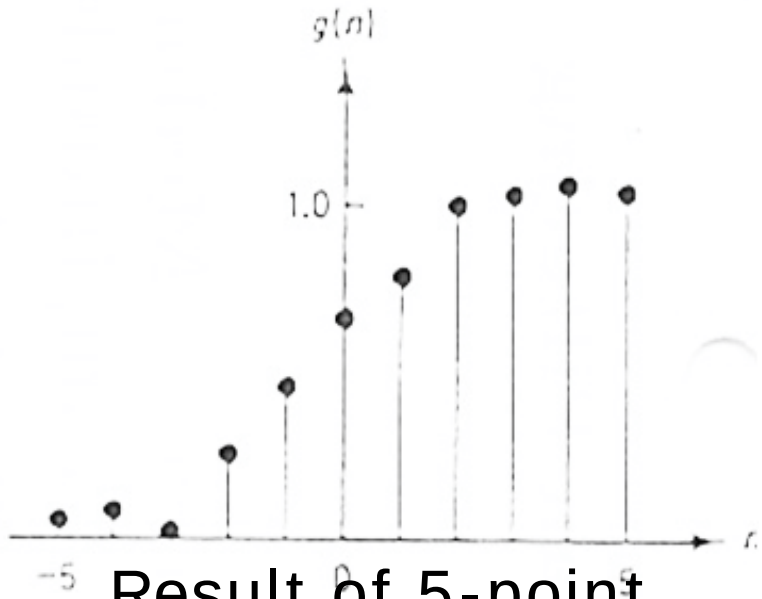
- LPF : blurred while noise is removed to some extent

  -> solution : median filtering

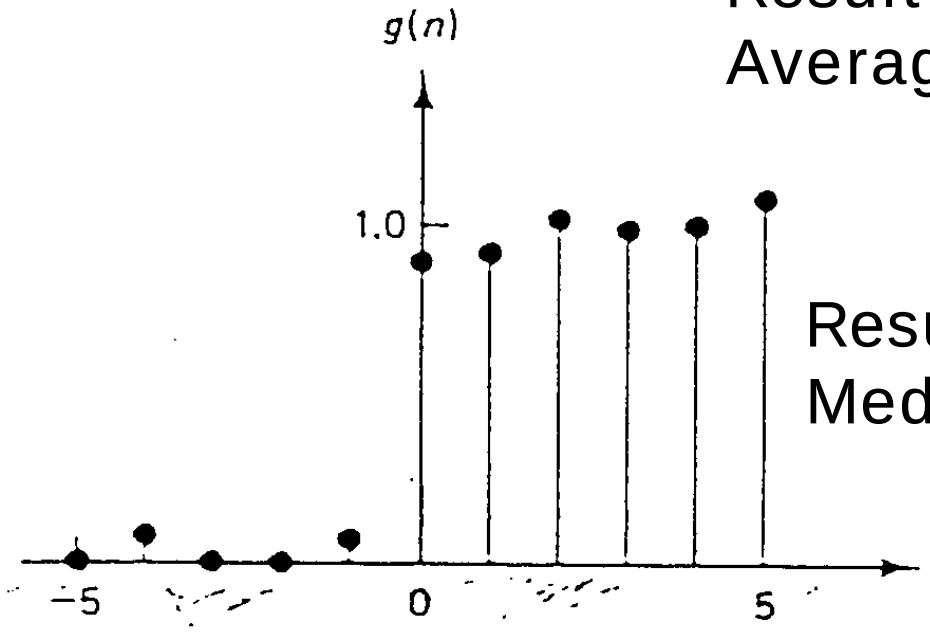
  -> avoids blurring in noise removing
- ex1: Average and median value of [1, 2, 3, 4, 100] ?
- ex2: median value of [10,20,20,20,15,20,20,25,100]?
- nonlinear process
- especially useful for reducing **impulse** or **salt-and-pepper noise**.
- **Median filtering**: with a sliding window passing through the image, choose the **median** (middle) intensity value -> nonlinear.
- It preserves **edges**.

