CHAPTER

Image Processing Basics 14

CHAPTER OUTLINE

Digital Signal Processing. http://dx.doi.org/10.1016/B978-0-12-415893-1.00014-7

OBJECTIVES:

In today's modern computers, media information such as audio, images, and video have become necessary for daily business operations and entertainment. In this chapter, we will study the digital image and its processing techniques. This chapter introduces the basics of image processing, including image enhancement using histogram equalization and filtering methods, and proceeds to study pseudo-color generation for object detection and recognition. Finally, the chapter investigates image compression techniques and the basics of video signals.

14.1 IMAGE PROCESSING NOTATION AND DATA FORMATS

The digital image is picture information in a digital form. The image can be filtered to remove noise to enhance it. 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. Figure 14.1 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

$$p(2, 8) = 86 \tag{14.1}$$

The number of pixels in the presentation of a digital image is its *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 1,600 × 1,200 (1,920,000 pixels = 1.92 megapixels), or as low as 320 × 200 (64,000 pixels = 64 kilopixels). 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. Image quality also depends on the numbers of bits used in encoding each pixel level, which will be discussed in next section.

14.1.1 8-Bit Gray Level Images

If a pixel is encoded on a gray scale from 0 to 255, 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

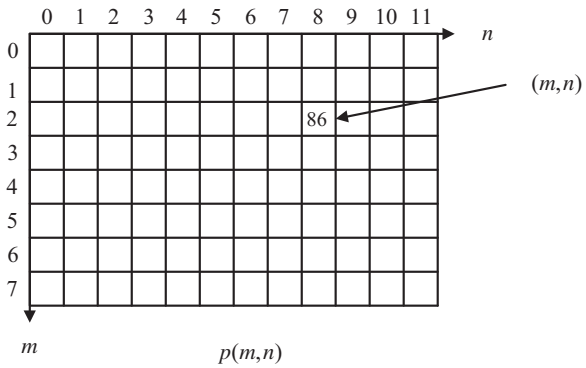


FIGURE 14.1

Image pixel notation.

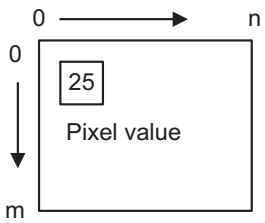


FIGURE 14.2

Grayscale image format.

kilobytes are required for storage. Figure 14.2 shows a grayscale image format. As shown in the figure, the pixel indicated in the box has an 8-bit value of 25.

The image of a cruise ship with a spatial resolution of 320 × 240 using an 8-bit grayscale format is shown in Figure 14.3.

FIGURE 14.3

Grayscale image (8-bit 320 × 240).

14.1.2 24-bit Color Images

In a 24-bit color image representation, each pixel is recoded with red, green, and blue (RGB) components. 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 kilobytes for storage. Figure 14.4 shows the format for the 24-bit color image where the indicated pixel has 8-bit RGB components.

Figure 14.5 shows a 24-bit color image of the Grand Canyon, along with grayscale displays for the 8-bit RGB component images. The full color picture at the upper left is included in the color insert.

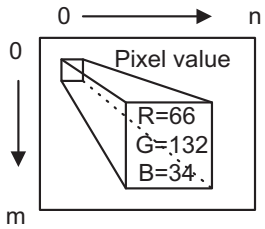


FIGURE 14.4

The 24-bit color image format.

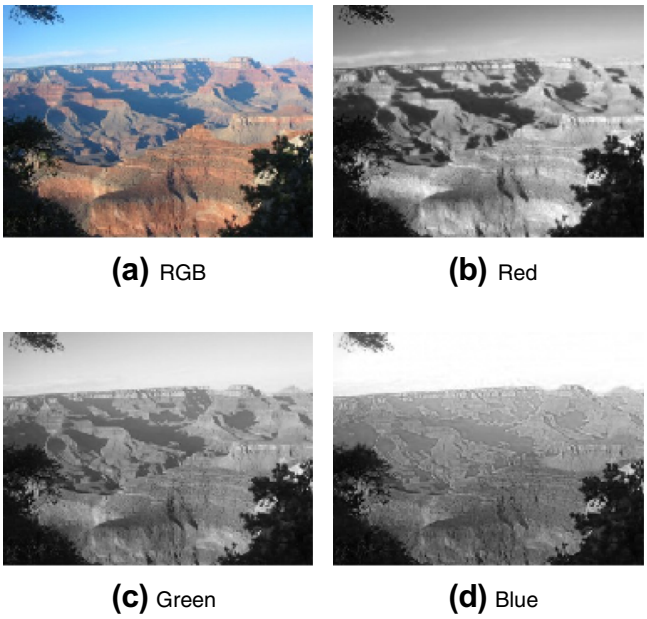


FIGURE 14.5

The 24-bit color image and its respective RGB components. See color image on book's companion site.

14.1.3 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 lookup table containing RGB components. We call this a *color indexed image*, and its format is shown in Figure 14.6. 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 location 5 in the color table, 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 requires 307.2 kilobytes for data storage and $3 \times 256 = 768$ bytes for color map storage. The 8-bit color image for the cruise ship shown in Figure 14.3 is displayed in Figure 14.7.

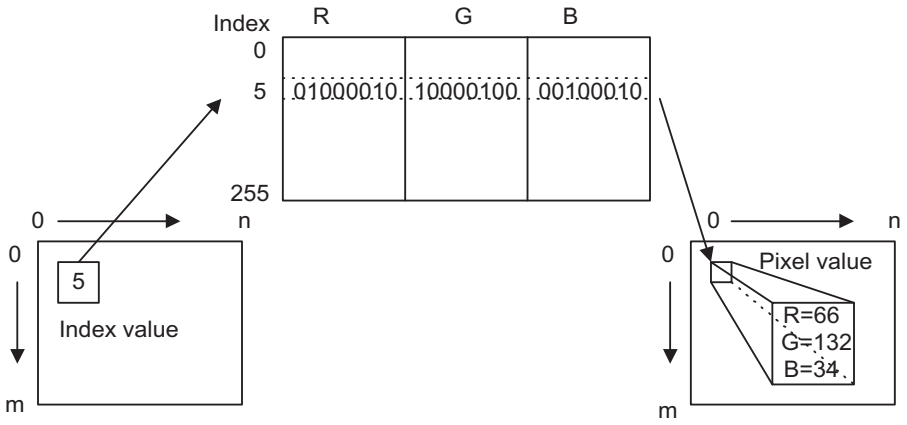


FIGURE 14.6

The 8-bit color indexed image format.

FIGURE 14.7

The 8-bit color indexed image. See color image on book's companion site.

14.1.4 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 of 255 represents white.

In some processing environments such as MATLAB (*matrix laboratory*), floating-point operations are used. The grayscale image has an 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, then scale it back to the standard 8-bit range after processing for display. With the intensity image in floating point format, the digital filter implementation can be easily applied. Figure 14.8 shows the format of the grayscale intensity image, where the indicated pixel shows the intensity value of 0.5988.

14.1.5 Red, Green, and Blue 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 grayscale format in a database system. As another example, in color image compression, the transformation converts the RGB color space to the YIQ color space (Li and Drew, 2004; 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

$$\begin{bmatrix} Y(m,n) \\ I(m,n) \\ Q(m,n) \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R(m,n) \\ G(m,n) \\ B(m,n) \end{bmatrix} \tag{14.2}$$

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

$$\begin{bmatrix} R(m,n) \\ G(m,n) \\ B(m,n) \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y(m,n) \\ I(m,n) \\ Q(m,n) \end{bmatrix} \tag{14.3}$$

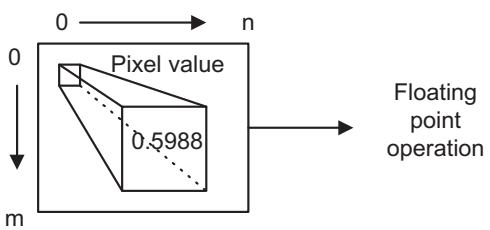


FIGURE 14.8

The grayscale intensity image format.

