Image Processsing and Analysis
Lecture 3、Image Enhancement in the Spatial Domain(I)

Weiqiang Wang

School of Computer and Control Engineering, UCAS

October 15, 2015

Preview

- The principal objective of enhancement is to process an image so that the result is more suitable than the original image for a specific application
  - "Specific" means the techniques are very much problem oriented.
- Image enhancement approaches fall into two broad categories, spatial domain methods and frequency domain method.
  - spatial domain methods are based on direct manipulation of pixel in an image.
  - frequency domain techniques are based on modifying the Fourier transform of an image.
- There is no general theory of image enhancement.
  - When an image is processed for visual interpretation, the viewer is the ultimate judge of how well a particular method works, which makes it difficult to compare the performance of different methods.
  - For machine perception, the evaluation task is somewhat easier, e.g., character recognition task.

Outline

Background

- A mathematical representation of spatial domain processing:

$$g(x, y) = T[f(x, y)],$$

where $f(x, y)$: the input image
$g(x, y)$: the processed image
T: an operator on f, defined over some neighborhood of (x, y)

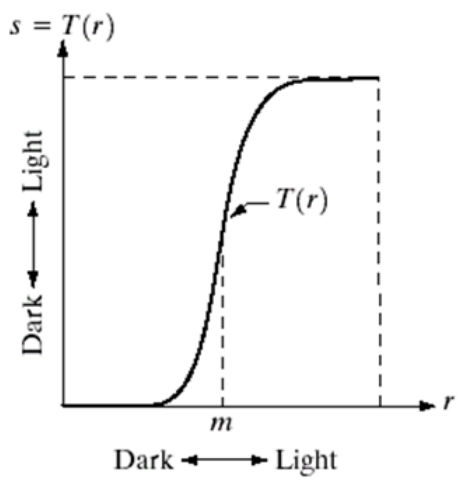
  - T can operate on a set of images, e.g., noise reduction
  - Square and rectangular neighborhood are by far the most popular due to their ease of implementation, although circle is also used.

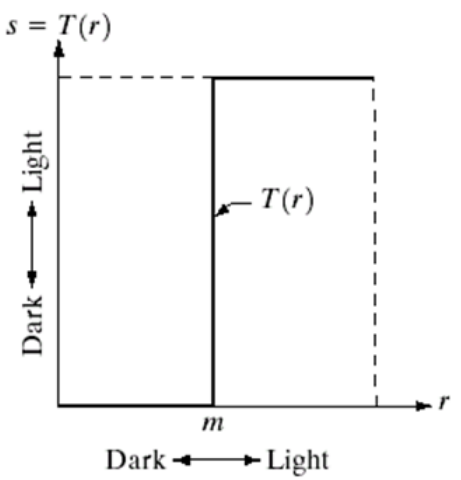
- The simplest form of T is when the neighborhood is of size 1×1. In this case, g depends only on the value of f at (x, y), i.e.,

$$s = T(r),$$

that is called Intensity Transformation

Gray-level Transformation





Power-Law Transformation

- $s = cr^{\gamma}$ where c, γ: positive constants

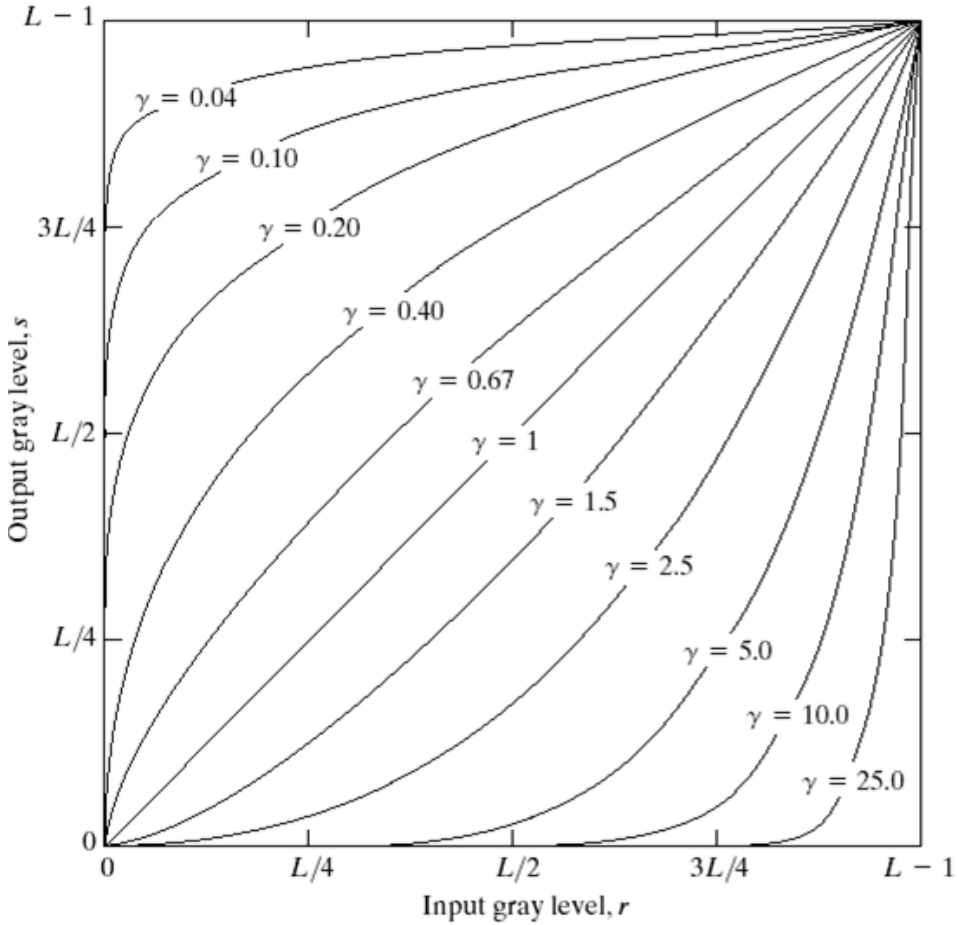
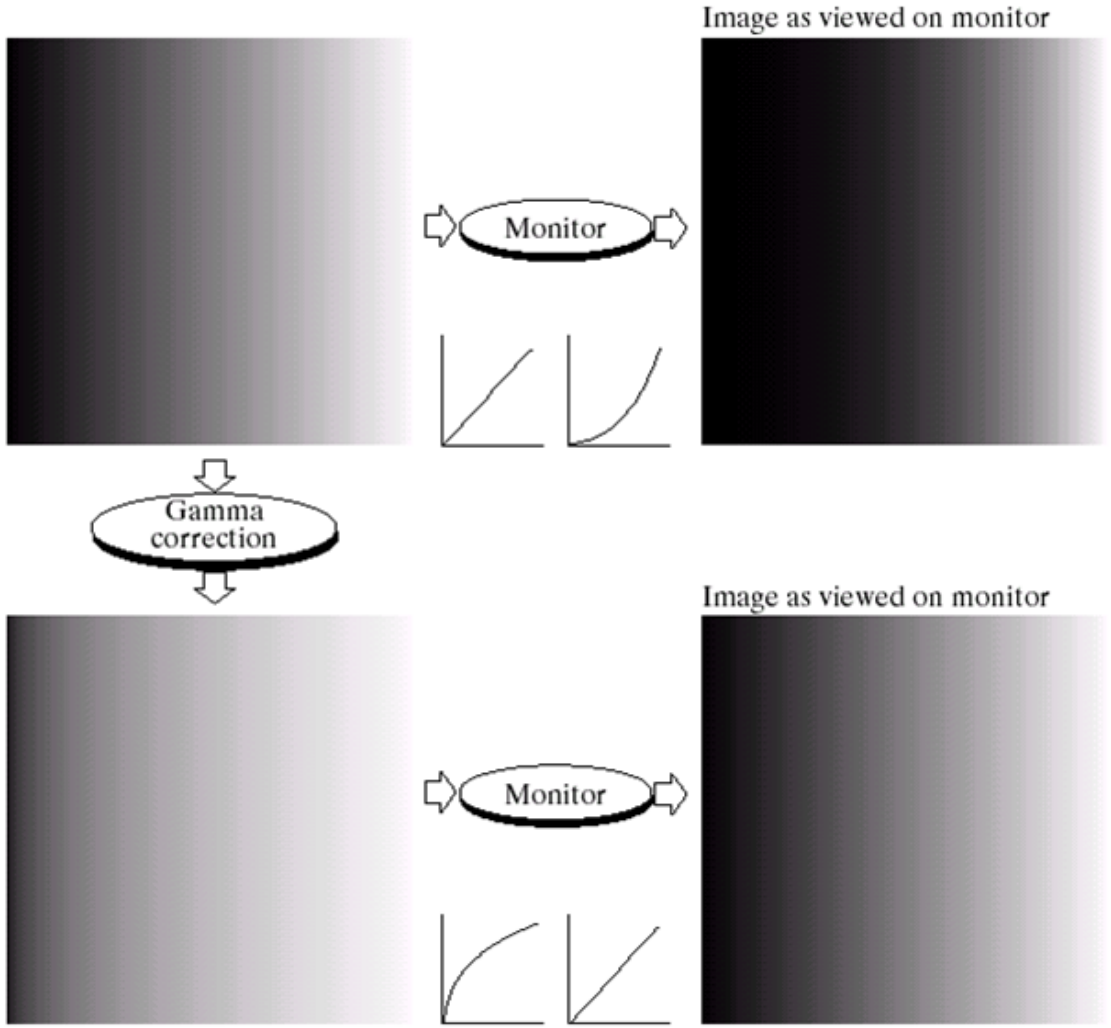


FIGURE 3.6 Plots of the equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases).

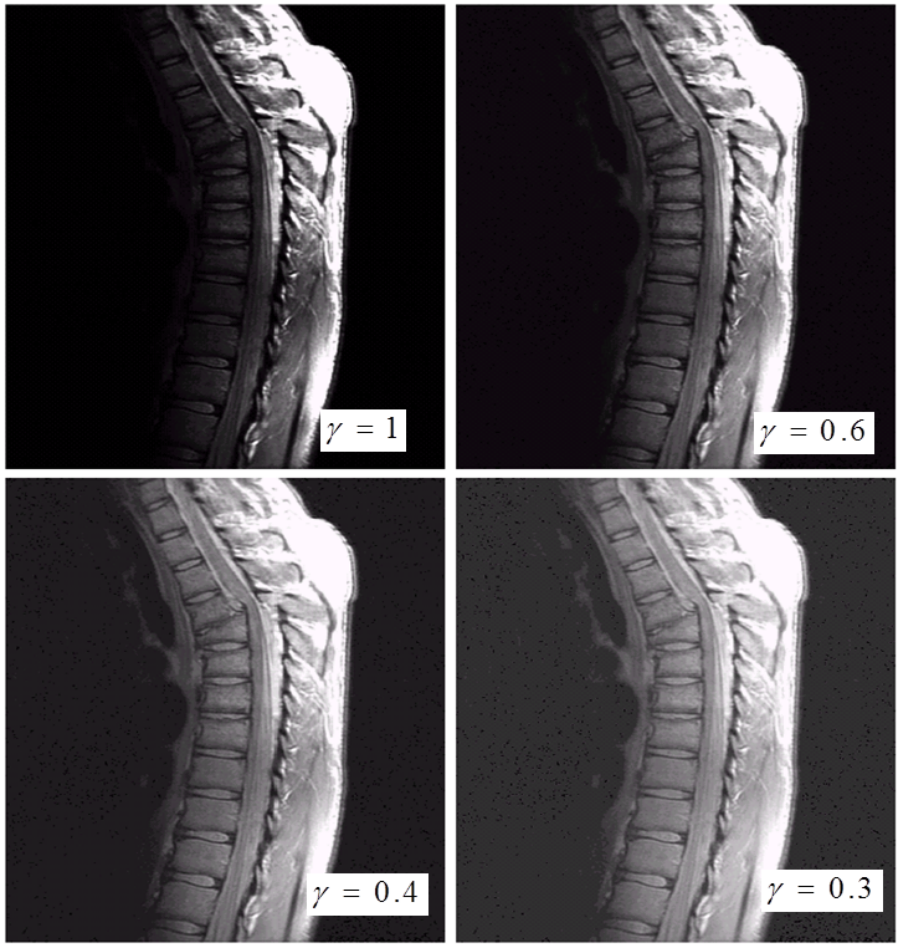
Power-Law Transformation Example 1: Gamma Correction

a b
c d

FIGURE 3.7
(a) Linear-wedge gray-scale image. (b) Response of monitor to linear wedge. (c) Gamma-corrected wedge. (d) Output of monitor.



Power-Law Transform Example 2: Contrast manipulation



a b
c d

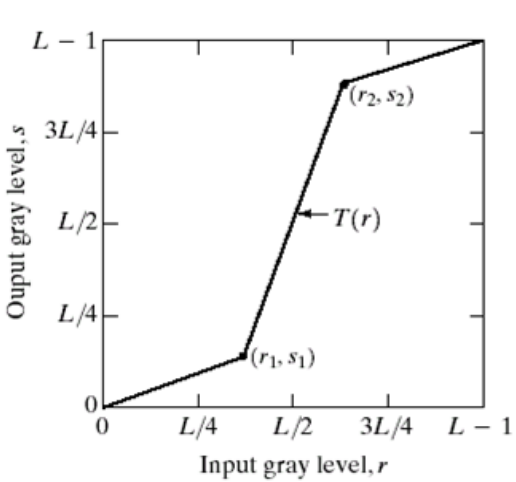
FIGURE 3.8
(a) Magnetic resonance (MR) image of a fractured human spine.
(b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 0.6, 0.4$, and 0.3, respectively.
(Original image for this example courtesy of Dr. David R. Pickens, Department of Radiology and Radiological Sciences, Vanderbilt University Medical Center.)

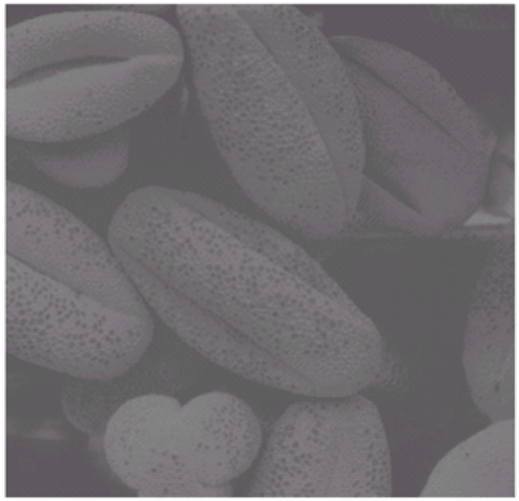
Power-Law Transform Example 3: Contrast manipulation

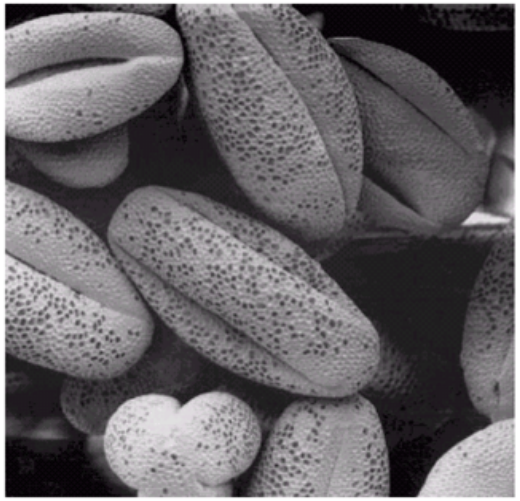
a b
c d

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.)

Piecewise-Linear Transformation Functions
Case 1: Contrast Stretching



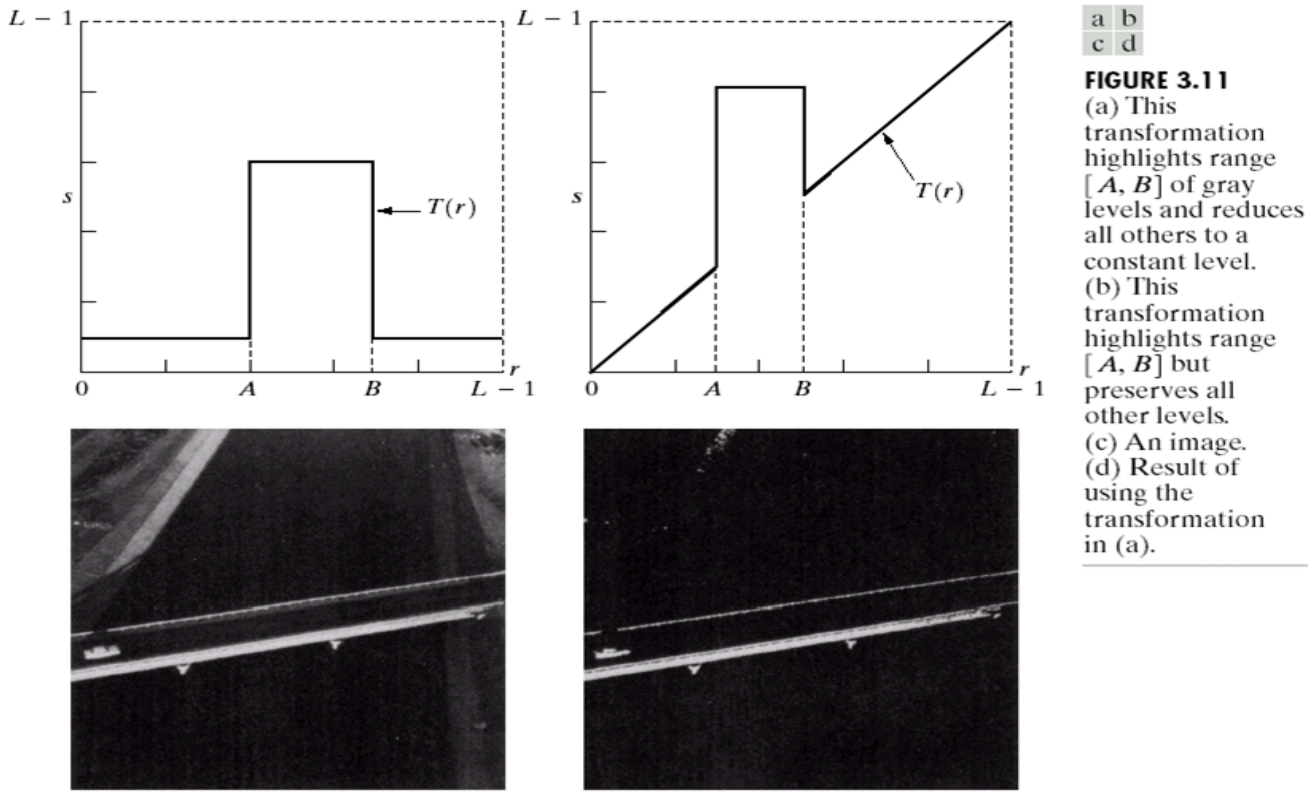




a b
c d

FIGURE 3.10 Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

Piecewise-Linear Transformation Functions
Case 2:Gray-level Slicing



a b
c d

FIGURE 3.11
(a) This transformation highlights range $[A, B]$ of gray levels and reduces all others to a constant level. (b) This transformation highlights range $[A, B]$ but preserves all other levels. (c) An image. (d) Result of using the transformation in (a).

Piecewise-Linear Transformation Functions
Case 3:Gray-level Slicing

- Bit-plane slicing:
  - It can highlight the contribution made to total image appearance by specific bits.
  - Each pixel in an image represented by 8 bits.
  - Image is composed of eight 1-bit planes, ranging from bit-plane 0 for the least significant bit to bit plane 7 for the most significant bit.

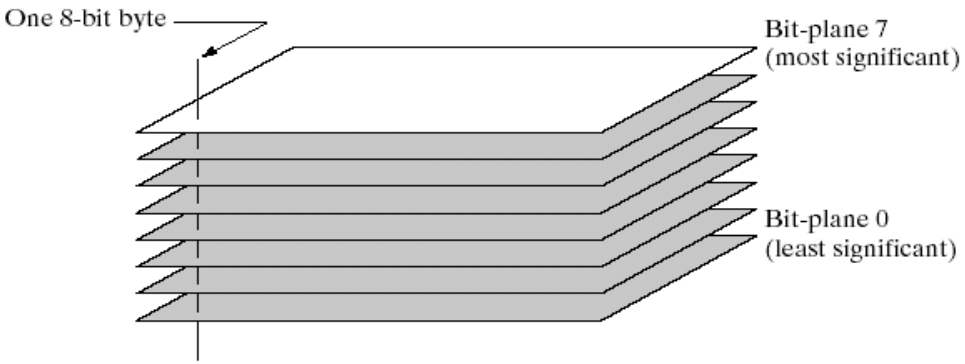


FIGURE 3.12
Bit-plane representation of an 8-bit image.

Piecewise-Linear Transformation Functions
Bit-plane Slicing: A Fractal Image

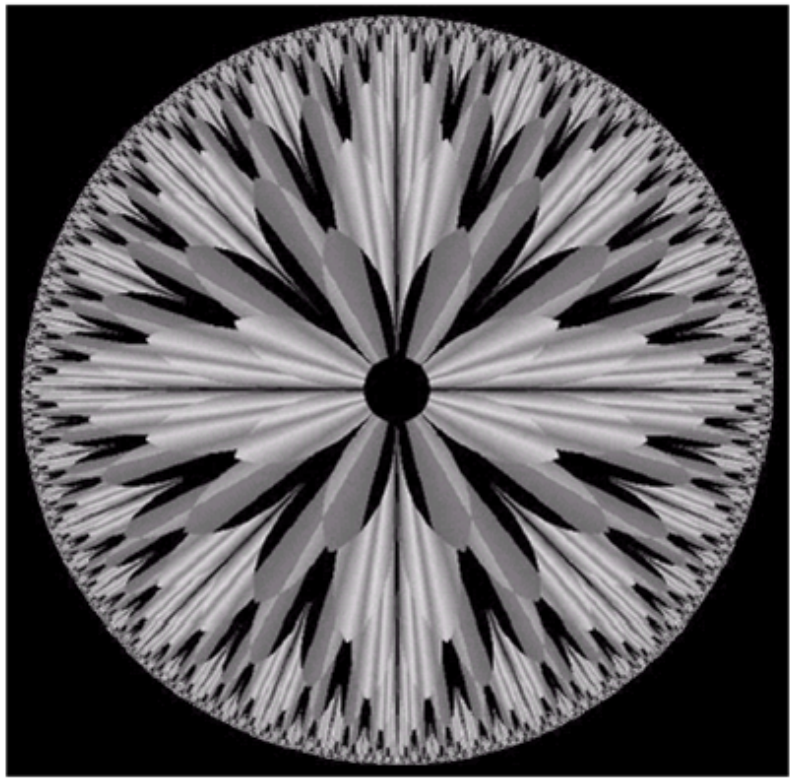


FIGURE 3.13 An 8-bit fractal image. (A fractal is an image generated from mathematical expressions). (Courtesy of Ms. Melissa D. Binde, Swarthmore College, Swarthmore, PA.)

Piecewise-Linear Transformation Functions
Bit-plane Slicing: A Fractal Image

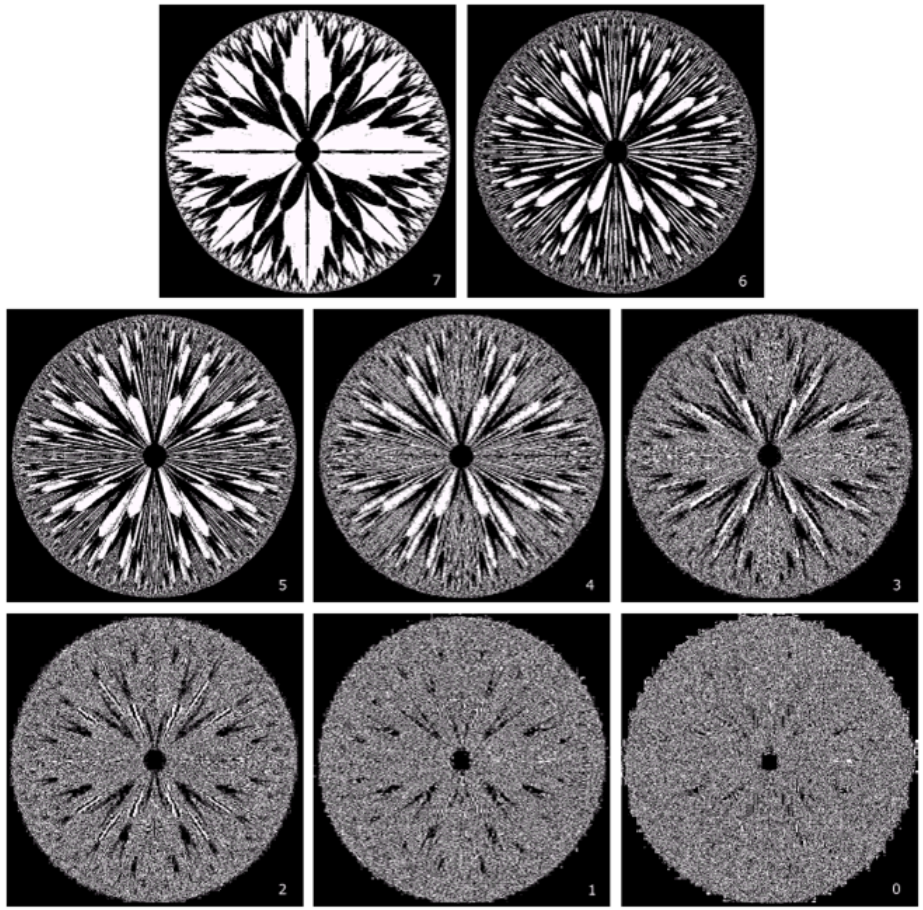
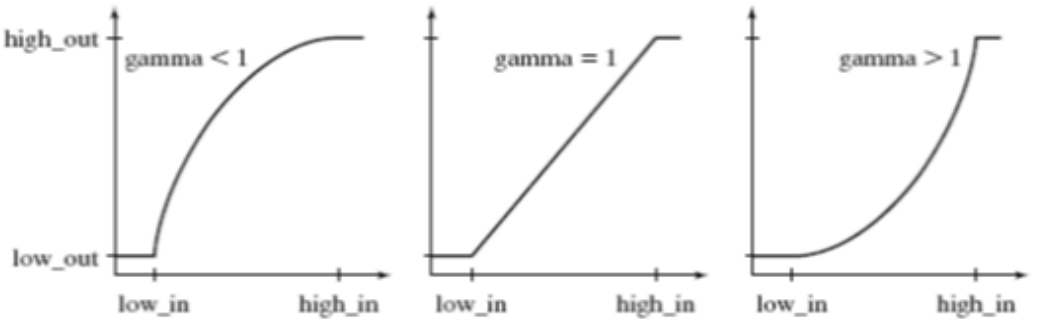


FIGURE 3.14 The eight bit planes of the image in Fig. 3.13. The number at the bottom, right of each image identifies the bit plane.

Function imadjust in IPT

- $g = imadjust(f, [low\_in, high\_in], [low\_out, high\_out], gamma)$
- This function maps the intensity values in image f to new values in g.
- Such that values between low_in and $high\_in$ map to values between low_out and $high\_out$.
- Values below low_in and above $high\_in$ are clipped; that is, values below low_in map to low_out, and those above $high\_in$ map to $high\_out$.

Function imadjust in IPT



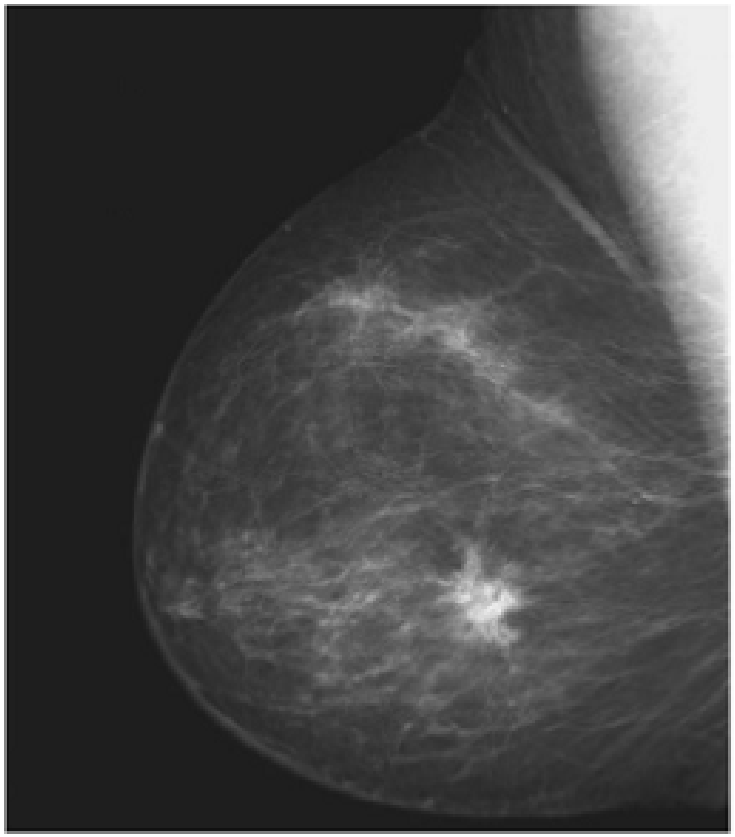
a b c

FIGURE 3.2 The various mappings available in function `imadjust`.

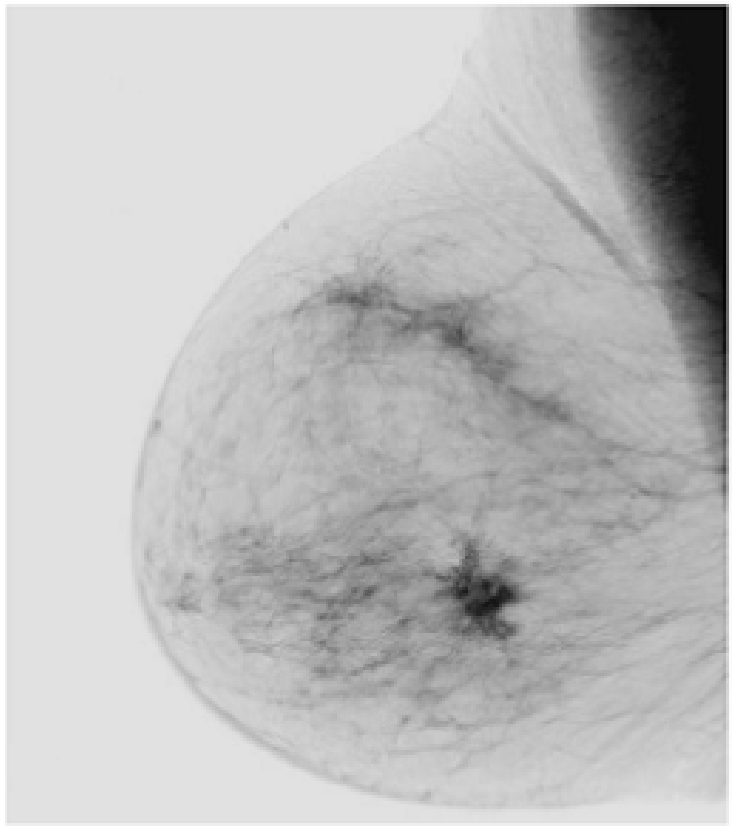
- Parameter $gamma$ specifies the shape of the curve that maps the intensity values in f to create g.
- If $gamma$ is less than 1, the mapping is weighted toward higher $brighter$ output values, as Fig. 3.2(a) shows.
- If $gamma$ is greater than 1,the mapping is weighted toward lower $darker$ output values.
- If it is omitted from the function argument, $gamma$ defaults to 1 $linearmapping$.

An Example

- I=imread('Fig0303(a)(breast).tif');
- imshow(I)



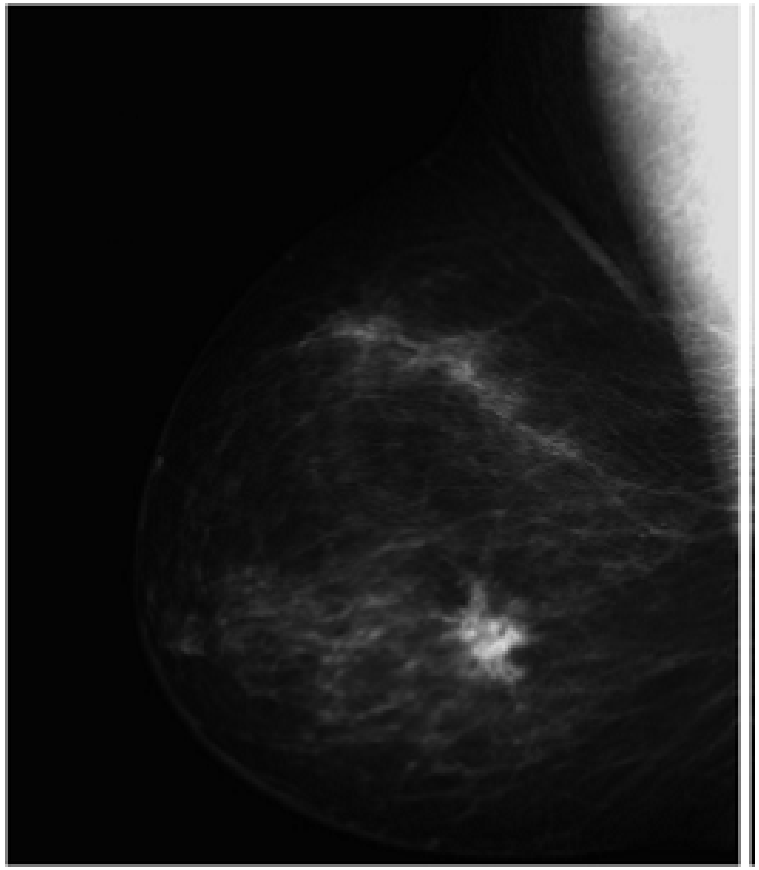
- G=imadjust(I,[0 1],[1 0]);
- imshow(G)



An Example

- G=imadjust(I,[0.5 0.75],[0 1]);
- imshow(G)

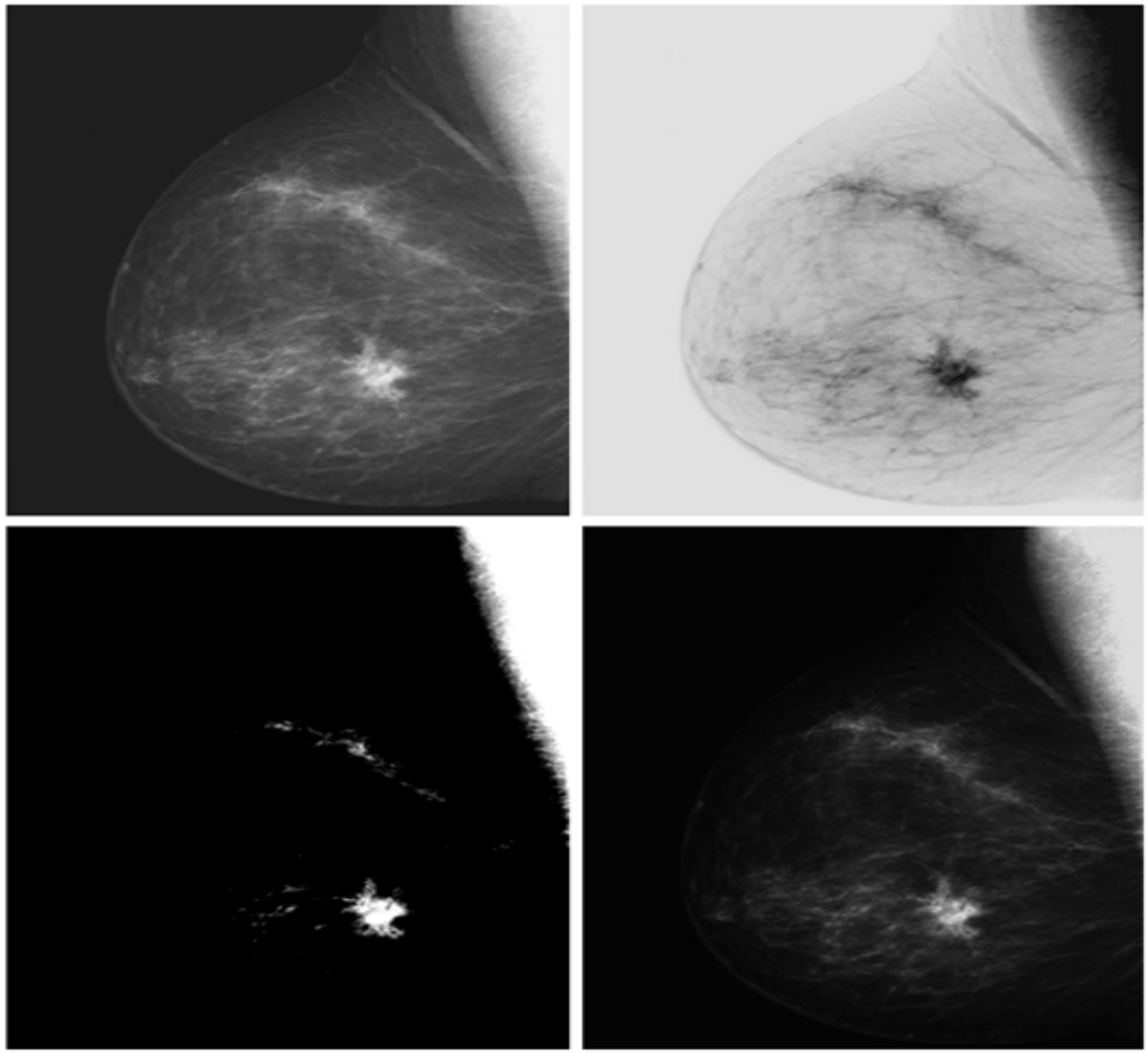
- G=imadjust(I,[],[],2);
- imshow(G)



Function imadjust

a b
c d

FIGURE 3.3 (a) Original digital mammogram. (b) Negative image. (c) Result of expanding the intensity range [0.5, 0.75]. (d) Result of enhancing the image with `gamma = 2`. (Original image courtesy of G. E. Medical Systems.)



Logarithmic Transformation

- Logarithmic and contrast-stretching transformations are basic tools for dynamic range manipulation.
- Logarithm transformations are implemented using the expression

$$s = c\log(1 + r)$$

  - where c is a constant.
  - The shape of this transformation is similar to the $gamma$ curve with the low values set at 0 and the high values set to 1 on both scales.
  - Note, however, that the shape of the $gamma$ curve is variable,whereas the shape of the log function is fixed.

Logarithmic Transformation

- One of the principal uses of the log transformation is to compress dynamic range.
  - For example, it is not unusual to have a Fourier spectrum (Chapter 4) with values in the range [0,10^6] or higher. When displayed on a monitor that is scaled linearly to 8 bits, the high values dominate the display, resulting in lost visual detail for the lower intensity values in the spectrum. By computing the log, a dynamic range on the order of, for example, 10^6 is reduced to approximately 14, which is much more manageable.
- When performing a logarithmic transformation, it is often desirable to bring the resulting compressed values back to the full range of the display. For 8 bits, the easiest way to do this in MATLAB is with the statement

$$gs = im2uint8(mat2gray(g));$$

  - Use of $mat2gray$ brings the values to the range [0, 1] and $im2uint8$ brings them to the range [0, 255].

Logarithmic Transformation

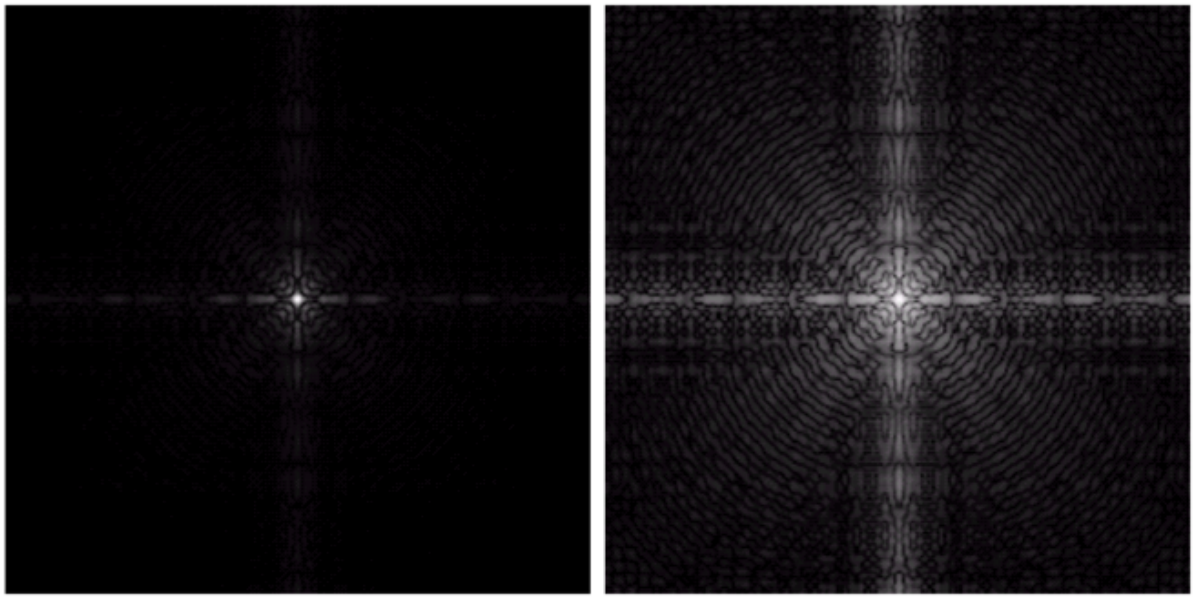
- Figure 3.5(a) is a Fourier spectrum with values in the range 0 to displayed on a linearly scaled, 8-bit system. Figure 3.5(b) shows the result obtained using the commands

  >> g = im2uint8(mat2gray(log(1 + double(f))));

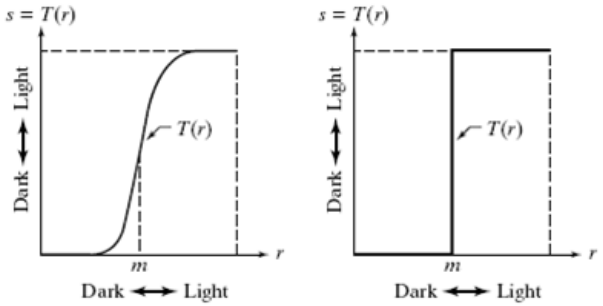
  >> imshow(g)

a b

FIGURE 3.5
(a) Fourier spectrum.
(b) Result of applying the log transformation given in Eq. (3.2-2) with $c = 1$.



Contrast-Stretching Transformation



a b

FIGURE 3.4
(a) Contrast-stretching transformation. (b) Thresholding transformation.

- A contrast-stretching transformation function can be defined as

$$s = T(r) = \frac{1}{1 + (m/r)^E}$$

  - it compresses the input levels lower than m into a narrow range of dark levels in the output image;
  - similarly, it compresses the values above m into a narrow band of light levels in the output;
  - where r represents the intensities of the input image, s the corresponding intensity values in the output image, and E controls the slope of the function.
- This equation is implemented in MATLAB for an entire image as

```
g = 1./(1 + (m./(double(f) + eps)).^E)
```

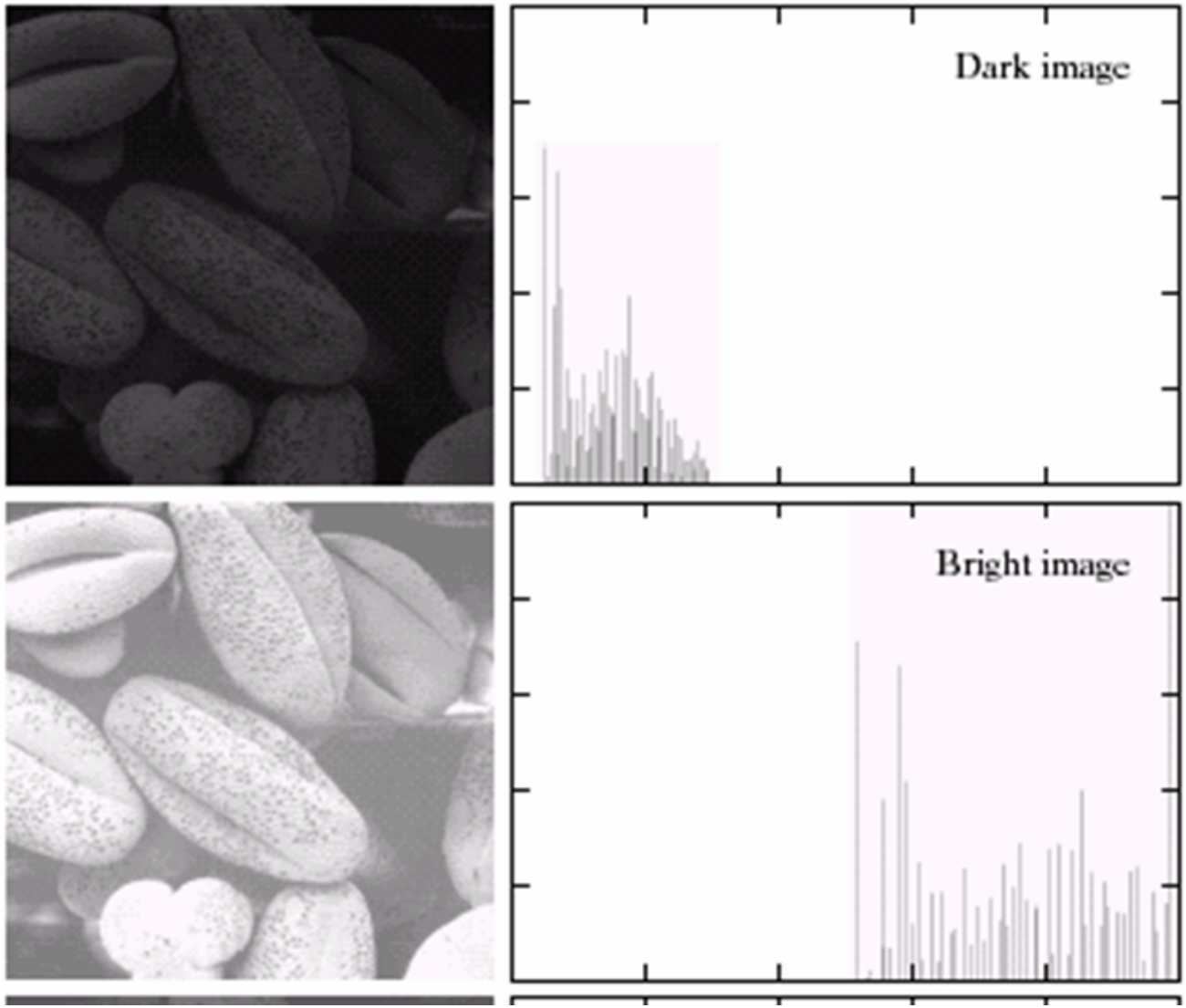
Histogram Processing

- Intensity transformation are based on information extracted from image intensity histograms, and histogram plays a basic role in image processing, in areas such as enhancement, compression, segmentation, and description.
- The histogram of a digital image with L total possible intensity levels in the range [0,G] is defined as the discrete function

$$h(r_k) = n_k$$

  - where r_k is the kth intensity level in the interval [0, G] and n_k is the number of pixels in the image whose intensity level is r_k.The value of G is 255 for images of class uint8, 65535 for images of class uint16, and 1.0 for images of class double.

Generating and Plotting Image Histograms



Normalized Histogram

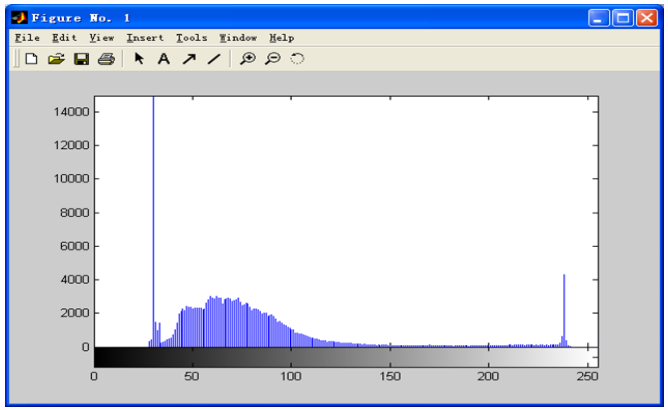
- Through dividing all elements of $h(r_k)$ by the total number of pixels in the image, a normalized histogram can be obtained, i.e.,

$$p(r_k) = \frac{h(r_k)}{N} = \frac{n_k}{N}, N = \sum_{k=1}^{L} n_k, k = 1, 2, ..., L$$

- From basic probability, we know $p(r_k)$ can be considered as an estimate of the probability of occurrence of intensity level r_k.

Generating Image Histograms

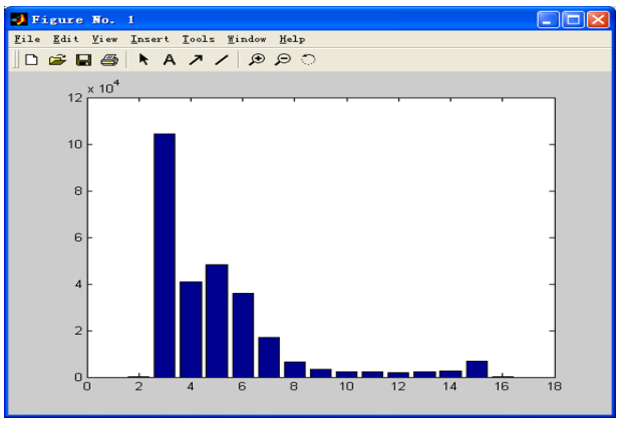
- h=imhist(f,b)
  - Where f is the input image, h is its histogram, and b is the number of bins used in forming the histogram.
  - If b is not included in the argument, b = 256 is used by default.
  - A bin is simply a subdivision of the intensity scale.
- We obtain the normalized histogram simply by using the expression:
  p=imhist(f,b)/numel(f)
- An example
  - f=imread('Fig3_8_a.tif');
  - imhist(f)



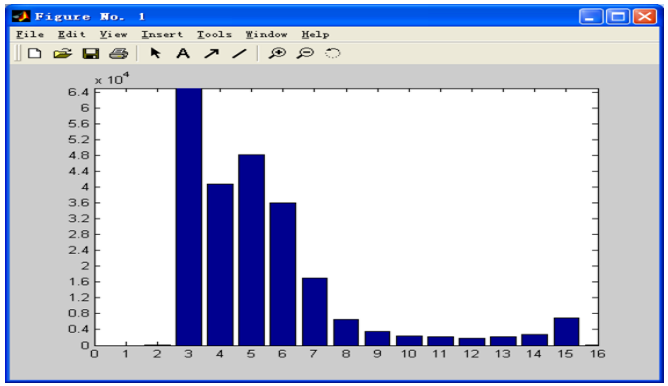
Plotting Image Histograms

- Histograms often are plotted using bar graphs. For this purpose we can use the function bar(horz, v, width)
  - Where v is a row vector containing the points to be plotted.
  - horz is a vector of the same dimension as v that contains the increments of the horizontal scale. If horz is omitted, the horizontal axis is divided in units from 0 to length(v).
  - width is a number between 0 and 1.

- s=imread('Fig0303(a)(breast).tif');
- h1=imhist(s,16);
- horz=1:16;
- bar(horz,h1,0.8)



- axis([0 16 0 65000]);
- set(gca,'xtick',0:1:16);
- set(gca,'ytick',0:4000:65000);



Plotting Image Histograms using stem

- A stem graph is similar to a bar graph. The syntax is

  stem(horz, v, 'color_linestyle_marker', 'fill')

  - where v is row vector containing the points to be plotted, and horz is as described for bar.
  - $color\_linestyle\_marker$ is a triplet of values from Table below.

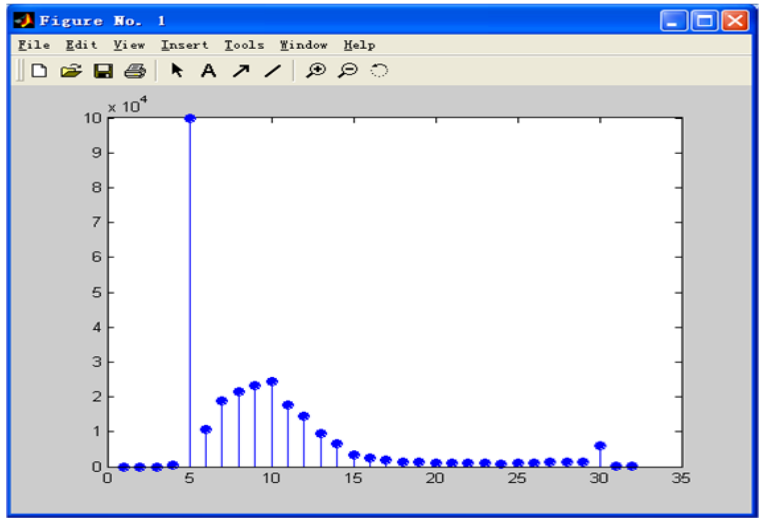
Symbol	Color	Symbol	Line Style	Symbol	Marker
k	Black	-	Solid	+	Plus sign
w	White	--	Dashed	o	Circle
r	Red	:	Dotted	*	Asterisk
g	Green	-.	Dash-dot	.	Point
b	Blue	none	No line	x	Cross
c	Cyan			s	Square
y	Yellow			d	Diamond
m	Magenta			none	No marker

TABLE 3.1
Attributes for functions `stem` and `plot`. The `none` attribute is applicable only to function `plot`, and must be specified individually. See the syntax for function `plot` below.

Some examples using stem

- stem(v, 'rs')
  produces a stem plot where the lines and markers are red, the lines are dashed, and the markers are squares.
- If $fill$ is used, and the $marker$ is a circle, square, or diamond, the $marker$ is filled with the color specified in color.
- The default $color$ is black, the $linestyle$ default is solid, and the default $marker$ is a circle.

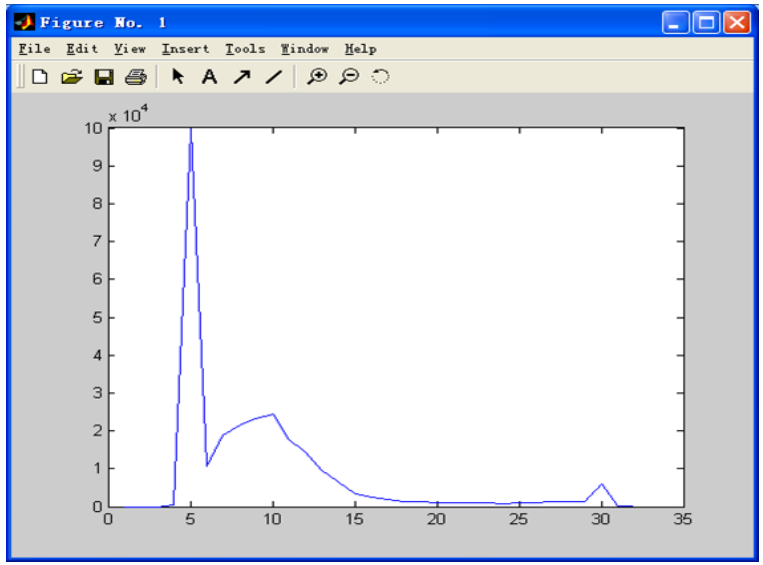
- s=imread('Fig0303(a)(breast).tif');
- hi=imhist(s,32);
- stem(hi,'b-o','fill');



Plotting Image Histograms using plot

- Function $plot$ plots a set of points by linking them with straight lines. The syntax is plot(horz, v, 'color_linestyle_marker')
- where the arguments are as defined previously for stem plots.
- Function $plot$ is used frequently to display transformation functions.

- s=imread('Fig0303(a)(breast).tif');
- hi=imhist(s,32);
- plot(hi);



Histogram Equalization

- Histogram equalization:
  - To improve the contrast of an image.
  - To transform an image in such a way that the transformed image has a nearly uniform distribution of pixel values.
- Transformation:
  - Assume r has been normalized to the interval [0,1], with $r = 0$ representing black and $r = 1$ representing white

$$s = T(r), 0 \leq r \leq 1$$

  - The transformation function satisfies the following conditions:
    - $T(r)$is single-valued and monotonically increasing in the interval $0 \leq r \leq 1$
    - $0 \leq T(r) \leq 1$ for $0 \leq r \leq 1$

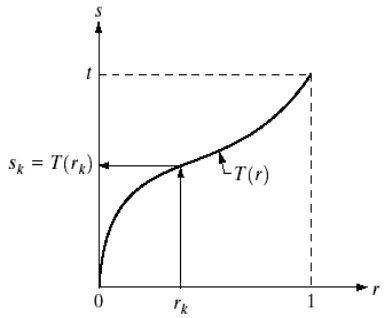


FIGURE 3.16 A gray-level transformation function that is both single valued and monotonically increasing.

Histogram Equalization

- Histogram equalization is based on a transformation of the probability density function of a random variable.
- Let $p_r(r)$ and $p_s(s)$ denote the probability density function of random variable r and s, respectively.
- If $p_r(r)$ and $T(r)$ are known, then the probability density function $p_s(s)$ of the transformed variable s can be obtained

$$p_s(s) = p_r(r)|\frac{dr}{ds}|$$

- Define a transformation function

$$s = T(r) = \int_0^r p_r(w)dw$$

  - where w is a dummy variable of integration and the right side of this equation is the cumulative distribution function of random variable r.

Histogram Equalization

- Given transformation function $T(r)$

$$s = T(r) = \int_0^r p_r(w)dw$$

$$\frac{ds}{dr} = \frac{dT(r)}{dr} = \frac{d[\int_0^r p_r(w)dw]}{dr} = p_r(r)$$

$$p_s(s) = p_r(r)|\frac{dr}{ds}| = p_r(r)|\frac{1}{p_r(r)}| = 1, 0 \leq s \leq 1$$

- $p_s(s)$ now is a uniform probability density function.
- $T(r)$ depends on $p_r(r)$, but the resulting $p_s(s)$ always is uniform.

Histogram Equalization

- In discrete version:
  - The probability of occurrence of gray level r_k in an image is

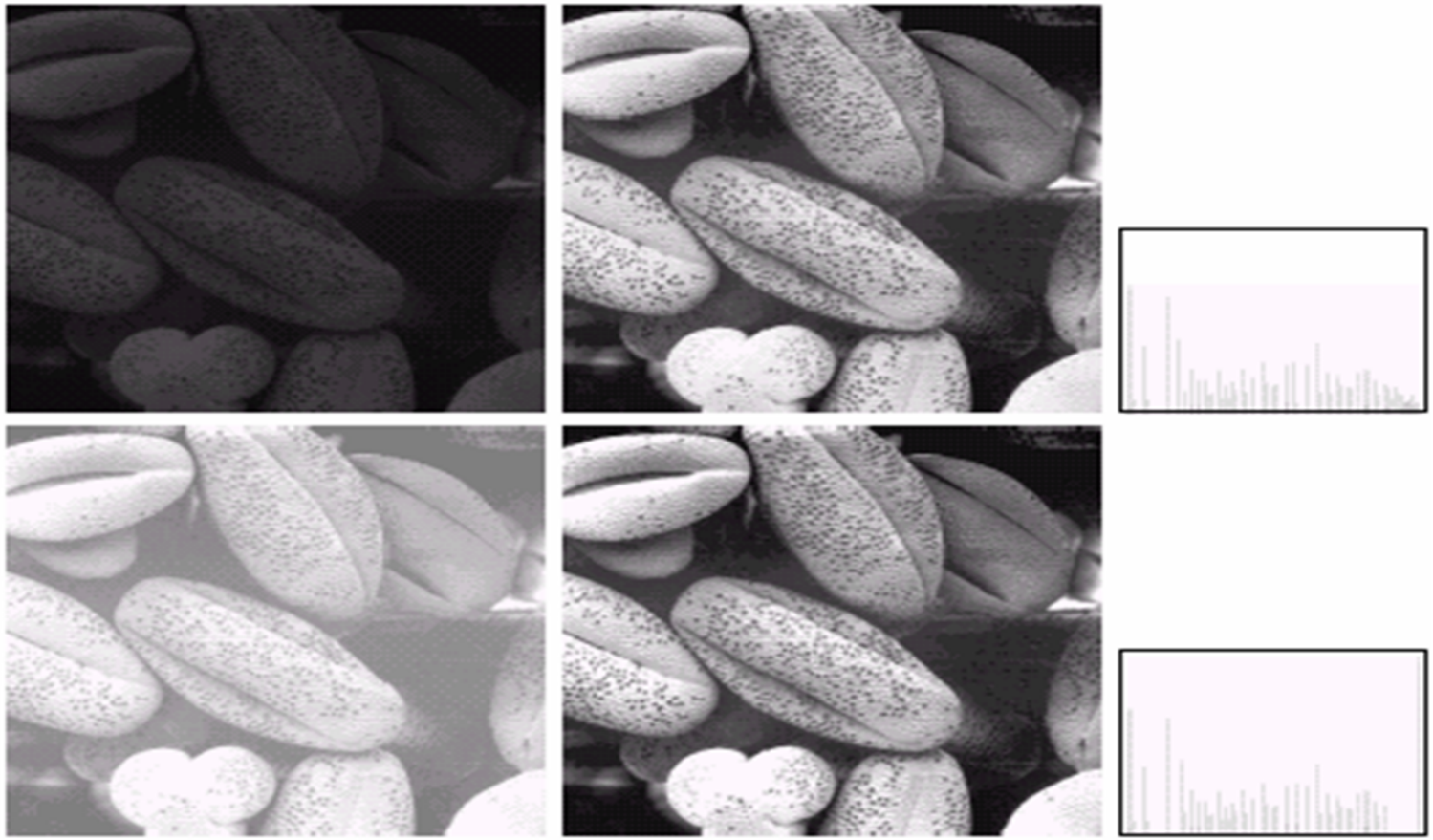
$$p_r(r) = \frac{n_k}{k}, k = 0, 1, 2, ..., L-1$$

  - The transformation function is

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n}, k = 0, 1, 2, ..., L-1$$

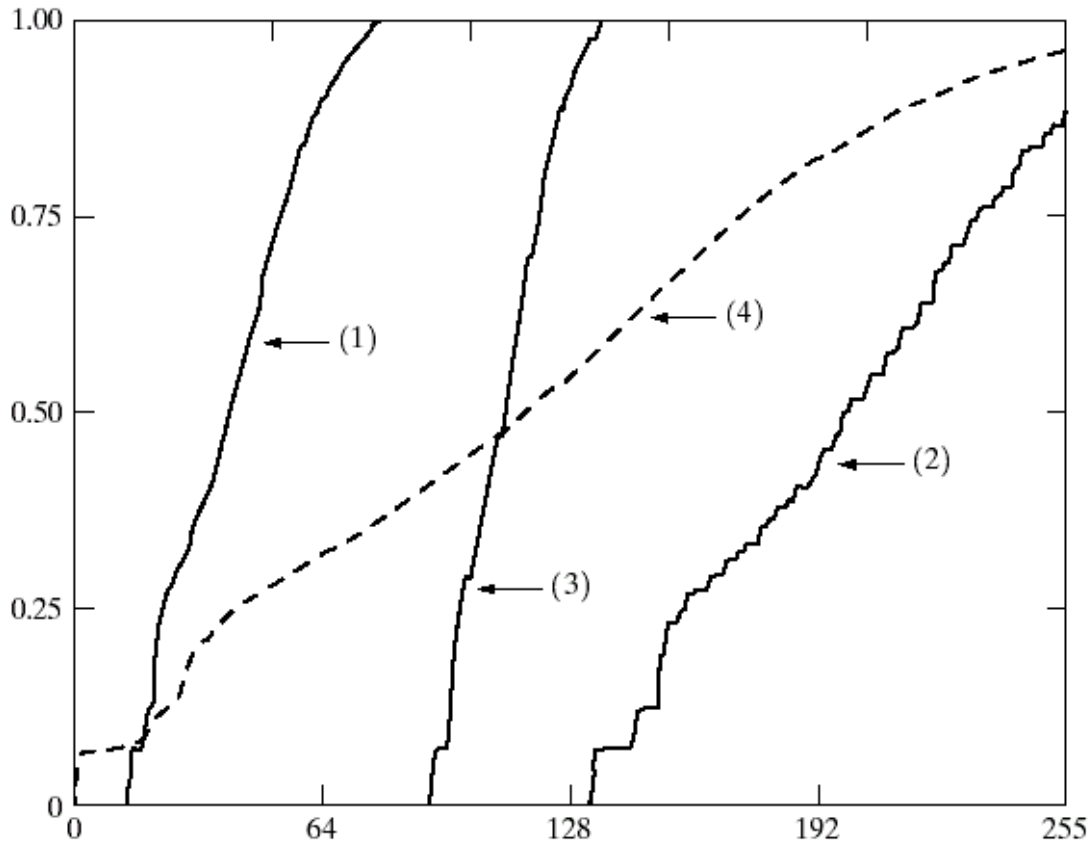
  - Thus, an output image is obtained by mapping each pixel with level r_k in the input image into a corresponding pixel with level s_k.