Median Filters Preserve Edges



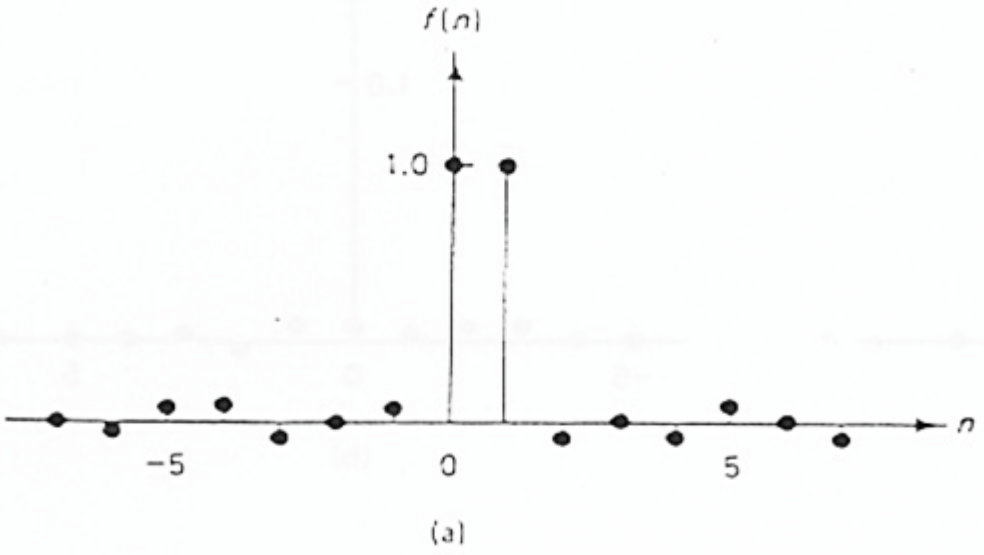


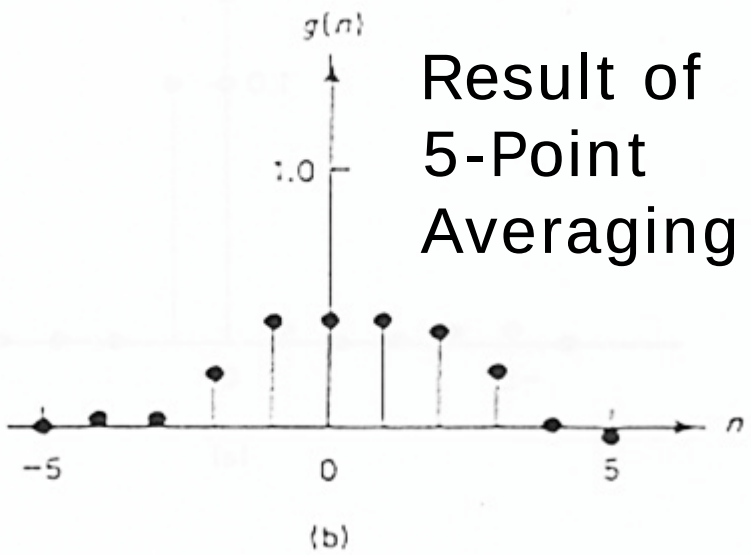
Result of 5-point Average Filter



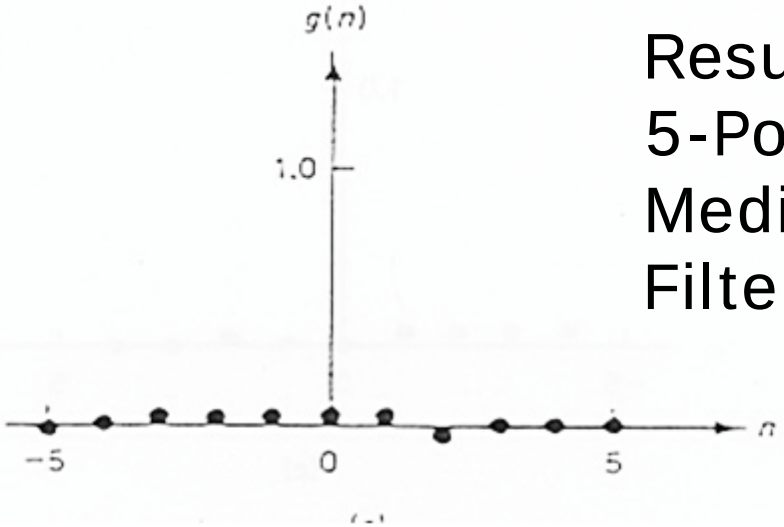
Result of 5-Point Median Filter

Median Filters Remove Impulse Noise



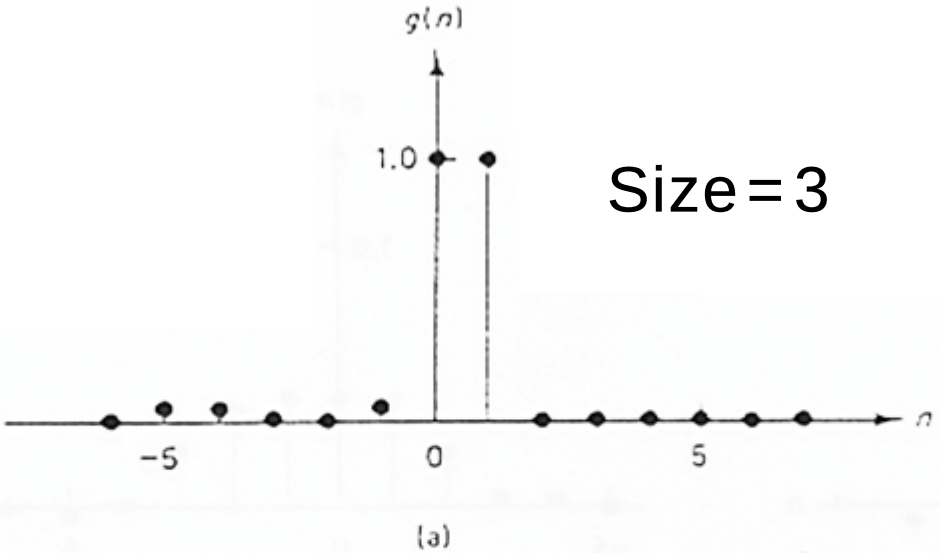


Result of 5-Point Averaging

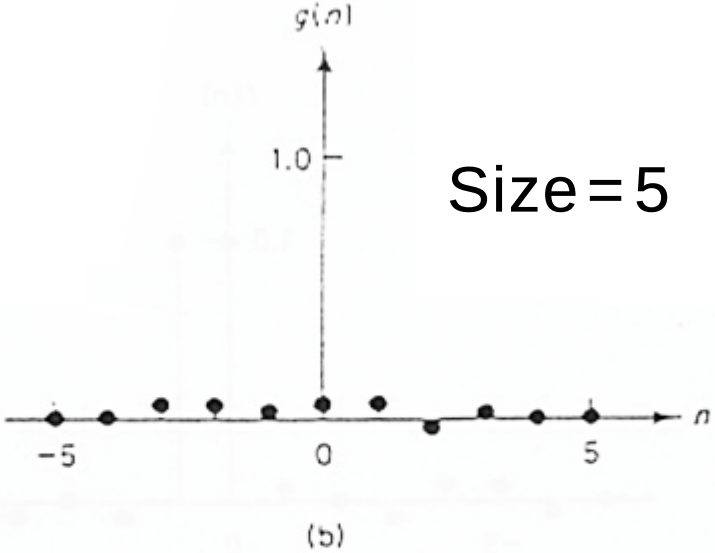


Result of 5-Point Median Filtering

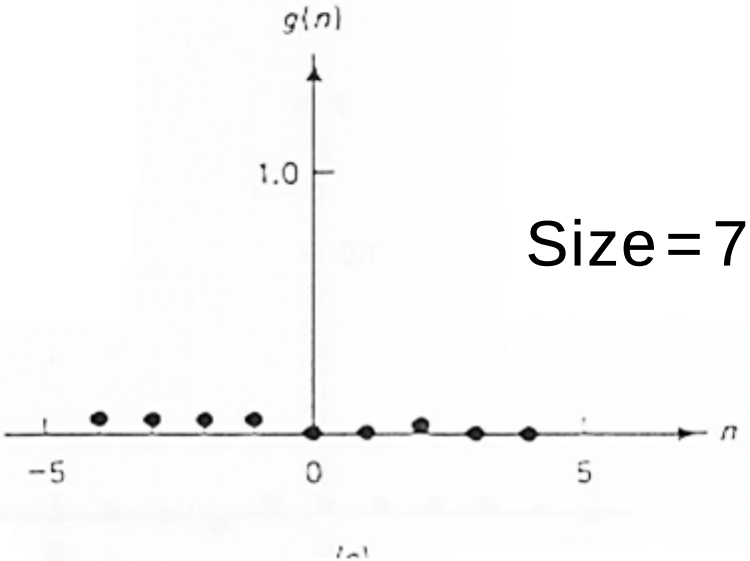
Effect of Median Filter Window Sizes



Size = 3



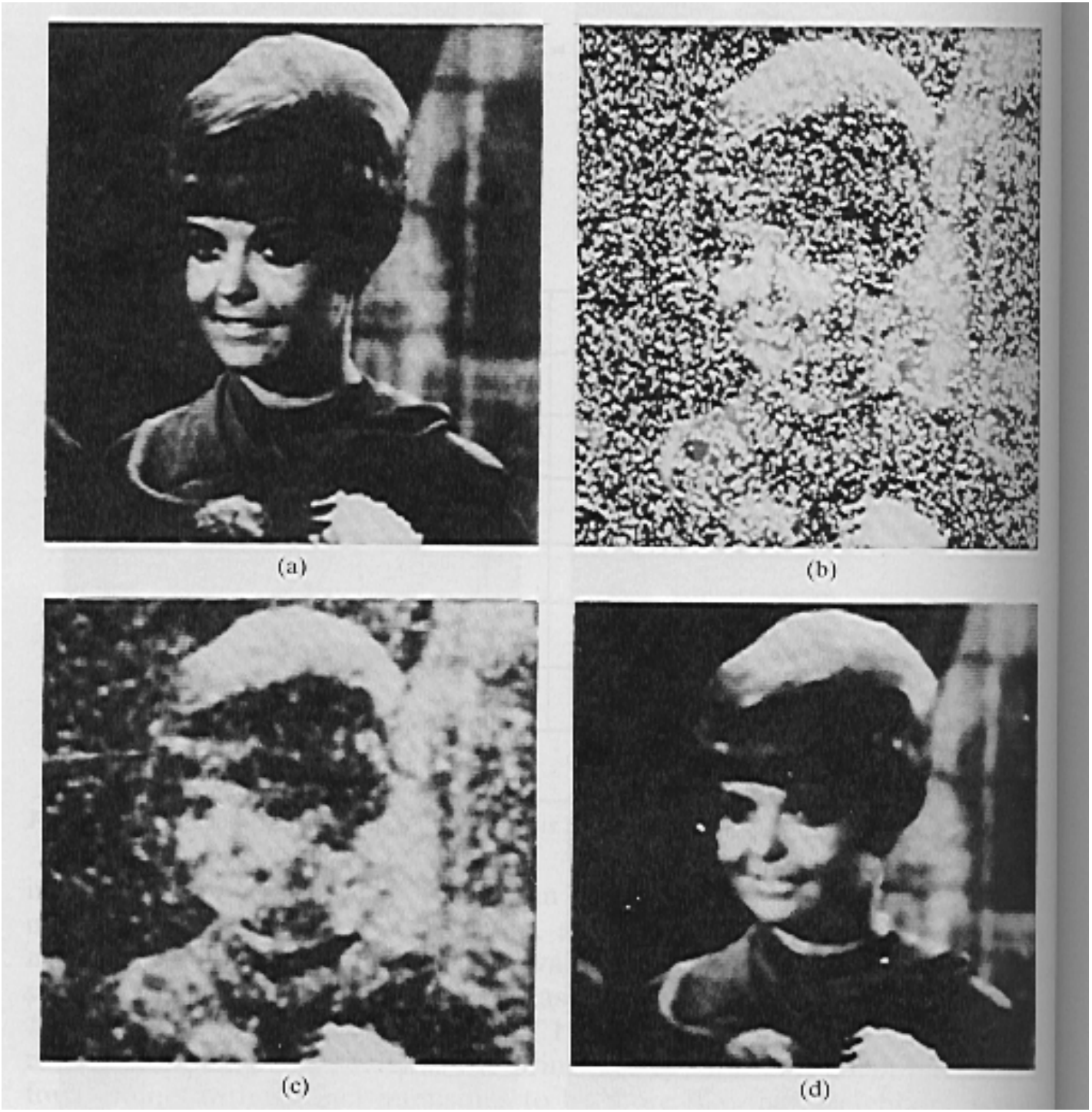
Size = 5



Size = 7

Removing Impulse Noise

(a) original
(b) impulse noise
(c) 5x5 averaging
(d) 5x5 median

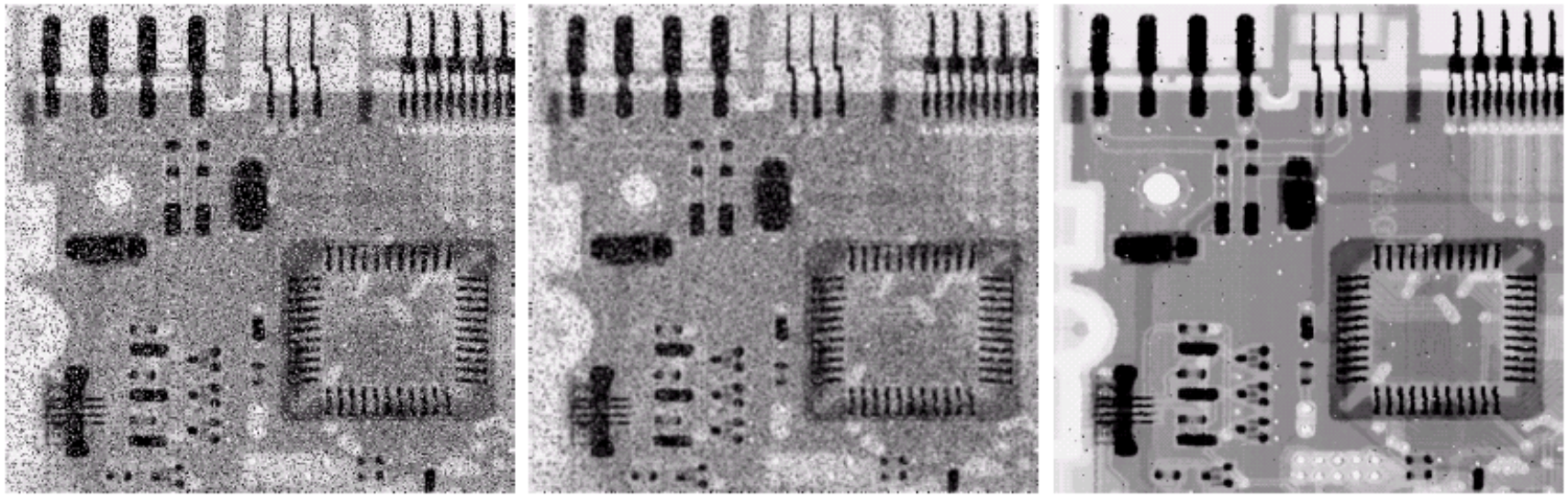


Removing Impulse Noise

x-ray image salt-and-pepper noise

3x3 averaging

3x3 median



a b c

FIGURE 3.37 (a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a 3×3 averaging mask. (c) Noise reduction with a 3×3 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

Region of Interest Processing

```
>> ig=imread('iguana.tif');
```

```
>> roi=roipoly(ig,[406 600 600 406],[58 58 231 231]);
```

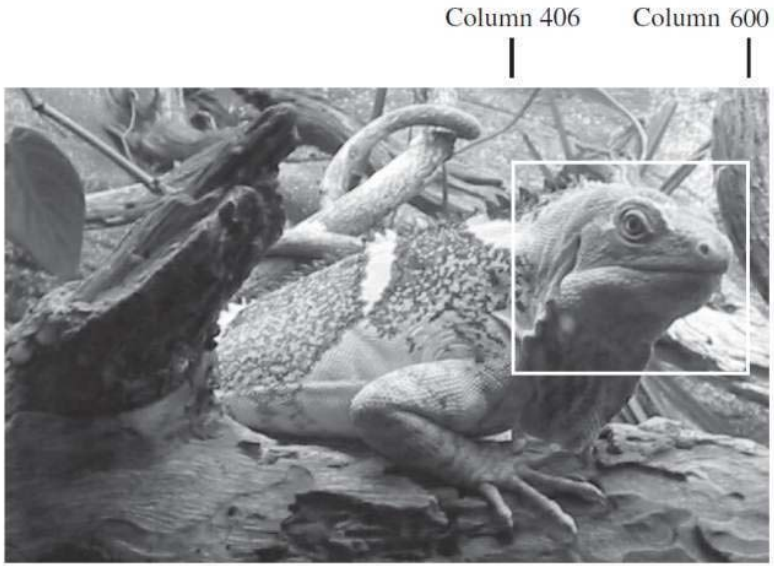


FIGURE 5.16 *An image with a ROI.*

