CHAPTER

IMAGE PROCESSING BASICS 13

CHAPTER OUTLINE

13.1 Image Processing Notation and Data Formats 650
13.1.1 8-Bit Gray Level Images 650
13.1.2 24-Bit Color Images 651
13.1.3 8-Bit Color Images 652
13.1.4 Intensity Images 653
13.1.5 RGB Components and Grayscale Conversion 654
13.1.6 MATLAB Functions for Format Conversion 656
13.2 Image Histogram and Equalization 658
13.2.1 Grayscale Histogram and Equalization 658
13.2.2 24-Bit Color Image Equalization 663
13.2.3 8-Bit Indexed Color Image Equalization 664
13.2.4 MATLAB Functions for Equalization 665
13.3 Image Level Adjustment and Contrast 669
13.3.1 Linear Level Adjustment 669
13.3.2 Adjusting the Level for Display 670
13.3.3 MATLAB Functions for Image Level Adjustment 672
13.4 Image Filtering Enhancement 673
13.4.1 Lowpass Noise Filtering 673
13.4.2 Median Filtering 677
13.4.3 Edge Detection 679
13.4.4 MATLAB Functions for Image Filtering 682
13.5 Image Pseudo-Color Generation and Detection 684
13.6 Image Spectra 687
13.7 Image Compression by Discrete Cosine Transform 691
13.7.1 Two-Dimensional Discrete Cosine Transform 692
13.7.2 Two-Dimensional JPEG Grayscale Image Compression Example 694
13.7.3 JPEG Color Image Compression 697
13.7.4 Image Compression Using Wavelet Transform Coding 701
13.8 Creating a Video Sequence by Mixing two Images 708
13.9 Video Signal Basics 709
13.9.1 Analog Video 709
13.9.2 Digital Video 714

Digital Signal Processing. https://doi.org/10.1016/B978-0-12-815071-9.00013-0

13.10 Motion Estimation in Video 716
13.11 Summary 719
13.12 Problems 720

13.1 IMAGE PROCESSING NOTATION AND DATA FORMATS

A digital image is picture information in a digital form. The image could be filtered to remove noise to obtain enhancement. It can also be transformed to extract features for pattern recognition. The image can be compressed for storage and retrieval as well as transmitted via a computer network or a communication system.

The digital image consists of pixels. The position of each pixel is specified in terms of an index for the number of columns and another for the number of rows. Fig. 13.1 shows that a pixel $p(2,8)$ has a level of 86 and is located in the second row, eighth column. We express it in notation as

$$p(2,8) = 86. \tag{13.1}$$

The number of pixels in the presentation of a digital image is referred to *spatial resolution*, which relates to the image quality. The higher the spatial resolution, the better quality the image has. The resolution can be fairly high, for instance, as high as 1600×1200 (1,920,000 pixels = 1.92 megapixels), or as low as 320×200 (64,000 pixels = 64 kpixels). In notation, the number to the left of the multiplication symbol represents the width, and that to the right of the symbol represents the height in terms of pixels. Image quality also depends on the numbers of bits used in encoding each pixel level, which is discussed in the next section.

13.1.1 8-BIT GRAY LEVEL IMAGES

If a pixel is encoded on a gray scale from 0 to 255 (256 values), where 0 = black and 255 = white, the numbers in between represent levels of gray forming a *grayscale image*. For a 640×480 8-bit image, 307.2 kbytes are required for storage. Fig. 13.2 shows a grayscale image format. As shown in Fig. 13.2, the pixel indicated in the box has an 8-bit value of 25.

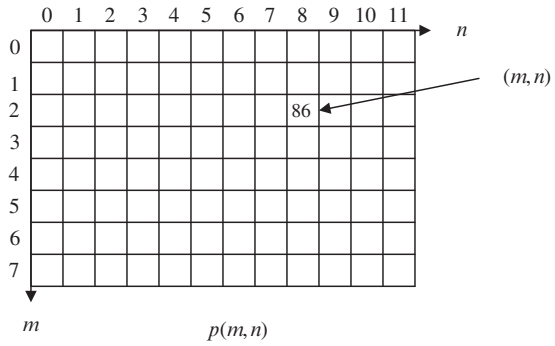


FIG. 13.1

Image pixel notation.

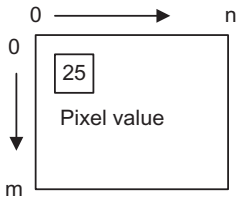


FIG. 13.2

Grayscale image format.

FIG. 13.3

Grayscale image (8-bit 320 × 240).

The image of a cruise ship with a spatial resolution of 320 × 240 in an 8-bit grayscale level is shown in Fig. 13.3. The tagged image file format (TIFF) is the common encoding file format, which is described in the textbook by Li et al. (2014).

13.1.2 24-BIT COLOR IMAGES

In a 24-bit color image representation, each pixel is recoded with the red, green, and blue (RGB) components (RGB channels). With each component value encoded in 8 bits, resulting in 24 bits in total, we achieve a full color RGB image. With such an image, we can have $2^{24} = 16.777216 \times 10^6$ different colors. A 640 × 480 24-bit color image requires 921.6 kbytes for storage. Fig 13.4 shows the format for the 24-bit color image where the indicated pixel has 8-bit RGB components.

Fig. 13.5 shows a 24-bit color image of the Grand Canyon, along with its grayscale displays for the 8-bit RGB component images. The common encoding file format is TIFF.

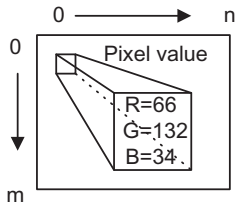


FIG. 13.4

The 24-bit color image format.

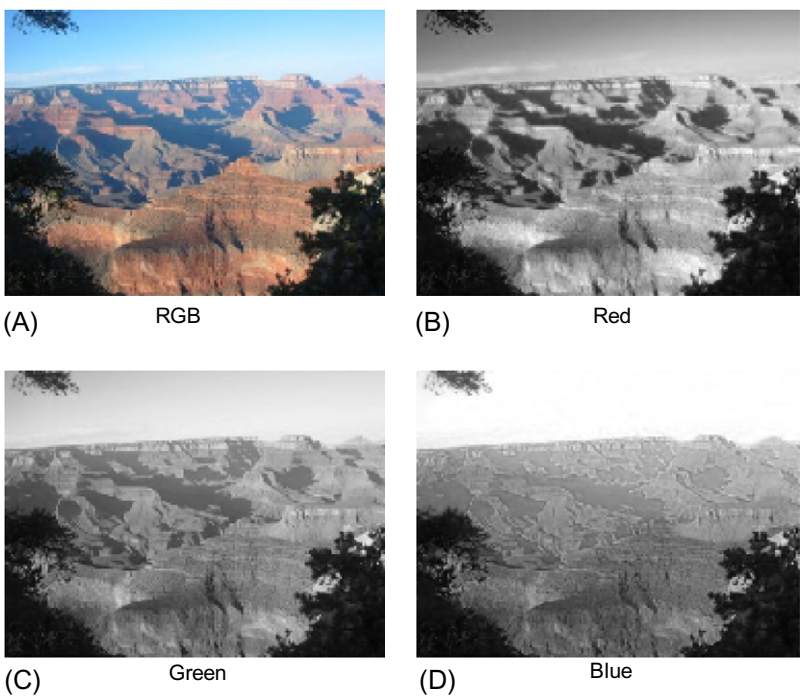


FIG. 13.5

The 24-bit color image and its respective RGB components.

13.1.3 8-BIT COLOR IMAGES

The 8-bit color image is also a popular image format. Its pixel value is a color index that points to a color look-up table containing RGB components. We call this a *color indexed image*, and its format is shown in Fig. 13.6. As an example in the figure, the color indexed image has a pixel index value of 5, which is the index for the entry of the color table, called the *color map*. At the index of location 5, there are three color components with RGB values of 66, 132, and 34, respectively. Each color component is encoded in 8 bits. There are only 256 different colors in the image. A 640 × 480 8-bit color image

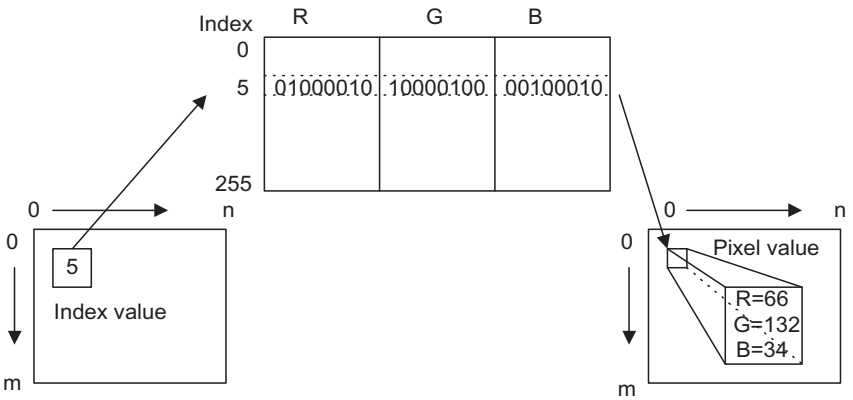


FIG. 13.6

The 8-bit color indexed image format.

FIG. 13.7

The 8-bit color indexed image (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

requires memory of 307.2 kbytes for data storage and $3 \times 256 = 768$ bytes for color map storage. The 8-bit color image for the cruise ship shown in Fig. 13.3 is displayed in Fig. 13.7. The common encoding file format is the graphics interchange format (GIF), which described in the textbook by Li et al. (2014).

13.1.4 INTENSITY IMAGES

As we noted in the first section, the grayscale image uses a pixel value ranging from 0 to 255 to present luminance, or the light intensity. A pixel value of 0 designates black, and a value 255 encodes for white.

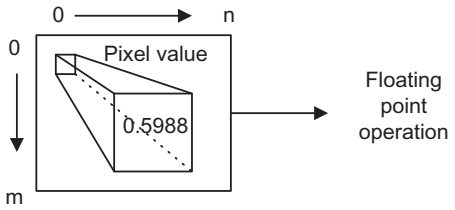


FIG. 13.8

The grayscale intensity image format.

In some processing environment such as MATLAB, floating-point operations are used. The grayscale intensity image has the intensity value that is normalized to the range from 0 to 1.0, where 0 represents black and 1 represents white. We often change the pixel value to the normalized range to get the grayscale intensity image before processing it, and then scale it back to the standard 8-bit range after processing for display. With the intensity image in the floating-point format, the digital filter implementation can be easily applied. Fig. 13.8 shows the format of the grayscale intensity image, where the indicated pixel shows the intensity value of 0.5988.

13.1.5 RGB COMPONENTS AND GRAYSCALE CONVERSION

In some applications, we need to convert a color image to a grayscale image so that storage space can be saved. As an example, fingerprint images are stored in the grayscale format in a database system. In color image compression as another example, the transformation converts the RGB color space to YIQ color space (Li et al., 2014; Rabbani and Jones, 1991), where Y is the luminance (Y) channel representing light intensity while the I (in-space) and Q (quadrature) chrominance channels represent color details.

The luminance $Y(m,n)$ carries grayscale information with most of the signal energy (as much as 93%), and the chrominance channels $I(m,n)$ and $Q(m,n)$ carry color information with much less energy (as little as 7%). The transformation in terms of the standard matrix notion is given by

$$\begin{bmatrix} Y(m,n) \\ I(m,n) \\ Q(m,n) \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R(m,n) \\ G(m,n) \\ B(m,n) \end{bmatrix}. \tag{13.2}$$

As an example of data compression, after transformation, we can encode $Y(m,n)$ with a higher resolution using a larger number of bits, since it contains most of the signal energy, while we encode chrominance channels $I(m,n)$ and $Q(m,n)$ with less resolution using a smaller number of bits. Inverse transformation can be solved as

$$\begin{bmatrix} R(m,n) \\ G(m,n) \\ B(m,n) \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y(m,n) \\ I(m,n) \\ Q(m,n) \end{bmatrix}. \tag{13.3}$$

To obtain the grayscale image, we simply convert each RGB pixel to the YIQ pixel; and then keep its luminance channel and discard IQ channel chrominance. The conversion formula is hence given by

$$Y(m,n) = 0.299 \times R(m,n) + 0.587 \times G(m,n) + 0.114 \times B(m,n). \tag{13.4}$$

Note that $Y(m,n)$, $I(m,n)$, and $Q(m,n)$ can be matrices that represents the luminance image two color component images, respectively. Similarly, $R(m,n)$, $G(m,n)$, and $B(m,n)$ can be matrices for the RGB component images.

EXAMPLE 13.1

Given a pixel in an RGB image as follows:

$$R = 200,\ G = 10,\ B = 100,$$

convert the pixel values to the YIQ values.

Solution:

Applying Eq. (13.2), it follows that

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} 200 \\ 10 \\ 100 \end{bmatrix}.$$

Carrying out the matrix operations leads

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 \times 200 & 0.587 \times 10 & 0.114 \times 100 \\ 0.596 \times 200 & -0.274 \times 10 & -0.322 \times 100 \\ 0.212 \times 200 & -0.523 \times 10 & 0.311 \times 100 \end{bmatrix} = \begin{bmatrix} 77.07 \\ 84.26 \\ 68.27 \end{bmatrix}.$$

Round the values to integer, we have

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = round \begin{bmatrix} 77.07 \\ 84.26 \\ 68.27 \end{bmatrix} = \begin{bmatrix} 77 \\ 84 \\ 68 \end{bmatrix}.$$

Now let us study the following example to convert the YIQ values back to the RGB values.

EXAMPLE 13.2

Given a pixel of an image in the YIQ color format as follows:

$$Y = 77,\ I = 84,\ Q = 68,$$

convert the pixel values back to the RGB values.

Solution:

Applying Eq. (13.3) yields

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} 77 \\ 84 \\ 68 \end{bmatrix} = \begin{bmatrix} 199.53 \\ 10.16 \\ 99.90 \end{bmatrix}.$$

After rounding, it follows that

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = round \begin{bmatrix} 199.53 \\ 10.16 \\ 99.9 \end{bmatrix} = \begin{bmatrix} 200 \\ 10 \\ 100 \end{bmatrix}.$$

EXAMPLE 13.3

Given the following 2×2 RGB image,

$$R = \begin{bmatrix} 100 & 50 \\ 200 & 150 \end{bmatrix} G = \begin{bmatrix} 10 & 25 \\ 20 & 50 \end{bmatrix} B = \begin{bmatrix} 10 & 5 \\ 20 & 15 \end{bmatrix},$$

convert the RGB color image into a grayscale image.

Solution:

Since only Y components are kept in the grayscale image, we apply Eq. (13.4) to each pixel in the 2×2 image and round the results to integers as follows:

$$Y = 0.299 \times \begin{bmatrix} 100 & 50 \\ 200 & 150 \end{bmatrix} + 0.587 \times \begin{bmatrix} 10 & 25 \\ 20 & 50 \end{bmatrix} + 0.114 \times \begin{bmatrix} 10 & 5 \\ 20 & 15 \end{bmatrix} = \begin{bmatrix} 37 & 30 \\ 74 & 76 \end{bmatrix}.$$

Fig. 13.9 shows that the grayscale image converted from the 24-bit color image in Fig. 13.5 using the RGB-to-YIQ transformation, where only the luminance information is retained.

13.1.6 MATLAB FUNCTIONS FOR FORMAT CONVERSION

The following list summarizes MATLAB functions for image format conversion:

imread = read image data file with the specified format

X = 8-bit grayscale image, 8-bit indexed image, or 24-bit RGB color image.

map = color map table for the indexed image (256 entries)

imshow(X,map) = 8-bit image display

imshow(X) = 24-bit RGB color image display if image X is in a 24-bit RGB color format; grayscale image display if image X is in an 8-bit grayscale format.

FIG. 13.9

Grayscale image converted from the 24-bit color image in Fig. 13.5 using RGB- to-YIQ transformation.

ind2gray = 8-bit indexed color image to 8-bit grayscale image conversion
ind2rgb = 8-bit indexed color image to 24-bit RGB color image conversion.
rgb2ind = 24-bit RGB color image to 8-bit indexed color image conversion
rgb2gray = 24-bit RGB color image to 8-bit grayscale image conversion
im2double = 8-bit image to intensity image conversion
mat2gray = image data to intensity image conversion
im2uint8 = intensity image to 8-bit grayscale image conversion
Fig. 13.10 outlines the applications of image format conversions.

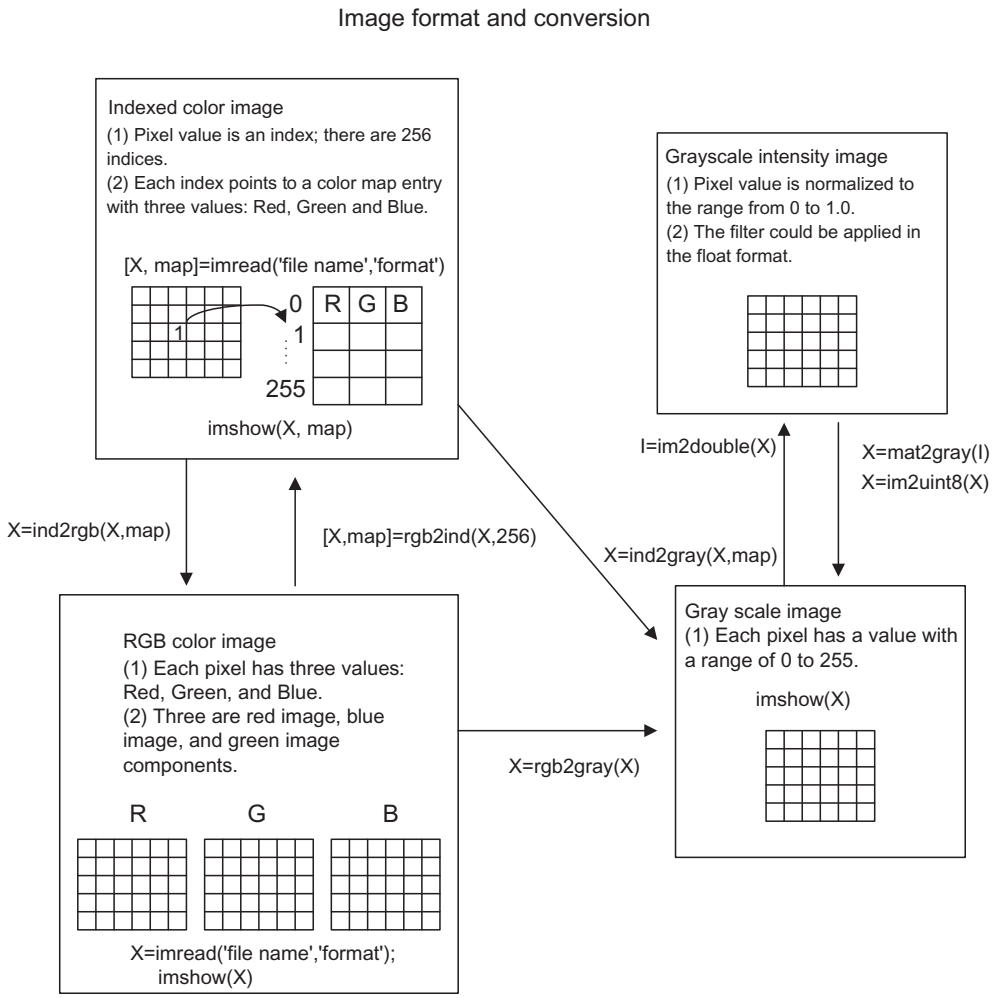


FIG. 13.10

Image format conversions.

13.2 IMAGE HISTOGRAM AND EQUALIZATION

An image histogram is a graph to show how many pixels are at each scale level or at each index for the indexed color image. The histogram contains information needed for image equalization, where the image pixels are stretched to give a reasonable contrast.

13.2.1 GRAYSCALE HISTOGRAM AND EQUALIZATION

We can obtain a histogram by plotting pixel value distribution over the full grayscale range.

EXAMPLE 13.4

Produce a histogram given the following image (a matrix filled with integers) with the grayscale value ranging from 0 to 7, that is, with each pixel encoded into 3 bits.

$$\begin{bmatrix} 0 & 1 & 2 & 2 & 6 \\ 2 & 1 & 1 & 2 & 1 \\ 1 & 3 & 4 & 3 & 3 \\ 0 & 2 & 5 & 1 & 1 \end{bmatrix}$$

Solution:

Since the image is encoded using 3 bits for each pixel, we have the pixel value ranging from 0 to 7. The count for each grayscale is listed in Table 13.1

Based on the grayscale distribution counts, the histogram is created as shown in Fig. 13.11.

As we can see, the image has pixels whose levels are more concentrated in the dark scale in this example.

Table 13.1 Pixel Counts Distribution

Pixel $p(m,n)$ Level	Number of Pixels
0	2
1	7
2	5
3	3
4	1
5	1
6	1
7	0

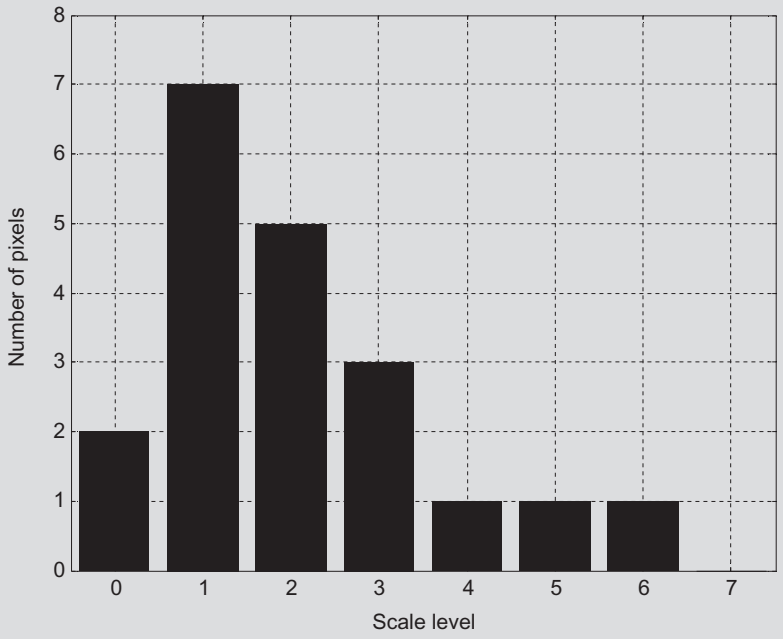


FIG. 13.11

Histogram in Example 13.4.

With the histogram, the equalization technique can be developed. Equalization stretches the scale range of the pixel levels to the full range to give an improved contrast for the given image. To do so, the equalized new pixel value is redefined as

$$p_{eq}(m,n) = \frac{\text{Number of pixels with scale level} \leq p(mn)}{\text{Total number of pixels}} \times (\text{maximum scale level}) \tag{13.5}$$

The new pixel value is reassigned using the value obtained by multiplying maximum scale level by the scaled ratio of the accumulative counts up to the current image pixel value over the total number of the pixels. Clearly, since the accumulate counts can range from 0 up to the total number of pixels, the equalized pixel value can vary from 0 to the maximum scale level. It is due to the accumulation procedure that the pixel values are spread over the whole range from 0 to the maximum scale level (255). Let us look at a simplified equalization example.