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

$$Y(m,n) = 0.299 \cdot R(m,n) + 0.587 \cdot G(m,n) + 0.114 \cdot B(m,n) \tag{14.4}$$

Note that $Y(m,n)$, $I(m,n)$, and $Q(m,n)$ can be matrices that represent the luminance image and 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 14.1

Given a pixel in an RGB image

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

convert the pixel values to the YIQ values.

Solution:
Applying Equation (14.2), 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 to

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

Rounding the values to integers, 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 14.2

Given a pixel of an image in the YIQ color format

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

convert the pixel values back to the RGB values.

Solution:
Applying Equation (14.3) 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 14.3

Given the 2 × 2 RGB image

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

convert the RGB color image into a grayscale image.

Solution:

Since only Y components are kept in the grayscale image, we apply Equation (14.4) to each pixel in the 2 x 2 image and round the results to integers as follows:

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

Figure 14.9 shows the grayscale image converted from the 24-bit color image in Figure 14.5 using the RGB-to-YIQ transformation, where only the luminance information is retained.

14.1.6 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

FIGURE 14.9

Grayscale image converted from the 24-bit color image in Figure 14.5 using RGB-to-YIQ transformation.

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
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

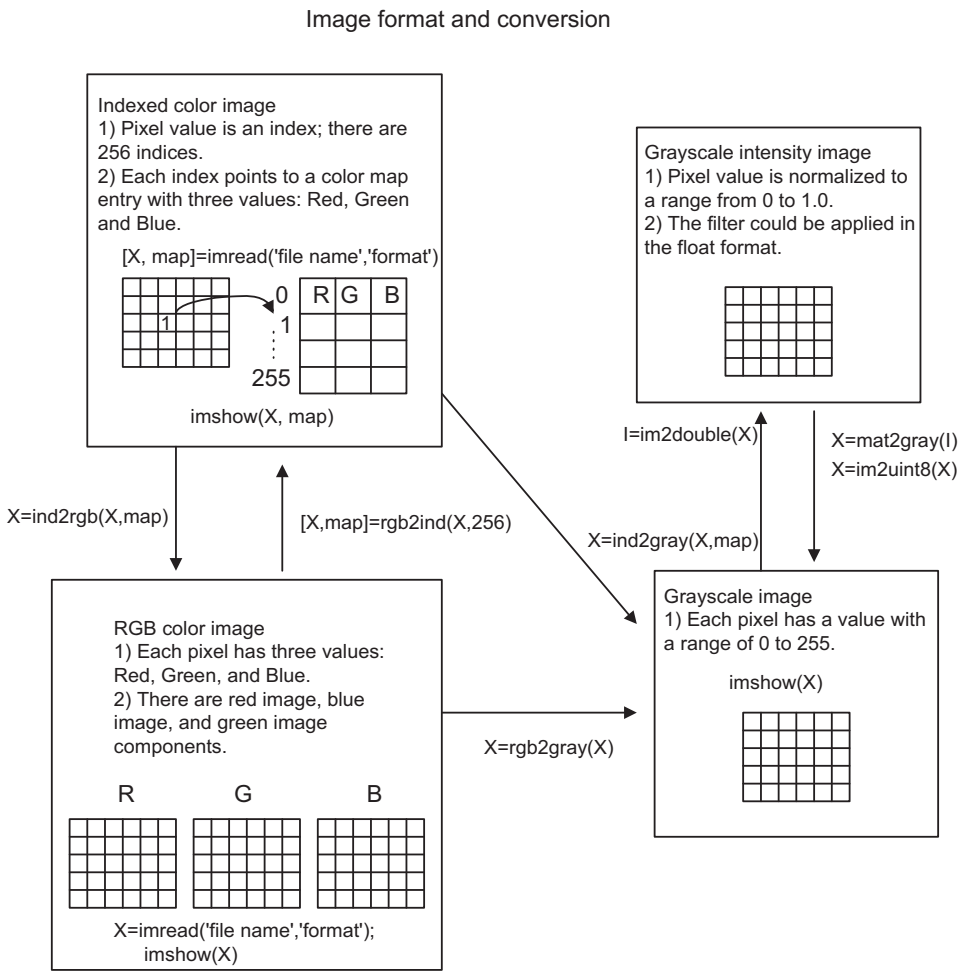


FIGURE 14.10

Outlines the applications of image format conversions.

14.2 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.

14.2.1 Grayscale Histogram and Equalization

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

EXAMPLE 14.4

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, the pixel value ranges from 0 to 7. The count for each grayscale is listed in Table 14.1.

Table 14.1 Pixel Count 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

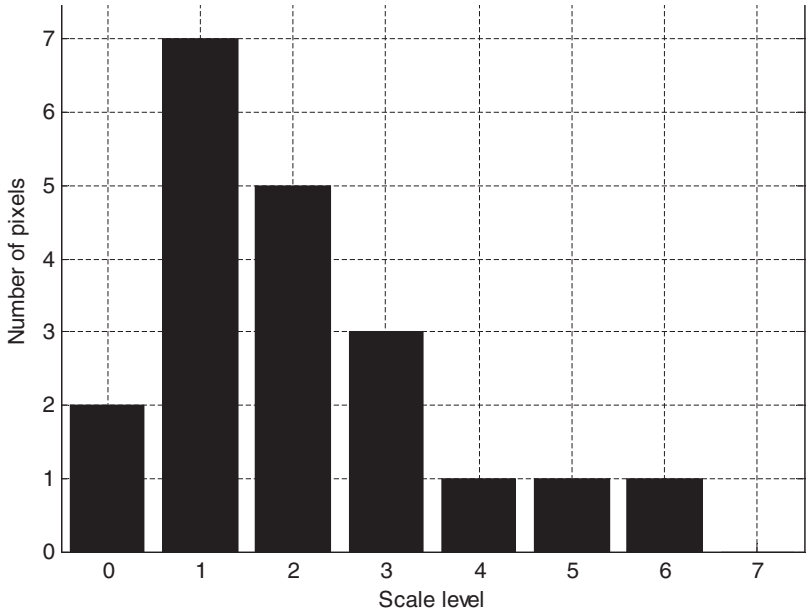


FIGURE 14.11

Histogram in Example 14.4.

Based on the grayscale distribution counts, the histogram is created as shown in Figure 14.11. As we can see, the image has pixels whose levels are more concentrated in the dark scale in this example.

With the histogram, the equalization technique can be developed. Equalization stretches the scale range of the pixel levels to the full range to improve the contrast of the given image. To utilize this technique, the equalized new pixel value is redefined as

$$p_{eq}(m,n) = \frac{\text{Number of pixels with scale level} \leq p(m,n)}{\text{Total number of pixels}} \times (\text{maximum scale level}) \qquad (14.5)$$

The new pixel value is reassigned using the value obtained by multiplying the maximum scale level by the scaled ratio of the accumulative counts up to the current image pixel value over the total number of pixels. Clearly, since the accumulative 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 this 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 14.5

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

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

Table 14.2 Image Equalization in Example 14.5.

Pixel $p(m,n)$ Level	Number of Pixels	Number of Pixels $\leq p(m,n)$	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

Perform equalization using the histogram in Example 14.4, and plot the histogram for the equalized image.

Solution:
Using the histogram result in Table 14.1, we can compute an accumulative count for each grayscale level as shown in Table 14.2. The equalized pixel level using Equation (14.5) 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 Equation (14.5):

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

The equalized image using Table 14.2 is finally obtained by replacing each old pixel value in the old image with its corresponding equalized new pixel value:

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

To see how the histogram is changed, we compute the pixel level counts according to the equalized image. The result is given in Table 14.3, and Figure 14.12 shows the new histogram for the equalized image.