Histogram Equalization



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).



Histogram Equalization

- Histogram equalization is implemented in the toolbox by function histeq, which has the syntax

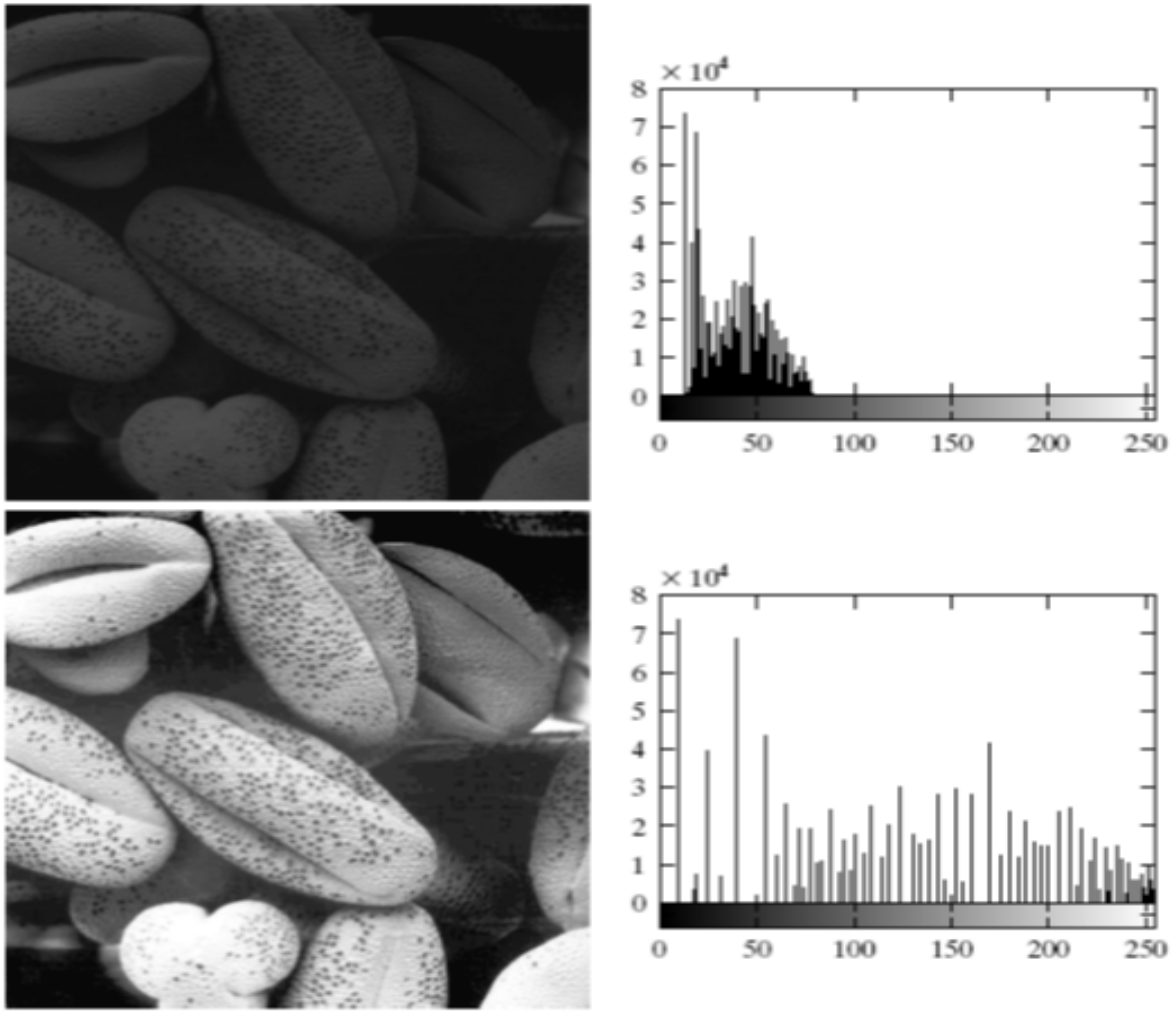
$$g = \text{histeq}(f, \text{nlev})$$

  - where f is the input image and nlev is the number of intensity levels specified for the output image.
  - If nlev is equal to L (the total number of possible levels in the input image), then histeq implements the transformation function directly. If nlev is less than L, then histeq attempts to distribute the levels so that they will approximate a flat histogram.

Histogram Equalization in Matlab

- >> f = imread('Fig0308(a)(pollen).tif');
- >> imshow(f);
- >> figure,imhist(f);
- >> ylim('auto');
- >> g = histeq(f,256);
- >> figure, imshow(g);
- >> figure, imhist(g);
- >> ylim('auto');

Histogram Equalization in Matlab

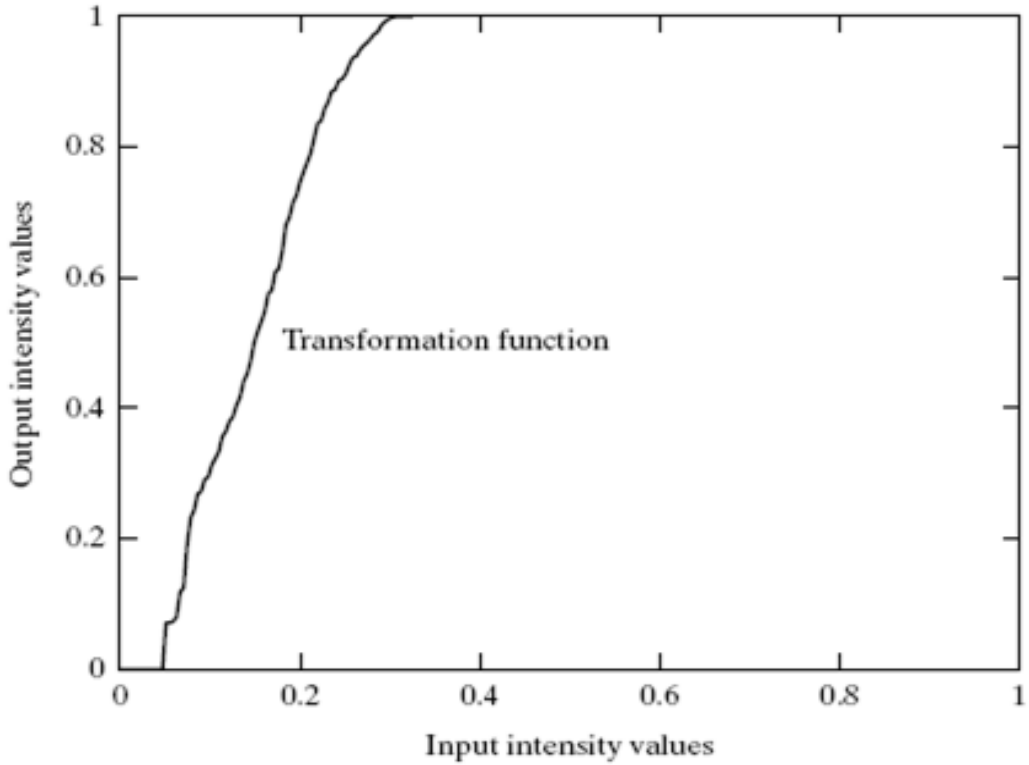


a b
c d

FIGURE 3.8 Illustration of histogram equalization. (a) Input image, and (b) its histogram. (c) Histogram-equalized image, and (d) its histogram. The improvement between (a) and (c) is quite visible. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra.)

Histogram Equalization

FIGURE 3.9
Transformation function used to map the intensity values from the input image in Fig. 3.8(a) to the values of the output image in Fig. 3.8(c).



Histogram Matching

- Histogram matching is similar to histogram equalization, except that instead of trying to make the output image have a flat histogram, we would like it to have a histogram of a specified shape.
- Consider for a moment continuous levels that are normalized to the interval [0, 1], and let r and z denote the intensity levels of the input and output images. The input levels have probability density function $p_r(r)$ and the output levels have the specified probability density function $p_z(z)$.

Histogram Matching

- We know from the discussion in the previous section that the transformation:

$$s = T(r) = \int_0^r p_r(w)dw$$

result in a ideal equalized histogram $p_s(s)$.

- Suppose now we define a variable z with the property

$$H(z) = \int_0^z p_z(w)dw = s$$

- From the preceding two equations, it follows that

$$z = H^{-1}(s) = H^{-1}(T(r));$$

- We can find $T(r)$ from the input image (this is the histogram equalization transformation discussed in the previous section), so it follows that we can use the preceding equation to find the transformed levels z whose PDF is the specified $p_z(z)$ as long as we can find H^{-1}.

Histogram Matching

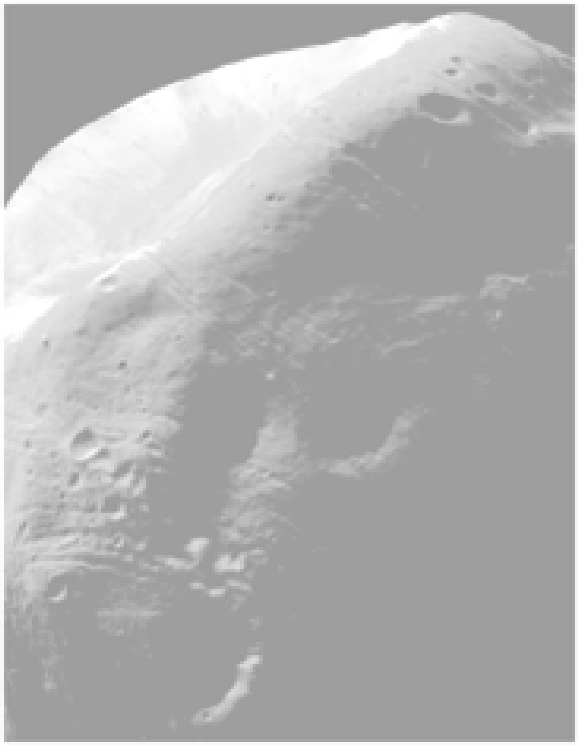
- The toolbox implements histogram matching using the following syntax in histeq:

$$g = \text{histeq}(f, \text{hspec})$$

  - where f is the input image, hspec is the specified histogram (a row vector of specified values), and g is the output image, whose histogram approximates the specified histogram, hspec.
- This vector should contain integer counts corresponding to equally spaced bins. A property of histeq is that the histogram of g generally better matches hspec when length (hspec) is much smaller than the number of intensity levels in f.

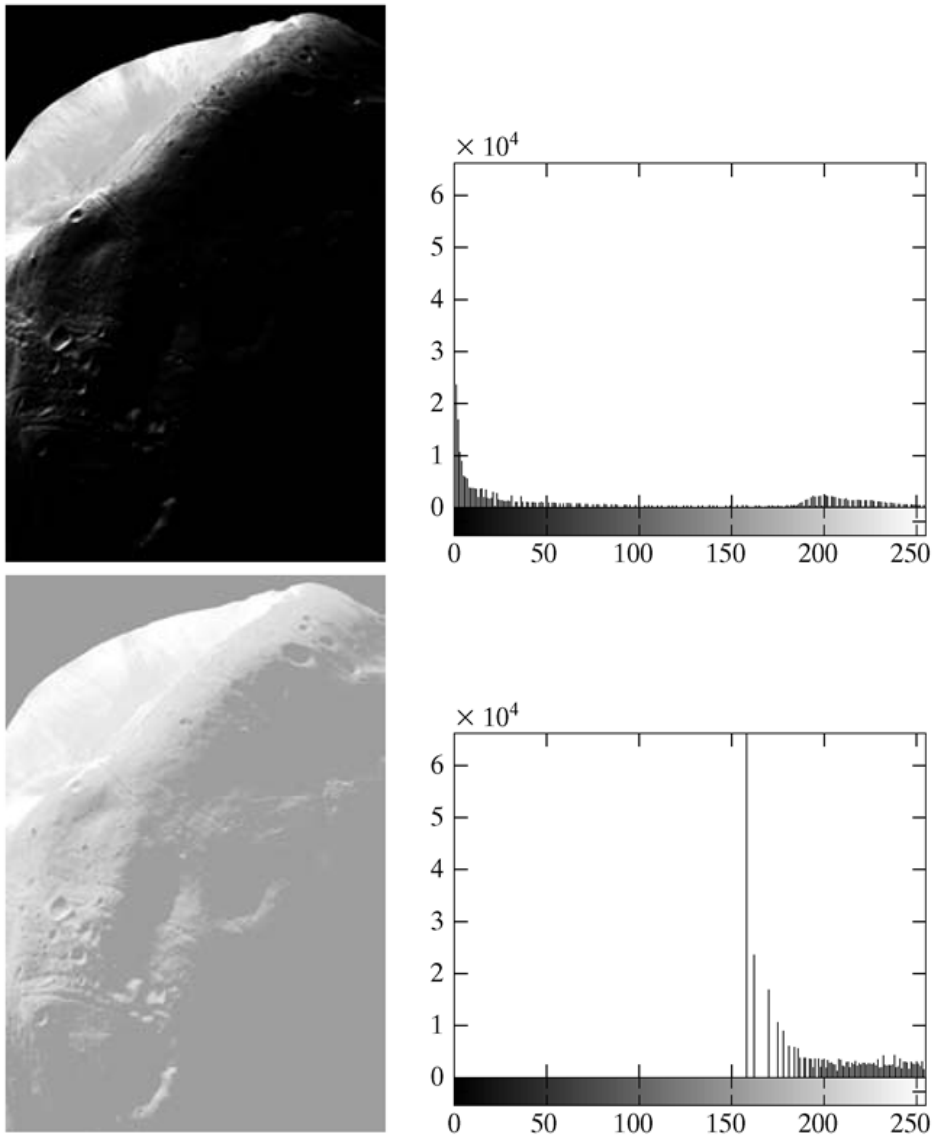
Histogram Matching

- f=imread('Fig0310(a)(Moon Phobos).tif');
- f1=histeq(f,256);
- imshow(f1);



- It shows that histogram equalization in fact did not produce a particularly good result in this case.
- The reason for this can be seen by studying the histogram of the equalized image.

Histogram Matching



a b
c d

FIGURE 3.10
(a) Image of the Mars moon Phobos.
(b) Histogram.
(c) Histogram-equalized image.
(d) Histogram of (c).
(Original image courtesy of NASA).

Histogram Matching

- One possibility for remedying this situation is to use histogram matching, with the desired histogram having a lesser concentration of components in the low end of the gray scale, and maintaining the general shape of the histogram of the original image.
- We note that the histogram of original image is basically bimodal, with one large mode at the origin, and another, smaller, mode at the high end of the gray scale. These types of histograms can be modeled, for example, by using multimodal Gaussian functions.

Histogram Matching

- The following M-function computes a bimodal Gaussian function normalized to unit area, so it can be used as a specified histogram.
  - Function twomodegauss:

$$p(x) = k + \frac{A_1}{\sqrt{2\pi}\sigma_1}\exp(-\frac{(x-m_1)^2}{2\sigma_1^2}) + \frac{A_2}{\sqrt{2\pi}\sigma_2}\exp(-\frac{(x-m_2)^2}{2\sigma_2^2})$$

- The following interactive function accepts inputs from a keyboard and plots the resulting Gaussian function. Refer to Section 2.10.5 for an explanation of the functions input and str2num. Note how the limits of the plots are set.
  - Function manualhist

Histogram Matching

- Since the problem with histogram equalization in this example is due primarily to a large concentration of pixels in the original image with levels near 0,a reasonable approach is to modify the histogram of that image so that it does not have this property.
- Figure 3.11(a) shows a plot of a function that preserves the general shape of the original histogram, but has a smoother transition of levels in the dark region of the intensity scale. The output of the program, p, consists of 256 equally spaced points from this function and is the desired specified histogram.

Histogram Matching

- An image with the specified histogram was generated using the command

  >> g = histeq(f, p);

  all commands are:

  >> f=imread('Fig0310(a)(Moon Phobos).tif');

  >> g=histeq(f,manualhist);

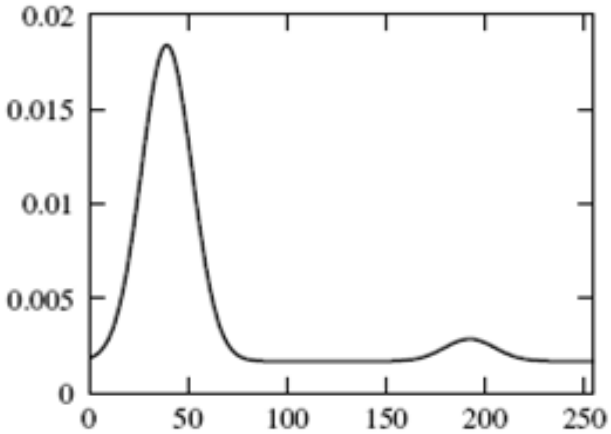
  Enter m1, sig1, m2, sig2, A1, A2, k OR x to quit:x;

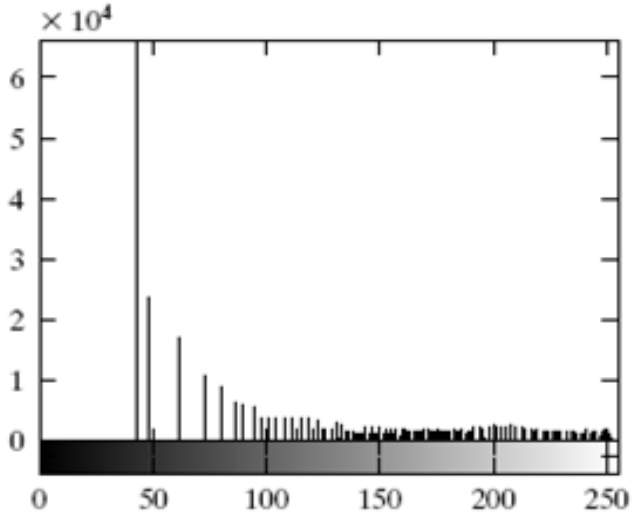
  >> imshow(g);

Histogram Matching

a b
c

FIGURE 3.11
(a) Specified histogram.
(b) Result of enhancement by histogram matching.
(c) Histogram of (b).





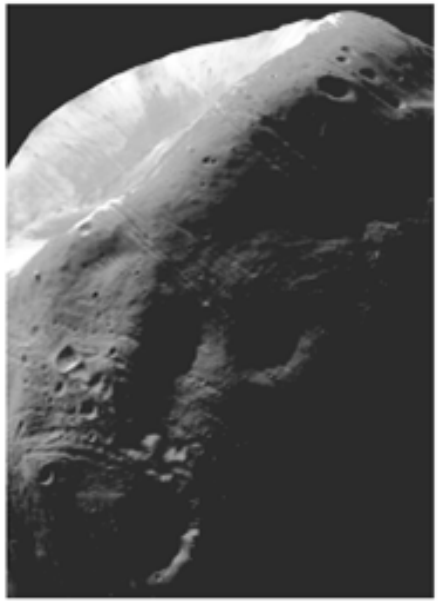


Image Processing and Analysis
Lecture 4、Image Enhencement in Spatial Filtering (II)

Weiqiang Wang

School of Computer and Control Engineering, UCAS

October 27, 2015

Outline

Outline

Outline

Histogram Matching

- Histogram matching is similar to histogram equalization, except that instead of trying to make the output image have a flat histogram, we would like it to have a histogram of a specified shape.
- Consider for a moment continuous levels that are normalized to the interval [0, 1], and let r and z denote the intensity levels of the input and output images. The input levels have probability density function $p_r(r)$ and the output levels have the specified probability density function $p_z(z)$.

Histogram Matching

- We know from the discussion in the previous section that the transformation:

$$s = T(r) = \int_0^r p_r(w)dw$$

result in a ideal equalized histogram $p_s(s)$.

- Suppose now we define a variable z with the property

$$H(z) = \int_0^z p_z(w)dw = s$$

- From the preceding two equations, it follows that

$$z = H^{-1}(s) = H^{-1}(T(r));$$

- We can find $T(r)$ from the input image (this is the histogram equalization transformation), so it follows that we can use the preceding equation to find the transformed levels z whose PDF is the specified $p_z(z)$ as long as we can find H^{-1}.

Histogram Matching

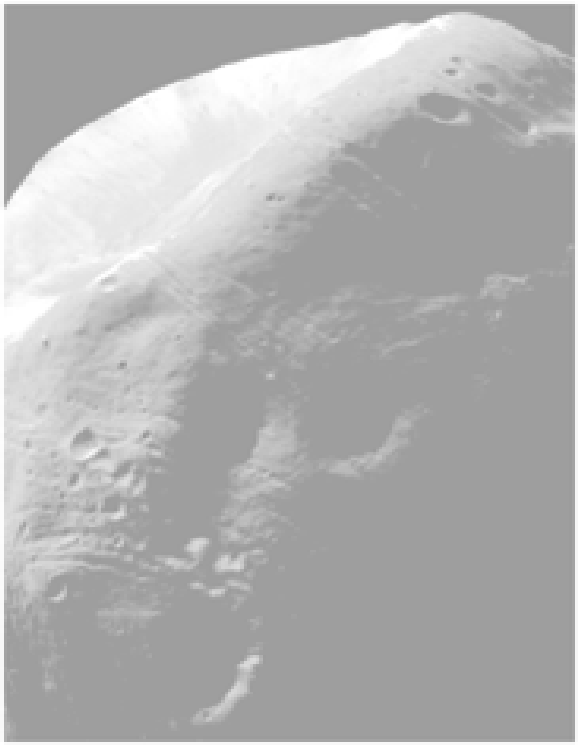
- The toolbox implements histogram matching using the following syntax in histeq:

$$g = \text{histeq(f, hspec)}$$

  - where f is the input image, hspec is the specified histogram (a row vector of specified values), and g is the output image, whose histogram approximates the specified histogram, hspec.
- This vector should contain integer counts corresponding to equally spaced bins. A property of histeq is that the histogram of g generally better matches hspec when length (hspec) is much smaller than the number of intensity levels in f.

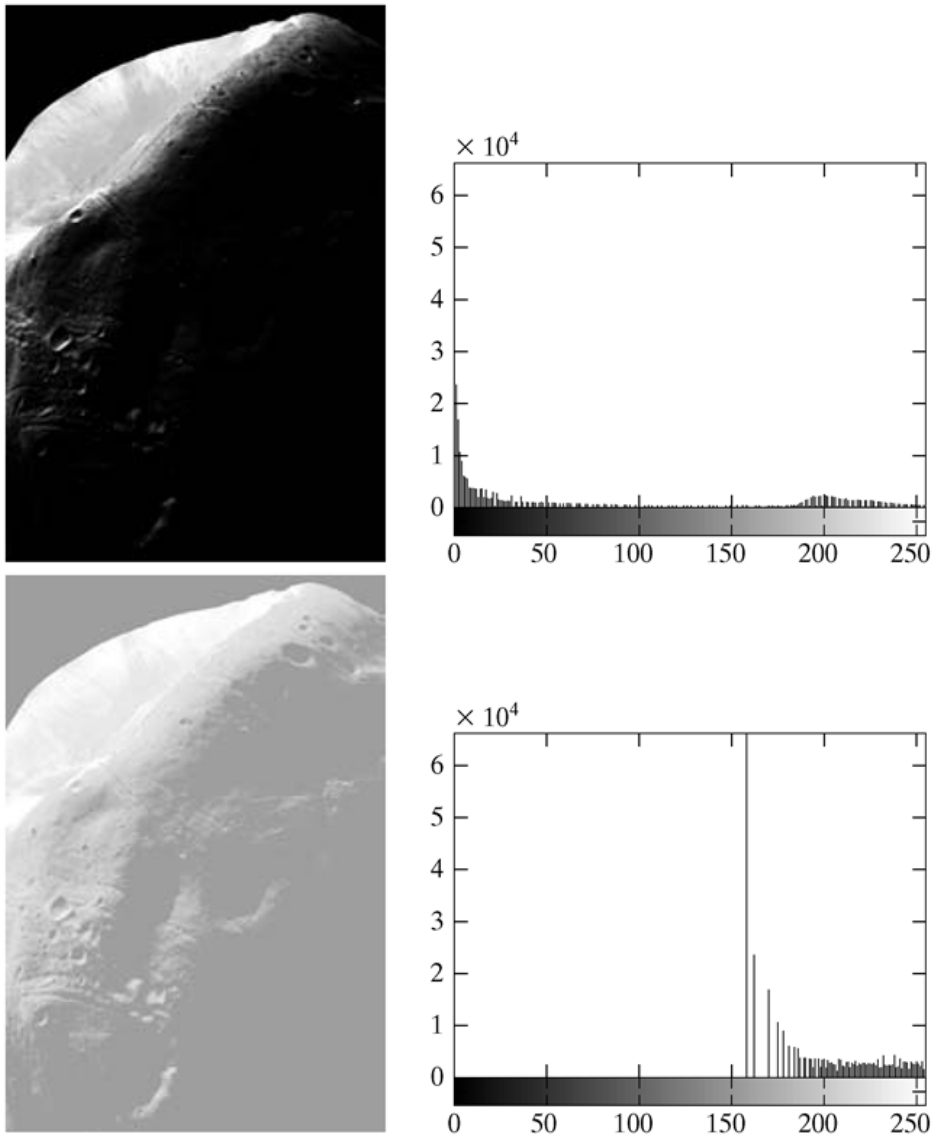
Histogram Matching

- f=imread('Fig0310(a)(Moon Phobos).tif');
- f1=histeq(f,256);
- imshow(f1);



- It shows that histogram equalization in fact did not produce a particularly good result in this case.
- The reason for this can be seen by studying the histogram of the equalized image.

Histogram Matching



a b
c d

FIGURE 3.10
(a) Image of the Mars moon Phobos.
(b) Histogram.
(c) Histogram-equalized image.
(d) Histogram of (c).
(Original image courtesy of NASA).

Histogram Matching

- One possibility for remedying this situation is to use histogram matching, with the desired histogram having a lesser concentration of components in the low end of the gray scale, and maintaining the general shape of the histogram of the original image.
- We note that the histogram of original image is basically bimodal, with one large mode at the origin, and another, smaller, mode at the high end of the gray scale. These types of histograms can be modeled, for example, by using multimodal Gaussian functions.

Histogram Matching

- The following M-function computes a bimodal Gaussian function normalized to unit area, so it can be used as a specified histogram.
  - Function twomodegauss:

$$p(x) = k + \frac{A_1}{\sqrt{2\pi}\sigma_1}\exp(-\frac{(x-m_1)^2}{2\sigma_1^2}) + \frac{A_2}{\sqrt{2\pi}\sigma_2}\exp(-\frac{(x-m_2)^2}{2\sigma_2^2})$$

- The following interactive function accepts inputs from a keyboard and plots the resulting Gaussian function. Refer to Section 2.10.5 for an explanation of the functions input and str2num. Note how the limits of the plots are set.
  - Function manualhist

Histogram Matching

- Since the problem with histogram equalization in this example is due primarily to a large concentration of pixels in the original image with levels near 0, a reasonable approach is to modify the histogram of that image so that it does not have this property.
- Figure 3.11(a) shows a plot of a function that preserves the general shape of the original histogram, but has a smoother transition of levels in the dark region of the intensity scale. The output of the program, p, consists of 256 equally spaced points from this function and is the desired specified histogram.

Histogram Matching

- An image with the specified histogram was generated using the command

  >> g = histeq(f, p);

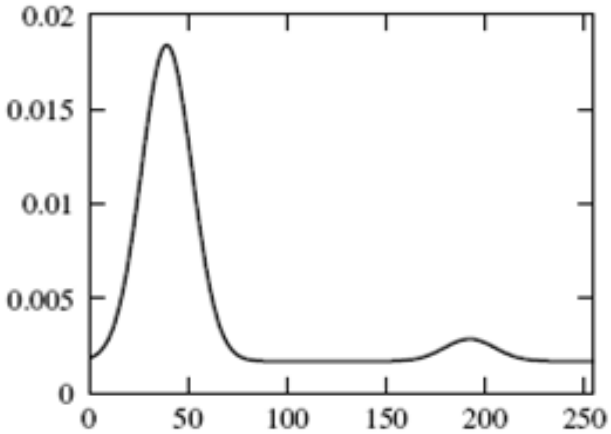
  all commands are:

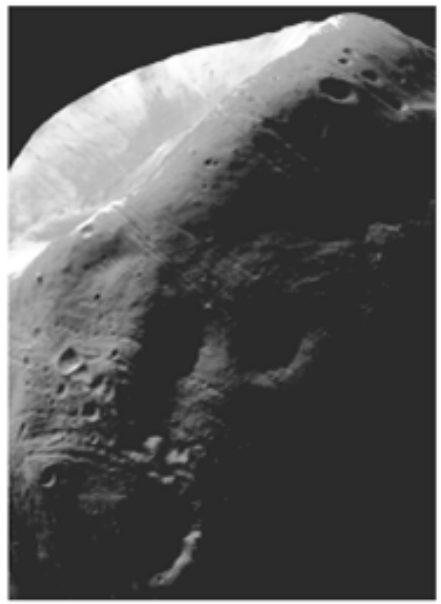
  ```
  >> f=imread('Fig0310(a)(Moon Phobos).tif');
  >> g=histeq(f,manualhist);
  Enter m1, sig1, m2, sig2, A1, A2, k OR x to quit:x;
  >> imshow(g);
  ```

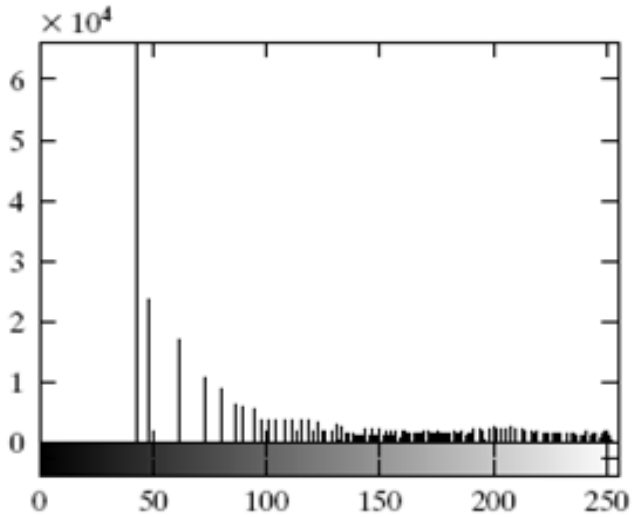
Histogram Matching

a b
c

FIGURE 3.11
(a) Specified histogram.
(b) Result of enhancement by histogram matching.
(c) Histogram of (b).







Spatial Filtering

- In spatial filtering (vs. frequency domain filtering), the output image is computed directly by simple calculations on the pixels of the input image.
- Spatial filtering can be either linear or non-linear.
- For each output pixel, some neighborhood of input pixels is used in the computation.

Linear Spatial Filtering

- In general, linear filtering of an image f of size $M \times N$ with a filter mask of size $m \times n$ is given by

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x+s, y+t)$$

where $a = (m-1)/2$ and $b = (n-1)/2$

- This concept called convolution. Filter masks are sometimes called convolution masks or convolution kernels.

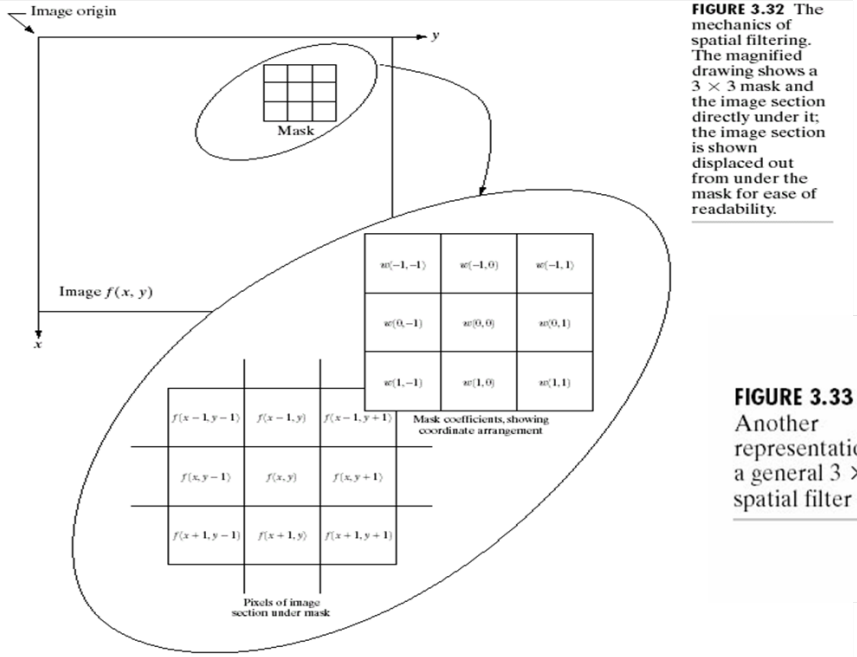


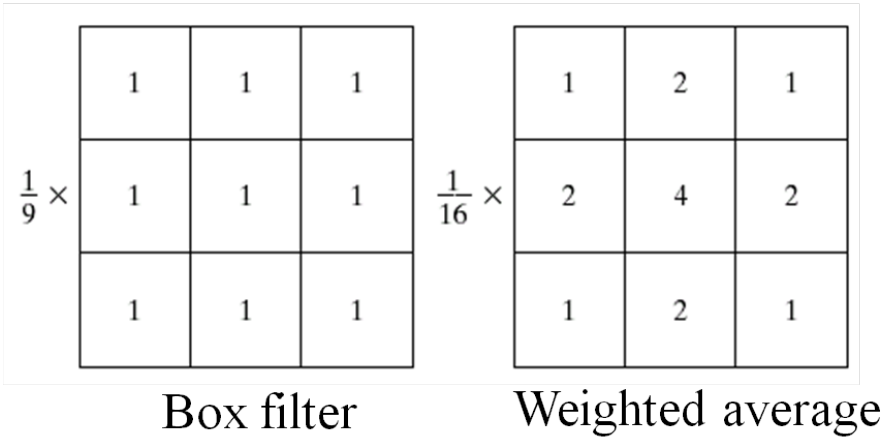
FIGURE 3.32 The mechanics of spatial filtering. The magnified drawing shows a 3×3 mask and the image section directly under it; the image section is shown displaced out from under the mask for ease of readability.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

FIGURE 3.33 Another representation of a general 3×3 spatial filter mask.

Linear Spatial Filtering(cont.)

- Smoothing linear filters
  - Averaging filters (Lowpass filters in Chapter 4))
    - Box filter
    - Weighted average filter



a b

FIGURE 3.34 Two 3×3 smoothing (averaging) filter masks. The constant multiplier in front of each mask is equal to the sum of the values of its coefficients, as is required to compute an average.

Linear Spatial Filtering(cont.)

- There are two closely related concepts that must be understood clearly when performing linear spatial filtering. One is correlation; the other is convolution.
- Correlation is the process of passing the mask ω by the image array f in the manner described in Fig. 3.32.
- Mechanically, convolution is the same process, except that ω is rotated by 180 degree prior to passing it by f.
- These two concepts are best explained by some simple examples.

Linear Spatial Filtering(cont.)

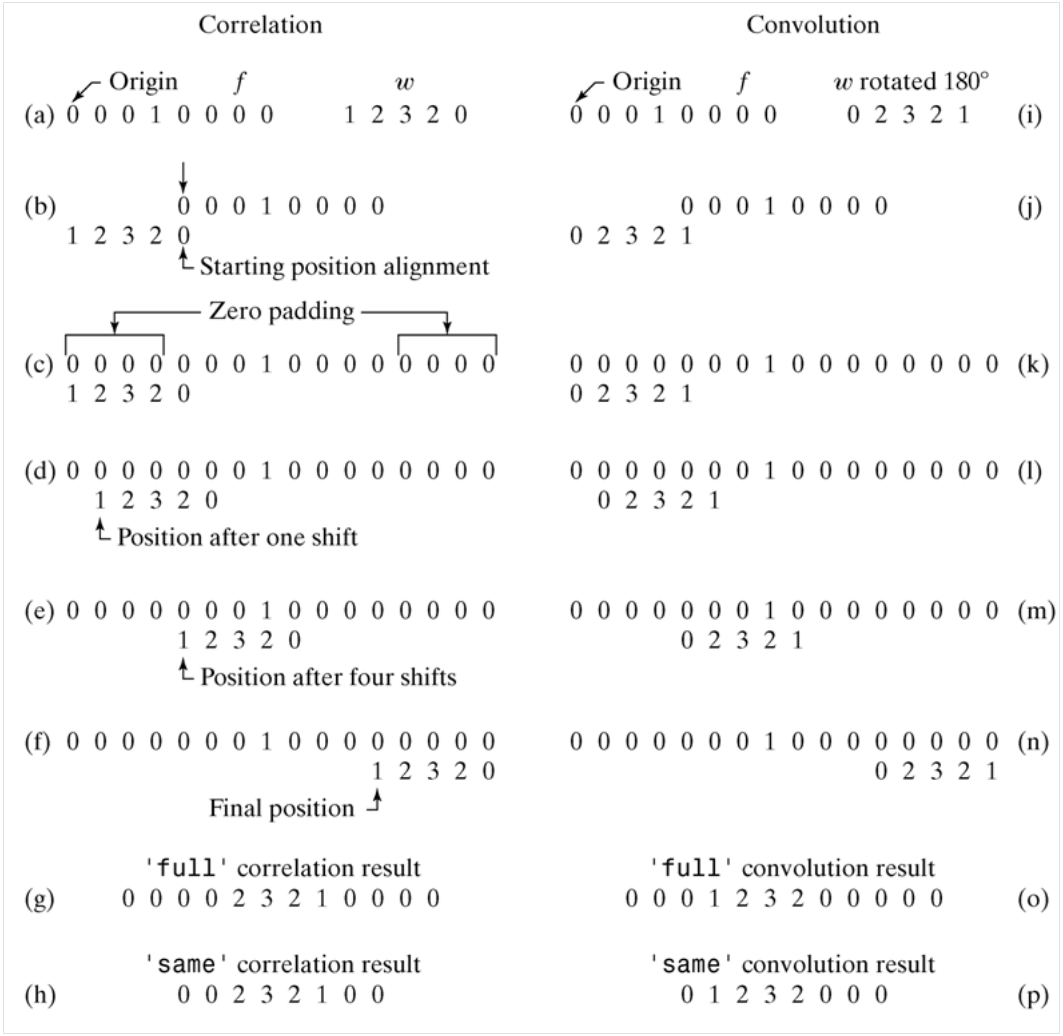


FIGURE 3.13 Illustration of one-dimensional correlation and convolution.

Linear Spatial Filtering(cont.)

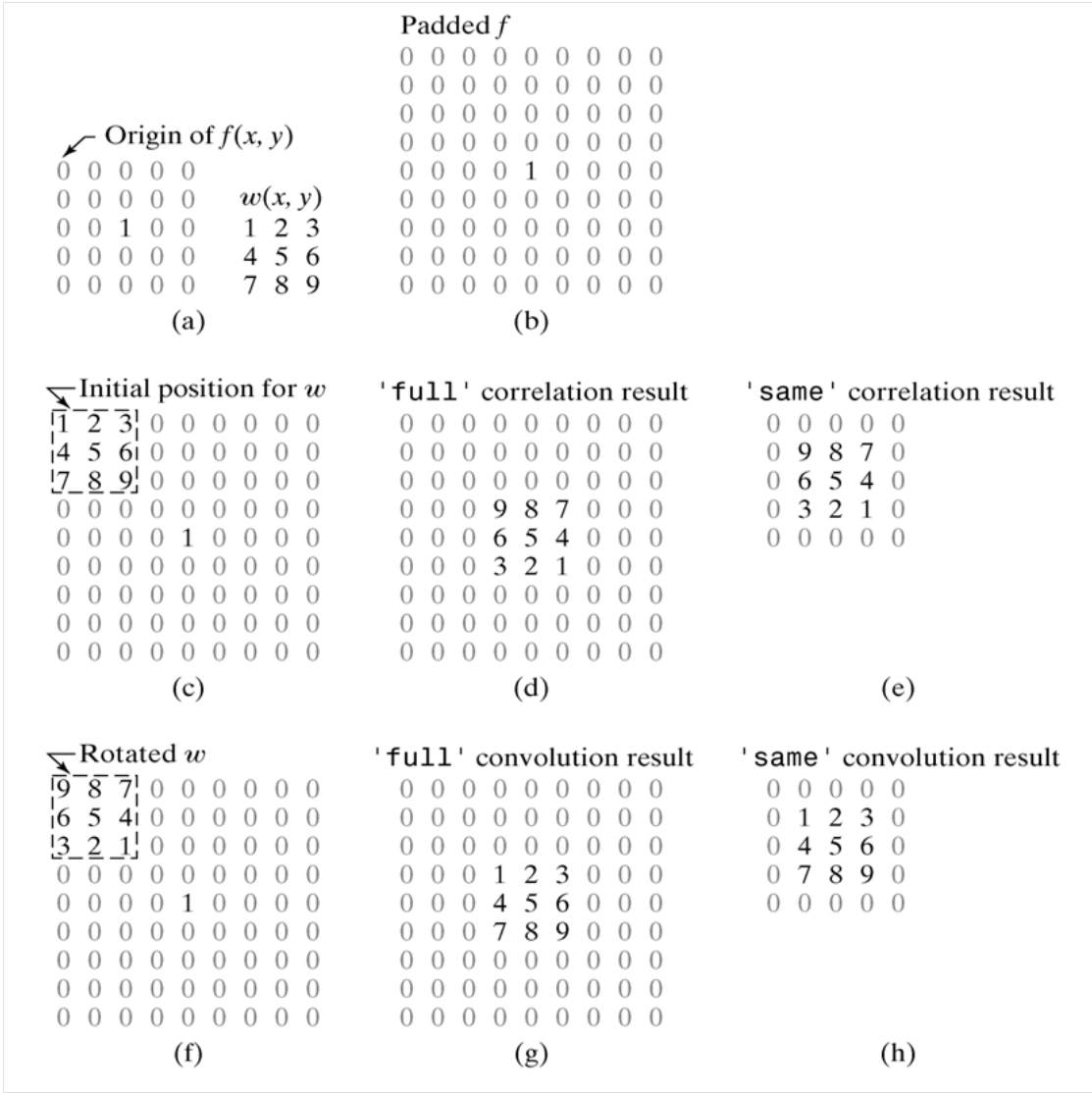


FIGURE 3.14 Illustration of two-dimensional correlation and convolution. The 0s are shown in gray to simplify viewing.

Linear Spatial Filtering(cont.)

- The toolbox implements linear spatial filtering using function imfilter, which has the following syntax:

  g = imfilter(f, w, filtering_mode, boundary_options, size_options)

  where f is the input image, w is the filter mask, g is the filtered result, and the other parameters are summarized in Table 3.2.

Options	Description
Filtering Mode	
`'corr'`	Filtering is done using correlation (see Figs. 3.13 and 3.14). This is the default.
`'conv'`	Filtering is done using convolution (see Figs. 3.13 and 3.14).
Boundary Options	
`P`	The boundaries of the input image are extended by padding with a value, `P` (written without quotes). This is the default, with value 0.
`'replicate'`	The size of the image is extended by replicating the values in its outer border.
`'symmetric'`	The size of the image is extended by mirror-reflecting it across its border.
`'circular'`	The size of the image is extended by treating the image as one period a 2-D periodic function.
Size Options	
`'full'`	The output is of the same size as the extended (padded) image (see Figs. 3.13 and 3.14).
`'same'`	The output is of the same size as the input. This is achieved by limiting the excursions of the center of the filter mask to points contained in the original image (see Figs. 3.13 and 3.14). This is the default.

Linear Spatial Filtering(cont.)

- Each element of the filtered image is computed using double-precision, floating-point arithmetic. However, imfilter converts the output image to the same class of the input.Therefore, if input is an integer array, then output elements that exceed the range of the integer type are truncated, and fractional values are rounded. If more precision is desired in the result, then should be converted to class double by using im2double or double before using imfilter.
  - f=imread('Fig0315(a)(original_test_pattern).tif');
  - f=im2double(f);
  - w=ones(31);
  - gd = imfilter(f,w);
  - imshow(gd,[])

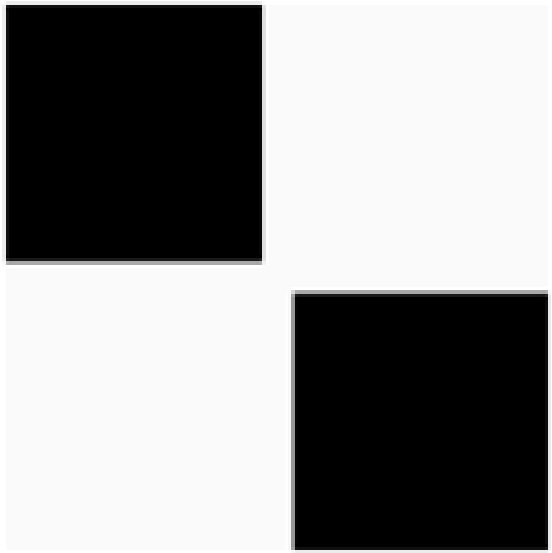
Linear Spatial Filtering(cont.)

- >>gr = imfilter(f, w, 'replicate');
- >>figure, imshow(gr, [ ])
- >>gc = imfilter(f, w, 'circular');
- >>figure, imshow(gc, [ ])

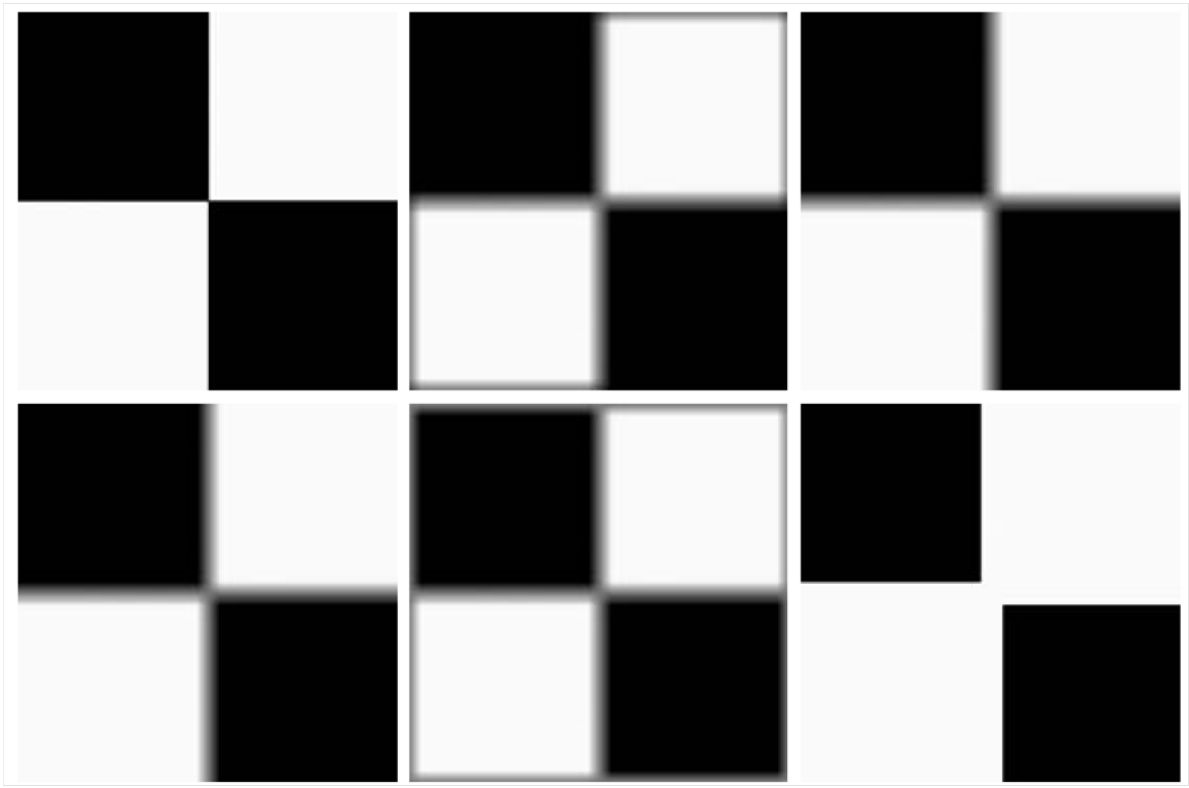
Please help me choose the corresponding output!!

Linear Spatial Filtering(cont.)

- f=imread('Fig0315(a)(original_test_pattern).tif');
- w=ones(31);
- gd = imfilter(f,w);
- imshow(gd,[])



Linear Spatial Filtering(cont.)



a b c
d e f

FIGURE 3.15
(a) Original image. (b) Result of using `imfilter` with default zero padding. (c) Result with the `'replicate'` option. (d) Result with the `'symmetric'` option. (e) Result with the `'circular'` option. (f) Result of converting the original image to class `uint8` and then filtering with the `'replicate'` option. A filter of size 31×31 with all 1s was used throughout.

Nonlinear Spatial Filtering

- Nonlinear spatial filtering usually uses a neighborhood too, but some other mathematical operations are used. For example, letting the response at each center point be equal to the maximum pixel value in its neighborhood is a nonlinear filtering operation.
- Another basic difference is that the concept of a mask is not as prevalent in nonlinear processing.
- The toolbox provides two functions for performing general nonlinear filtering: nlfilter and colfilt.
- The former performs operations directly in 2-D. use the command open nlfilter to see the source code.
- colfilt organizes the data in the form of columns. requires more memory, but executes significantly faster than nlfilter.

Nonlinear Spatial Filtering(cont.)

- Given an input image, f, of size $M \times N$ and a neighborhood of size $m \times n$, function colfilt generates a matrix, call it A, of maximum size $mn \times MN$
- each column corresponds to the pixels encompassed by the neighborhood centered at a location in the image.
- For example, the first column corresponds to the pixels encompassed by the neighborhood when its center is located at the top, leftmost point in f.
- The former performs operations directly in 2-D. use the command open nlfilter to see the source code.
- colfilt organizes the data in the form of columns. requires more memory, but executes significantly faster than nlfilter.

Nonlinear Spatial Filtering(cont.)

- The syntax of function colfilt is

  g = colfilt(f, [m n], 'sliding', @fun, parameters)

  - where, m and n are the dimensions of the filter region
  - 'sliding' indicates that the process is one of sliding the region from pixel to pixel in the input image f
  - @fun references a function, which we denote arbitrarily as fun
  - parameters indicates parameters (separated by commas) that may be required by function fun.
- Because of the way in which matrix A is organized, function fun must operate on each of the columns of A individually and return a row vector, g, containing the results for all the columns.
- The kth element of g is the result of the operation performed by fun on the kth column of A.

Nonlinear Spatial Filtering(cont.)

- When using colfilt, the input image must be padded explicitly before filtering. For this we use function padarray, which, for 2-D functions, has the syntax

  fp = padarray(f, [r c], method, direction)

  where f is the input image, fp is the padded image, [r c] gives the number of rows and columns by which to pad f, and method and direction are as explained in Table 3.3.

TABLE 3.3 Options for function `padarray`.

Options	Description
Method	
`'symmetric'`	The size of the image is extended by mirror-reflecting it across its border.
`'replicate'`	The size of the image is extended by replicating the values in its outer border.
`'circular'`	The size of the image is extended by treating the image as one period of a 2-D periodic function.
Direction	
`'pre'`	Pad before the first element of each dimension.
`'post'`	Pad after the last element of each dimension.
`'both'`	Pad before the first element and after the last element of each dimension. This is the default.

Linear Spatial Filtering

- The toolbox supports a number of predefined 2-D linear spatial filters, obtained by using function fspecial, which generates a filter mask, w, using the syntax

  w = fspecial('type', parameters)

  where 'type' specifies the filter type, and parameters further define the specified filter.

Type	Syntax and Parameters
'average'	fspecial('average', [r c]). A rectangular averaging filter of size r × c. The default is 3 × 3. A single number instead of [r c] specifies a square filter.
'disk'	fspecial('disk', r). A circular averaging filter (within a square of size 2r + 1) with radius r. The default radius is 5.
'gaussian'	fspecial('gaussian', [r c], sig). A Gaussian lowpass filter of size r × c and standard deviation sig (positive). The defaults are 3 × 3 and 0.5. A single number instead of [r c] specifies a square filter.
'laplacian'	fspecial('laplacian', alpha). A 3 × 3 Laplacian filter whose shape is specified by alpha, a number in the range [0, 1]. The default value for alpha is 0.5.
'log'	fspecial('log', [r c], sig). Laplacian of a Gaussian (LoG) filter of size r × c and standard deviation sig (positive). The defaults are 5 × 5 and 0.5. A single number instead of [r c] specifies a square filter.
'motion'	fspecial('motion', len, theta). Outputs a filter that, when convolved with an image, approximates linear motion (of a camera with respect to the image) of len pixels. The direction of motion is theta, measured in degrees, counterclockwise from the horizontal. The defaults are 9 and 0, which represents a motion of 9 pixels in the horizontal direction.
'prewitt'	fspecial('prewitt'). Outputs a 3 × 3 Prewitt mask, wv, that approximates a vertical gradient. A mask for the horizontal gradient is obtained by transposing the result: wh = wv'.
'sobel'	fspecial('sobel'). Outputs a 3 × 3 Sobel mask, sv, that approximates a vertical gradient. A mask for the horizontal gradient is obtained by transposing the result: sh = sv'.
'unsharp'	fspecial('unsharp', alpha). Outputs a 3 × 3 unsharp filter. Parameter alpha controls the shape; it must be greater than 0 and less than or equal to 1.0; the default is 0.2.

Sharpening Spatial Filters

- The principal objective of sharping is to highlight fine detail in an image or to enhance detail that has been blurred.
- sharping is generally accomplished by spatial differentiation.
- The derivatives of a digital function are defined in terms of differences.
- For a first derivative, we require that it must satisfy:
  1. must be zero in flat segments(areas of constant gray-level values)
  2. must be nonzero at the onset of a gray-level step or ramp
  3. must be nonzero along ramps
- For a second derivative, we require that it must satisfy:
  1. must be zero in flat segments(areas of constant gray-level values).
  2. must be nonzero at the onset and end of a gray-level step or ramp.
  3. must be zero along ramps of constant slope.

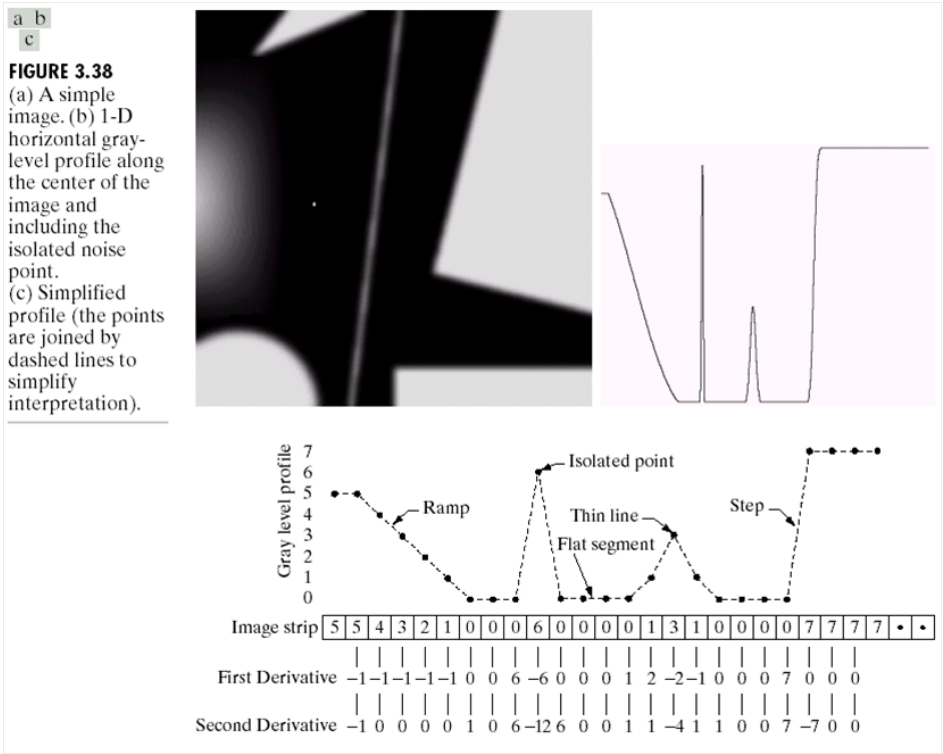
Sharpening Spatial Filters(cont.)

- A basic definition of the first-order derivative of a one-dimensional function $f(x)$ is the difference

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

- Similarly, we define the second derivative as

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



Laplacian Filter

- Laplacian function is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)$$

- Function fspecial('laplacian', alpha) implements a more general Laplacian mask:

$$\begin{pmatrix} \frac{\alpha}{1+\alpha}, & \frac{1-\alpha}{1+\alpha}, & \frac{\alpha}{1+\alpha} \\ \frac{1-\alpha}{1+\alpha}, & \frac{-4}{1+\alpha}, & \frac{1-\alpha}{1+\alpha} \\ \frac{\alpha}{1+\alpha}, & \frac{1-\alpha}{1+\alpha}, & \frac{\alpha}{1+\alpha} \end{pmatrix}$$

- We now proceed to enhance the image in Fig. 3.16(a) using the Laplacian. This image is a mildly blurred image of the North Pole of the moon. Enhancement in this case consists of sharpening the image, while preserving as much of its gray tonality as possible.

Image Enhancement with Laplacian Filters

- $g(x,y) = f(x,y) - \nabla^2 f(x,y)$

a b
c d

FIGURE 3.16
(a) Image of the North Pole of the moon.
(b) Laplacian filtered image, using `uint8` formats.
(c) Laplacian filtered image obtained using `double` formats.
(d) Enhanced result, obtained by subtracting (c) from (a).
(Original image courtesy of NASA.)

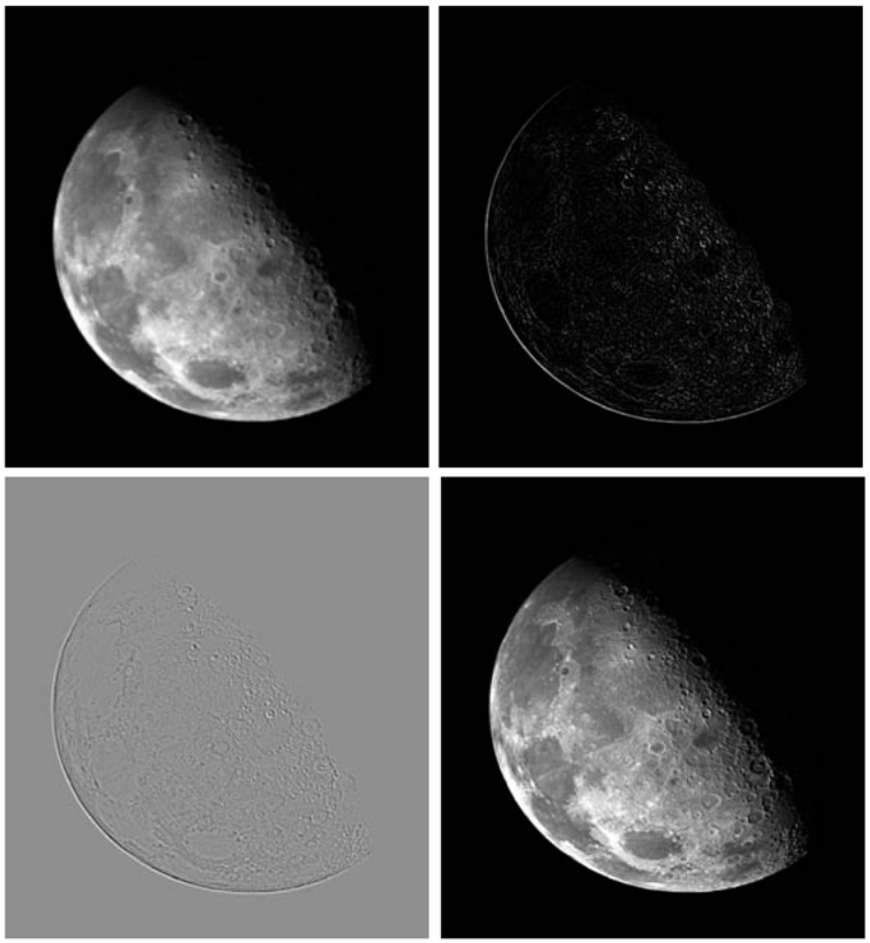
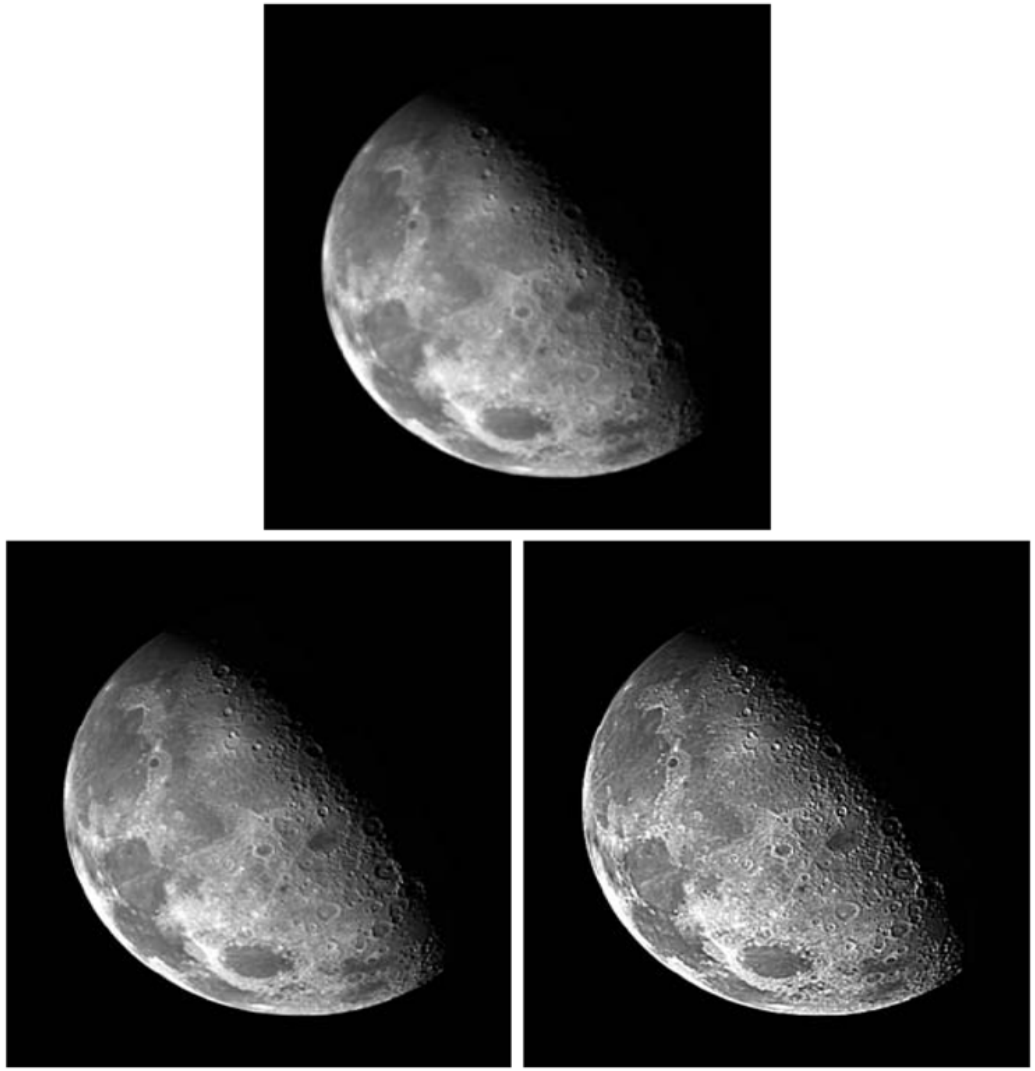


Image Enhancement with Laplacian Filters(cont.)