EXAMPLE 13.5

Given the following image (matrix filled with integers) with a grayscale value ranging from 0 to 7, that is, with each pixel is encoded in 3 bits,

$$\begin{bmatrix} 0 & 1 & 2 & 2 & 6 \\ 2 & 1 & 1 & 2 & 1 \\ 1 & 3 & 4 & 3 & 3 \\ 0 & 2 & 5 & 1 & 1 \end{bmatrix}$$

Continued

EXAMPLE 13.5—CONT'D

Perform equalization using the histogram in Example 13.4, and plot the histogram for the equalized image.

Solution:

Using the histogram result in Table 13.1, we can compute an accumulative count for each gray-scale level as shown in Table 13.2. The equalized pixel level using Eq. (13.5) is given in the last column.

To see how the old pixel level $p(m,n)=4$ is equalized to the new pixel level $p_{eq}(m,n)=6$, we apply Eq. (13.5):

$$p_{eq}(m,n)=round\left(\frac{18}{20}\times 7\right)=6.$$

The equalized image using Table 13.2 is finally obtained by replacing each old pixel value in the old image with its corresponding equalized new pixel value and given by

$$\begin{bmatrix} 1 & 3 & 5 & 5 & 7 \\ 5 & 3 & 3 & 5 & 3 \\ 3 & 6 & 6 & 6 & 6 \\ 1 & 5 & 7 & 3 & 3 \end{bmatrix}.$$

To see how the histogram is changed, we compute the pixel level counts according to the equalized image. The result is given in Table 13.3, and Fig. 13.12 shows the new histogram for the equalized image.

As we can see, the pixel levels in the equalized image are stretched to the larger scale levels. This technique works for underexposed images.

Table 13.2 Image Equalization in Example 13.5

Pixel $p(m,n)$ Level	Number of Pixels	Number of Pixels $\le p(m,n)$	Equalized Pixel Level
0	2	2	1
1	7	9	3
2	5	14	5
3	3	17	6
4	1	18	6
5	1	19	7
6	1	20	7
7	0	20	7

Table 13.3 Pixel Level Distribution Counts of the Equalized Imaged in Example 13.5

Pixel $p(m,n)$ Level	Number of Pixels
0	0
1	2
2	0
3	6
4	0
5	5
6	4
7	2

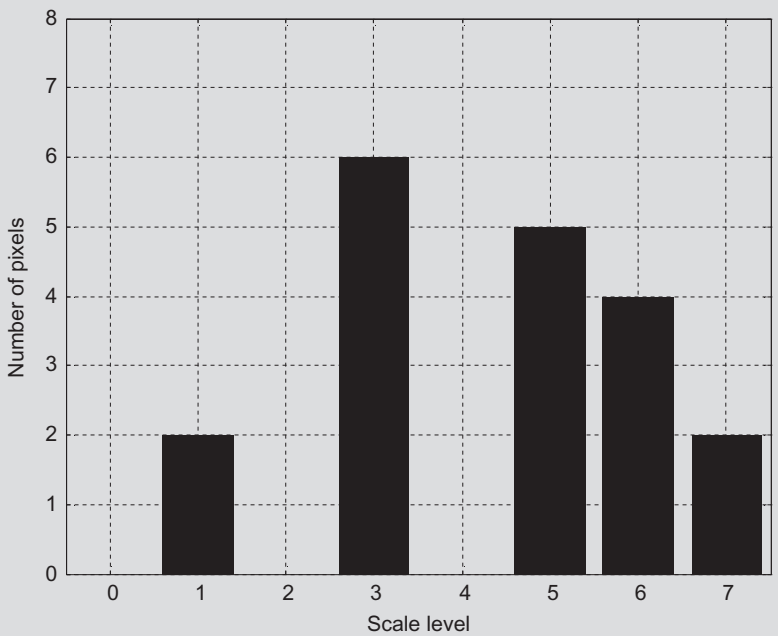


FIG. 13.12

Histogram for the equalized image in Example 13.5.

Next, we apply the image histogram equalization to enhance a biomedical image of a human neck in Fig. 13.13A, while Fig. 13.13B shows the original image histogram. We see that there are many pixel counts residing at the lower scales in the histogram. Hence, the image looks rather dark and may be underexposed.

Fig. 13.14A and B show the equalized grayscale image using the histogram method and its histogram, respectively. As shown in the histogram, the equalized pixels reside more on the larger scale, and hence the equalized image has improved contrast.

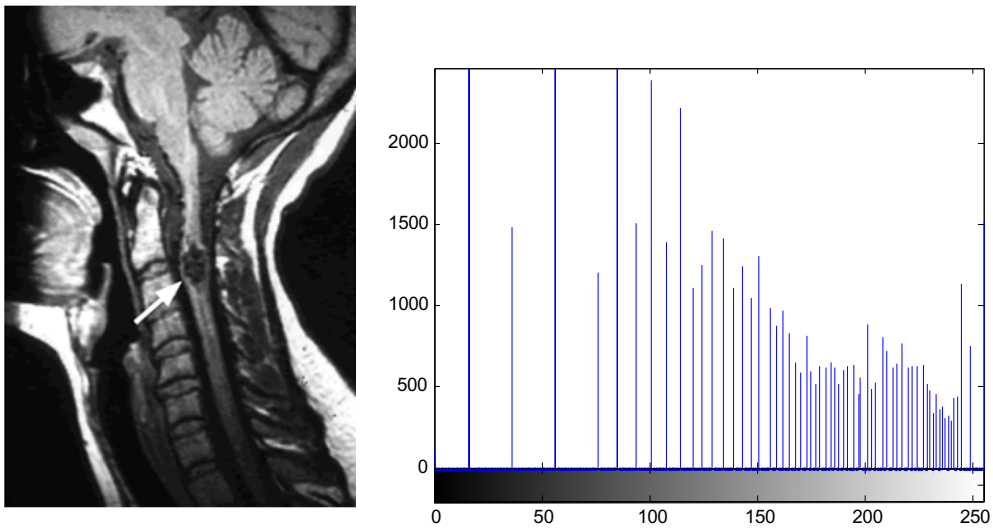


FIG. 13.13

(A) Original grayscale image, (B) Histogram for the original grayscale image.

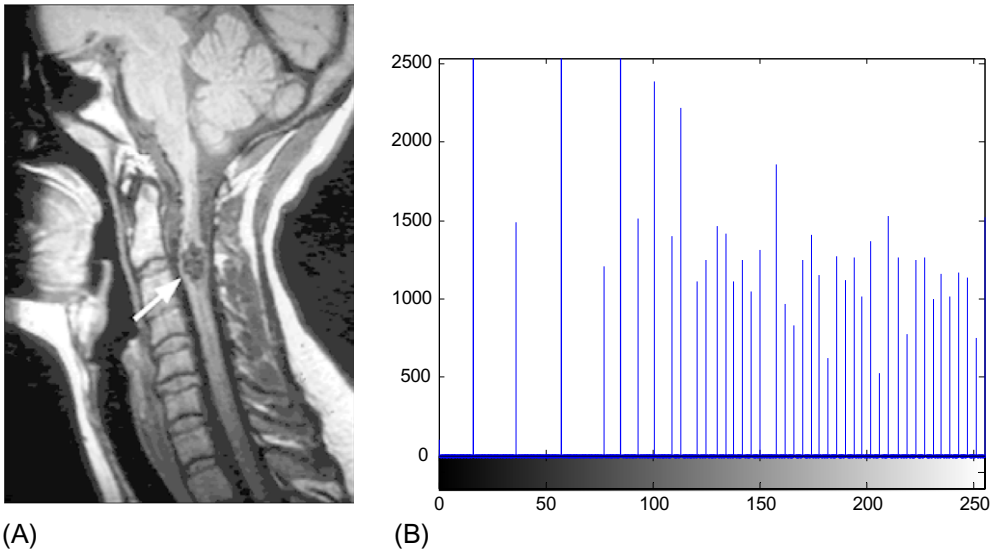


FIG. 13.14

(A) Grayscale equalized image. (B) Histogram for the grayscale equalized image.

13.2.2 24-BIT COLOR IMAGE EQUALIZATION

For equalizing the RGB image, we first transform RGB values to YIQ values since Y channel contains most of the signal energy, about 93%. Then Y channel is equalized just like the grayscale equalization to enhance the luminance. We leave the I and Q channels as they are, since these contain color information only and we do not equalize them. Next, we can repack the equalized Y channel back to the YIQ format. Finally, the YIQ values are transformed back to the RGB values for display. Fig. 13.15 shows the procedure.

Fig. 13.16A shows an original RGB color outdoors scene that is underexposed. Fig. 13.16B shows the equalized RGB image using the method for equalizing Y channel only. We can verify significant improvement with the equalized image showing much detailed information.

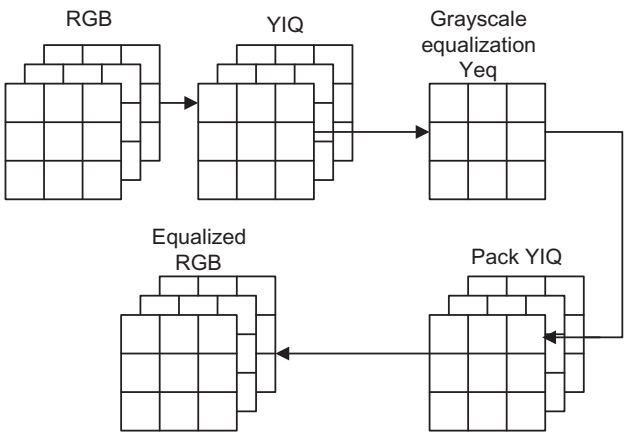


FIG. 13.15

Color image equalization.

(A) (B)

FIG. 13.16

(A) Original RGB color image (B) Equalized RGB color image (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

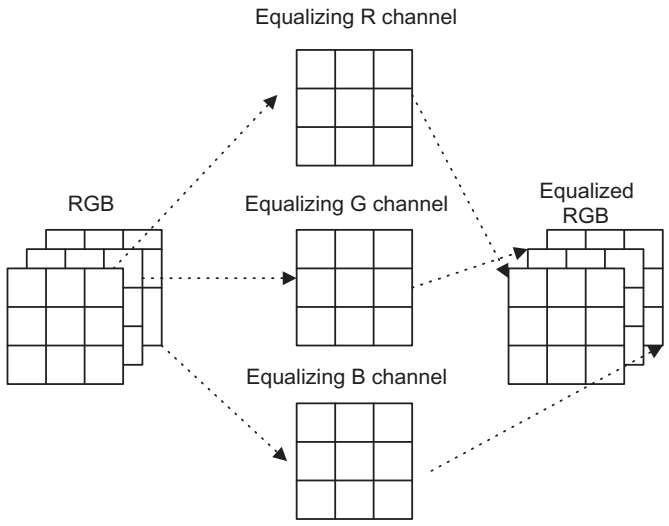


FIG. 13.17

Equalizing RGB channels.

We can also using the histogram equalization method to equalize each of the RGB channels, or their possible combinations. Fig. 13.17 illustrates such a procedure.

Some color effects can be observed. Equalization of the R channel only would make the image looks more red since the red pixel values are stretched out to the full range. Similar observations can be made for equalizing the G channel, or the B channel only. The equalized images for the R, G, and B channels, respectively, are shown in Fig. 13.18. The image from equalizing the R, G, and B channels simultaneously is shown in the upper left corner, which offers improved image contrast.

13.2.3 8-BIT INDEXED COLOR IMAGE EQUALIZATION

Equalization of the 8-bit color indexed image is more complicated. This is because the pixel value is the index for color map entries, and there are three RGB color components for each entry. We expect that after equalization, the index for each pixel will not change from its location on the color map table. Instead, the RGB components in the color map are equalized and changed. The procedure is described in the following and is shown in Fig. 13.19.

Step 1: The RGB color map is converted to the YIQ color map. Note that there are only 256 color table entries. Since the image contains the index values, which point to locations on the color table containing RGB components, it is natural to convert the RGB color table to the YIQ color table.

Step 2: The grayscale image is generated using the Y channel value, so that grayscale equalization can be performed.

Step 3: Grayscale equalization is executed.

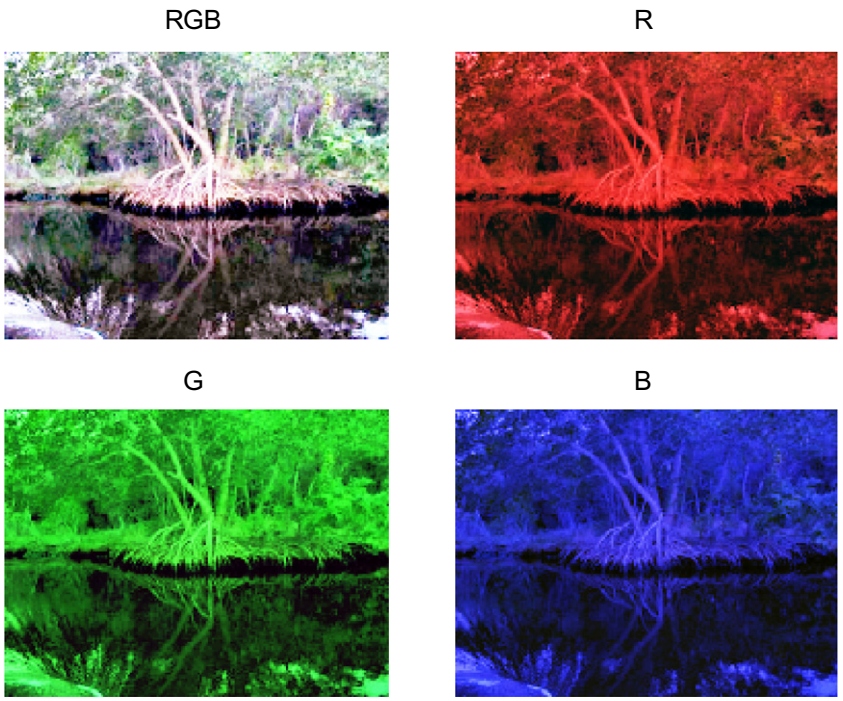


FIG. 13.18

Equalization effects for RGB channels. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Step 4: The equalized 256 Y values are divided by their corresponding old Y values to obtain the relative luminance scale factors.

Step 5: Finally, the RGB values are each scaled in the old RGB color table with the corresponding relative luminance scale factor and are normalized as new RGB channels in color table in the correct range. Then the new RGB color map is the output.

Note that original index values are not changed; only the color map content is.

Using the previous outdoors picture for the condition of underexposure, Fig. 13.20 shows the equalized indexed color image. We see that the equalized image displays much more detail.

13.2.4 MATLAB FUNCTIONS FOR EQUALIZATION

Fig. 13.21 lists MATLAB functions for performing equalization for the different image formats. The MATLAB functions are explained as follows:

histeq = grayscale histogram equalization, or 8-bit indexed color histogram equalization

imhis t = histogram display

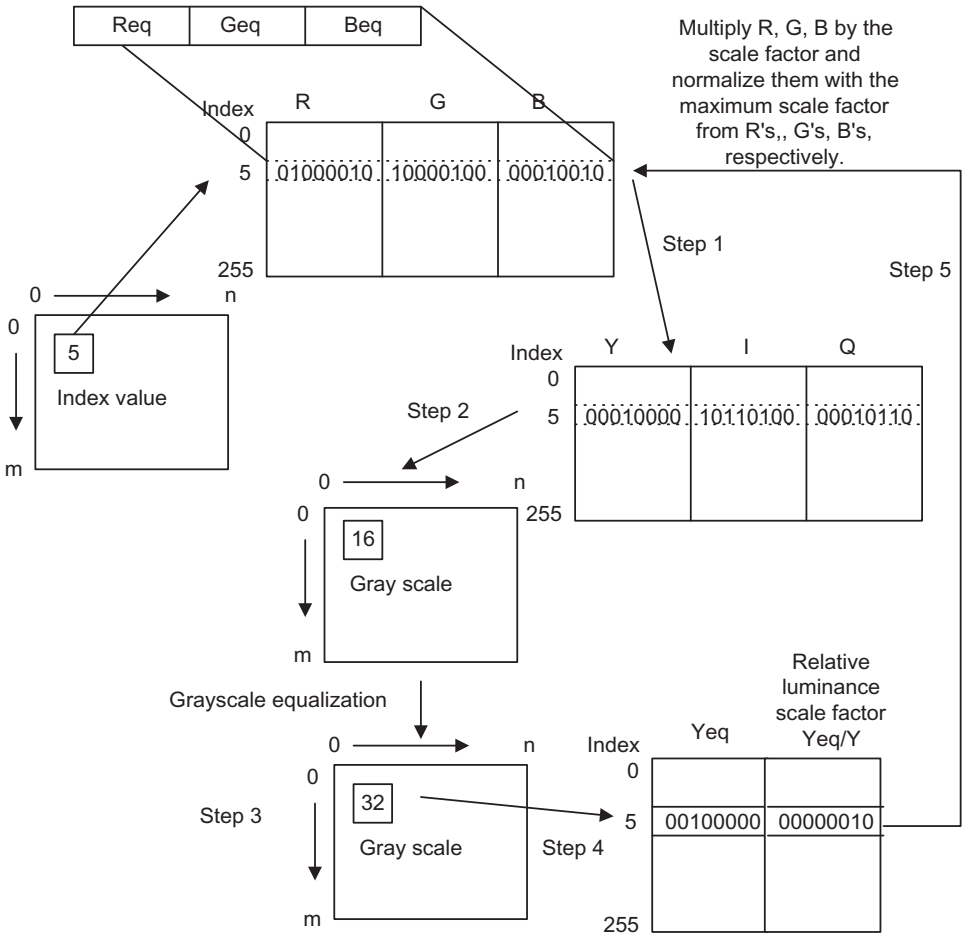


FIG. 13.19

Equalization of the 8-bit indexed color image.

FIG. 13.20

Equalized indexed the 8-bit color image. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

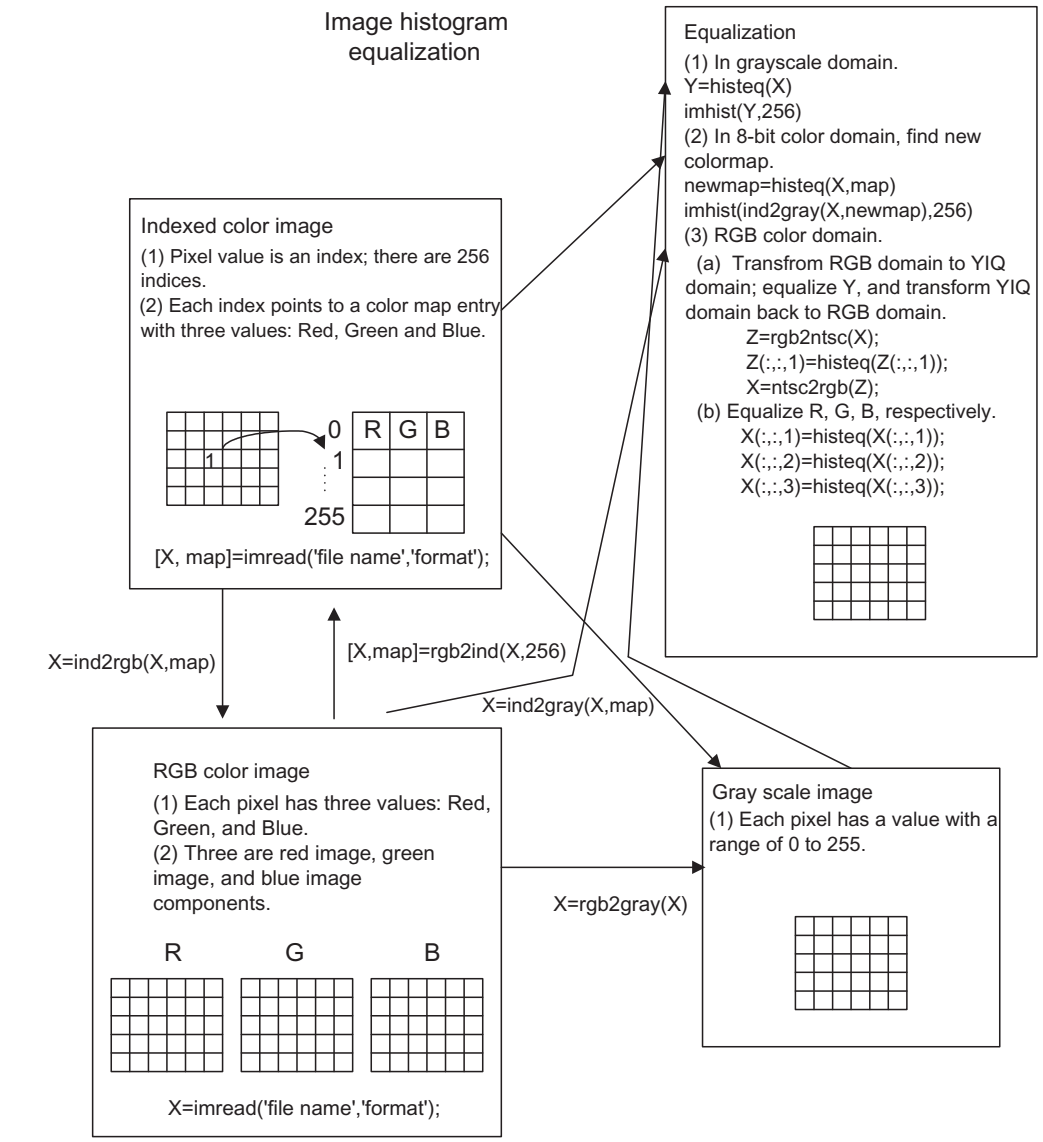


FIG. 13.21

MATLAB functions for image equalization.

rgb2ntsc = 24-bit RGB color image to 24-bit YIQ color image conversion
ntsc2rgb = 24-bit YIQ color image to 24-bit RGB color image conversion

Examples using the MATLAB functions for image format conversion and equalization are given in Program 13.1.