Table 14.3 Pixel Level Distribution Counts of the Equalized Image in Example 14.5

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

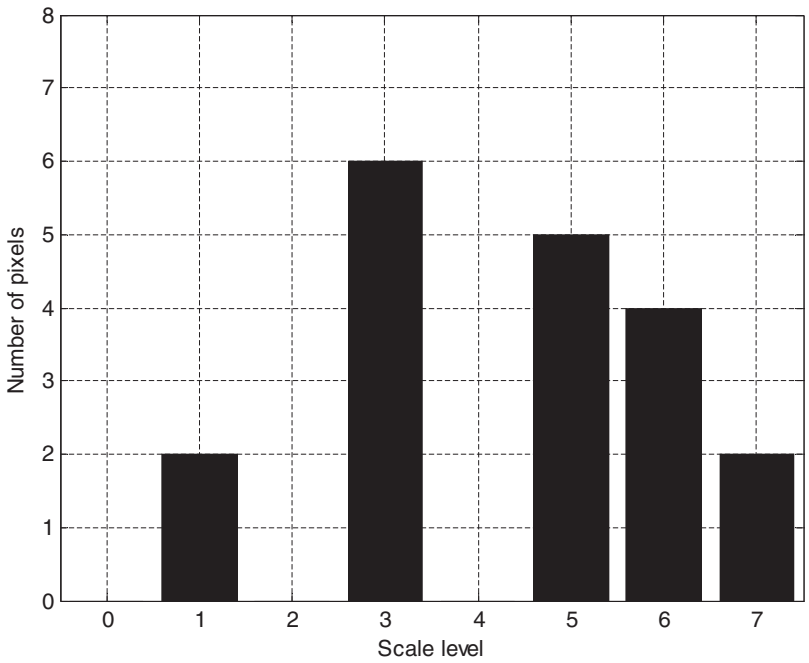


FIGURE 14.12

Histogram for the equalized image in Example 14.5.

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

Next, we apply image histogram equalization to enhance a biomedical image of a human neck in Figure 14.13A, while Figure 14.13B shows the original image histogram. (The purpose of the arrow in Figure 14.13A will be explained later.) 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.

Figure 14.14A and Figure 14.14B show the equalized grayscale image using the histogram method and its histogram, respectively. As shown in the histogram, the equalized pixels reside on a larger scale, and hence the equalized image has improved contrast.

14.2.2 24-Bit Color Image Equalization

For equalizing the RGB image, we first transform RGB values to YIQ values since the 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. Figure 14.15 shows the procedure.

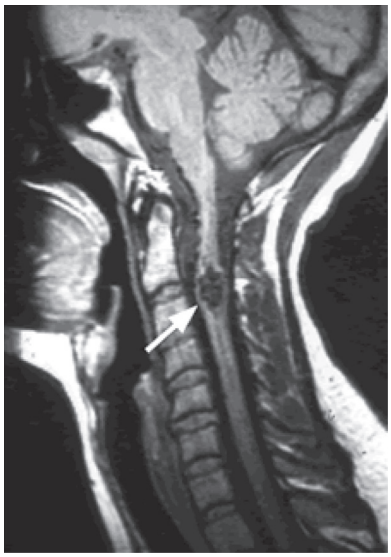


FIGURE 14.13A

Original grayscale image.

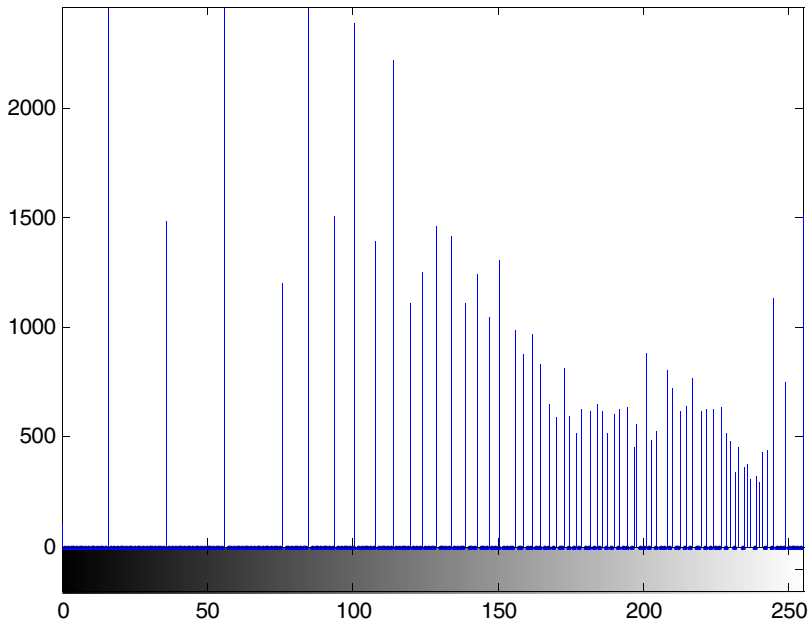


FIGURE 14.13B

Histogram for the original grayscale image.

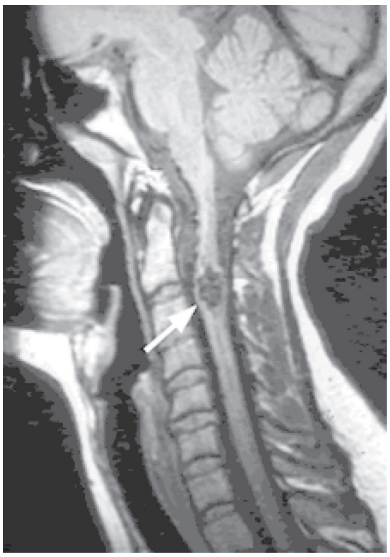


FIGURE 14.14A

Grayscale equalized image.

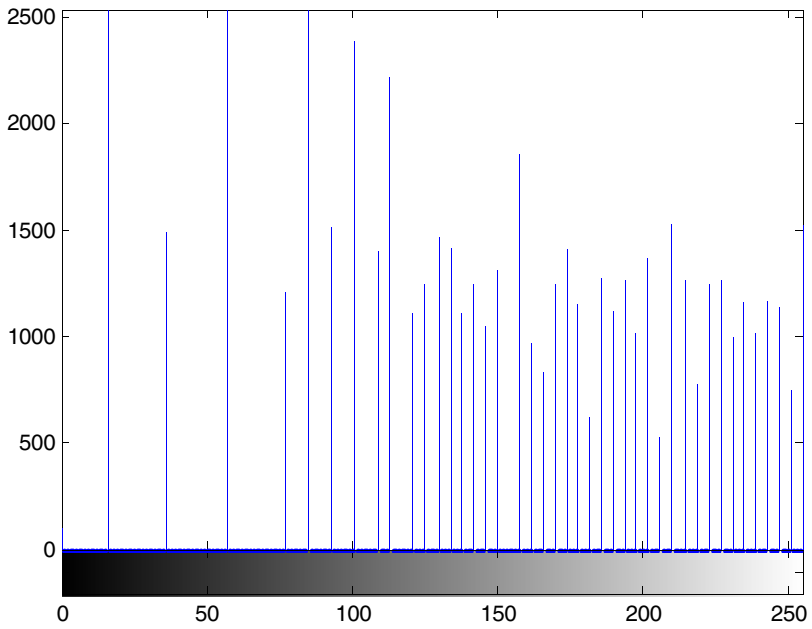


FIGURE 14.14B

Histogram for the grayscale equalized image.