- Enhancement problems often require the specification of filters beyond those available in the toolbox. The Laplacian is a good example. The toolbox supports a 3×3 Laplacian filter with a -4 in the center. Usually, sharper enhancement is obtained by using the 3×3 Laplacian filter that has a -8 in the center and is surrounded by 1s, as discussed earlier.

```
>>f = imread('Fig0316(a)(moon).tif');
>>w4 = fspecial('laplacian', 0);
>>w8 = [1 1 1; 1 -8 1; 1 1 1];
>>f = im2double(f);
>>g4 = f-imfilter(f, w4, 'replicate');
>>g8 = f-imfilter(f, w8, 'replicate');
>>imshow(f)
>>figure, imshow(g4)
>>figure, imshow(g8)
```

Image Enhancement with Laplacian Filters(cont.)



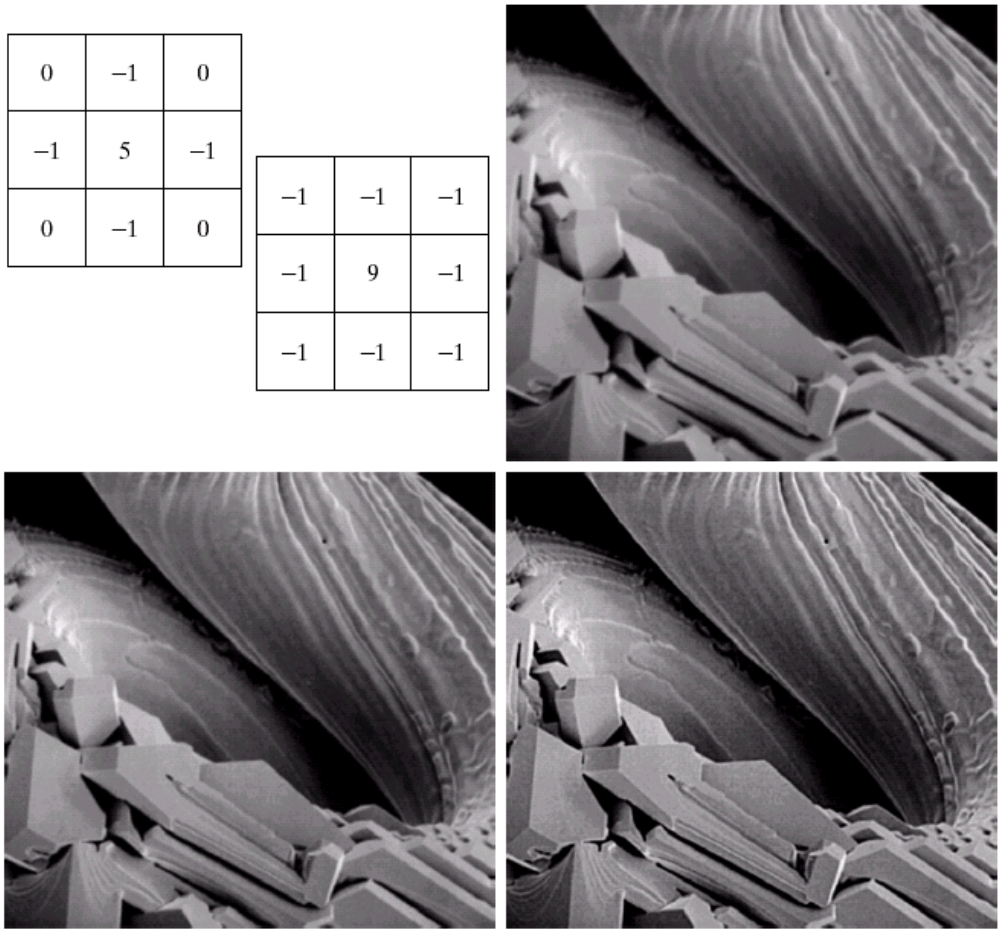
a
b c

FIGURE 3.17 (a) Image of the North Pole of the moon. (b) Image enhanced using the Laplacian filter `'laplacian'`, which has a −4 in the center. (c) Image enhanced using a Laplacian filter with a −8 in the center.

Image Enhancement with Laplacian Filters(cont.)

Simplification:

$$g(x,y) = f(x,y) - \nabla^2 f(x,y) = 5f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)]$$



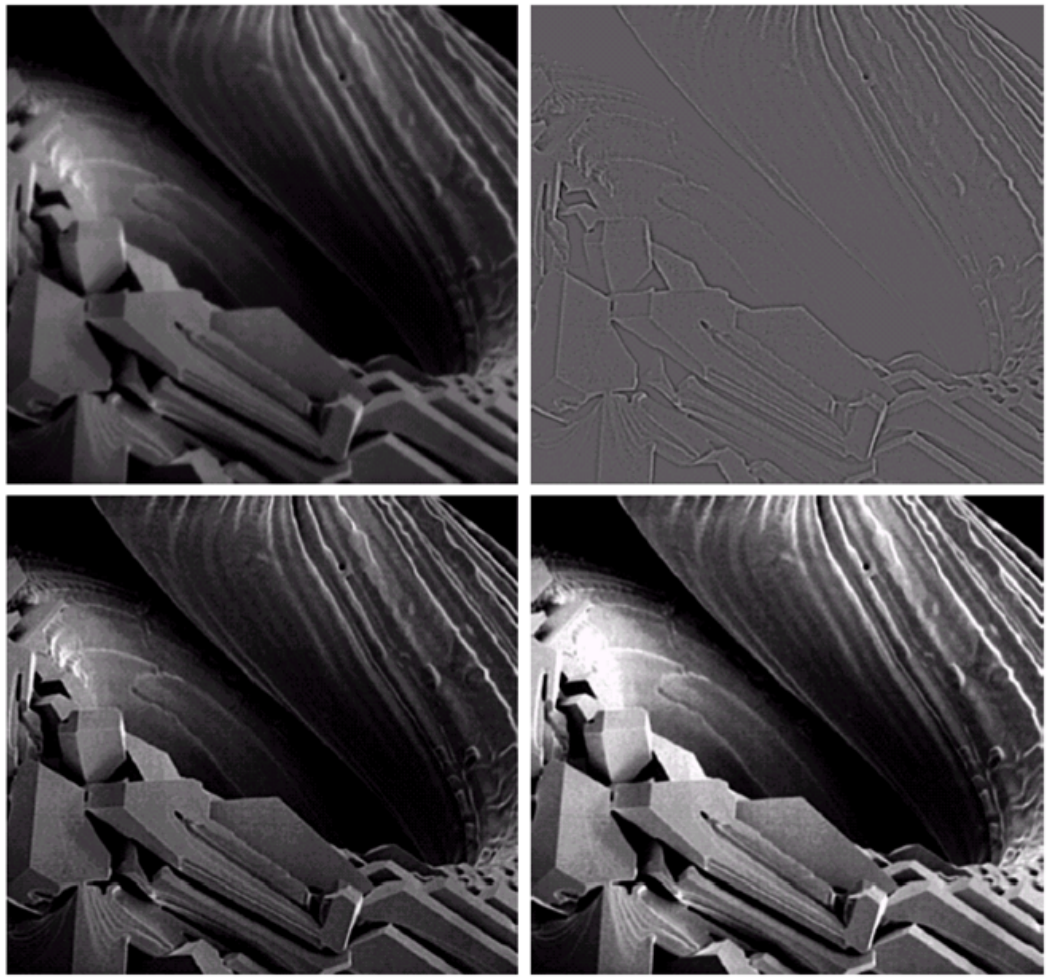
a b c
d e

FIGURE 3.41 (a) Composite Laplacian mask. (b) A second composite mask. (c) Scanning electron microscope image. (d) and (e) Results of filtering with the masks in (a) and (b), respectively. Note how much sharper (e) is than (d). (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene.)

Unsharp masking and high-boost filtering

- Unsharp masking is implemented by subtract a blurred version of an image from the image itself, i.e.,
$$f_s(x,y) = f(x,y) - \bar{f}(x,y)$$
where $f_s(x,y)$ denotes the sharpened image and $\bar{f}(x,y)$ is a blurred version of $f(x,y)$
- A slight further generalizaiton of unsharp masking is called high-boost filtering, and a high-boost filtered image, $f_{hb}(x,y)$, is defined as
$$f_{hb}(x,y) = Af(x,y) - \bar{f}(x,y)$$
where $A \geq 1$, and $\bar{f}(x,y)$ is a blurred version of $f(x,y)$
- Further, we can obtain
$$f_{hb}(x,y) = (A-1)f(x,y) + f(x,y) - \bar{f}(x,y)$$
$$f_{hb}(x,y) = (A-1)f(x,y) + f_s(x,y)$$
- If we use the Laplacian to compute $f_s(x,y)$, we have
$$f_{hb}(x,y) = (A-1)f(x,y) - \nabla^2 f(x,y)$$
or $$f_{hb}(x,y) = (A-1)f(x,y) + \nabla^2 f(x,y)$$

High-Boost Filtering with Laplacian Filters(cont.)



a b
c d

FIGURE 3.43
(a) Same as Fig. 3.41(c), but darker.
(a) Laplacian of (a) computed with the mask in Fig. 3.42(b) using $A = 0$.
(c) Laplacian enhanced image using the mask in Fig. 3.42(b) with $A = 1$. (d) Same as (c), but using $A = 1.7$.

Image Enhancement with the Gradient

a
b c
d e

FIGURE 3.44
A 3×3 region of an image (the z's are gray-level values) and masks used to compute the gradient at point labeled z_5. All masks coefficients sum to zero, as expected of a derivative operator.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

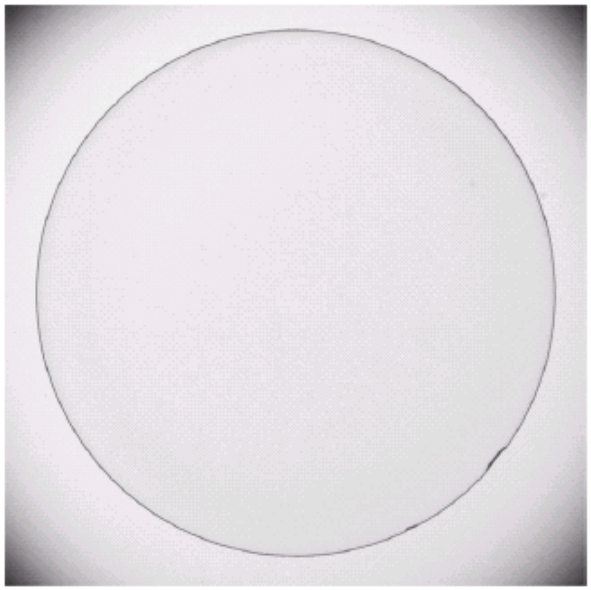
−1	0
0	1

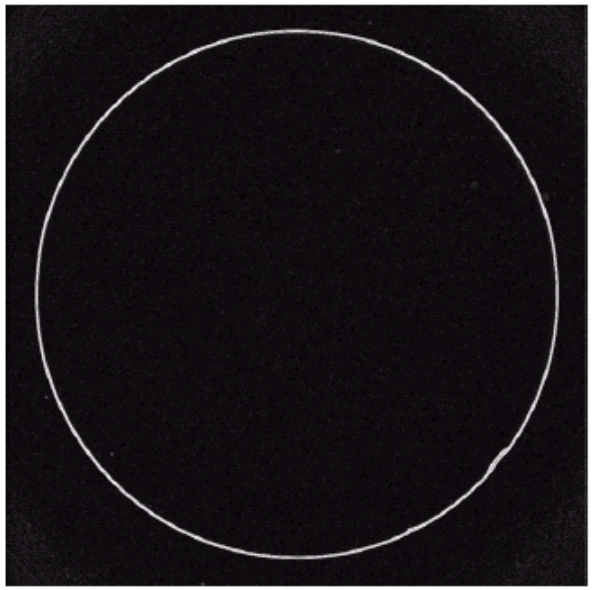
0	−1
1	0

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Image Enhancement with the Gradient(cont.)





a b

FIGURE 3.45 Optical image of contact lens (note defects on the boundary at 4 and 5 o'clock). (b) Sobel gradient. (Original image courtesy of Mr. Pete Sites, Perceptics Corporation.)

Nonlinear Spatial Filters

- A commonly-used tool for generating nonlinear spatial filters in IPT is function ordfilt2, which generates order-statistic filters (also called rank filters).
- The syntax of function ordfilt2 is

  g = ordfilt2(f, order, domain)

  This function creates the output image g by replacing each element of f by the order-th element in the sorted set of neighbors specified by the nonzero elements in domain.
- For example, to implement *a min filter* (order 1) of size $m \times n$ we use the syntax

  g = ordfilt2(f, 1, ones(m, n))

  g = ordfilt2(f, m*n, ones(m, n))

  g = ordfilt2(f, median(1:m*n), ones(m, n))

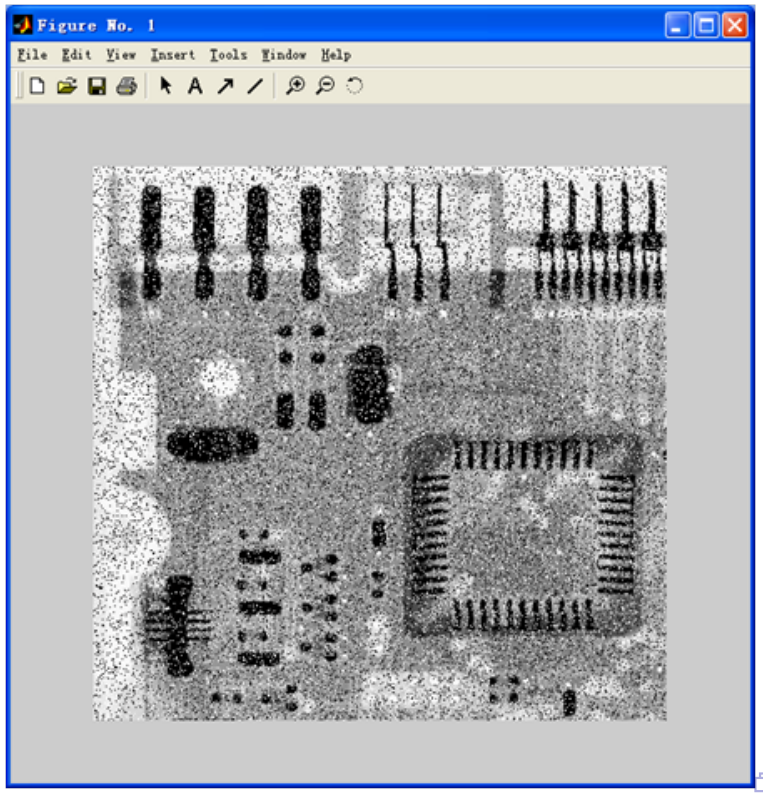
Nonlinear Spatial Filters-Medium Filter

- The best-known order-statistic filter in digital image processing is the median filter
- Because of its practical importance, the toolbox provides a specialized implementation of the 2-D median filter:
  - g = medfilt2(f, [m n], padopt)

    where the tuple [m n] defines a neighborhood of size m × n over which the median is computed, and padopt specifies one of three possible border padding options: 'zeros' (the default), 'symmetric' in which f is extended symmetrically by mirror-reflecting it across its border, and 'indexed', in which f is padded with 1s if it is of class double and with 0s otherwise.

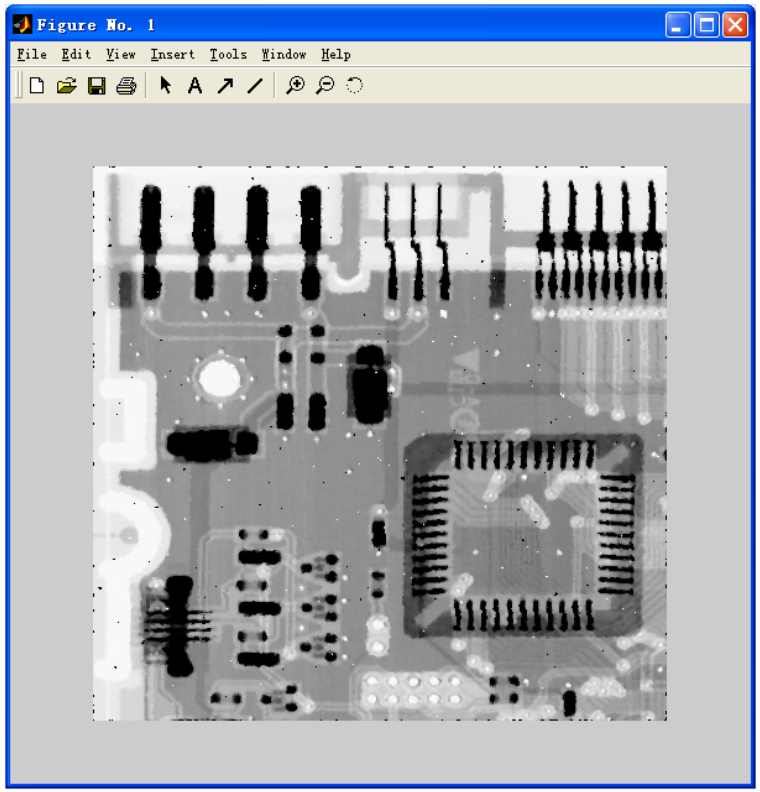
Nonlinear Spatial Filters(cont.)

- Median filtering is a useful tool for reducing salt-and-pepper noise in an image.
  - f=imread('Fig0318(a)(ckt-board-orig).tif');
  - fn=imnoise(f, 'salt & pepper', 0.2);
  - imshow(fn)



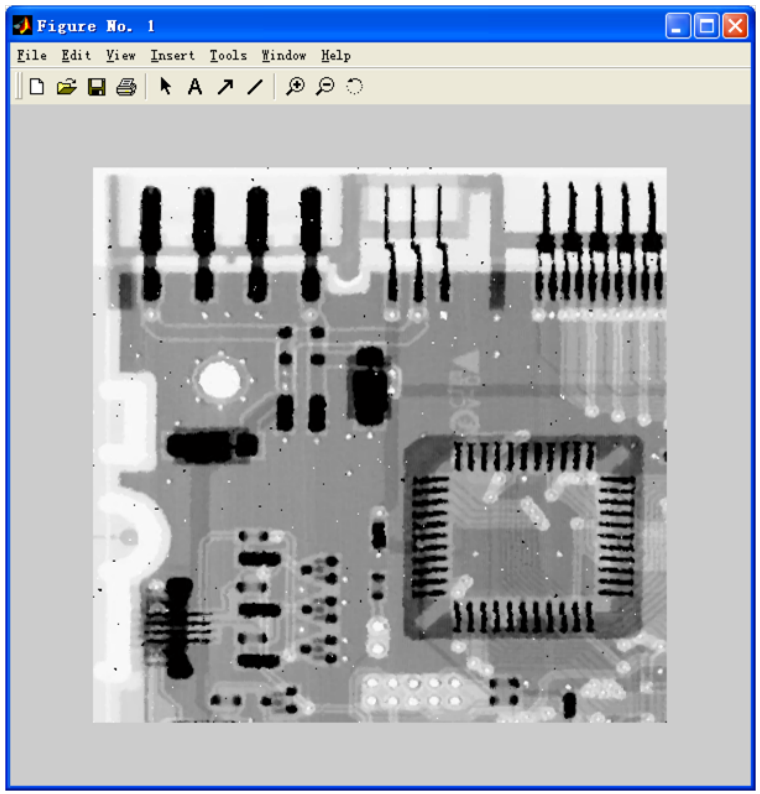
Nonlinear Spatial Filters(cont.)

- gm = medfilt2(fn);
- imshow(gm)

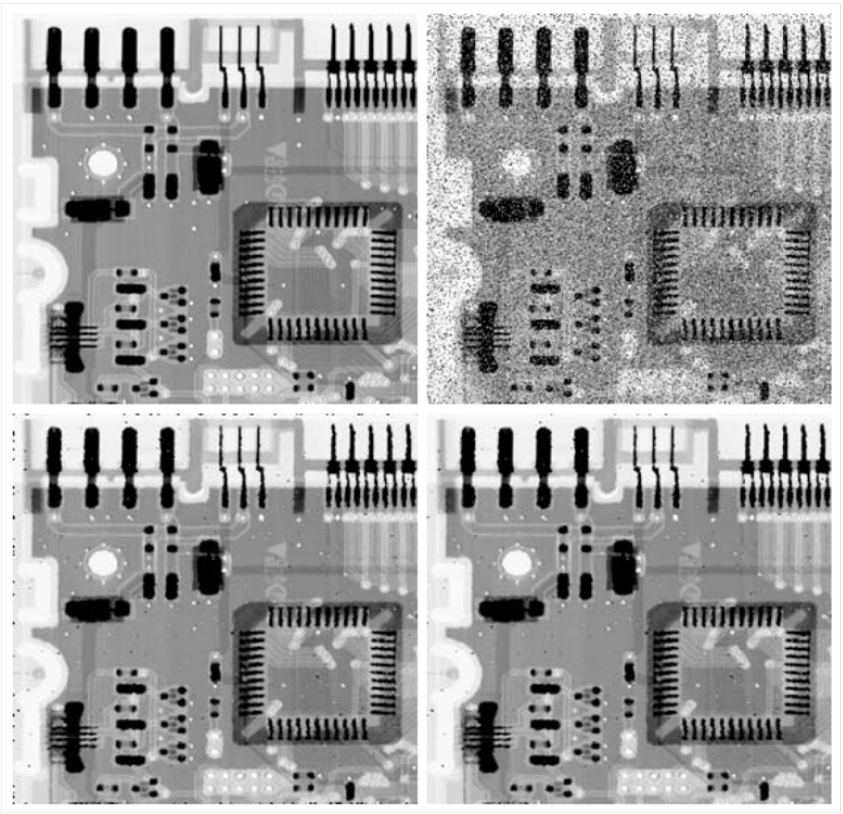


Nonlinear Spatial Filters(cont.)

- gms = medfilt2(fn, 'symmetric');
- imshow(gms)



Nonlinear Spatial Filters(cont.)



a b
c d

FIGURE 3.18 Median filtering, (a) X-ray image. (b) Image corrupted by salt-and-pepper noise. (c) Result of median filtering with `medfilt2` using the default settings. (d) Result of median filtering using the `'symmetric'` image extension option. Note the improvement in border behavior between (d) and (c). (Original image courtesy of Lixi, Inc.)

Image Processing and Analysis
Lecture 5、Image Enhencement in Frequency Domain(I)

Weiqiang Wang

School of Computer and Control Engineering, UCAS

November 10, 2015

Outline

2-D Fourier Transform

- Any function that periodically repeats itself can be expressed as the sum of sines and/or cosines of different frequencies, each multiplied by a different coefficient (Fourier series).
- Even functions that are not periodic (but whose area under the curve is finite) can be expressed as the integral of sines and/or cosines multiplied by a weighting function (Fourier transform).

- The frequency domain refers to the plane of the two dimensional discrete Fourier transform of an image.
- The purpose of the Fourier transform is to represent a signal as a linear combination of sinusoidal signals of various frequencies.

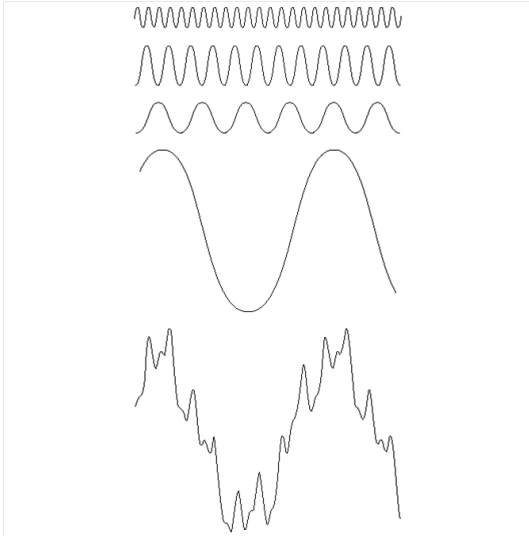


FIGURE 4.1 The function at the bottom is the sum of the four functions above it. Fourier's idea in 1807 that periodic functions could be represented as a weighted sum of sines and cosines was met with skepticism.

2-D Continuous Fourier Transform

- The one-dimensional Fourier transform and its inverse
  - Fourier transform
    $$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx, where\ j = \sqrt{-1}$$
  - Inverse Fourier transform:
    $$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du \qquad e^{j\theta} = \cos\theta + j\sin\theta$$
- The two-dimensional Fourier transform and its inverse
  - Fourier transform
    $$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} dxdy$$
  - Inverse Fourier transform:
    $$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v) e^{j2\pi(ux+vy)} dudv$$

2-D Discrete Fourier Transform

- The one-dimensional Discrete Fourier transform (DFT) and its inverse
  - Fourier transform

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-\frac{j2\pi ux}{M}} \quad for\ u = 0, 1, 2, ..., M-1$$

  - Inverse Fourier transform:

$$f(x) = \sum_{u=0}^{M-1} F(u)e^{\frac{j2\pi ux}{M}} \quad for\ x = 0, 1, 2, ..., M-1$$

- Since $e^{j\theta} = \cos\theta + j\sin\theta$, then DFT can be redefined as

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)[\cos\frac{2\pi ux}{M} - j\sin\frac{2\pi ux}{M}]$$
$$for\ u = 0, 1, 2, ..., M-1$$

  - Frequency (time) domain: the domain (values of u) over which the values of $F(u)$ range; because u determines the frequency of the components of the transform.
  - Frequency (time) component: each of the M terms of $F(u)$.

2-D Discrete Fourier Transform

- $F(u)$ can be expressed in polar coordinates:

$$F(u) = |F(u)|e^{j\phi(u)}$$

$$where\ |F(u)| = [R(u) + I(u)]^{\frac{1}{2}}\ (magnitude\ or\ spectrum)$$

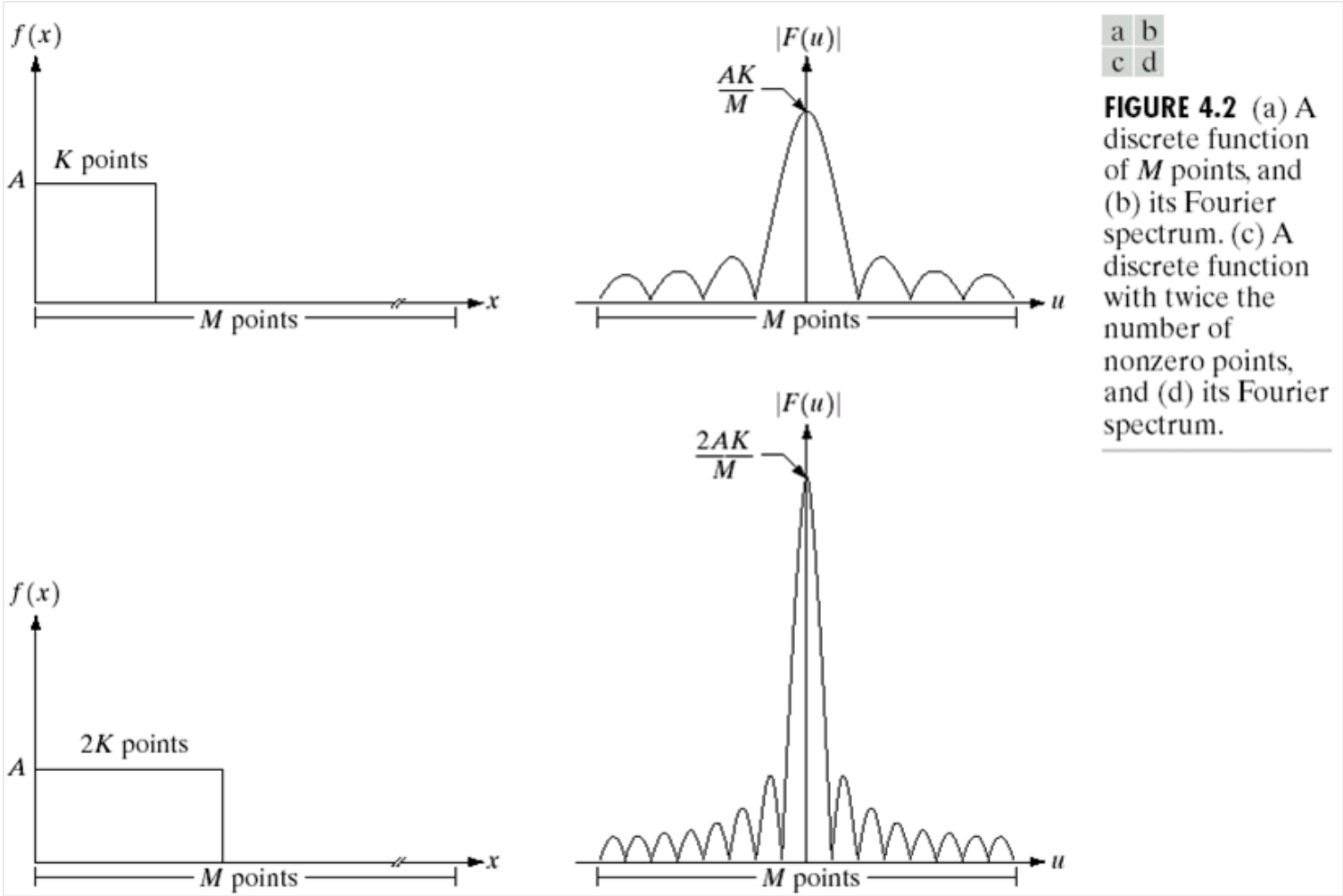
$$\phi(u) = \tan^{-1}[\frac{I(u)}{R(u)}]\ (phase\ angle\ or\ phase\ spectrum)$$

  - $I(u)$: the imaginary part of $F(u)$.
  - $R(u)$: the real part of $F(u)$.

- Power spectrum:

$$P(u) = |F(u)|^2 = R^2(u) + I^2(u)$$

2-D Discrete Fourier Transform



a b
c d

FIGURE 4.2 (a) A discrete function of M points, and (b) its Fourier spectrum. (c) A discrete function with twice the number of nonzero points, and (d) its Fourier spectrum.

2-D Discrete Fourier Transform

- The two-dimensional Fourier transform and its inverse
  - Fourier transform (discrete case)DTC

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$
$$for\ u = 0,1,2,...,M-1,\ v = 0,1,2,...,N-1$$

  - Inverse Fourier transform:

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$
$$for\ x = 0,1,2,...,M-1,\ y = 0,1,2,...,N-1$$

    - u,v: the transform or frequency variables
    - x,y: the spatial or image variables

2-D Discrete Fourier Transform

- We define the Fourier spectrum, phase angle, and power spectrum as follows:

$$|F(u,v)|^2 = [R^2(u,v) + I^2(u,v)]^{\frac{1}{2}} \quad (spectrum)$$

$$\phi(u,v) = \tan^{-1}[\frac{I(u,v)}{R(u,v)}] \quad (phase\ angle)$$

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v) \quad (powerspectrum)$$

  - $I(u,v)$: the imaginary part of $F(u,v)$.
  - $R(u,v)$: the real part of $F(u,v)$.

Properties of 2-D DFT

- Time-shifting

$$\Im[f(x-x_0, y-y_0)] = F(u,v)e^{-j2\pi(\frac{ux_0}{M}+\frac{vy_0}{N})}$$

- Frequency shifting

$$\Im[f(x,y)e^{-j2\pi(\frac{u_0x}{M}+\frac{v_0y}{N})}] = F(u-u_0, v-v_0)$$

$$\Im[f(x,y)(-1)^{x+y}] = F(u-\frac{M}{2}, v-\frac{N}{2})$$

- Average and Symmetry

$$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) \quad (average)$$

$$F(u,v) = F^*(-u,-v) \quad (conjugate\ symmetric)$$

$$|F(u,v)| = |F(-u,-v)| \quad (symmetric)$$

Properties of 2-D DFT (cont.)

- Seperability

$$
\begin{aligned}
F(u,v) &= \Im[f(x,y)] \\
&= \Sigma_y[\Sigma_x f(x,y) exp(-j2\pi \tfrac{xu}{M})] exp(-j2\pi \tfrac{yv}{N}) \\
&= \Sigma_y F(u,y) exp(-j2\pi \tfrac{yv}{N})
\end{aligned}
$$

The 2D DFT $F(u,v)$ can be obtained by

1. Taking the 1D DFT of every row of image $f(x,y)$, $F(u,y)$.
2. The 1D DFT of every column of $F(u,y)$.

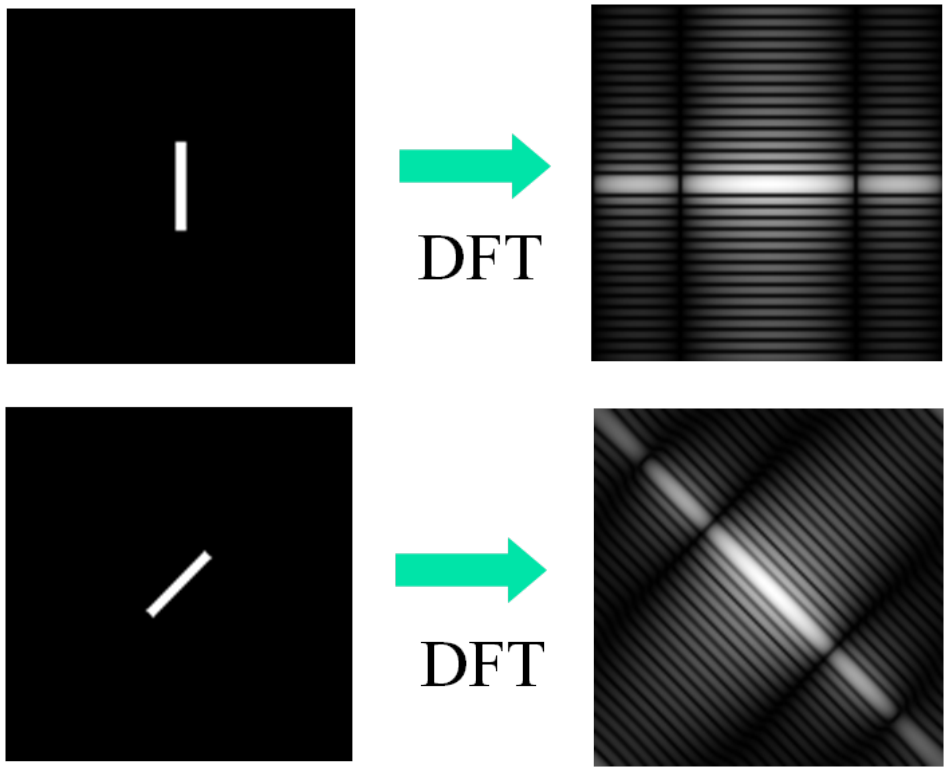
(a)f(x,y) (b) F(u,y) (c)F(u,v)

Properties of 2-D DFT (cont.)

- Rotation

let $x = r\cos\theta, y = r\sin\theta, u = \omega\cos\varphi, v = \omega\sin\varphi$

$$f(r, \theta + \theta_0) \Leftrightarrow F(\omega, \varphi + \theta_0)$$



Properties of 2-D DFT (cont.)

- Periodicity

$$f(x,y) = f(x+M,y) = f(x,y+N) = f(x+M,y+N)$$

$$F(u,v) = F(u+M,v) = F(u,v+N) = F(u+M,v+N)$$

- Linearity

$$\Im(af(x,y) + bg(x,y)) = a\Im(f(x,y)) + b\Im(g(x,y))$$

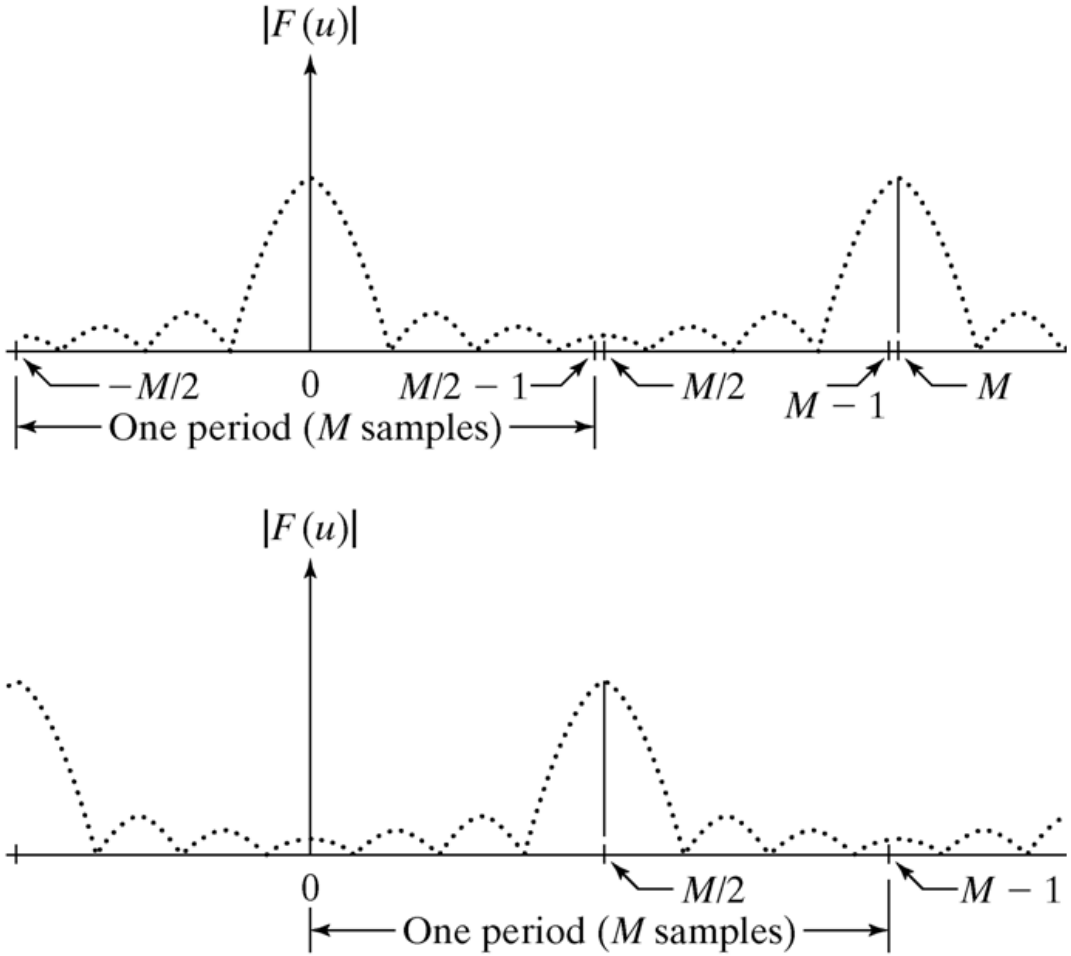
- Differentiation

$$\Im(\frac{\partial^n f(x,y)}{\partial x^n}) = (j2\pi u)^n \Im(f(x,y)) = (j2\pi u)^n F(u,v)$$

$$\Im((-j2\pi u)^n f(x,y)) = \frac{\partial^n F(u,v)}{\partial u^n}$$

$$\Im(\nabla^2 f(x,y)) = -4\pi^2(u^2+v^2)F(u,v)$$

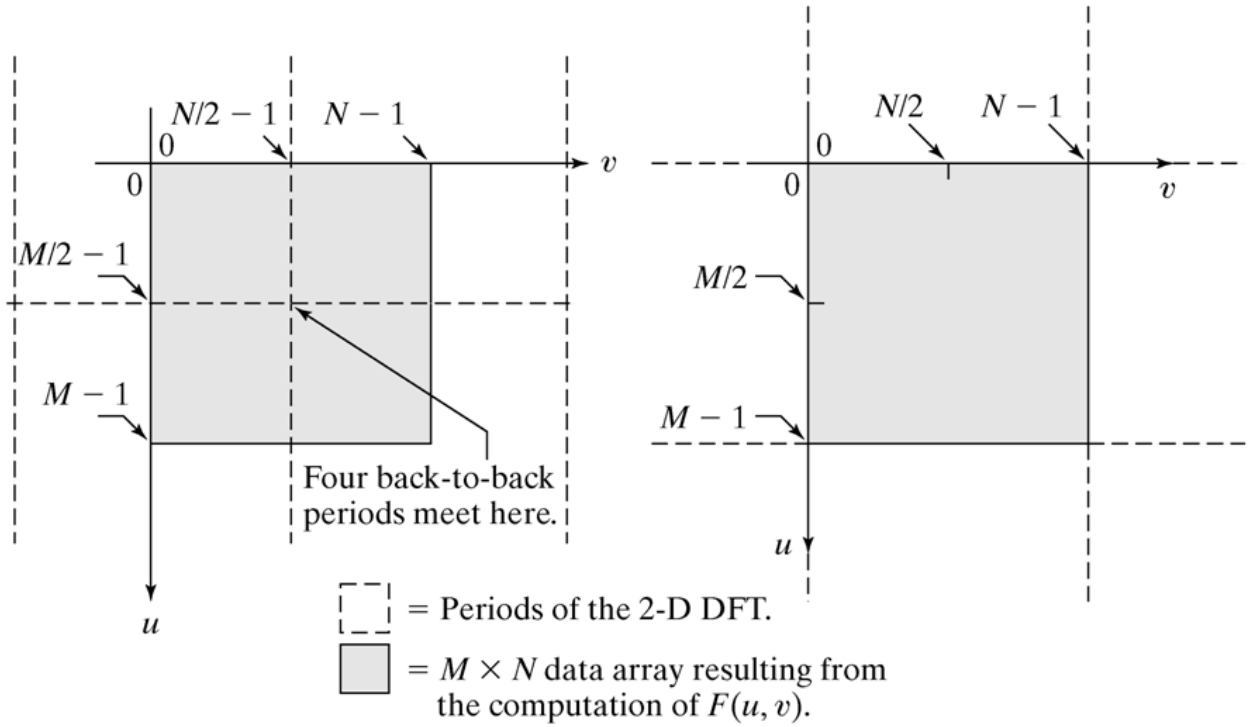
2-D Discrete Fourier Transform



a
b

FIGURE 4.1
(a) Fourier spectrum showing back-to-back half periods in the interval $[0, M-1]$. (b) Centered spectrum in the same interval, obtained by multiplying $f(x)$ by $(-1)^x$ prior to computing the Fourier transform.

2-D Discrete Fourier Transform (cont.)



a b

FIGURE 4.2 (a) $M \times N$ Fourier spectrum (shaded), showing four back-to-back quarter periods contained in the spectrum data. (b) Spectrum obtained by multiplying $f(x, y)$ by $(-1)^{x+y}$ prior to computing the Fourier transform. Only one period is shown shaded because this is the data that would be obtained by an implementation of the equation for $F(u, v)$.

Properties of 2-D DFT (cont.)

- Convolution

$$\Im(f(x,y) * g(x,y)) = F(u,v)G(u,v)$$

$$\Im(f(x,y)g(x,y)) = F(u,v) * G(u,v)$$

- Correlation

$$\Im(f(x,y) \circ g(x,y)) = F^*(u,v)G(u,v)$$

$$\Im(f(x,y) \circ f(x,y)) = |F(u,v)|^2$$

$$\Im(f^*(x,y)g(x,y)) = F(u,v) \circ G(u,v)$$

$$\Im(|f(x,y)|^2) = F(u,v) \circ F(u,v)$$

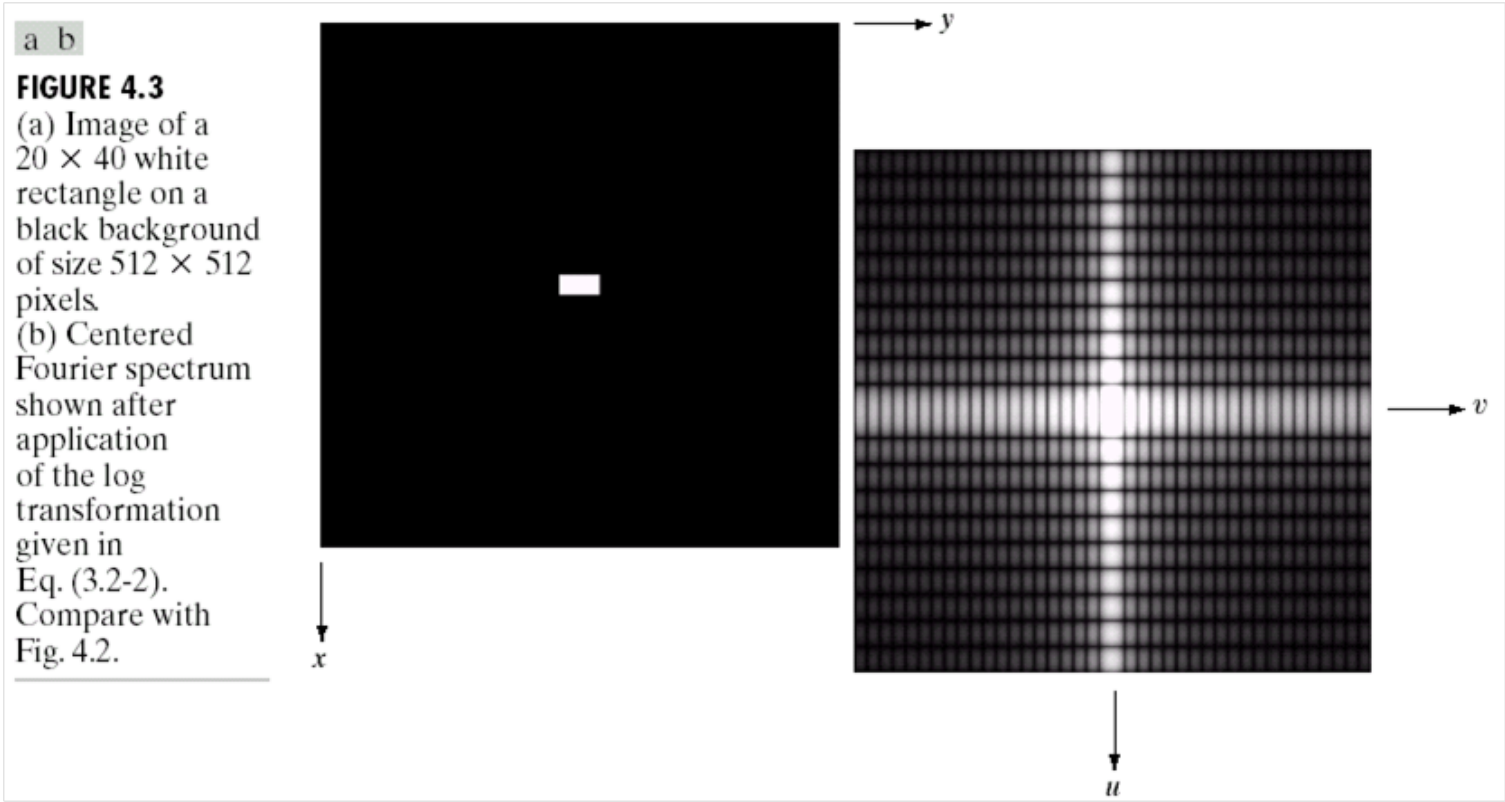
- Similarity

$$\Im(f(ax,by)) = \frac{1}{|ab|}F(\frac{u}{a},\frac{v}{b})$$

Some useful FT pairs

- $\delta(x,y) \Leftrightarrow 1$
- $A2\pi\sigma^2 exp(-2\pi^2\sigma^2(x^2+y^2)) \Leftrightarrow Aexp(-\frac{(u^2+v^2)}{2\sigma^2})$
- $\cos(2\pi u_0 x + 2\pi v_0 y) \Leftrightarrow \frac{1}{2}[\delta(u+u_0, v+v_0) + \delta(u-u_0, v-v_0)]$
- $\sin(2\pi u_0 x + 2\pi v_0 y) \Leftrightarrow \frac{1}{2}j[\delta(u+u_0, v+v_0) - \delta(u-u_0, v-v_0)]$

2-D Discrete Fourier Transform



2-D DFT in Matlab

- The DFT and its inverse are obtained in practice using a Fast Fourier Transform(FFT) algorithm. The FFT of an $M \times N$ image array f is obtained in the toolbox with function fft2,which has the simple syntax:

  F=fft2(f)

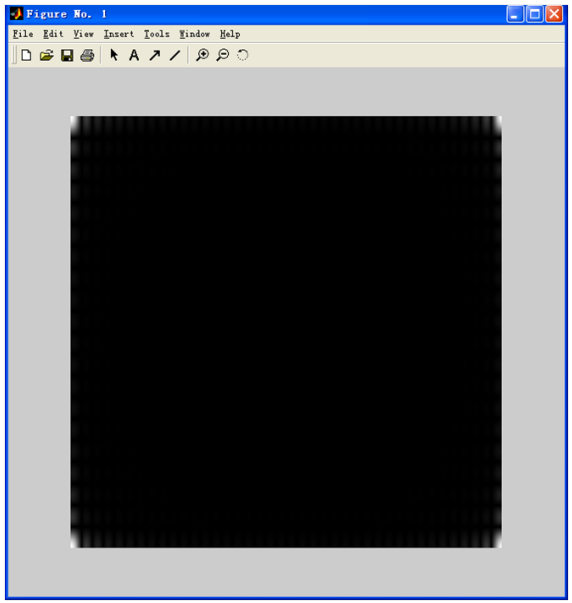
  This function returns a Fourier transform that is also of size $M \times N$, with the origin of the data at the top left, and with four quarter periods meeting at the center of the frequency rectangle.

- The Fourier spectrum is obtained by using function abs:

  S=abs(F)

2-D DFT in Matlab(cont.)

- f=imread('Fig0403(a)(image).tif');
- F=fft2(f);
- S=abs(F);
- imshow(S,[])

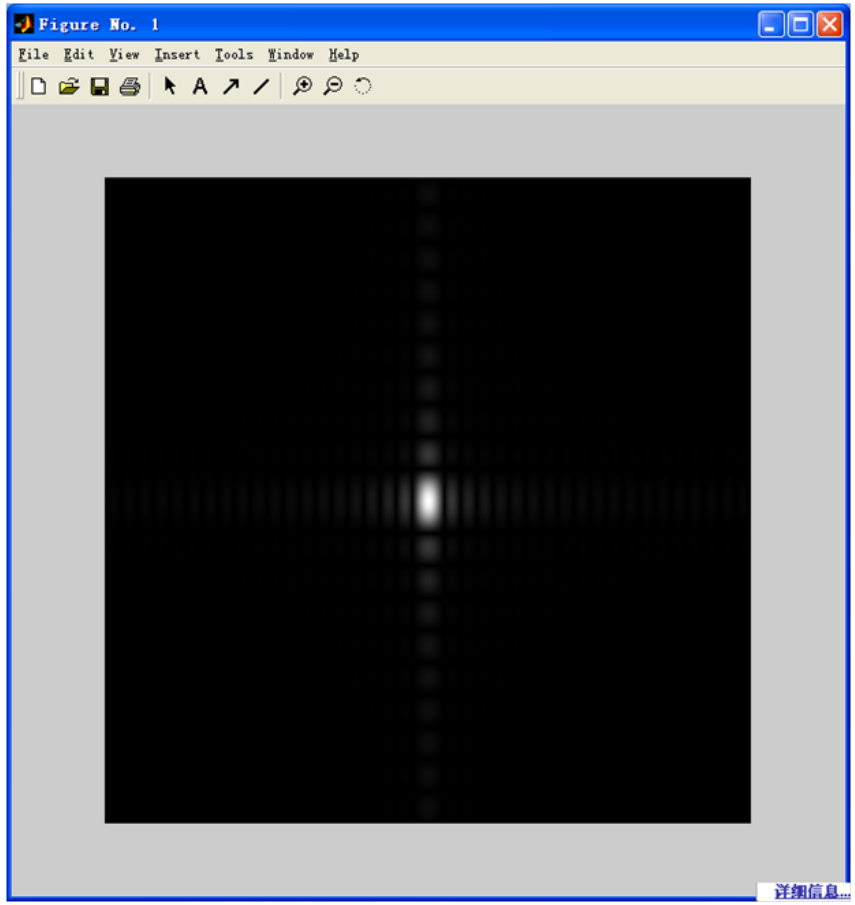


2-D DFT in Matlab(cont.)

- Fc=fftshift(F);
- imshow(abs(Fc),[])

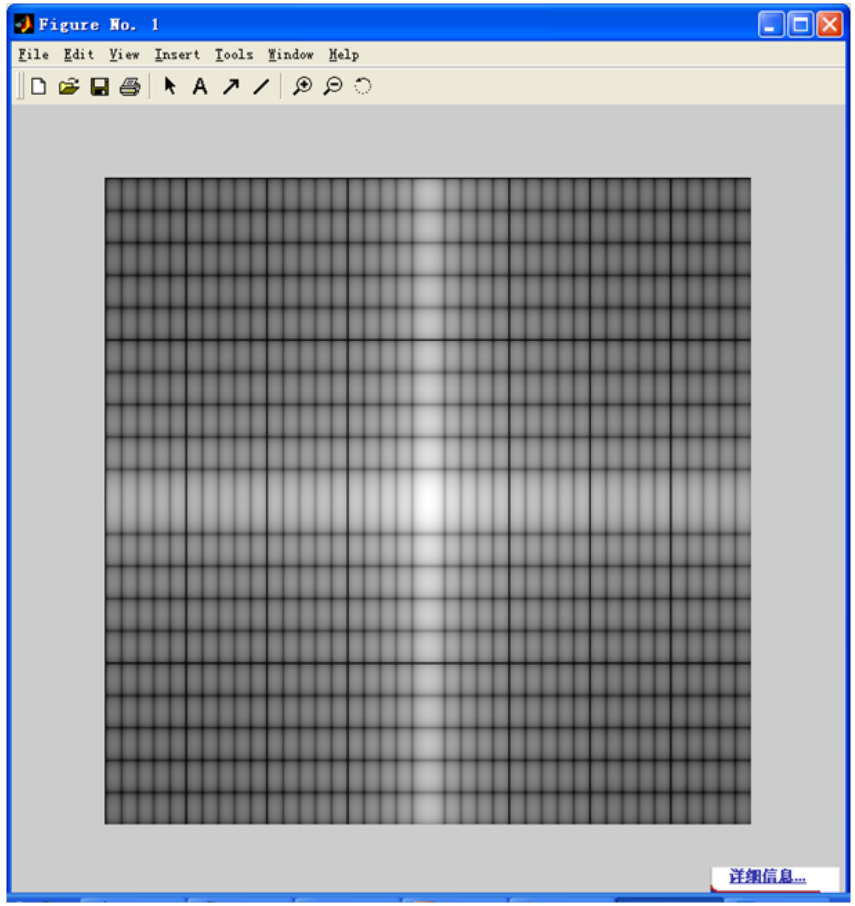
  The net result of using fftshift is the same as if the input image had been multiplied by $(-1)^{x+y}$ *prior to computing the transform.*

  Note,however,that the two processes are not interchangeable.

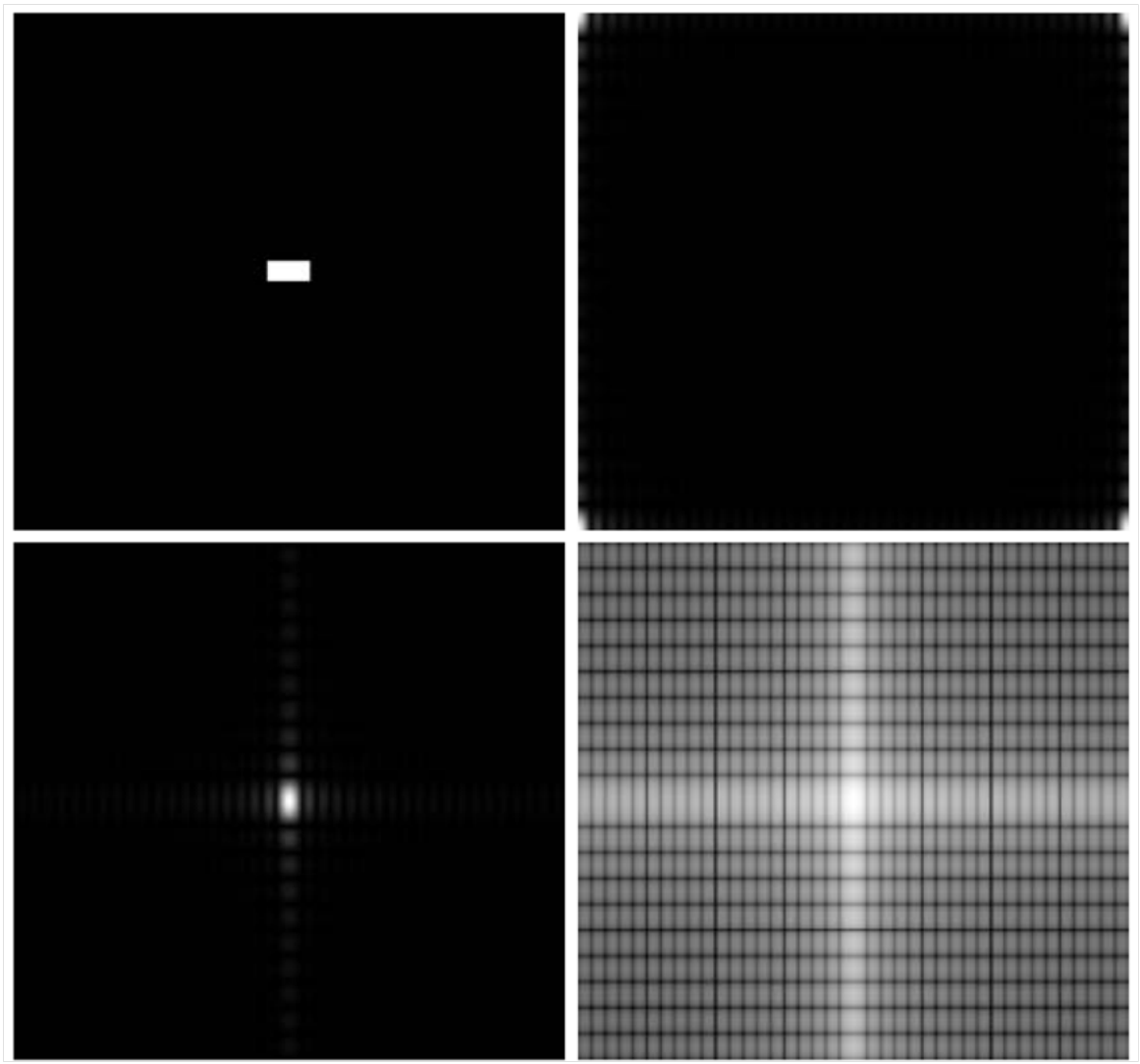


2-D DFT in Matlab(cont.)

- S2=log(1+abs(Fc));
- imshow(S2,[])



2-D DFT in Matlab(cont.)



a b
c d

FIGURE 4.3
(a) A simple image.
(b) Fourier spectrum.
(c) Centered spectrum.
(d) Spectrum visually enhanced by a log transformation.

2-D DFT in Matlab(cont.)

- We point out that the inverse Fourier transform is computed using function if ifft2. which has the basic syntax

  f=ifft2(F)

- In practice, the output of ifft2 often has very small imaginary components resulting from round-off errors. Thus, it is good practice to extract the real part of the result.

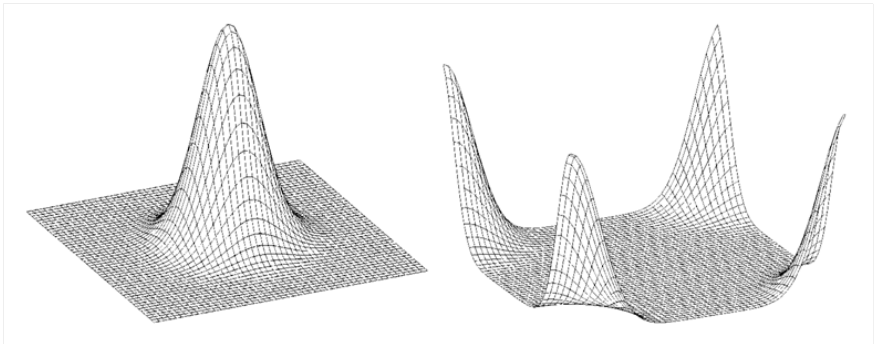
  f=real(ifft2(F))

Fundamental Concepts

- The foundation for linear filtering in both the spatial and frequency domains is the convolution theorem, which may be written as.

$$f(x,y) * h(x,y) \Longleftrightarrow F(u,v)H(u,v)$$

- The idea in frequency domain filtering is to select a filter transfer function that modifies $F(u,v)$ in a specified manner.
- For example, the lowpass filter in figure 4.4



a b

FIGURE 4.4 Transfer functions of (a) a centered lowpass filter, and (b) the format used for DFT filtering. Note that these are frequency domain filters.