Program 13.1. Examples of image format conversion and equalization.

```
disp('Read the rgb image');
XX=imread('trees','JPEG');            % Provided by the instructor
figure, imshow(XX);
title('24-bit color')
disp('the grayscale image and histogram');
Y=rgb2gray(XX);                          %RGB to grayscale conversion
figure, subplot(1,2,1);imshow(Y);
title('original');subplot(1.2.2);imhist(Y, 256);

disp('Equalization in grayscale domain');
Y=histeq(Y);
figure, subplot(1,2,1); imshow(Y);
title('EQ in grayscale-domain'); subplot(1,2,2); imhist(Y, 256);

disp('Equalization of Y channel for RGB color image');
figure
subplot(1,2,1); imshow(XX);
title('EQ in RGB color');
subplot(1,2,2); imhist(rgb2gray(XX),256);

Z1=rgb2ntsc(XX);                 % Conversion from RGB to YIQ
Z1(:,:,1)=histeq(Z1(:,:,1));     %e Equalizing Y channel
ZZ=ntsc2rgb(Z1);                 %Conversion from YIQ to RGB

figure
subplot(1,2,1); imshow(ZZ);
title('EQ for Y channel for RGB color image');
subplot(1,2,2); imhist(im2uint8(rgb2gray(ZZ)),256);

ZZZ=XX;
ZZZ(:,:,1)=histeq(ZZZ(:,:,1)); %Equalizing R channel
ZZZ(:,:,2)=histeq(ZZZ(:,:,2)); %Equalizing G channel
ZZZ(:,:,3)=histeq(ZZZ(:,:,3)); %Equalizing B channel
figure
subplot(1,2,1); imshow(ZZZ);
title('EQ for RGB channels');
subplot(1,2,2); imhist(im2uint8(rgb2gray(ZZZ)),256);

disp('Equalization in 8-bit indexd color');
[Xind, map]=rgb2ind(XX, 256); % RGB to 8-bit index image conversion
newmap=histeq(Xind,map);
figure
subplot(1,2,1); imshow(Xind,newmap);
title('EQ in 8-bit indexed color');
subplot(1,2,2); imhist(ind2gray(Xind,newmap),256);
```

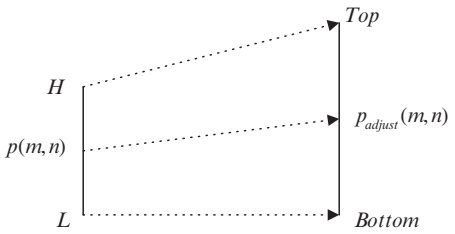


FIG. 13.22

Linear level adjustment.

13.3 IMAGE LEVEL ADJUSTMENT AND CONTRAST

Image level adjustment can be used to linearly stretch pixel level in an image to increase contrast and shift the pixel level to change viewing effects. Image level adjustment is also a requirement for modifying results from image filtering or other operations to an appropriate range for display. We study this technique in the following subsections.

13.3.1 LINEAR LEVEL ADJUSTMENT

Sometimes, if the pixel range in an image is small, we can adjust the image pixel level to make use of a full pixel range. Hence, contrast of the image is enhanced. Fig. 13.22 illustrates linear level adjustment.

The linear level adjustment is given by the following formula:

$$p_{adjust}(m,n) = Bottom + \frac{p(mn) - \mathrm{L}}{\mathrm{H} - \mathrm{L}} \times (\mathrm{Top} - \mathrm{Bottom}), \tag{13.6}$$

where $p(m,n)$ = original image pixel

$p_{adjust}(m,n)$ = desired image pixel

H = maximum pixel level in the original image

L = minimum pixel level in the original image

Top = maximum pixel level in the desired image

Bottom = minimum pixel level in the desired image

Besides adjusting image level to a full range, we can also apply the method to shift the image pixel levels up or down.

EXAMPLE 13.6

Given the following image (matrix filled with integers) with a grayscale ranging from 0 to 7, that is, with each pixel encoded in 3 bits,

$$\begin{bmatrix} 3 & 4 & 4 & 5 \\ 5 & 3 & 3 & 3 \\ 4 & 4 & 4 & 5 \\ 3 & 5 & 3 & 4 \end{bmatrix},$$

Continued

EXAMPLE 13.6—CONT'D

(a) Perform level adjustment to full range.
(b) Shift the level to the range from 3 to 7.
(c) Shift the level to the range from 0 to 3.

Solution:

(a) From the given image, we set the following for level adjustment to the full range:

$$\mathrm{H}=5,\ \mathrm{L}=3,\ \mathrm{Top}=2^3-1=7,\ \mathrm{Bottom}=0.$$

Applying Eq. (13.6) yields the second column in Table 13.4.

(b) For the shift-up operation, it follows that

$$\mathrm{H}=5,\ \mathrm{L}=3,\ \mathrm{Top}=7,\ \mathrm{Bottom}=3.$$

(c) For the shift-down operation, we set

$$\mathrm{H}=5,\ \mathrm{L}=3,\ \mathrm{Top}=3,\mathrm{Bottom}=0.$$

The results for (b) and (c) are listed in the third and fourth column, respectively, of Table 13.4. According to Table 13.4, we have three images:

$$\begin{bmatrix}0&4&4&7\\7&0&0&0\\4&4&4&7\\0&7&0&4\end{bmatrix}\quad\begin{bmatrix}3&5&5&7\\7&3&3&3\\5&5&5&7\\3&7&3&5\end{bmatrix}\quad\begin{bmatrix}0&2&2&3\\3&0&0&0\\2&2&2&3\\0&3&0&2\end{bmatrix}$$

Table 13.4 Image Adjustment Results in Example 13.6

Pixel $p(m,n)$ Level	Full Range	Range [3 7]	Range [0–3]
3	0	3	0
4	4	5	2
5	7	7	3

Next, applying the level adjustment for the neck image of Fig. 13.13A, we get results as shown in Fig. 13.23: the original image, the full range stretched image, the level shift-up image, and the level shift-down image. As we can see, the stretching operation increases image contrast while the shift-up operation lightens the image and the shift-down operation darkens the image.

13.3.2 ADJUSTING THE LEVEL FOR DISPLAY

When two 8-bit images are added together or undergo other mathematical operations, the sum of two pixel values could be as low as 0 and as high as 510. We can apply the linear adjustment to scale the range back to 0–255 for display. The following is the addition of two 8-bit images, which yields a sum that is out of the 8-bit range:

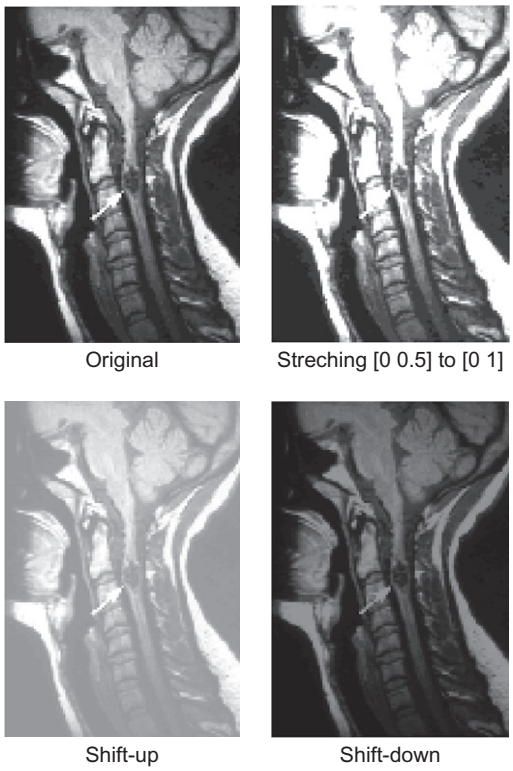


FIG. 13.23

Image level adjustment.

$$\begin{bmatrix} 30 & 25 & 5 & 170 \\ 70 & 210 & 250 & 30 \\ 225 & 125 & 50 & 70 \\ 28 & 100 & 30 & 50 \end{bmatrix} + \begin{bmatrix} 30 & 255 & 50 & 70 \\ 70 & 3 & 30 & 30 \\ 50 & 200 & 50 & 70 \\ 30 & 70 & 30 & 50 \end{bmatrix} = \begin{bmatrix} 60 & 280 & 55 & 240 \\ 140 & 213 & 280 & 60 \\ 275 & 325 & 100 & 140 \\ 58 & 179 & 60 & 100 \end{bmatrix}$$

To scale the combined image, modify Eq. (13.6) as

$$p_{scaled}(m,n) = \frac{p(mn) - \text{Minimum}}{\text{Maximum} - \text{Minimum}} \times (\text{Maximum scale level}). \tag{13.7}$$

Note that in the image to be scaled,

$$\text{Minimum} = 325$$

$$\text{Minimum} = 55$$

$$\text{Maixmum scale level} = 255,$$

we have after scaling:

$$\begin{bmatrix} 5 & 213 & 0 & 175 \\ 80 & 149 & 213 & 5 \\ 208 & 255 & 43 & 80 \\ 3 & 109 & 5 & 43 \end{bmatrix}.$$

13.3.3 MATLAB FUNCTIONS FOR IMAGE LEVEL ADJUSTMENT

Fig. 13.24 lists applications of the MTALAB level adjustment function, which is defined as:

J = imajust (I, [bottom level, top level], [adjusted bottom, adjusted top], gamma)

I = input intensity image

J = output intensity image

gamma = 1 (linear interpolation function as we discussed in the text)

0 < gamma < 1 lightens image; and gamma > 1 darkens image.

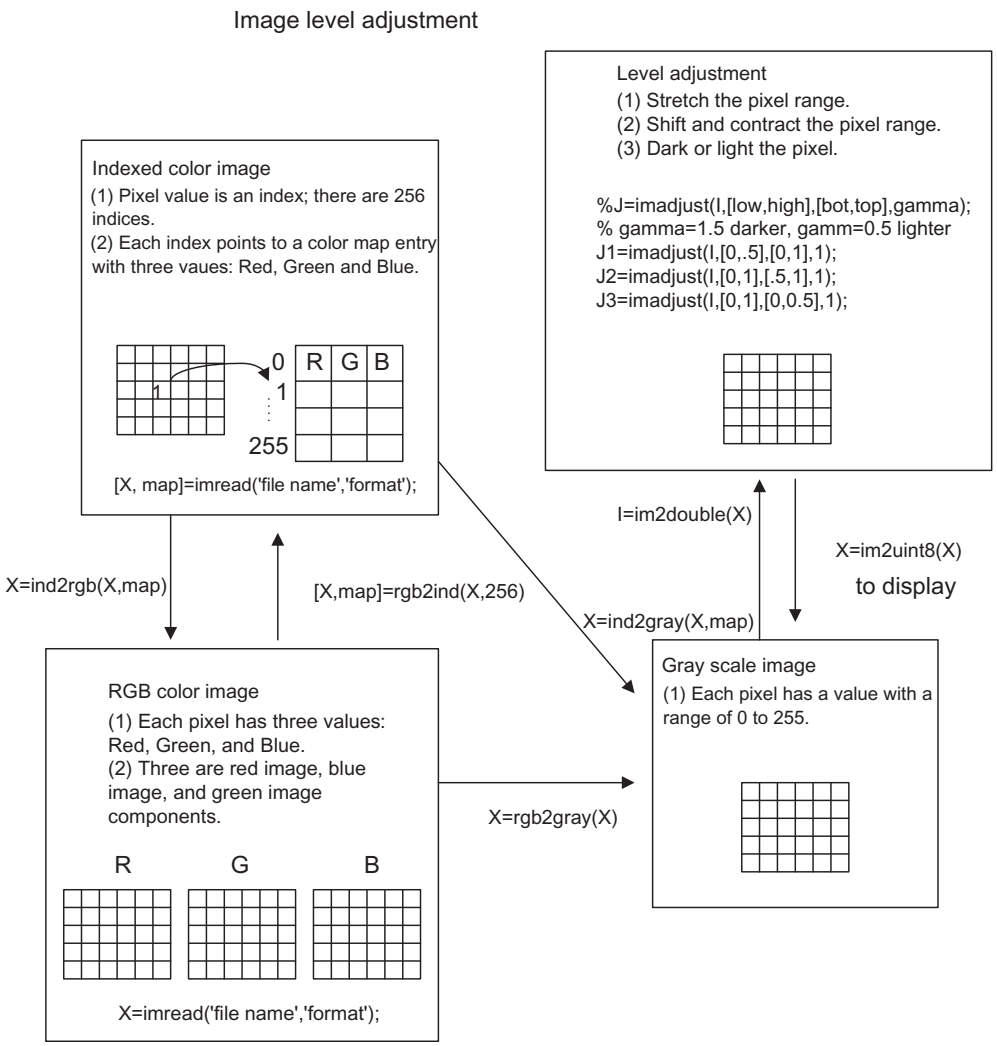


FIG. 13.24

MATLAB functions for image level adjustment.

13.4 IMAGE FILTERING ENHANCEMENT

As with one-dimensional (1D) digital signal processing, we can design a digital image filter such as lowpass, highpass, bandpass, and notch to process the image to obtain the desired effect. In this section, we discuss the most common ones: lowpass filters to remove the noise, median filters to remove impulse noise, and edge-detection filter to gain the boundaries of objects in images. More advanced treatment of this subject can be explored in the well-known text by Gonzalez and Wintz (1987).

13.4.1 LOWPASS NOISE FILTERING

One of the simplest lowpass filter is the average filter. The noisy image is filtered using the average convolution kernel with a size 3×3 block, 4×4 block, 8×8 block, and so on, in which the elements in the block have the same filter coefficients. The 3×3, 4×4, and 8×8 average kernels are as follows:

3×3 average kernel:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{13.8}$$

4×4 average kernel:

$$\frac{1}{16}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{13.9}$$

8×8 average kernel

$$\frac{1}{64}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{13.10}$$

Each of elements in the average kernel is 1 and the scale factor is the reciprocal of the total number of elements in the kernel. The convolution operates to modify each pixel in the image as follows. By passing the center of a convolution kernel through each pixel in the noisy image, we can sum each product of the kernel element and the corresponding image pixel value and multiply the sum by the scale factor to get the processed pixel. To understand the filter operation with the convolution kernel, let us study the following example.

EXAMPLE 13.7

Perform digital filtering on the noisy image using 2×2 and 3×3 convolutional average kernels, respectively, and compare the enhanced image with the original one given the following 8-bit grayscale original and corrupted (noisy) images.

$$4 \times 4 \text{ original image}: \begin{bmatrix} 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \end{bmatrix}$$

$$4 \times 4 \text{ corrupted image}: \begin{bmatrix} 99 & 107 & 113 & 96 \\ 92 & 116 & 84 & 107 \\ 103 & 93 & 86 & 108 \\ 87 & 109 & 106 & 107 \end{bmatrix}$$

(a) 2×2 average kernel: $\frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$.

(b) 3×3 average kernel: $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

Solution:

(a) In the following diagram, we pad edges with zeros in the last row and column before processing at the point where the first kernel and the last kernel are shown in the dotted-line boxes, respectively.

99	107	113	96	0
92	116	84	107	0
103	93	86	108	0
87	109	106	107	0
0	0	0	0	0

To process the first element, we know that the first kernel covers the image elements as $\begin{bmatrix} 99 & 107 \\ 92 & 116 \end{bmatrix}$. Summing each product of the kernel element and the corresponding image pixel value, multiplying a scale factor of ¼, and rounding the result, it follows that

$$\frac{1}{4}(99 \times 1 + 107 \times 1 + 92 \times 1 + 116 \times 1) = 103.5$$

$$round(103.5) = 104.$$

In the processing of the second element, the kernel covers $\begin{bmatrix} 107 & 113 \\ 116 & 84 \end{bmatrix}$. Similarly, we have

$$\frac{1}{4}(107 \times 1 + 113 \times 1 + 116 \times 1 + 84 \times 1) = 105$$

$$round(105) = 105.$$

The process continues for the rest of image pixels. To process the last element of the first row, 96, since the kernel covers only $\begin{bmatrix} 96 & 0 \\ 107 & 0 \end{bmatrix}$, we assume that the last two elements are zeros. Then:

$$\frac{1}{4}(96 \times 1 + 107 \times 1 + 0 \times 1 + 0 \times 1) = 50.75$$

$$round(50.75) = 51.$$

Finally, we yield the following filtered image:

$$\begin{bmatrix} 104 & 105 & 100 & 51 \\ 101 & 95 & 96 & 54 \\ 98 & 98 & 102 & 54 \\ 49 & 54 & 53 & 27 \end{bmatrix}.$$

As we know, due to zero padding for boundaries, the last-row and last-column values are in error. However, for a large image, these errors at boundaries can be neglected without affecting image quality. The first 3×3 elements in the processed image have values that are close to those of the original image. Hence, the image is enhanced.

(b) To use 3×3 kernel, we usually pad zeros surround the image as shown as follows:

0	0	0	0	0	0
0	99	107	113	96	0
0	92	116	84	107	0
0	103	93	86	108	0
0	87	109	106	107	0
0	0	0	0	0	0

Similar to (a), we finally achieve the filtered image as

$$\begin{bmatrix} 46 & 68 & 69 & 44 \\ 68 & 99 & 101 & 66 \\ 67 & 97 & 102 & 66 \\ 44 & 65 & 68 & 45 \end{bmatrix}$$

Notice that after neglecting the boundaries, the enhanced 2×2 image is given by

$$\begin{bmatrix} 99 & 101 \\ 97 & 102 \end{bmatrix}.$$

Fig. 13.25 shows the noisy image and enhanced images using the 3×3, 4×4, 8×8 average lowpass filter kernels, respectively. The average kernel removes noise. However, it also blurs the image. When using a large-sized kernel, the quality of the processed image becomes unacceptable.

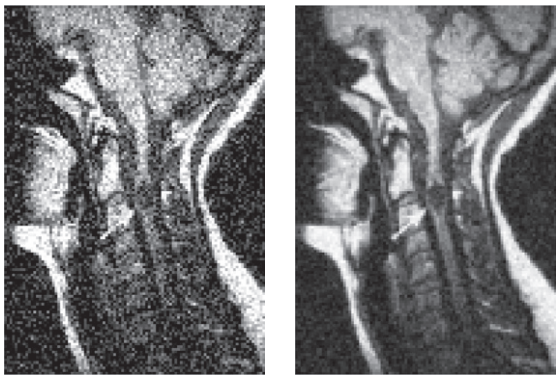


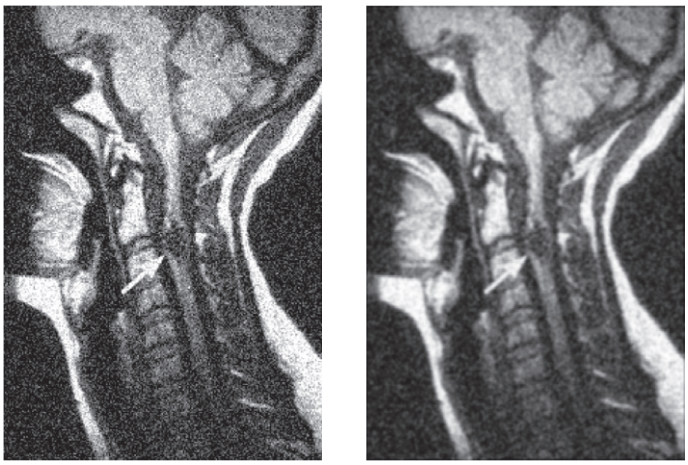
FIG. 13.25

Noise filtering using the lowpass average kernels.

The sophisticated large-size kernels are used for noise filtering. Although it is beyond the scope of the text, the Gaussian filter kernel with the standard deviation $\sigma = 0.9$, for instance, is given by the following:

$$\frac{1}{25}\begin{bmatrix} 0 & 2 & 4 & 2 & 0 \\ 2 & 15 & 27 & 15 & 2 \\ 4 & 27 & 50 & 27 & 4 \\ 2 & 15 & 27 & 15 & 2 \\ 0 & 2 & 4 & 2 & 0 \end{bmatrix}. \tag{13.11}$$

This kernel weighs the center pixel to be processed most and weighs less and less to the pixels away from the center pixel. In this way, the blurring effect can be reduced when filtering the noise. The plot of kernel values in the special domain looks like the bell shape. The steepness of shape is controlled by the standard deviation of the Gaussian distribution function, in which the larger the standard deviation, the flatter the kernel; hence, the more blurring effect will occur.



(A) (B)

FIG. 13.26

(A) Noisy image for a human neck. (B) Enhanced image using Gaussian lowpass filter.

Fig. 13.26A shows the noisy image, while Fig. 13.6B shows the enhanced image using the 5×5 Gaussian filter kernel. Clearly, the majority of the noise has been filtered, while the blurring effect is significantly reduced.

13.4.2 MEDIAN FILTERING

The median filter is the one type of nonlinear filters. It is very effective at removing impulse noise, the "salt and pepper" noise, in the image. The principle of the median filter is to replace the gray level of each pixel by the median of the gray levels in a neighborhood of the pixels, instead of using the average operation. For median filtering, we specify the kernel size, list the pixel values, covered by the kernel, and determine the median level. If the kernel covers an even number of pixels, the average of two median values is used. Before beginning median filtering, zeros must be padded around the row edge and the column edge. Hence, edge distortion is introduced at image boundary. Let us look at Example 13.8.

EXAMPLE 13.8

Given a 3×3 median filter kernel and the following 8-bit grayscale original and corrupted (noisy) images,

$$4 \times 4 \text{ original image}: \begin{bmatrix} 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \end{bmatrix}$$

$$4 \times 4 \text{ corrupted image by impulse noise}: \begin{bmatrix} 100 & 255 & 100 & 100 \\ 100 & 255 & 100 & 100 \\ 255 & 100 & 100 & 0 \\ 100 & 100 & 100 & 100 \end{bmatrix}$$

Continued

EXAMPLE 13.8—CONT'D

$$3 \times 3 \text{ median filter kernel}: \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}_{3\times 3}$$

Perform digital filtering, and compare the filtered image with the original one.

Solution:

Step 1: The 3×3 kernel requires zero padding $3/2 = 1$ column of zeros at the left and right edges while $3/2 = 1$ row of zeros at the upper and bottom edges:

0	0	0	0	0	0
0	100	255	100	100	0
0	100	255	100	100	0
0	255	100	100	0	0
0	100	100	100	100	0
0	0	0	0	0	0

Step 2: To process the first element, we cover the 3×3 kernel with the center pointing to the first element to be processed. The sorted data within the kernel are listed in terms of its value as.

$$0,0,0,0,0,0,100,100,255,255.$$

The median value = median (0, 0, 0, 0, 0, 100, 100, 255, 255) = 0. Zero will replace 100.
Step 3: Continue for each element until the last is replaced.
Let us see the element at the location (1,1):

0	0	0	0	0	0
0	100	255	100	100	0
0	100	255	100	100	0
0	255	100	100	0	0
0	100	100	100	100	0
0	0	0	0	0	0

The values covered by the kernel are:

$$100,100,100,100,100,100,255,255,255.$$

The median value = median (100, 100, 100, 100, 100, 100, 255, 255, 255) = 100. The final processed image is

$$\begin{bmatrix} 0 & 100 & 100 & 0 \\ 100 & 100 & 100 & 100 \\ 0 & 100 & 100 & 0 \\ 100 & 100 & 100 & 100 \end{bmatrix}.$$

Some boundary pixels are distorted due to zero padding effect. However, for a large image, the distortion can be omitted versus the overall quality of the image. The 2×2 middle portion matches the original image exactly. The effectiveness of the median filter is verified via this example.