URE 5.17 *The mask corresponding to the ROI defined in Figure 5.16.*

Regions of Interest in MATLAB

```
>> roi=roipoly(ig);
```

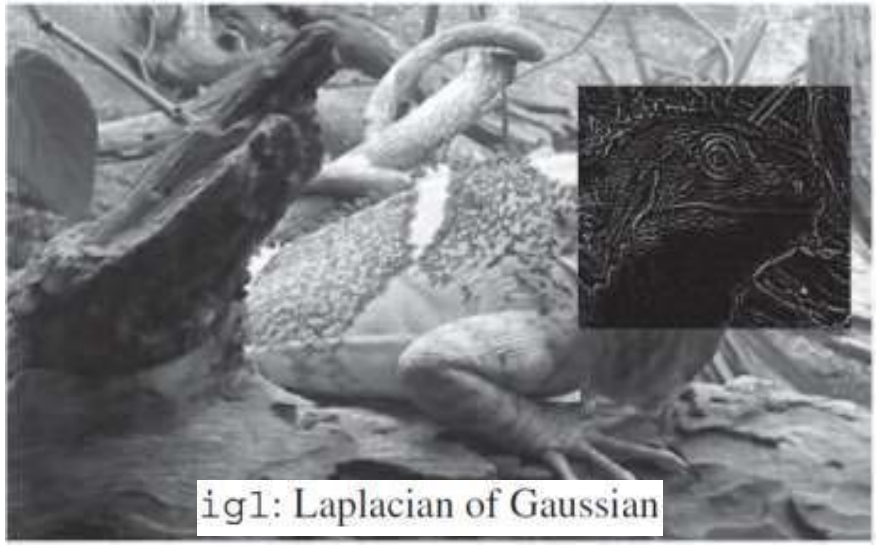
- This will bring up the iguana image if it isn't shown already.
- Vertices of the ROI can be selected with the mouse.

```
>> a=fspecial('average',[15,15]);
>> iga=roifilt2(a,ig,roi);
>> imshow(iga)
>> u=fspecial('unsharp');
>> igu=roifilt2(u,ig,roi);
>> figure,imshow(igu)
>> l=fspecial('log');
>> igl=roifilt2(l,ig,roi);
>> figure,imshow(igl)
```

ROI Processing: Results

iga: Average filtering

igu: Unsharp masking

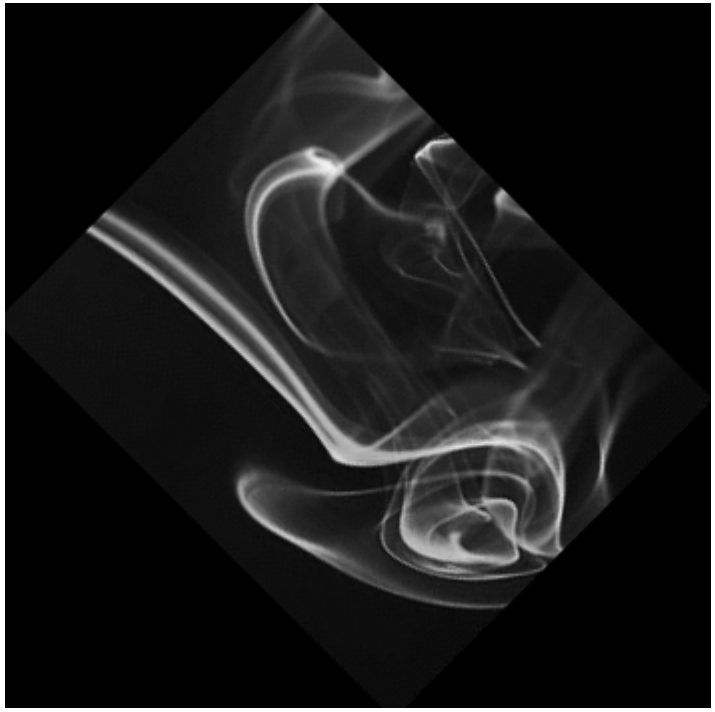


igl: Laplacian of Gaussian

FIGURE 5.18 *Examples of the use of* `roifilt2`.

Summary

- **Chap 4. Point Processing**
  - ✓ Histogram(Contrast) Stretching
  - ✓ Piecewise Contrast Stretching
  - ✓ Histogram Equalization & Specification
- **Chap 5. Neighborhood Processing**
  - ✓ Convolution for Spatial Filtering
  - ✓ Lowpass Filtering: Gaussian filter
  - ✓ Highpass Filtering: Laplacian, LoG filter
  - ✓ Edge Sharpening by Unsharp Masking
  - ✓ High Boost Filtering
  - ✓ Nonlinear Filtering: median, min, max



Chapter 6:

Image Geometry

6.1 Interpolation of Data

- Suppose we have a collection of four values that we wish to enlarge to eight

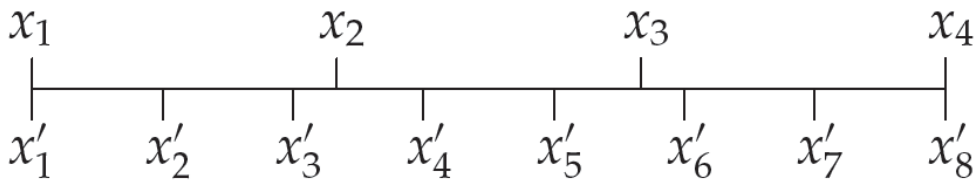


FIGURE 6.1 *Replacing four points with eight.*

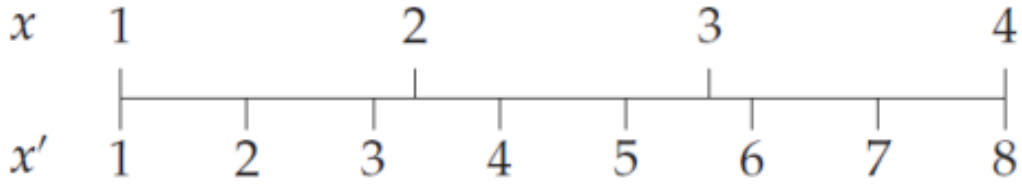


FIGURE 6.2 *Figure 6.1 slightly redrawn.*

6.1 Interpolation of Data

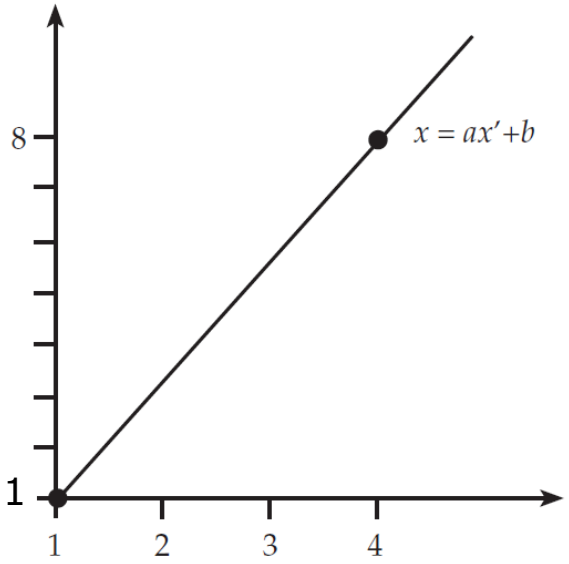


FIGURE 6.3 *Filled circles are the coincide points of x and x′.*

- The a and b of the linear function can be solved by

$$1 = a + b$$

$$4 = 8a + b$$

- Then we can obtain the linear function -> continuous

$$x = \frac{3}{7}x' + \frac{4}{7} \qquad x' = \frac{1}{3}(7x - 4),$$

$$x = \frac{1}{7}(3x' + 4)$$

6.1 Interpolation of Data

- In digital (discrete) signals, none of the x_i' points coincide exactly with an original x_j, except for the first and last.
- We have to estimate function values $f(x_i')$ based on the known values of nearby $f(x_j)$
- Such estimation of function values based on surrounding values is called **interpolation.**

Nearest-neighbor interpolation

$$f(x_i') = f(x_j)$$

x'_i : point to be interpolated x_j: original point closest to x'_i

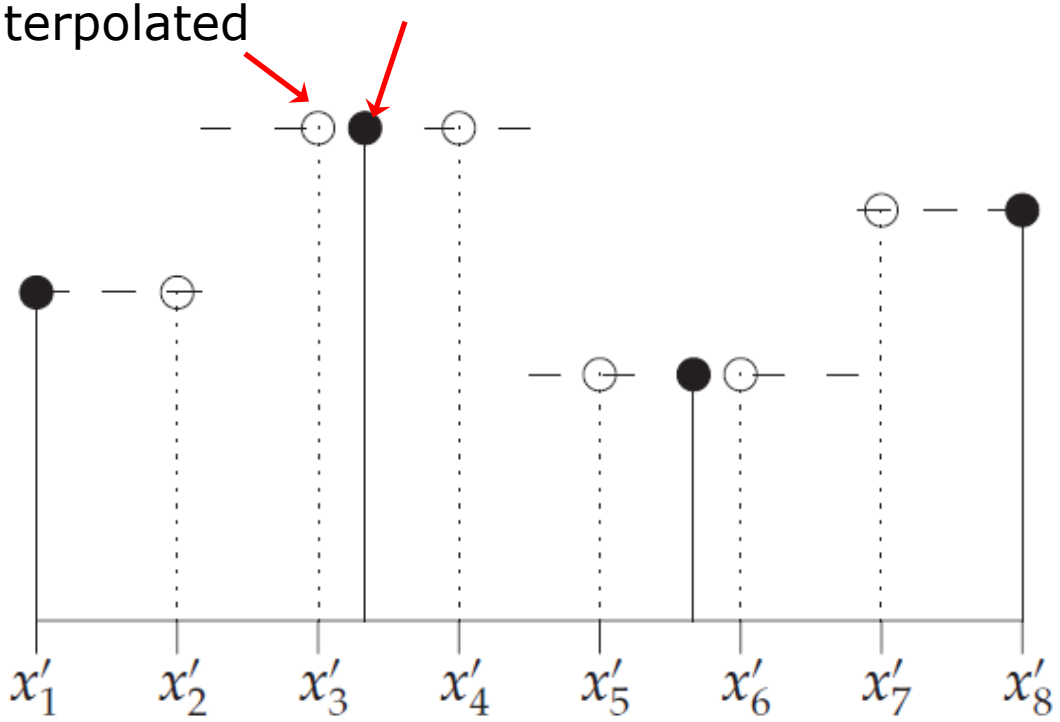


FIGURE 6.4 *Nearest-neighbor interpolation.*

Linear interpolation

using distance(λ) as a weight

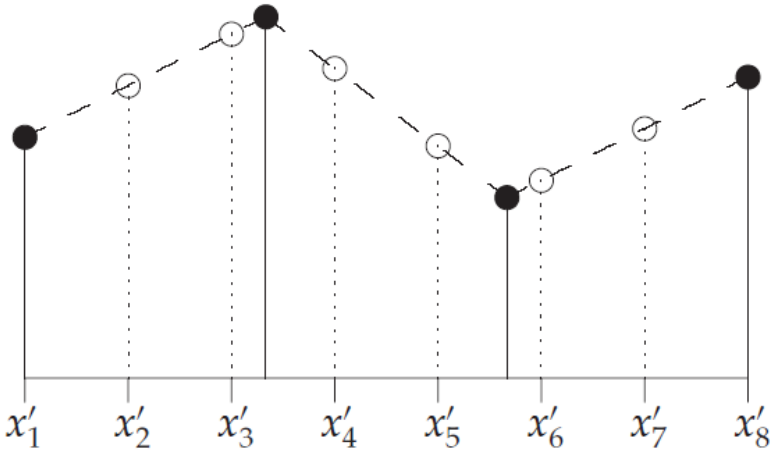


FIGURE 6.5 *Linear interpolation.*

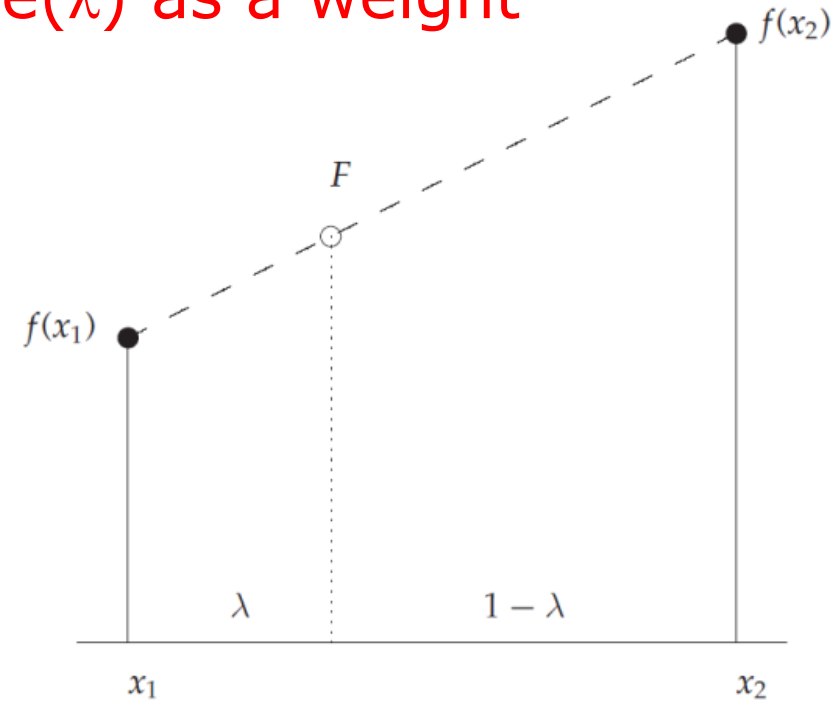


FIGURE 6.6 *Calculating linearly interpolated values.*

$$\frac{F - f(x_1)}{\lambda} = \frac{f(x_2) - f(x_1)}{1}$$

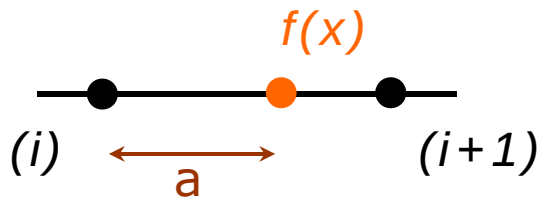
$$F = \lambda f(x_2) + (1 - \lambda) f(x_1)$$