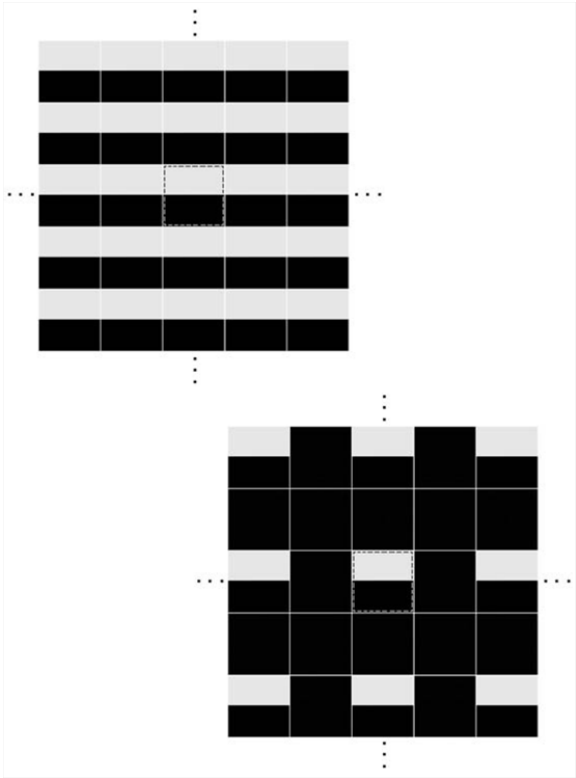
Fundamental Concepts(cont.)

- Based on the convolution theorem, we know that to obtain the corresponding filtered image in the spatial domain we simply compute the inverse Fourier transform of the product $H(u,v)F(u,v)$.
- Convolving periodic functions can cause interference of the nonzero periods if the periods are close with respect to the duration of the nonzero parts of the functions. This interference, called *wraparound error*, can be avoided by padding the functions with zeros.
- For example, the lowpass filter in figure 4.4

a b c

FIGURE 4.5 (a) A simple image of size 256 × 256. (b) Image lowpass-filtered in the frequency domain without padding. (c) Image lowpass-filtered in the frequency domain with padding. Compare the light portion of the vertical edges in (b) and (c).

Fundamental Concepts(cont.)



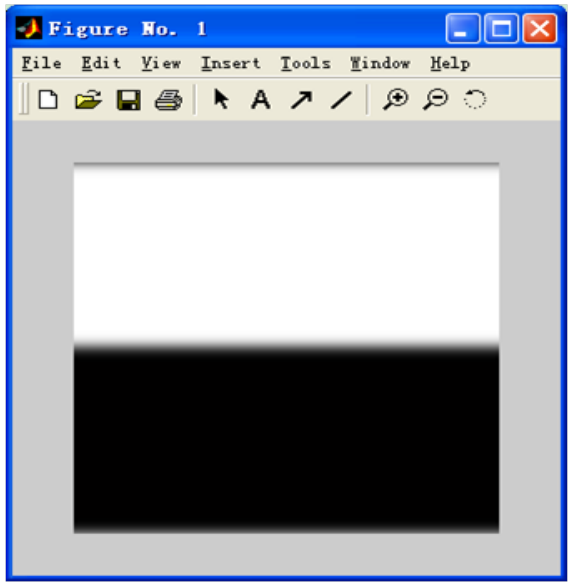
a
b

FIGURE 4.6
(a) Implied, infinite periodic sequence of the image in Fig. 4.5(a). The dashed region represents the data processed by `fft2`. (b) The same periodic sequence after padding with 0s. The thin white lines in both images are shown for convenience in viewing; they are not part of the data.

FIGURE 4.7 Full padded image resulting from `ifft2` after filtering. This image is of size 512×512 pixels.

Fundamental Concepts(cont.)

- Image lowpass-filtered in the frequency domain without padding
  - f=imread('Fig0405(a)(square_original).tif');
  - [m n]=size(f)
  - F=fft2(f);
  - H=lpfilter('gaussian',m,n,10);
  - G=H.*F;
  - g=real(ifft2(G))
  - imshow(g,[])



Basic Steps in DFT Filtering

- 1. Obtain the padding parameters using function paddedsize:

  PQ=paddedsize(size(f));
- 2. Obtain the Fourier transform with padding:

  F=fft2(f,PQ(1),PQ(2));
- 3. Generate a filter function, H, of size PQ(1)×PQ(2) using any of the methods discussed in the remainder of this chapter.

  The filter must be in the format shown in Fig. 4.4(b). If it is centered instead, as in Fig. 4.4(a),let H = fftshift (H) before using the filter.
- 4. Multiply the transform by the filter:G = H.*F;
- 5. Obtain the real part of the inverse FFT of G g=real(ifft2(G));
- 6. Crop the top, left rectangle to the original size:

  g=g(1:size(f,1),1:size(f,2));

Basic Steps in DFT Filtering(cont.)

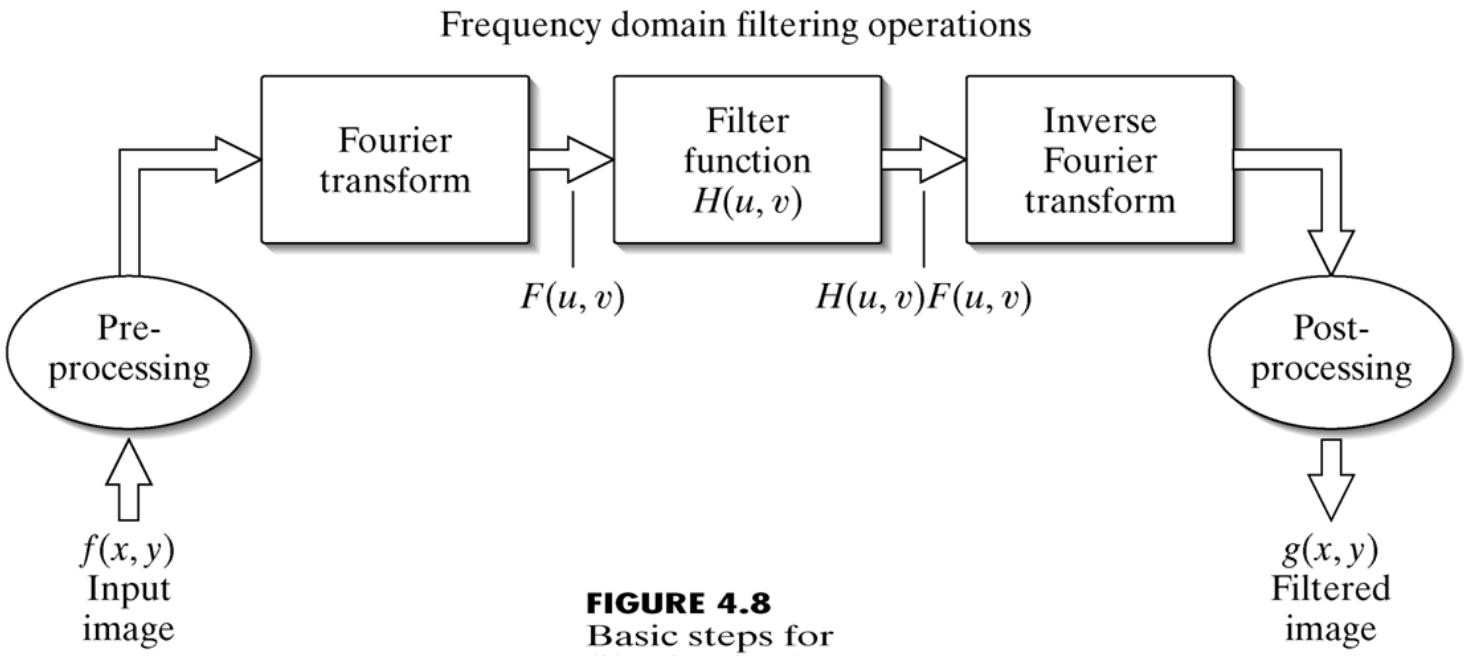


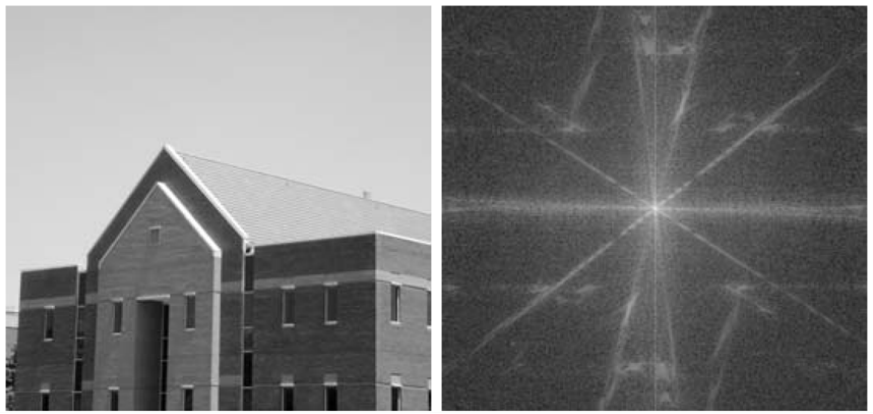
FIGURE 4.8 Basic steps for filtering in the frequency domain.

Obtaining Frequency Domain Filters from Spatial Filters

- Why obtains Frequency Domain Filters from Spatial Filters?
  - Efficiency
  - Meaningful comparisons
- How?
  - How to convert spatial filters into equivalent frequency domain filters;
  - How to compare the results between spatial domain filtering using imfilter, and frequency domain filtering.freqz2

Obtaining Frequency Domain Filters from Spatial Filters(cont.)

- >>f=imread('Fig0409(a)(bld).tif');
- >>F=fft2(f);
- >>S=fftshift(log(1+abs(F)));
- >>S=gscale(S);
- >>imshow(S)

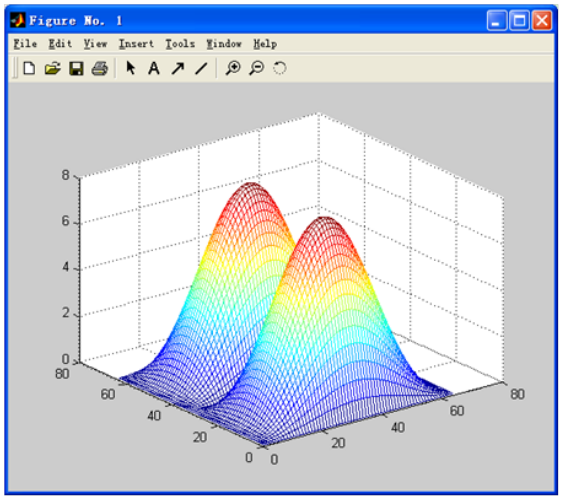


a b

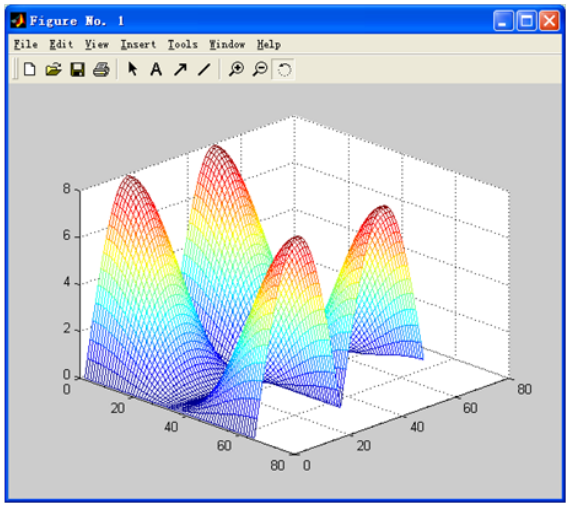
FIGURE 4.9
(a) A gray-scale image. (b) Its Fourier spectrum.

Obtaining Frequency Domain Filters from Spatial Filters(cont.)

- h=fspecial('sobel');
- H=freqz2(h);
- mesh(abs(H))



- H1=ifftshift(H);
- mesh(abs(H1))
- view(45,30)



Obtaining Frequency Domain Filters from Spatial Filters(cont.)

Spatial domain	Frequent domain
gs=imfilter(double(f),h);	PQ=paddedsize(size(f)); H=freqz2(h,PQ(1),PQ(2)); H1=ifftshift(H); gf=dftfilt(f,H1);

- d=abs(gs-gf);
- max(d(:))

  ans=

  5.4015e-012
- min(d(:))

  ans=

  0

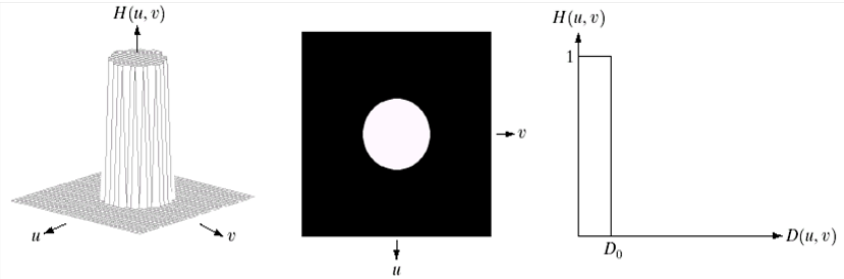
Generating Filters Directly in the Frequency Domain

- Ideal lowpass filter(ILPF)

$$H(u,v) = \begin{cases} 1 & if D(u,v) \le D_0 \\ 0 & if D(u,v) > D_0 \end{cases}$$

where $D(u,v)$ *: the distance from point* (u,v) *to the center of the frequency rectangle*

$$D(u,v) = [(u - M/2)^2 + (v - N/2)^2]^{\frac{1}{2}}$$



a b c

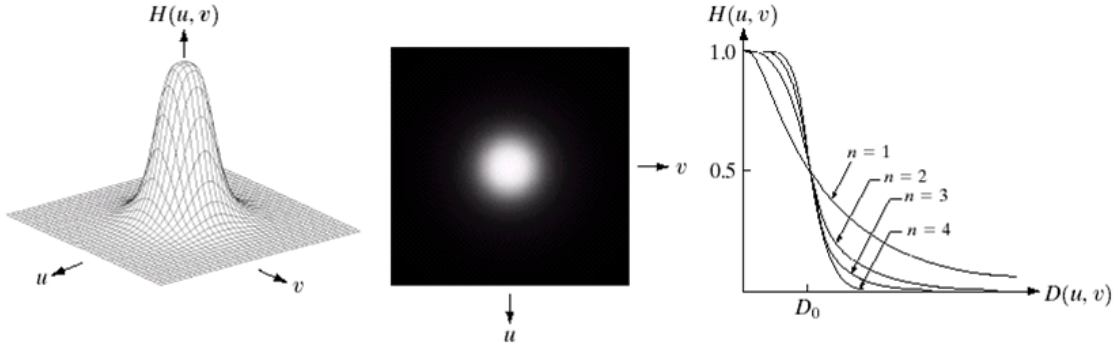
FIGURE 4.10 (a) Perspective plot of an ideal lowpass filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross section.

- Create meshgrid arrays for use in implementing Filters in the frequency domain dftuv

Generating Filters Directly in the Frequency Domain(cont.)

- Butterworth Lowpass Filters (BLPFs) With order n

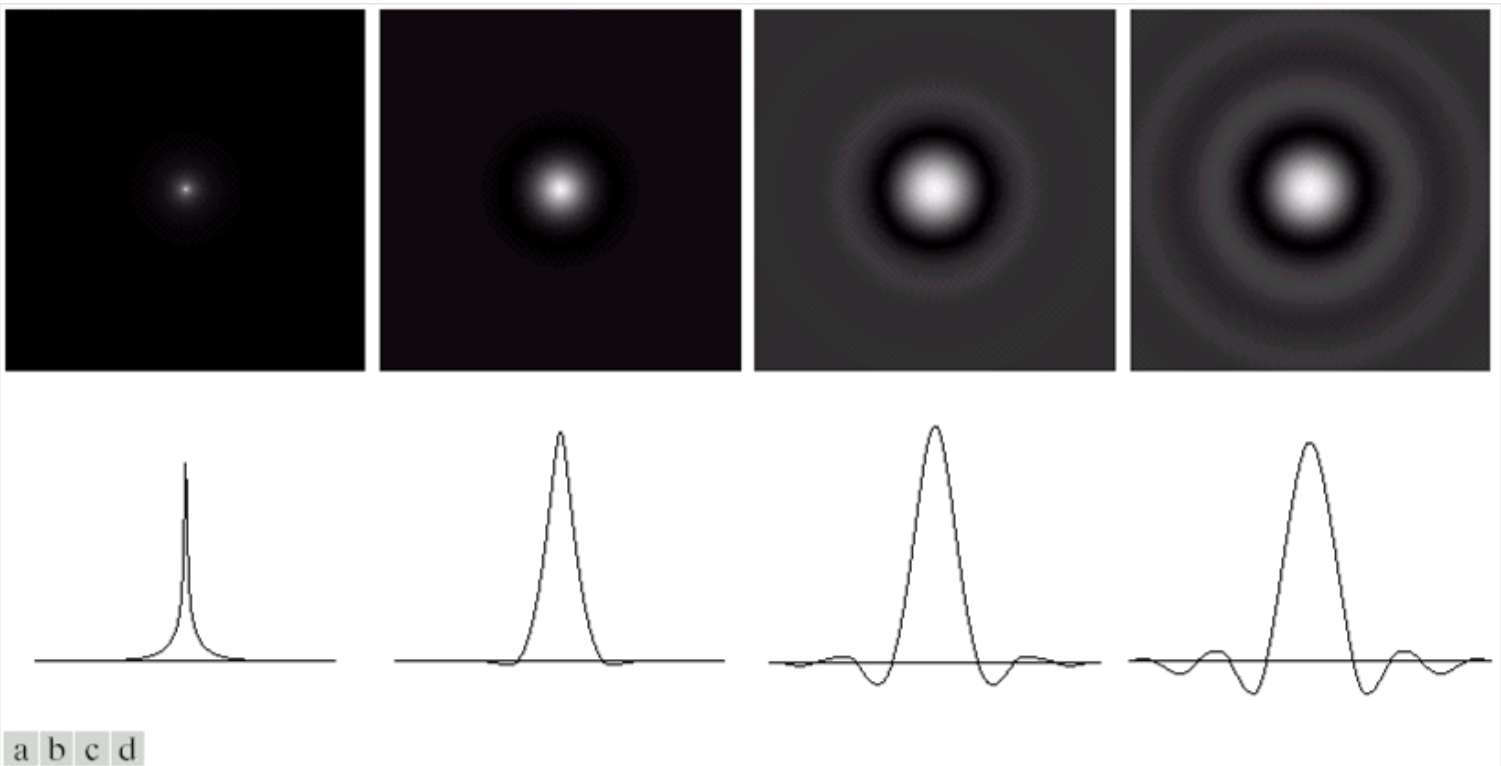
$$H(u,v) = \frac{1}{1+[D(u,v)/D_0]^{2n}}$$



a b c

FIGURE 4.14 (a) Perspective plot of a Butterworth lowpass filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

Generating Filters Directly in the Frequency Domain(cont.)

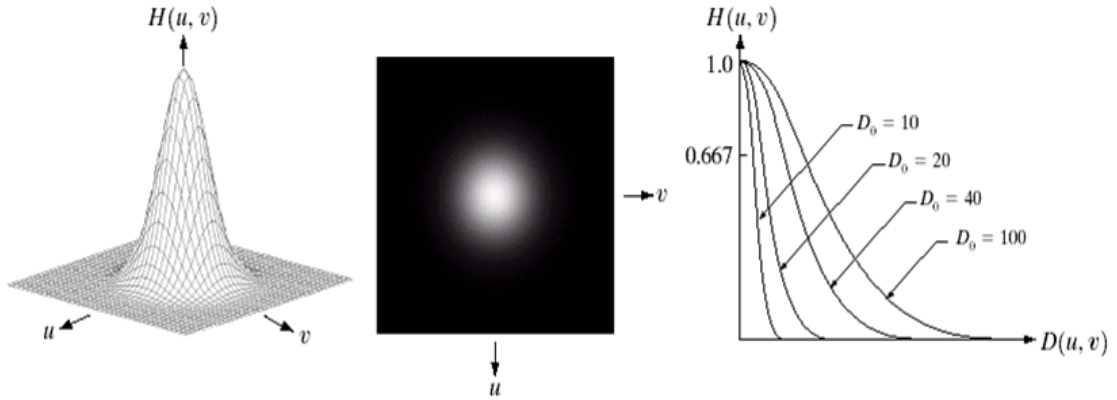


a b c d

FIGURE 4.16 (a)–(d) Spatial representation of BLPFs of order 1, 2, 5, and 20, and corresponding gray-level profiles through the center of the filters (all filters have a cutoff frequency of 5). Note that ringing increases as a function of filter order.

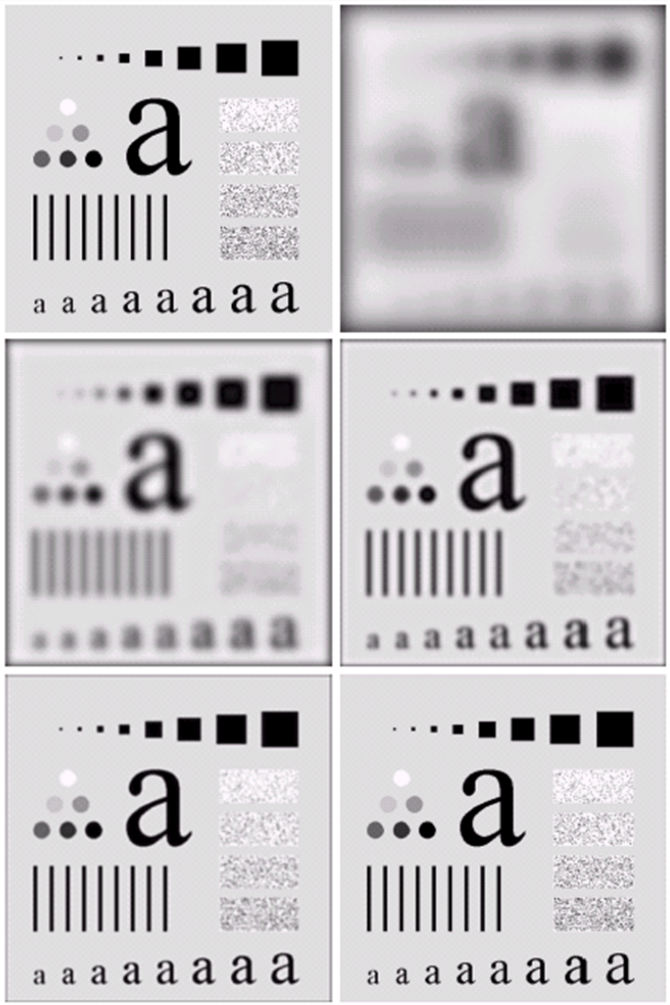
Gaussian Lowpass Filters (GLPFs)

- $H(u,v) = e^{-D^2(u,v)/2D_0^2}$



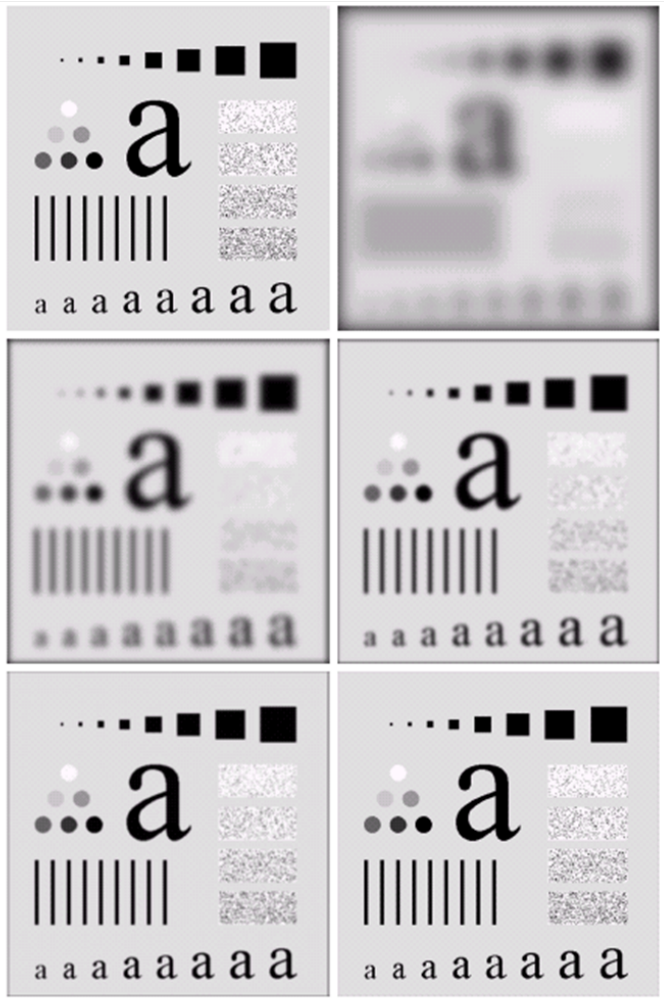
a b c

FIGURE 4.17 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.



a b c d e f

FIGURE 4.15 (a) Original image. (b)–(f) Results of filtering with BLPFs of order 2, with cutoff frequencies at radii of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Fig. 4.12.



a b c d e f

FIGURE 4.18 (a) Original image. (b)–(f) Results of filtering with Gaussian lowpass filters with cutoff frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Figs. 4.12 and 4.15.

Examples of Lowpass Filtering

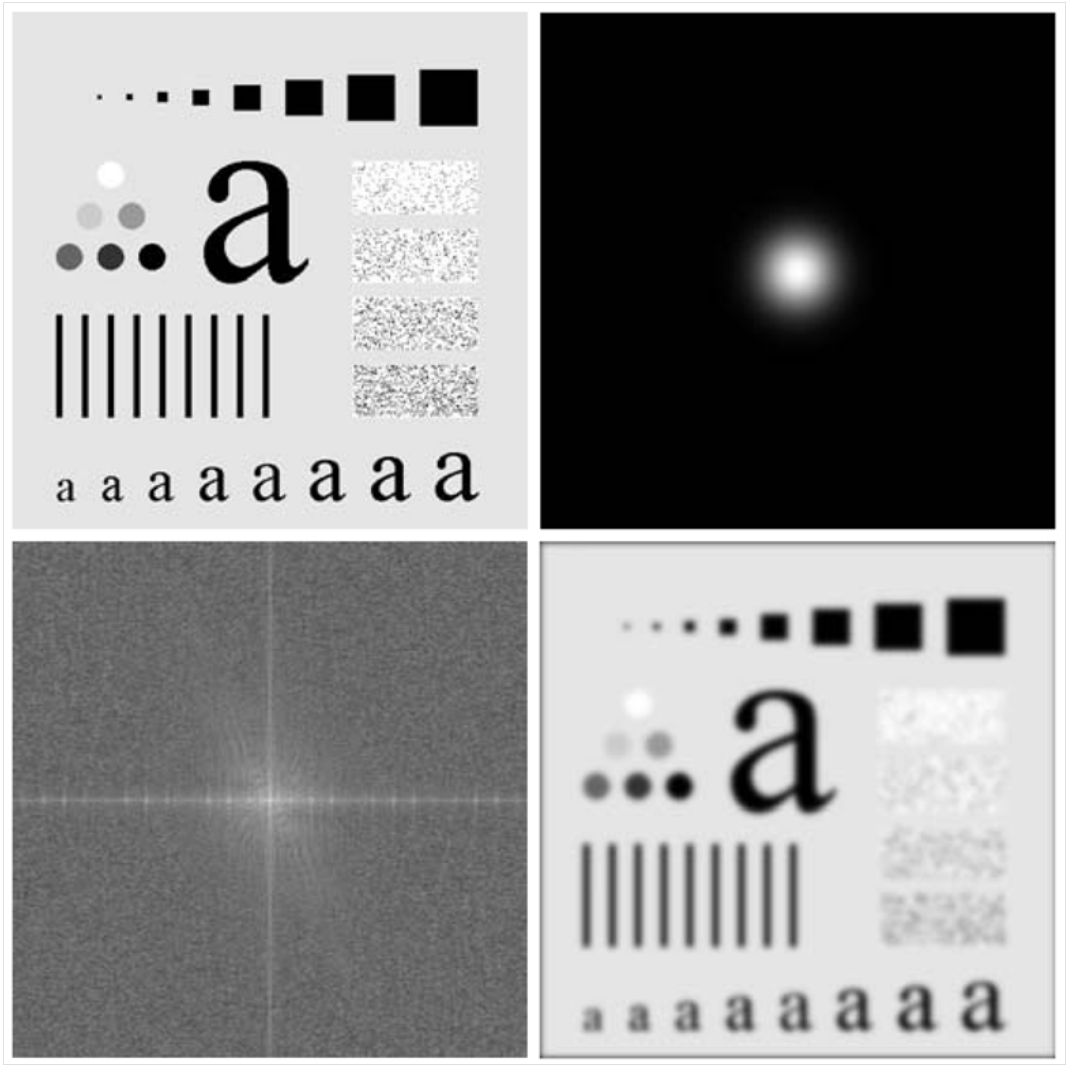
a b c

FIGURE 4.20 (a) Original image (1028 × 732 pixels). (b) Result of filtering with a GLPF with $D_0 = 100$. (c) Result of filtering with a GLPF with $D_0 = 80$. Note reduction in skin fine lines in the magnified sections of (b) and (c).

Generating Filters Directly in the Frequency Domain

- >>f=imread('Fig0413(a)(original_test_pattern).tif');
- >>PQ=paddedsize(size(f));
- >>[U V]=dftuv(PQ(1),PQ(2));
- >>D0=0.05*PQ(2);
- >>F=fft2(f,PQ(1),PQ(2));
- >>H=exp(-(U.^2+V.^2)/(2*(D0^2)));
- >>g=dftfilt(f,H);
- >>imshow(g,[]);

Generating Filters Directly in the Frequency Domain(cont.)



a b
c d

FIGURE 4.13
Lowpass filtering. (a) Original image. (b) Gaussian lowpass filter shown as an image. (c) Spectrum of (a). (d) Processed image.

Image Analysis and Computer Vision
Lecture 5、Image Enhencement in Frequency Domain(I)

Weiqiang Wang

School of Computer and Control Engineering, UCAS

September 30, 2014

Outline

2-D Fourier Transform

- Any function that periodically repeats itself can be expressed as the sum of sines and/or cosines of different frequencies, each multiplied by a different coefficient (Fourier series).
- Even functions that are not periodic (but whose area under the curve is finite) can be expressed as the integral of sines and/or cosines multiplied by a weighting function (Fourier transform).

- The frequency domain refers to the plane of the two dimensional discrete Fourier transform of an image.
- The purpose of the Fourier transform is to represent a signal as a linear combination of sinusoidal signals of various frequencies.

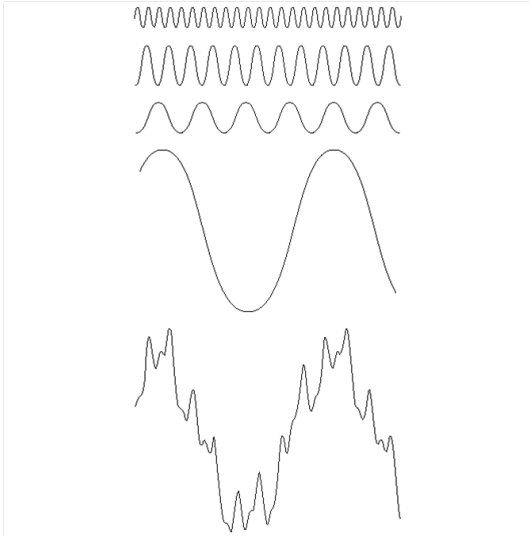


FIGURE 4.1 The function at the bottom is the sum of the four functions above it. Fourier's idea in 1807 that periodic functions could be represented as a weighted sum of sines and cosines was met with skepticism.

2-D Continuous Fourier Transform

- The one-dimensional Fourier transform and its inverse
  - Fourier transform
    $$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx, where\ j = \sqrt{-1}$$
  - Inverse Fourier transform:
    $$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du \qquad e^{j\theta} = \cos\theta + j\sin\theta$$
- The two-dimensional Fourier transform and its inverse
  - Fourier transform
    $$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} dxdy$$
  - Inverse Fourier transform:
    $$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v) e^{j2\pi(ux+vy)} dudv$$

2-D Discrete Fourier Transform

- The one-dimensional Discrete Fourier transform (DFT) and its inverse
  - Fourier transform

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-\frac{j2\pi ux}{M}} \quad for\ u = 0, 1, 2, ..., M-1$$

  - Inverse Fourier transform:

$$f(x) = \sum_{u=0}^{M-1} F(u)e^{\frac{j2\pi ux}{M}} \quad for\ x = 0, 1, 2, ..., M-1$$

- Since $e^{j\theta} = \cos\theta + j\sin\theta$, then DFT can be redefined as

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)[\cos\frac{2\pi ux}{M} - j\sin\frac{2\pi ux}{M}]$$
$$for\ u = 0, 1, 2, ..., M-1$$

  - Frequency (time) domain: the domain (values of u) over which the values of $F(u)$ range; because u determines the frequency of the components of the transform.
  - Frequency (time) component: each of the M terms of $F(u)$.

2-D Discrete Fourier Transform

- $F(u)$ can be expressed in polar coordinates:

$$F(u) = |F(u)|e^{j\phi(u)}$$

$$where\ |F(u)| = [R(u) + I(u)]^{\frac{1}{2}}\ (magnitude\ or\ spectrum)$$

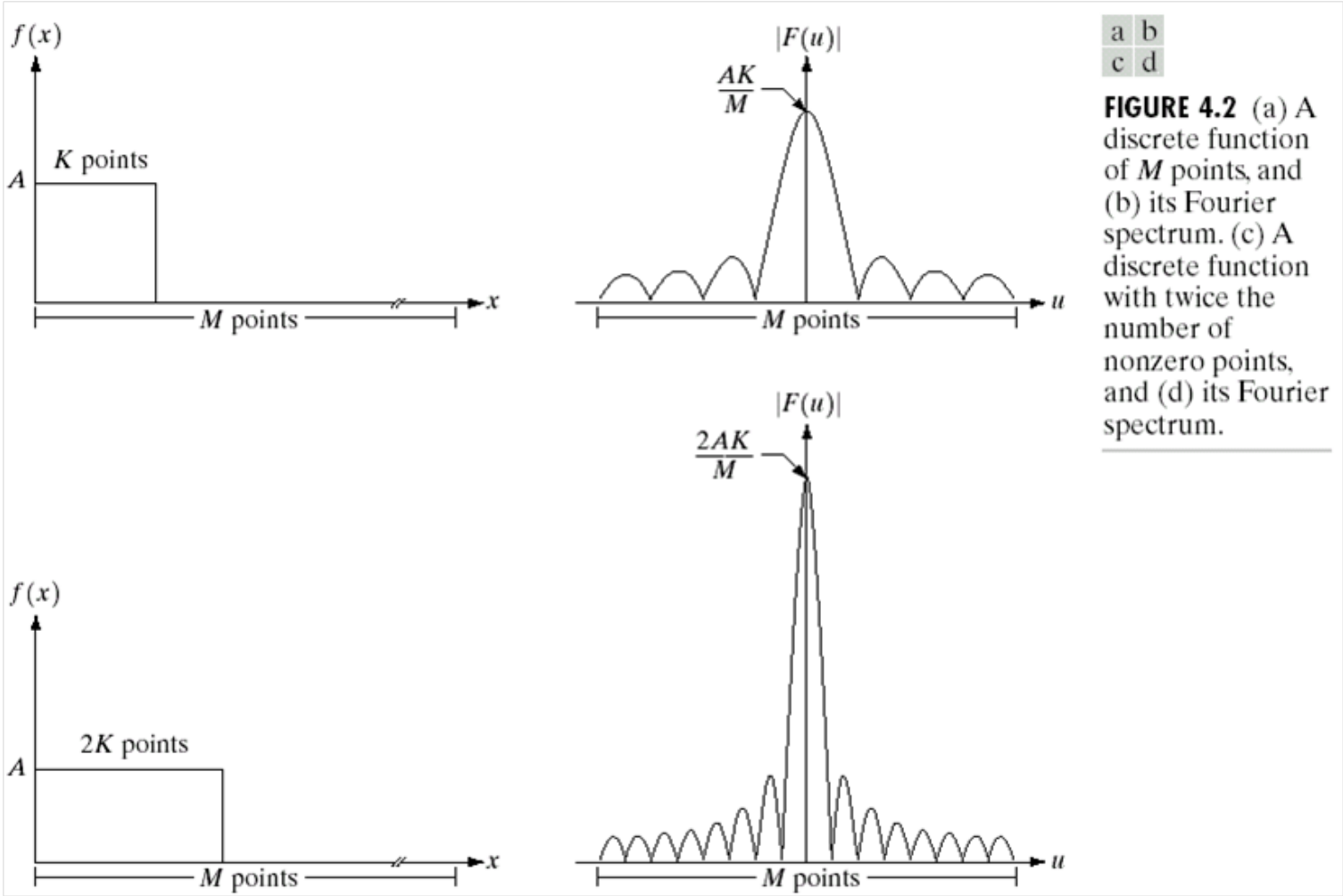
$$\phi(u) = \tan^{-1}[\frac{I(u)}{R(u)}]\ (phase\ angle\ or\ phase\ spectrum)$$

  - $I(u)$: the imaginary part of $F(u)$.
  - $R(u)$: the real part of $F(u)$.

- Power spectrum:

$$P(u) = |F(u)|^2 = R^2(u) + I^2(u)$$

2-D Discrete Fourier Transform



a b
c d

FIGURE 4.2 (a) A discrete function of M points, and (b) its Fourier spectrum. (c) A discrete function with twice the number of nonzero points, and (d) its Fourier spectrum.

2-D Discrete Fourier Transform

- The two-dimensional Fourier transform and its inverse
  - Fourier transform (discrete case)DFT

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$
$$for\ u = 0,1,2,...,M-1,\ v = 0,1,2,...,N-1$$

  - Inverse Fourier transform:

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$
$$for\ x = 0,1,2,...,M-1,\ y = 0,1,2,...,N-1$$

    - u,v: the transform or frequency variables
    - x,y: the spatial or image variables

2-D Discrete Fourier Transform

- We define the Fourier spectrum, phase angle, and power spectrum as follows:

$$|F(u,v)|^2 = [R^2(u,v) + I^2(u,v)]^{\frac{1}{2}} \quad (spectrum)$$

$$\phi(u,v) = \tan^{-1}[\frac{I(u,v)}{R(u,v)}] \quad (phase\ angle)$$

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v) \quad (powerspectrum)$$

  - $I(u,v)$: the imaginary part of $F(u,v)$.
  - $R(u,v)$: the real part of $F(u,v)$.

Properties of 2-D DFT

- Time-shifting

$$\Im[f(x-x_0, y-y_0)] = F(u,v)e^{-j2\pi(\frac{ux_0}{M}+\frac{vy_0}{N})}$$

- Frequency shifting

$$\Im[f(x,y)e^{j2\pi(\frac{u_0x}{M}+\frac{v_0y}{N})}] = F(u-u_0, v-v_0)$$

$$\Im[f(x,y)(-1)^{x+y}] = F(u-\frac{M}{2}, v-\frac{N}{2})$$

- Average and Symmetry

$$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) \quad (average)$$

$$F(u,v) = F^*(-u,-v) \quad (conjugate\ symmetric)$$

$$|F(u,v)| = |F(-u,-v)| \quad (symmetric)$$

Properties of 2-D DFT (cont.)

- Seperability

$$
\begin{aligned}
F(u,v) &= \Im[f(x,y)] \\
&= \Sigma_y[\Sigma_x f(x,y) exp(-j2\pi \tfrac{xu}{M})] exp(-j2\pi \tfrac{yv}{N}) \\
&= \Sigma_y F(u,y) exp(-j2\pi \tfrac{yv}{N})
\end{aligned}
$$

The 2D DFT $F(u,v)$ can be obtained by

1. Taking the 1D DFT of every row of image $f(x,y)$, $F(u,y)$.
2. The 1D DFT of every column of $F(u,y)$.

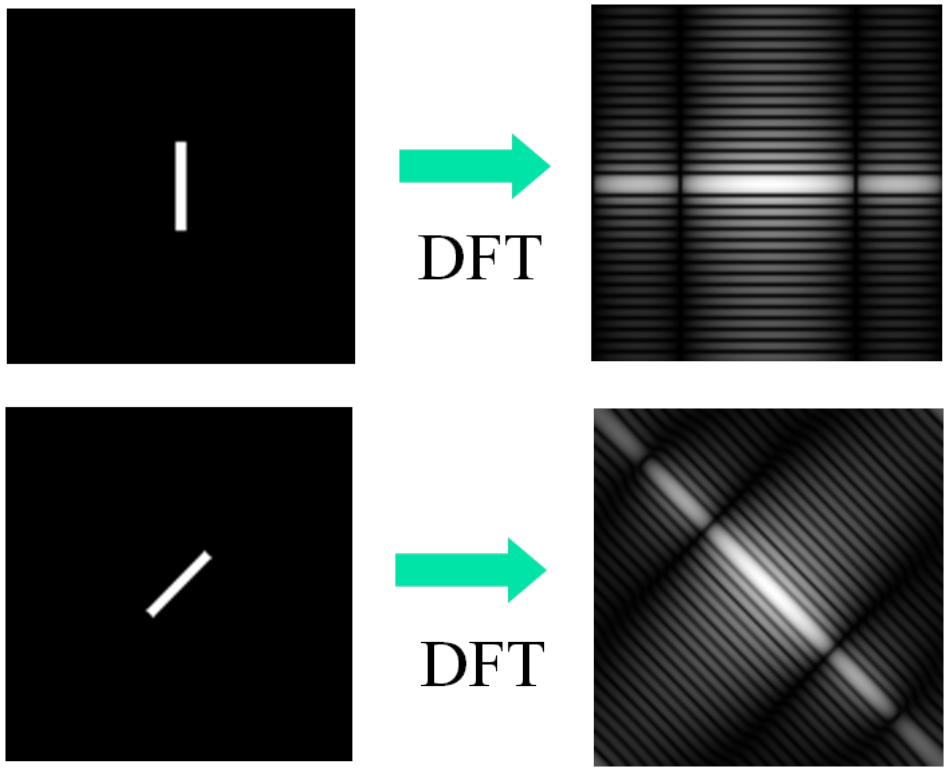
(a)f(x,y) (b) F(u,y) (c)F(u,v)

Properties of 2-D DFT (cont.)

- Rotation

let $x = r\cos\theta, y = r\sin\theta, u = \omega\cos\varphi, v = \omega\sin\varphi$

$$f(r, \theta + \theta_0) \Leftrightarrow F(\omega, \varphi + \theta_0)$$



Properties of 2-D DFT (cont.)

- Periodicity

$$f(x,y) = f(x+M,y) = f(x,y+N) = f(x+M,y+N)$$

$$F(u,v) = F(u+M,v) = F(u,v+N) = F(u+M,v+N)$$

- Linearity

$$\Im(af(x,y) + bg(x,y)) = a\Im(f(x,y)) + b\Im(g(x,y))$$

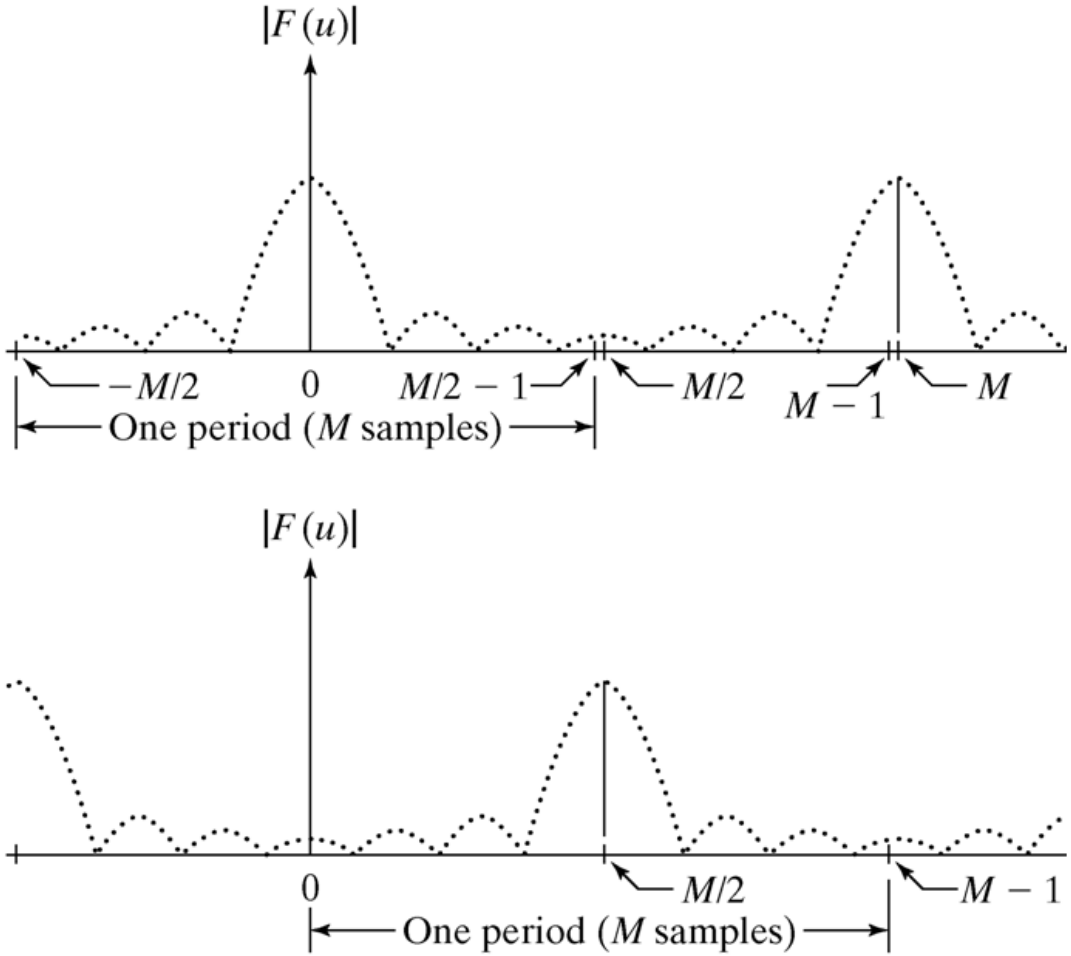
- Differentiation

$$\Im(\frac{\partial^n f(x,y)}{\partial x^n}) = (j2\pi u)^n \Im(f(x,y)) = (j2\pi u)^n F(u,v)$$

$$\Im((-j2\pi u)^n f(x,y)) = \frac{\partial^n F(u,v)}{\partial u^n}$$

$$\Im(\nabla^2 f(x,y)) = -4\pi^2(u^2+v^2)F(u,v)$$

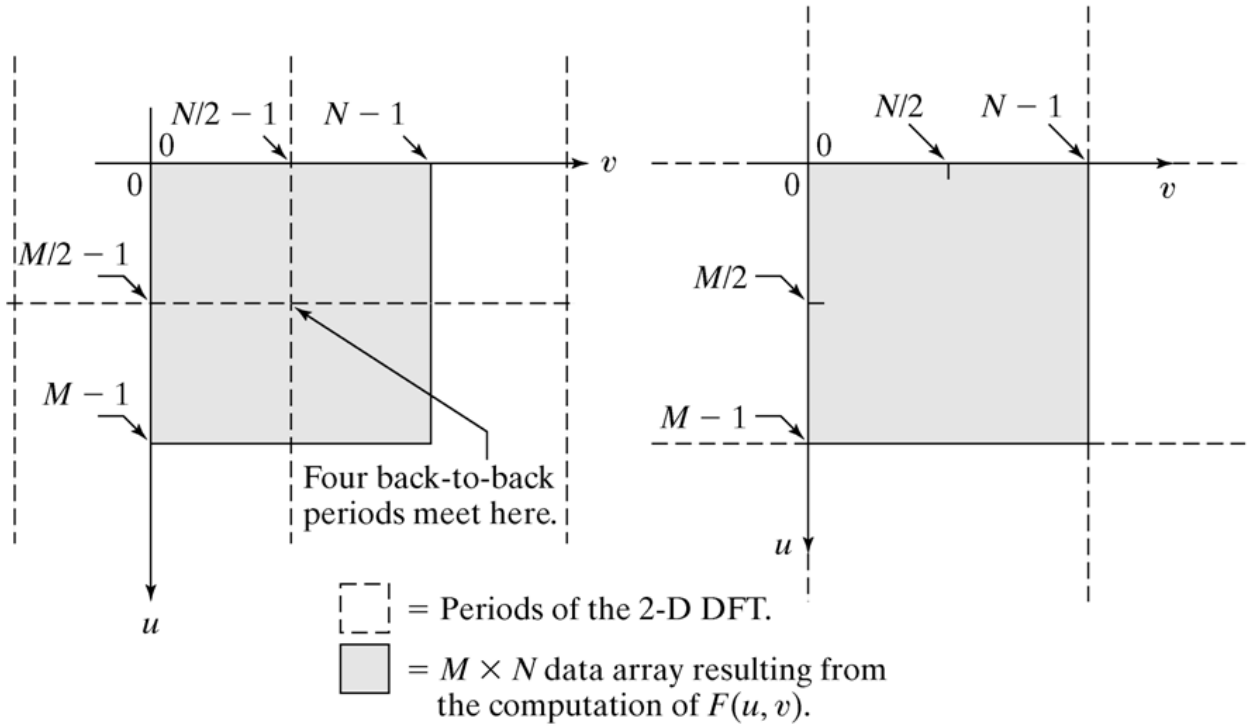
2-D Discrete Fourier Transform



a
b

FIGURE 4.1
(a) Fourier spectrum showing back-to-back half periods in the interval $[0, M-1]$. (b) Centered spectrum in the same interval, obtained by multiplying $f(x)$ by $(-1)^x$ prior to computing the Fourier transform.

2-D Discrete Fourier Transform (cont.)



a b

FIGURE 4.2 (a) $M \times N$ Fourier spectrum (shaded), showing four back-to-back quarter periods contained in the spectrum data. (b) Spectrum obtained by multiplying $f(x, y)$ by $(-1)^{x+y}$ prior to computing the Fourier transform. Only one period is shown shaded because this is the data that would be obtained by an implementation of the equation for $F(u, v)$.

Properties of 2-D DFT (cont.)

- Convolution

$$\Im(f(x,y) * g(x,y)) = F(u,v)G(u,v)$$

$$\Im(f(x,y)g(x,y)) = F(u,v) * G(u,v)$$

- Correlation

$$\Im(f(x,y) \circ g(x,y)) = F^*(u,v)G(u,v)$$

$$\Im(f(x,y) \circ f(x,y)) = |F(u,v)|^2$$

$$\Im(f^*(x,y)g(x,y)) = F(u,v) \circ G(u,v)$$

$$\Im(|f(x,y)|^2) = F(u,v) \circ F(u,v)$$

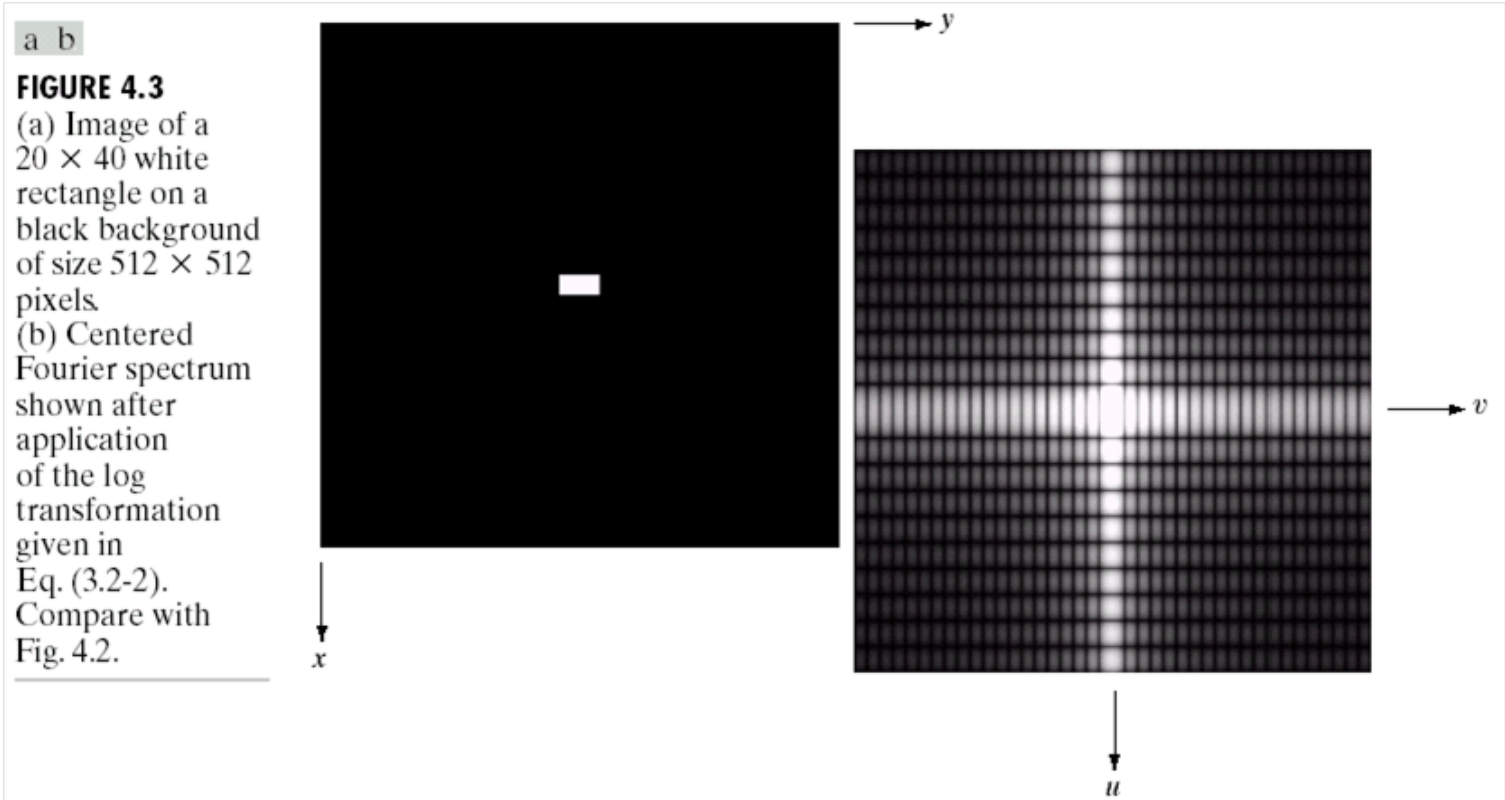
- Similarity

$$\Im(f(ax,by)) = \frac{1}{|ab|}F(\frac{u}{a},\frac{v}{b})$$

Some useful FT pairs

- $\delta(x,y) \Leftrightarrow 1$
- $A2\pi\sigma^2 exp(-2\pi^2\sigma^2(x^2+y^2)) \Leftrightarrow Aexp(-\frac{(u^2+v^2)}{2\sigma^2})$
  $exp(-\pi(x^2+y^2)) \Leftrightarrow exp(-\pi(u^2+v^2))$
- $\cos(2\pi u_0 x + 2\pi v_0 y) \Leftrightarrow \frac{1}{2}[\delta(u+u_0, v+v_0) + \delta(u-u_0, v-v_0)]$
- $\sin(2\pi u_0 x + 2\pi v_0 y) \Leftrightarrow \frac{1}{2}j[\delta(u+u_0, v+v_0) - \delta(u-u_0, v-v_0)]$

2-D Discrete Fourier Transform



2-D DFT in Matlab

- The DFT and its inverse are obtained in practice using a Fast Fourier Transform(FFT) algorithm. The FFT of an $M \times N$ image array f is obtained in the toolbox with function fft2,which has the simple syntax:

  F=fft2(f)

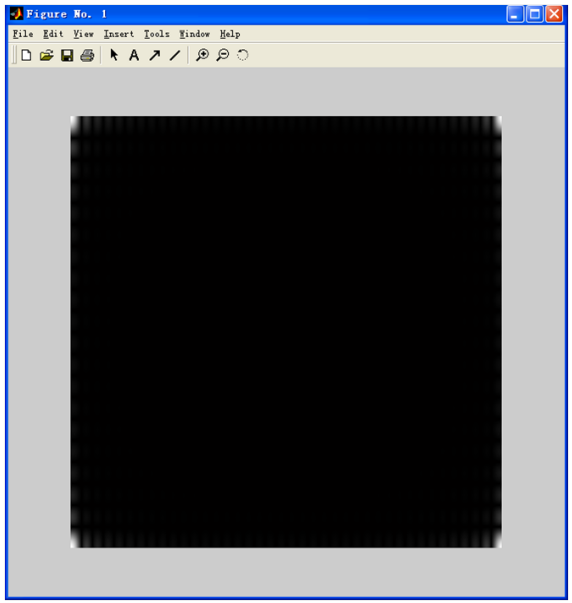
  This function returns a Fourier transform that is also of size $M \times N$, with the origin of the data at the top left, and with four quarter periods meeting at the center of the frequency rectangle.

- The Fourier spectrum is obtained by using function abs:

  S=abs(F)

2-D DFT in Matlab(cont.)

- f=imread('Fig0403(a)(image).tif');
- F=fft2(f);
- S=abs(F);
- imshow(S,[])

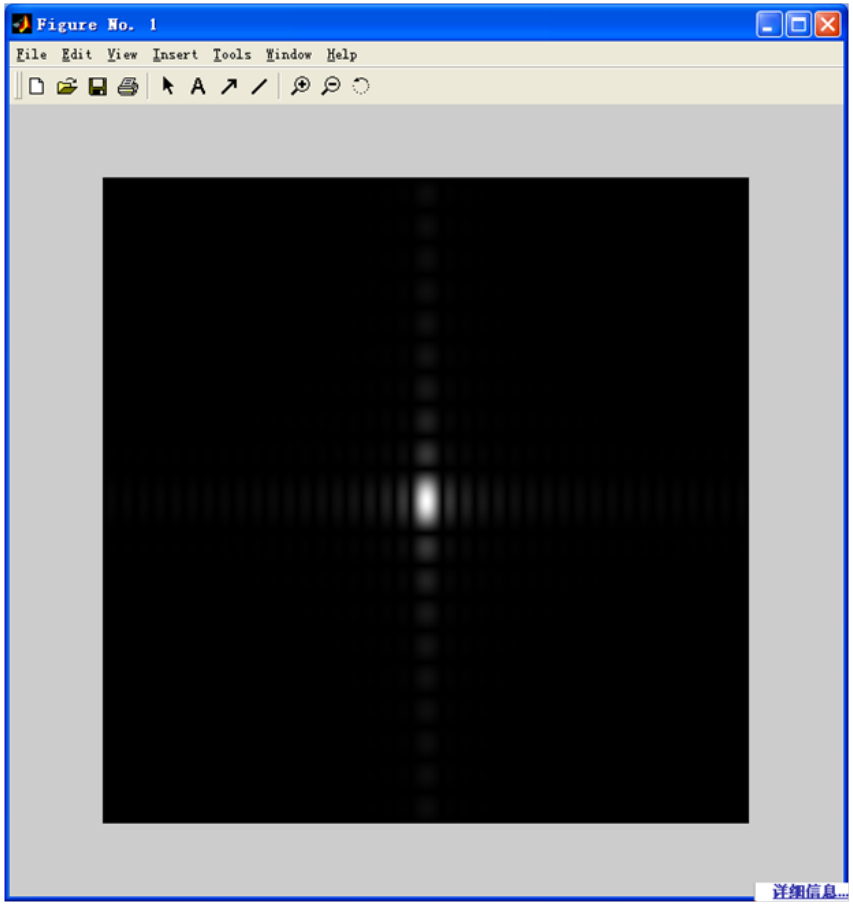


2-D DFT in Matlab(cont.)

- Fc=fftshift(F);
- imshow(abs(Fc),[])

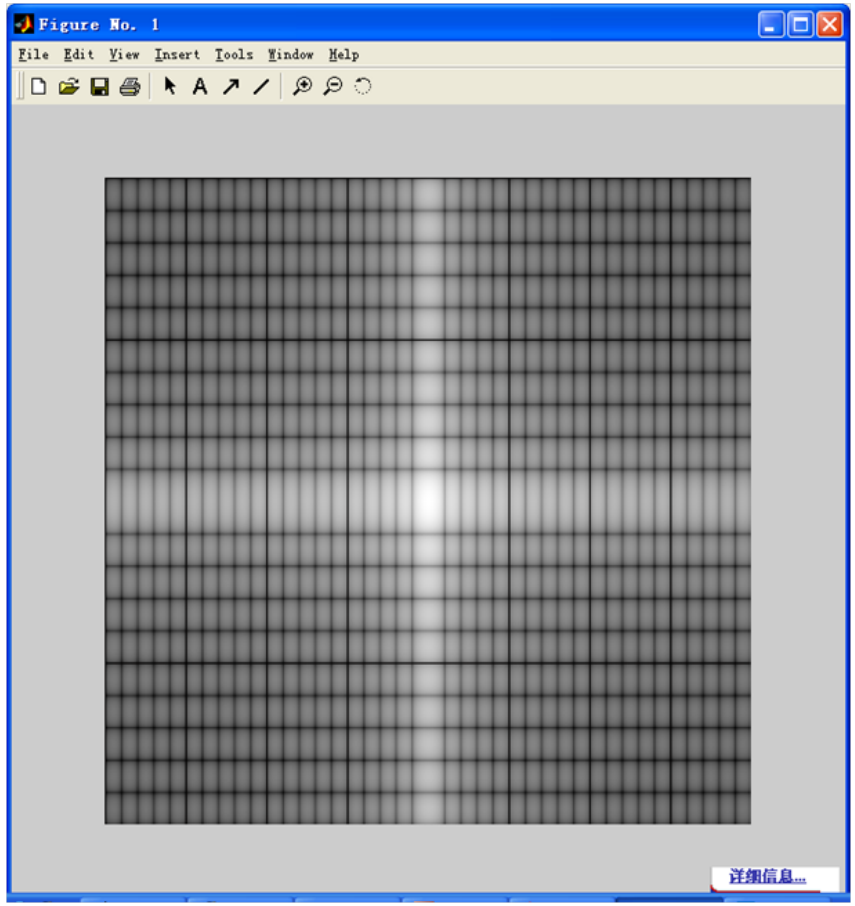
  The net result of using fftshift is the same as if the input image had been multiplied by $(-1)^{x+y}$ *prior to computing the transform.*

  Note,however,that the two processes are not interchangeable.