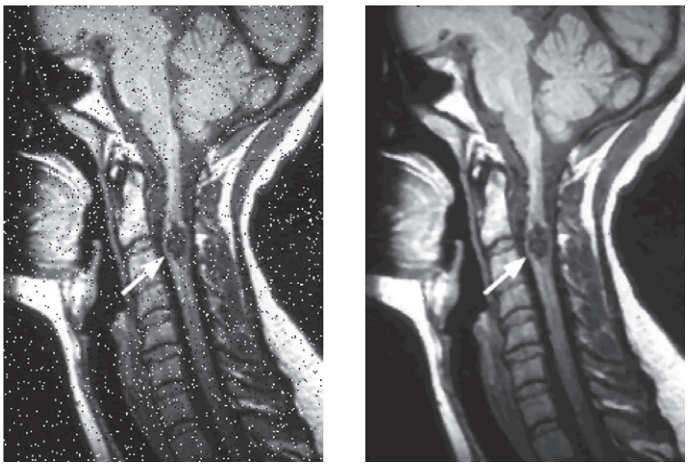


FIG. 13.27

(A) Noisy image (corrupted by "salt and pepper"noise). (B) The enhanced image using the 3×3 median filter.

The image in Fig. 13.27A is corrupted by "salt and pepper" noise. The median filter with a 3×3 kernel is used to filter the impulse noise. The enhanced image shown in Fig. 13.27B has a significant quality improvement. However, the enhanced image also seems smoothed, thus, the high-frequency information is reduced. Note that a larger size kernel is not appropriate for median filtering, because for a larger set of pixels the median value deviates from the pixel value.

13.4.3 EDGE DETECTION

In many applications, such as pattern recognition and fingerprint identification, and iris biometric identification, image edge information is required. To obtain the edge information, a differential convolution kernel is used. Of these kernels, Sobel convolution kernels are used for horizontal and vertical edge detection. They are listed as follows:

Horizontal Sobel edge detector:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{13.12}$$

The kernel subtracts the first row in the kernel from the third row to detect the horizontal difference.

Vertical Sobel edge detector:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{13.13}$$

The kernel subtracts the first column in the kernel from the third column to detect the vertical difference.

Besides Sobel edge detector, a Laplacian edge detector is devised to tackle both vertical and horizontal edges. It is listed as follows.

Laplacian edge detector:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{13.14}$$

EXAMPLE 13.9

Given the following 8-bit grayscale image,

$$5 \times 4 \text{ original image}: \begin{bmatrix} 100 & 100 & 100 & 100 \\ 110 & 110 & 110 & 110 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \end{bmatrix},$$

Use the Sobel horizontal edge detector to detect horizontal edges.

Solution:

We pad the image with zeros before processing as follows:

$$\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 110 & 110 & 110 & 110 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

After processing using the Sobel horizontal edge detector, we have

$$\begin{bmatrix} 330 & 440 & 440 & 330 \\ 0 & 0 & 0 & 0 \\ -30 & -40 & -40 & -30 \\ 0 & 0 & 0 & 0 \\ -300 & -400 & -400 & -300 \end{bmatrix}.$$

Adjusting the scale level leads to

$$\begin{bmatrix} 222 & 255 & 255 & 222 \\ 121 & 121 & 121 & 121 \\ 112 & 109 & 109 & 112 \\ 121 & 121 & 121 & 121 \\ 30 & 0 & 0 & 30 \end{bmatrix}.$$

Disregarding the first row and first column and the last row and column, since they are at image boundaries, we identify a horizontal line of 109 in the third row.

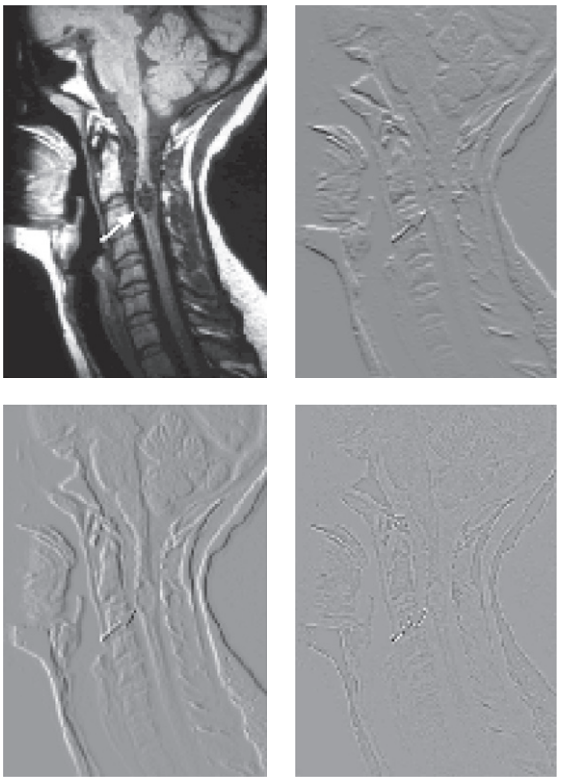


FIG. 13.28

Image edge detection. (Upper left) original image; (upper right) result from Sobel horizontal edge detector; (lower left) result from Sobel vertical edge detector; (lower right) result from Laplacian edge detector.

Fig. 13.28 shows the results from the edge detection.

Fig. 13.29 shows the edge detection for the grayscale image of the cruise ship in Fig. 13.3.

Sobel edge detection can tackle only the horizontal edge or the vertical edge, as shown in Fig. 13.29, where the edges of the image have both horizontal and vertical features. We can simply combine two horizontal and vertical edge-detected images and then rescale the resultant image in the full range. Fig. 13.29C shows that the edge-detection result is equivalent to that of the Laplacian edge detector.

Next, we apply a more sophisticated Laplacian of Gaussian filter for the edge detection, which a combined Gaussian lowpass filter and Laplacian derivative operator (highpass filter). The filter *smoothes* the image to suppress noise using the lowpass Gaussian filter, then performs Laplacian derivative operation for the edge detection, since the noisy image is very sensitive to the Laplacian derivative operation. As we discussed for the Gaussian lowpass filter, the standard deviation in the Gaussian distribution function controls degree of noise filtering before Laplacian derivative operation. A larger value of the standard deviation may blur the image; hence, some edge boundaries could be lost. Its selection should be based on the particular noisy image. The filter kernel with the standard deviation of $\sigma = 0.8$ is given by

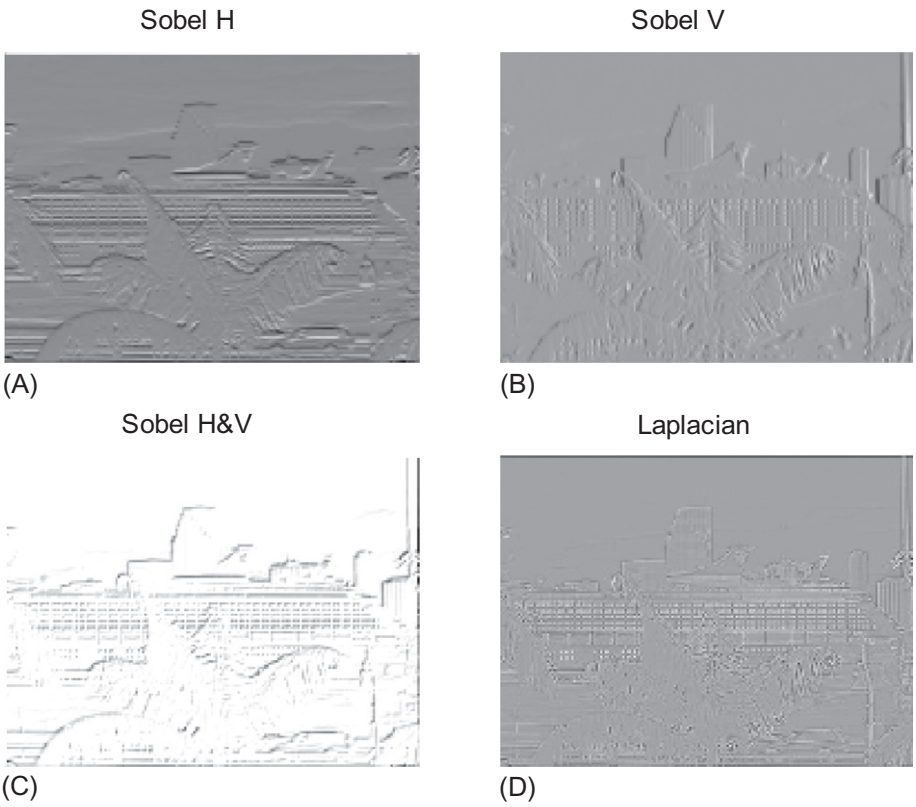


FIG. 13.29

Edge detection. (H, horizontal, V, vertical, H&V, horizontal and vertical.)

$$\begin{bmatrix} 4 & 13 & 16 & 13 & 4 \\ 13 & 9 & -25 & 9 & 13 \\ 16 & -25 & -124 & -25 & 16 \\ 13 & 9 & -25 & 9 & 13 \\ 4 & 13 & 16 & 13 & 4 \end{bmatrix}. \tag{13.15}$$

The processed edge detection using the Laplacian of Gaussian filter in Eq. (13.15) is shown in Fig. 13.30. We can further use a threshold value to convert the processed image to a black and white image, where the contours of objects can be clearly displayed.

13.4.4 MATLAB FUNCTIONS FOR IMAGE FILTERING

MATLAB image filter design and implementation functions are summarized in Fig. 13.31. MATLAB functions are explained as:

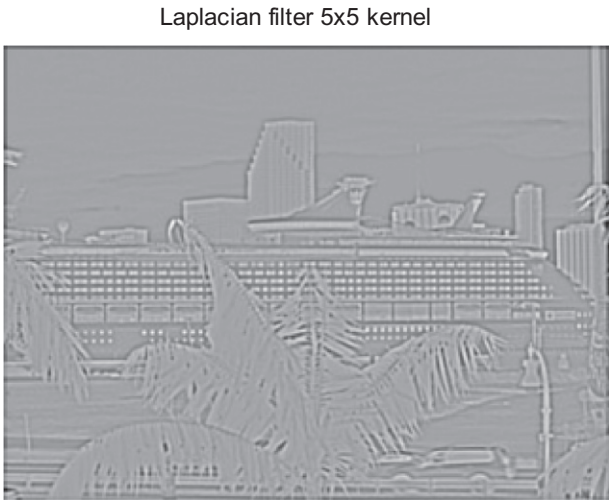


FIG. 13.30

Image edge detection using Laplacian of Gaussian filter.

```
X = image to be processed
fspecial('filter type', kernel size, parameter) = convolution kernel generation
H = fspecial('gaussian',HSIZE,SIGMA) returns a rotationally
symmetric Gaussian lowpass filter of size HSIZE with standard
deviation SIGMA (positive).
H = fspecial('log',HSIZE,SIGMA) returns a rotationally symmetric
Laplacian of Gaussian filter of size HSIZE with standard deviation SIGMA
(positive).
Y = filter2([convolution kernel], X) = two-dimensional filter using the
convolution kernel
Y = medfilt2(X, [row size, column size]) = two-dimensional median filter
```

Program 13.2 lists the sample MATLAB codes for filtering applications. Fig. 13.31 outlines the applications of the MATLAB functions.

Program 13.2. Examples of Gaussian filtering, media filtering, and Laplacian of Gaussian filtering.

```
close all;clear all; clc;
X=imread('cruise.','jpeg');       % Provided by the instructor
Y=rgb2gray(X);                    % Convert the rgb image to the grayscale image
I=im2double(Y);                   % Get the intensity image
image1_g=imnoise(I,'gaussian'); % Add random noise to the intensity image
ng=mat2gray(image1_g);            % Adjust the range
ng=im2uint8(ng);                  % 8-bit corrupted image
```

```
%Linear Filtering
K_size=5;                                   % Kernel size=5x5
sigma=0.8;                                % sigma (the bigger, the smoother the image)
h=fspecial('gaussian,'K_size,sigma);      % Determine Gaussian filter coefficients
%This command will construct a Guassian filter
%of size 5x5 with a mainlobe width of 0.8.
image1_out=filter2(h,image1_g);           % Perform filetring
image1_out=mat2gray(image1_out);          % Adjust the range
image1_out=im2uint8(image1_out);          % Get the 8-bit image
subplot(1,2,1); imshow(ng),title('Noisy image');
subplot(1,2,2); imshow(image1_out);
title('5  5 Gaussian kernel');

%Median Filtering
image1_s=imnoise(I,'salt & pepper');     % Add "salt and pepper" noise to the image
mn=mat2gray(image1_s);                    % Adjust the range
mn=im2uint8(mn);                          % Get the 8-bit image

K_size=3;                                          % kernel size
image1_m=medfilt2(image1_s,[K_size, K_size]);      % Perform median filtering
image1_m=mat2gray(image1_m);                       % Adjust the range
image1_m=im2uint8(image1_m);                       % Get the 8-bit image
figure, subplot(1,2,1);imshow(mn)
title('Median noisy');
subplot(1,2,2);imshow(image1_m);
title('3  3 median kernel');

%Laplacian of Gaussian filtering
K_size=5;                                  % Kernel size
sigma=0.9;                                 % Sigma parameter
h=fspecial('log',K_size,alpha);           % Determine the Laplacian of Gaussian filter
image1_out=filter2(h,I);                   % kernel
                                           % Perform filtering
image1_out=mat2gray(image1_out);           % Adjust the range
image1_out=im2uint8(image1_out);           % Get the 8-bit image
figure,subplot(1,2,1); imshow(Y)
title('Original');
subplot(1,2,2); imshow(image1_out);
title('Laplacian filter 5  5 kernel');
```

13.5 IMAGE PSEUDO-COLOR GENERATION AND DETECTION

We can apply certain transformations to the grayscale image so that it becomes a color image, and a wider range of pseudo-color enhancement can be obtained. In object detection, pseudo-color generation can produce the specific color for the object that is to be detected, say, red. This would significantly increase the accuracy of the identification. To do so, we choose three transformations of the grayscale level to the RGB components, as shown in Fig. 13.32.

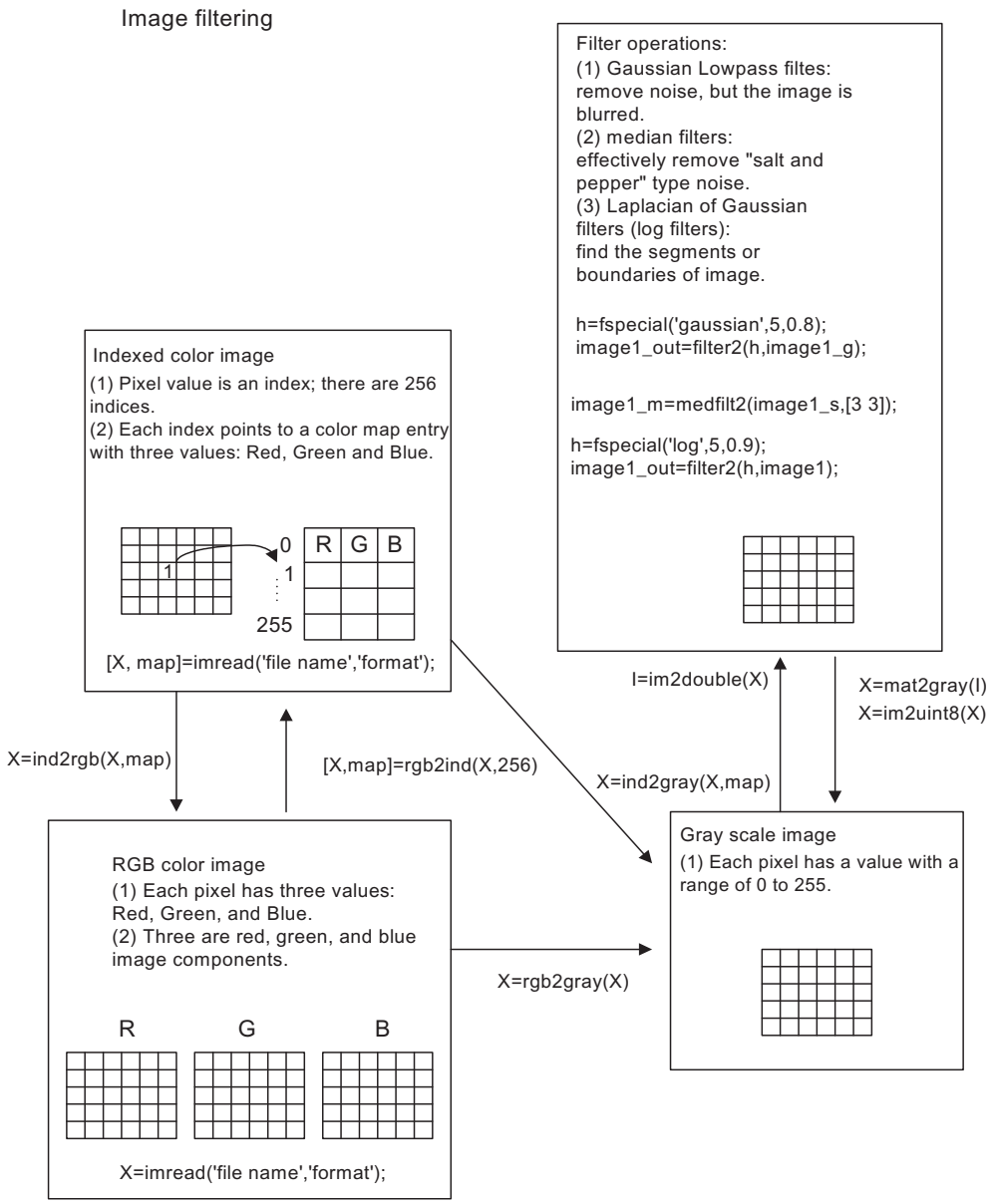


FIG. 13.31

MATLAB functions for filter design and implementation.

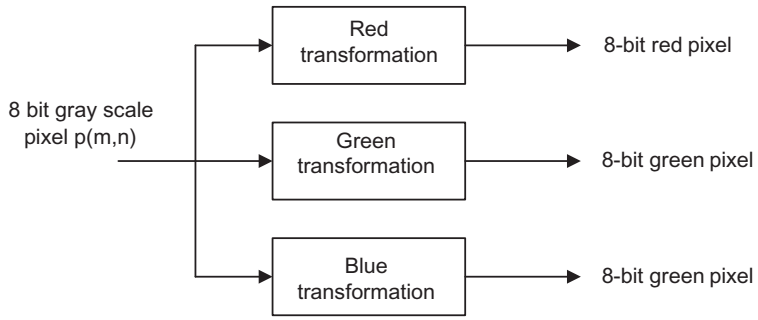


FIG. 13.32

Block diagram for transforming a grayscale pixel to a pseudo-color pixel.

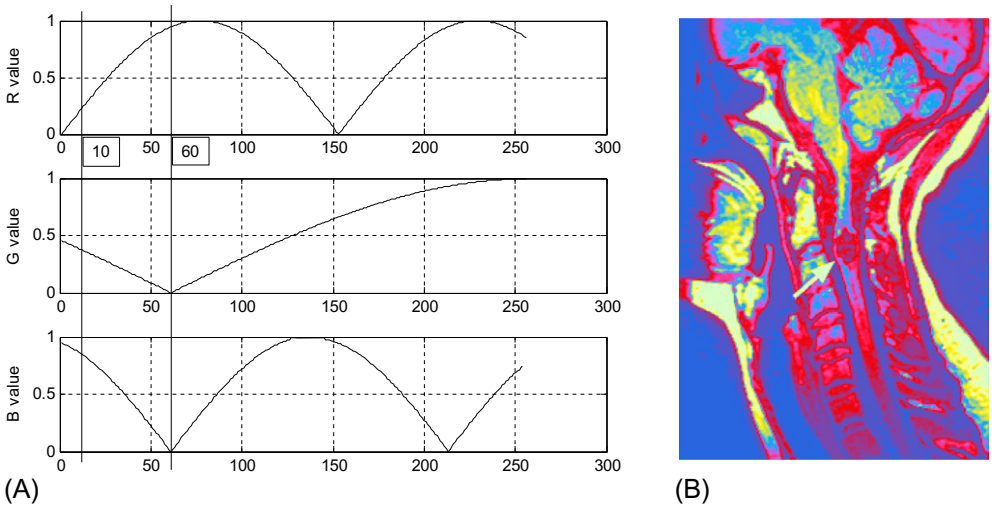


FIG. 13.33

(A) Three sine functions for grayscale transformation. (B) The pseudo-color image (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

As a simple choice, we choose three sine functions for RGB transformations, as shown in Fig. 13.33A. Changing the phase and period of one sine function can be easily done so that the grayscale pixel level of the object to be detected is aligned to the desired color with its component value as large as possible, while the other two functions transform the same grayscale level to have their color component values as small as possible. Hence, the single specified color object can be displayed in the image for identification. By carefully choosing the phase and period of each sine function, certain object(s) could be transformed to the red, green, or blue color with a favorable choice.

EXAMPLE 13.10

In the grayscale image in Fig. 13.13A, the area pointed to by the arrow has a grayscale value approximately equal to 60. The background has a pixel value approximately equal to 10.
Generate the background to be as close to blue as possible, and make the area pointed to by the arrow as close to red color as possible.

Solution:

The transformation functions are chosen as shown in Fig. 13.33A, where the red value is largest at 60 and the blue and green values approach zero. At the grayscale of 10, the blue value is dominant. Fig. 13.33B shows the processed pseudo-color image.

Fig. 13.34 illustrates the pseudo-color generation procedure.
Program 13.3 lists the sample MATLAB codes for pseudo-color generation for a grayscale image.