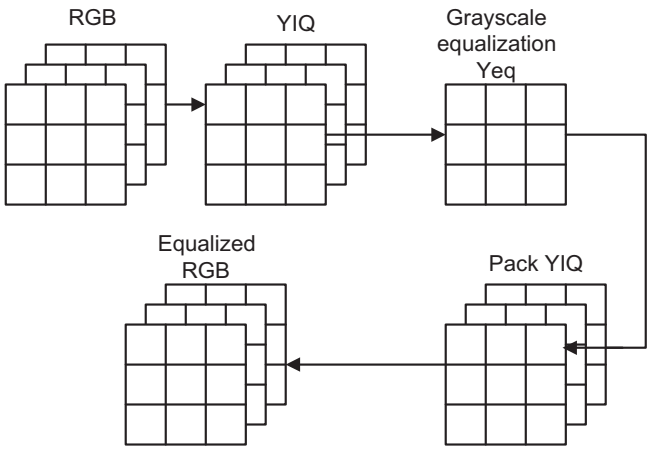


FIGURE 14.15

Color image equalization.

Figure 14.16A shows an original RGB color outdoors scene that is underexposed. Figure 14.16B shows the equalized RGB image using the method of equalizing the Y channel only. We can verify significant improvement with the equalized image showing much detailed information. The color print of the image is included in the color insert.

We can also use the histogram equalization method to equalize each of the R, G , and B channels, or their possible combinations. Figure 14.17 illustrates such a procedure.

Some color effects can be observed. Equalization of the R channel only would make the image look redder since the red pixel values are stretched out to the full range. Similar

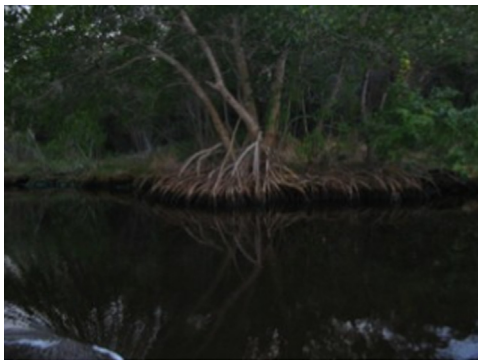


FIGURE 14.16A

Original RGB color image. See color image on book's companion site.

FIGURE 14.16B

Equalized RGB color image. See color image on book's companion site.

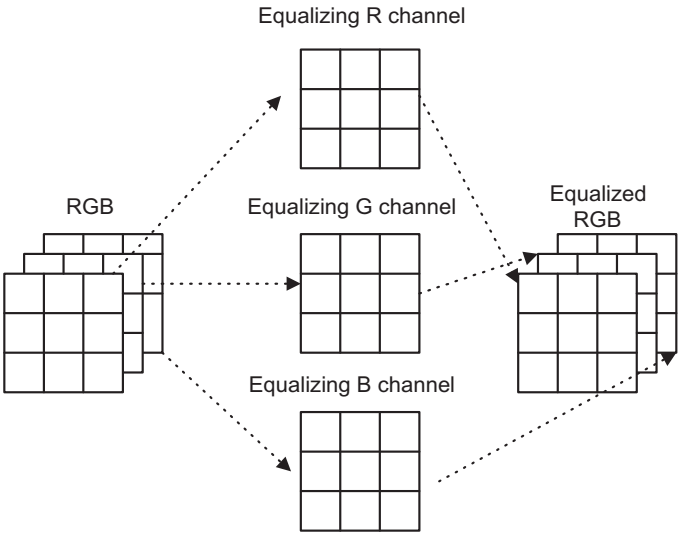


FIGURE 14.17

Equalizing RGB channels.

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 Figure 14.18. The image from equalizing the R, G, and B channels simultaneously is shown in the upper left corner, which offers improved image contrast.

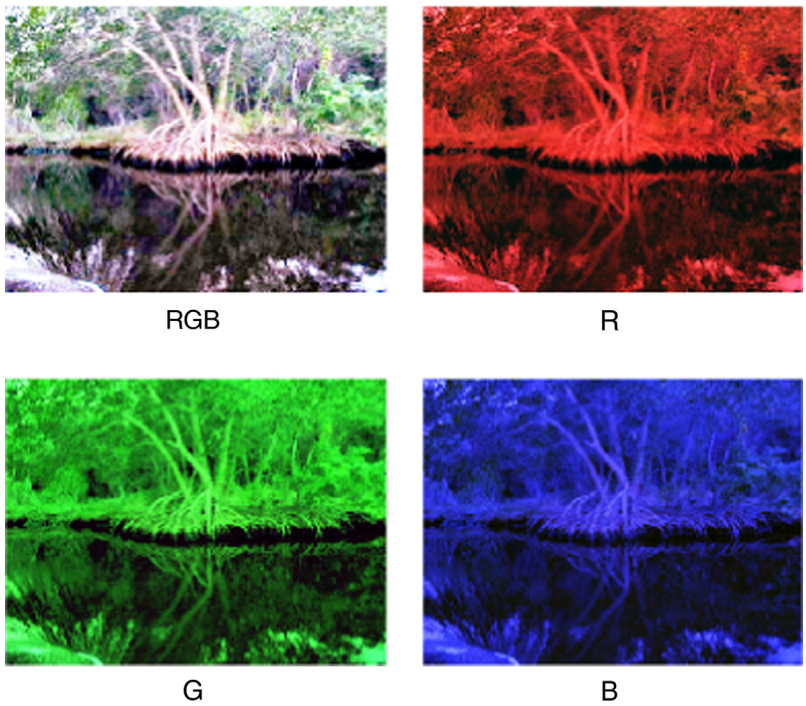


FIGURE 14.18

Equalization effects for RGB channels. See color image on book's companion site.

14.2.3 8-Bit Indexed Color Image Equalization

Equalization of the 8-bit color indexed image is more complicated. This is due to the fact that 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 is shown in Figure 14.19.

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

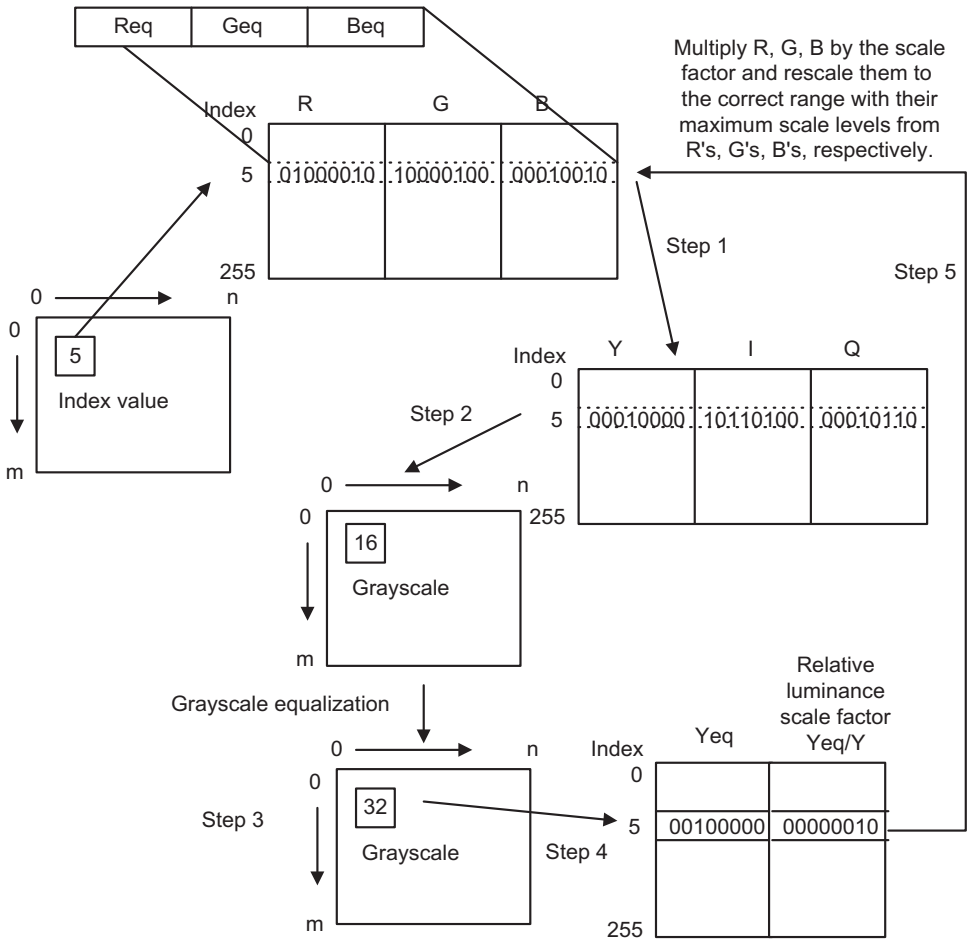


FIGURE 14.19

Equalization of 8-bit indexed color image.