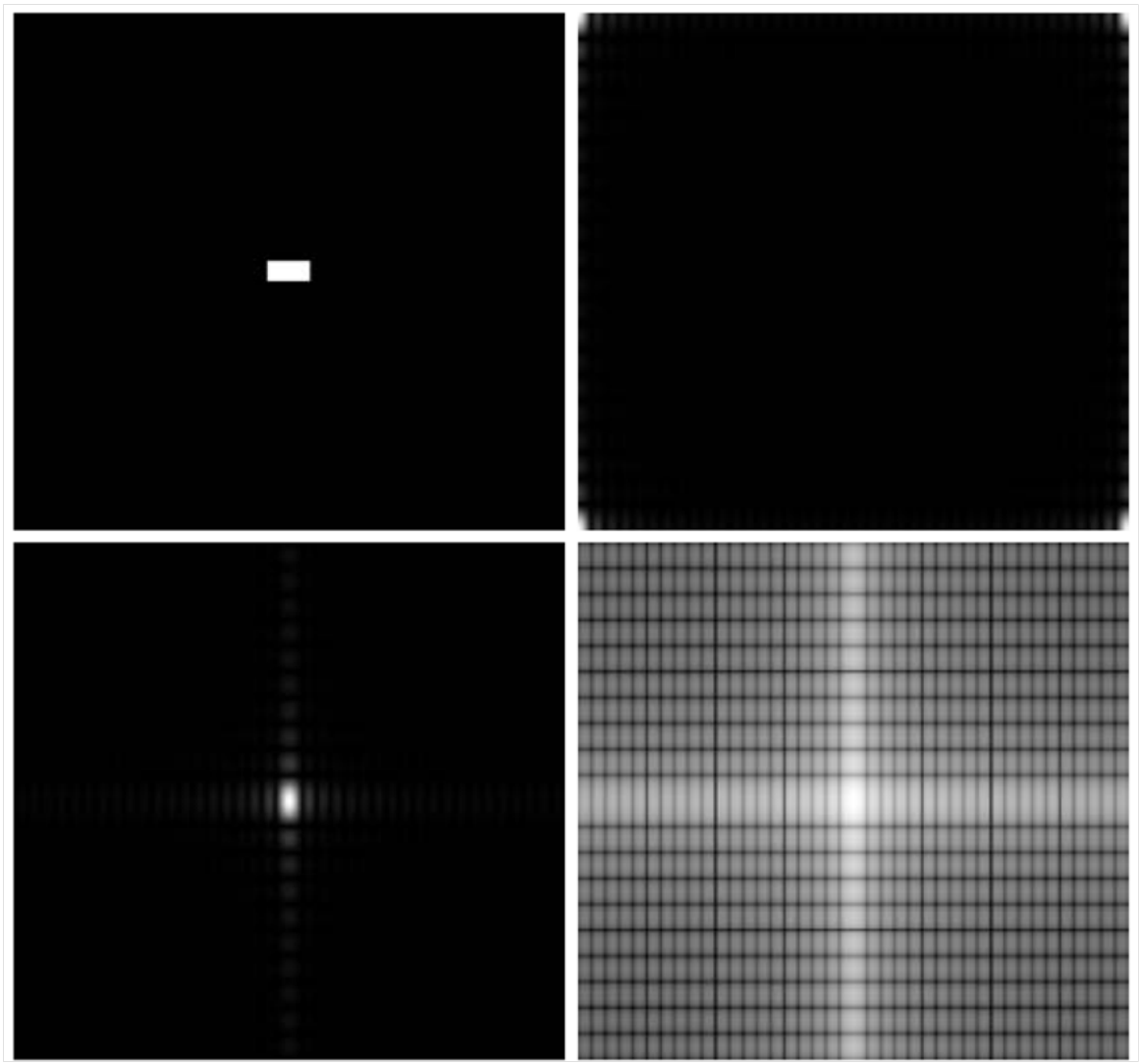


2-D DFT in Matlab(cont.)

- S2=log(1+abs(Fc));
- imshow(S2,[])



2-D DFT in Matlab(cont.)



a b
c d

FIGURE 4.3
(a) A simple image.
(b) Fourier spectrum.
(c) Centered spectrum.
(d) Spectrum visually enhanced by a log transformation.

2-D DFT in Matlab(cont.)

- We point out that the inverse Fourier transform is computed using function if ifft2. which has the basic syntax

  f=ifft2(F)

- In practice, the output of ifft2 often has very small imaginary components resulting from round-off errors. Thus, it is good practice to extract the real part of the result.

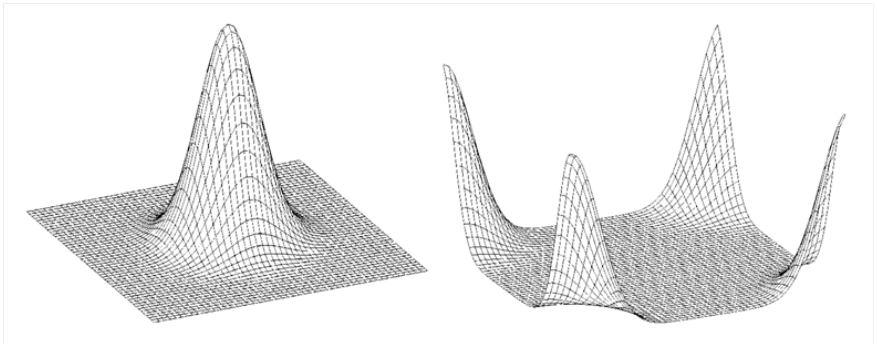
  f=real(ifft2(F))

Fundamental Concepts

- The foundation for linear filtering in both the spatial and frequency domains is the convolution theorem, which may be written as.

$$f(x,y) * h(x,y) \Longleftrightarrow F(u,v)H(u,v)$$

- The idea in frequency domain filtering is to select a filter transfer function that modifies $F(u,v)$ in a specified manner.
- For example, the lowpass filter in figure 4.4



a b

FIGURE 4.4 Transfer functions of (a) a centered lowpass filter, and (b) the format used for DFT filtering. Note that these are frequency domain filters.

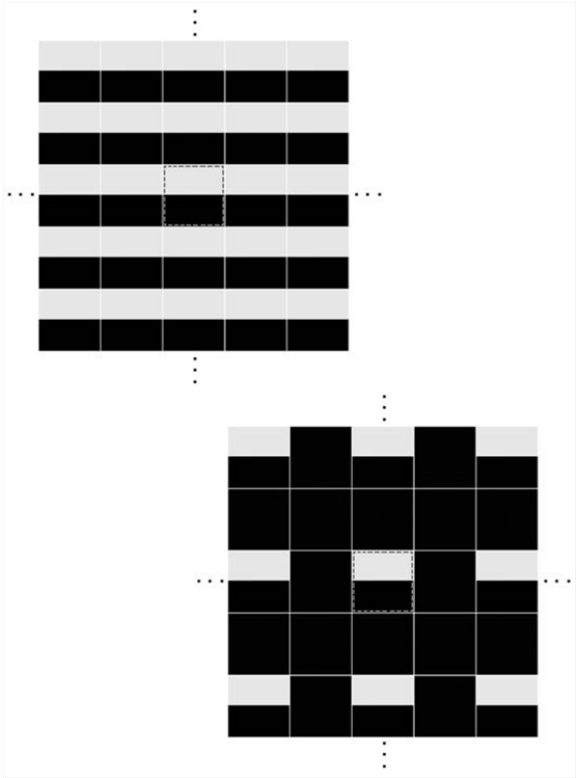
Fundamental Concepts(cont.)

- Based on the convolution theorem, we know that to obtain the corresponding filtered image in the spatial domain we simply compute the inverse Fourier transform of the product $H(u,v)F(u,v)$.
- Convolving periodic functions can cause interference of the nonzero periods if the periods are close with respect to the duration of the nonzero parts of the functions. This interference, called *wraparound error*, can be avoided by padding the functions with zeros.
- For example, the lowpass filter in figure 4.4

a b c

FIGURE 4.5 (a) A simple image of size 256 × 256. (b) Image lowpass-filtered in the frequency domain without padding. (c) Image lowpass-filtered in the frequency domain with padding. Compare the light portion of the vertical edges in (b) and (c).

Fundamental Concepts(cont.)



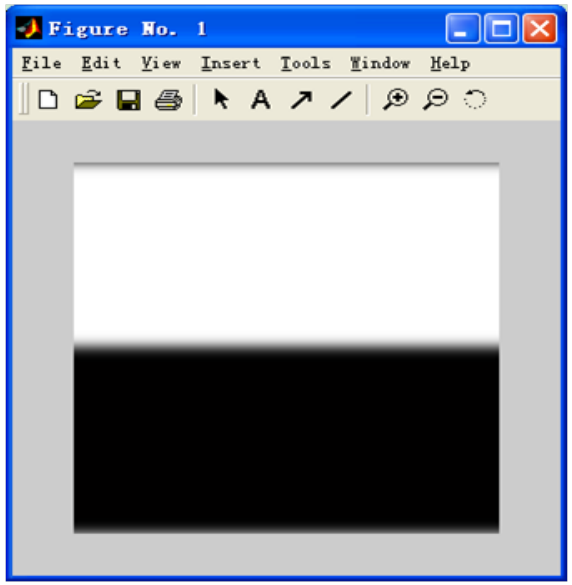
a
b

FIGURE 4.6
(a) Implied, infinite periodic sequence of the image in Fig. 4.5(a). The dashed region represents the data processed by `fft2`. (b) The same periodic sequence after padding with 0s. The thin white lines in both images are shown for convenience in viewing; they are not part of the data.

FIGURE 4.7 Full padded image resulting from `ifft2` after filtering. This image is of size 512×512 pixels.

Fundamental Concepts(cont.)

- Image lowpass-filtered in the frequency domain without padding
  - f=imread('Fig0405(a)(square_original).tif');
  - [m n]=size(f)
  - F=fft2(f);
  - H=lpfilter('gaussian',m,n,10);
  - G=H.*F;
  - g=real(ifft2(G))
  - imshow(g,[])



Basic Steps in DFT Filtering

- 1. Obtain the padding parameters using function paddedsize:

  PQ=paddedsize(size(f));
- 2. Obtain the Fourier transform with padding:

  F=fft2(f,PQ(1),PQ(2));
- 3. Generate a filter function, H, of size PQ(1)×PQ(2) using any of the methods discussed in the remainder of this chapter.

  The filter must be in the format shown in Fig. 4.4(b). If it is centered instead, as in Fig. 4.4(a),let H = fftshift (H) before using the filter.
- 4. Multiply the transform by the filter:G = H.*F;
- 5. Obtain the real part of the inverse FFT of G g=real(ifft2(G));
- 6. Crop the top, left rectangle to the original size:

  g=g(1:size(f,1),1:size(f,2));

Basic Steps in DFT Filtering(cont.)

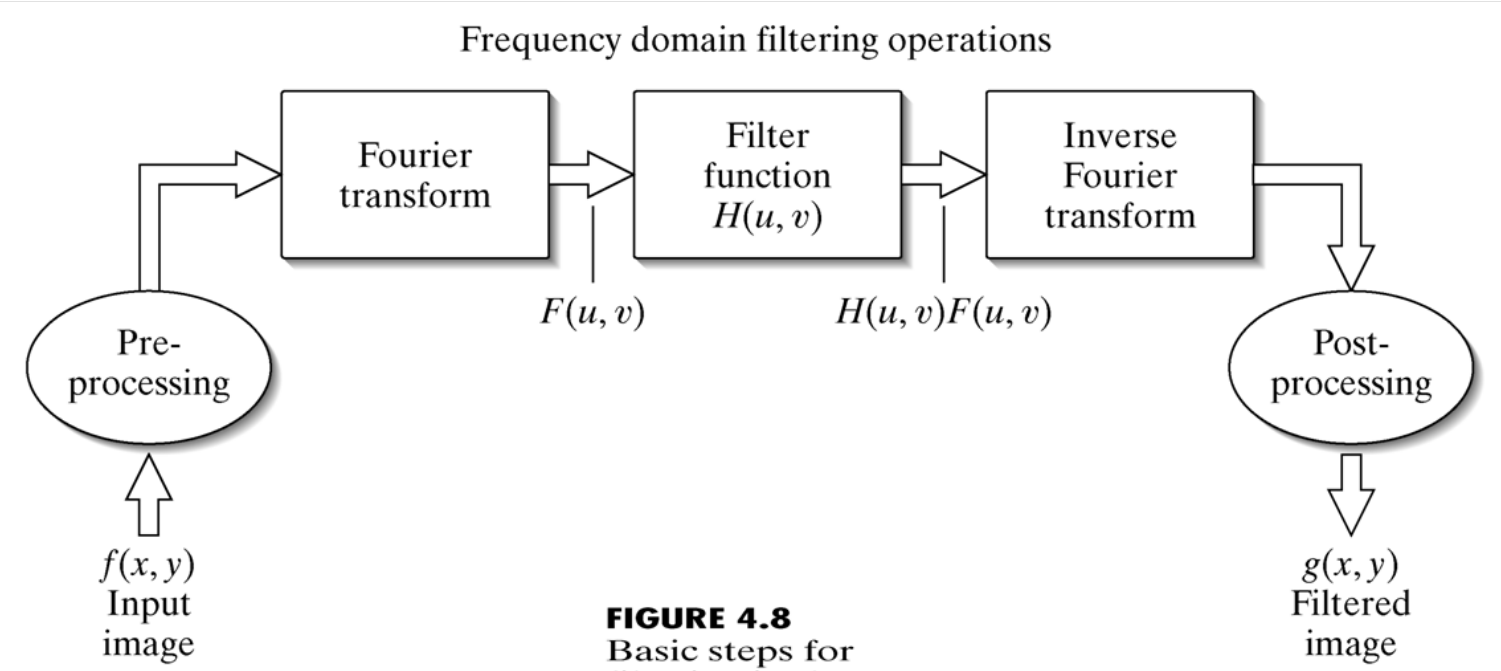


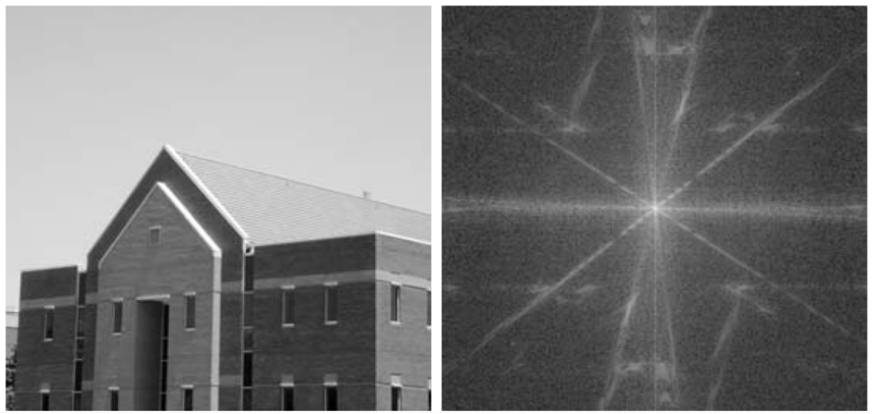
FIGURE 4.8 Basic steps for filtering in the frequency domain.

Obtaining Frequency Domain Filters from Spatial Filters

- Why obtains Frequency Domain Filters from Spatial Filters?
  - Efficiency
  - Meaningful comparisons
- How?
  - How to convert spatial filters into equivalent frequency domain filters;
  - How to compare the results between spatial domain filtering using imfilter, and frequency domain filtering.freqz2

Obtaining Frequency Domain Filters from Spatial Filters(cont.)

- >>f=imread('Fig0409(a)(bld).tif');
- >>F=fft2(f);
- >>S=fftshift(log(1+abs(F)));
- >>S=gscale(S);
- >>imshow(S)

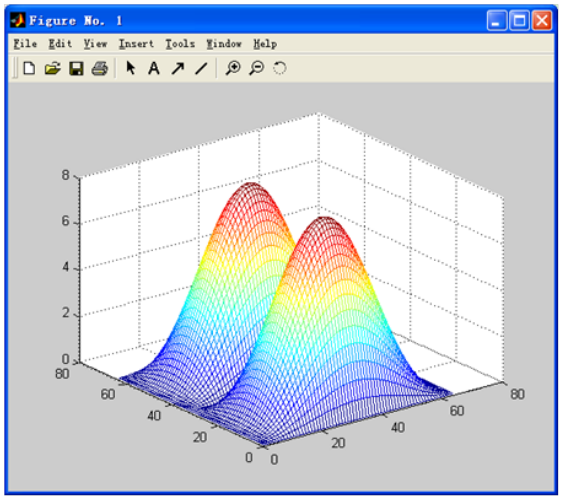


a b

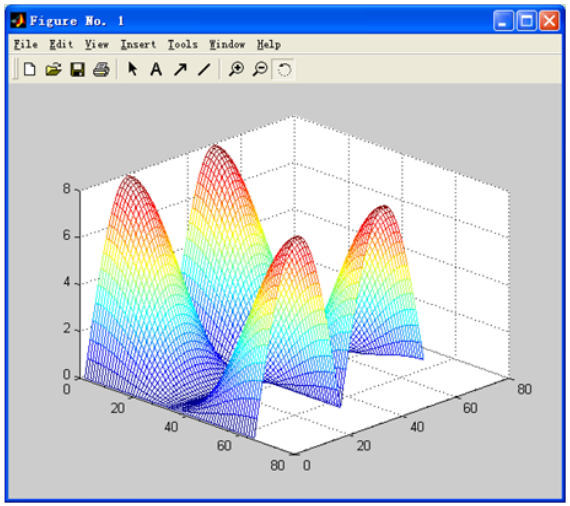
FIGURE 4.9
(a) A gray-scale image. (b) Its Fourier spectrum.

Obtaining Frequency Domain Filters from Spatial Filters(cont.)

- h=fspecial('sobel');
- H=freqz2(h);
- mesh(abs(H))



- H1=ifftshift(H);
- mesh(abs(H1))
- view(45,30)



Obtaining Frequency Domain Filters from Spatial Filters(cont.)

Spatial domain	Frequent domain
gs=imfilter(double(f),h);	PQ=paddedsize(size(f)); H=freqz2(h,PQ(1),PQ(2)); H1=ifftshift(H); gf=dftfilt(f,H1);

- d=abs(gs-gf);
- max(d(:))

  ans=

  5.4015e-012
- min(d(:))

  ans=

  0

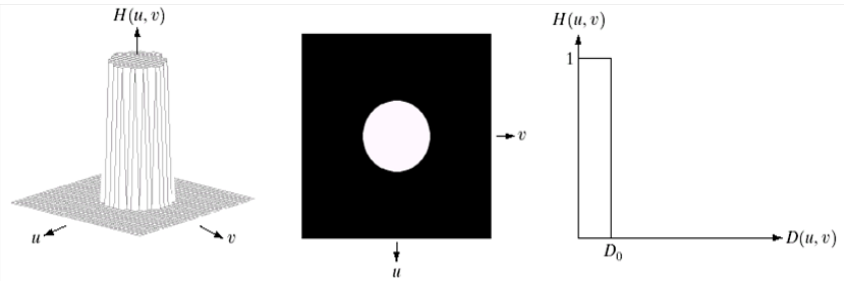
Generating Filters Directly in the Frequency Domain

- Ideal lowpass filter(ILPF)

$$H(u,v) = \begin{cases} 1 & if D(u,v) \le D_0 \\ 0 & if D(u,v) > D_0 \end{cases}$$

where $D(u,v)$ *: the distance from point* (u,v) *to the center of the frequency rectangle*

$$D(u,v) = [(u - M/2)^2 + (v - N/2)^2]^{\frac{1}{2}}$$



a b c

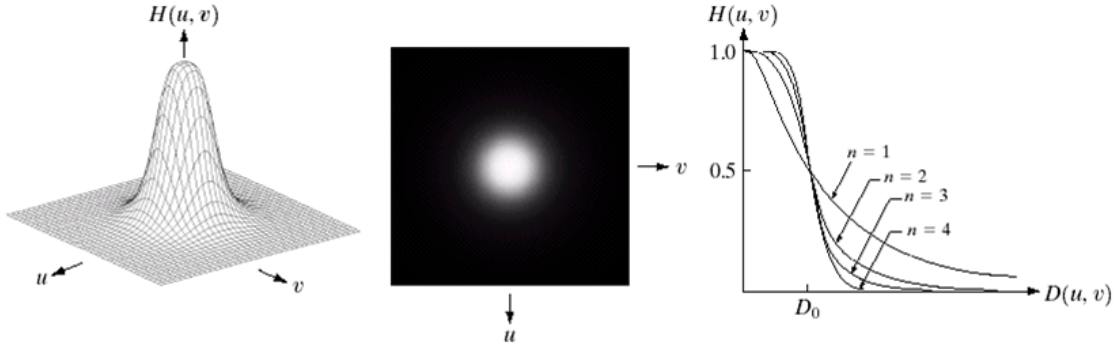
FIGURE 4.10 (a) Perspective plot of an ideal lowpass filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross section.

- Create meshgrid arrays for use in implementing Filters in the frequency domain dftuv

Generating Filters Directly in the Frequency Domain(cont.)

- Butterworth Lowpass Filters (BLPFs) With order n

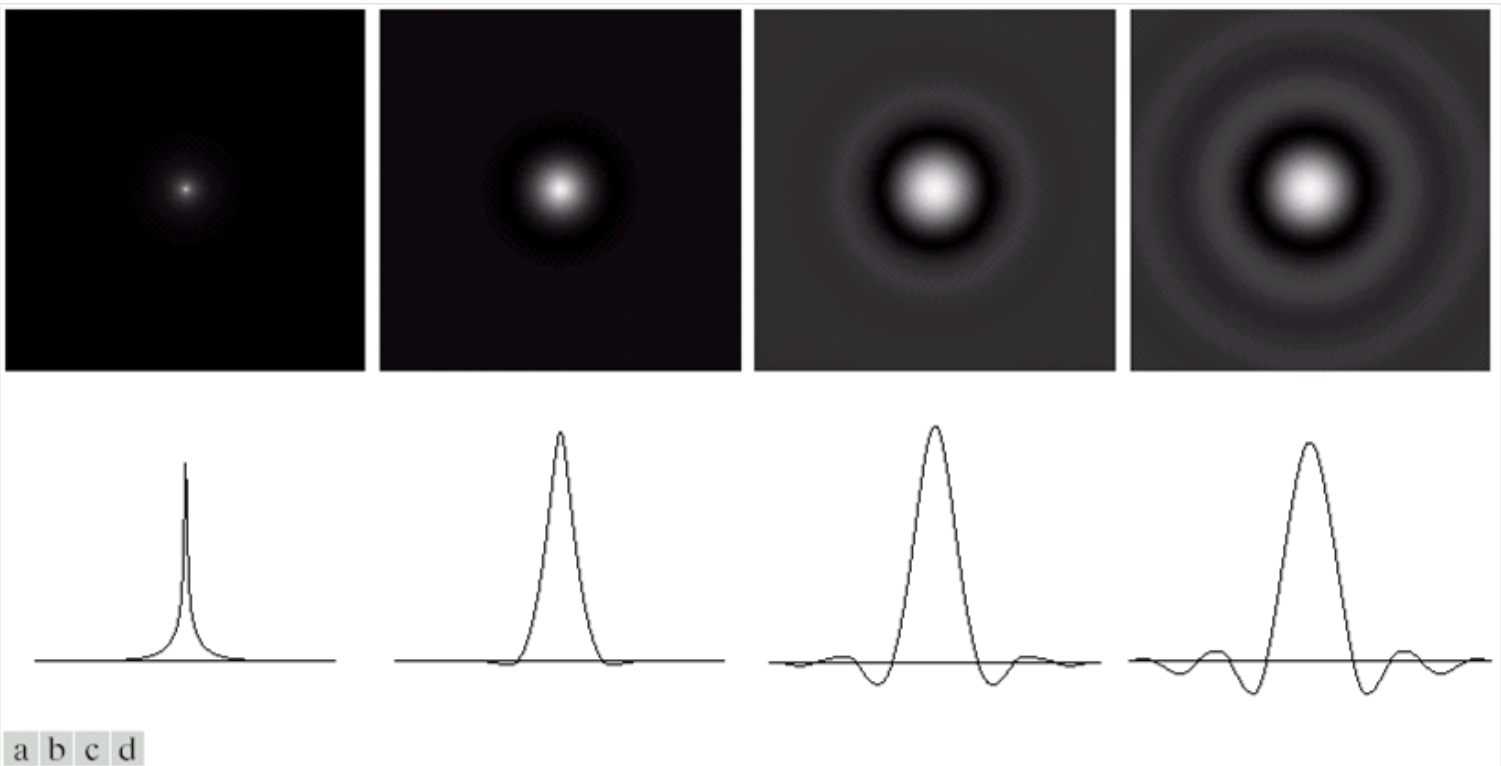
$$H(u,v) = \frac{1}{1+[D(u,v)/D_0]^{2n}}$$



a b c

FIGURE 4.14 (a) Perspective plot of a Butterworth lowpass filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

Generating Filters Directly in the Frequency Domain(cont.)

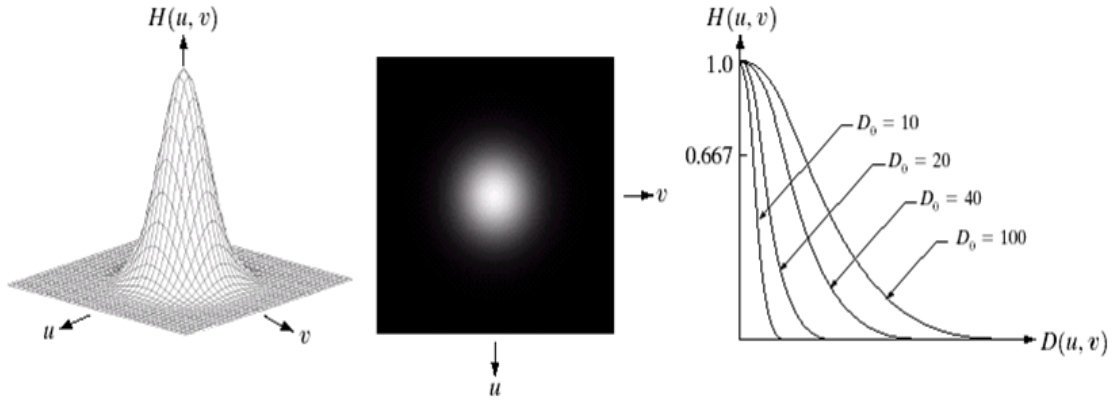


a b c d

FIGURE 4.16 (a)–(d) Spatial representation of BLPFs of order 1, 2, 5, and 20, and corresponding gray-level profiles through the center of the filters (all filters have a cutoff frequency of 5). Note that ringing increases as a function of filter order.

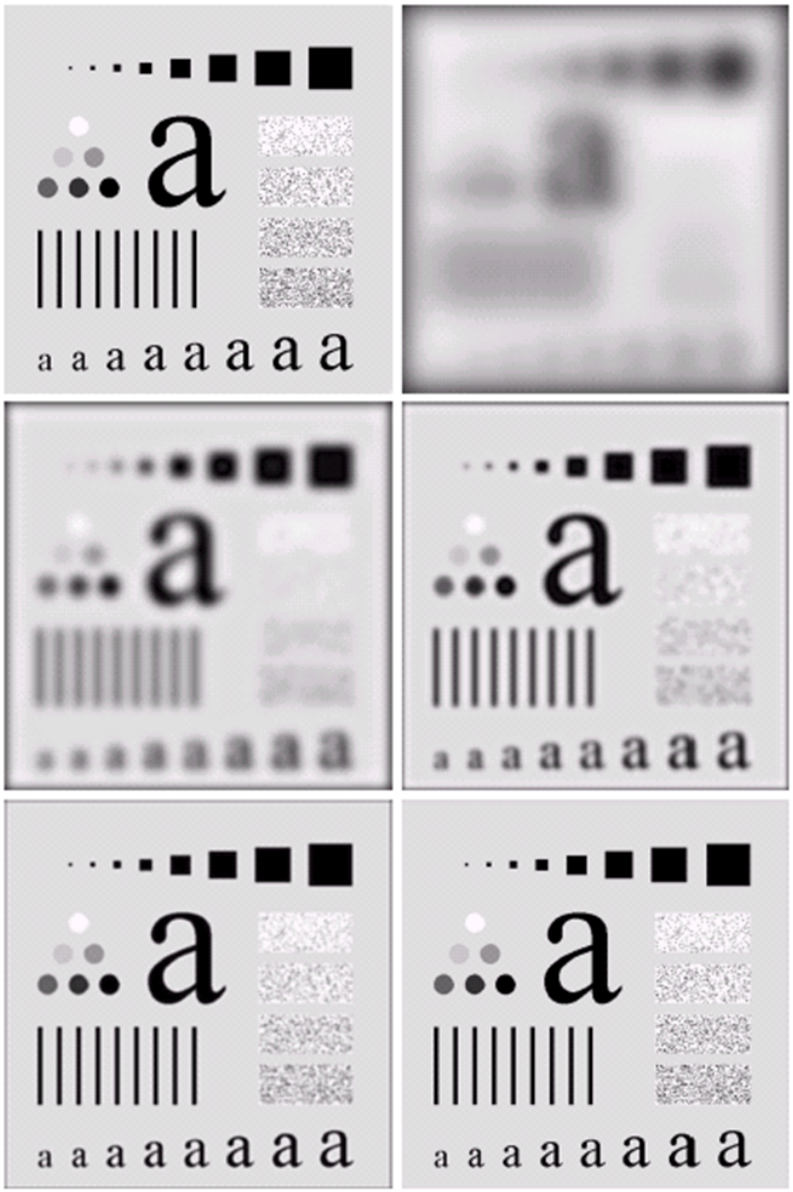
Gaussian Lowpass Filters (GLPFs)

- $H(u,v) = e^{-D^2(u,v)/2D_0^2}$



a b c

FIGURE 4.17 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.



a b c d e f

FIGURE 4.15 (a) Original image. (b)–(f) Results of filtering with BLPFs of order 2, with cutoff frequencies at radii of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Fig. 4.12.

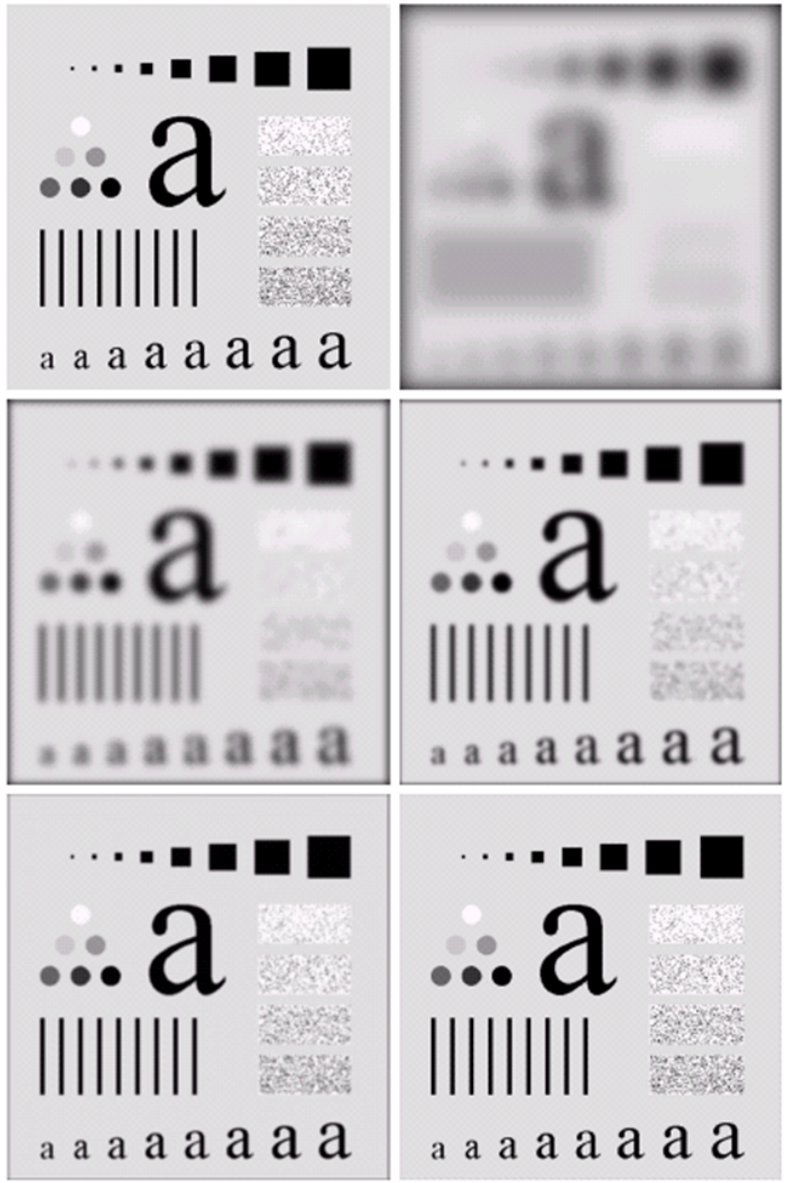


FIGURE 4.18 (a) Original image. (b)–(f) Results of filtering with Gaussian lowpass filters with cutoff frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Figs. 4.12 and 4.15.

a b c d e f

Examples of Lowpass Filtering

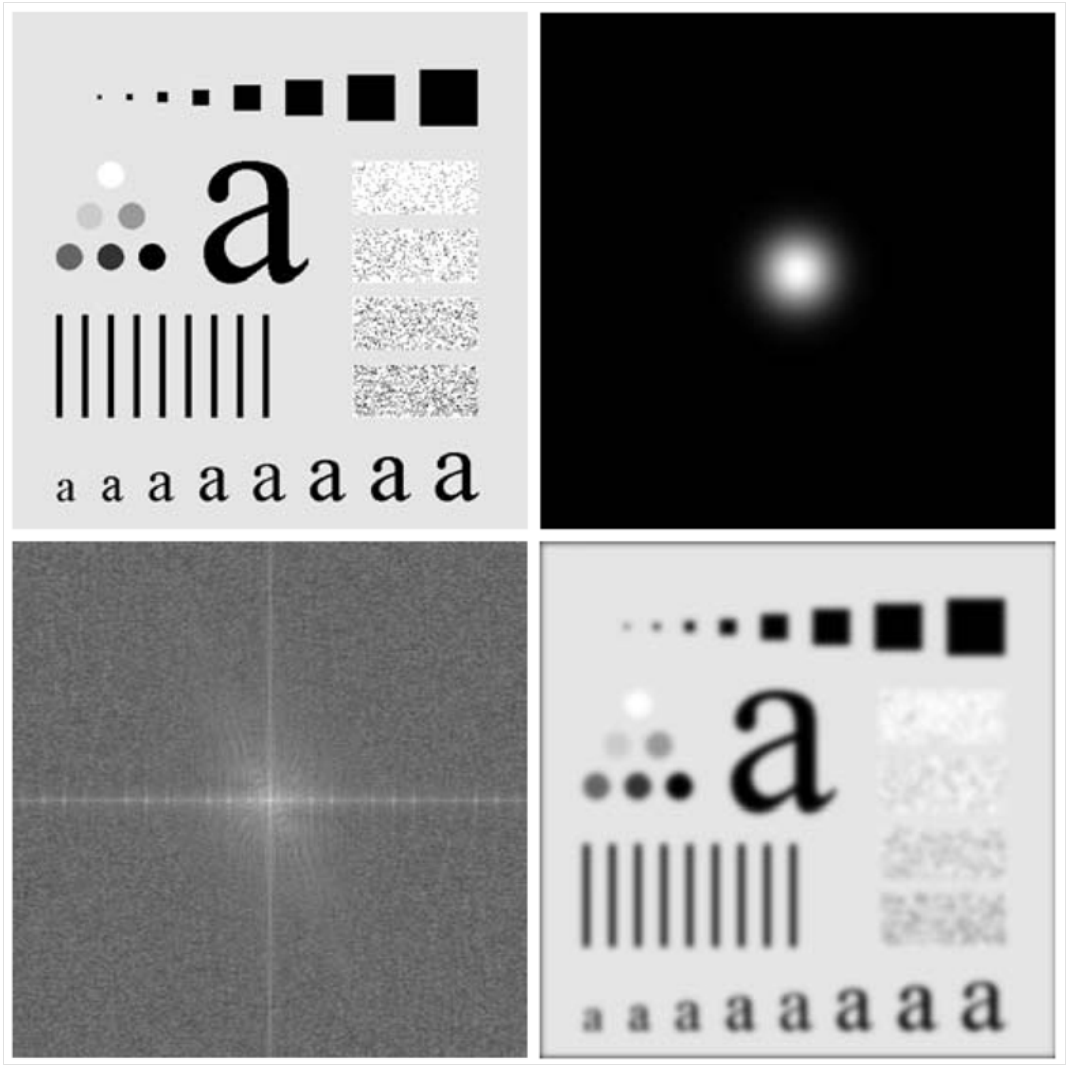
a b c

FIGURE 4.20 (a) Original image (1028 × 732 pixels). (b) Result of filtering with a GLPF with $D_0 = 100$. (c) Result of filtering with a GLPF with $D_0 = 80$. Note reduction in skin fine lines in the magnified sections of (b) and (c).

Generating Filters Directly in the Frequency Domain

- >>f=imread('Fig0413(a)(original_test_pattern).tif');
- >>PQ=paddedsize(size(f));
- >>[U V]=dftuv(PQ(1),PQ(2));
- >>D0=0.05*PQ(2);
- >>F=fft2(f,PQ(1),PQ(2));
- >>H=exp(-(U.^2+V.^2)/(2*(D0^2)));
- >>g=dftfilt(f,H);
- >>imshow(g,[]);

Generating Filters Directly in the Frequency Domain(cont.)



a b
c d

FIGURE 4.13
Lowpass filtering.
(a) Original image.
(b) Gaussian lowpass filter shown as an image.
(c) Spectrum of (a). (d) Processed image.

Sharpening Frequency Domain Filters

- General high-pass frequency domain filters

  $H_{hp}(u, v) = 1 - H_{lp}(u, v)$

  Why? how to prove it?

Sharpening Frequency Domain Filters

- Ideal highpass filter

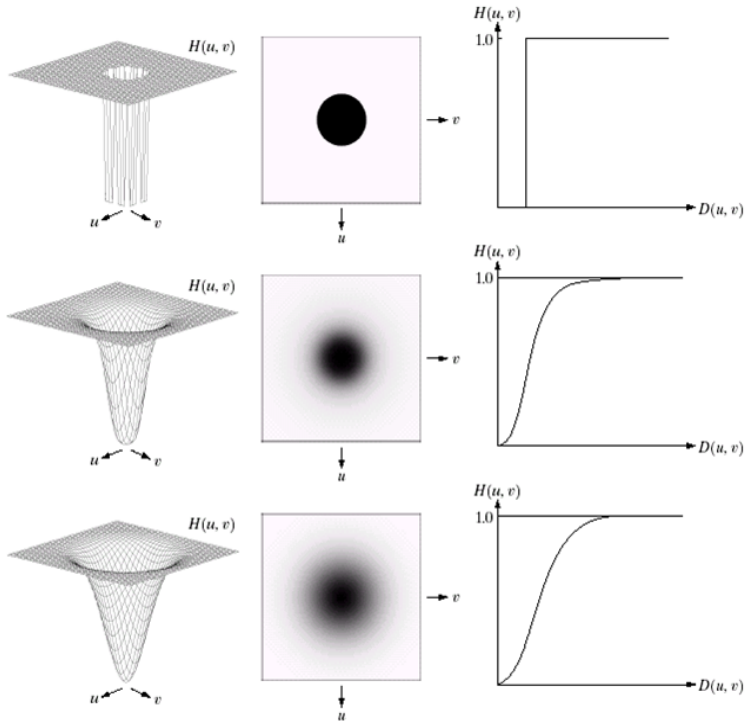
$$H(u,v) = \begin{cases} 1 & if\ D(u,v) \leq D_0 \\ 0 & if\ D(u,v) > D_0 \end{cases}$$

- Butterworth highpass filter

$$H(u,v) = \frac{1}{1+[D_0/D(u,v)]^{2n}}$$

- Gaussian highpass filter

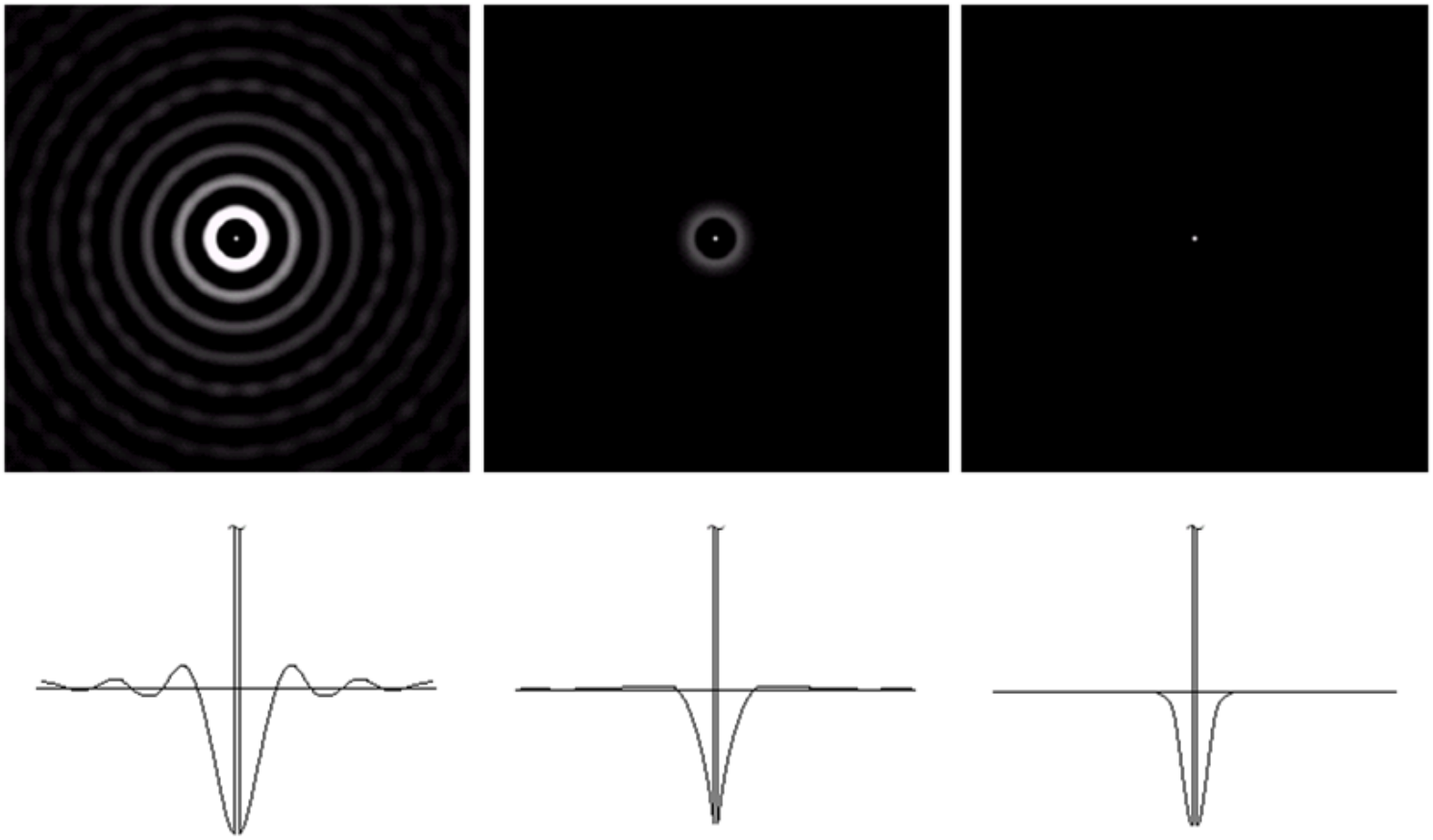
$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$



a b c
d e f
g h i

FIGURE 4.22 Top row: Perspective plot, image representation, and cross section of a typical ideal highpass filter. Middle and bottom rows: The same sequence for typical Butterworth and Gaussian highpass filters.

Highpass Filters Spatial Representations



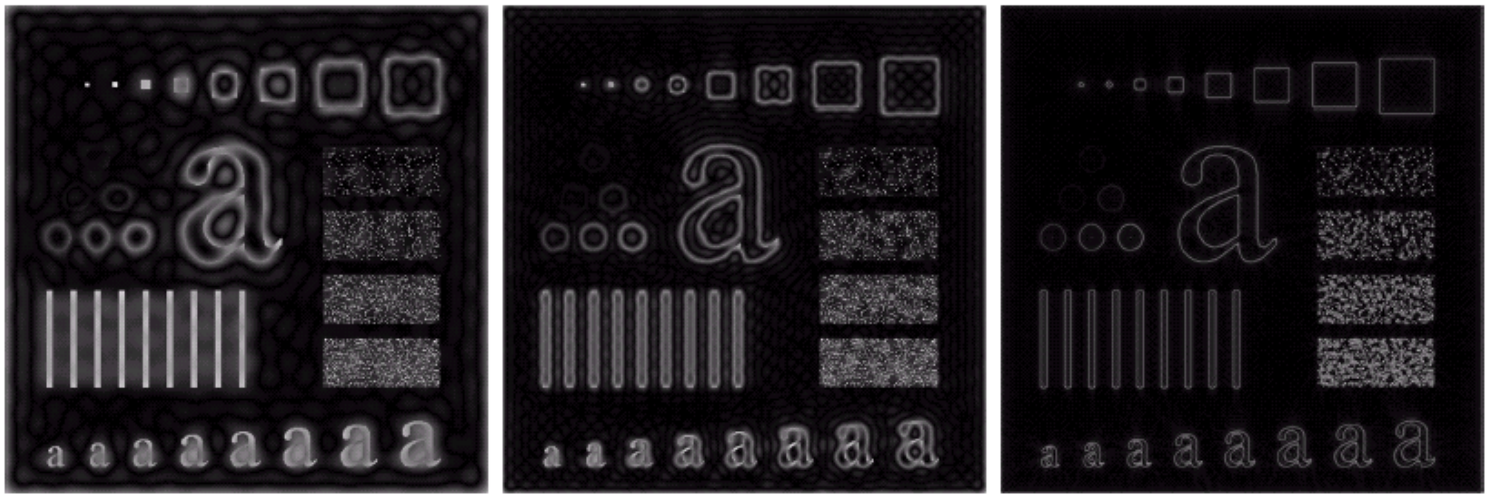
a b c

FIGURE 4.23 Spatial representations of typical (a) ideal, (b) Butterworth, and (c) Gaussian frequency domain highpass filters, and corresponding gray-level profiles.

Ideal Highpass Filters

- Ideal highpass filter

$$H(u,v) = \begin{cases} 0 & if\ D(u,v) \leq D_0 \\ 1 & if\ D(u,v) > D_0 \end{cases}$$



a b c

FIGURE 4.24 Results of ideal highpass filtering the image in Fig. 4.11(a) with $D_0 = 15$, 30, and 80, respectively. Problems with ringing are quite evident in (a) and (b).

Butterworth Highpass Filters

- Butterworth highpass filter

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}}$$

a b c

FIGURE 4.25 Results of highpass filtering the image in Fig. 4.11(a) using a BHPF of order 2 with $D_0 = 15$, 30, and 80, respectively. These results are much smoother than those obtained with an ILPF.

Gaussian Highpass Filters

- Gaussian highpass filter

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

a b c

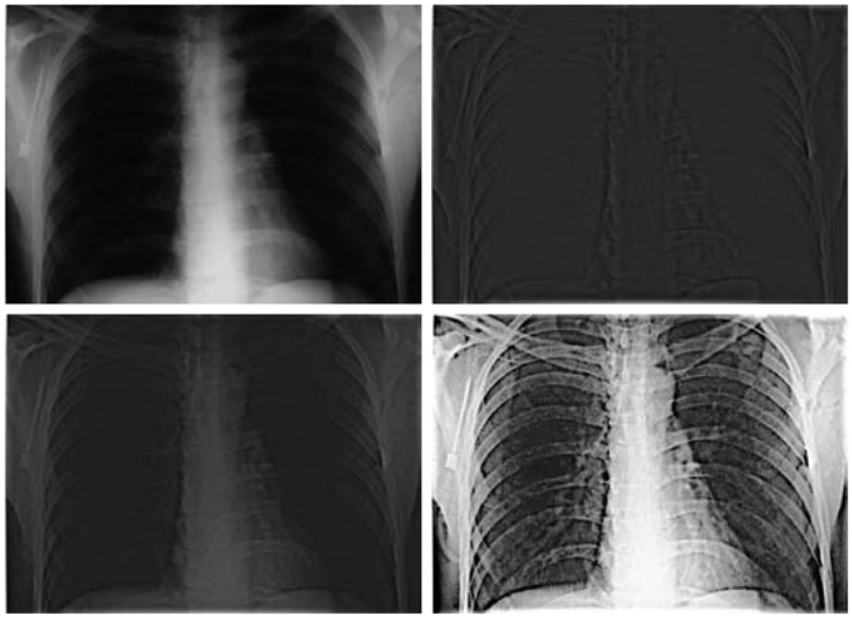
FIGURE 4.26 Results of highpass filtering the image of Fig. 4.11(a) using a GHPF of order 2 with $D_0 = 15$, 30, and 80, respectively. Compare with Figs. 4.24 and 4.25.

Sharpening Frequency Domain Filters

- High-Frequency Emphasis Filtering

$$H_{hfe}(u,v) = a + bH_{hp}(u,v)$$

```
>>f=imread('Fig0419(a)(chestXray_original).tif');
>>PQ=paddedsize(size(f));
>>D0=0.05*PQ(1);
>>HBW=hpfilter('btw',PQ(1),PQ(2),D0,2);
>>H=0.5+2*HBW;
>>gbf=dftfilt(f,H);
>>ghf=gscale(gbf);
>>ghe=histeq(ghf,256);
>>imshow(ghe);
```



a b
c d

FIGURE 4.19 High-frequency emphasis filtering. (a) Original image. (b) Highpass filtering result. (c) High-frequency emphasis result. (d) Image (c) after histogram equalization. (Original image courtesy of Dr. Thomas R. Gest, Division of Anatomical Sciences, University of Michigan Medical School.)

Image Processing and Analysis
Lecture 8,9、Image Restoration

Weiqiang Wang

School of Computer and Control Engineering and Engineering, UCAS

November 17, 2015

Outline

Image Degradation and Restoration

- The objective of restoration

  To improve a given image in some predefined sense.

- The difference between restoration and enhancement

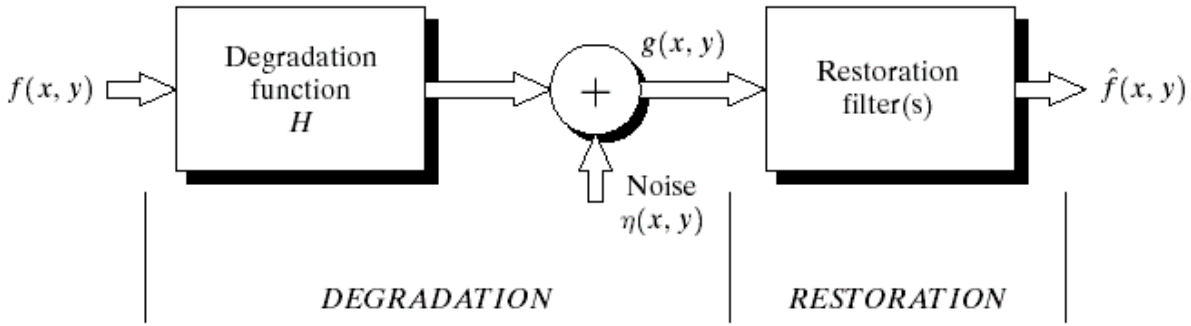


FIGURE 5.1 A model of the image degradation/restoration process.

$$g(x,y) = H[f(x,y)] + \eta(x,y)$$
$$g(x,y) = h(x,y) * f(x,y) + \eta(x,y)$$
$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

Noise Models

- Adding Noise with Function imnoise
- Generating Spatial Random Noise with a Specified Distribution
- Periodic Noise
- Estimating Noise Parameters

Adding Noise with Function imnoise

- g = imnoise(f, type, parameters)

  f: input image, type and parameters will explain later.

  [note]: converts the input image to class double in the range [0,1] before adding noise to it.

  For Example

  - G=imnoise(f,'gussian',m,var)
  - G=imnoise(f,'localvar',V)
  - G=imnoise(f,'localvar',image_intensity,var)
  - G=imnoise(f,'salt&pepper',d))
  - G=imnoise(f,'speckle', var)
  - G=imnoise(f,'poisson')

Noise Models

- Adding Noise with Function imnoise
- Generating Spatial Random Noise with a Specified Distribution
- Periodic Noise
- Estimating Noise Parameters

Generating Random Noise with a Specified Distribution

- A famous result:if w is a uniformly distributed random varible in the interval (0,1), then we can obtained a random variable z with a specified CDF, F_z, by solving the equation

$$z = F_z^{-1}(w)$$

- An Example: generate random numbers, z, *with a Rayleigh CDF*

$$F_z(z) = \begin{cases} 1 - e^{-(z-a)^2/b} & if \ z \geq a \\ 0 & if \ z < a \end{cases}$$

We can obtain z
by solving the following equation

$$1 - e^{-(z-a)^2/b} = w$$

get

$$z = a + \sqrt{-b \ln(1-w)}$$

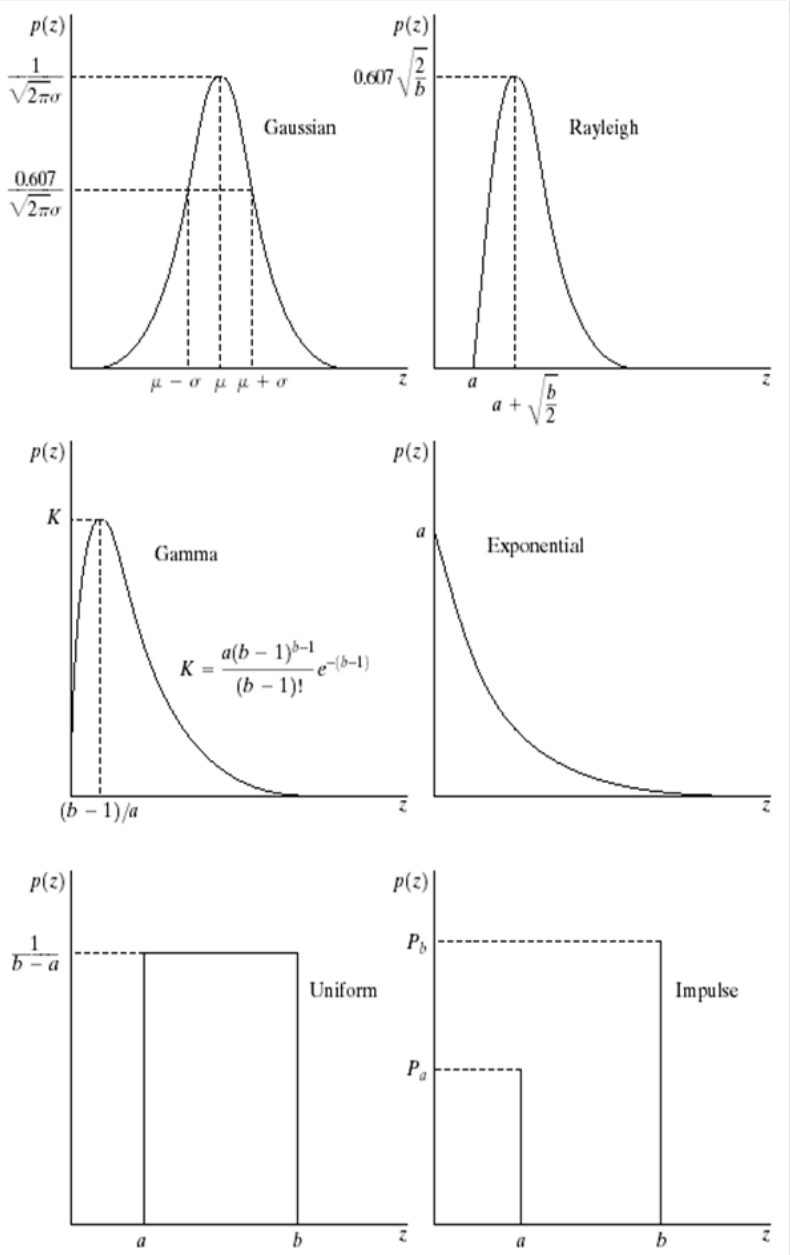
Generating Random Noise with a Specified Distribution

TABLE 5.1 Generation of random variables.

Name	PDF	Mean and Variance	CDF	Generator†
Uniform	$p_z(z) = \begin{cases} \frac{1}{b-a} & \text{if } a \le z \le b \\ 0 & \text{otherwise} \end{cases}$	$m = \frac{a+b}{2}, \quad \sigma^2 = \frac{(b-a)^2}{12}$	$F_z(z) = \begin{cases} 0 & z < a \\ \frac{z-a}{b-a} & a \le z \le b \\ 1 & z > b \end{cases}$	MATLAB function `rand`
Gaussian	$p_z(z) = \frac{1}{\sqrt{2\pi}b} e^{-(z-a)^2/2b^2}$ $-\infty < z < \infty$	$m = a, \quad \sigma^2 = b^2$	$F_z(z) = \int_{-\infty}^{z} p_z(v)\, dv$	MATLAB function `randn`
Salt & Pepper ??	$p_z(z) = \begin{cases} P_a & \text{for } z = a \\ P_b & \text{for } z = b \\ 0 & \text{otherwise} \end{cases}$ $b > a$	$m = aP_a + bP_b$ $\sigma^2 = (a-m)^2 P_a + (b-m)^2 P_b$	$F_z(z) = \begin{cases} 0 & \text{for } z < a \\ P_a & \text{for } a \le z < b \\ P_a + P_b & \text{for } b \le z \end{cases}$	MATLAB function `rand` with some additional logic
Lognormal	$p_z(z) = \frac{1}{\sqrt{2\pi}bz} e^{-[\ln(z)-a]^2/2b^2}$?? $z \ge 0$	$m = e^{a+(b^2/2)}, \sigma^2 = [e^{b^2} - 1]e^{2a+b^2}$	$F_z(z) = \int_0^z p_z(v)\, dv$	$z = ae^{bN(0,1)}$??
Rayleigh	$p_z(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$m = a + \sqrt{\pi b/4}, \sigma^2 = \frac{b(4-\pi)}{4}$	$F_z(z) = \begin{cases} 1 - e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$z = a + \sqrt{b \ln[1 - U(0,1)]}$
Exponential	$p_z(z) = \begin{cases} ae^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$m = \frac{1}{a}, \quad \sigma^2 = \frac{1}{a^2}$	$F_z(z) = \begin{cases} 1 - e^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$z = -\frac{1}{a}\ln[1 - U(0,1)]$
Erlang	$p_z(z) = \frac{a^b z^{b-1}}{(b-1)!} e^{-az}$ $z \ge 0$	$m = \frac{b}{a}, \quad \sigma^2 = \frac{b}{a^2}$	$F_z(z) = \left[1 - e^{-az} \sum_{n=0}^{b-1} \frac{(az)^n}{n!}\right]$ $z \ge 0$	$z = E_1 + E_2 + \cdots + E_b$ (The E's are exponential random numbers with parameter a.)

† $N(0,1)$ denotes normal (Gaussian) random numbers with mean 0 and a variance of 1. $U(0,1)$ denotes uniform random numbers in the range $(0,1)$.

Generating Random Noise with a Specified Distribution

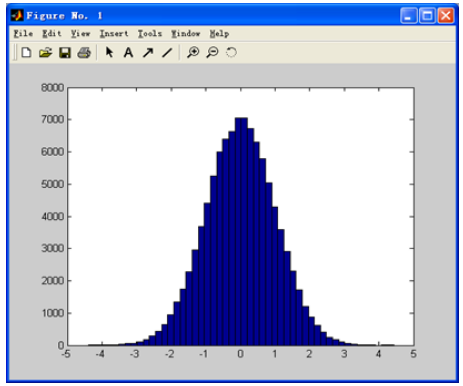


a b
c d
e f

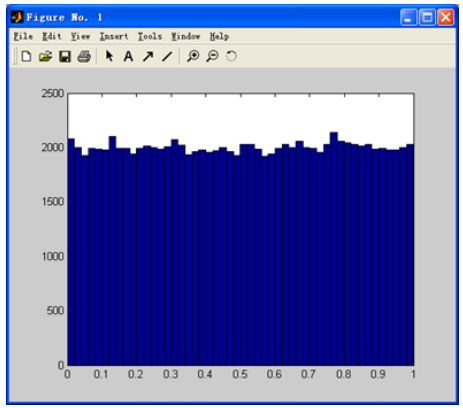
FIGURE 5.2 Some important probability density functions.