Program 13.3. Program examples for the pseudo-color generation.

```
close all; clear all;clc
disp('Convert the grayscale image to the pseudo-color image1');
 [X, map]=imread('clipim2','gif'); % Read 8-bit index image, provided by the instructor
 Y=ind2gray(X,map);          % 8-bit color image to the grayscale conversion
% Apply pseudo-color functions using sinusoids
 C_r=304;                    % Cycle change for the red channel
 P_r=0;                      % Phase change for the red channel
 C_b=304;                    % Cycle change for the blue channel
 P_b=60;                     % Phase change for the blue channel
 C_g=804;                    % Cycle change for the green channel
 P_g=60;                     % Phase change for the green channel
 r=abs(sin(2*pi*[-P_r:255-P_r]/C_r));
b=abs(sin(2*pi*[-P_b:255-P_b]/C_b));
g=abs(sin(2*pi*[-P_g:255-P_g]/C_g));
figure, subplot(3.1,1);plot(r,'r');grid;ylabel('R value')
subplot(3,1,2);plot(g,'g');grid;ylabel('G value');
subplot(3,1,3);plot(b,'b');grid;ylabel('B value');
figure, imshow(Y);
map=[r;g;b;]';          % Construct the color map
figure, imshow(X,map);   % Display the pseudo-color image
```

13.6 IMAGE SPECTRA

In 1D signal processing such as for speech and audio, we need to examine the frequency contents, check filtering effects, and perform feature extraction. Image processing is similar. However, we need apply a two-dimensional discrete Fourier transform (2D-DFT) instead of an 1D DFT. The spectrum including the magnitude and phase is also in two dimensions. The equations of the 2D-DFT are given by:

$$X(u,v)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}p(m,n)W_M^{um}W_N^{vn}, \tag{13.16}$$

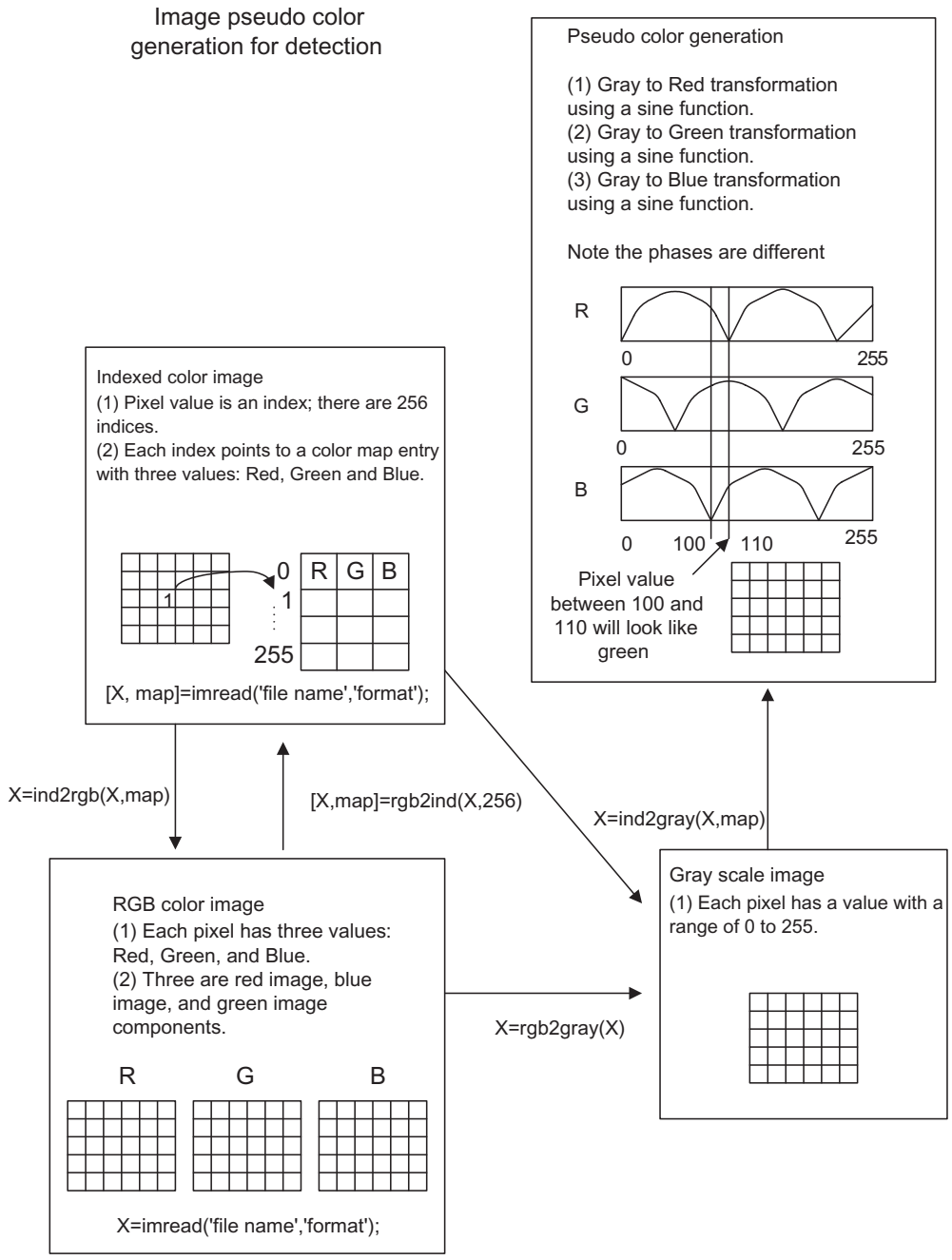


FIG. 13.34

Illustrative procedure for pseudo-color generation.

where $W_M = e^{-j\frac{2\pi}{M}}$ and $W_N = e^{-j\frac{2\pi}{N}}$.

m and n = pixel locations.

u and v = frequency indices.

Taking absolute value of the 2D-DFT coefficients $X(u,v)$ and dividing the absolute value by $(M \times N)$, we get the magnitude spectrum as

$$A(u,v) = \frac{1}{(N \times M)}|X(u,v)|. \tag{13.17}$$

Instead of going through the details of the 2D-DFT, we focus on application results via examples.

EXAMPLE 10.11

Determine the 2D-DFT coefficients and magnitude spectrum for the following 2×2 image.

$$\begin{bmatrix} 100 & 50 \\ 100 & -10 \end{bmatrix}.$$

Solution:

Since $M = N = 2$, applying Eq. (13.16) leads to

$$\begin{aligned} X(u,v) = {} & p(0,0)e^{-j\frac{2\pi u\times 0}{2}} \times e^{-j\frac{2\pi v\times 0}{2}} + p(0,1)e^{-j\frac{2\pi u\times 0}{2}} \times e^{-j\frac{2\pi v\times 1}{2}} \\ & + p(1,0)e^{-j\frac{2\pi u\times 1}{2}} \times e^{-j\frac{2\pi v\times 0}{2}} + p(1,1)e^{-j\frac{2\pi u\times 1}{2}} \times e^{-j\frac{2\pi v\times 1}{2}}. \end{aligned}$$

For $u = 0$ and $v = 0$, we have

$$\begin{aligned} X(0,0) &= 100e^{-j0} \times e^{-j0} + 50e^{-j0} \times e^{-j0} + 100e^{-j0} \times e^{-j0} - 10e^{-j0} \times e^{-j0} \\ &= 100 + 50 + 100 - 10 = 240 \end{aligned}$$

For $u = 0$ and $v = 1$, we have

$$\begin{aligned} X(0,1) &= 100e^{-j0} \times e^{-j0} + 50e^{-j0} \times e^{-j\pi} + 100e^{-j0} \times e^{-j0} - 10e^{-j0} \times e^{-j\pi} \\ &= 100 + 50 \times (-1) + 100 - 10 \times (-1) = 160 \end{aligned}$$

Following similar operations,

$$X(1,0) = 60 \text{ and } X(1,1) = -60.$$

Thus, we have DFT coefficients as

$$X(u,v) = \begin{bmatrix} 240 & 160 \\ 60 & -60 \end{bmatrix}.$$

Using Eq. (13.17), we can calculate the magnitude spectrum as

$$A(u,v) = \begin{bmatrix} 60 & 40 \\ 15 & 15 \end{bmatrix}.$$

We can use the MABLAB function **fft2()** to verify the calculated DFT coefficients:

```
>> X=fft2([100 50;100 -10])
   X =
        240   160
         60   -60.
```

EXAMPLE 13.12

Given the following 200 × 200 grayscale image with a white rectangular (11 × 3 pixels) at its center and a black background, shown in Fig. 13.35A, we can compute the image's amplitude spectrum whose magnitudes are scaled in the range from 0 to 255. We can display the spectrum in terms of the grayscale. Fig. 13.35B shows the spectrum image.

The displayed spectrum has four quarters. The left upper quarter corresponds to the frequency components, and the other three quarters are the image counterparts. From the spectrum image, the area of the upper left corner is white and hence has the higher scale value. So, the image signal has low-frequency dominant components. The spectrum exhibits horizontal and vertical null lines (dark lines). The first vertical null line can be estimated as $200/11 = 18$, while the first horizontal null line happens at $200/3 = 67$. Next, let us apply the 2D spectrum to understand the image filtering effects in image enhancement.

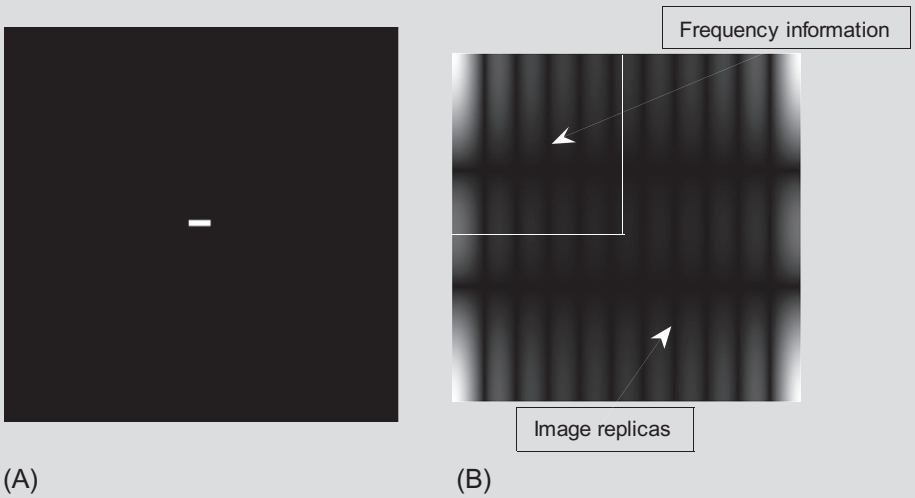


FIG. 13.35

(A) A square image. (B) Magnitude spectrum for the square image.

EXAMPLE 13.13

Fig. 13.36A is a biomedical image corrupted by random noise. Before we apply lowpass filtering, its 2D-DFT coefficients are calculated. We then compute its magnitude spectrum and scale it to the range from 0 to 255. To see noise spectral components, the spectral magnitude is further multiplied by a factor of 100. Once the spectral value is larger than 255, it is clipped to 255. The resultant spectrum is displayed in Fig. 13.36B, where we can see that noise occupies the entirety of the image.

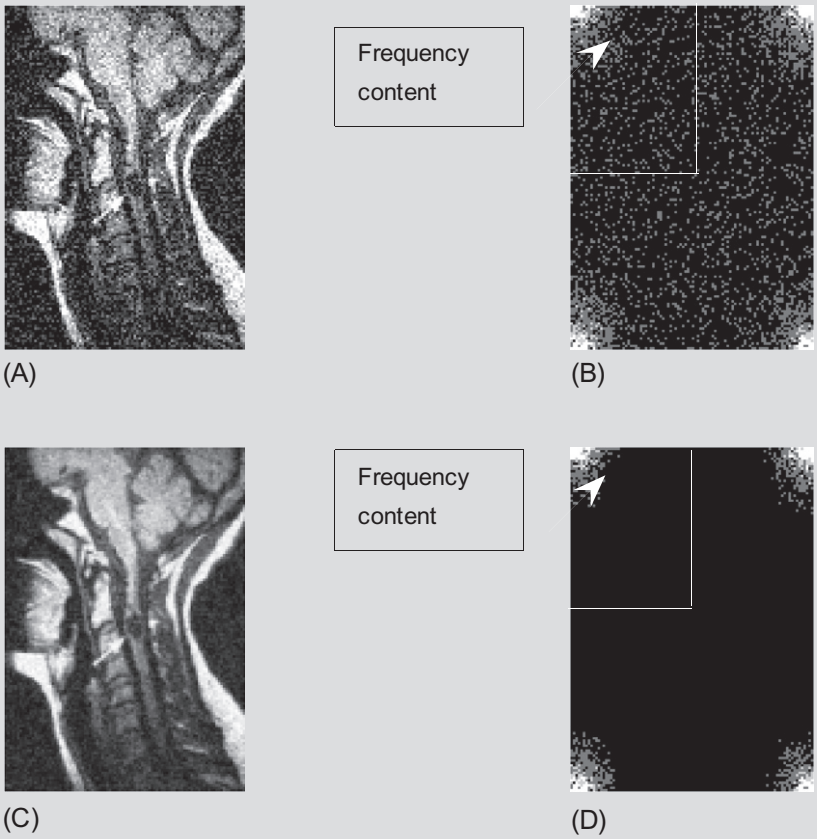


FIG. 13.36

Magnitude spectrum plots for the noisy image and the noise-filtered image. (A) The noisy image, (B) magnitude spectrum of the noisy image, (C) noise-filtered image, (D) magnitude spectrum of the noise-filtered image.

To enhance image, we apply a Gaussian lowpass filter. The enhanced image is shown in Fig. 13.36C, in which the enhancement is easily observed. Fig. 13.36D displays the spectra for the enhanced image with the same scaling process as described. As we can see, the noise is significantly reduced in comparison with Fig. 13.36B.

13.7 IMAGE COMPRESSION BY DISCRETE COSINE TRANSFORM

Image compression is necessary in our modern media systems, such as digital still and video cameras and computer systems. The purpose of compression is to reduce information storage or transmission bandwidth without losing image quality or at least without losing it significantly. Image compression can be classified as lossless compression or lossy compression. Here we focus on the lossy compression using discrete-cosine transform (DCT).

The DCT is a core compression technology used in the industry standards JPEG (Joint Photographic Experts Group) for still image compression and MPEG (Motion Picture Experts Group) for video compression, achieving compression ratio of 20:1 without noticeable quality degradation. JPEG standard image compression is used every day in real life.

The principle of the DCT is to transform the original image pixels to their DCT coefficients with the same number of the original image pixels, where the DCT coefficients have nonuniform distribution of direct current (DC) terms representing the average values, and alternate current (AC) terms representing fluctuations. The compression is achieved by applying the advantages of encoding DC terms (of a large dynamic range) with a large number of bits and low-frequency AC terms (a few, with a reduced dynamic range) with a reduced number of bits, and neglecting some high-frequency AC terms having small dynamic ranges (most of them do not affect the visual quality of the picture).

13.7.1 TWO-DIMENSIONAL DISCRETE COSINE TRANSFORM

Image compression uses two-dimensional discrete cosine transform (2D-DCT), whose transform pairs are defined as:

Forward DCT:

$$F(u,v)=\frac{2C(u)C(v)}{\sqrt{MN}}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}p(ij)\cos\left(\frac{(2i+1)u\pi}{2M}\right)\cos\left(\frac{(2j+1)v\pi}{2N}\right) \tag{13.18}$$

Inverse DCT:

$$p(i,j)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}\frac{2C(u)C(v)}{\sqrt{MN}}F(uv)\cos\left(\frac{(2i+1)u\pi}{2M}\right)\cos\left(\frac{(2j+1)v\pi}{2N}\right) \tag{13.19}$$

where

$$C(m)=\begin{cases}\frac{\sqrt{2}}{2} & \text{if } m=0\\ 1 & \text{otherwise}\end{cases} \tag{13.20}$$

$p(i,j)$ = pixel level at the location (i,j)

$F(u,v)$ = DCT coefficient at the frequency indices (u,v).

JPEG divides an image into 8×8 image subblocks and applies DCT transform for each subblock individually. Hence, we simplify the general 2D-DCT in terms of 8×8 size. The equation for 2D 8×8 DCT is modified as:

$$F(u,v)=\frac{C(u)C(v)}{4}\sum_{i=0}^{7}\sum_{j=0}^{7}p(ij)\cos\left(\frac{(2i+1)u\pi}{16}\right)\cos\left(\frac{(2j+1)v\pi}{16}\right) \tag{13.21}$$

The inverse of 2D 8×8 DCT is expressed as:

$$p(i,j)=\sum_{u=0}^{7}\sum_{v=0}^{7}\frac{C(u)C(v)}{4}F(uv)\cos\left(\frac{(2i+1)u\pi}{16}\right)\cos\left(\frac{(2j+1)v\pi}{16}\right) \tag{13.22}$$

To become familiar with the 2D-DCT formulas, we study Example 13.14.

EXAMPLE 13.14

Determine the 2D-DCT coefficients for the following image:

$$\begin{bmatrix} 100 & 50 \\ 100 & -10 \end{bmatrix}.$$

Solution:

Applying $N=2$ and $M=2$ to Eq. (13.18) yields

$$F(u,v)=\frac{2C(u)C(v)}{\sqrt{2\times 2}}\sum_{i=0}^{1}\sum_{j=0}^{1}p(ij)\cos\left(\frac{(2i+1)u\pi}{4}\right)\cos\left(\frac{(2j+1)v\pi}{4}\right)$$

For $u=0$ and $v=0$, we achieve

$$\begin{aligned} F(0,0) &= c(0)c(0)\sum_{i=0}^{1}\sum_{j=0}^{1}p(i,j)\cos(0)\cos(0) \\ &= \left(\frac{\sqrt{2}}{2}\right)^2[p(0,0)+p(0,1)+p(1,0)+p(1,1)] \\ &= \frac{1}{2}\quad(100+50+100-10)=120 \end{aligned}$$

For $u=0$ and $v=1$, we achieve

$$\begin{aligned} F(0,1) &= c(0)c(1)\sum_{i=0}^{1}\sum_{j=0}^{1}p(i,j)\cos(0)\cos\left(\frac{(2j+1)\pi}{4}\right) \\ &= \left(\frac{\sqrt{2}}{2}\right)\times 1\times\left(p(0,0)\cos\frac{\pi}{4}+p(0,1)\cos\frac{3\pi}{4}+p(1,0)\cos\frac{\pi}{4}+p(1,1)\cos\frac{3\pi}{4}\right) \\ &= \frac{\sqrt{2}}{2}\quad\left(100\times\frac{\sqrt{2}}{2}+50\left(-\frac{\sqrt{2}}{2}\right)+100\times\frac{\sqrt{2}}{2}-10\left(-\frac{\sqrt{2}}{2}\right)\right)=80 \end{aligned}$$

Similarly,

$$F(1,0)=30\,\text{and}\,F(1,1)=-30.$$

Finally, we get

$$F(u,v)=\begin{bmatrix} 120 & 80 \\ 30 & -30 \end{bmatrix}.$$

Applying the MATLAB function **dct2()** to verify the DCT coefficients as follows:

```
>> F=dct2([100 50;100 -10])
F =
 120.0000  80.0000
  30.0000 -30.0000
```

EXAMPLE 13.5

Given the following DCT coefficients from a 2 × 2 image:

$$F(u,v)=\begin{bmatrix}120 & 80\\ 30 & -30\end{bmatrix},$$

Determine the pixel $p(0,0)$.

Solution:

Applying Eq. (13.19) of the inverse 2D-DCT with $N=M=2$, $i=0$, and $j=0$, it follows that

$$\begin{aligned}p(0,0)&=\sum_{u=0}^{1}\sum_{v=0}^{1}c(u)c(v)F(u,v)\cos\left(\frac{u\pi}{4}\right)\cos\left(\frac{v\pi}{4}\right)\\&=\left(\frac{\sqrt{2}}{2}\right)\times\left(\frac{\sqrt{2}}{2}\right)\times F(0,0)+\left(\frac{\sqrt{2}}{2}\right)\times F(0,1)\times\left(\frac{\sqrt{2}}{2}\right)\\&\quad+\left(\frac{\sqrt{2}}{2}\right)\times F(1,0)\times\left(\frac{\sqrt{2}}{2}\right)+F(0,1)\left(\frac{\sqrt{2}}{2}\right)\times\left(\frac{\sqrt{2}}{2}\right)\\&=\frac{1}{2}\times 120+\frac{1}{2}\times 80+\frac{1}{2}\times 30+\frac{1}{2}(-30)=100.\end{aligned}$$

We apply the MATLAB function **idct2()** to verify the inverse DCT to get the pixel values as follows:

```
>> p=idct2([120 80; 30 -30])
   p =
     100.0000  50.0000
     100.0000 -10.0000.
```

13.7.2 TWO-DIMENSIONAL JPEG GRAYSCALE IMAGE COMPRESSION EXAMPLE

To understand the JPEG image compression, we examine the 8 × 8 grayscale subblock. Table 13.5 shows a subblock of the grayscale image shown in Fig. 13.37 to be compressed.

Applying 2D-DCT leads to Table 13.6.

Table 13.5 8 × 8 Subblock

150	**148**	**140**	**132**	**150**	**155**	**155**	**151**
155	152	143	136	152	155	155	153
154	149	141	135	150	150	150	150
156	150	143	139	154	152	152	155
156	151	145	140	154	152	152	155
154	152	146	139	151	149	150	151
156	156	151	142	154	154	154	154
151	154	149	139	151	153	154	153

FIG. 13.37

Original image.

Table 13.6 DCT Coefficients for the Subblock Image in Table 13.5

1198	**−10**	**26**	**24**	**−5**	**−16**	**0**	**12**
−8	−6	3	8	0	0	0	0
0	−3	0	0	−8	0	0	0
0	0	0	0	0	0	0	0
0	−4	0	0	0	0	0	0
0	0	−1	0	0	0	0	0
−10	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

These DCT coefficients have a big DC component of 1198 but small AC component values. These coefficients are further normalized (quantized) with a quality factor Q, defined in Table 13.7.

After normalization, as shown in Table 13.8, the DC coefficient is reduced to 75, and a few small AC coefficients exist, while most are zero. We can encode and transmit only nonzero DCT coefficients and omit transmitting zeros, since they do not carry any information. They can be easily recovered by resetting coefficients to zero during decoding. By this principle we achieve data compression.

As shown in Table 13.8, most nonzero coefficients resides at the upper left corner. Hence, the order of encoding for each value is based on the zigzag path in which the order is numbered, as in Table 13.9.

According to the order, we record the nonzero DCT coefficients as a JPEG vector, shown as:

$$\text{JPEG vector:} \quad \mathbf{[75\ -1\ -1\ 0\ -1\ 3\ 2\ EOB]}$$

where "EOB" = the end of block coding. The JPEG vector can further be compressed by encoding the difference of DC values between subblocks, in *differential pulse code modulation* (DPCM), as discussed in Chapter 10, as well as by run-length coding of AC values and Huffman coding, which both belong to lossless compression techniques. We discuss this in the next section.

Table 13.7 The Quality Factor

16	**11**	**10**	**16**	**24**	**40**	**51**	**61**
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 13.8 Normalized DCT Coefficients

75	**−1**	**3**	**2**	**0**	**0**	**0**	**0**
−1	−1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 13.9 The Order to Scan DCT Coefficients

0	**1**	**5**	**6**	**14**	**15**	**27**	**28**
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

During the decoding stage, the JPEG vector is recovered first. Then the quantized DCT coefficients are recovered according to the zigzag path. Next, the recovered DCT coefficients are multiplied by a quality factor to obtain the estimate of the original DCT coefficients. Finally, we apply the inverse DCT to achieve the recovered image subblock, which is shown in Table 13.10.

For comparison, the errors between the recovered image and original image are calculated and listed in Table 13.11.

Table 13.10 The Recovered Image Subblock

153	**145**	**138**	**139**	**147**	**154**	**155**	**153**
153	145	138	139	147	154	155	153
155	147	139	140	148	154	155	153
157	148	141	141	148	155	155	152
159	150	142	142	149	155	155	152
161	152	143	143	149	155	155	152
162	153	144	144	150	155	154	151
163	154	145	144	150	155	154	151

Table 13.11 The Coding Error of the Image Subblock

3	**−3**	**−2**	**7**	**−3**	**−1**	**0**	**2**
−2	−7	−5	3	−5	−1	0	0
1	−2	−2	5	−2	4	5	3
1	−2	−2	2	−6	3	3	−3
3	−1	−3	2	−5	3	3	−3
7	0	−3	4	−2	6	5	1
6	−3	−7	2	−4	1	0	−3
12	0	−4	5	−1	2	0	−2

The original and compressed images are displayed in Figs. 13.37 and 13.38, respectively. We do not see any noticeable difference between these two grayscale images.