Step 5. Finally, the R, G, B 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 the 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, Figure 14.20 shows the equalized indexed color image. We see that the equalized image displays much more detail. Its color version is reprinted in the color insert.

FIGURE 14.20

Equalized indexed 8-bit color image. See color image on book's companion site.

14.2.4 MATLAB Functions for Equalization

Figure 14.21 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
imhist = histogram display
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 14.1.
Program 14.1. 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
```

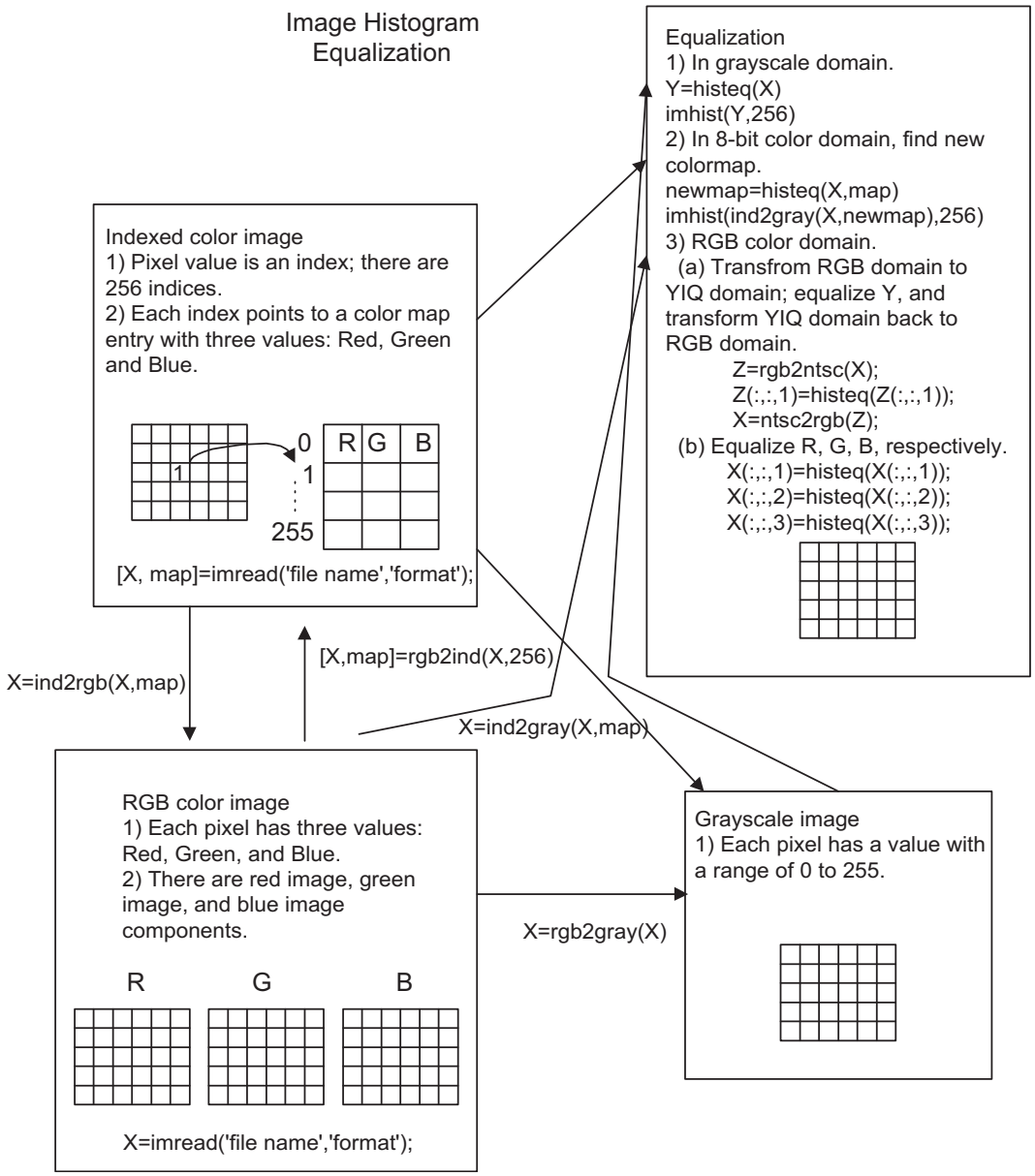


FIGURE 14.21

MATLAB functions for image equalization.