Generating Random Noise with a Specified Distribution

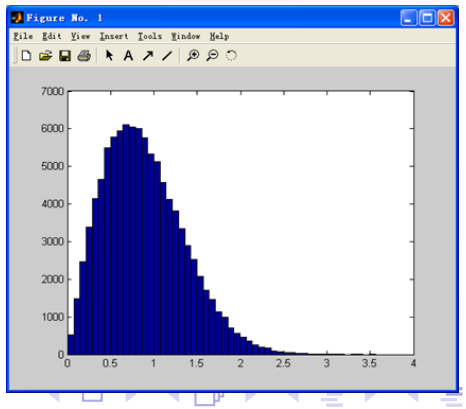
```
r=imnoise2('gaussian',100000,1,0,1);
hist(r,50);
```



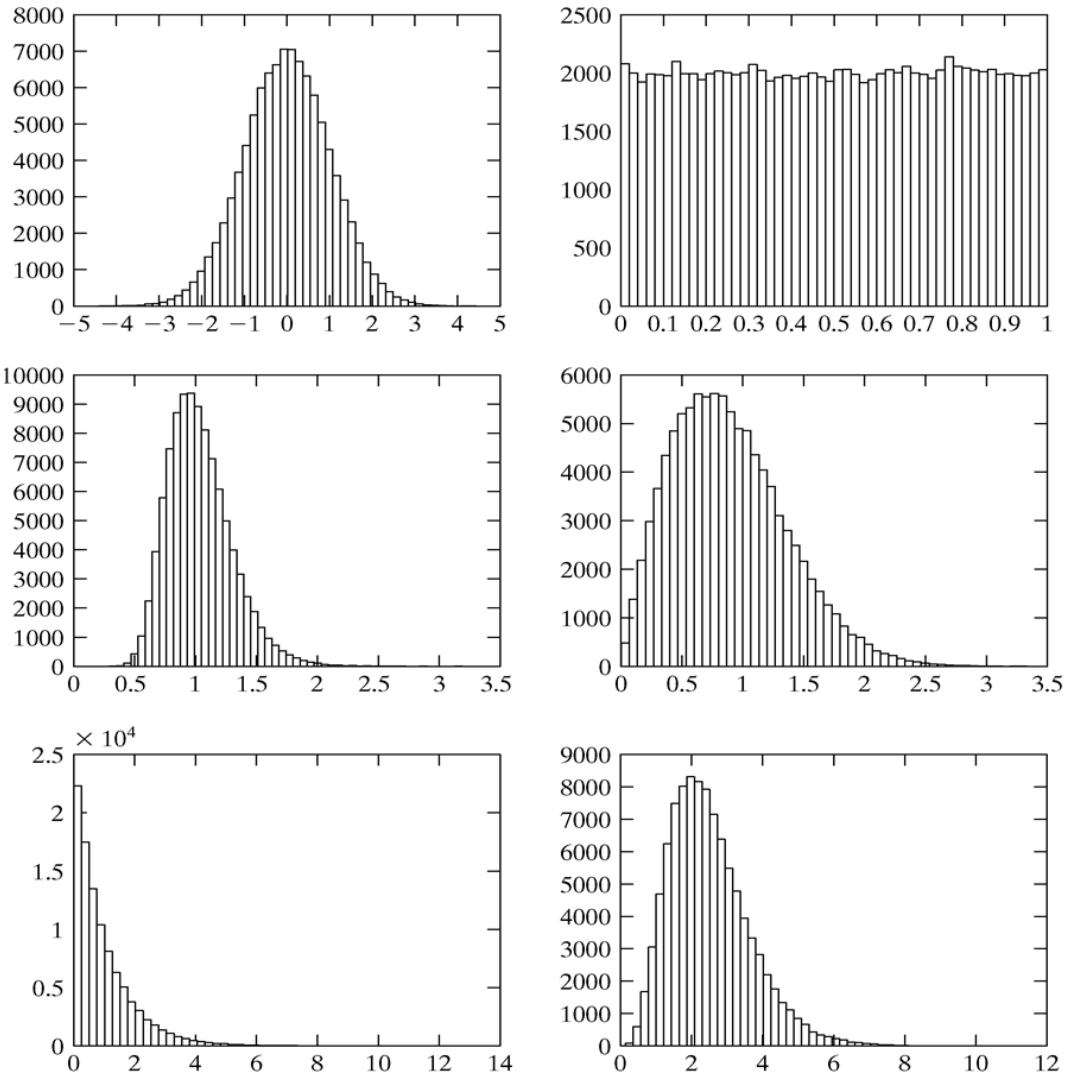

```
ru=imnoise2('uniform',100000,1,0,1);
hist(ru,50);
```




```
rr=imnoise2('rayleigh',100000,1,0,1);
hist(rr,50);
```



Generating Random Noise with a Specified Distribution



a b
c d
e f

FIGURE 5.2
Histograms of random numbers: (a) Gaussian, (b) uniform, (c) lognormal, (d) Rayleigh, (e) exponential, and (f) Erlang. In each case the default parameters listed in the explanation of function `imnoise2` were used.

Generating Random Noise with a Specified Distribution

- Adding Noise with Function imnoise
- Generating Spatial Random Noise with a Specified Distribution
- Periodic Noise
- Estimating Noise Parameters

Periodic Noise

- Periodic Noise Signal

$$r(x,y) = A\sin(2\pi u_0(x+B_x)/M + 2\pi v_0(y+B_y)/N)$$

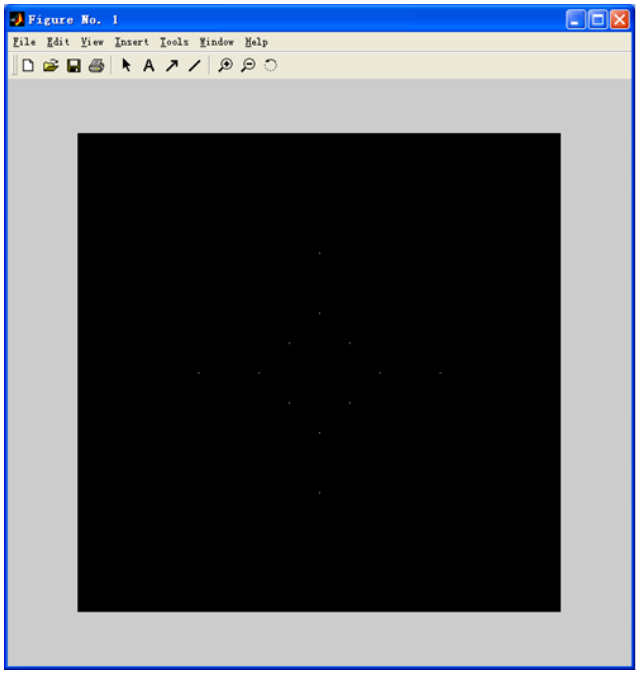
- Its FT

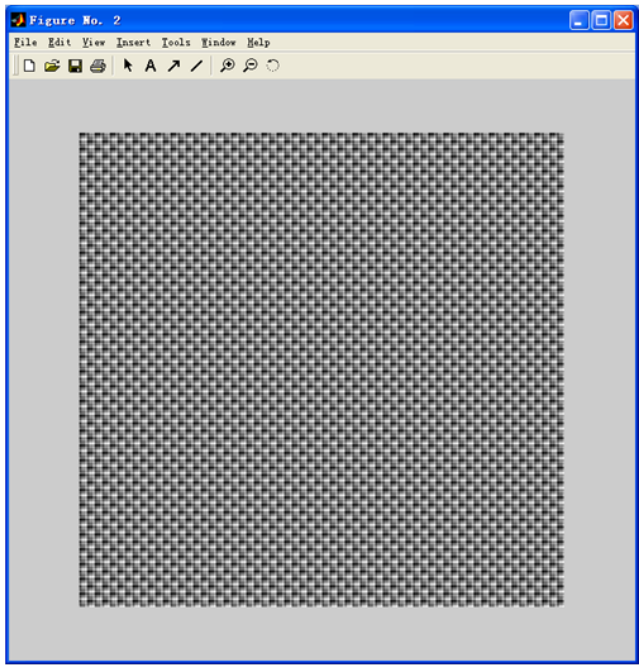
$$R(u,v) = j\frac{A}{2}[\exp(j2\pi u_0 B_x/M)\delta(u+u_0, v+v_0)$$
$$-\exp(j2\pi v_0 B_y/N)\delta(u-u_0, v-v_0)]$$

??

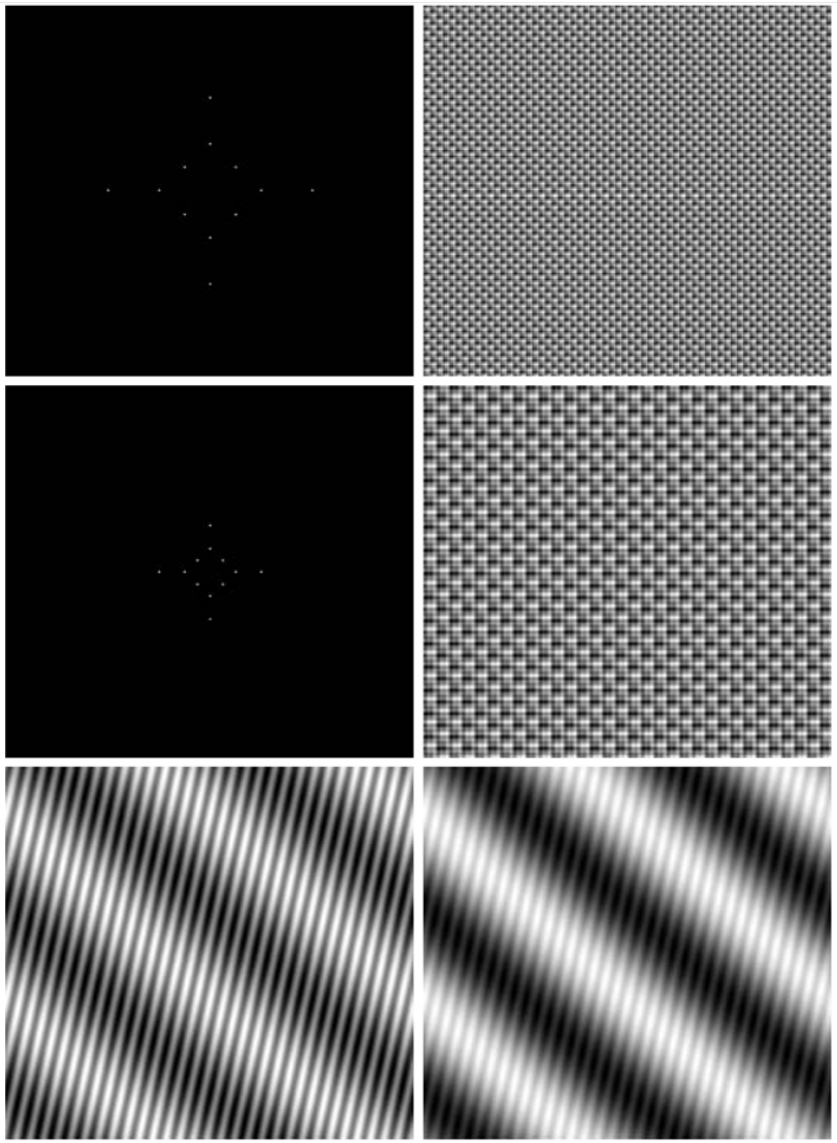
Periodic Noise

- C=[0 64;0 128;32 32;64 0;128 0;-32 32];
- [r,R,S]=imnoise3(512,512,C);
- imshow(S,[]);
- figure,imshow(r,[]);





Periodic Noise



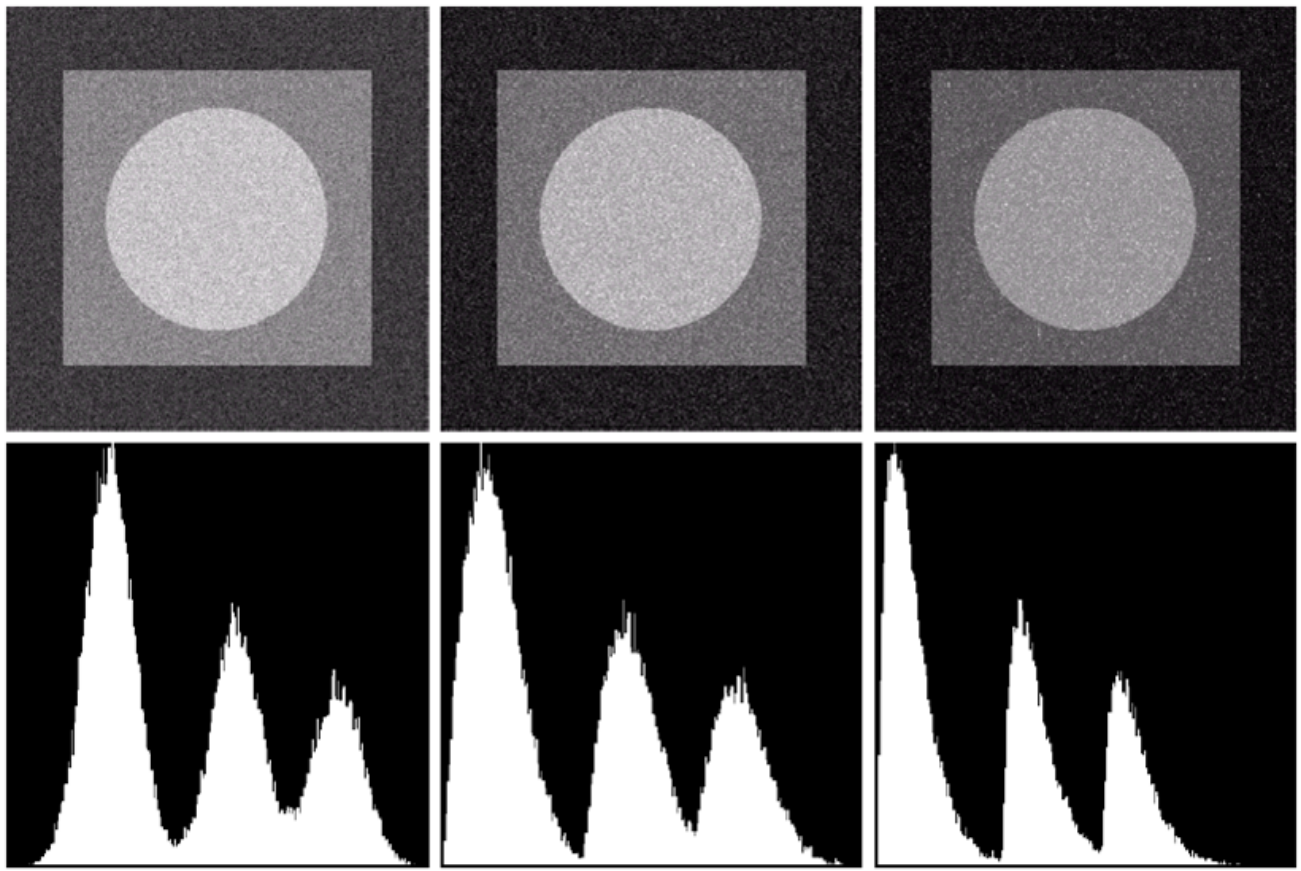
a b
c d
e f

FIGURE 5.3
(a) Spectrum of specified impulses. (b) Corresponding sine noise pattern. (c) and (d) A similar sequence. (e) and (f) Two other noise patterns. The dots in (a) and (c) were enlarged to make them easier to see.

Noise Models

- Adding Noise with Function imnoise
- Generating Spatial Random Noise with a Specified Distribution
- Periodic Noise
- Estimating Noise Parameters

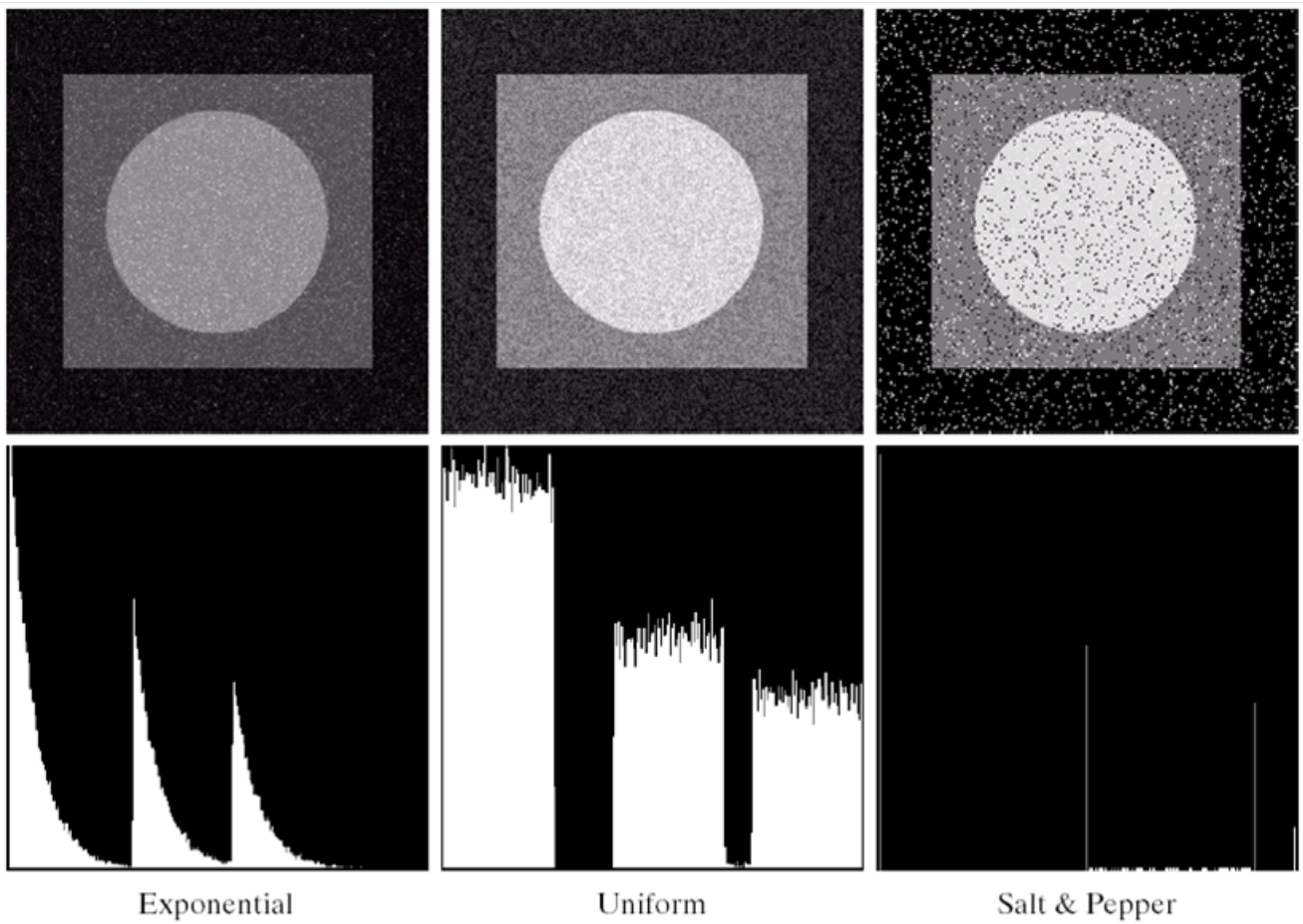
Example: Images corrupted by Independent Noises



a b c
d e f

FIGURE 5.4 Images and histograms resulting from adding Gaussian, Rayleigh, and gamma noise to the image in Fig. 5.3.

Example: Images corrupted by Independent Noises



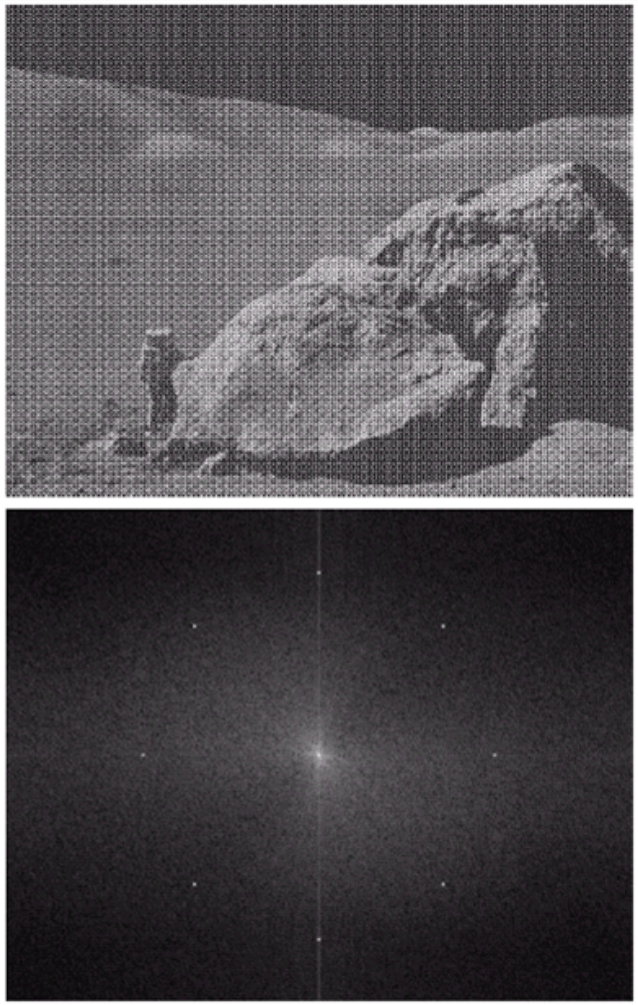
g h i
j k l

FIGURE 5.4 *(Continued)* Images and histograms resulting from adding exponential, uniform, and impulse noise to the image in Fig. 5.3.

Example: Image corrupted by Periodic Noise

a
b

FIGURE 5.5
(a) Image corrupted by sinusoidal noise. (b) Spectrum (each pair of conjugate impulses corresponds to one sine wave). (Original image courtesy of NASA.)



Estimating Noise Parameters

- The parameters of periodic noise typically are estimated by inspection of Fourier spectrum of the image. The frequency spikes can be detected by visual analysis.
- Another approach is to attempt to infer the periodicity of noise components directly from the image, but this is possible only in simplistic cases.
- Automated analysis is possible in situations in which the noise spikes are either exceptionally pronounced, or when some knowledge is available about the general location of the frequency components of the interference.
- The parameters of noise PDFs may be known partially from sensor specifications, but it is often necessary to estimate them for a particular imaging arrangement, e.g, a set of "flat" environments.
- When only images generated are available, it is frequently possible to estimate the parameters of the PDF from small patches of reasonably constant gray level.

Estimating Noise Parameters

- The simple way to use the data from the image strips is for calculating the mean and variance of the gray levels.
- The shape of the histogram identifies the closest PDF match. If the shape is approximately Gaussian, the mean and variance is all we need. For other shapes discussed in section 5.2.2, we use the mean and variance to solve for the parameters a and b.
- $\mu_n = \sum_{i=0}^{L-1}(z_i - m)^n p(z_i)$ where $m = \sum_{i=0}^{L-1} z_i p(z_i)$

when $n = 0$ $\quad \mu_0 = \sum_{i=0}^{L-1}(z_i - m)^0 p(z_i) = 1$

when $n = 1$ $\quad \mu_1 = \sum_{i=0}^{L-1}(z_i - m)p(z_i)$

$$= \sum_{i=0}^{L-1} z_i p(z_i) - m\sum_{i=0}^{L-1} p(z_i) = 0$$

when $n = 2$ $\quad \mu_2 = \sum_{i=0}^{L-1}(z_i - m)^2 p(z_i) = Var(z)$

Estimating Noise Parameters

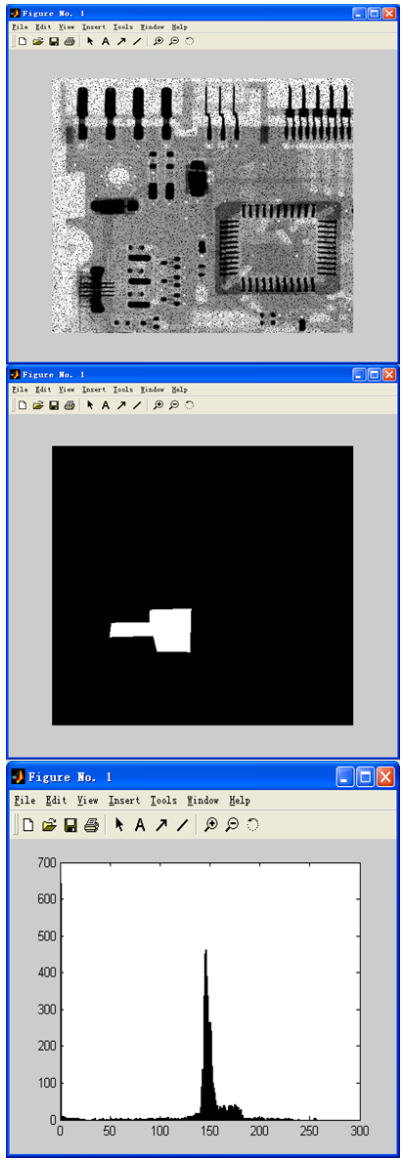
- Mean and central moments

  [v,unv]=statmoments(p,n)
- Select region of interest

  B=roipoly(f,c,r)
- f=imread('Fig0505(a)(ckt_pepper_only).tif');
- imshow(f)

- [B,c,r]=roipoly(f);
- Mouse select
- imshow(B)

- [p,npix]=histroi(f,c,r);
- bar(p,1)



Restoration in the Presence of Noise Only–Spatial Filtering

- If there is only noise, then

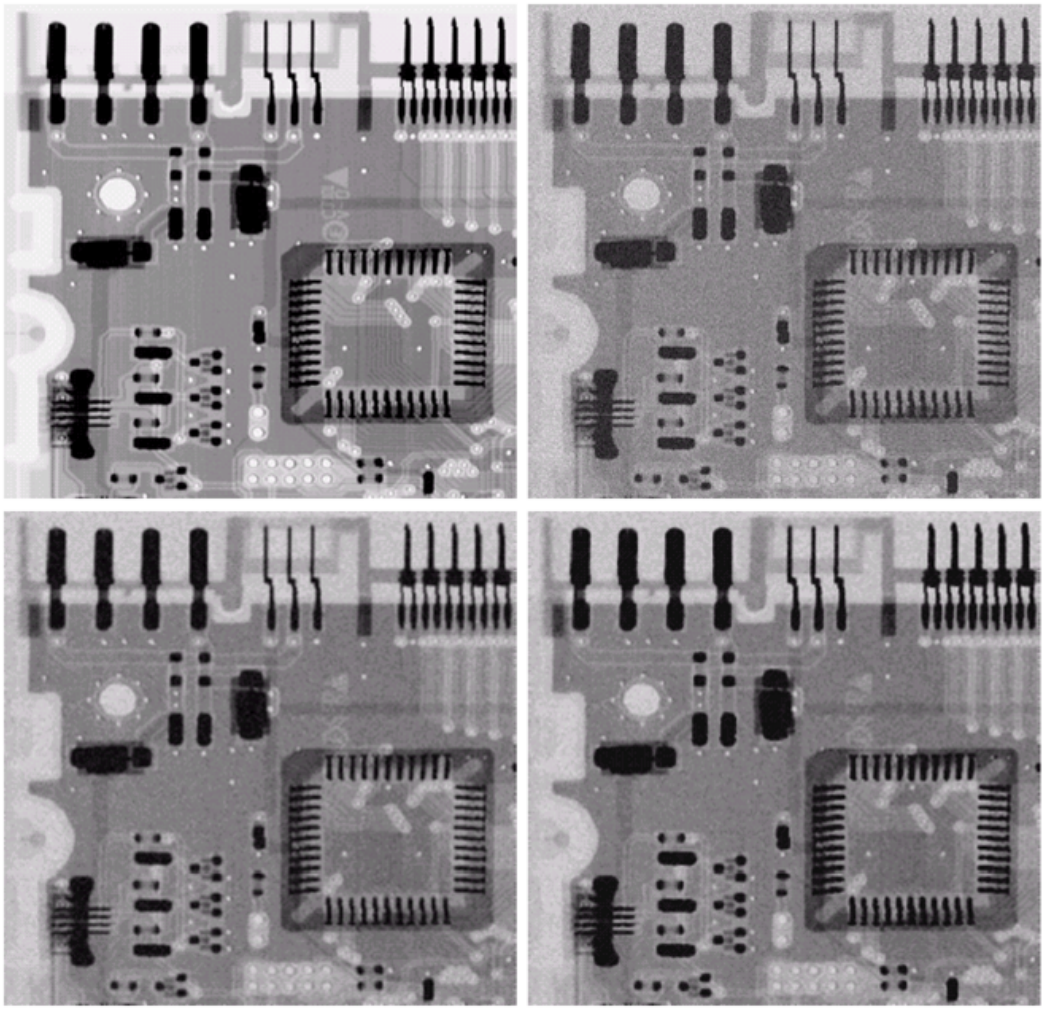
$$g(x,y) = f(x,y) + \eta(x,y)$$
$$G(u,v) = F(u,v) + N(u,v)$$

- Spatial Noise Filters
- Adaptive Spatial Filters

Mean Filters

- Arithmetic mean filter
  $\widehat{f}(x,y) = \frac{1}{mn}\sum_{(s,t)\in S_{xy}} g(s,t)$
- Geometric mean filter
  $\widehat{f}(x,y) = [\prod_{(s,t)\in S_{xy}} g(s,t)]^{\frac{1}{mn}}$
- Harmonic mean filter
  $\widehat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}$
- Contraharmonic mean filter
  $\widehat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$

- The contraharmonic mean filter is well suited for eliminating the effects of salt-and pepper noise. For positive values of Q, the filter eliminates pepper noise. For negative values of Q, it eliminates salt noise. It cannot do both simultaneously.



a b
c d

FIGURE 5.7 (a) X-ray image. (b) Image corrupted by additive Gaussian noise. (c) Result of filtering with an arithmetic mean filter of size 3×3. (d) Result of filtering with a geometric mean filter of the same size. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

Mean Filters(cont.)

- The harmonic mean filter works well for salt noise, but fails for peper noise. It does well for Gaussian noise

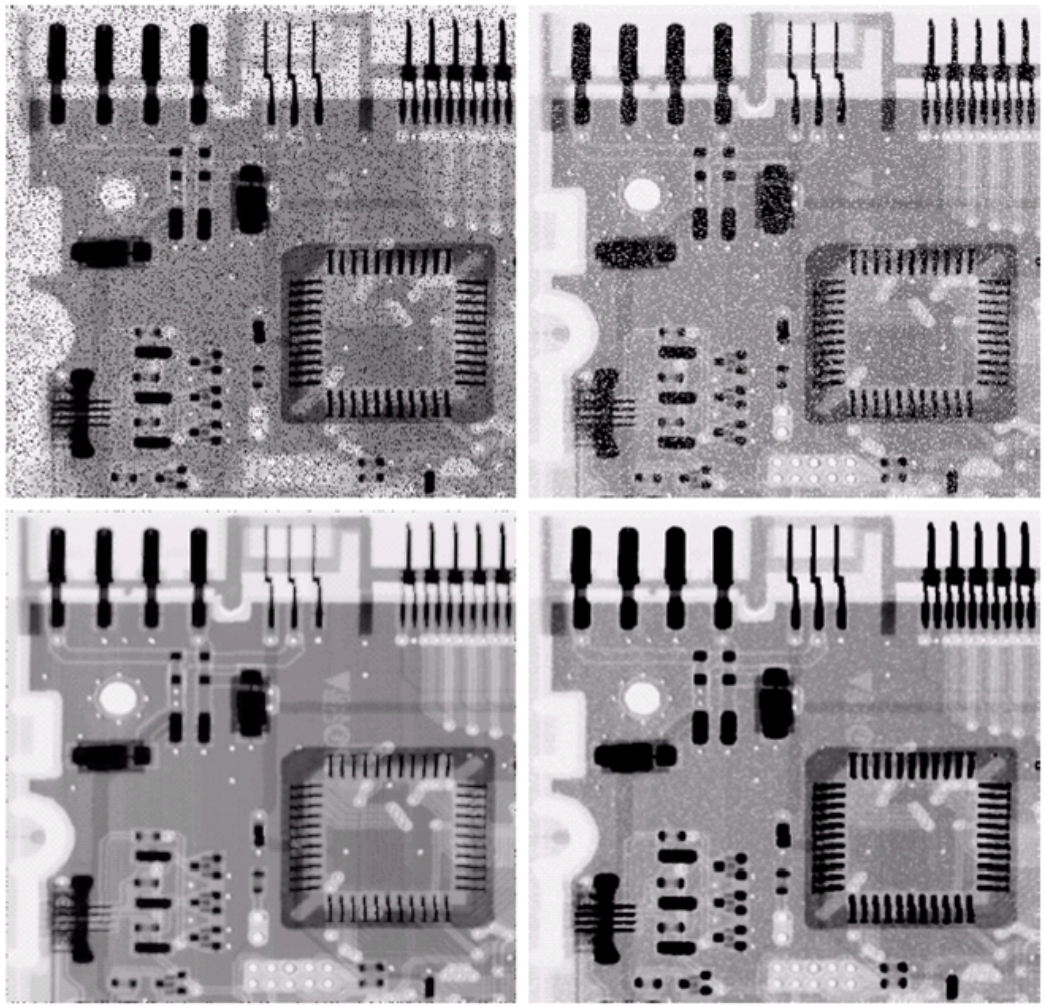
$$\widehat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}$$

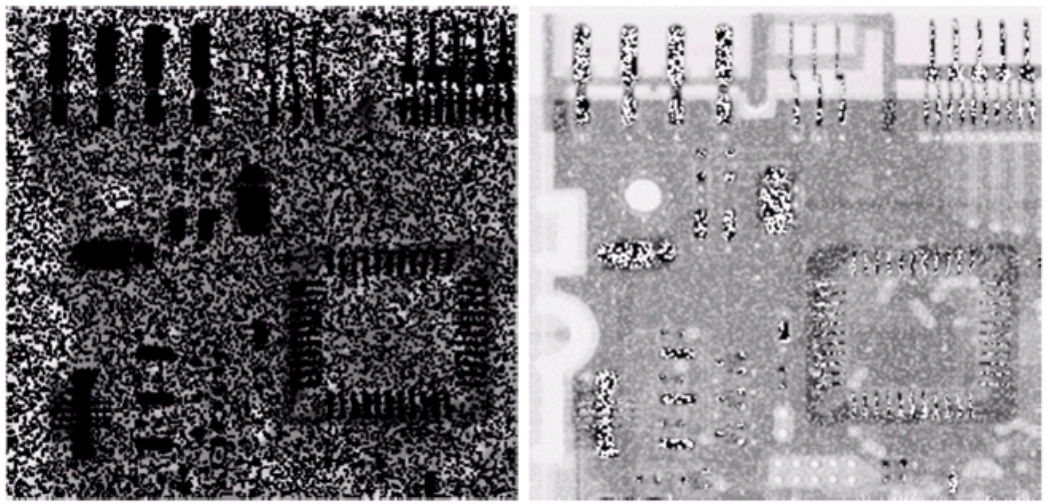
- The contraharmonic mean filter can effectively eliminate salt noise when Q is a negtive value, and works well for pepper noise when Q is a positive value. But it cannot do both simulteneously

$$\widehat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$$

a b
c d

FIGURE 5.8
(a) Image corrupted by pepper noise with a probability of 0.1. (b) Image corrupted by salt noise with the same probability. (c) Result of filtering (a) with a 3×3 contraharmonic filter of order 1.5. (d) Result of filtering (b) with $Q = -1.5$.





a b

FIGURE 5.9 Results of selecting the wrong sign in contraharmonic filtering. (a) Result of filtering Fig. 5.8(a) with a contraharmonic filter of size 3×3 and $Q = -1.5$. (b) Result of filtering 5.8(b) with $Q = 1.5$.

Order-statistics Filters

- Median filter

$$\widehat{f}(x,y) = \underset{(s,t)\in S_{xy}}{median}\ g(s,t)$$

- Midpoint filter

$$\widehat{f}(x,y) = \frac{1}{2}\left[\underset{(s,t)\in S_{xy}}{max}\ g(s,t) + \underset{(s,t)\in S_{xy}}{min}\ g(s,t)\right]$$

- Max and min filter

$$\widehat{f}(x,y) = \underset{(s,t)\in S_{xy}}{max}\ g(s,t)$$

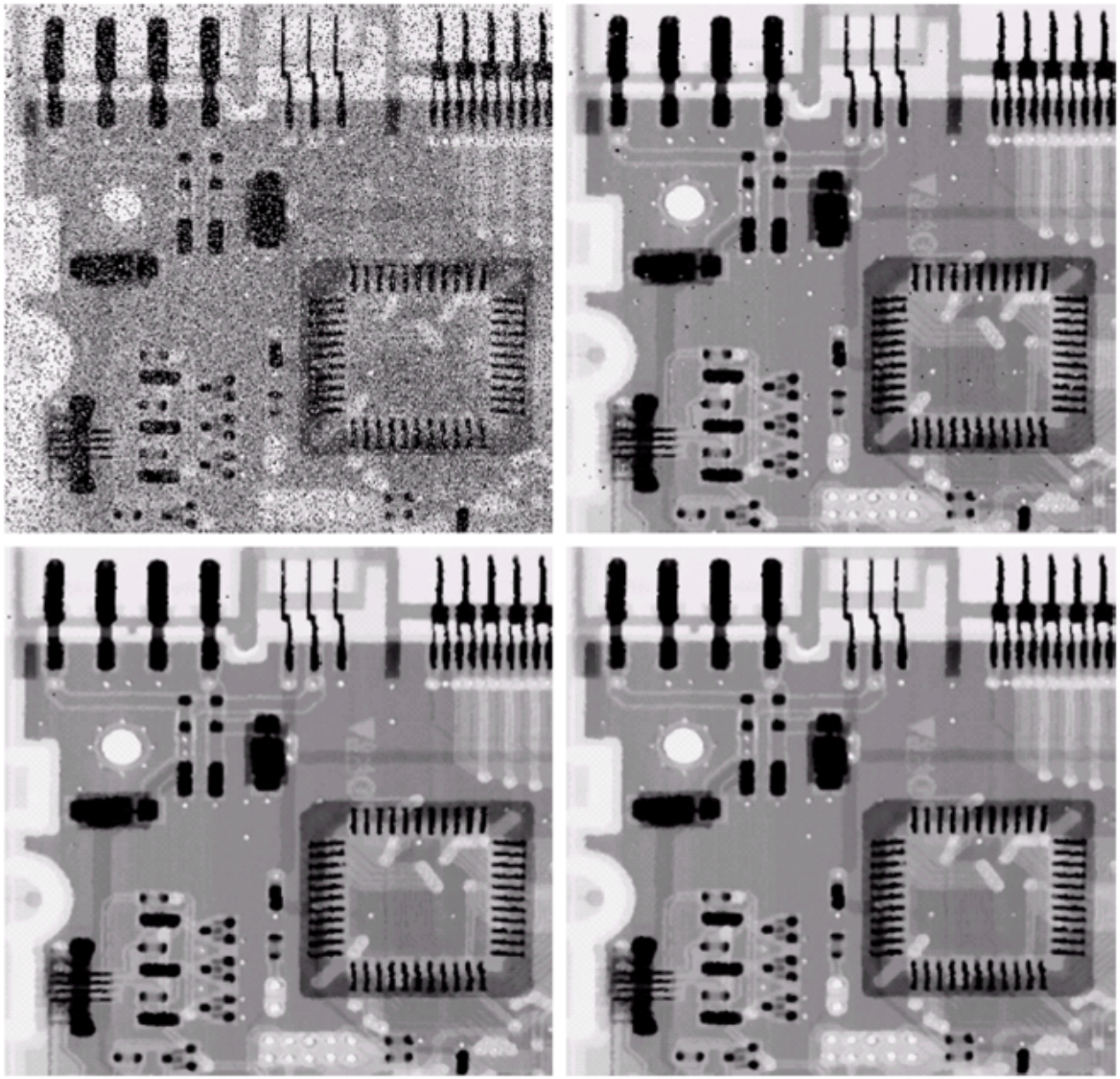
$$\widehat{f}(x,y) = \underset{(s,t)\in S_{xy}}{min}\ g(s,t)$$

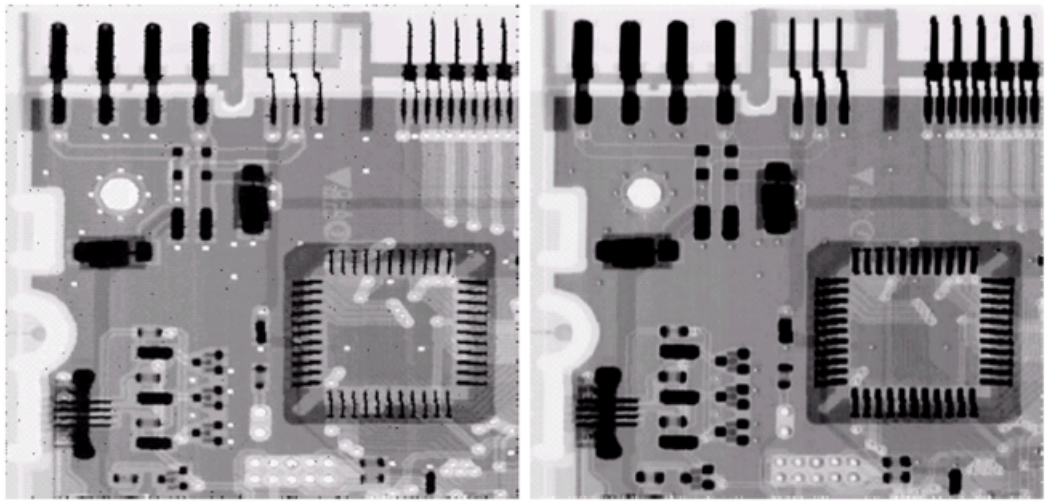
- Alpha-trimmed mean filter

$$\widehat{f}(x,y) = \frac{1}{mn-d}\sum_{(s,t)\in S_{xy}} g_r(s,t)$$

a b
c d

FIGURE 5.10
(a) Image corrupted by salt-and-pepper noise with probabilities $P_a = P_b = 0.1$.
(b) Result of one pass with a median filter of size 3×3.
(c) Result of processing (b) with this filter.
(d) Result of processing (c) with the same filter.





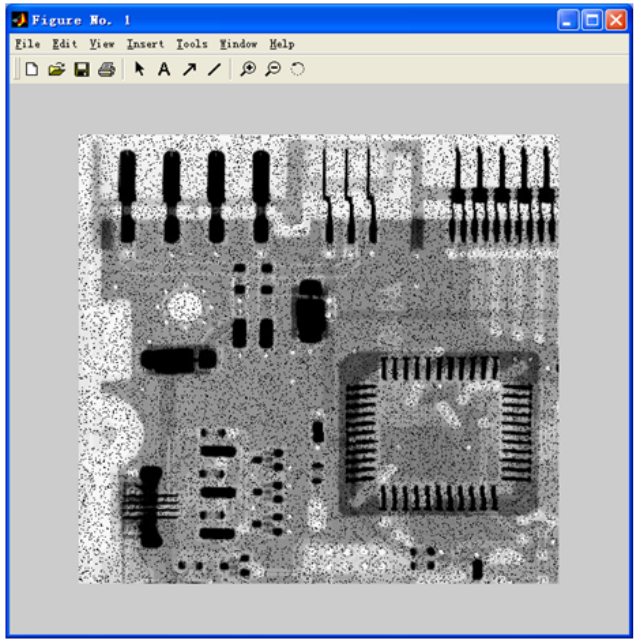
a b

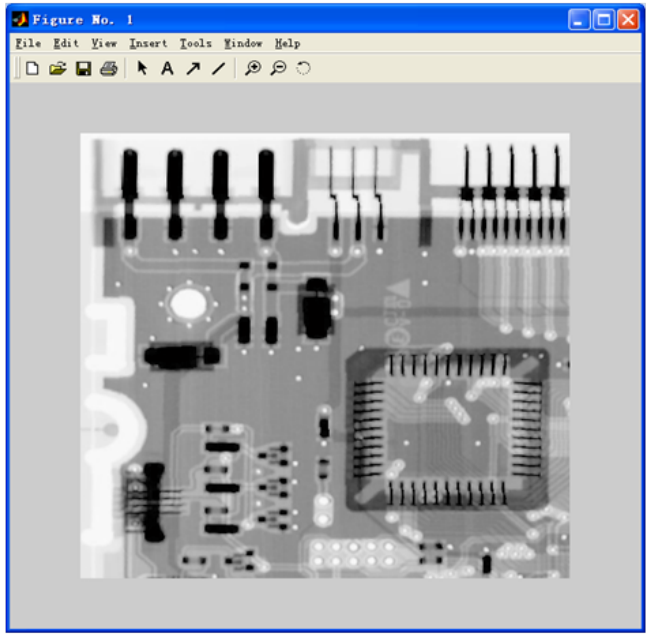
FIGURE 5.11
(a) Result of filtering Fig. 5.8(a) with a max filter of size 3×3. (b) Result of filtering 5.8(b) with a min filter of the same size.

Spatial Noise Filters

%Filter pepper noise is to use contraharmonic filter with a positive value

- gp=imread('Fig0505(a)(ckt_pepper_only).tif');
  %corrupted by pepper noise only with propability 0.1
- fp=spfilt(gp,'chmean',3,3,1.5);
- imshow(gp),figure,imshow(fp)

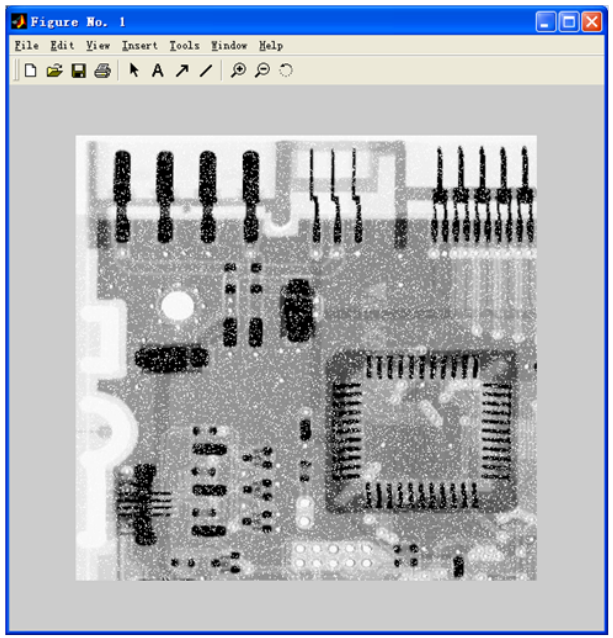


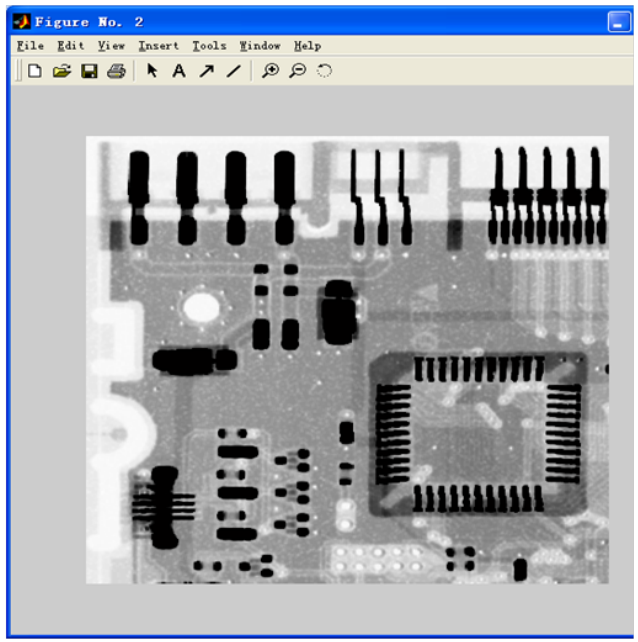


Spatial Noise Filters(cont.)

%Filter salt noise is to use contraharmonic filter with a negative value

- gs=imread('Fig0505(b)(ckt_salt_only).tif');
- fs=spfilt(gs,'chmean',3,3,-1.5);
- imshow(gs),figure,imshow(fs)





Adaptive Spatial Median Filters

- *Notations*:

  Z_{min} = *minimum intensity value in* S_{xy}

  Z_{max} = *maximum intensity value in* S_{xy}

  Z_{med} = *median intensity value in* S_{xy}

  Z_{xy} = *intensity value at coordinates* x, y

- Algorithm (adaptive median algorithm):
  - Level A:

    If $Z_{min} < Z_{med} < Z_{max}$, go to level B

    else increase the window size

    if window size $\leq S_{max}$, repeat level A

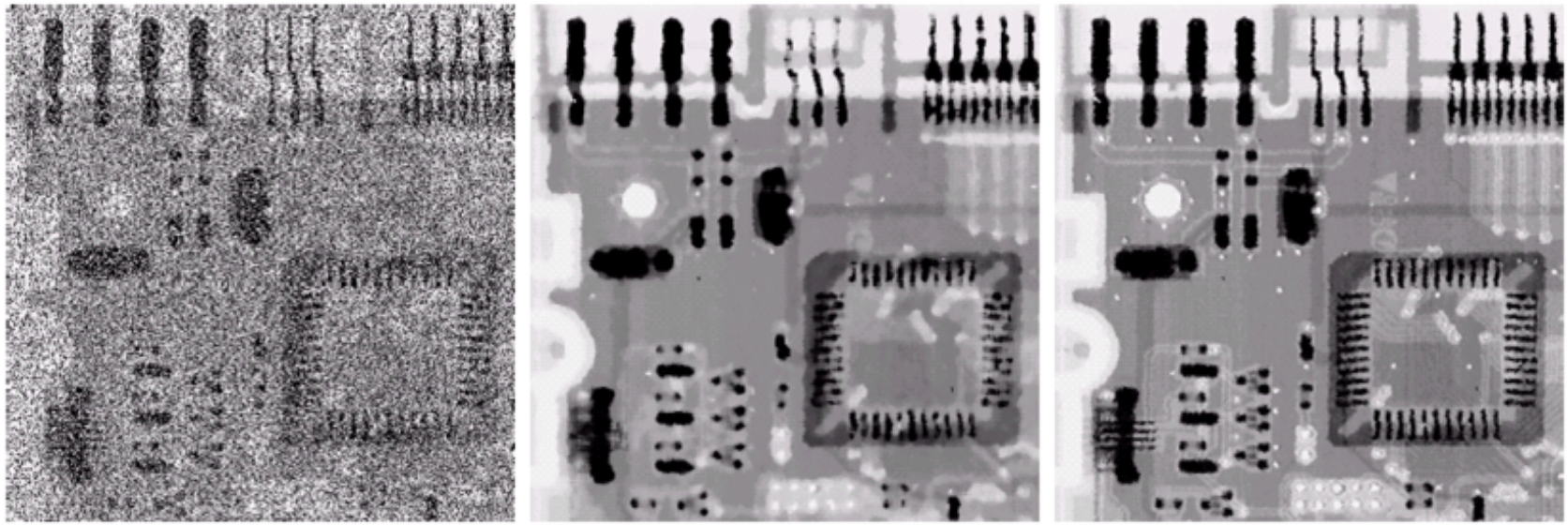
    else output Z_{med}
  - Level B:

    If $Z_{min} < Z_{xy} < Z_{max}$, output Z_{xy}

    else output Z_{med}

Adaptive Spatial Median Filters(cont.)

- f=adpmedian(g,Smax)

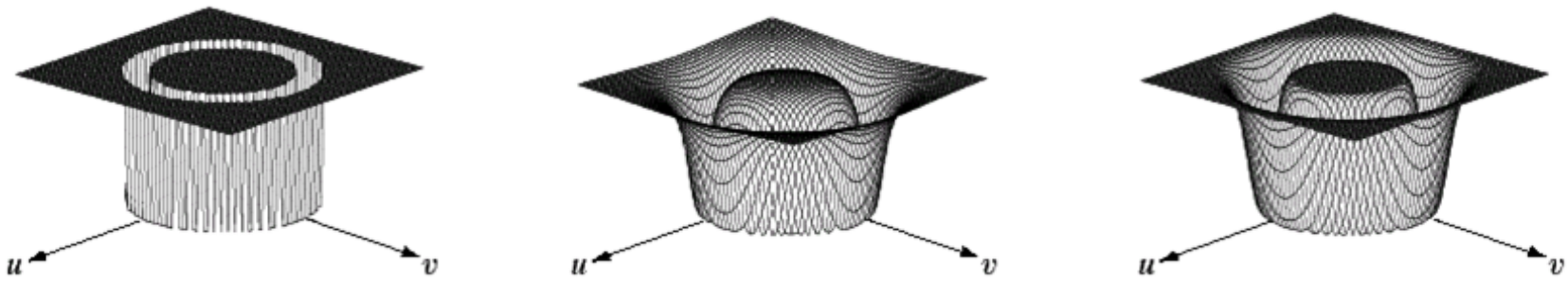


a b c

FIGURE 5.14 (a) Image corrupted by salt-and-pepper noise with probabilities $P_a = P_b = 0.25$. (b) Result of filtering with a 7×7 median filter. (c) Result of adaptive median filtering with $S_{max} = 7$.

Periodic Noise Reduction by Frequency Domain Filtering

- Bandreject Filters



a b c

FIGURE 5.15 From left to right, perspective plots of ideal, Butterworth (of order 1), and Gaussian bandreject filters.

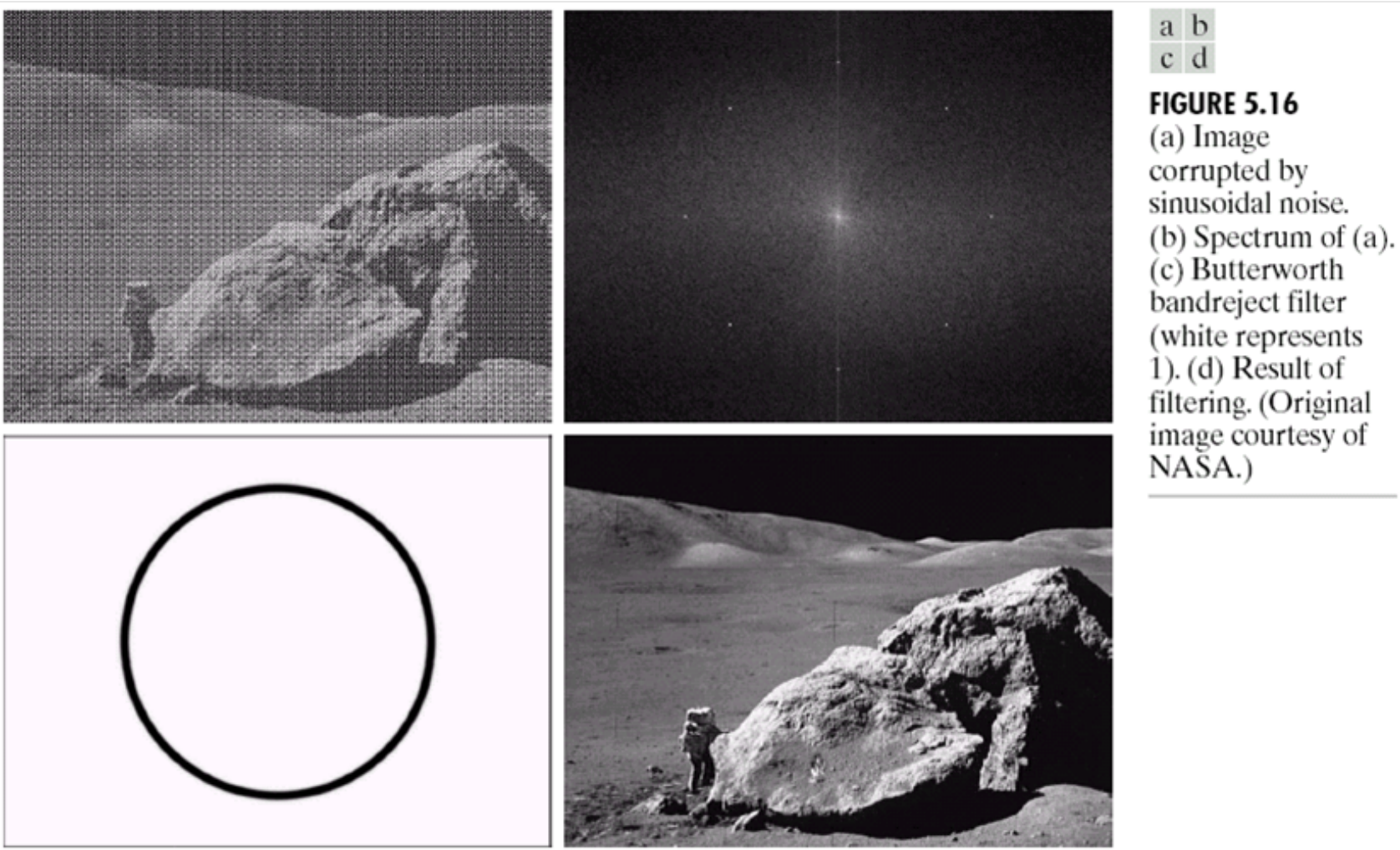
- $H(u,v) =$

$$\begin{cases} 1 & D(u,v) < D_0 - \frac{W}{2} \\ 0 & D_0 - \frac{W}{2} < D(u,v) < D_0 + \frac{W}{2} \\ 1 & D(u,v) > D_0 + \frac{W}{2} \end{cases}$$

- $H(u,v) = \frac{1}{1+[\frac{D(u,v)W}{D^2(u,v)-D_0^2}]^{2n}}$
- $H(u,v) = 1 - exp[-\frac{1}{2}(\frac{D^2(u,v)-D_0^2}{D(u,v)W})^2]$

Periodic Noise Reduction by Frequency Domain Filtering

- Application of Bandreject Filters



a b
c d

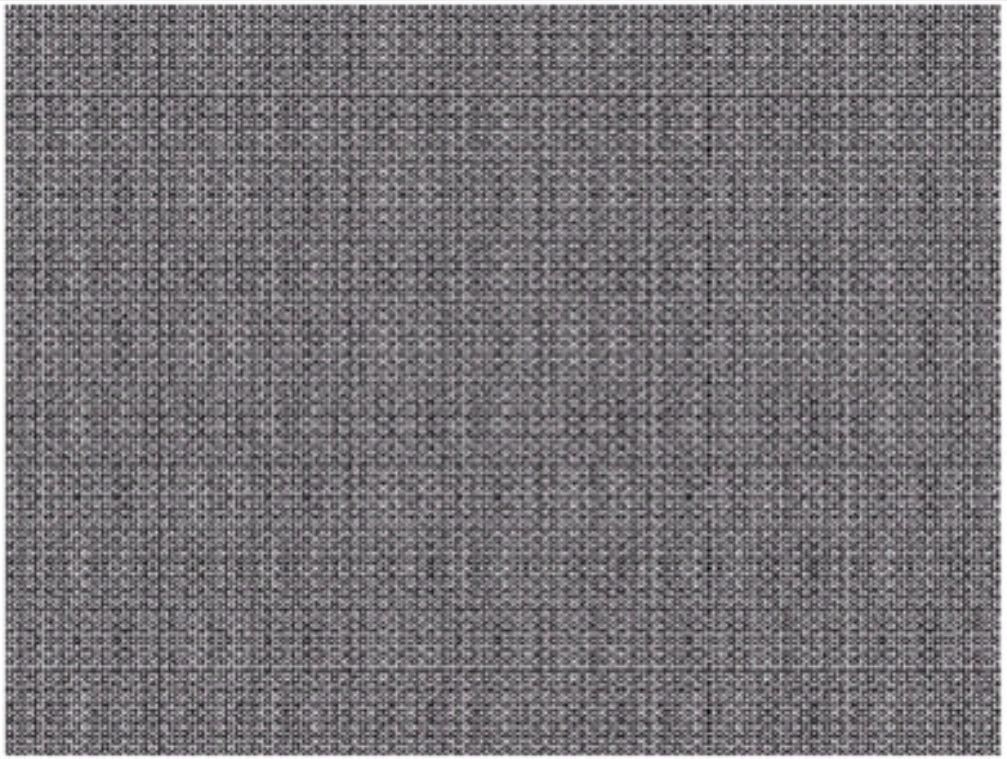
FIGURE 5.16
(a) Image corrupted by sinusoidal noise. (b) Spectrum of (a). (c) Butterworth bandreject filter (white represents 1). (d) Result of filtering. (Original image courtesy of NASA.)

Periodic Noise Reduction by Frequency Domain Filtering

- Bandpass Filters

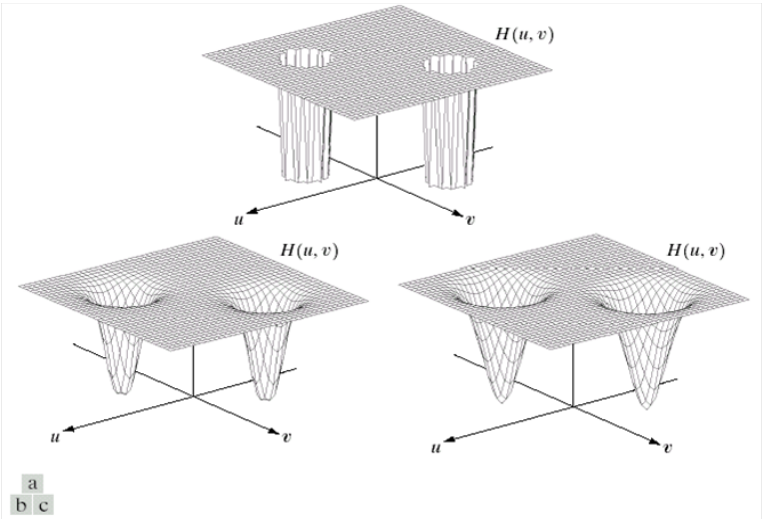
$$H_{bp}(u,v) = 1 - H_{br}(u,v)$$

FIGURE 5.17
Noise pattern of the image in Fig. 5.16(a) obtained by bandpass filtering.



Periodic Noise Reduction by Frequency Domain Filtering(3)

- Notch Filters



a
b c

FIGURE 5.18 Perspective plots of (a) ideal, (b) Butterworth (of order 2), and (c) Gaussian notch (reject) filters.

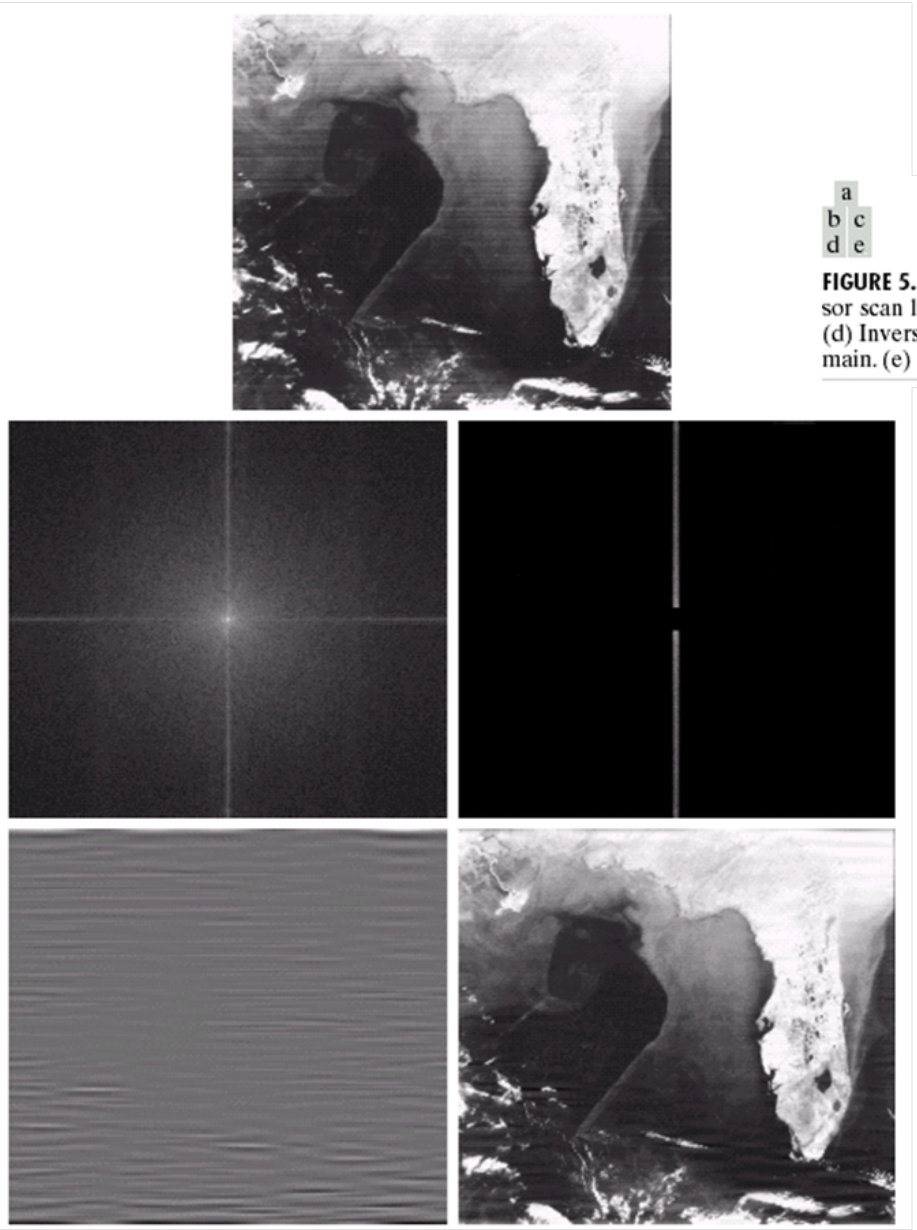
- $H(u,v) = \begin{cases} 0 & D_1(u,v) \le D_0 \ or \ D_2(u,v) \le D_0 \\ 1 & else \end{cases}$
- $H(u,v) = \frac{1}{1+[\frac{D_0^2}{D_1(u,v)D_2(u,v)}]^n}$
- $H(u,v) = 1 - exp[-\frac{1}{2}(\frac{D_1(u,v)D_2(u,v)}{D_0^2})]$

where

$$D_1(u,v) = [(u - \frac{M}{2} - u_0)^2 + (v - \frac{N}{2} - v_0)^2]^{0.5}$$

$$D_2(u,v) = [(u - \frac{M}{2} + u_0)^2 + (v - \frac{N}{2} + v_0)^2]^{0.5}$$

Removal of Periodic Noise by Notch Filters



a
b c
d e

FIGURE 5.19 (a) Satellite image of Florida and the Gulf of Mexico (note horizontal sensor scan lines). (b) Spectrum of (a). (c) Notch pass filter shown superimposed on (b). (d) Inverse Fourier transform of filtered image, showing noise pattern in the spatial domain. (e) Result of notch reject filtering. (Original image courtesy of NOAA.)