13.7.3 JPEG COLOR IMAGE COMPRESSION

This section is devoted to reviewing the JPEG standard compression and examines the steps briefly. We focus on the encoder, since the decoder is just the reverse process of the encoding. The block diagram for the JPEG encoder is in Fig. 13.39.

The JPEG encoder has the following main steps:

1. Transform RGB to YIQ or YUV (U and V = chrominance components).
2. Perform DCT on blocks.
3. Perform quantization.
4. Perform zigzag ordering, DPCM, and run-length encoding.
5. Perform entropy encoding (Huffman coding)

***RGB to YIQ transformation*:**

The first transformation is of the RGB image to a YIQ or YUV image. Transformation from RGB to YIQ has previously been discussed. The principle is that in YIQ format, the luminance channel carries more signal energy, up to 93%, while the chrominance channels carry up to only 7% of signal energy. After transformation, more effort can be spent on coding the luminance channel.

FIG. 13.38

JPEG compressed image.

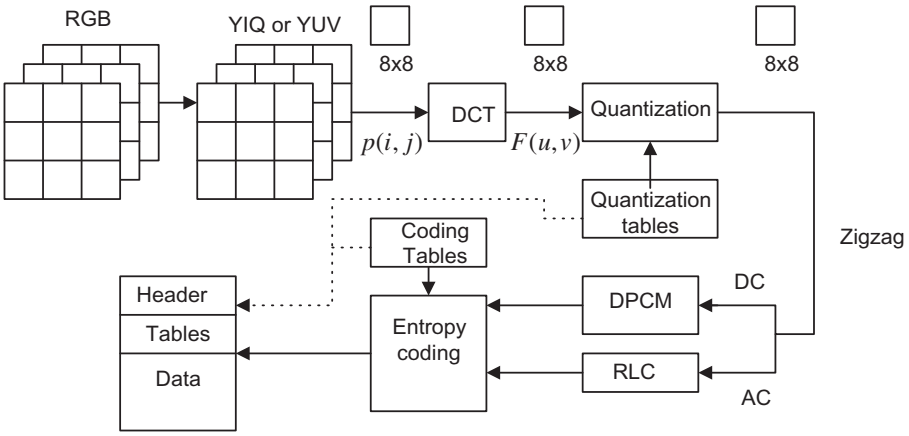


FIG. 13.39

Block diagram for JPEG encoder.

***DCT on image blocks*:**

Each image is divided into 8×8 blocks. 2D-DCT is applied to each block to obtain the 8×8 DCT coefficient block. Note that there are three blocks, Y, I, and Q.

***Quantization*:**

The quantization is operated using the 8×8 quantization matrix. Each DCT coefficient is quantized, divided by the corresponding value given in the quantization matrix. In this way, a smaller number of bits can be used for encoding the DCT coefficients. There are two different quantization tables, one for luminance as shown in Table 13.12 (which is the same as the one in the last section and listed here again for comparison) and the other one for chrominance shown in Table 13.13.

Table 13.12 The Quality Factor for Luminance

16	**11**	**10**	**16**	**24**	**40**	**51**	**61**
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 13.13 The quality Factor for Chrominance

17	**18**	**24**	**47**	**99**	**99**	**99**	**99**
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

We can see that the chrominance table has numbers with larger values, so that small values of DCT coefficients will result and hence a less number of bits are required for encoding each DCT coefficient. Zigzag ordering to produce the JPEC vector is similar to the grayscale case, except that there are three JPEG vectors.

***DPCM on Direct-Current Coefficients*:**

Since each 8×8 image block has only one DC coefficient, which can be a very large number and varies slowly, we make use of DPCM for coding DC coefficients. As an example for the first five image blocks, DC coefficients are 200, 190, 180, 160, and 170. DPCM with a coding rule of $d(n) = DC(n) - DC(n-1)$ with initial condition $d(0) = DC(0)$ produces a DPCM sequence as

$$200, -10, -10, -20, 10.$$

Hence, the reduced signal range of these values is feasible for entropy coding.

***Run-length coding on Alternating-Current Coefficients*:**

The run-length method encodes the pair of

- the number zeros to skip and
- next nonzero value.

The zigzag scan of the 8×8 matrix makes up a vector of 64 values with a long runs of zeros. For example, the quantized DCT coefficients are scanned as

$$[75, -1, 0-1, 0, 0, -1\,3, 0, 0, 0, 2, 0, 0, \ldots, 0],$$

with one run, two runs, and three runs of zeros in the middle and 52 extra zeros toward the end. The run-length method encodes AC coefficients by producing a pair (run length, value), where the run length is the number of zeros in the run, while the value is the next nonzero coefficient. A special pair (0, 0) indicates EOB. Here is the result from a run-length encoding of AC coefficients:

$$(0, -1), (1, -1), (2, -1), (0, 3), (3, 2), (0, 0).$$

***Lossless Entropy Coding*:**
The DC and AC coefficients are further compressed using entropy coding. JPEG allows Huffman coding and arithmetic coding. We focus on the Huffman coding here.

***Coding DC coefficients*:**
Each DPCM-coded DC coefficient is encoded by a pair of symbols (size, amplitude) with the size (4-bit code) designating the number of bits for the coefficient as shown in Table 13.14, while the amplitude is encoded by the actual bits. For the negative number of the amplitude, 1's complement is used.

For example, we can code the DPCM-coded DC coefficients 200, −10, −10, −20, 10 as

$$(8, 11001000), (4, 0101), (4, 0101), (5, 01011), (4, 1010).$$

Since there needs to be 4 bits for encoding each size, we can use 45 bits in total for encoding the DC coefficients for these five subblocks.

***Coding AC coefficients*:**
The run-length AC coefficients have the format as (run length, value). The value can be further compressed using the Huffman coding method, similar to coding the DPCM-coded DC coefficients. The run length and the size are each encoded by 4 bits and packed into a byte.

Symbol 1: (run length, size)

Symbol 2: (amplitude).

The 4-bit run length can tackle only the number of runs for zeros from 1 to 15. If the run length of zeros is larger than 15, then a special code (15,0) is used for Symbol 1. Symbol 2 is the amplitude in Huffman coding as shown in Table 13.14, while the encoded Symbol 1 is kept in its format:

(run length, size, amplitude).

Let us code the following run-length code of AC coefficients:

$$(0, -1), (1, -1), (2, -1), (0, 3), (3, 2), (0, 0).$$

Table 13.14 Huffman Coding Table

Size	Amplitude
1	−1,1
2	−3,−2,2,3
3	−7,..,−4.4, …,7
4	−15, …,−8.8, …,15
5	−31, …,−16,16, …,31
.	.
.	.
.	.
10	−1023, …,−512,512, …,1023

FIG. 13.40

JEPG compressed color image (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

We can produce a bit stream for AC coefficients as:

$$(0000, 0001, 0), (0001, 0001, 0), (0010, 0001, 0),$$
$$(0000, 0010, 11), (0011, 0010, 10), (0000, 0000).$$

There are 55 bits in total. Fig. 13.40 shows a JPEG compressed color image. The decompressed image is indistinguishable from the original image after comparison.

13.7.4 IMAGE COMPRESSION USING WAVELET TRANSFORM CODING

We can extend the 1D discrete wavelet transform (1D-DWT) discussed in chapter 12 to the 2D-DWT. The procedure is described as follows. Given an $N \times N$ image, the 1D DWT using level one is applied to each row of the image; and after all the rows are transformed, the level-1 1D DWT is applied again to each column. After the rows and columns are transformed, we achieve four first-level subbands labeled as LL, HL, LH, and HH shown in Fig. 13.41A. The same procedure repeats for the LL band only and results in the second-level subbands: LL2, HL2, LH2, and HH2 as shown in Fig. 13.41B. The process proceeds to the higher-level as desired. With the obtained wavelet transform, we can quantize coefficients to achieve compression requirement. For example, for the two-level coefficients, we can omit HL1, LH1, HH1 to simply achieve a 4:1 compression ratio. The decompression is a reverse process, that is, inversely transforms columns and then rows of the wavelet coefficients. We can apply the inverse discrete wavelet transform (IDWT) to the recovered LL band with the column and row inverse transform processes, and continue until the inverse transform at level one is completed. Let us look at an illustrative example.

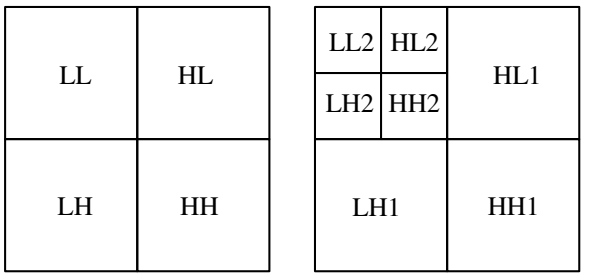


FIG. 13.41

The two-dimensional DWT for level 1 and level 2. (A) Level-1 transformation, (B) Level-2 transformation.

EXAMPLE 13.6

Given a 4 × 4 image shown as follows:

113	135	122	109
102	116	119	124
105	148	138	122
141	102	140	132

(a) Perform 2D-DWT using the 2-tap Haar wavelet.
(b) Using the result in (a), perform 2D-IDWT using the 2-tap Haar wavelet.

Solution:

(a) The MATLAB function **dwt()** is applied to each row. Execution result for the first-row is displayed as follows:

```
>> dwt([1 1]/sqrt(2),[113 135 122 109],1)'      %Row vector coefficients
ans =176.0696 163.3417 -13.8492  9.1924
```

The completed row transform is listed as follows:

176.0696	163.3417	−13.8492	9.1924
154.1493	171.8269	−9.8995	−3.5355
178.8980	183.8478	−30.4056	11.3137
171.8269	192.3330	27.5772	5.6569

Next, the result for the first column is shown as follows:

```
>> dwt([1 1]/sqrt(2),[ 176.0696 154.1493 178.8980 171.8269 ],1)'
ans = 233.5000 248.0000  15.5000  5.0000
```

Applying each column completes level-1 transformation:

$$\begin{matrix} 233.5000 & 237.0000 & -17.5000 & 4.0000 \\ 248.0000 & 266.0000 & -2.0000 & 12.0000 \\ 15.5000 & -6.0000 & -3.5000 & 9.0000 \\ 5.0000 & -6.0000 & -41.0000 & 4.0000 \end{matrix}$$

Now, we perform the Level-2 transformation. Applying the transform to the first row for LL1 band yields

```
>> dwt([1 1]/sqrt(2),[ 233.5000 237.0000],1)'
ans = 332.6937  -2.4749
```

After completing the row transformation, the result is given by

$$\begin{matrix} 332.6937 & -2.4749 & -17.5000 & 4.0000 \\ 363.4529 & -12.7279 & -2.0000 & 12.0000 \\ 15.5000 & -6.0000 & -3.5000 & 9.0000 \\ 5.0000 & -6.0000 & -41.0000 & 4.0000 \end{matrix}$$

Similarly, the first-column MATLAB result is listed as follows:

```
>> dwt([1 1]/sqrt(2),[ 332.6937 363.4529 ],1)'
ans = 492.2500 -21.7500
```

Finally, we achieve the completed level-2 DWT as

$$\begin{matrix} 492.2500 & -10.7500 & -17.5000 & 4.0000 \\ -21.7500 & 7.2500 & -2.0000 & 12.0000 \\ 15.5000 & -6.0000 & -3.5000 & 9.0000 \\ 5.0000 & -6.0000 & -41.0000 & 4.0000 \end{matrix}$$

(b) Recovering LL2 band first, the first column reconstruction is given as

```
>> idwt([1 1]/sqrt(2),[ 492.2500 -21.7500],1)'
ans = 332.6937 363.4529
```

Completing the inverse of the second column in the LL2 band gives

$$\begin{matrix} 332.6937 & -2.4749 & -17.5000 & 4.0000 \\ 363.4529 & -12.7279 & -2.0000 & 12.0000 \\ 15.5000 & -6.0000 & -3.5000 & 9.0000 \\ 5.0000 & -6.0000 & -41.0000 & 4.0000 \end{matrix}$$

Now, we show the first row result for LL2 band in MATLAB as follows:

```
>> idwt([1 1]/sqrt(2),[ 332.6937 -2.4749 ],1)'
ans = 233.5000 237.0000
```

Continued

EXAMPLE 13.6—CONT'D

The recovered LL1 band is shown as follows:

$$\begin{matrix} 233.5000 & 237.0000 & -17.5000 & 4.0000 \\ 248.0000 & 266.0000 & -2.0000 & 12.0000 \\ 15.5000 & -6.0000 & -3.5000 & 9.0000 \\ 5.0000 & -6.0000 & -41.0000 & 4.0000 \end{matrix}$$

Now we are at the level-1 inverse process. For simplicity, the first column result in MATLAB and the completed results are listed as follows, respectively.

```
>> idwt([1 1]/sqrt(2),[ 233.5000 248.0000 15.5000 5.0000],1)'
ans = 176.0696 154.1493 178.8980 171.8269
```

$$\begin{matrix} 176.0696 & 163.3417 & -13.8492 & 9.1924 \\ 154.1493 & 171.8269 & -9.8995 & -3.5355 \\ 178.8980 & 183.8478 & -30.4056 & 11.3137 \\ 171.8269 & 192.3330 & 27.5772 & 5.6569 \end{matrix}$$

Finally, we perform the inverse of row transform at level one. The first row result in MATLAB is listed as follows:

```
>> idwt([1 1]/sqrt(2),[ 176.0696 163.3417 -13.8492  9.1924],1)'
ans = 113.0000 135.0000 122.0000 109.0000
```

The final inversed DWT is yielded as

$$\begin{matrix} 113.0000 & 135.0000 & 122.0000 & 109.0000 \\ 102.0000 & 116.0000 & 119.0000 & 124.0000 \\ 105.0000 & 148.0000 & 138.0000 & 122.0000 \\ 141.0000 & 102.0000 & 140.0000 & 132.0000 \end{matrix}$$

Since there is no quantization for each coefficient, we obtain a perfect reconstruction.

Fig. 13.42 shows an 8-bit grayscale image compression by applying one-level wavelet transform, in which the Daubechies wavelet filter with 16-tap is used. The wavelet coefficients (each is coded using 8-bit) are shown in Fig. 13.42A. By discarding HL, LH, and HH band coefficients, we can achieve 4:1 compression. The decoded image is displayed in Fig. 13.42C. The MATLAB program is listed in Program 13.4.

Fig. 13.43 illustrates two-level wavelet transform and compression results. By discarding HL2, LH2, HH2, HL1, LH1, and HH1 subbands, we achieve 16:1 compression. However, as shown in Fig. 13.43C, we can observe a noticeable degradation of image quality. Since the high-frequency details are discarded, the compressed image shows a significant smooth effect. In addition, there are many advanced methods to quantize and compress the wavelet coefficients. Of these compression

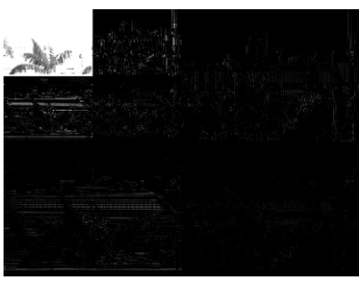
(A) Wavelet coefficients

(B) Original image

(C) 4:1 Compression

FIG. 13.42

(A) Wavelet coefficients; (B) original image; (C) 4:1 compression.



(A) Wavelet coefficients

(B) Original image

(C) 16:1 Compression

FIG. 13.43

(A) Wavelet coefficients; (B) original image; (C) 16:1 compression.

techniques, the embedded zerotree of the wavelet coefficient (EZW) method is the most efficient one, which can be found in Li et al. (2014).

Program 13.4. One-level wavelet transform and compression.

```
close all; clear all; clc
X=imread('cruise','JPEG');
Y=rgb2gray(X); % Convert the image into grayscale
h0=[0.054415842243144  0.312871590914520 0.675630736297712 ...
  0.585354683654425  -0.015829105256675 -0.284015542962009 ...
  0.000472484573805  0.128747426620538  -0.017369301001845 ...
  -0.044088253930837  0.013981027917411 0.008746094047413 ...
  -0.004870352993456  -0.000391740373377 0.000675449406451 ...
  -0.000117476784125];
M=length(h0);
h1(1:2:M-1)=h0(M:-2:2);h1(2:2:M)=-h0(M-1:-2:1);%Obtain QMF highpass filter
[m n]=size(Y);
% Level-1 transform
[m n]=size(Y);
for i=1:m
    W1(i,:)=dwt(h0,double(Y(i,:)),1)';
end
for i=1:n
    W1(:,i)=dwt(h0,W1(:,i),1); % Wavelet coefficients at level-1
end
%Quantization using 8-bit
wmax=double(max(max(abs(W1)))); %Scale factor
W1=round(double(W1)*2^7/wmax); %Get 8-bit data
W1=double(W1)*wmax/2^7;%Recover the wavelet
Figure (1); imshow(uint8(W1));xlabel('Wavelet coefficients');
% 8-bit Quantization
[m, n]=size(W1);
WW=zeros(m,n);
WW(1:m/2,1:n/2)=W1(1:m/2,1:n/2);
W1=WW;
%Decoding from level-1 using W1
[m, n]=size(W1);
for i=1:n
    Yd1(:,i)=idwt(h0,double(W1(:,i)),1);
end
for i=1:m
    Yd1(i,:)=idwt(h0,double(Yd1(i,:)),1)';
end
YY1=uint8(Yd1);
Figure (2),imshow(Y);xlabel('Original image');
Figure (3),imshow(YY1);xlabel('4:1 Compression');
```

Program 13.5. Two-level wavelet compression.

```
close all; clear all; clc
X=imread('cruise','JPEG');
Y=rgb2gray(X);
h0=[0.054415842243144 0.312871590914520 0.675630736297712 ...
  0.585354683654425-0.015829105256675 -0.284015542962009 ...
  0.000472484573805 0.128747426620538-0.017369301001845 ...
  -0.044088253930837 0.013981027917411 0.008746094047413 ...
  -0.004870352993456 -0.000391740373377 0.000675449406451 ...
  -0.000117476784125];
M=length(h0);
h1(1:2:M-1)=h0(M:-2:2);h1(2:2:M)=-h0(M-1:-2:1);%Obtain QMF highpass filter
[m n]=size(Y);
% Level-1 transform
[m n]=size(Y);
for i=1:m
    W1(i,:)=dwt(h0,double(Y(i,:)),1)';
end
for i=1:n
    W1(:,i)=dwt(h0,W1(:,i),1); % Wavelet coefficients at level-1 transform
end
% Level-2 transform
Y1=W1(1:m/2,1:n/2); %Obtain LL subband
[m n]=size(Y1);
for i=1:m
    W2(i,:)=dwt(h0,Y1(i,:),1)';
end
for i=1:n
W2(:,i)=dwt(h0,W2(:,i),1);
end
W22=W1; W22(1:m,1:n)=W2; % Wavelet coefficients at level-2 transform
wmax=max(max(abs(W22)));
% 8-bit Quantization
W22=round(W22*2^7/wmax);
W22=double(W22)*wmax/2^7;
Figure(1), imshow(uint8(W22));xlabel('Wavelet coefficients');
[m, n]=size(W22); WW=zeros(m,n);
WW(1:m/4,1:n/4)=W22(1:m/4,1:n/4);
W22=WW; %Discard HL2,LH2, HH2, HL1, LH1, HH1 subbands
% Decoding from Level-2 transform
[m,n]=size(W22); Wd2=W22(1:m/2,1:n/2);
% Level-2
[m n]=size(Wd2);
for i=1:n
    Wd1(:,i)=idwt(h0,double(Wd2(:,i)),1);
end
for i=1:m
    Wd1(i,:)=idwt(h0,double(Wd1(i,:))',1);
end
% Level-1
```

```
[m, n]=size(W22);Yd11=W22;
Yd11(1:m/2,1:n/2)=Wd1;
for i=1:n
Yd(:,i)=idwt(h0,Yd11(:,i),1);
end
for i=1:m
Yd(i,:)=idwt(h0,double(Yd(i,:)),1)';
end
Figure(2),imshow(Y),xlabel('Original image');
Y11=uint8(Yd); Fig. (3),imshow(Y11);xlabel('16:1 compression');
```

13.8 CREATING A VIDEO SEQUENCE BY MIXING TWO IMAGES

In this section, we introduce a method to mix two images to generate an image (video) sequence. Applications of mixing the two images may include fading in and fading out images, blending two images, or overlaying text on an image.

In mixing two images in a video sequence, a smooth transition is required from fading out one image of interest to fading in another image of interest. We want to fade out the first image and gradually fade in the second. This cross-fade scheme is implemented using the following operation:

$$\text{Mixed image} = (1-\alpha) \times \text{image}_1 + \alpha \times \text{image}_2 \tag{13.23}$$

where α = fading in proportionally to the weight of the second image (value between 0 and 1), and $(1-\alpha)$ = fade out proportionally to the weight of the second image.

The video sequence in Fig. 13.44A consisting of six frames is generated using $\alpha=0$, $\alpha=0.2$, $\alpha=0.4$, $\alpha=0.6$, $\alpha=0.8$, $\alpha=1.0$, respectively, for two images. The equations for generating these frames are listed as follows:

$$\text{Mixed image}_1 = 1.0 \times \text{image}_1 + 0.0 \times \text{image}_2$$
$$\text{Mixed image}_2 = 0.8 \times \text{image}_1 + 0.2 \times \text{image}_2$$
$$\text{Mixed image}_3 = 0.6 \times \text{image}_1 + 0.4 \times \text{image}_2$$
$$\text{Mixed image}_4 = 0.4 \times \text{image}_1 + 0.6 \times \text{image}_2$$
$$\text{Mixed image}_5 = 0.2 \times \text{image}_1 + 0.8 \times \text{image}_2$$
$$\text{Mixed image}_6 = 0.0 \times \text{image}_1 + 1.0 \times \text{image}_2.$$

The sequence begins with the Grand Canyon image and fades in with the cruise ship image. At frame 4, the 60% of the cruise ship is faded in, and the image begins to be discernible as such. The sequence ends with the cruise ship in 100% fade-in. Fig. 13.44A displays the generated grayscale sequence. Fig. 13.44B shows the color RGB video sequence.