CENGAGE Learning

Interpolation

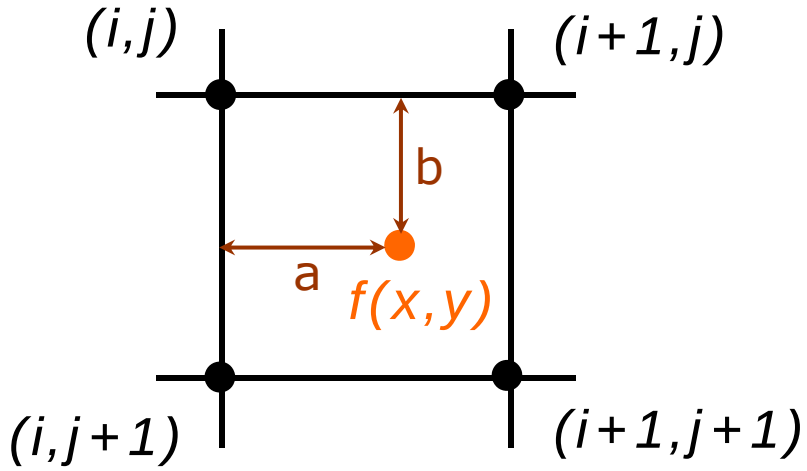
- linear interpolation : 1D

  $f(x) = (1-a) * f(i) + a * f(i+1)$



- bilinear interpolation : 2D uses 4 neighbors

$$f(x,y) = (1-a)(1-b) * f(i,j) + a(1-b) * f(i+1,j) + (1-a)b * f(i,j+1) + ab * f(i+1,j+1)$$



Interpolation

[Q] 2x2 image -> 6x6, p_5 ?

2	8
7	1

->

2	p_1	p_2	8
p_3	p_4	p_5	p_6
p_7	p_8	p_9	p_{10}
7	p_{11}	p_{12}	1

[A]

i) nearest neighbor : $p_5 = 8$

ii) bilinear :

a = 2/3, b = 1/3

p_5 = 1/3*2/3*2 + 2/3*2/3*8
+ 1/3*1/3*7 + 2/3*1/3*1
= 45/9 = 5

bilinear interpolation : 2D

$$f(x,y) = (1-a)(1-b) * f(i,j) + a(1-b) * f(i+1,j) + (1-a)b * f(i,j+1) + ab * f(i+1,j+1)$$

6.2 Image Interpolation

- Function `imresize`

```
imresize(A,k,'method')
```

- Where A is an image of any type, k is a scaling factor, and `'method'` is either `'nearest'` or `'bilinear'`, etc.

```
>> c=imread('cameraman.tif');
>> head=c(33:96,90:153);
>> imshow(head)
>> head4n=imresize(head,4,'nearest');imshow(head4n)
>> head4b=imresize(head,4,'bilinear');imshow(head4b)
```

FIGURE 6.9 & 6.10

FIGURE 6.9 *The head.*

Mathworks

(a)

(b)

FIGURE 6.10 *Scaling by interpolation. (a) Nearest neighbor scaling. (b) Bilinear interpolation.*

Generalized Weight Function, R(u)

$$x_1 \le x' \le x_2$$
$$x' - x_1 = \lambda$$

$$f(x') = R(-\lambda)f(x_1) + R(1-\lambda)f(x_2).$$

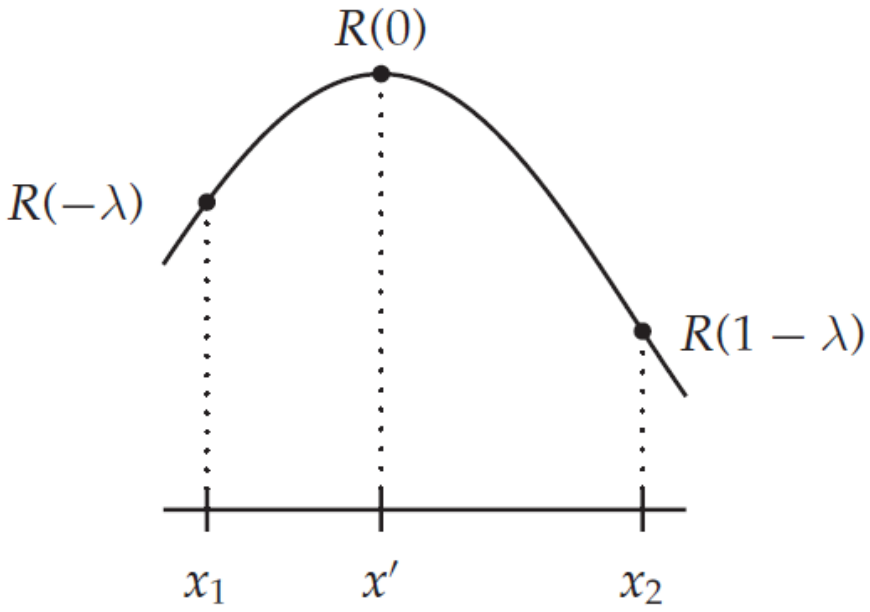


FIGURE 6.11 *Using a general interpolation function.*

Weight function R(u)

- nearest neighbor

$$R_0(u) = \begin{cases} 0 & \text{if } u \le -0.5, \\ 1 & \text{if } -0.5 < u \le 0.5, \\ 0 & \text{if } u > 0.5, \end{cases}$$

- bilinear

$$R_1(u) = \begin{cases} 1+u & \text{if } u \le 0, \\ 1-u & \text{if } u \ge 0. \end{cases}$$

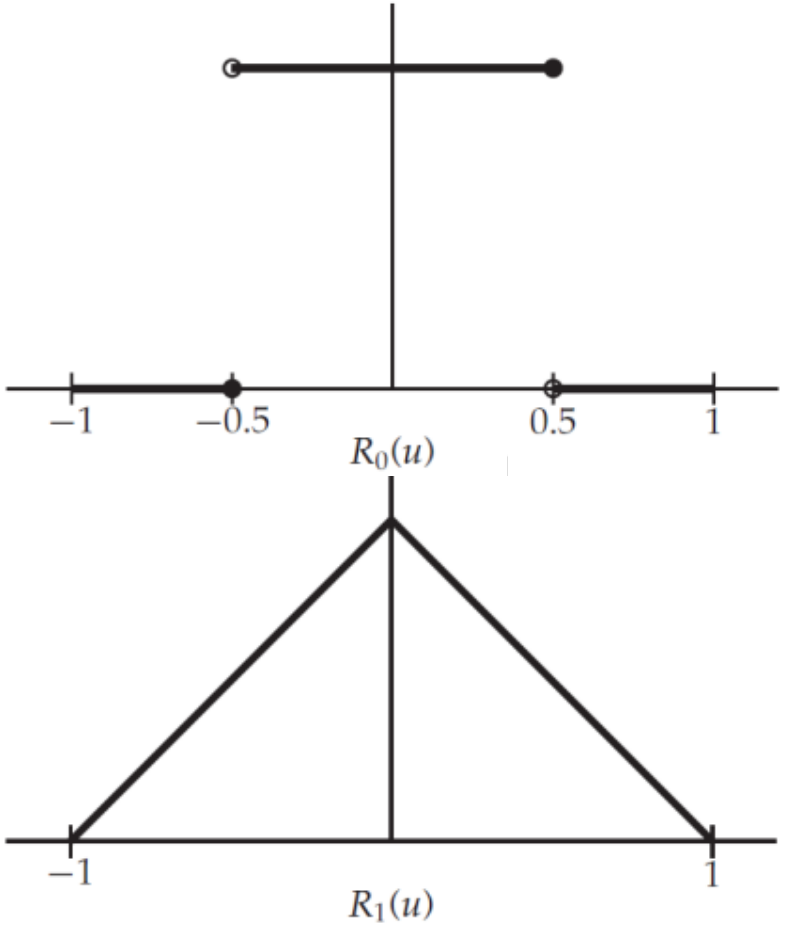


FIGURE 6.12 *Two interpolation functions.*

Cubic Interpolation

- defined over the interval $-2 \le u \le 2$

$$R_3(u) = \begin{cases} 1.5|u|^3 - 2.5|u|^2 + 1 & \text{if } |u| \le 1, \\ -0.5|u|^3 + 2.5|u|^2 - 4|u| + 2 & \text{if } 1 < |u| \le 2. \end{cases}$$

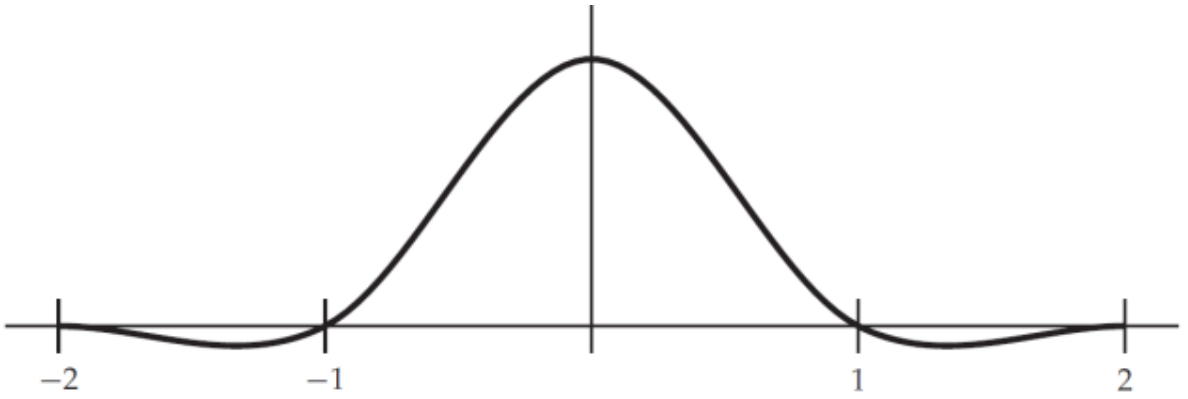


FIGURE 6.13 *The cubic interpolation function* $R_3(u)$.