Optimum Notch Filtering

- When several interference components are present, the methods mentioned before are not always acceptable since too much image information is removed during the filtering.
- The method discussed here is optimum, in the sense that it minimizes the local variances of the restored estimate $\hat{f}(x,y)$.
- First, obtain an initial estimate of noises by

$$N(u,v) = F_N(u,v)G(u,v)$$
$$\eta(x,y) = \Im^{-1}(F_N(u,v)G(u,v))$$

where $F_N(u,v)$ is constructed to pass only the components associated with the interference pattern.

- Let

$$\hat{f}(x,y) = g(x,y) - w(x,y)\eta(x,y)$$

We will determine the modulation function $w(x,y)$, to minimize the local variances of $\hat{f}(x,y)$.

Optimum Notch Filtering(Cont.)

- Objective: $\min \sigma^2 = \frac{1}{(2a+1)(2b+1)} \sum_{s=-a}^{a} \sum_{t=-b}^{b} [\hat{f}(x+s, y+t) - \bar{f}]^2$

  where $\bar{f} = \frac{1}{(2a+1)(2b+1)} \sum_{s=-a}^{a} \sum_{t=-b}^{b} \hat{f}(x+s, y+t)$

- Further

$$\sigma^2 = \frac{1}{(2a+1)(2b+1)} \sum_{s=-a}^{a} \sum_{t=-b}^{b} ([g(x+s, y+t) - w(x+s, y+t) \cdot \eta(x+s, y+t)] - [\overline{g(x,y)} - \overline{w(x,y)\eta(x,y)}])^2$$

- Let $w(x+s, y+t) = w(x, y)$
- We have

$$\sigma^2 = \frac{1}{(2a+1)(2b+1)} \sum_{s=-a}^{a} \sum_{t=-b}^{b} ([g(x+s, y+t) - w(x, y) \cdot \eta(x+s, y+t)] - [\overline{g(x,y)} - w(x,y)\overline{\eta(x,y)}])^2$$

Optimum Notch Filtering(Cont.)

- To minimize σ^2, we solve:

$$\frac{\partial \sigma^2}{\partial w(x,y)} = 0$$

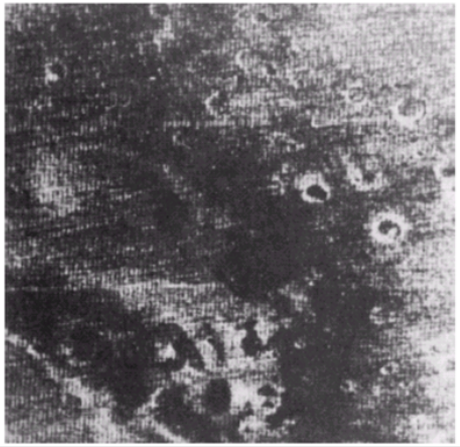
- The result is

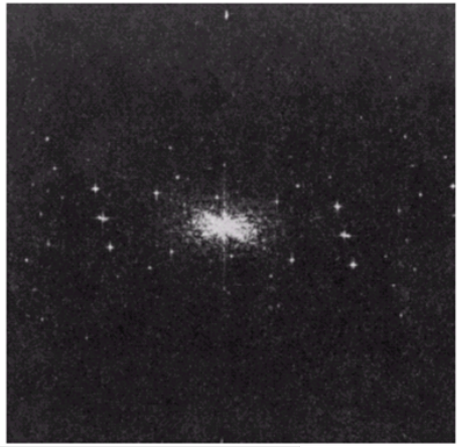
$$w(x,y) = \frac{\overline{g(x,y)\eta(x,y)} - \bar{g}(x,y)\bar{\eta}(x,y)}{\overline{\eta^2(x,y)} - \bar{\eta}^2(x,y)}$$

Optimum Notch Filtering(Cont.)

a b

FIGURE 5.20
(a) Image of the Martian terrain taken by *Mariner 6*.
(b) Fourier spectrum showing periodic interference.
(Courtesy of NASA.)





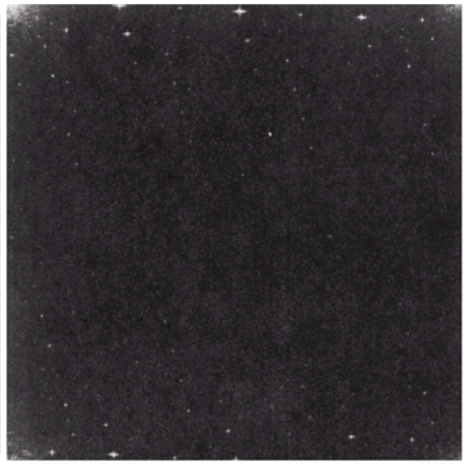
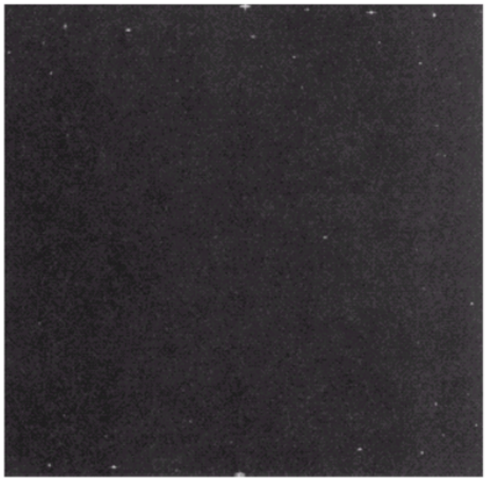
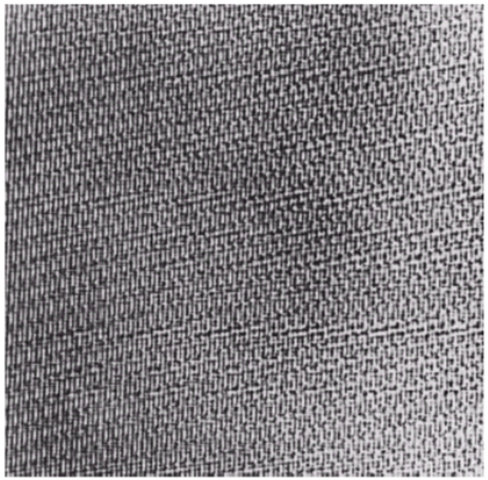


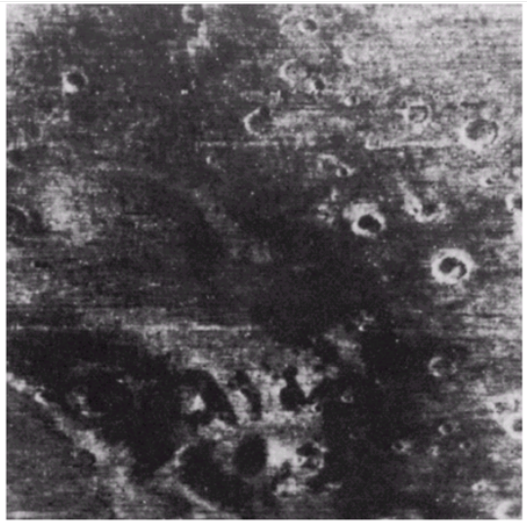
FIGURE 5.21 Fourier spectrum (without shifting) of the image shown in Fig. 5.20(a). (Courtesy of NASA.)





a b

FIGURE 5.22 (a) Fourier spectrum of $N(u, v)$, and (b) corresponding noise interference pattern $\eta(x, y)$. (Courtesy of NASA.)



E 5.23 Processed image. (Courtesy of NASA.)

Estimating the Degradation Function

- Estimation by image observation

  Let $G_s(u,v)$ denote the observed subimage, and $\hat{F}_s(u,v)$ denotes the estimate of the original subimage, and assuming the noise is negligible because of our choice of a strong-signal area, we have

$$H_s(u,v) = \frac{G_s(u,v)}{\hat{F}_s(u,v)}$$

  Then we can deduce the complete function $H(u,v)$ from $H_s(u,v)$

- Estimation by experimentation

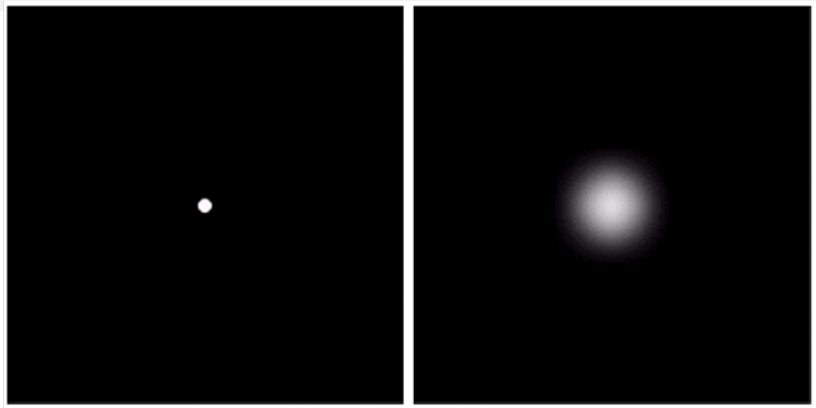
$$H(u,v) = \frac{G(u,v)}{A}$$

- Estimation by modeling

  A degradation model proposed by Hufnagel et al.[1964] is based on the phyiscal characteristics of atmosphere turbulence,

$$H(u,v) = exp[-k(u^2+v^2)^{5/6}]$$

Illustration of the Atmospheric Turbulence Model



a b

FIGURE 5.24 Degradation estimation by impulse characterization. (a) An impulse of light (shown magnified). (b) Imaged (degraded) impulse.

a b
c d

FIGURE 5.25 Illustration of the atmospheric turbulence model. (a) Negligible turbulence. (b) Severe turbulence, $k = 0.0025$. (c) Mild turbulence, $k = 0.001$. (d) Low turbulence, $k = 0.00025$. (Original image courtesy of NASA.)

Image Blur due to Motion

Now, assume an image has been blurred by uniform linear motion.

$$g(x,y) = \int_0^T f(x - x_0(t), y - y_0(t))dt$$

Then, its Fourier transform is

$$\begin{aligned} G(u,v) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)exp(-j2\pi(ux+vy))dxdy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}[\int_0^T f(x - x_0(t), y - y_0(t))dt]exp(-j2\pi(ux+vy))dxdy \\ &= \int_0^T[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x - x_0(t), y - y_0(t))exp(-j2\pi(ux+vy))dxdy]dt \end{aligned}$$

Image Blur due to Motion(Cont.)

$$= \int_0^T F(u,v)exp(-j2\pi(ux_0(t)+vy_0(t)))dt$$

$$= F(u,v)\int_0^T exp(-j2\pi(ux_0(t)+vy_0(t)))dt$$

Thus,

$$H(u,v) = \int_0^T exp(-j2\pi(ux_0(t)+vy_0(t)))dt$$

if $x_0(t) = at/T, y_0(t) = 0$, then

$$H(u,v) = \int_0^T exp(-j2\pi uat/T)dt = \frac{T}{\pi ua}\sin(\pi ua)exp(-j\pi ua)$$

if $y_0(t) = bt/T$ instead of 0, then

$$H(u,v) = \frac{T}{\pi(ua+vb)}\sin(\pi(ua+vb))exp(-j\pi(ua+vb))$$

Using matlab to Model Image Blur due to Motion

Image blur due to motion can be modeled by the function fspecial in IPT

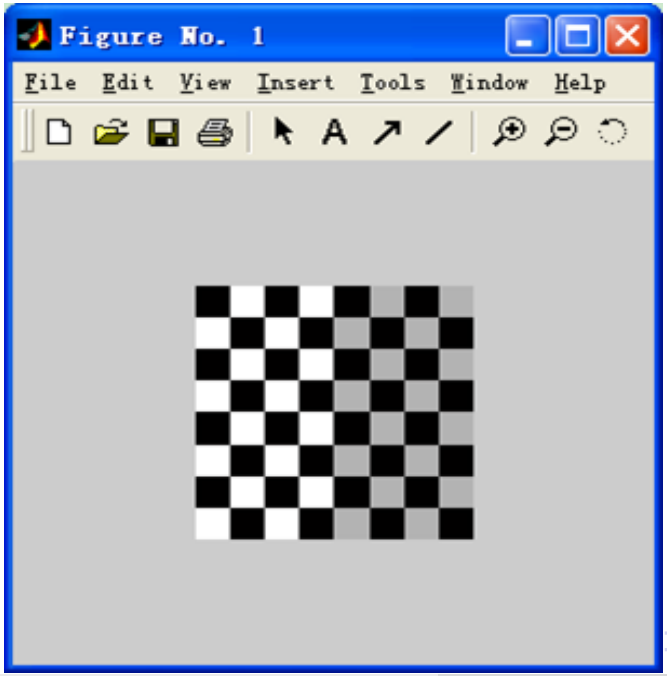
PSF = fspecial('motion',len,theta)

- g = imfilter(f,PSF, 'circular');
- g = g+noise;

The test pattern generated by checkerboard is very useful, and its syntax is

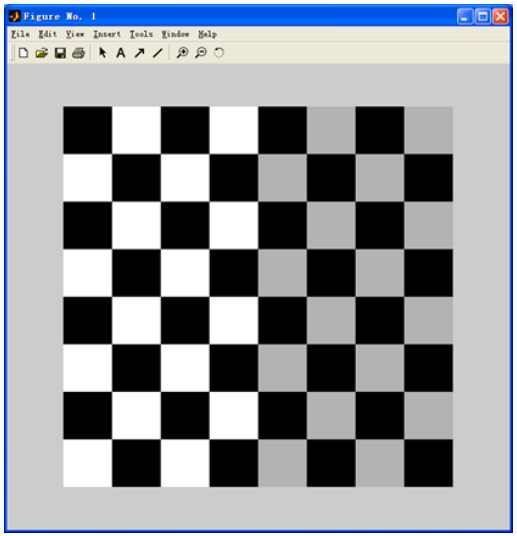
C = checkerboard(NP,M,N)

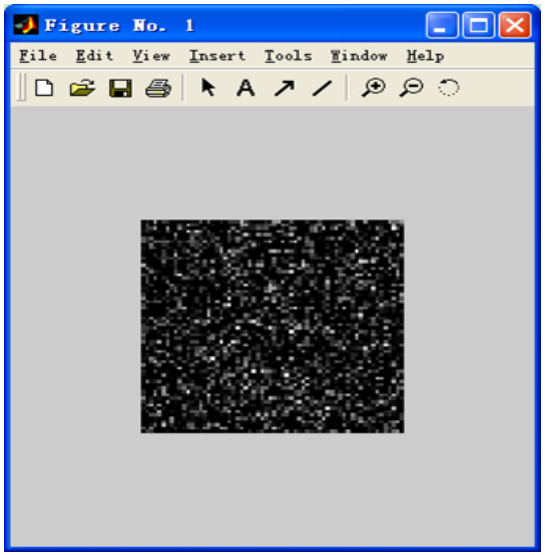
- f=checkerboard(8);
- imshow(f)

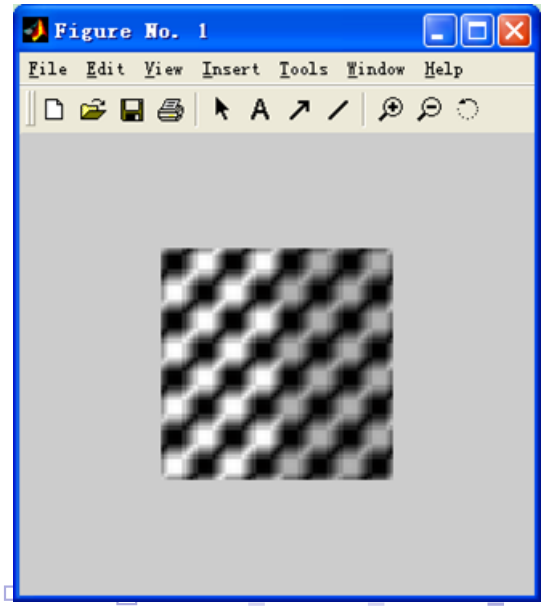


Using matlab to Model Image Blur due to Motion(cont.)

- PSF=fspecial('motion',7,45);
- gb=imfilter(f,PSF,'circular');
- noise=imnoise(zeros(size(f)),'gaussian',0,0.001);
- g=gb+noise;
- imshow(pixeldup(f,8),[])
- imshow(gb); imshow(noise,[])







Direct inverse filtering

- For an image degraded by a degradation function H, the simplest approach to restoration is direct inverse filtering

  $$\widehat{F}(u,v) = \frac{G(u,v)}{H(u,v)}$$

- Further, we can obtain

  $$\widehat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}$$

- If the degradation has zero or very small values, then the ratio $N(u,v)/H(u,v)$ could easily dominate the estimation of $F(u,v)$. One way to overcome this is to limit the filter frequency to values near the origin.

Direct inverse filtering (cont.)

a b
c d

FIGURE 5.27 Restoring Fig. 5.25(b) with Eq. (5.7-1). (a) Result of using the full filter. (b) Result with H cut off outside a radius of 40; (c) outside a radius of 70; and (d) outside a radius of 85.

Wiener Filtering

- A wiener filter seeks an estimate $\widehat{f}$ that minimizes the statistical error function

$$e^2 = E\{(f - \widehat{f})^2\}$$

- E is the expected value operator and f is the undergraded image.The solution to this expression in the frequency domain is

$$\widehat{F}(u,v) = [\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}]G(u,v)$$

where $H(u,v) =$ the degradation function

$|H(u,v)|^2 = H^*(u,v)H(u,v)$

$H^*(u,v) =$ the complex conjugate of $H(u,v)$

$S_\eta(u,v) = |N(u,v)|^2 =$ the power spectrum of the noise
$S_f(u,v) = |F(u,v)|^2 =$ the power spectrum of the undegraded image

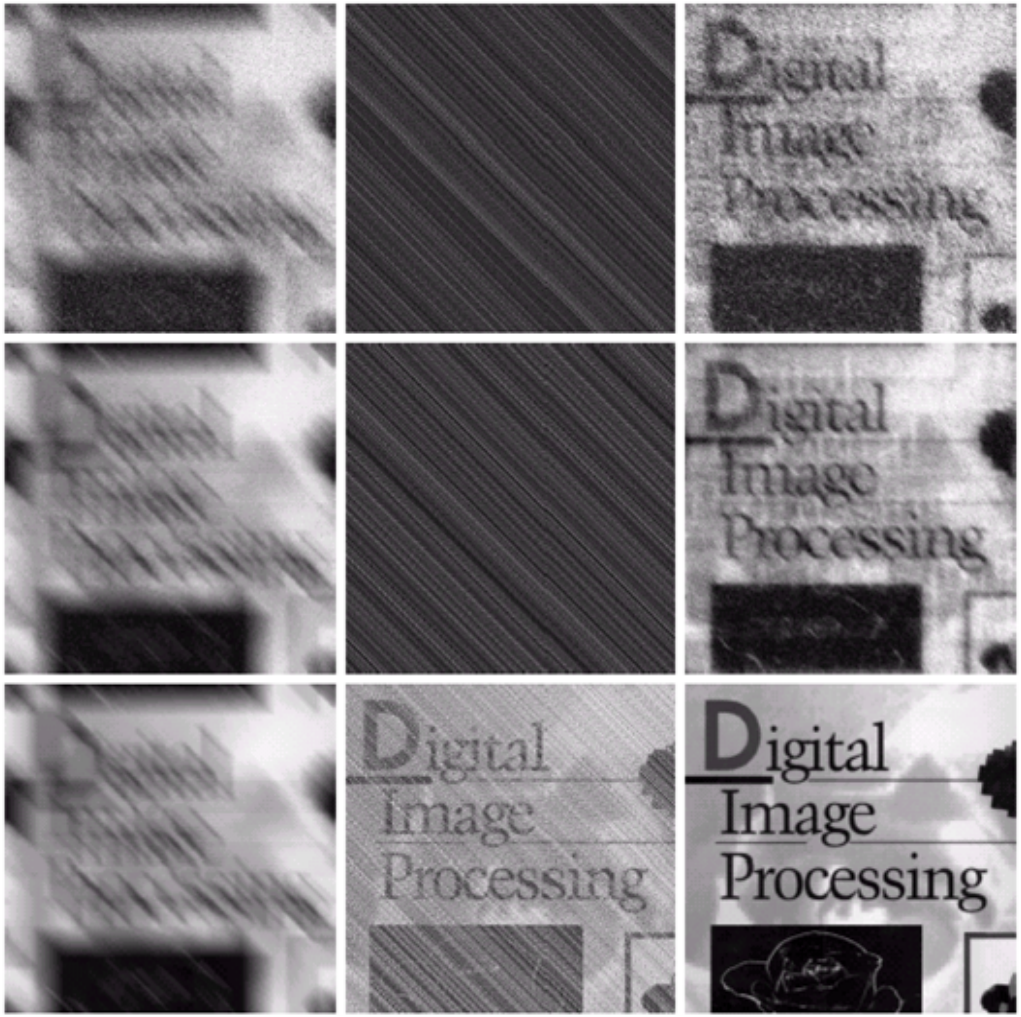
Wiener Filtering(cont.)

$$\widehat{F}(u,v) = [\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + K}]G(u,v)$$

a b c

FIGURE 5.28 Comparison of inverse- and Wiener filtering. (a) Result of full inverse filtering of Fig. 5.25(b). (b) Radially limited inverse filter result. (c) Wiener filter result.

Wiener Filtering(cont.)



a b c
d e f
g h i

FIGURE 5.29 (a) Image corrupted by motion blur and additive noise. (b) Result of inverse filtering. (c) Result of Wiener filtering. (d)–(f) Same sequence, but with noise variance one order of magnitude less. (g)–(i) Same sequence, but noise variance reduced by five orders of magnitude from (a). Note in (h) how the deblurred image is quite visible through a "curtain" of noise.

Wiener Filtering in Matlab(cont.)

- fr=deconvwnr(g,PSF)
- fr=deconvwnr(g,PSF,NSPR)
- fr=deconvwnr(g,PSF,NACORR,FACORR)

  $|F(u,v)|^2 = \Im[f(x,y) \circ f(x,y)]$
- If the restored image exhibits ringing, it sometimes helps to use function edgetaper prior to calling deconvwnr

  J=edgetaper(I,PSF)

Wiener filtering(cont.)

- PSF=fspecial('motion',7,45);
- g=imread('Fig0507(d).tif');
- fr1=deconvwnr(g,PSF);
- Sn=abs(fft2(noise)).^2;
- nA=sum(Sn(:))/prod(size(noise));
- Sf=abs(fft2(f)).^2;
- fA=sum(Sf(:))/prod(size(f));
- R=nA/fA;
- fr2=deconvwnr(g,PSF,R);
- NCORR=fftshift(real(ifft2(Sn)));
- ICORR=fftshift(real(ifft2(Sf)));
- fr3=deconvwnr(g,PSF,NCORR,ICORR);

Constrained Least Squares Filtering

- The problem of having to know something about the degradation function H is common to all methods discussed in this chapter.
- The Wiener filtering method presents additional difficulties: the power spectra of undegraded image and noise must be known. A constant estimate of the ratio of the power spectra can achieve excellent results sometimes, but does not mean it is always a suitable solution.
- The constrained least squares filtering method only requires knowledge of the mean and variance of noise.
- If we use the matrix to model the degradation procedure, we have

$$g(x,y) = h(x,y) * f(x,y) + \eta(x,y)$$

$$g = Hf + \eta$$

Constrained Least Squares Filtering(Cont.)

- Central to the method is the issue of sensitivity of H to noise. One way to alleviate the problem is to base the restoration on a measure of smoothness.
- The Laplacian (the second derivative of an image) seems to be a good choice.

$$\min C = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} [\nabla^2 f(x,y)]^2$$

Subject to: $\|g - H\hat{f}\| = \|\eta\|$

- The frequency solution of the problem is

$$\hat{F}(u,v) = [\frac{H^*(u,v)}{|H(u,v)|^2 + \gamma|P(u,v)|^2}]G(u,v)$$

and

$$p(x,y) = \begin{pmatrix} 0, & -1, & 0 \\ -1, & 4, & -1 \\ 0, & -1, & 0 \end{pmatrix}$$

Constrained Least Squares Filtering(Cont.)

- Let $r = g - H\hat{f}$ and $\varphi(r) = r^T r = \|r\|^2$
- It can be proved that $\varphi(r)$ is a monotonically increasing function of γ So we can adjust γ, so that:

$$\|r\|^2 = \|\eta\|^2 \pm a$$

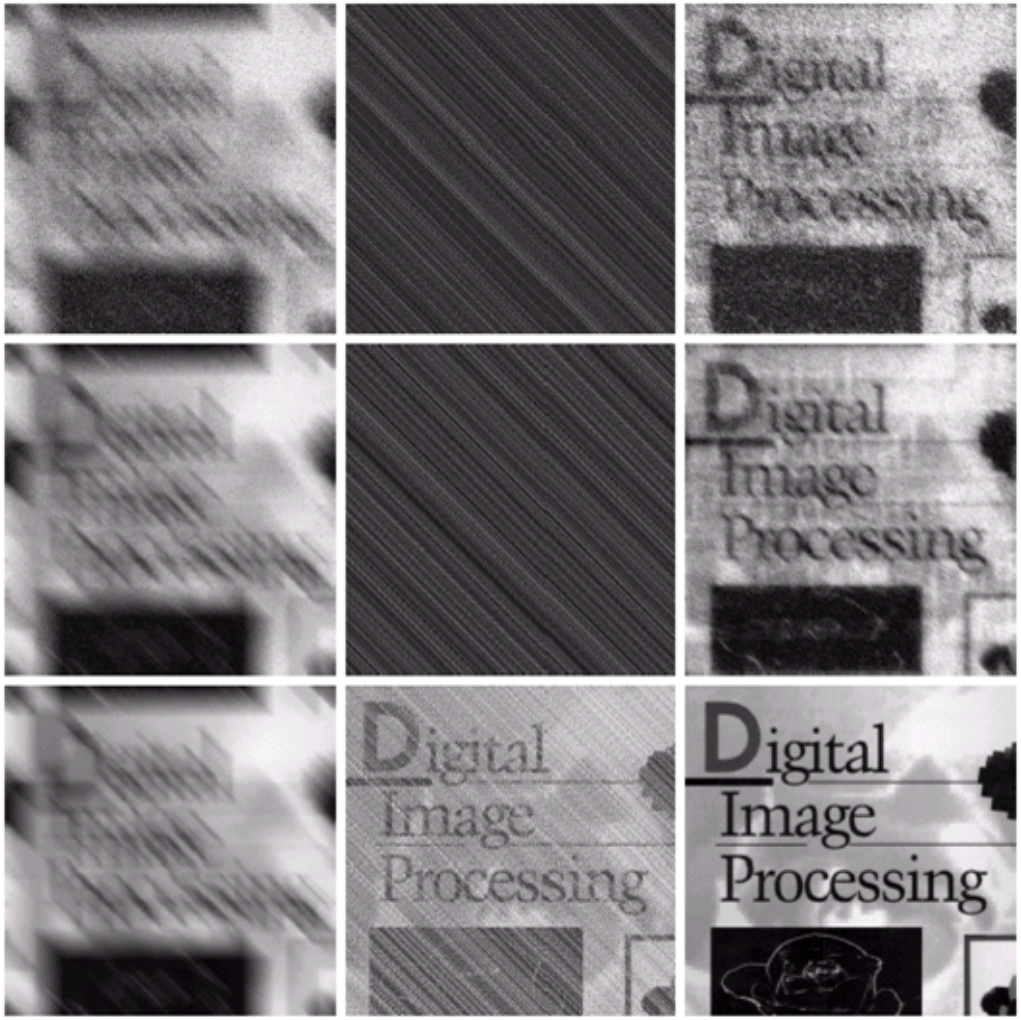
- How to evaluate the $\|\eta\|^2$(???)

$$\|\eta\|^2 = MN[\sigma_\eta^2 + m_\eta^2]$$

$$\sigma_\eta^2 = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\eta(x,y) - m_\eta]^2$$

$$m_\eta = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\eta(x,y)$$

Constrained Least Squares Filtering(Cont.)



a b c
d e f
g h i

FIGURE 5.29 (a) Image corrupted by motion blur and additive noise. (b) Result of inverse filtering. (c) Result of Wiener filtering. (d)–(f) Same sequence, but with noise variance one order of magnitude less. (g)–(i) Same sequence, but noise variance reduced by five orders of magnitude from (a). Note in (h) how the deblurred image is quite visible through a "curtain" of noise.

Constrained Least Squares Filtering(Cont.)

a b c

FIGURE 5.30 Results of constrained least squares filtering. Compare (a), (b), and (c) with the Wiener filtering results in Figs. 5.29(c), (f), and (i), respectively.

Image Processing and Analysis
lecture 11-12 Color Image Processing

Weiqiang Wang

School of Computer and Control Engineering, UCAS

December 1, 2015

Outline

Color Image Representation in MATLAB

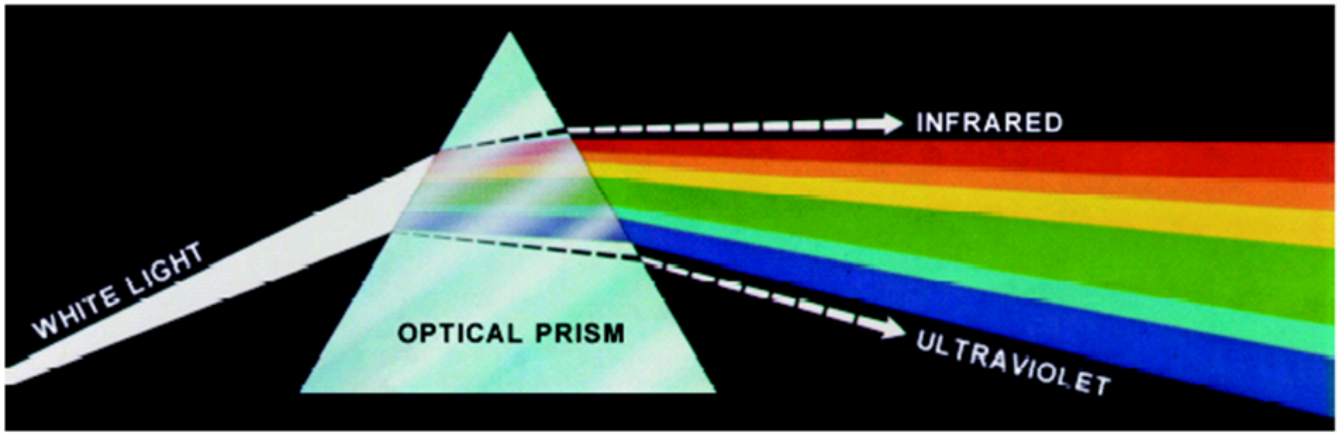


FIGURE 6.1 Color spectrum seen by passing white light through a prism. (Courtesy of the General Electric Co., Lamp Business Division.)

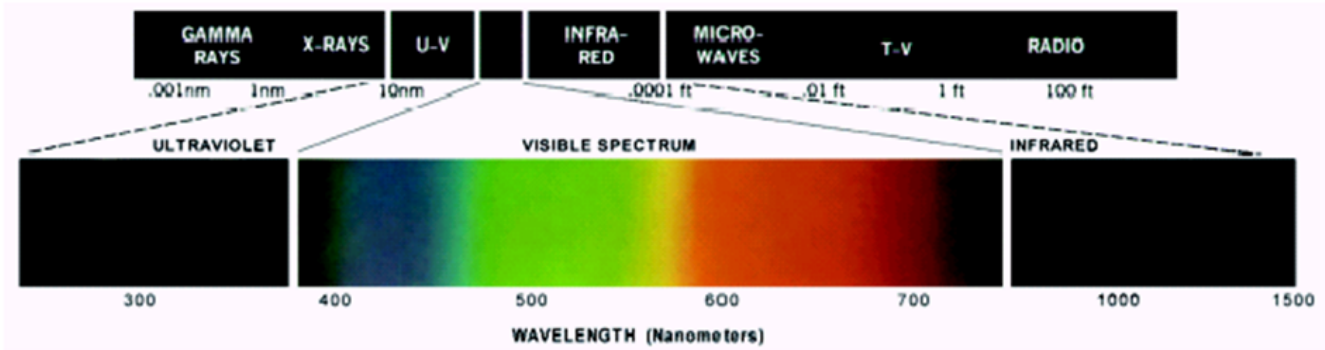


FIGURE 6.2 Wavelengths comprising the visible range of the electromagnetic spectrum. (Courtesy of the General Electric Co., Lamp Business Division.)

Color Image Representation in MATLAB(cont.)

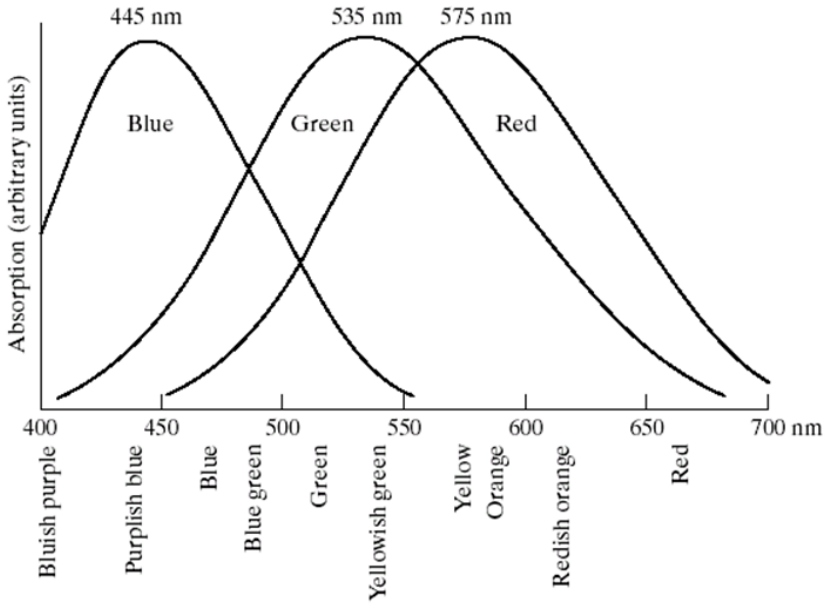
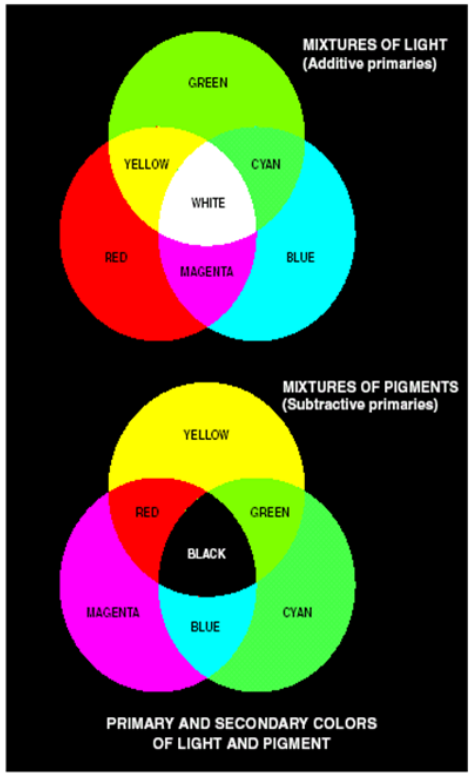


FIGURE 6.3 Absorption of light by the red, green, and blue cones in the human eye as a function of wavelength.



a
b

FIGURE 6.4 Primary and secondary colors of light and pigments. (Courtesy of the General Electric Co., Lamp Business Division.)

Color Image Representation in MATLAB(cont.)

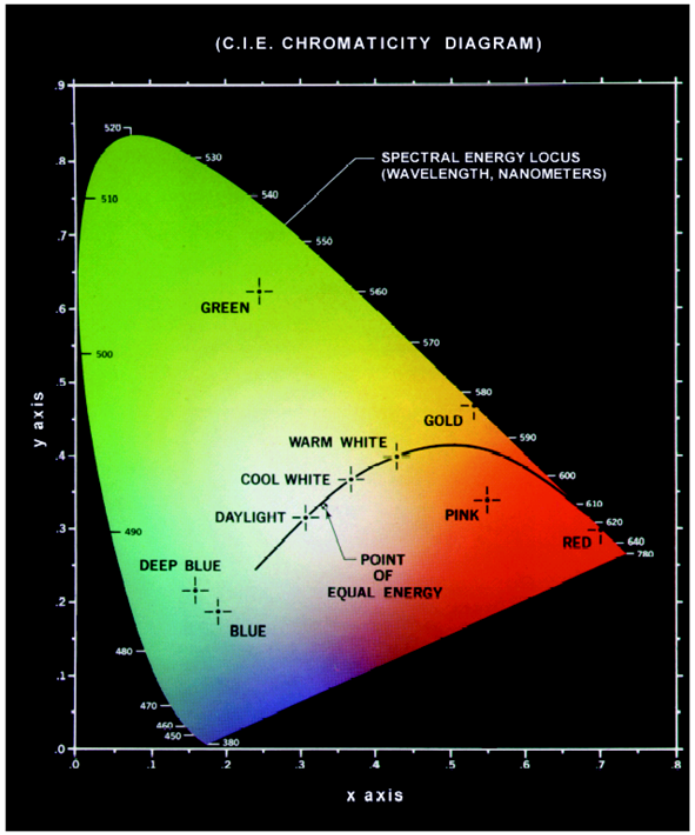


FIGURE 6.5 Chromaticity diagram. (Courtesy of the General Electric Co., Lamp Business Division.)

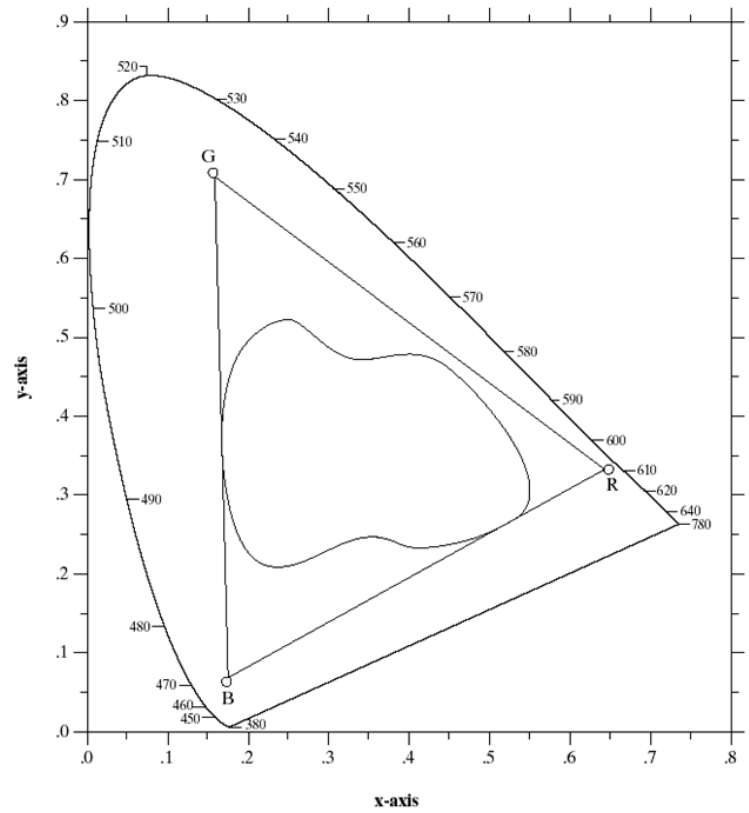
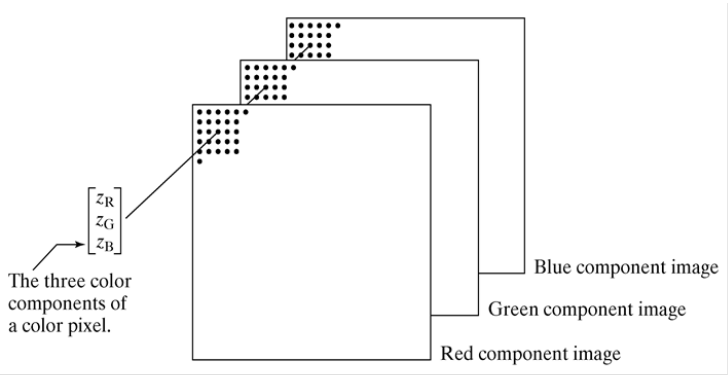
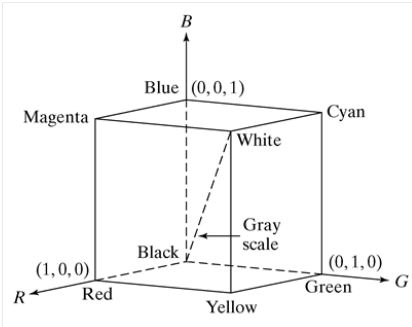


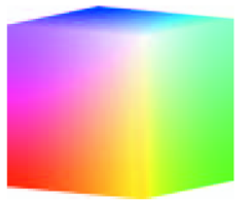
FIGURE 6.6 Typical color gamut of color monitors (triangle) and color printing devices (irregular region).

Color Image Representation in MATLAB(cont.)

- RGB: M×N×3 array of color pixels







- Cat operator to stack the images:

  rgb_image = cat(3,fr,fg,fb);

  fr= rgb_image(:,:,1);

  fg= rgb_image(:,:,2);

  fb= rgb_image(:,:,3);

- View the color cubic

  rgbcube(vx,vy,vz)

Color Image Representation in MATLAB(cont.)

- An indexed image consists of a data matrix of integers (two dimensions): X and Colormap matrix :map

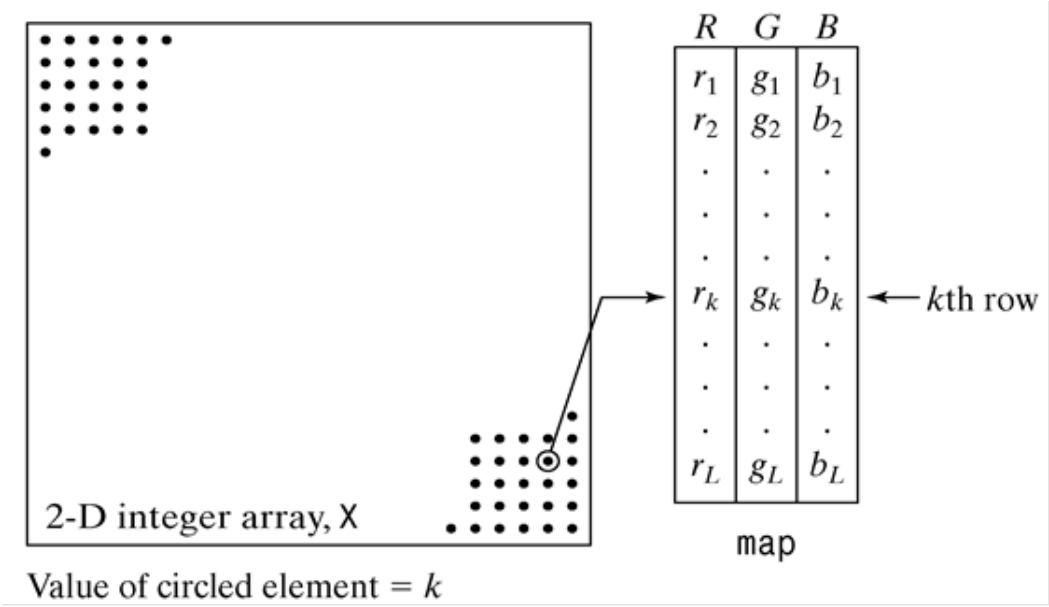


FIGURE 6.3 Elements of an indexed image. Note that the value of an element of integer array X determines the row number in the colormap. Each row contains an RGB triplet, and L is the total number of rows.

- *Display it using matlab:*

  imshow(x,map) Or image(X);colormap(map)

Color Image Representation in MATLAB(cont.)

- Approximate an indexed image by one with fewer colors

[Y,newmap]= imapprox(x,map,n)

Long name	Short name	RGB values
Black	k	[0 0 0]
Blue	b	[0 0 1]
Green	g	[0 1 0]
Cyan	c	[0 1 1]
Red	r	[1 0 0]
Magenta	m	[1 0 1]
Yellow	y	[1 1 0]
White	w	[1 1 1]

Name	Description
autumn	Varies smoothly from red, through orange, to yellow.
bone	A gray-scale colormap with a higher value for the blue component. This colormap is useful for adding an "electronic" look to gray-scale images.
colorcube	Contains as many regularly spaced colors in RGB color space as possible, while attempting to provide more steps of gray, pure red, pure green, and pure blue.
cool	Consists of colors that are shades of cyan and magenta. It varies smoothly from cyan to magenta.
copper	Varies smoothly from black to bright copper.
flag	Consists of the colors red, white, blue, and black. This colormap completely changes color with each index increment.
gray	Returns a linear gray-scale colormap.
hot	Varies smoothly from black, through shades of red, orange, and yellow, to white.
hsv	Varies the hue component of the hue-saturation-value color model. The colors begin with red, pass through yellow, green, cyan, blue, magenta, and return to red. The colormap is particularly appropriate for displaying periodic functions.
jet	Ranges from blue to red, and passes through the colors cyan, yellow, and orange.
lines	Produces a colormap of colors specified by the ColorOrder property and a shade of gray. Consult online help regarding function ColorOrder.
pink	Contains pastel shades of pink. The pink colormap provides sepia tone colorization of grayscale photographs.
prism	Repeats the six colors red, orange, yellow, green, blue, and violet.
spring	Consists of colors that are shades of magenta and yellow.
summer	Consists of colors that are shades of green and yellow.
white	This is an all white monochrome colormap.
winter	Consists of colors that are shades of blue and green.

Color Image Representation in MATLAB(cont.)

Function	Purpose
`dither`	Creates an indexed image from an RGB image by dithering.
`grayslice`	Creates an indexed image from a gray-scale intensity image by multilevel thresholding.
`gray2ind`	Creates an indexed image from a gray-scale intensity image.
`ind2gray`	Creates a gray-scale intensity image from an indexed image.
`rgb2ind`	Creates an indexed image from an RGB image.
`ind2rgb`	Creates an RGB image from an indexed image.
`rgb2gray`	Creates a gray-scale image from an RGB image.

TABLE 6.3
IPT functions for converting between RGB, indexed, and gray-scale intensity images.

a
b c
d e

FIGURE 6.4
(a) RGB image. (b) Number of colors reduced to 8 without dithering. (c) Number of colors reduced to 8 with dithering. (d) Gray-scale version of (a) obtained using function `rgb2gray`. (e) Dithered gray-scale image (this is a binary image).

Converting to Other Color Spaces

- RGB$\longrightarrow$NTSC

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

yiq_image=rgb2ntsc(rgb_image)

- NTSC$\longrightarrow$RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

rgb_image=ntsc2rgb(yiq_image)

Converting to Other Color Spaces(cont.)

- RGB$\longrightarrow$YCbCr

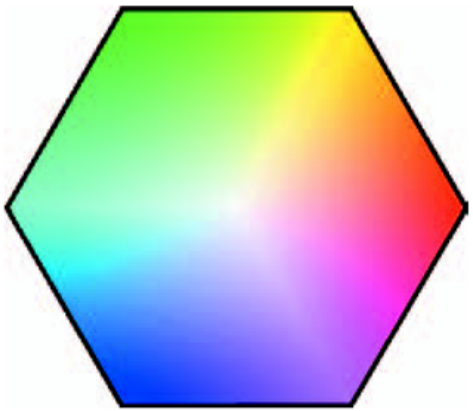
$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} /256(about)$$

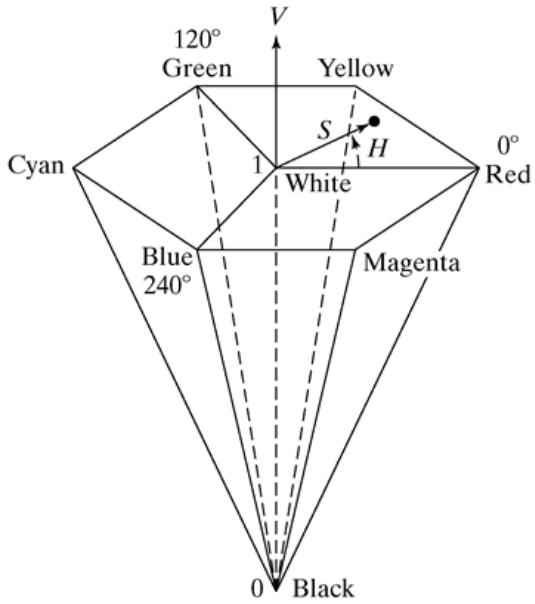
ycbcr_image=rgb2ycbcr(rgb_image)

rgb_image=ycbcr2rgb(ycbcr_image)

Converting to Other Color Spaces(cont.)

- HSV





a b

FIGURE 6.5
(a) The HSV color hexagon. (b) The HSV hexagonal cone.

Converting to Other Color Spaces(cont.)

- CMY & CMYK

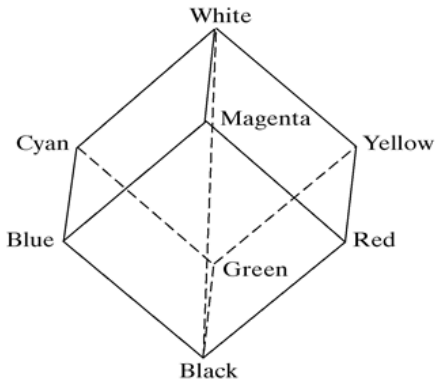
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

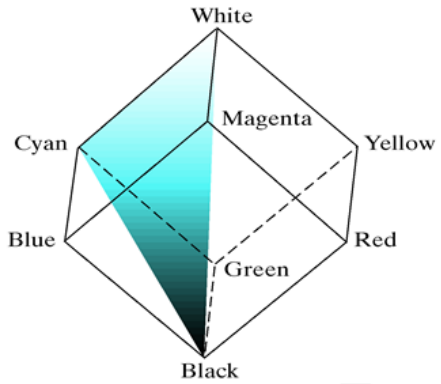
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

cmy_image=imcomplent(rgb_image) rgb_image=imcomplent(cmy_image)

Converting to Other Color Spaces(cont.)

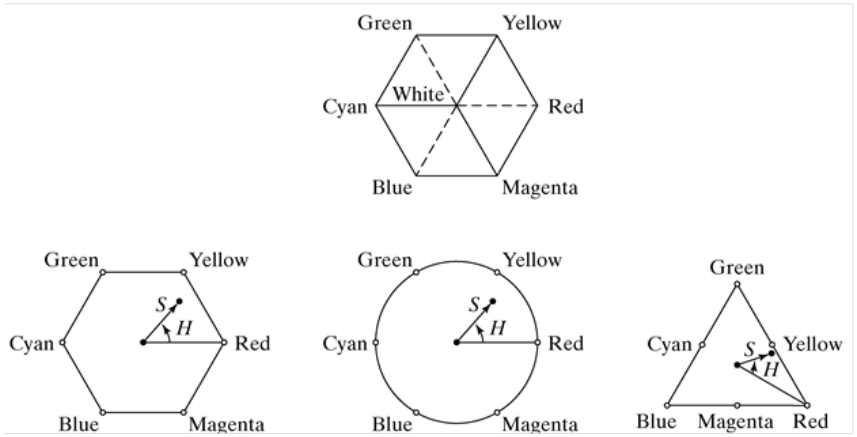
- HIS





a b

FIGURE 6.6 Relationship between the RGB and HSI color models.



a
b c d

FIGURE 6.7 Hue and saturation in the HSI color model. The dot is an arbitrary color point. The angle from the red axis gives the hue, and the length of the vector is the saturation. The intensity of all colors in any of these planes is given by the position of the plane on the vertical intensity axis.

Converting to Other Color Spaces(cont.)

- RGB$\longrightarrow$HSI

$$H = \begin{cases} \theta & if B \leq G \\ 360 - \theta & if B > G \end{cases}$$

$$\theta = \cos^{-1}\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}}\}$$

$$S = 1 - \frac{3}{R+G+B}[min(R,G,B)]$$

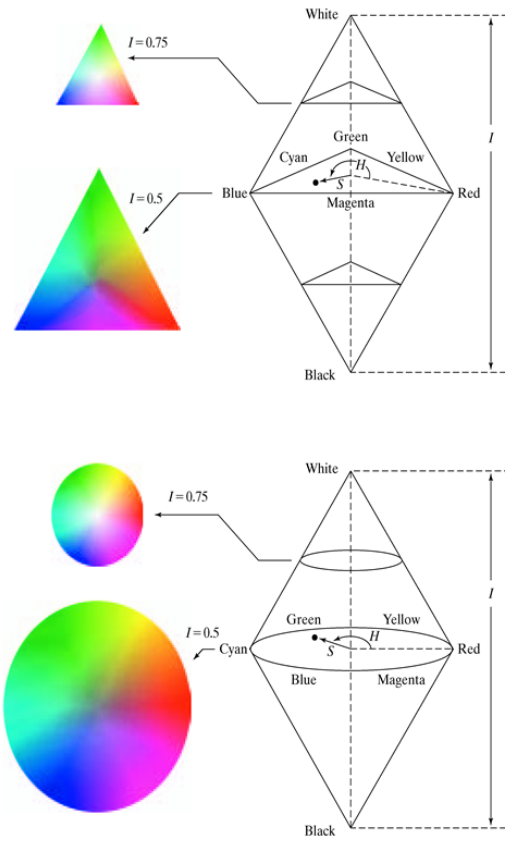
$$I = \frac{1}{3}(R+G+B)$$

- HSI$\longrightarrow$RGB
  - BR sector$(240° \leq H \leq 360°)$

$$H = H - 240° \quad G = I(1-S)$$

$$B = I[1 + \frac{S\cos H}{\cos(60°-H)}]$$

$$R = 3I - (R+B)$$



a
b

FIGURE 6.8 The HSI color model based on (a) triangular and (b) circular color planes. The triangles and circles are perpendicular to the vertical intensity axis.

Converting to Other Color Spaces(cont.)

- HSI$\longrightarrow$RGB
  - BR sector$(240^\circ \leq H \leq 360^\circ)$

$$H = H - 240^\circ \quad G = I(1 - S)$$

$$B = I[1 + \frac{S\cos H}{\cos(60^\circ - H)}]$$

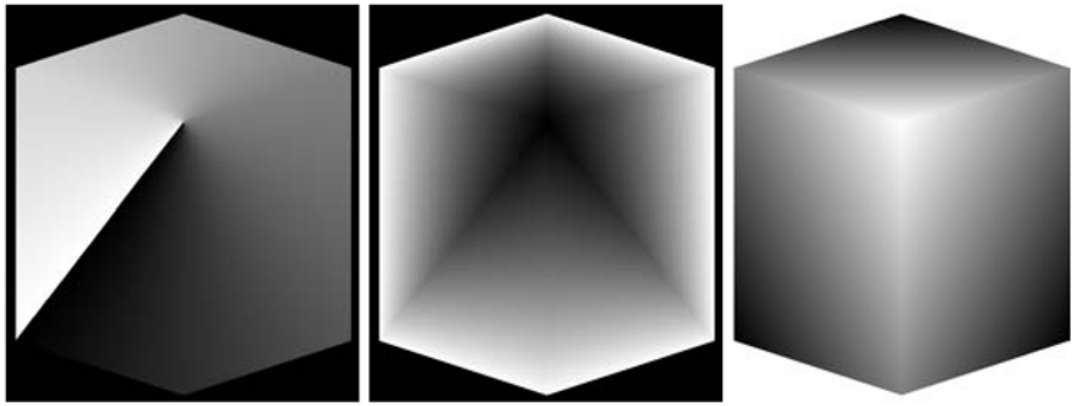
$$R = 3I - (R + B)$$

- HSI$\longrightarrow$RGB
  - GB sector$(120^\circ \leq H \leq 240^\circ)$

$$R = I(1 - S) \quad H = H - 120^\circ$$

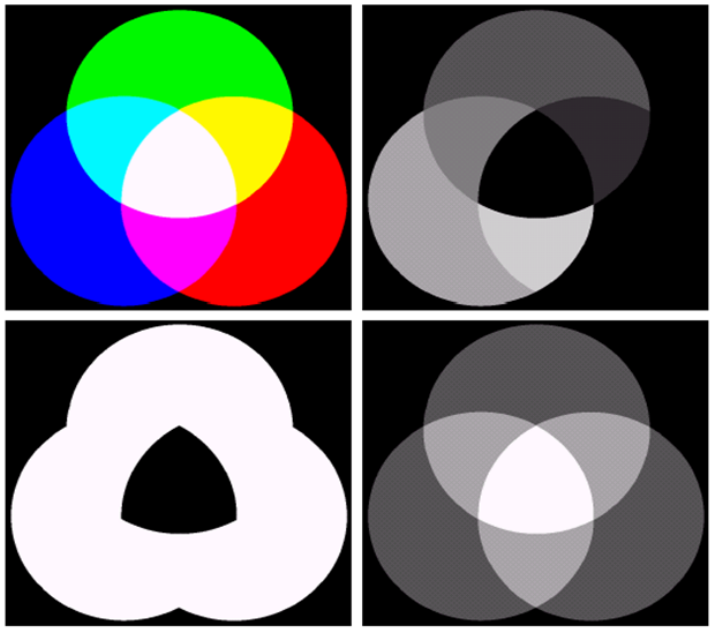
$$G = I[1 + \frac{S\cos H}{\cos(60^\circ - H)}]$$

$$B = 3I - (R + B)$$



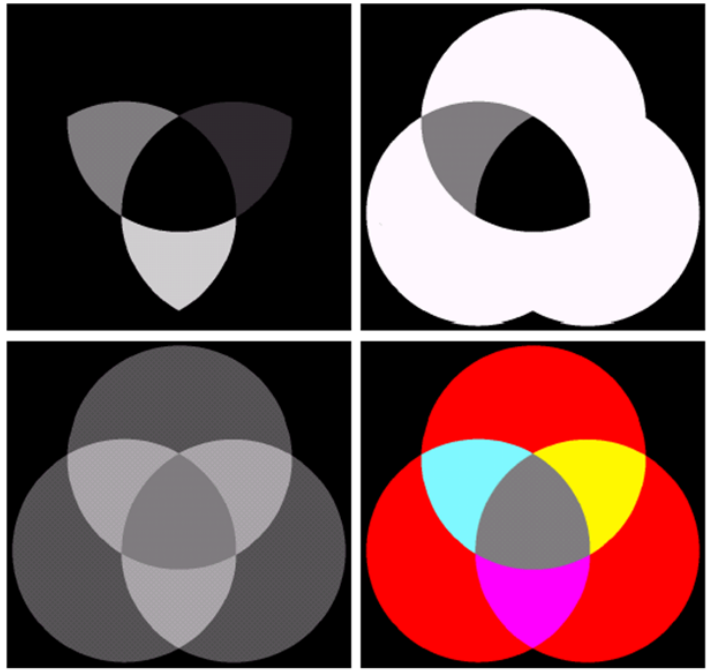
a b c

FIGURE 6.9 HSI component images of an image of an RGB color cube. (a) Hue, (b) saturation, and (c) intensity images.



a b
c d

FIGURE 6.16 (a) RGB image and the components of its corresponding HSI image: (b) hue, (c) saturation, and (d) intensity.

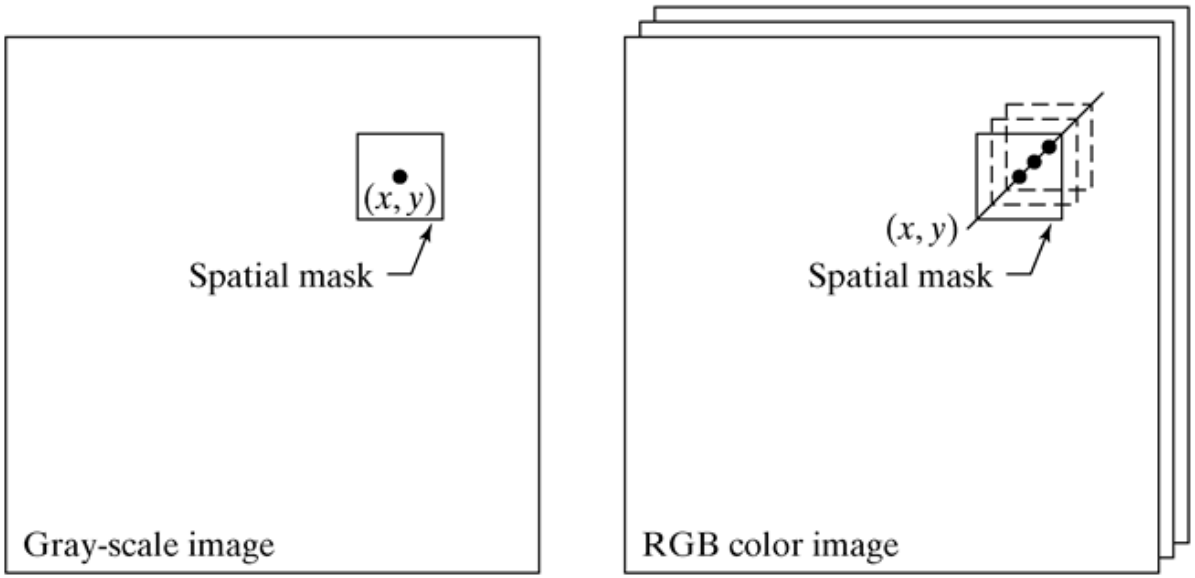


a b
c d

FIGURE 6.17 (a)–(c) Modified HSI component images. (d) Resulting RGB image. (See Fig. 6.16 for the original HSI images.)