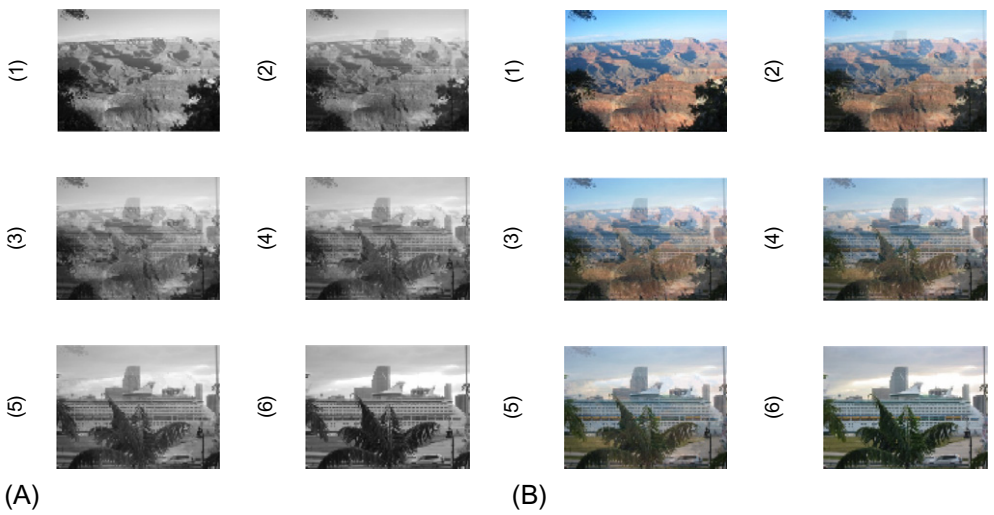


FIG. 13.44

A Grayscale video sequence.

13.9 VIDEO SIGNAL BASICS

Video signals generally can be classified as component video, composite video, and S-video.

In *component video*, three video signals—such as the RGB channels or the Y, I, and Q channels—are used. Three wires are required for connection to a camera or TV. Most computer systems use component video signals. *Composite video* has the intensity (luminance) and two-color (chrominance) components that modulate the carrier wave. This signal is used in the broadcast color TV. The standard by the US-based National Television System Committee (NTSC) combines channel signals into a chroma signal, which is modulated to a higher frequency for transmission. Connecting TVs or VCRs requires only one wire, since both video and audio are mixed into the same signal. *S-video* sends luminance and chrominance separately, since the luminance presenting black-and-white intensity contains most of the signal information for visual perception.

13.9.1 ANALOG VIDEO

In computer systems, progressive scanning traces a whole picture, called *frame* via *row wise*. A higher-resolution computer uses 72 frames per second (fps). The video is usually played at a frame rate varying from 15 to 30 frames.

In TV reception and some monitors, *interlaced scanning* is used in a cathode-ray tube display, or raster. The odd-numbered lines are traced first, and the even-numbered lines are traced next. We then get the odd-field and even-field scans per frame. The interlaced scheme is illustrated in Fig. 13.45, where the odd lines are traced, such as A to B, then C to D, and so on, ending in the middle at E. The even field begins at F in the middle of the first line of the even field and ends at G. The purpose

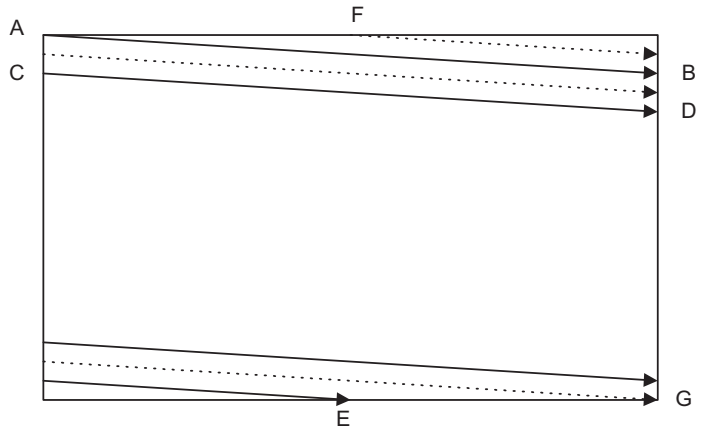


FIG. 13.45

Interlaced raster scanning.

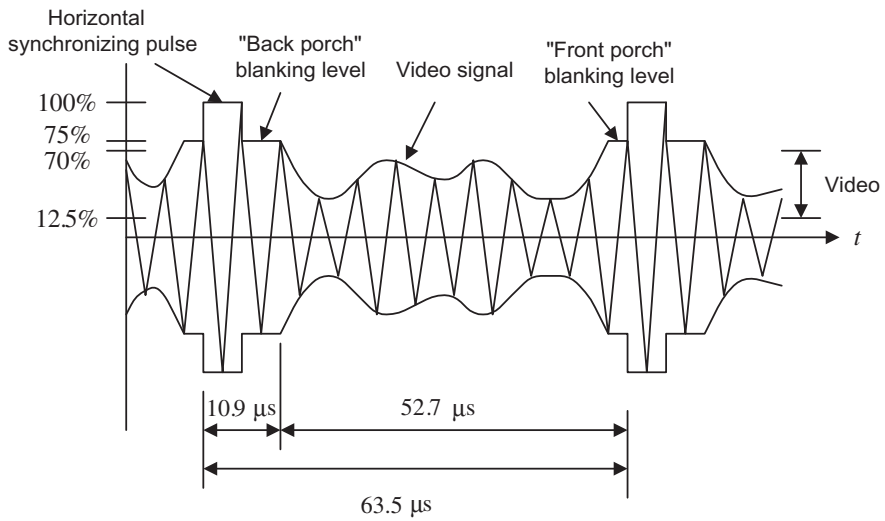


FIG. 13.46

Video-modulated waveform.

of using interlaced scanning is to transmit a full frame quickly to reduce flicker. Trace jumping from B to C is called horizontal retrace, while trace jumping from E to F or G to A is called vertical retrace.

The video signal is amplitude modulated. The modulation levels for NTSC video are shown in Fig. 13.46. In the United States, negative modulation is used, considering that less amplitudes come from a brighter scene, while more amplitudes come from a darker one. This is due to the fact that most pictures contain more white levels than black levels. With negative modulation, possible power efficiency can be achieved for transmission. The reverse process will apply for display at the receiver.

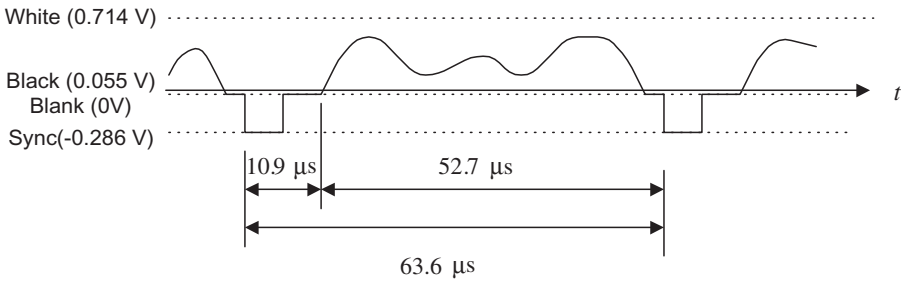


FIG. 13.47

The demodulated signal level for one NTSC scan line.

The horizontal synchronizing pulse controls the timing for the horizontal retrace. The blanking levels are used for synchronizing as well. The "back porch" (Fig. 13.46) of the blanking also contains the color subcarrier burst for the color demodulation.

The demodulated electrical signal can be seen in Fig. 13.47, where a typical electronic signal for one scan line is depicted. The white intensity has a peak value of 0.714 V, and the black has voltage level of 0.055 V, which is close to zero. The blank corresponds to 0 V, and the synchronizing pulse is at the level of −0.286 V. The time duration for synchronizing is 10.9 μs; that of the video occupies 52.7 μs; and that of one entire scan line occupies 63.6 μs. Hence, the line scan rate can be determined as 15.75 kHz.

Fig. 13.48 describes vertical synchronization. A pulse train is generated at the end of each filed. The pulse train contains six equalizing pulses, six vertical synchronizing pulses, and another six equalizing pulses at the rate of twice the size of the line scan rate (31.5 kHz), so that the timing for sweeping half the width of the field is feasible. In NTSC, the vertical retrace takes the time interval of 20 horizontal lines designated for control information at the beginning of each field. The 18 pulses of the vertical blanking occupy the time interval that is equivalent to nine lines. This leaves lines 10–20 for other uses.

A color subcarrier resides the back porch, as shown in Fig. 13.48. The eight cycles of the color subcarrier are recovered via a delayed gating circuit trigged by the horizontal sync pulse. Synchronization includes the color burst frequency and phase information. The color subcarrier is then applied to demodulate the color (chrominance).

Let us summarize NTSC video signals. The NTSC TV standard uses an aspect ratio of 4:3 (ratio of picture width to height), and 525 scan lines per frame at 30 fps. Each frame has an odd field and an even field. So there are 525/2 = 262.5 lines per field. NTSC actually uses 29.97 fps. The horizontal sweep frequency is 525 × 29.97 = 15,734 lines per second, and each line takes 1/15,734 = 63.6 μ sec. Horizontal retrace takes 10.9 μ sec, while the line signal takes 52.7 μ sec. For one line of image display. Vertical retrace and sync are also needed so that the first 20 lines for each field are reserved to be used. The active video lines per frame are 485. The layout of the video data, retrace, and sync data is shown in Fig. 13.49.

Blanking regions can be used for V-chip information, stereo audio channel data, and subtitles in various languages. The active line then is sampled for display. A pixel clock divides each horizontal line of video into samples. For example, vertical helical scan (VHS) uses 240 samples per line, Super VHS, 400–425 samples per line.

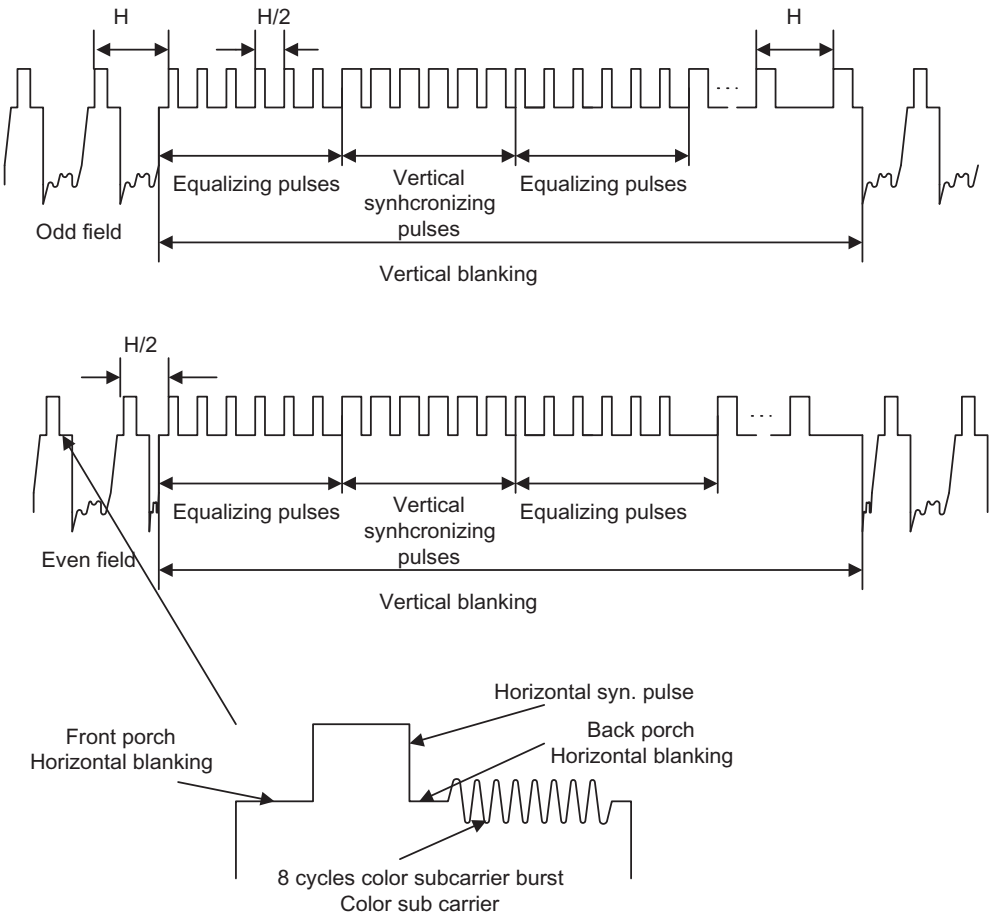


FIG. 13.48

Vertical synchronization for each field and the color subcarrier burst.

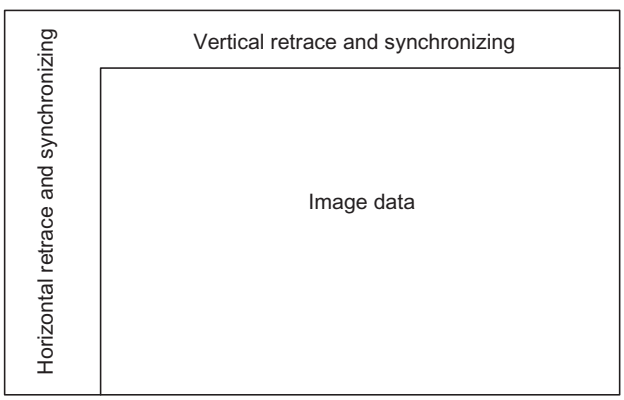


FIG. 13.49

Video data, retrace, and sync layout.

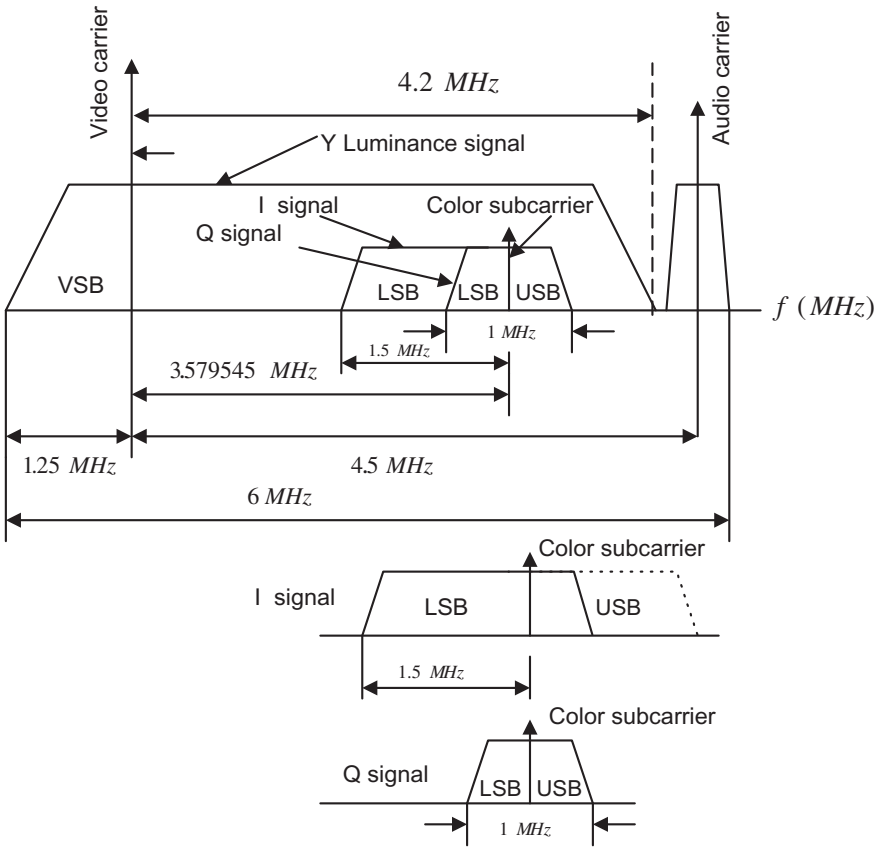


FIG. 13.50

NTSC Y, I, and Q spectra.

Fig. 13.50 shows the NTSC video signal spectra. The NTSC standard assigns a bandwidth of 4.2 MHz for luminance Y, 1.6 MHz for I, and 0.6 for Q, due to the human perception of color information. Since the human eye has higher resolution to the I color component than to the Q color component, the wider bandwidth for I is allowed.

As shown in Fig. 13.50, *vestigial sideband modulation* (VSB) is employed for the luminance, with a picture carrier of 1.25 MHz relative to the VSB left edge. The space between the picture carrier and the audio carrier is 4.5 MHz.

The audio signal containing the frequency range from 50 Hz to 15 kHz is stereo frequency modulated (FM), using a peak frequency deviation of 25 kHz. Therefore, the stereo FM audio requires a transmission bandwidth of 80 kHz, with an audio carrier located at 4.5 MHz relative to the picture carrier.

The color burst carrier is centered at 3.58 MHz above the picture carrier. The two color components I and Q undergo *quadrature amplitude modulation* (QAM) with modulated component I output, which

is VSB filtered to remove the two-thirds of the upper sideband, so that all chroma signals fall within a 4.2 MHz video bandwidth. The color burst carrier of 3.58 MHz is chosen such that the chroma signal and luminance are interleaved in the frequency domain to reduce interference between them.

Generating a chroma signal with QAM gives

$$C = I\cos(2\pi f_{sc}t) + Q\sin(2\pi f_{sc}t), \tag{13.24}$$

where C = chroma component and f_{sc} = color subcarrier = 3.58 MHz. The NTSC signal is further combined into a composite signal:

$$Composite = Y + C = Y + I\cos(2\pi f_{sc}t) + Q\sin(2\pi f_{sc}t). \tag{13.25}$$

At decoding, the chroma signal is obtained by separating Y and C first. Generally, the lowpass filters located at the lower end of the channel can be used to extract Y. The comb filters may be employed to cancel interferences between the modulated luminance signal and the chroma signal (Li et al., 2014). Then we perform demodulation for I and Q as follows:

$$\begin{aligned} C \times 2\cos(2\pi f_{sc}t) &= I2\cos^2(2\pi f_{sc}t) + Q \times 2\sin(2\pi f_{sc}t)\cos(2\pi f_{sc}t) \\ &= I + I \times \cos(2 \times 2\pi f_{sc}t) + Q\sin(2 \times 2\pi f_{sc}t). \end{aligned} \tag{13.26}$$

Applying a lowpass filter yields the I component. Similar operation applying a carrier signal of $2\sin(2\pi f_{sc}t)$ for demodulation recovers the Q component.

***PAL Video*:**

The phase alternative line (PAL) system uses 625 scan lines per frame at 25 fps, with an aspect ratio of 4:3. It is widely used in Western Europe, China, and India. PAL uses the YUV color model, with an 8-MHz channel in which Y has 5.5 MHz and U and V each have 1.8 MHz with the color subcarrier frequency of 4.43 MHz relative to the picture carrier. The U and V are the color difference signals (chroma signals) of the B-Y signal and R-Y signal, respectively. The chroma signals have alternate signs (e.g., +V and − V) in successive scan lines. Hence, the signal and its sign reversed one in the consecutive lines are averaged to cancel out the phase errors that could be displayed as the color errors.

***SECAM Video*:**

The SECAM (Séquentiel Couleur à Mémoire) system uses 625 scan lines per frame at 25 fps, with an aspect ratio of 4:3 and interlaced fields. The YUV color model is employed, and U and V signals are modulated using separate color subcarriers of 4.25 and 4.41 MHz, respectively. The U and V signals are sent on each line alternatively. In this way, the quadrature multiplexing and the possible cross coupling of color signals could be avoided by halving the color resolution in the vertical dimension.

Table 13.15 lists the frame rates, numbers of scan lines, and bandwidths for each analog broadband TV systems.

13.9.2 DIGITAL VIDEO

Digital video has become dominant over the long-standing analog method in the modern systems and devices because it offers high image quality; flexibility of storage, retrieval, and editing capabilities; digital enhancement of video images; encryption; channel noise tolerance; and multimedia system applications.

Digital video formats are developed by the Consultative Committee for International Radio (CCIR). A most important standard is CCIR-601, which became ITU-R-601, an international standard for professional video applications.

Table 13.15 Analog Broadband TV Systems

TV System	Frame Rate (fps)	Number of Scan Lines	Total Bandwidth (MHz)	Y Bandwidth (MHz)	U or I Bandwidth (MHz)	V or Q Bandwidth (MHz)
NTSC	29.97	525	6.0	4.2	1.6	0.6
PAL	25	625	8.0	5.5	1.8	1.8
SECAM	25	625	8.0	6.0	2.0	2.0

Source: Li, Z.N., Drew, M.S., Liu, J., 2014. Fundamentals of Multimedia, second ed. Springer.

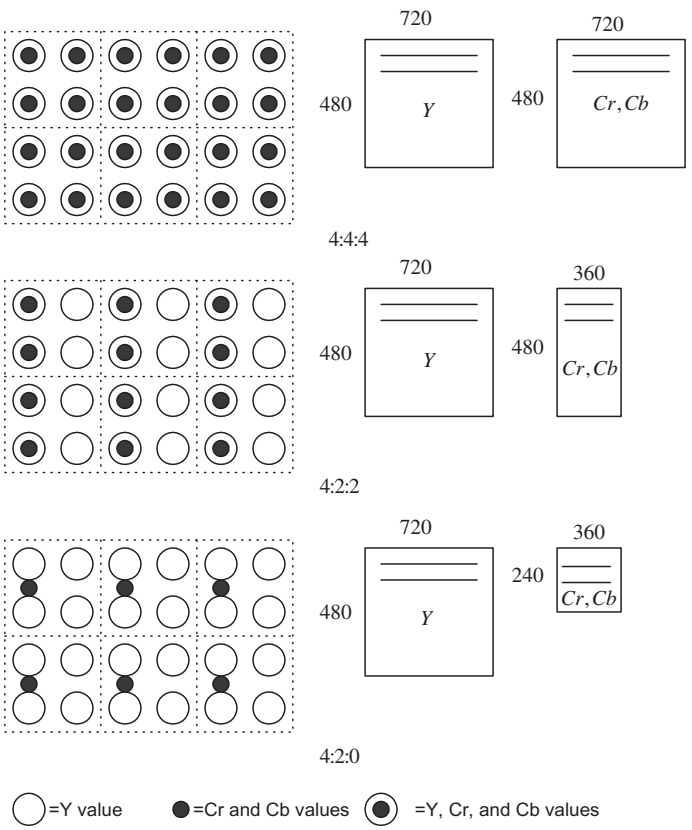


FIG. 13.51

Chroma subsampling.

In CCIR-601, chroma subsampling is carried for digital video. Each pixel is in YCbCr color space, where Y is the luminance, and Cb and Cr are the chrominance. Subsampling schemes include 4:4:4 (no chroma subsampling); 4:2:2, 4:1:1, and 4:2:0 as illustrated in Fig. 13.51.

In a 4:4:4 video format in each frame, the number of values for each chrominance component, Cb or Cr, is the same as that for luminance, Y, both horizontally and vertically. This format finds applications

Table 13.16 Digital Video Specifications

	CCIR 601 525/60 NTSC	CCR 601 625/50 PAL/SECAM	CIF	QCIF
Luminance resolution	720 × 480	720 × 576	352 × 288	176 × 144
Chrominance resolution	360 × 480	360 × 576	176 × 144	88 × 72
Color subsampling	04:02:02	04:02:02	04:02:00	04:02:00
Aspect ratio	04:03	04:03	04:03	04:03
Fields/secs	60	50	30	30
Interlaced	Yes	Yes	No	No

CCR, comparison category rating; CIF, common intermediate format; QCIF, quarter-CIF.
Source: Li, Z.N., Drew, M.S., Liu, J., 2014. Fundamentals of Multimedia, second ed., Springer.

in the computer graphics, in which the chrominance resolution is required for both horizontal and vertical dimension. The format is not widely used in video applications due to a huge storage requirement.