```
Z1(:,:,1)=histeq(Z1(:,:,1));              % 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 indexed 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);
```

14.3 IMAGE LEVEL ADJUSTMENT AND CONTRAST

Image level adjustment can be used to linearly stretch the 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 will study this technique in the following subsections.

14.3.1 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. Figure 14.22 illustrates linear level adjustment.

The linear level adjustment is given by the following formula:

$$p_{adjust}(m,n) = Bottom + \frac{p(m,n) - \mathrm{L}}{\mathrm{H} - \mathrm{L}} \times (\text{Top-Bottom}) \tag{14.6}$$

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

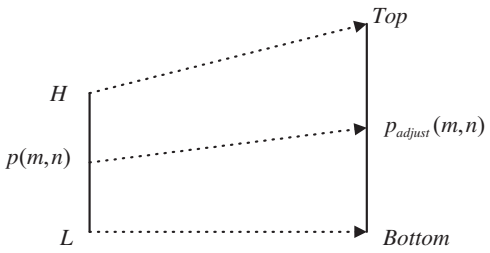


FIGURE 14.22

Linear level adjustment.

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

EXAMPLE 14.6

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

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

a. Perform level adjustment to the 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:

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

Applying Equation (14.6) yields the second column in Table 14.4.

Table 14.4 Image Adjustment Results in Example 14.6.

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

b. For the shift-up operation, it follows that

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

c. For the shift-down operation, we set

$$H = 5,\ L = 3,\ Top = 3,\ Bottom = 0$$

The results for (b) and (c) are listed in the third and fourth column, respectively, of Table 14.4.

According to Table 14.4, we have three images:

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

Next, applying the level adjustment for the neck image of Figure 14.13A, we get the results shown in Figure 14.23: 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.

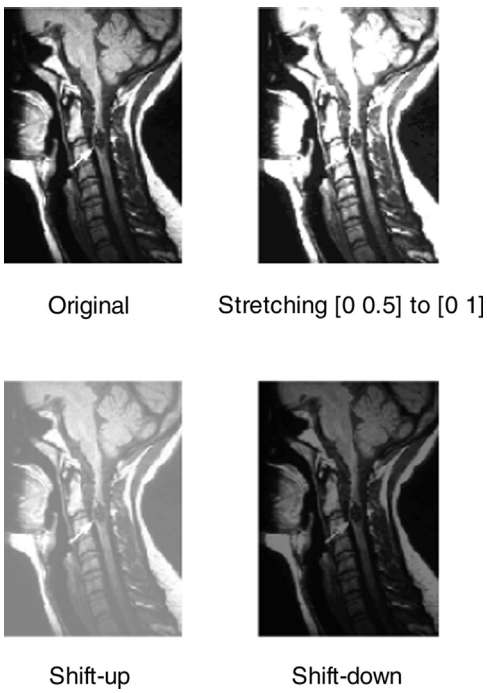


FIGURE 14.23

Image level adjustment.

14.3.2 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 to 255 for display. The following addition of two 8-bit images yields a sum that is out of the 8-bit range:

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

To scale the combined image, modify Equation (14.6) as follows:

$$p_{scaled}(m,n) = \frac{p(m,n) - \text{Minimum}}{\text{Maximum} - \text{Minimum}} \times (\text{Maximum scale level}) \tag{14.7}$$

Note that in the image to be scaled,

Maximum = 325
Minimum = 55
Maximum scale level = 255
After scaling we have

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

14.3.3 MATLAB Functions for Image Level Adjustment

Figure 14.24 lists applications of the MATLAB level adjustment function, which is defined as follows:

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 Section 14.3.1)
0 < gamma < 1 lightens image; gamma > 1 darkens image

14.4 IMAGE FILTERING ENHANCEMENT

As with one-dimensional digital signal processing, we can design a digital image filter such as low-pass, highpass, bandpass, and notch to process the image to obtain the desired effect. In this section, we

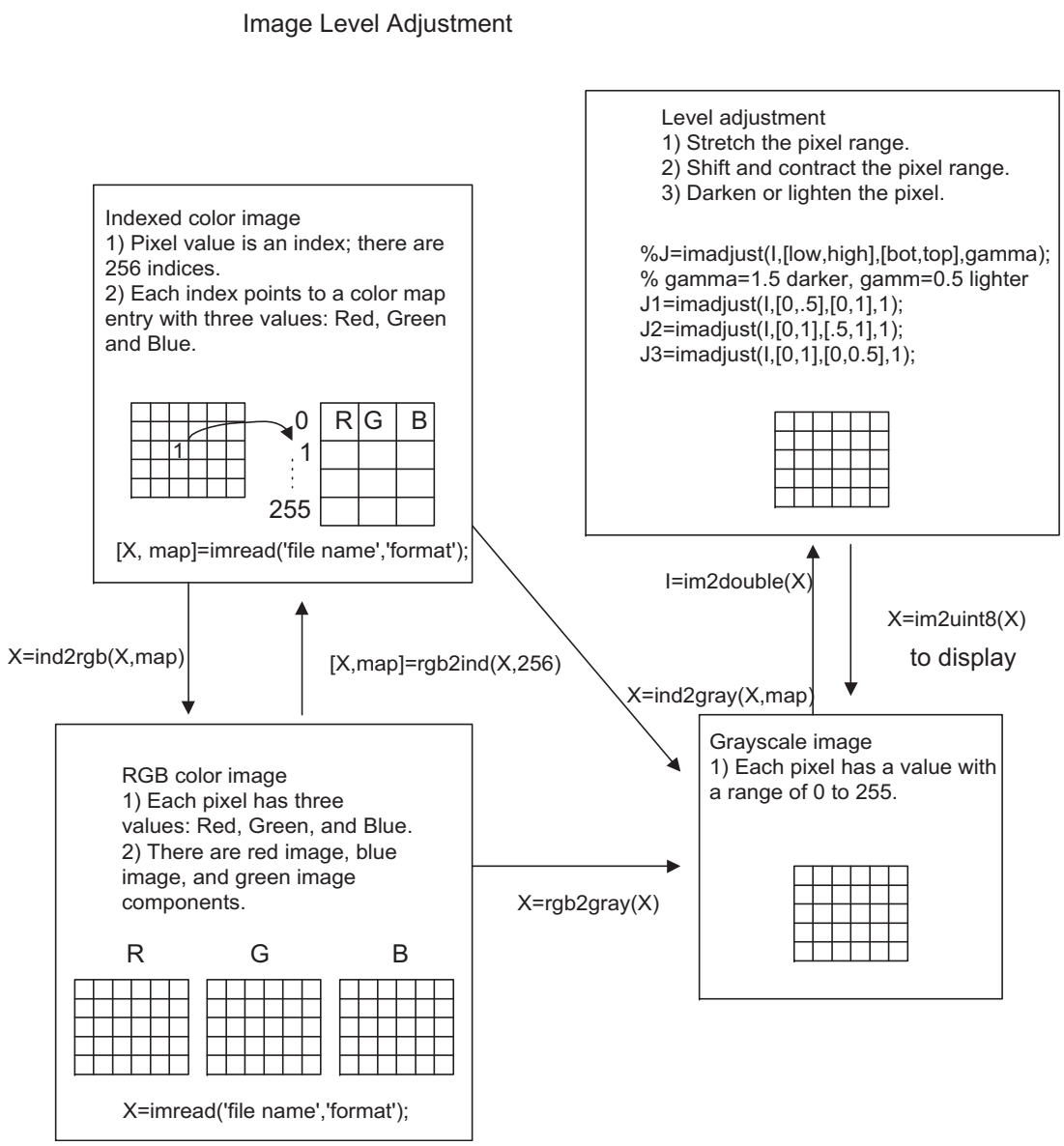


FIGURE 14.24

MATLAB functions for image level adjustment.

discuss the most common ones: lowpass filters to remove noise, median filters to remove impulse noise, and edge detection filters to discover 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).

14.4.1 Lowpass Noise Filtering

One of the simplest lowpass filters 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:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{14.8}$$

4×4 average kernel:

$$\frac{1}{16}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{14.9}$$

8×8 average kernel:

$$\frac{1}{64}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{14.10}$$

Each of the 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 14.7

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

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

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

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

Solution:

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 by 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 the 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.

Figure 14.25 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.

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

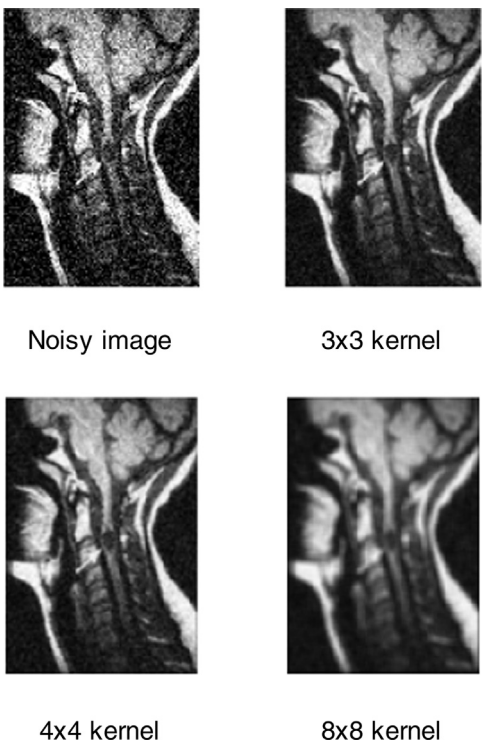


FIGURE 14.25

Noise filtering using the lowpass average kernels.

$$\frac{1}{25}\begin{bmatrix} 0 & 2 & 4 & 2 & 0 \\ 2 & 15 & 27 & 15 & 2 \\ 4 & 27 & 50 & 27 & 4 \\ 2 & 15 & 27 & 15 & 2 \\ 0 & 2 & 4 & 2 & 0 \end{bmatrix} \tag{14.11}$$

In this kernel the center pixel is weighted the most and the weights become lower and lower as we move away from the center. 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. The larger the standard deviation, the flatter the kernel; with a flatter kernel, the blurring effect will be more pronounced.

Figure 14.26A shows the noisy image, while Figure 14.26B shows the enhanced image using a 5×5 Gaussian filter kernel. Clearly, the majority of the noise has been filtered, while the blurring effect is significantly reduced.

14.4.2 Median Filtering

The median filter is one type of nonlinear filter. It is very effective at removing impulse noise, the "pepper and salt" noise, in an 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

FIGURE 14.26A

Noisy image for a human neck.

FIGURE 14.26B

Enhanced image using a Gaussian lowpass filter.