The Basics of Color Image Processing

$$c = \begin{bmatrix} cr \\ cg \\ cb \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad c(x,y) = \begin{bmatrix} cr(x,y) \\ cg(x,y) \\ cb(x,y) \end{bmatrix} = \begin{bmatrix} R(x,y) \\ G(x,y) \\ B(x,y) \end{bmatrix}$$



a b

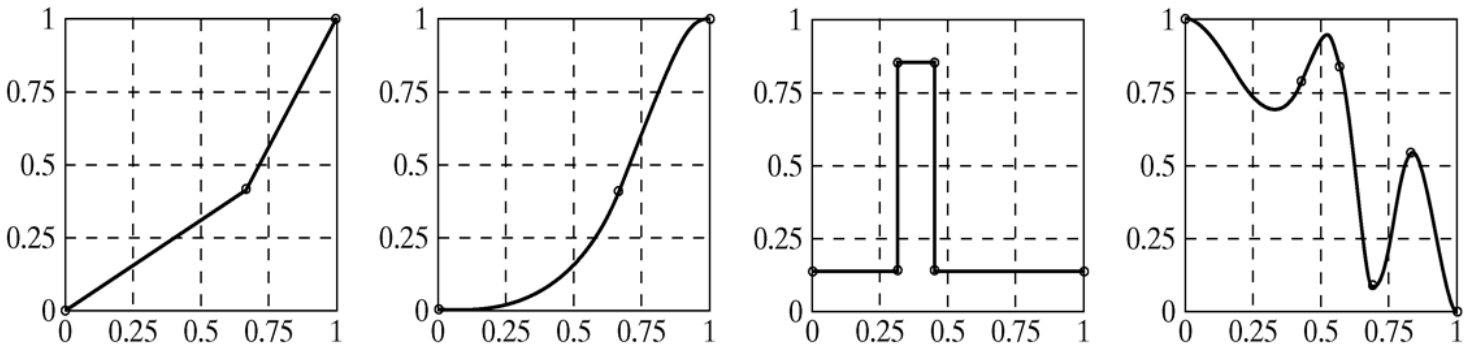
FIGURE 6.10 Spatial masks for gray-scale and RGB color images.

Color Transformations

- $s_i = T_i(r_i),\ i = 1, 2, ..., n$

  $monochrome$

  $s_i = T_i(r),\ i = 1, 2, ..., n$
- Interpolation



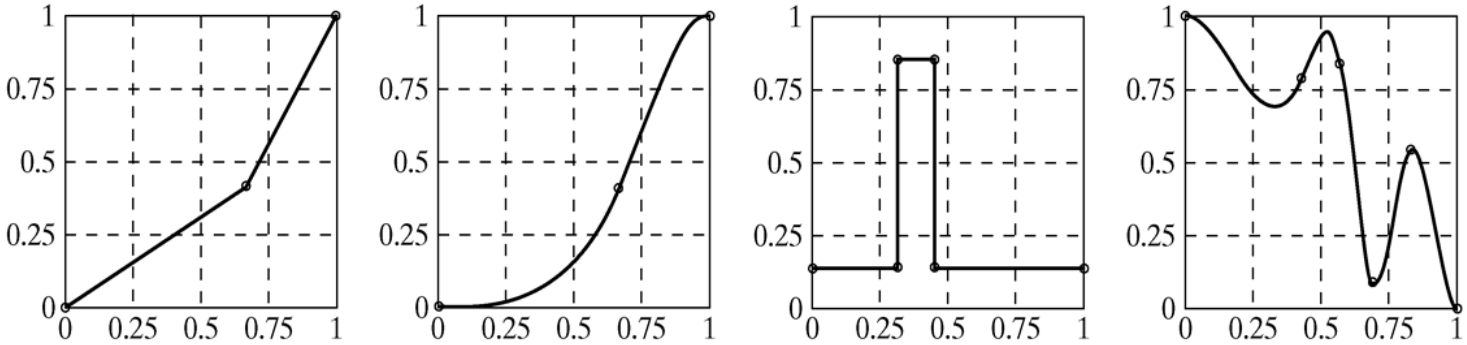
a b c d

FIGURE 6.11 Specifying mapping functions using control points: (a) and (c) linear interpolation, and (b) and (d) cubic spline interpolation.

Color Transformations

- z=interp1q(x,y,xi);
- Figure 6.11(a)
  z=interp1q([0 0.6 1]',[0 0.4 1]',linspace(0,1,10)');
  plot(linspace(0,1,10),z);
  grid;

- z=spline(x,y,xi);
- Figure 6.11(b)
  z=spline([0 0.6 1]',[0 0.4 1]',linspace(0,1,10)');
  plot(linspace(0,1,10),z);
  grid;



a b c d

FIGURE 6.11 Specifying mapping functions using control points: (a) and (c) linear interpolation, and (b) and (d) cubic spline interpolation.

Color Transformations

- ICE (interactive color editing)

  g=ice('Property Name' ,' property value' ,...)

Property Name	Property Value
'image'	An RGB or monochrome input image, f, to be transformed by interactively specified mappings.
'space'	The color space of the components to be modified. Possible values are 'rgb', 'cmy', 'hsi', 'hsv', 'ntsc' (or 'yiq'), and 'ycbcr'. The default is 'rgb'.
'wait'	If 'on' (the default), g is the mapped input image. If 'off', g is the handle of the mapped input image.

```
>>ice
>>f=imread('Fig0303(a)(breast).tif');
>>g=ice('image',f);
>>g=ice('image',f, 'wait', 'off');
>>g=ice('image',f, 'space', 'hsi');
```

Color Transformations

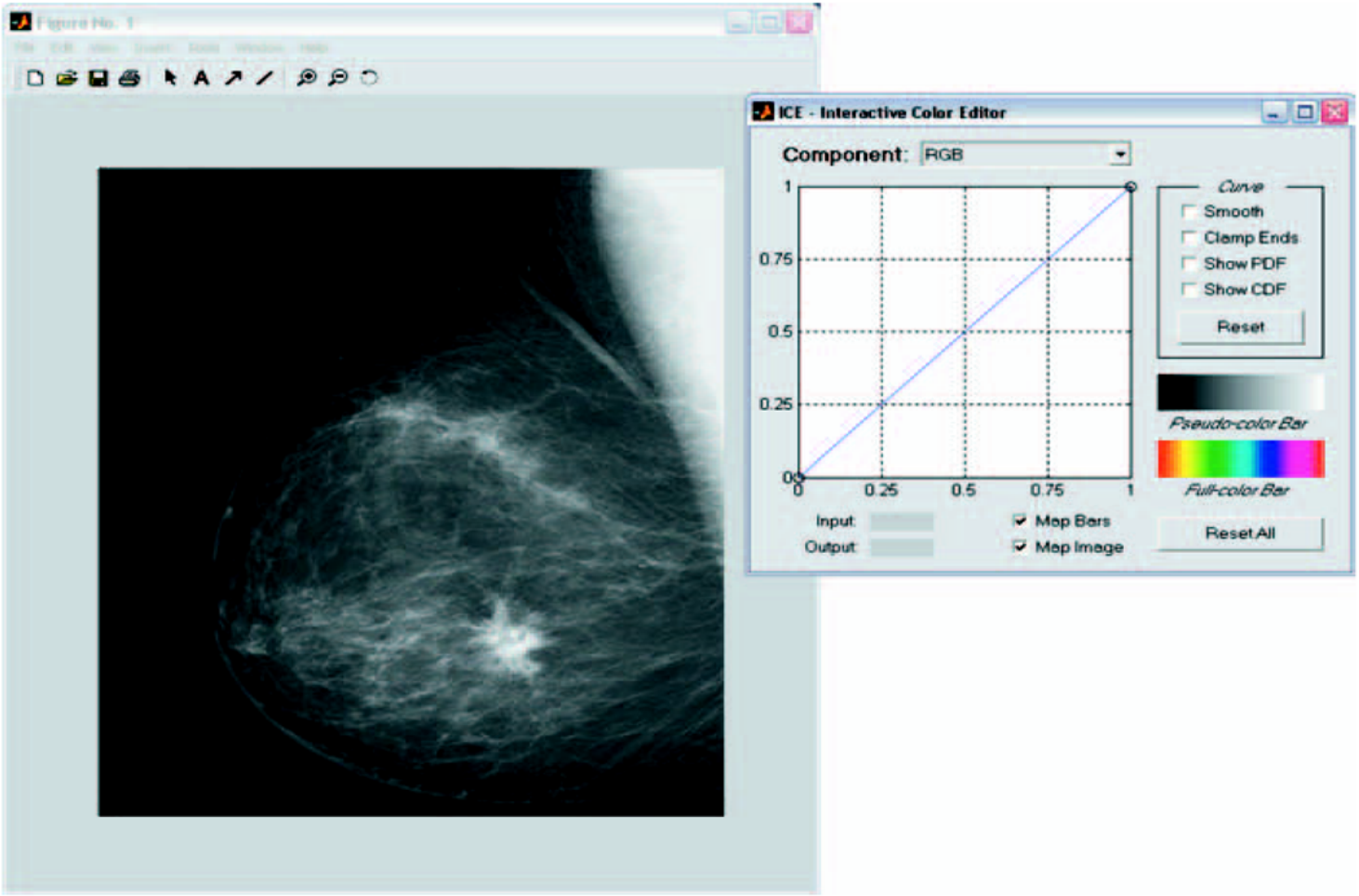
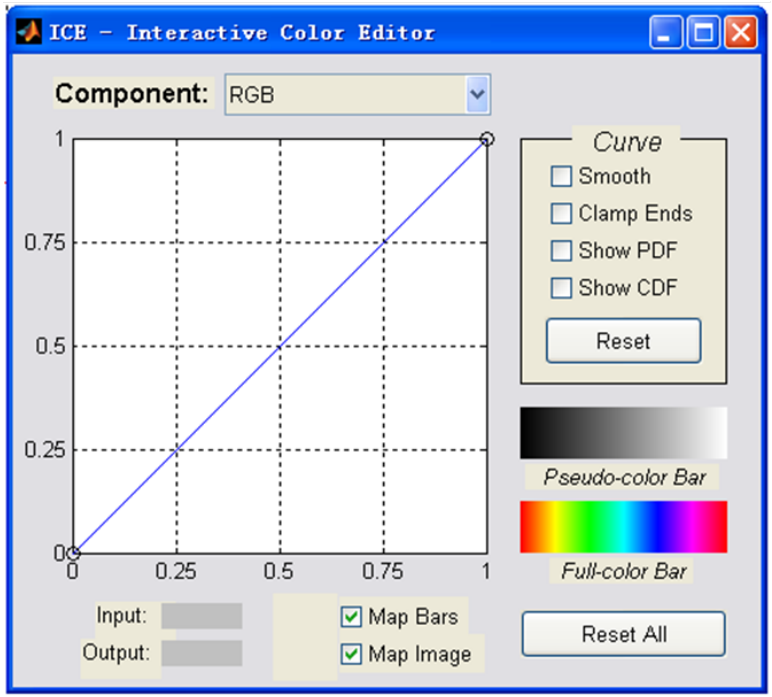


FIGURE 6.12 The typical opening windows of function `ice`. (Image courtesy of G. E. Medical Systems.)

ICE

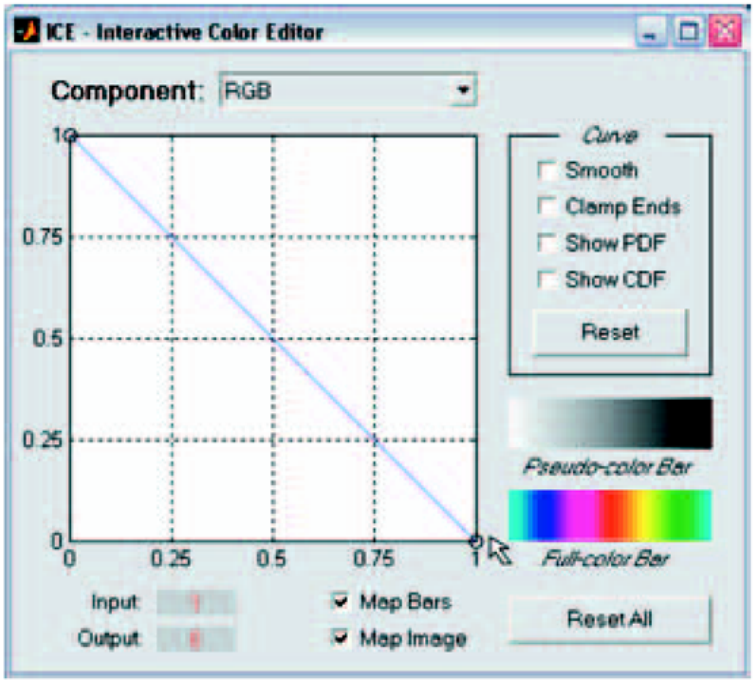


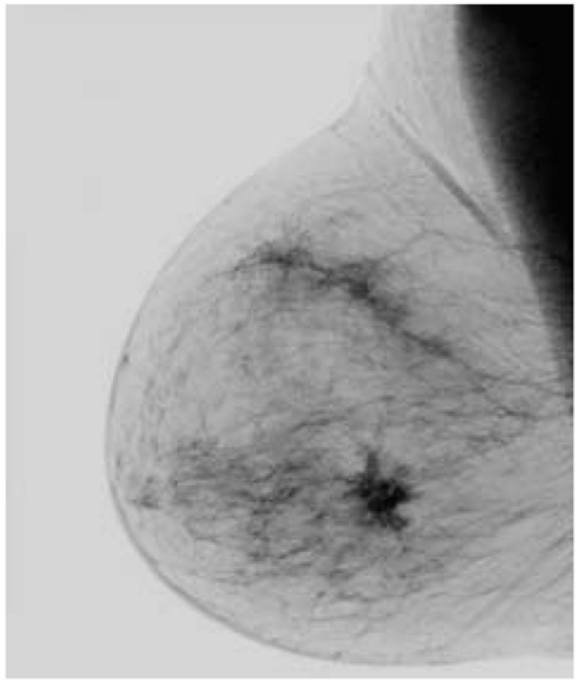
Mouse Action†	Result
Left Button	Move control point by pressing and dragging.
Left Button + Shift Key	Add control point. The location of the control point can be changed by dragging (while still pressing the Shift Key).
Left Button + Control Key	Delete control point.

† For three button mice, the left, middle, and right buttons correspond to the move, add, and delete operations in the table.

GUI Element	Function
Smooth	Checked for cubic spline (smooth curve) interpolation. If unchecked, piecewise linear interpolation is used.
Clamp Ends	Checked to force the starting and ending curve slopes in cubic spline interpolation to 0. Piecewise linear interpolation is not affected.
Show PDF	Display probability density function(s) [i.e., histogram(s)] of the image components affected by the mapping function.
Show CDF	Display cumulative distribution function(s) instead of PDFs. (Note: PDFs and CDFs cannot be displayed simultaneously.)
Map Image	If checked, image mapping is enabled; otherwise it is not.
Map Bars	If checked, pseudo- and full-color bar mapping is enabled; otherwise the unmapped bars (a gray wedge and hue wedge, respectively) are displayed.
Reset	Initialize the currently displayed mapping function and uncheck all curve parameters.
Reset All	Initialize all mapping functions.
Input/Output	Shows the coordinates of a *selected* control point on the transformation curve. Input refers to the horizontal axis, and Output to the vertical axis.
Component	Select a mapping function for interactive manipulation. In RGB space, possible selections include R, G, B, and RGB (which maps all three color components). In HSI space, the options are H, S, I, and HSI, and so on.

Inverse Mappings: Monochrome negatives





a b

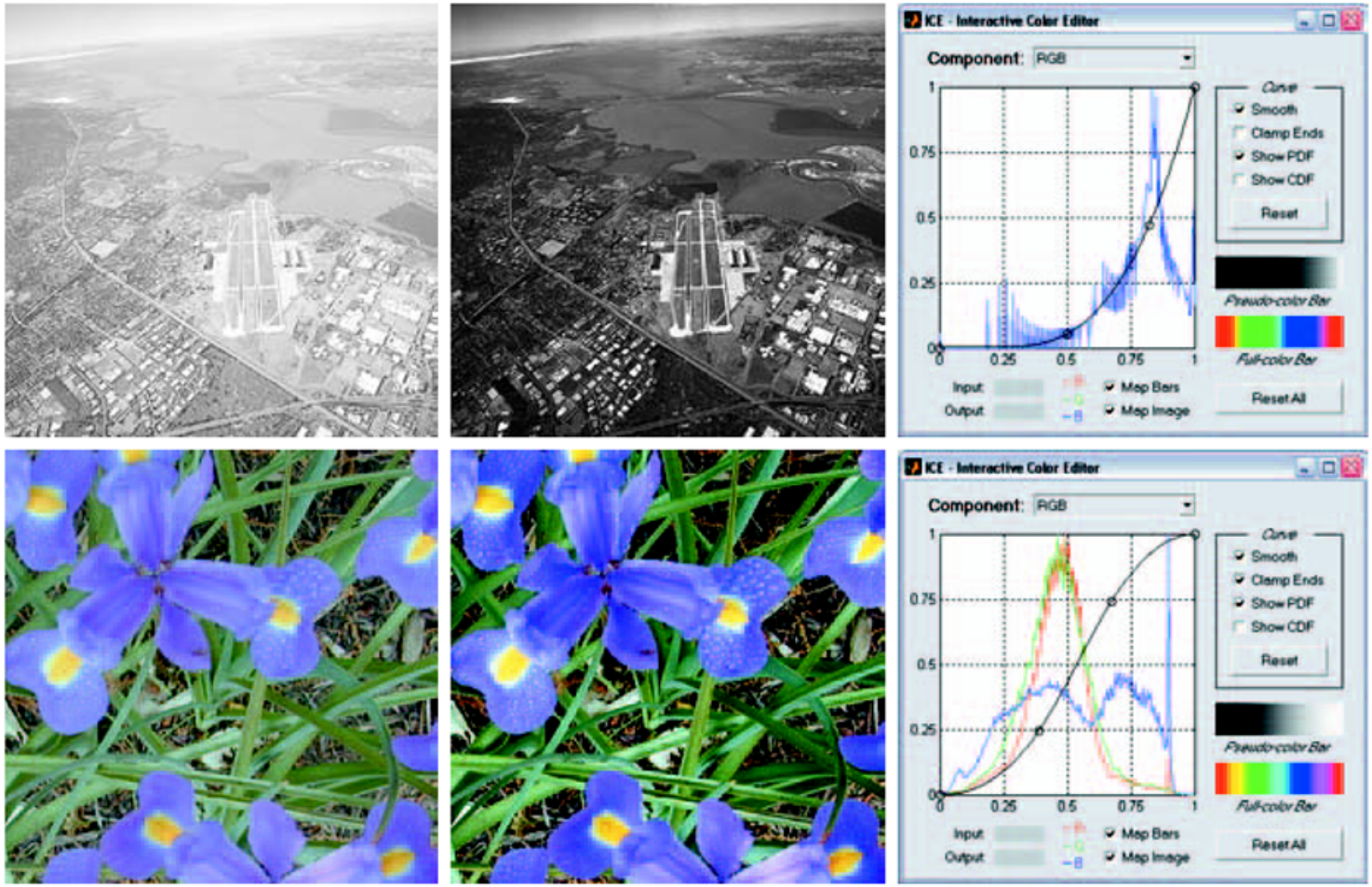
FIGURE 6.13
(a) A negative mapping function, and (b) its effect on the monochrome image of Fig. 6.12.

Inverse Mappings: Color Complement

a b

FIGURE 6.14
(a) A full color image, and (b) its negative (color complement).

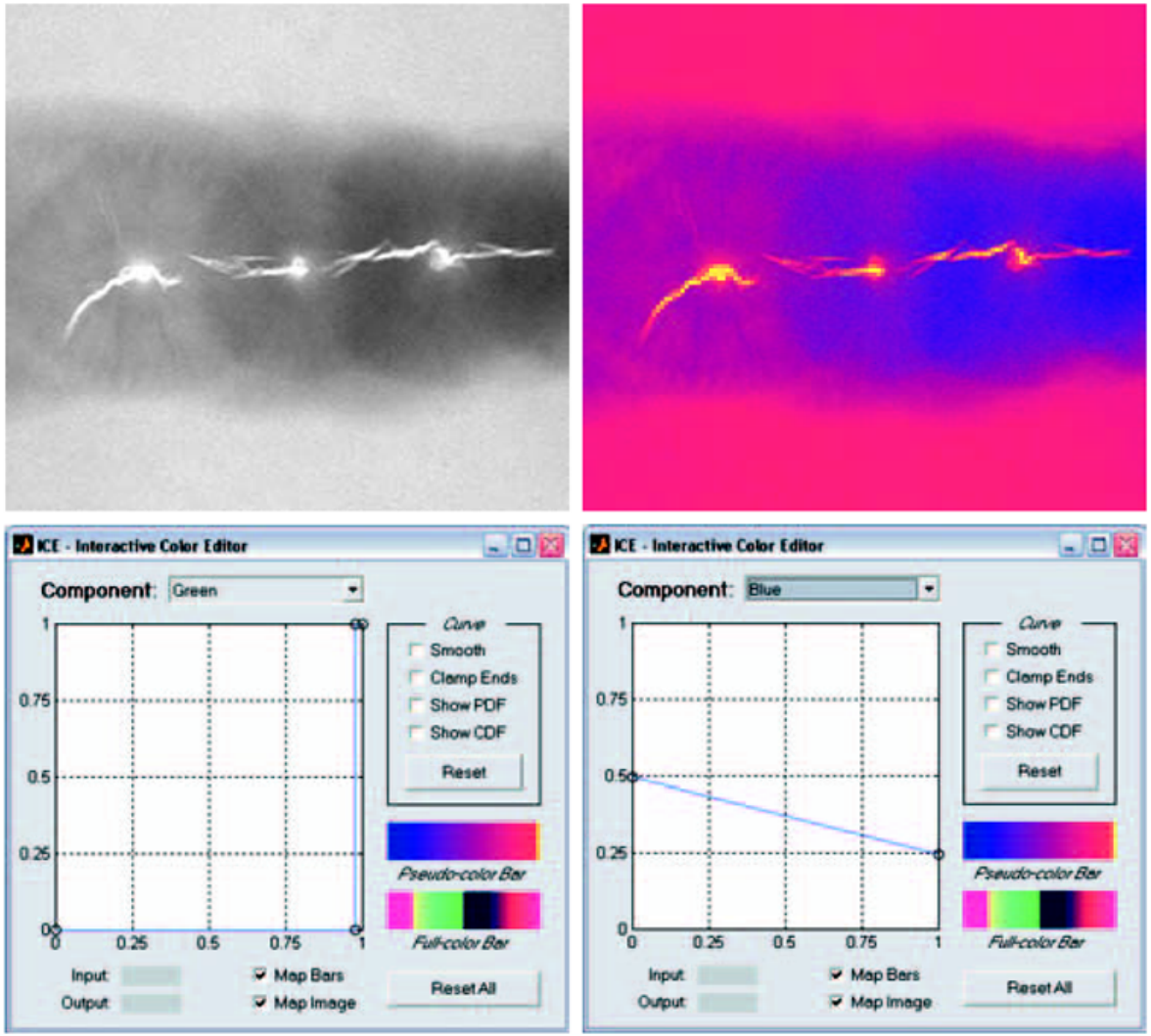
Color Transformations



a b c
d e f

FIGURE 6.15 Using function `ice` for monochrome and full color contrast enhancement: (a) and (d) are the input images, both of which have a "washed-out" appearance; (b) and (e) show the processed results; (c) and (f) are the `ice` displays. (Original monochrome image for this example courtesy of NASA.)

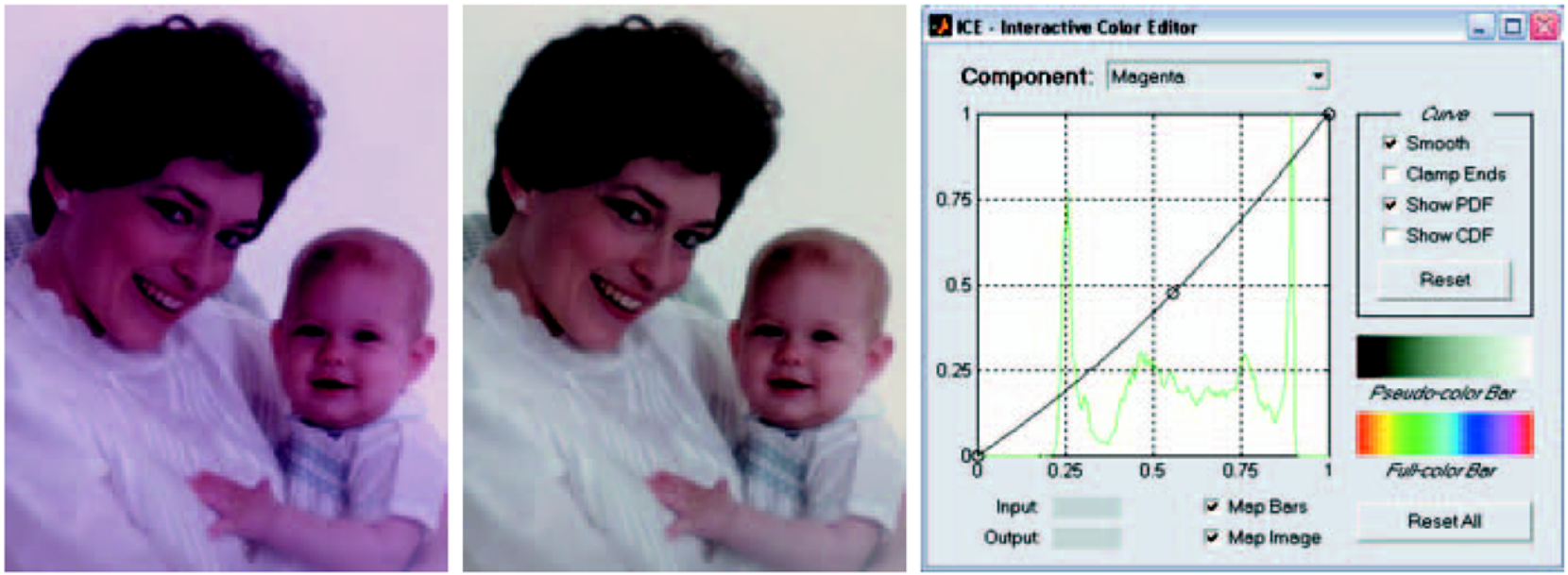
Pseudocolor Mappings



a b
c d

FIGURE 6.16
(a) X-ray of a defective weld; (b) a pseudo-color version of the weld; (c) and (d) mapping functions for the green and blue components. (Original image courtesy of X-TEK Systems, Ltd.)

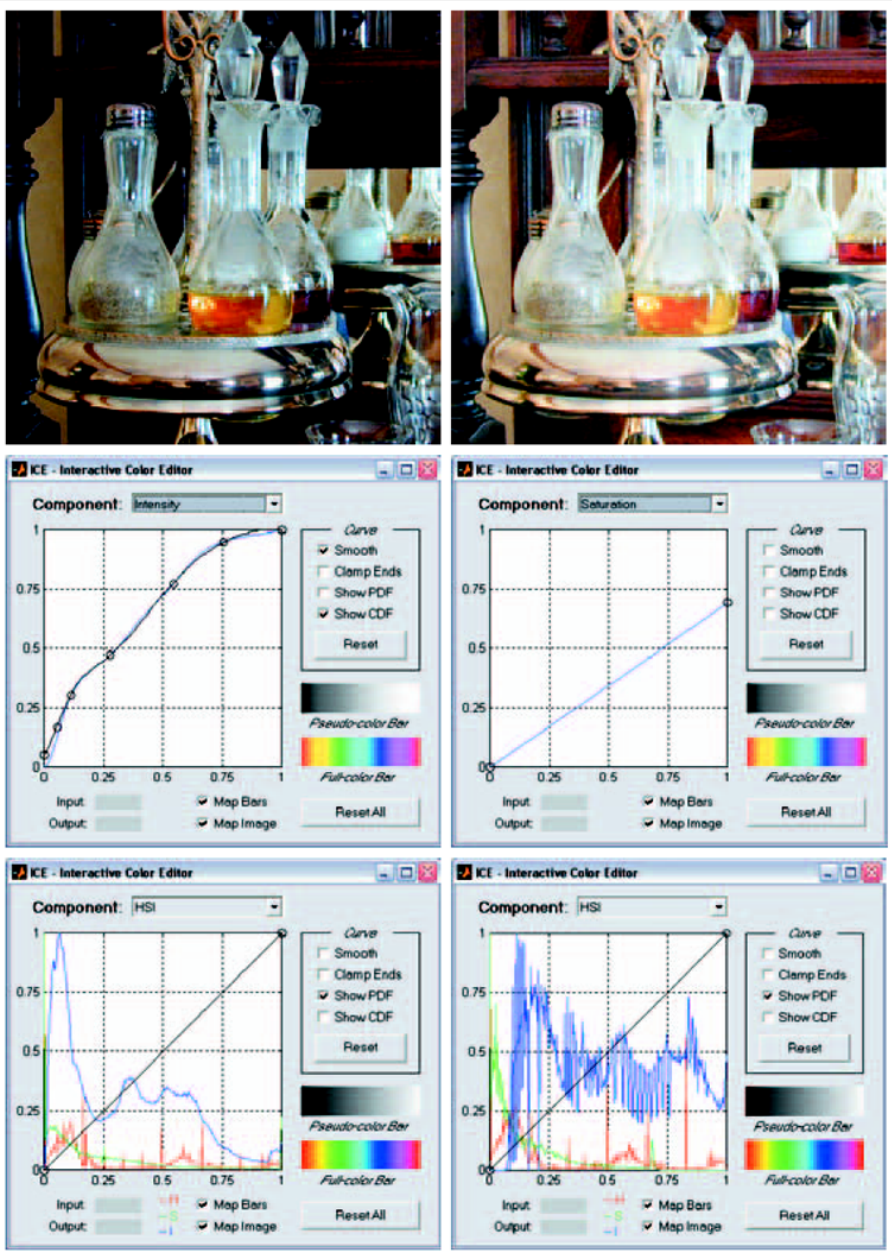
Color Balancing



a b c

FIGURE 6.17 Using function `ice` for color balancing: (a) an image heavy in magenta; (b) the corrected image; and (c) the mapping function used to correct the imbalance.

Histogram Equalization for Color Images



a b
c d
e f

FIGURE 6.18 Histogram equalization followed by saturation adjustment in the HSI color space: (a) input image; (b) mapped result; (c) intensity component mapping function and cumulative distribution function; (d) saturation component mapping function; (e) input image's component histograms; and (f) mapped result's component histograms.

Spatial Filtering of Color Images

- The average of the RGB vectors in its neighborhood is

$$\bar{c}(x,y) = \frac{1}{K} \sum_{(s,t)\in S_{xy}} c(s,t)$$

$$\bar{c}(x,y) = \begin{bmatrix} \frac{1}{K} \sum_{(s,t)\in S_{xy}} R(s,t) \\ \frac{1}{K} \sum_{(s,t)\in S_{xy}} G(s,t) \\ \frac{1}{K} \sum_{(s,t)\in S_{xy}} B(s,t) \end{bmatrix}$$

Spatial Filtering of Color Images

- Smoothing an RGB color image,fc, with a linear spatial filter consists of the following steps:
  - 1.Extract

    fR=fc(:,:,1);...
  - 2.Filter each component image individually

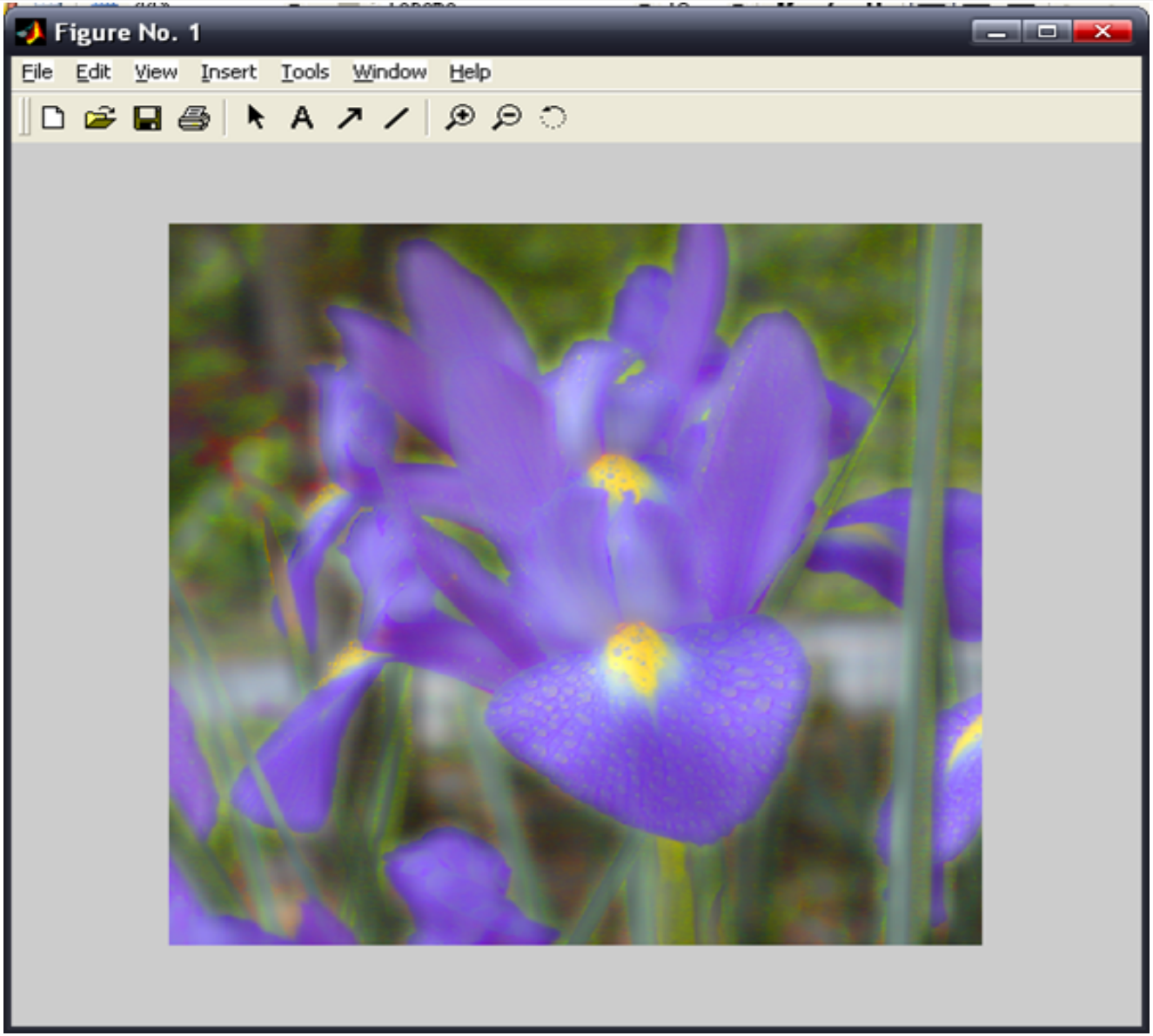
    fR_filtered=imfilter(fR,w);...
  - 3.Reconstruct

    fc_filtered=cat(3,fR_filtered,..);

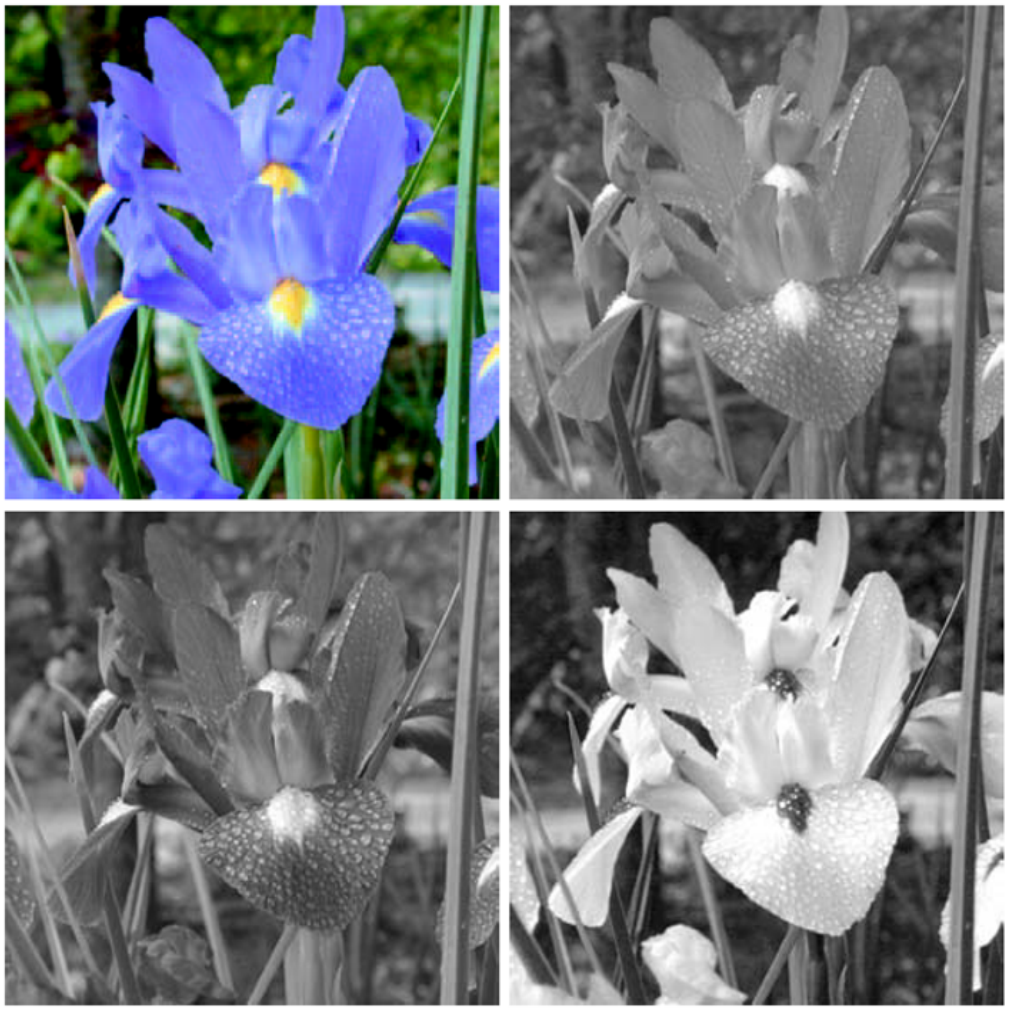
Spatial Filtering of Color Images

- >> fc=imread('Fig0619(a)(RGB_iris).tif');
- >> h=rgb2hsi(fc);
- >> H=h(:,:,1);S=h(:,:,2);I=h(:,:,3);
- >> w=fspecial('average',25);
- >> I_filtered=imfilter(I,w,'replicate');
- >> h=cat(3,H,S,I_filtered);
- >> f=hsi2rgb(h);
- >> f=min(f,1);
- >> imshow(f);

Spatial Filtering of Color Images



Spatial Filtering of Color Images



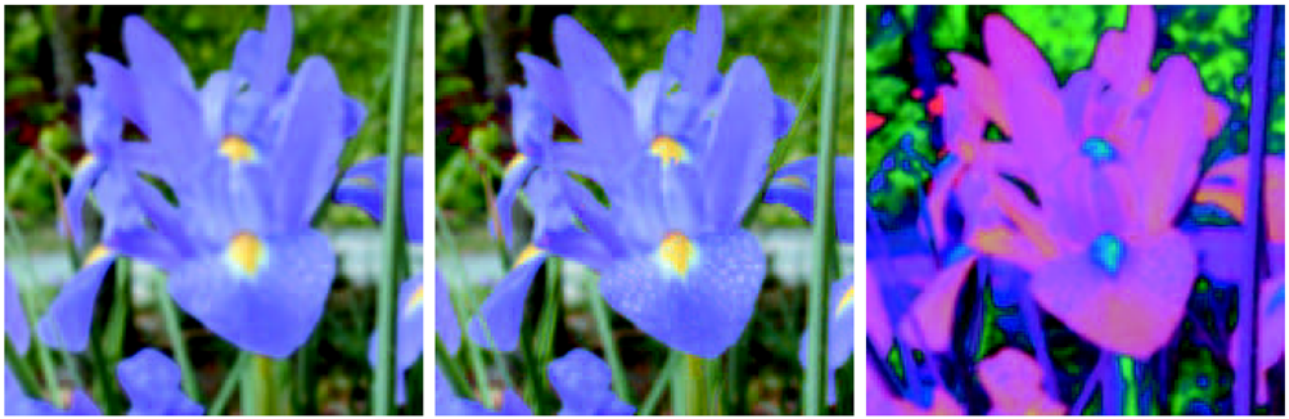
a b
c d

FIGURE 6.19
(a) RGB image; (b) through (d) are the red, green and blue component images, respectively.

Spatial Filtering of Color Images

a b c

FIGURE 6.20 From left to right: hue, saturation, and intensity components of Fig. 6.19(a).



a b c

FIGURE 6.21 (a) Smoothed RGB image obtained by smoothing the R, G, and B image planes separately. (b) Result of smoothing only the intensity component of the HSI equivalent image. (c) Result of smoothing all three HSI components equally.

Spatial Filtering of Color Images

- Color image sharpening

$$\nabla^2[c(x,y)] = \begin{bmatrix} \nabla^2 R(x,y) \\ \nabla^2 G(x,y) \\ \nabla^2 B(x,y) \end{bmatrix}$$

  - fb=imread('Fig0619(a)(RGB_iris).tif');
  - lapmask=[1 1 1;1 -8 1;1 1 1];
  - fen=imsubtract(fb,imfilter(fb,lapmask,'replicate'));
  - imshow(fen);

Spatial Filtering of Color Images

a b

FIGURE 6.22
(a) Blurred image. (b) Image enhanced using the Laplacian, followed by contrast enhancement using function `ice`.

Working Directly in RGB Vector Space

- Color Edge Detection Using the Gradient

gradient

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

magnitude

$$|\nabla f| = mag(\nabla f) = [G_x^2 + G_y^2]^{1/2} = [(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2]^{1/2}$$

angle

$$\alpha(x, y) = tan^{-1}(\frac{G_x}{G_y})$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

a b c

FIGURE 6.23 (a) A small neighborhood. (b) and (c) Sobel masks used to compute the gradient in the x (vertical) and y (horizontal) directions, respectively, with respect to the center point of the neighborhood.

Working Directly in RGB Vector Space

- r,g and b are the *unit vectors* along the R,G and B axis.

$$u = \frac{\partial R}{\partial x}r + \frac{\partial G}{\partial x}g + \frac{\partial B}{\partial x}b$$

and

$$v = \frac{\partial R}{\partial y}r + \frac{\partial G}{\partial y}g + \frac{\partial B}{\partial y}b$$

- Quantities

$$g_{xx} = u \cdot u = u^T u = |\frac{\partial R}{\partial x}|^2 + |\frac{\partial G}{\partial x}|^2 + |\frac{\partial B}{\partial x}|^2$$

$$g_{yy} = v \cdot v = v^T v = |\frac{\partial R}{\partial y}|^2 + |\frac{\partial G}{\partial y}|^2 + |\frac{\partial B}{\partial y}|^2$$

$$g_{xy} = u \cdot v = u^T v = |\frac{\partial R}{\partial x}\frac{\partial R}{\partial y}| + |\frac{\partial G}{\partial x}\frac{\partial G}{\partial y}| + |\frac{\partial B}{\partial x}\frac{\partial B}{\partial y}|$$

Working Directly in RGB Vector Space

- Angle=the direction of max rate of change of c(x,y)

$$\theta(x,y) = \frac{1}{2} tan^{-1}[\frac{2g_{xy}}{g_{xx} - g_{yy}}]$$

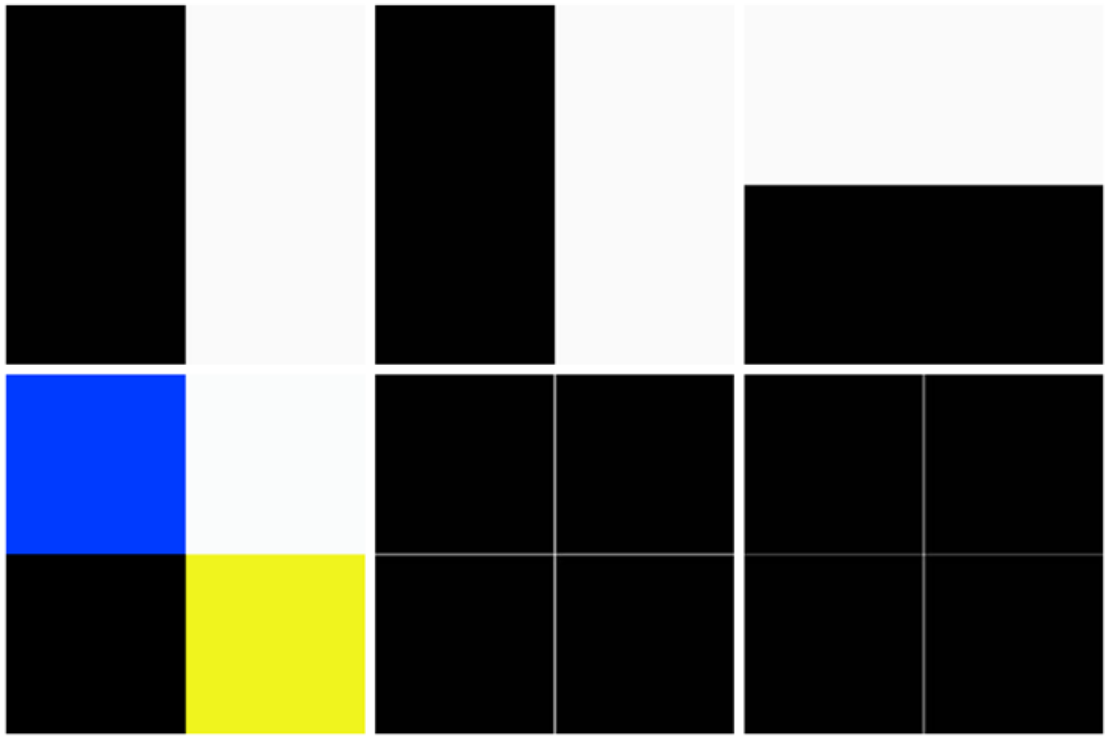
- The value of the rate of the change

$$F_\theta(x,y) = \{\frac{1}{2}[(g_{xx} + g_{yy}) + (g_{xx} - g_{yy})cos2\theta + 2g_{xy}sin2\theta]\}^{1/2}$$

Working Directly in RGB Vector Space

- The IPT for color gradient for RGB images

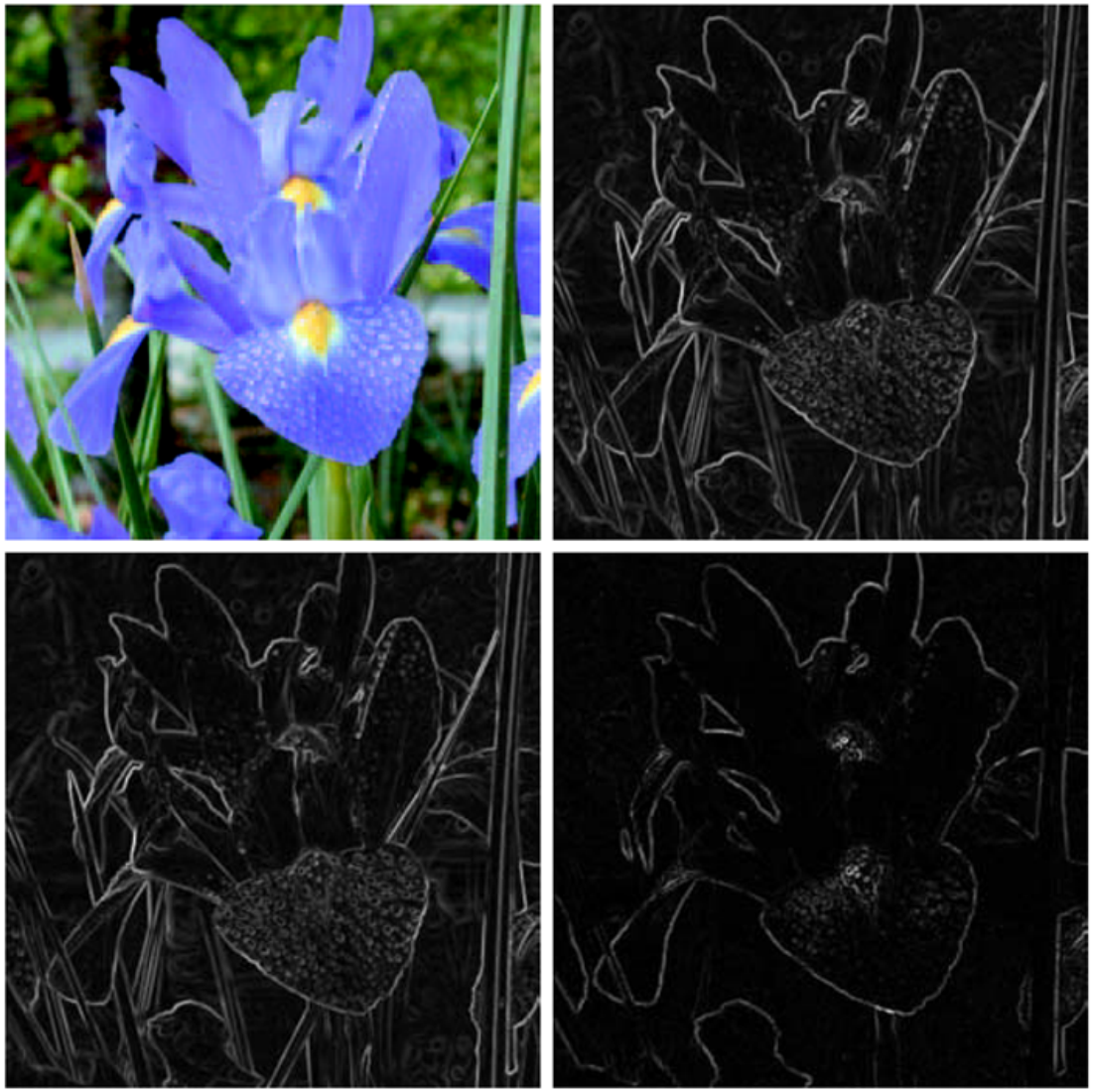
  [VG,A,PPG]=colorgrad(f,T)



a b c
d e f

FIGURE 6.24 (a) through (c) RGB component images (black is 0 and white is 255). (d) Corresponding color image. (e) Gradient computed directly in RGB vector space. (f) Composite gradient obtained by computing the 2-D gradient of each RGB component image separately and adding the results.

Working Directly in RGB Vector Space



a b
c d

FIGURE 6.25
(a) RGB image.
(b) Gradient computed in RGB vector space.
(c) Gradient computed as in Fig. 6.24(f).
(d) Absolute difference between (b) and (c), scaled to the range [0, 1].

Working Directly in RGB Vector Space

- Image Segmentation in RGB Vector Space

$$
\begin{aligned}
D(z,m) &= \|z-m\| \\
&= [(z-m)'(z-m)]^{1/2} \\
&= [(z_R-m_R)^2+(z_G-m_G)^2+(z_B-m_B)^2]^{1/2}
\end{aligned}
$$

- A useful generalization of the preceding equation is a distance measure of the form.

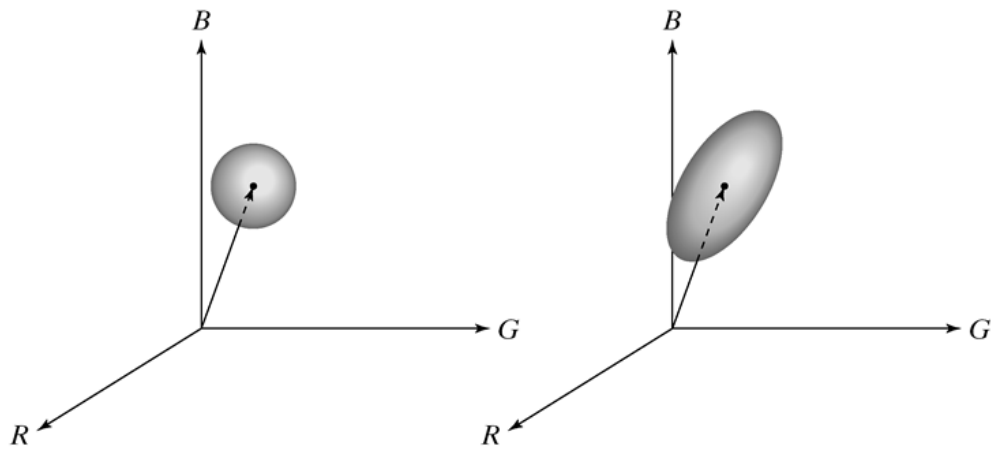
$$D(z,m) = [(z-m)'C^{-1}(z-m)]^{1/2}$$

The matrix C can be defined any format(if you like)

Working Directly in RGB Vector Space

- IPT for segmentaion

  S=colorseg(method,f,T,parameters)



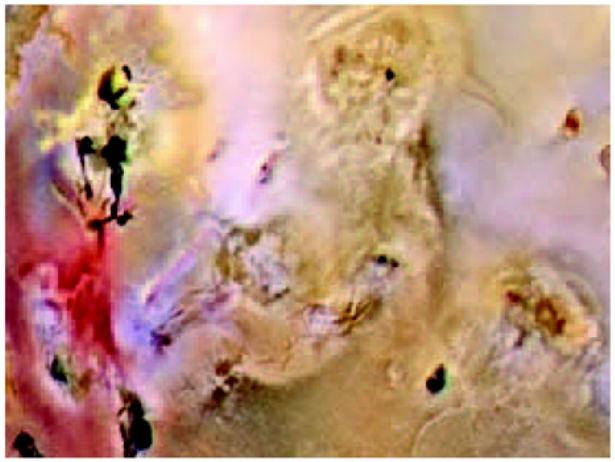
a b

FIGURE 6.26 Two approaches for enclosing data in RGB vector space for the purpose of segmentation.

Working Directly in RGB Vector Space

- >> f=imread('Fig0627(a)(jupitermoon_original).tif');
- >> mask=roipoly(f);
- >> imshow(mask);
- >> red=immultiply(mask,f(:,:,1));
- >> green=immultiply(mask,f(:,:,2));
- >> blue=immultiply(mask,f(:,:,3));
- >> g=cat(3,red,green,blue);
- >> imshow(g);

Working Directly in RGB Vector Space



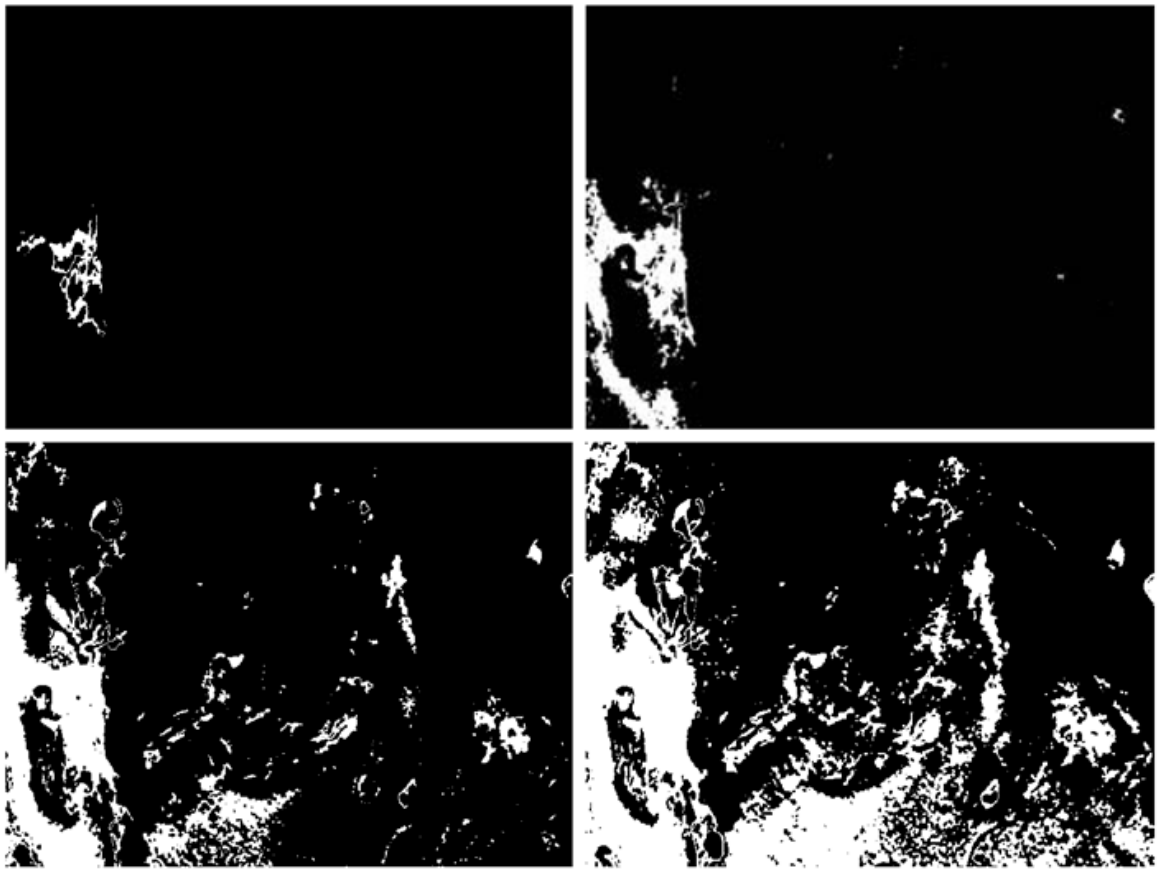
a b

FIGURE 6.27 (a) Pseudocolor of the surface of Jupiter's Moon Io. (b) Region of interest extracted interactively using function `roipoly`. (Original image courtesy of NASA.)

Working Directly in RGB Vector Space

- >> [M,N,K]=size(g);
- >> I=reshape(g,M*N,3);
- >> idx=find(mask);
- >> I=double(I(idx,1:3));
- >> [C,m]=covmatrix(I);
- >> d=diag(C);
- >> sd=sqrt(d)';
- >> e25=colorseg('euclidean',f,25,m);

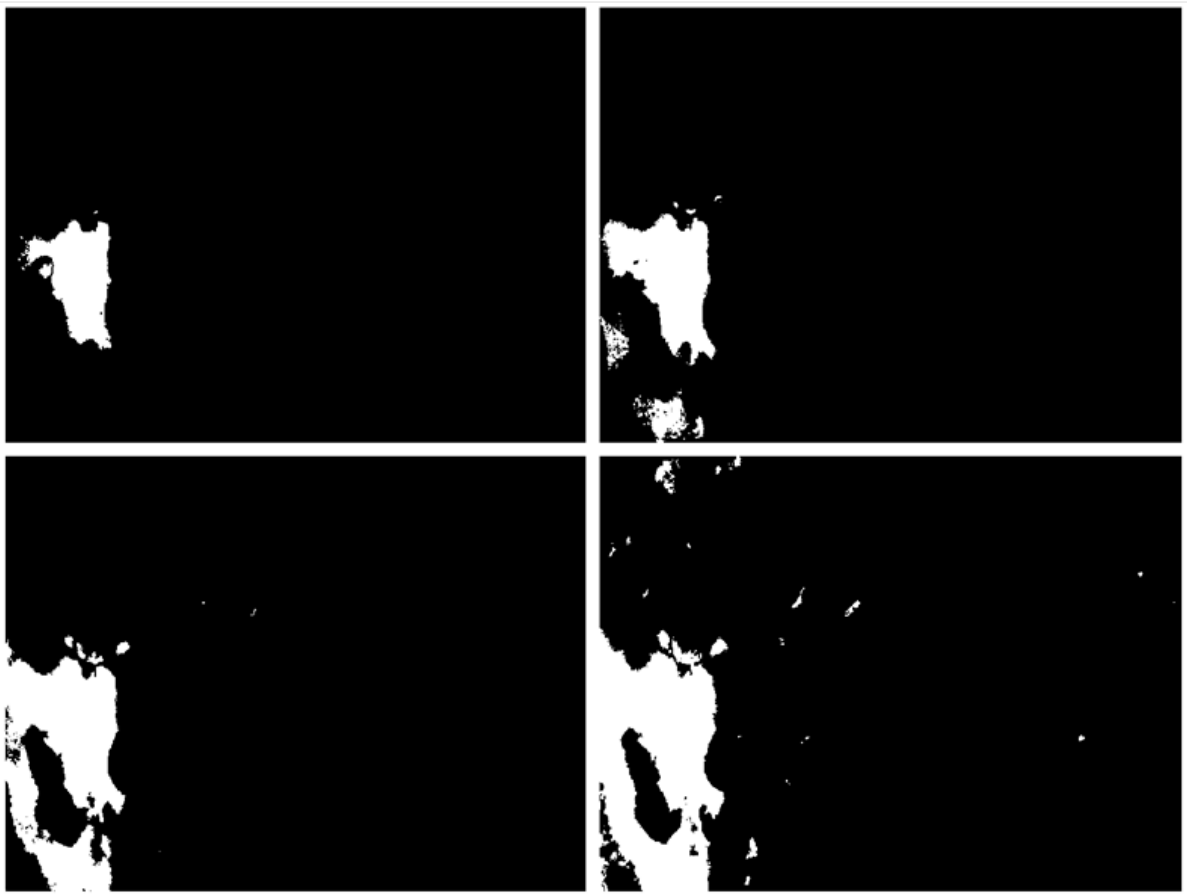
Working Directly in RGB Vector Space



a b
c d

FIGURE 6.28 (a) through (d) Segmentation of Fig. 6.27(a) using option `'euclidean'` in function `colorseg` with $T = 25, 50, 75,$ and 100, respectively.

Working Directly in RGB Vector Space



a b
c d

FIGURE 6.29 (a) through (d) Segmentation of Fig. 6.27(a) using option `'mahalanobis'` in function `colorseg` with $T = 25, 50, 75,$ and 100, respectively. Compare with Fig. 6.28.