As shown in Fig. 13.51, for each frame in the 4:2:2 video format, the number of chrominance components for Cr or Cb is half the number of luminance components for Y. The resolution is full vertically, and the horizontal resolution is downsampled by a factor of 2. Considering the first line of six pixels, transmission occurs in the following form: (Y0, Cb0), (Y1, Cr0), (Y2,Cb2), (Y3,Cr2), (Y4,Cb4), (Y5, Cr4), and so on. Six Y values are sent for every two Cb and Cr values that are sent.

In the 4:2:0 video format, the number of values for each chrominance Cb and Cr is half the number of luminance Y for both horizontal and vertical directions. That is, the chroma is downsampled horizontally and vertically by a factor of 2. The location for both Cb and Cr is shown in Fig. 13.51. Digital video specifications are given in Table 13.16.

CIF was specified by the Comité Consultatif International Téléphonique et Télégraphique (CCITT), which is now the International telecommunications Union (ITU). CIF produces a low-bit rate video and supports a progressive scan. QCIF achieves an even lower bit rate video. Neither formats supports the interlaced scan mode.

Table 13.17 outlines the high-definition TV (HDTV) formats supported by the Advanced Television System Committee (ATSC), where "I" means interlaced scan and "P" indicates progressive scan. MPEG compressions of video and audio are employed.

13.10 MOTION ESTIMATION IN VIDEO

In this section, we study motion estimation since this technique is widely used in the MPEG video compression. A video contains a time-ordered sequence of frames. Each frame consists of image data. When the objects in an image are still, the pixel values do not change under constant lighting condition. Hence, there is no motion between the frames. However, if the objects are moving, then the pixels are moved. If we can find the motions, which are the pixel displacements, with *motion vectors*, the frame

Table 13.17 High-Definition TV (HDTV) Formats

Number of Active Pixels per Line	Number of Active Lines	Aspect Ratio	Picture Rate
1920	1080	16:09	60I 30P 24P
1280	720	16:09	60P 30P 24 P
704	480	16:9 and 4:3	60I 60P 30P 24P
640	480	4:03	60I 60P 30P 24P

Source: Li, Z.N., Drew, M.S., Liu, J., 2014. Fundamentals of Multimedia, second ed., Springer.

data can be recovered from the reference frame by copying and pasting at locations specified by the motion vector. To explore such an idea, let us look at Fig. 13.52.

As shown Fig. 13.52, the reference frame is displayed first, and the next frame is the target frame containing a moving object. The image in the target frame is divided into N × N macroblocks (20 macroblocks). A macroblock match is searched within the search window in the reference frame to find the closest match between a macroblock under consideration in the target frame and the macroblock in the reference frame. The differences between two locations (motion vectors) for the matched macroblocks are encoded.

The criteria for searching the best matching can be chosen using the mean absolute difference (MAD) between the reference frame and the target frame:

$$MAD(i,j)=\frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}|T(m+k,n+l)-R(m+k+i,n+l+j)| \tag{13.27}$$

$$u=i,v=j for\, MAD(i,j)=\text{minimum, and}-p\le i,j\le p. \tag{13.28}$$

There are many search methods for finding the motion vectors, including optimal sequential or brute force searches; and suboptimal searches such as 2D-logarithmic and hierarchical searches. Here we examine the sequential search to understand the basic idea.

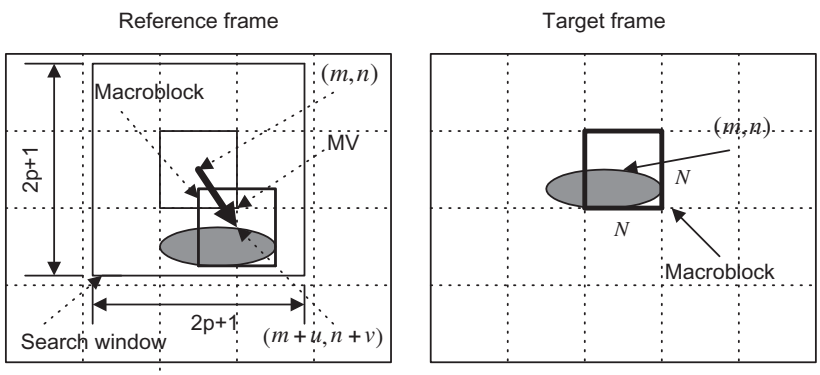


FIG. 13.52

Macroblocks and motion vectors in the reference frame and target frame.

The sequential search for finding the motion vectors employs methodical "brute force" to search the entire $(2p+1) \times (2p+1)$ search window in the reference frame. The macroblock in the target frame compares each macroblock centered at each pixel in the search window of the reference frame. Comparison using Eq. (13.27) proceeds pixel by pixel to find the best match in which the vector (i, j) produces the smallest MAD. Then the motion vector (MV(u,v)) is found to be $u=i$, and $v=j$. The algorithm is described as follows:

```
min_MAD=large value
for i=-p, ..., p
  for j=-p,...,p
    cur_MAD=MDA(i,j);
    if cur_MAD < min_MAD
        min_MAD= cur_MAD;
        u=i;
        v=j;
   end
  end
end
```

The sequential search provides best matching with the least MAD. However, it requires a huge amount of computations. Other suboptimal search methods can be employed to reduce the computational requirement, but with sacrifices of image quality. These topics are beyond our scope.

EXAMPLE 13.17

An 80×80 reference frame, target frame, and their difference are displayed in Fig. 13.53. A macroblock with a size of 16×16 is used, and the search window has a size of 32×32. The target frame is obtained by moving the reference frame to the right by 6 pixels and to the bottom by 4 pixels. The sequential search method is applied to find all the motion vectors. The reconstructed target frame using the motion vectors and reference image is given in Fig. 13.53.

Since $80 \times 80/(16 \times 16) = 25$, there are 25 macroblocks in the target frame and 25 motion vectors in total. The motion vectors are found to be:

Horizontal direction =

$$-6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6$$
$$-6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6\ -6$$

Vertical direction =

$$-4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4$$
$$-4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4\ -4$$

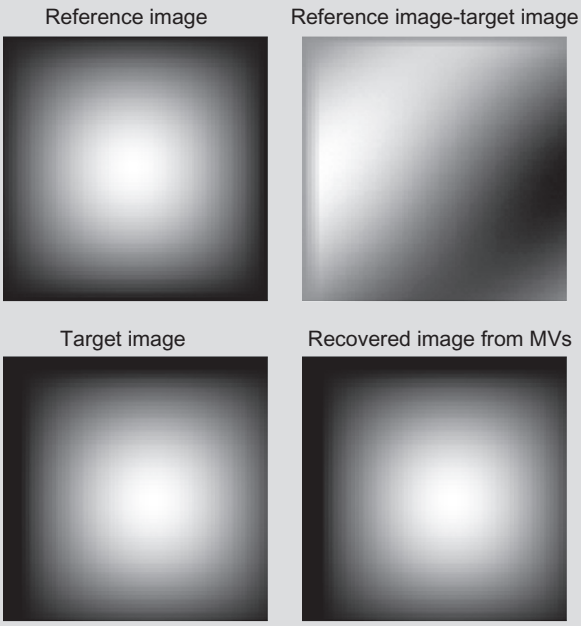


FIG. 13.53

Reference frame, target frame, their difference, and the reconstructed frame by the motion vectors.

The motion vector comprises the pixel displacements from the target frame to the reference frame. Hence, given the reference frame, directions specified in the motion vector should be switched to indicate the motion toward the target frame. As indicated by the obtained motion vectors, the target image is the version of the reference image moving to the right by 6 pixels and down by 4 pixels.

13.11 SUMMARY

1. A digital image consists of pixels. For a grayscale image, each pixel is assigned a grayscale level that presents luminance of the pixel. For an RGB color image, each pixel is assigned a red component, a green component, and a blue component. For an indexed color image, each pixel is assigned an address that is the location of the color table (map) made up of the RGB components.
2. Common image data formats are 8-bit grayscale image, 24-bit color, and 8-bit indexed color.
3. The larger the number of pixels in an image, or the larger the numbers of the RGB components, the finer is the spatial resolution in the image. Similarly, the more scale levels used for each pixel, the better the scale-level image resolution. The more pixels and more bits used for the scale levels in the image, the more storage is required.

4. The RGB color pixel can be converted to YIQ color pixels. Y component is the luminance occupying 93% of the signal energy, while the I and Q components represent the color information of the image, occupying the remainder of the energy.
5. The histogram for a grayscale image shows the number of pixels at each grayscale level. The histogram can be modified to enhance the image. Image equalization using the histogram can improve the image contrast and effectively enhances contrast for image underexposure. Color image equalization can be done only in the luminance channel or RGB channels.
6. Image enhancement techniques such as average lowpass filtering can filter out random noise in the image; however, it also blurs the image. The degree of blurring depends on the kernel size. The bigger the kernel size, the more blurring occurs.
7. Median filtering effectively remove the "salt and pepper" noise in an image.
8. The edge-detection filter with Sobel convolution, Laplacian, and Laplacian of Gaussian kernels can detect the image boundaries.
9. The grayscale image can be made into a facsimile of the color image by pseudo-color image generation, using the RGB transformation functions.
10. RGB-to-YIQ transformation is used to obtain the color image in YIQ space, or vice versa. It can also be used for color-to-grayscale conversion, that is, keep only the luminance channel after the transformation.
11. 2D spectra can be calculated and are used to examine filtering effects.
12. JPEG compression uses the 2D-DCT transform for both grayscale and color images in YIQ color space. JPEG uses different quality factors to normalize DCT coefficients for the luminance (Y) channel and the chrominance (IQ) channels.
13. The mixing two images, in which two pixels are linearly interpolated using the weights $1-\alpha$ and α, can produce video sequences that have effects such as fading in and fading out of images, blending of two images, and overlaying of text on an image.
14. Analog video uses interlaced scanning. A video frame contains odd and even fields. Analog video standards include NTSC, PAL, and SECAM.
15. Digital video carries the modulated information for each pixel in YCbCr color space, where Y is the luminance and Cb and Cr are the chrominance. Chroma subsampling creates various digital video formats. The industrial standards include CCIR601, CCR601, CIF, and QCIF.
16. The motion compensation of a video sequence produces motion vector for all the image blocks in the target video frame, which contain displacements of these image blocks relative to the reference video frame. Recovering the target frame involves simply copying each image block of the reference frame to the target frame at the location specified in the motion vector. Motion compensation is a key element in MPEG video.

13.12 PROBLEMS

13.1 Determine the memory storage requirement for each of the following images:
- **(a)** 320×240 8-bit grayscale.
- **(b)** 640×480 24-bit color image.
- **(c)** 1600×1200 8-bit indexed image.

13.2 Determine the number of colors for each of the following images:
(a) 320 × 240 16-bit indexed image.
(b) 200 × 100 24-bit color image.

13.3 Given a pixel in an RGB image as follows:

$$R = 200, G = 120, B = 100,$$

convert the RGB values to the YIQ values.

13.4 Given a pixel of an image in YIQ color format as follows:

$$Y = 141, I = 46, Q = 5,$$

convert the YIQ values back to RGB values.

13.5 Given the following 2 × 2 RGB image,

$$R = \begin{bmatrix} 100 & 50 \\ 100 & 50 \end{bmatrix} G = \begin{bmatrix} 20 & 40 \\ 10 & 30 \end{bmatrix} B = \begin{bmatrix} 100 & 50 \\ 200 & 150 \end{bmatrix},$$

convert the image into grayscale.

13.6 Produce a histogram of the following image, which has a grayscale value ranging from 0 to 7, that is, each pixel is encoded in 3 bits.

$$\begin{bmatrix} 0 & 1 & 2 & 2 & 0 \\ 2 & 1 & 1 & 2 & 1 \\ 1 & 1 & 4 & 2 & 3 \\ 0 & 2 & 5 & 6 & 1 \end{bmatrix}$$

13.7 Given the following image with a grayscale value ranging from 0 to 7, that is, each pixel being encoded in 3 bits,

$$\begin{bmatrix} 0 & 1 & 2 & 2 & 0 \\ 2 & 1 & 1 & 2 & 1 \\ 1 & 1 & 4 & 2 & 3 \\ 0 & 2 & 5 & 6 & 1 \end{bmatrix},$$

perform equalization using the histogram in Problem 13.6, and plot the histogram for the equalized image.

13.8 Given the following image with a grayscale value ranging from 0 to 7, that is, each pixel being encoded in 3 bits,

$$\begin{bmatrix} 2 & 4 & 4 & 2 \\ 2 & 3 & 3 & 3 \\ 4 & 4 & 4 & 2 \\ 3 & 2 & 3 & 4 \end{bmatrix},$$

perform level adjustment to the full range, shift the level to the range from 3 to 7, and shift the level to the range from 0 to 3.

13.9 Given the following 8-bit grayscale original and noisy images, and 2×2 convolution average kernel,

$$4 \times 4 \text{ original image}: \begin{bmatrix} 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \end{bmatrix}$$

$$4 \times 4 \text{ corrupted image}: \begin{bmatrix} 93 & 116 & 109 & 96 \\ 92 & 107 & 103 & 108 \\ 84 & 107 & 86 & 107 \\ 87 & 113 & 106 & 99 \end{bmatrix}$$

$$2 \times 2 \text{ average kernel}: \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

perform digital filtering on the noisy image, and compare the enhanced image with the original image.

13.10 Given the following 8-bit grayscale original and noisy image, and 3×3 median filter kernel,

$$4 \times 4 \text{ original image}: \begin{bmatrix} 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \end{bmatrix}$$

$$4 \times 4 \text{ corrupted image by impulse noise}: \begin{bmatrix} 100 & 255 & 100 & 100 \\ 0 & 255 & 255 & 100 \\ 100 & 0 & 100 & 0 \\ 100 & 255 & 100 & 100 \end{bmatrix}$$

$$3 \times 3 \text{ average kernel}: \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix},$$

perform digital filtering, and compare the filtered image with the original image.

13.11 Given the following 8-bit 5×4 original grayscale image,

$$\begin{bmatrix} 110 & 110 & 110 & 110 \\ 110 & 100 & 100 & 110 \\ 110 & 100 & 100 & 110 \\ 110 & 110 & 110 & 110 \\ 110 & 110 & 110 & 110 \end{bmatrix},$$

apply the following edge detectors to the image:
(a) Sobel vertical edge detector.
(b) Laplacian edge detector.

and scale the resultant image pixel value to the range of 0–255.

13.12 In Example 13.10, if we switch the transformation functions between the red function and the green function, what is the expected color for the area pointed to by the arrow, and what is the expected background color?

13.13 In Example 13.10, if we switch the transformation functions between the red function and the blue function, what is the expected color for the area pointed to by the arrow, and what is the expected background color?

13.14 Given the following grayscale image $p(i,j)$:

$$\begin{bmatrix} 100 & -50 & 10 \\ 100 & 80 & 100 \\ 50 & 50 & 40 \end{bmatrix},$$

determine the 2D-DFT coefficient $X(1,2)$ and the magnitude spectrum $A(1,2)$.

13.15 Given the following grayscale image $p(i,j)$:

$$\begin{bmatrix} 10 & 100 \\ 200 & 150 \end{bmatrix},$$

determine the 2D-DFT coefficients $X(u,v)$ and magnitude $A(u,v)$.

13.16 Given the following grayscale image $p(i,j)$:

$$\begin{bmatrix} 10 & 100 \\ 200 & 150 \end{bmatrix},$$

apply the 2D-DCT to determine the DCT coefficients.

13.17 Given the following DCT coefficients $F(u,v)$:

$$\begin{bmatrix} 200 & 10 \\ 10 & 0 \end{bmatrix},$$

apply the inverse 2D-DCT to determine 2D data.

13.18 In JPEG compression, DCT DC coefficients from several blocks are 400, 390, 350, 360, and 370. Use DPCM to produce the DPCM sequence, and use the Huffman table to encode the DPCM sequence.

13.19 In JPEG compression, DCT coefficients from the an image subblock are

$$[175, -2, 0, 0, 0, 4, 0, 0, -37, 0, 0, 0, 0, -2, 0, 0, \ldots, 0].$$

(a) Generate the run-length codes for AC coefficients.
(b) Perform entropy coding for the run-length codes using the Huffman table.

13.20 Given the following grayscale image $p(i,j)$:

$$\begin{bmatrix} 10 & 100 \\ 200 & 150 \end{bmatrix},$$

apply the 2D-DWT using the Haar wavelet to determine the level-1 DWT coefficients.

13.21 Given the following level-1 IDWT coefficients $W(u,v)$ obtained using the Harr wavelet:

$$\begin{bmatrix} 200 & 10 \\ 10 & 0 \end{bmatrix},$$

apply the IDWT to determine 2D data.

13.22 Given the following grayscale image $p(i,j)$:

$$\begin{bmatrix} 100 & 150 & 60 & 80 \\ 80 & 90 & 50 & 70 \\ 110 & 120 & 100 & 80 \\ 90 & 50 & 40 & 90 \end{bmatrix},$$

apply the 2D-DWT using the Haar wavelet to determine the level-1 DWT coefficients.

13.23 Given the following level-1 IDWT coefficients $W(u,v)$ obtained using the Harr wavelet:

$$\begin{bmatrix} 250 & 50 & -30 & -20 \\ 30 & 20 & 10 & -20 \\ 10 & 20 & 0 & 0 \\ 20 & 15 & 0 & 0 \end{bmatrix},$$

apply the IDWT to determine 2D data.

13.24 Explain the difference between horizontal retrace and vertical retrace. Which one would take more time?

13.25 What is the purpose of using the interlaced scanning in the traditional NTSC TV system?

13.26 What is the bandwidth in the traditional NTSC TV broadcast system? What is the bandwidth to transmit luminance Y, and what are the required bandwidths to transmit Q and I channels, respectively?

13.27 What type of modulation is used for transmitting audio signals in the NTSC TV system?

13.28 Given the composite NTSC signal

$$Composite = Y + C = Y + I\cos(2\pi f_{sc}t) + Q\sin(2\pi f_{sc}t),$$

show demodulation for the Q channel.

13.29 Where does the color subcarrier burst reside? What is the frequency of the color subcarrier, and how many cycles does the color burst have?

13.30 Compare differences of the NTSC, PAL, and SECAM video systems in terms of the number of scan lines, frame rates, and total bandwidths required for transmission.

13.31 In the NTSC TV system, what is the horizontal line scan rate? What is the vertical synchronizing pulse rate?

13.32 Explain which the following digital video formats achieves the most data transmission efficiency:
- **(a)** 4:4:4.
- **(b)** 4:2:2.
- **(c)** 4:2:0.

13.33 What is the difference between an interlaced scan and a progressive scan? Which of the following video systems use the progressive scan?
- **(a)** CCIR-601.
- **(b)** CIF.

13.34 In motion compensation, which of the following would require more computation? Explain.
- **(a)** Finding the motion vector using the sequential search
- **(b)** Recovering the target frame with the motion vectors and reference frame

13.35 Given a reference frame and target frame of size 80×80, a macroblock size of 16×16, and a search window size of 32×32, estimate the numbers of subtraction, absolute value, and additions for searching all the motion vectors using the sequential search method.

MATLAB Problems

Use MATLAB to solve Problems 13.36–13.42.

13.36 Given the image data "trees.jpg," use MATLAB functions to perform each of the following processing:
- **(a)** Use MATLAB to read and display the image.
- **(b)** Convert the image to the grayscale image.
- **(c)** Perform histogram equalization for the grayscale image in (b) and display the histogram plots for both original grayscale image and equalized grayscale image.
- **(d)** Perform histogram equalization for the color image in A and display the histogram plots of Y channel for both original color image and equalized color image.

13.37 Given the image data "cruise.jpg," perform the following linear filtering:
- **(a)** Convert the image to grayscale image and then create the 8-bit noisy image by adding Gaussian noise using the following code:

```
noise_image = imnoise(I,'gaussian');
```

where I is the intensity image obtained from normalizing the grayscale image.

- **(b)** Process the noisy image using the Gaussian filter with the following parameters: the convolution kernel size = 4, SIGMA = 0.8, and compare the filtered image with the noisy image.

13.38 Given the image data "cruise.jpg," perform the following filtering process:
- **(a)** Convert the image to grayscale image and then create the 8-bit noisy image by adding "pepper and salt" noise using the following code:

```
noise_image = imnoise(I,'salt & pepper');
```

where I is the intensity image obtained from normalizing the grayscale image.

- **(b)** Process the noisy image using the median filtering with a convolution kernel size of 4×4.

13.39 Given the image data "cruise.jpg," convert the image to the grayscale image and detect the image boundaries using Laplacian of Gaussian filtering with the following parameters:
- **(a)** Kernel size = 4 and SIGMA = 0.9.
- **(b)** Kernel size = 10 and SIGMA = 10.

Compare the results.

13.40 Given the image data "clipim2.gif," perform the following process:
- **(a)** Convert the indexed image to the grayscale image.
- **(b)** Adjust the color transformation functions (sine functions) to make the object indicated by the arrow in the image to be red and the background color to be green.

13.41 Given the image data "cruiseorg.tiff," perform JPEG compression by completing the following steps:
- **(a)** Convert the image to the grayscale image.
- **(b)** Write a MATLAB program for encoding: (1) dividing the image into 8×8 blocks, (2) transforming each block using the DCT, and (3) scaling and rounding DCT coefficients with the standard quality factor. Note that lossless-compressing the quantized DCT coefficients is omitted here for a simple simulation.
- **(c)** Continue to write the MATLAB program for decoding: (1) invert the scaling process for quantized DCT coefficients, (2) perform the inverse DCT for each 8×8 image block, and (3) recover the image.
- **(d)** Run the developed MATLAB program to examine the image quality using.
  - **I.** The quality factor.
  - **II.** The quality factor $\times$ 5.
  - **III.** The quality factor $\times$ 10.

13.42 Given the image data "cruiseorg.tiff," perform wavelet-based compression by completing the following steps:
- **(a)** Convert the image to grayscale image.
- **(b)** Write a MATLAB program for encoding: (1) use 8-tap Daubechies filter coefficients, (2) apply the two-level DWT, and (3) perform 8-bit quantization for subbands LL2, LH2, HL2, HH2, LH1, HL1, HH1 for simulation.
- **(c)** Write the MATLAB program for the decoding process.
- **(d)** Run the developed MATLAB program to examine the image quality using.
  - **I.** Reconstruct the image using all subband coefficients.
  - **II.** Reconstruct the image using LL2 subband coefficients.
  - **III.** Reconstruct the image using LL2, HL2, LH2, and HH2 subband coefficients.
  - **IV.** Reconstruct the image using LL2, HL2, and LH2 subband coefficients.
  - **V.** Reconstruct the image using LL2, HL2, LH2, HH2, LH1, and HL1 subband coefficients.