and the column edge. Hence, edge distortion is introduced at image boundary. Let us look at Example 14.8.

EXAMPLE 14.8

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

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

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

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

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 and $3/2 = 1$ row of zeros at the upper and bottom edges:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 255 & 100 & 100 & 0 \\ 0 & 100 & 255 & 100 & 100 & 0 \\ 0 & 255 & 100 & 100 & 0 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

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 thier value as

$$0,\ 0,\ 0,\ 0,\ 0,\ 100,\ 100,\ 255,\ 255$$

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 review the element at location (1,1):

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 255 & 100 & 100 & 0 \\ 0 & 100 & 255 & 100 & 100 & 0 \\ 0 & 255 & 100 & 100 & 0 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

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 the zero padding effect. However, for a large image, the portion of the boundary pixels (outmost image edges) is significant small so that their 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.

The image in Figure 14.27A is corrupted by "pepper and salt" noise. The median filter with a 3×3 kernel is used to filter the impulse noise. The enhanced image shown in Figure 14.27B has a significant quality improvement. 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.

FIGURE 14.27A

Noisy image (corrupted by "pepper and salt" noise).

FIGURE 14.27B

The enhanced image using a 3 × 3 median filter.

14.4.3 Edge Detection

In many applications, such as pattern recognition and fingerprint 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 in the following:

Horizontal Sobel edge detector:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{14.12}$$

The kernel subtracts the first row in the kernel from the third row to detect the horizontal difference. Vertical Sobel edge detector:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{14.13}$$

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

A Laplacian edge detector is devised to tackle both vertical and horizontal edges. It is described in the following:

Laplacian edge detector:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{14.14}$$

EXAMPLE 14.9

Given the following 8-bit grayscale image, use the Sobel horizontal edge detector to detect horizontal edges:

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

Solution:

We pad the image with zeros before processing as follows:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 110 & 110 & 110 & 110 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

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

Figure 14.28 shows the results from edge detection.

Figure 14.29 shows the edge detection for the grayscale image of the cruise ship in Figure 14.3. Sobel edge detection can tackle only the horizontal edge or the vertical edge, as shown in Figure 14.29, where the edges of the image have both horizontal and vertical features. We can simply combine the two horizontal and vertical edge-detected images and then rescale the resultant image in the full range. Figure 14.29(c) 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 edge detection, which is 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 uses the Laplacian derivative operation for 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 the degree of noise filtering before the 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 a standard deviation of $\sigma = 0.8$ is given by

$$\begin{bmatrix} 4 & 13 & 16 & 13 & 4 \\ 13 & 9 & -25 & 9 & 13 \\ 16 & -25 & -124 & -25 & 16 \\ 13 & 9 & -25 & 9 & 13 \\ 4 & 13 & 16 & 13 & 4 \end{bmatrix} \tag{14.15}$$

The processed edge detection using the Laplacian of Gaussian filter in Equation (14.15) is shown in Figure 14.30. 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.

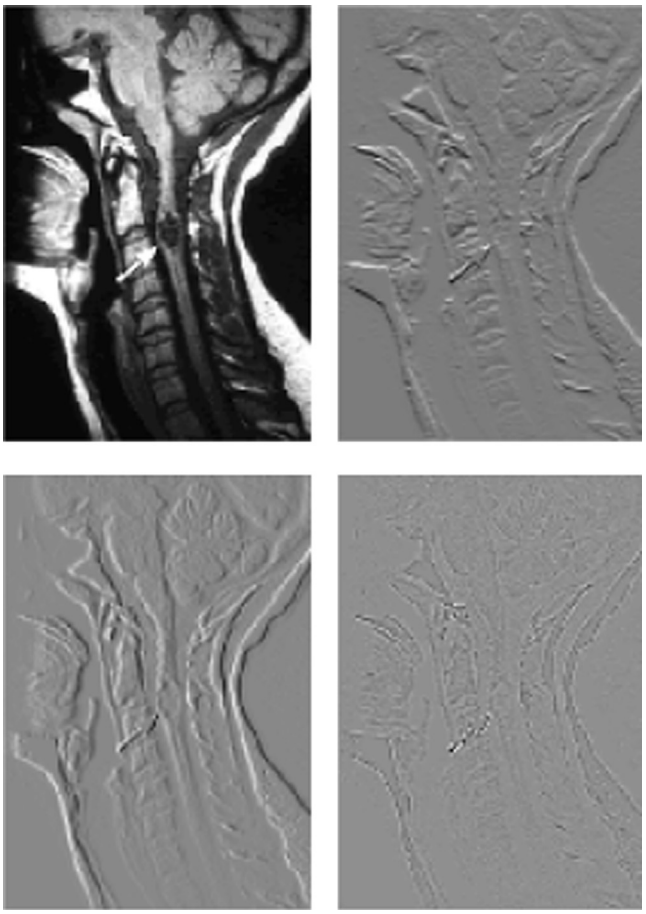


FIGURE 14.28

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.

14.4.4 MATLAB Functions for Image Filtering

MATLAB image filter design and implementation functions are summarized in Figure 14.31. The MATLAB functions are explained in the following:

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)

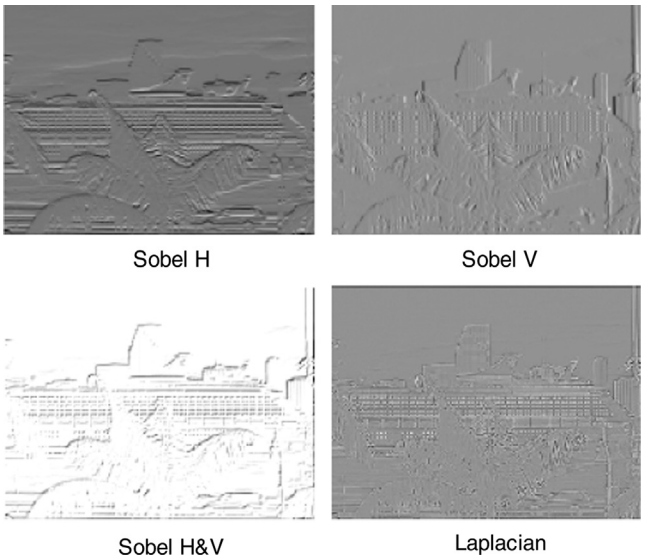


FIGURE 14.29

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

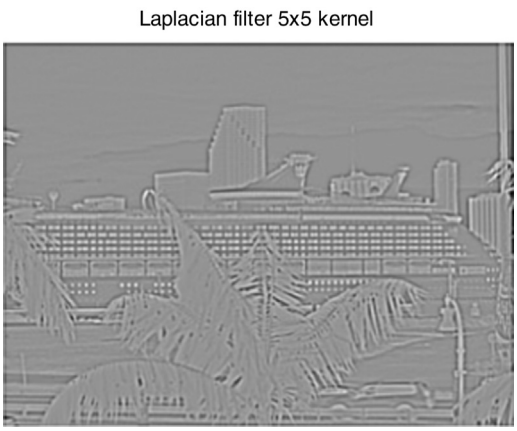


FIGURE 14.30

Image edge detection using a Laplacian of Gaussian filter.

Y = filter2([convolution kernel], X) = two-dimensional filter using the convolution kernel
Y = medfilt2(X, [row size, column size]) = two-dimensional median filter

Program 14.2 lists the sample MATLAB codes for filtering applications. Figure 14.31 outlines the applications of the MATLAB functions.

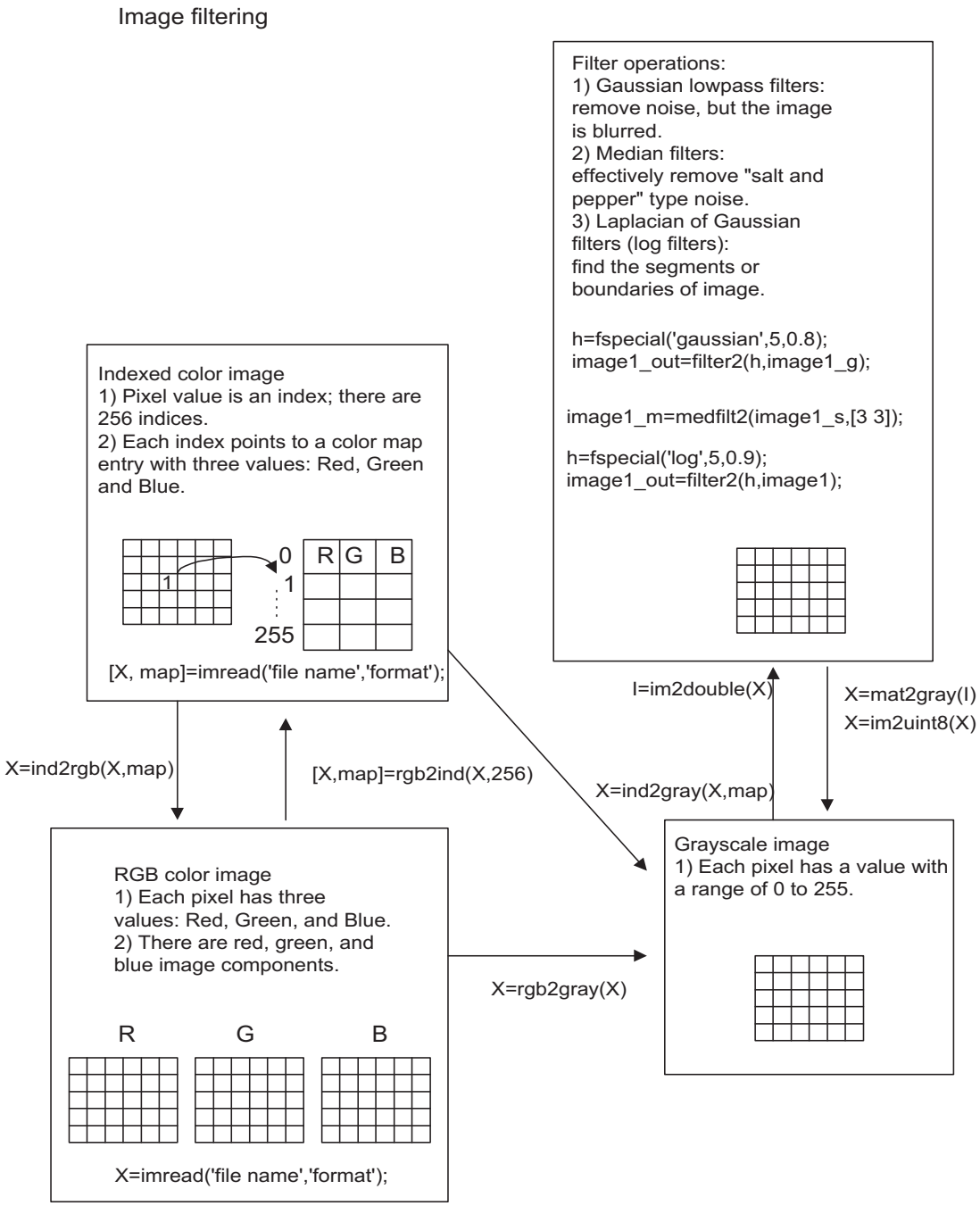


FIGURE 14.31

MATLAB functions for filter design and implementation.

Program 14.2. 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 = 5×5
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 Gaussian filter
% of size 5×5 with a main lobe width of 0.8.
image1_out=filter2(h,image1_g);       % Perform filtering
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
kernel
image1_out=filter2(h,I);              % 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');
```