Deciding Weights by R(u) with 4 Neighboring Pixels

$$f(x') = R_3(-1-\lambda)f(x_1) + R_3(-\lambda)f(x_2) + R_3(1-\lambda)f(x_3) + R_4(2-\lambda)f(x_4),$$

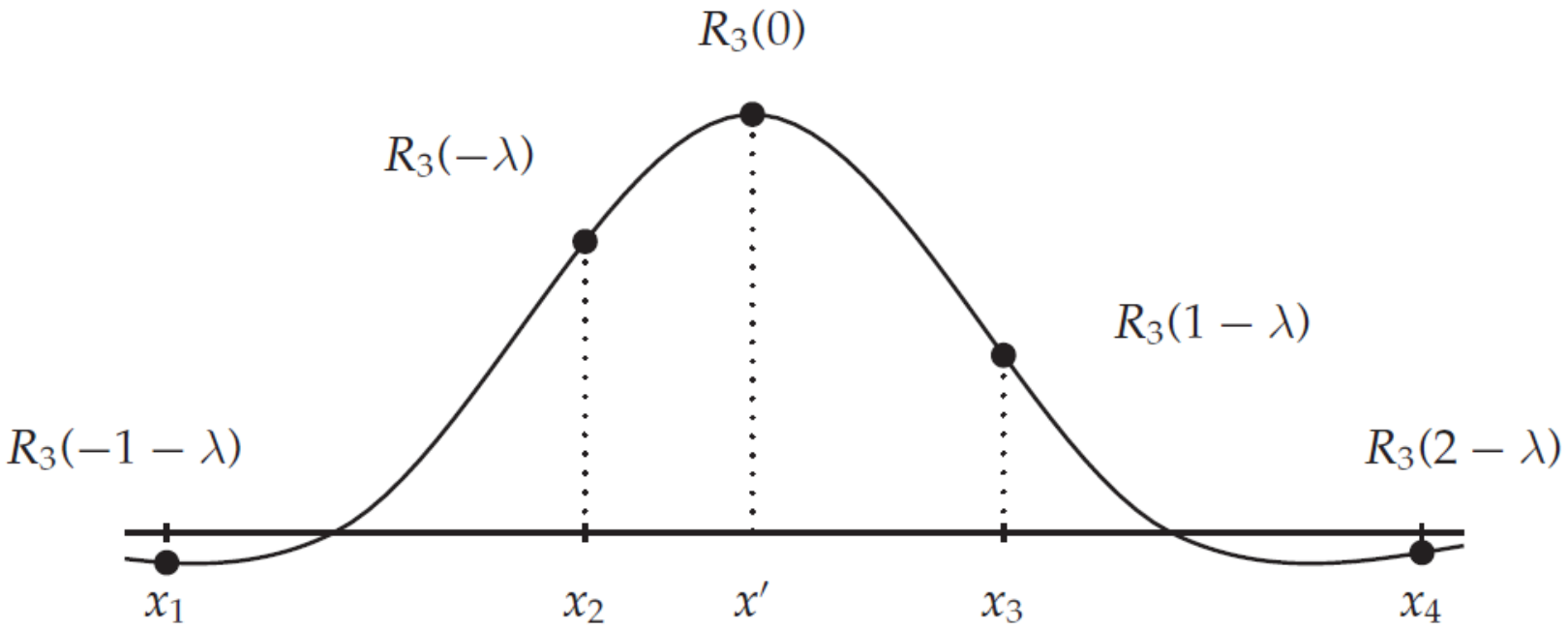


FIGURE 6.14 *Using $R_3(u)$ for interpolation.*

$$x' - x_2 = \lambda$$

Bicubic interpolation for Image

- uses 16 known values around the point (x', y')

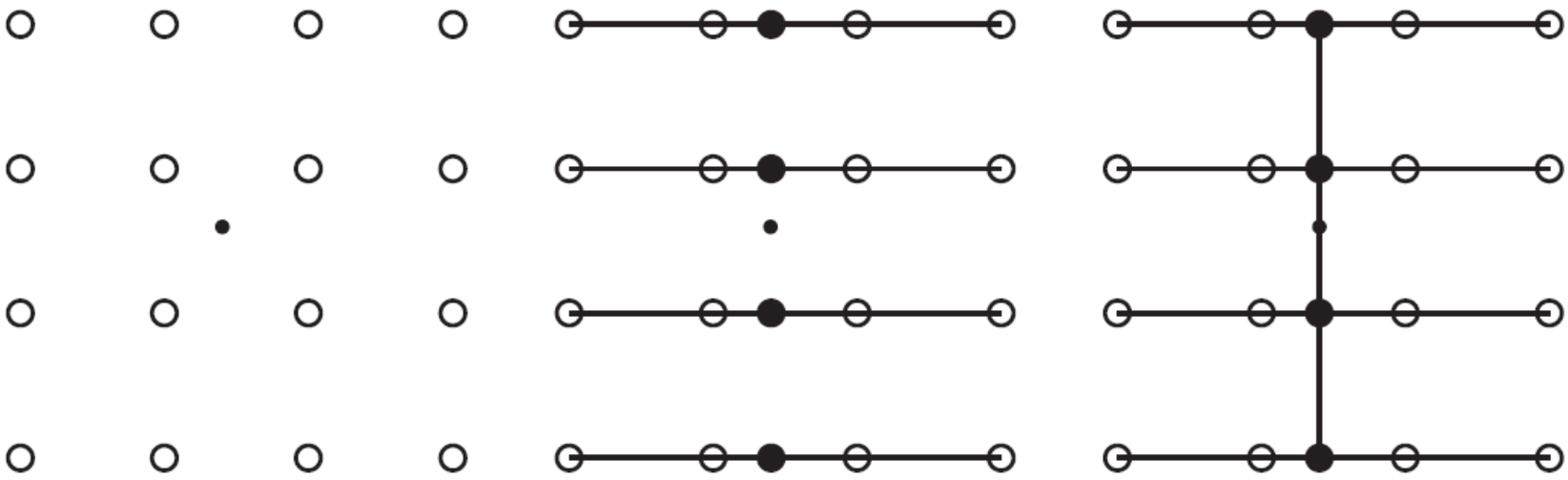


FIGURE 6.15 *How to apply bicubic interpolation.*

FIGURE 6.16

```
>> head4c=imresize(head,4,'bicubic');imshow(head4c)
```

Mathworks

FIGURE 6.16 *Enlargement using bicubic interpolation.*

6.4 Enlargement by Spatial Filtering

- If we merely wish to enlarge an image by a power of two, there is a quick and dirty method that uses linear filtering

```
>> m=magic(4)

m =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

✓ **zero-interleaved (inserting a zero between elements)**

$$m_2(i,j) = \begin{cases} m((i+1)/2, (j+1)/2) & \text{if } i \text{ and } j \text{ are both odd,} \\ 0 & \text{otherwise.} \end{cases}$$

Zero-Interleaved Array in Matlab

This can be implemented with a simple function

```
function out=zeroint(a)
%
% ZEROINT(A) produces a zero-interleaved version of the matrix A.
% For example:
%
%   a=[1 2 3;4 5 6];
%   zeroint(a)
%
%     1     0     2     0     3
%     0     0     0     0     0
%     4     0     5     0     6
%
[m,n]=size(a); a2=reshape([a;zeros(m,n)],m,2*n);
out=reshape([a2';zeros(2*n,m)],2*n,2*m)';
```

FIGURE 6.17 *A simple function for implementing zero interleaving.*

6.4 Enlargement by Spatial Filtering

```
>> m2=zeroint(m)

m2 =

    16     0     2     0     3     0    13     0
     0     0     0     0     0     0     0     0
     5     0    11     0    10     0     8     0
     0     0     0     0     0     0     0     0
     9     0     7     0     6     0    12     0
     0     0     0     0     0     0     0     0
     4     0    14     0    15     0     1     0
     0     0     0     0     0     0     0     0
```

We can now replace the zeros by applying a spatial filter to this matrix

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \frac{1}{64}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

nearest-neighbor bilinear bicubic

6.4 Enlargement by Spatial Filtering

```
>> filter2([1 1 0;1 1 0;0 0 0],m2)

ans =

    16    16     2     2     3     3    13    13
    16    16     2     2     3     3    13    13
     5     5    11    11    10    10     8     8
     5     5    11    11    10    10     8     8
     9     9     7     7     6     6    12    12
     9     9     7     7     6     6    12    12
     4     4    14    14    15    15     1     1
     4     4    14    14    15    15     1     1

>> filter2([1 2 1;2 4 2;1 2 1]/4,m2)

ans =

   16.0000    9.0000    2.0000    2.5000    3.0000    8.0000   13.0000    6.5000
   10.5000    8.5000    6.5000    6.5000    6.5000    8.5000   10.5000    5.2500
    5.0000    8.0000   11.0000   10.5000   10.0000    9.0000    8.0000    4.0000
    7.0000    8.0000    9.0000    8.5000    8.0000    9.0000   10.0000    5.0000
    9.0000    8.0000    7.0000    6.5000    6.0000    9.0000   12.0000    6.0000
    6.5000    8.5000   10.5000   10.5000   10.5000    8.5000    6.5000    3.2500
    4.0000    9.0000   14.0000   14.5000   15.0000    8.0000    1.0000    0.5000
    2.0000    4.5000    7.0000    7.2500    7.5000    4.0000    0.5000    0.2500
```

Image Enlargement in 4 Ways

```
>> imshow(hz)
>> imshow(filter2([1 1 0;1 1 0;0 0 0],hz)/255)
>> imshow(filter2([1 2 1;2 4 2;1 2 1]/4,hz)/255)
>> bfilt=[1 4 6 4 1;4 16 24 16 4;6 24 36 24 6;4 16 24 16 4;1 4 6 4 1]/64;
>> imshow(filter2(bfilt,hz)/255)
```

FIGURE 6.18 *Enlargement by spatial filtering.*

6.5 Scaling Smaller

- Making an image smaller is also called **image minimization**
- **Subsampling**
- problems in high freq. image

```
>> tr=imresize(t,0.25);
```

```
>> trc=imresize(t,0.25,'bicubic');
```


lowpass filtering

6.6 Rotation

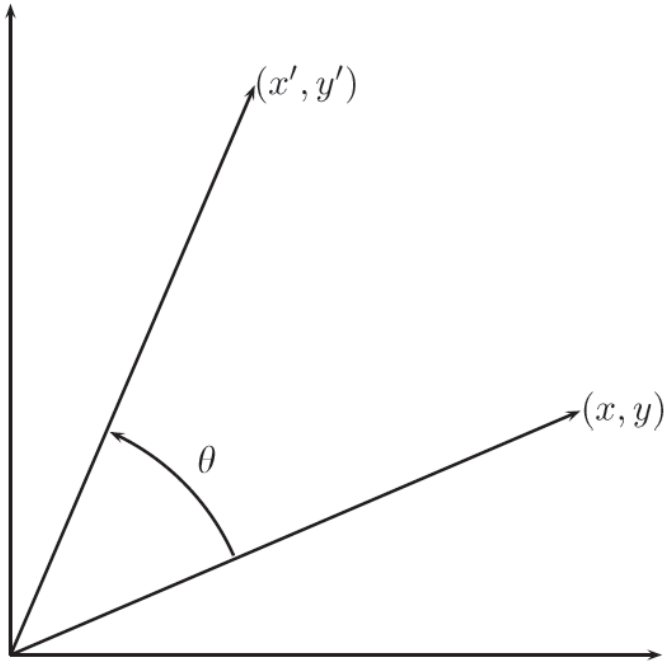


FIGURE 6.20 *Rotating a point through angle θ.*

does not fit for Cartesian grid after rotation

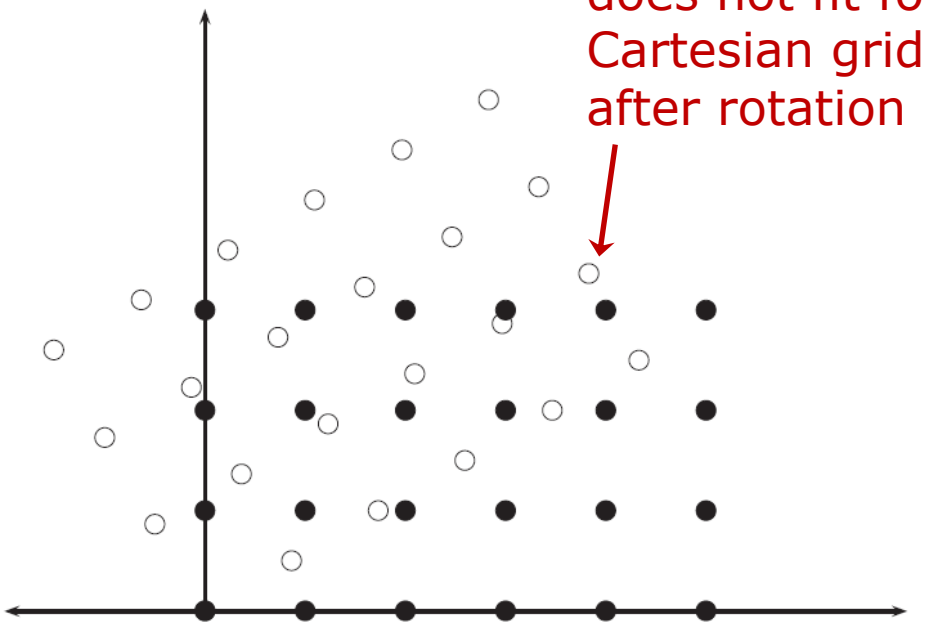


FIGURE 6.21 *Rotating a rectangle.*

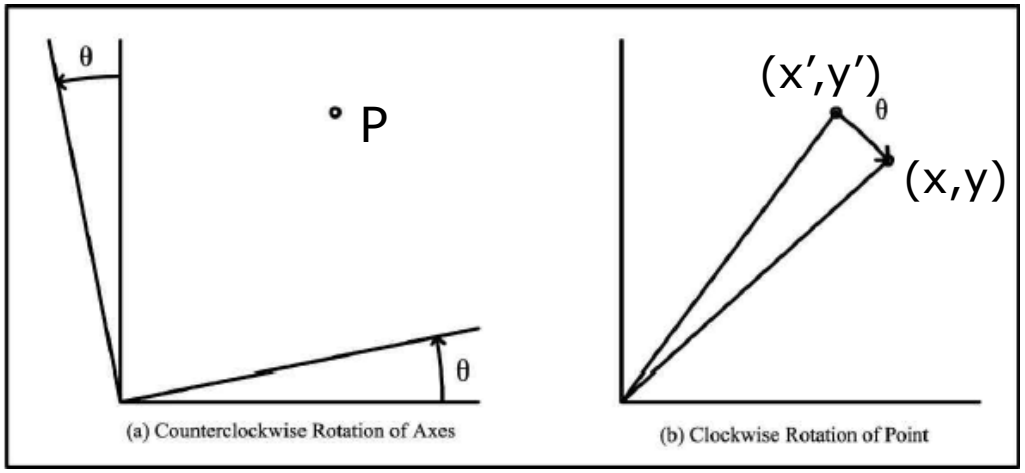
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

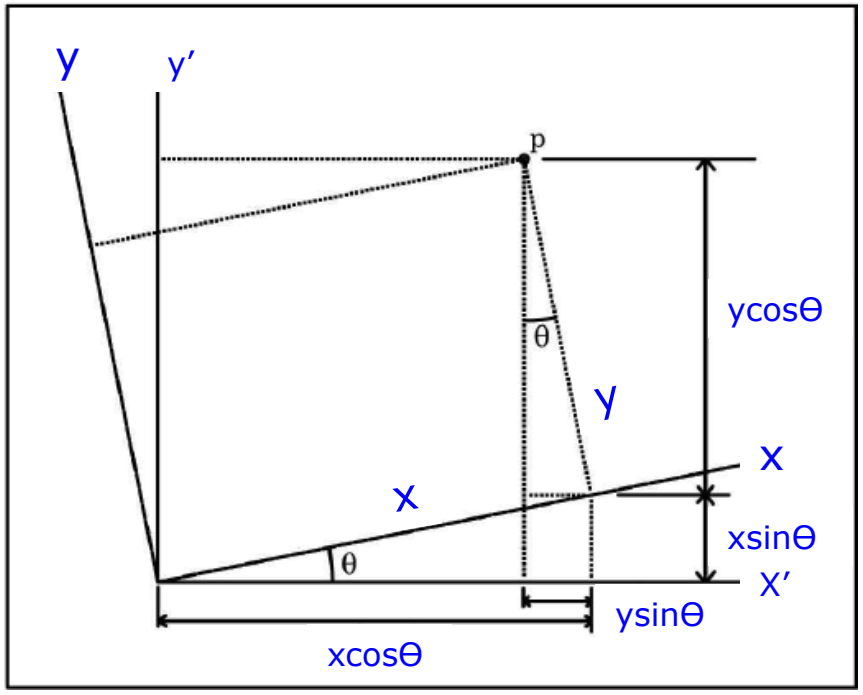
$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix}$$

The rotation matrix is orthogonal so its transpose equals inverse matrix. Multiplying on the left by the transpose of the matrix.

counter clockwise rotation

Derivation of Rotation Matrix



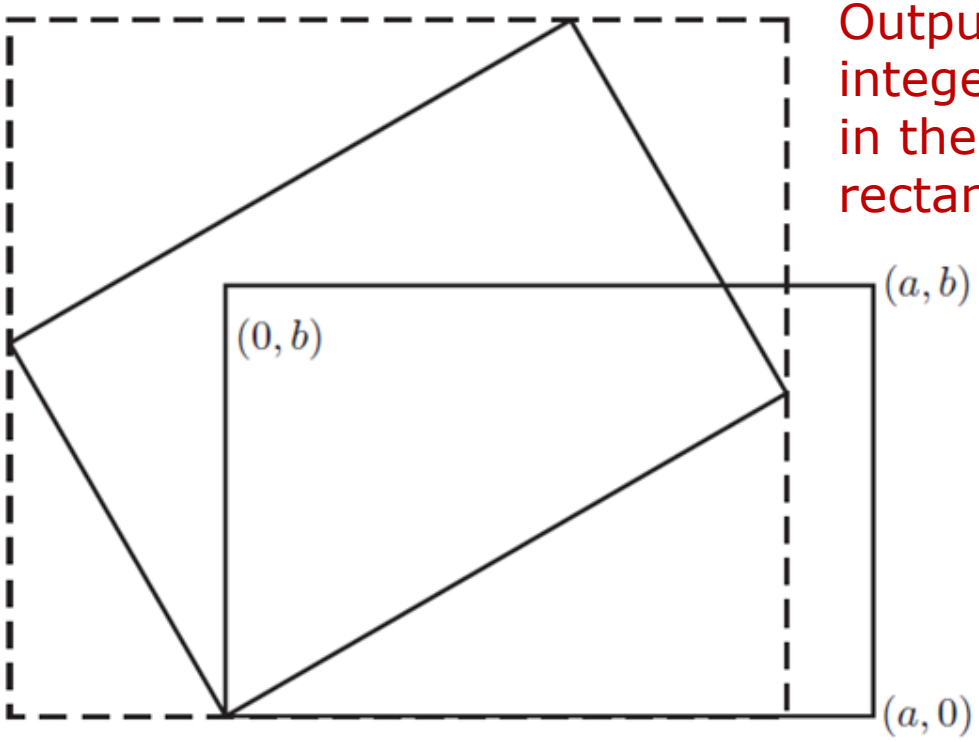


$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

CENGAGE Learning

Image Rotation



Output pixel value is integer x,y grid (x', y') in the dashed rectangle.

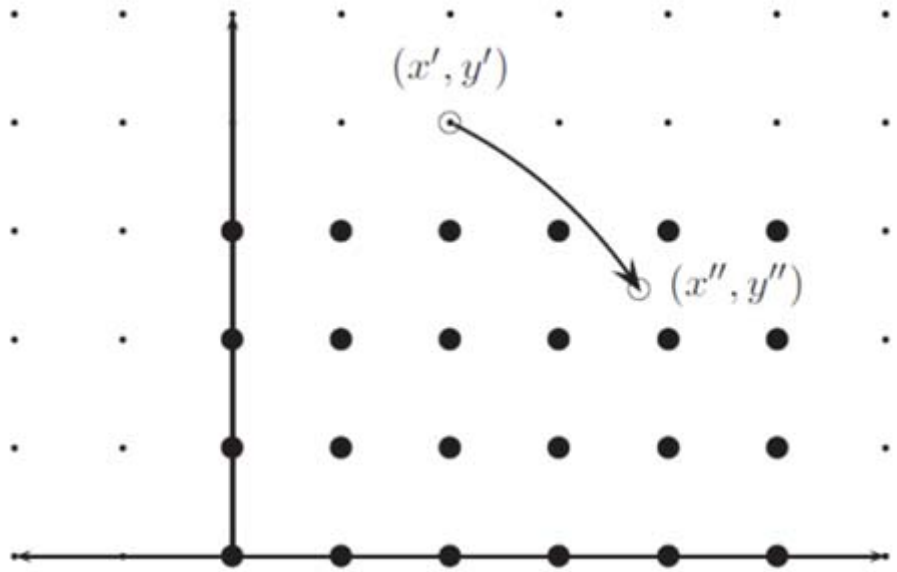
- When the output is rotated back, all the values should lie within the original image limits

$$0 \le x'\cos\theta + y'\sin\theta \le a,$$

$$0 \le -x'\sin\theta + y'\cos\theta \le b$$

Rotation: How to Get the Value at (x', y')

1. Get the location (x'', y'') by rotating back the (x',y') position.
2. Calculate the gray value at (x'', y'') by interpolation, using surrounding gray values.
3. The interpolated value (x'', y'') becomes the value for pixel at (x', y') in the rotated image



Rotation in Malab

```
>> cr=imrotate(c,60); % nearest neighbor
>> imshow(cr)
>> crc=imrotate(c,60,'bicubic');
>> imshow(crc)
```

Mathworks

(a) (b)

FIGURE 6.25 *Rotation with interpolation. (a) Nearest neighbor. (b) Bicubic interpolation.*

Anamorphosis

- Deliberately distorted image of an object shape for artistic or dramatic effect.

FIGURE 6.26 The Ambassadors *(1533) by Hans Holbein.*

Extraction of Anamorphosis Image

```
>> a=imread('AMBASSADORS.JPG');
>> a=rgb2gray(a);
```

```
>> skull=a(566:743,157:586);
```

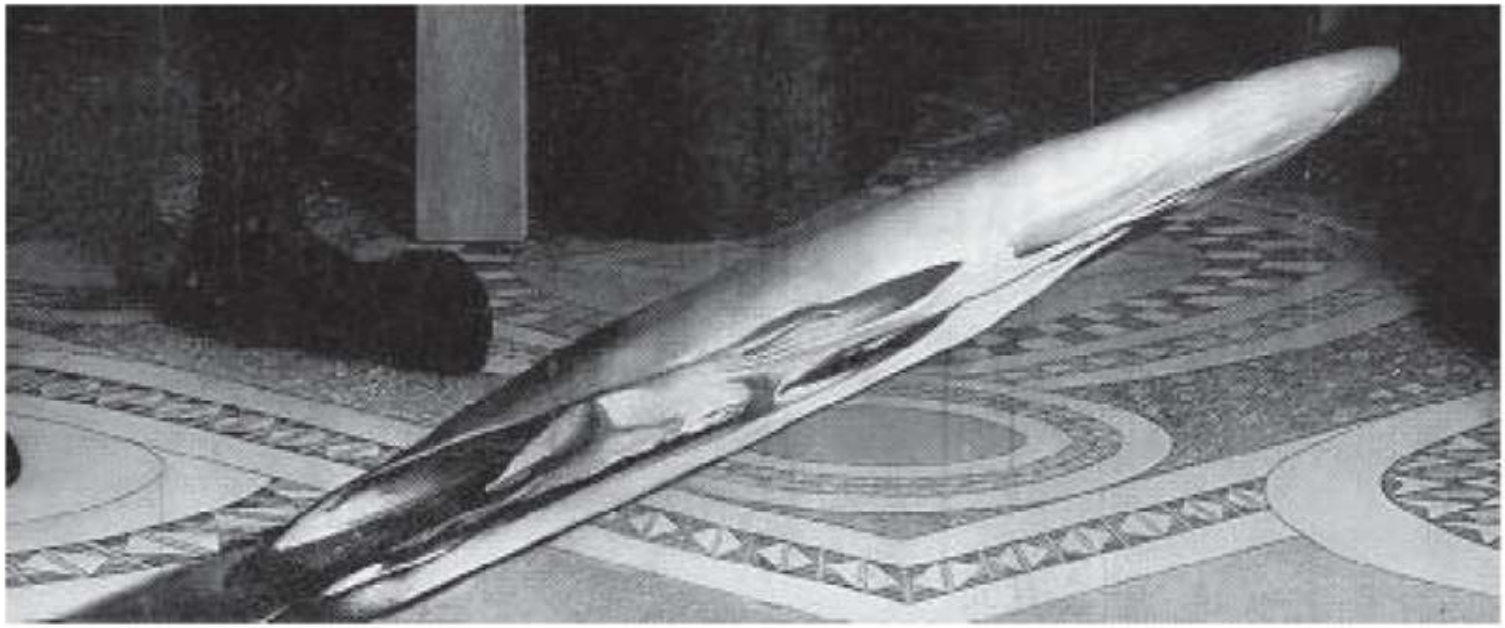


FIGURE 6.27 *The skull alone.*

Undoing Anamorphosis

```
>> skull2=imresize(imrotate(skull,-22,'bicubic'),[500,150],'bicubic');
```

```
>> imshow(skull2(200:350,:))  % extraction of skull area
```

rotation followed
by stretching
-> needs trial and
error

FIGURE 6.28 *The corrected skull.*

Summary

- **Chap 6. Image Geometry**
  - ✓ Interpolation: nearest neighbor, bilinear, bicubic
  - ✓ enlargement, minimization
  - ✓ rotation
  - ✓ anamorphosis