14.5 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 Figure 14.32.

As a simple choice, we choose three sine functions for RGB transformations, as shown in Figure 14.33A. The phase and period of one sine function can be easily changed so that the

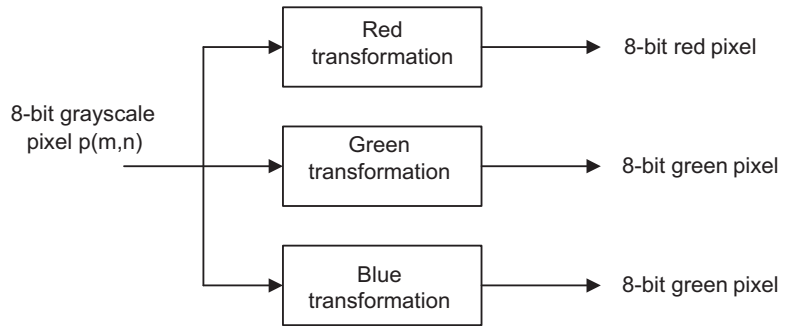


FIGURE 14.32

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

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 such that their color component values are 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) can be transformed to the red, green, or blue with a favorable choice.

EXAMPLE 14.10

In the grayscale image in Figure 14.13A, 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. Make the background to as close to blue as possible, and make the area pointed to by the arrow as close to red as possible.

Solution:

The transformation functions are chosen as shown in Figure 14.33A, 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. Figure 14.33B shows the processed pseudo-color image; it is included in the color insert.

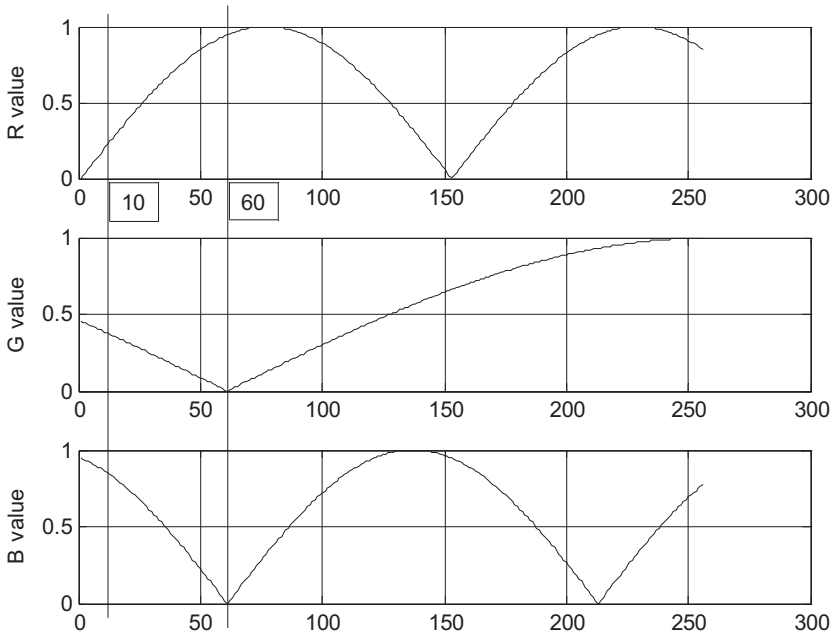


FIGURE 14.33A

Three sine functions for grayscale transformation.

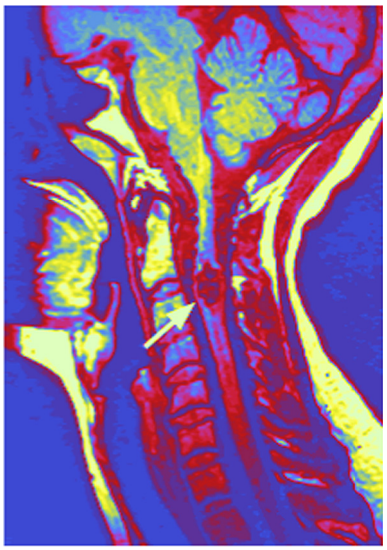


FIGURE 14.33B

The pseudo-color image. See color image on book's companion site.

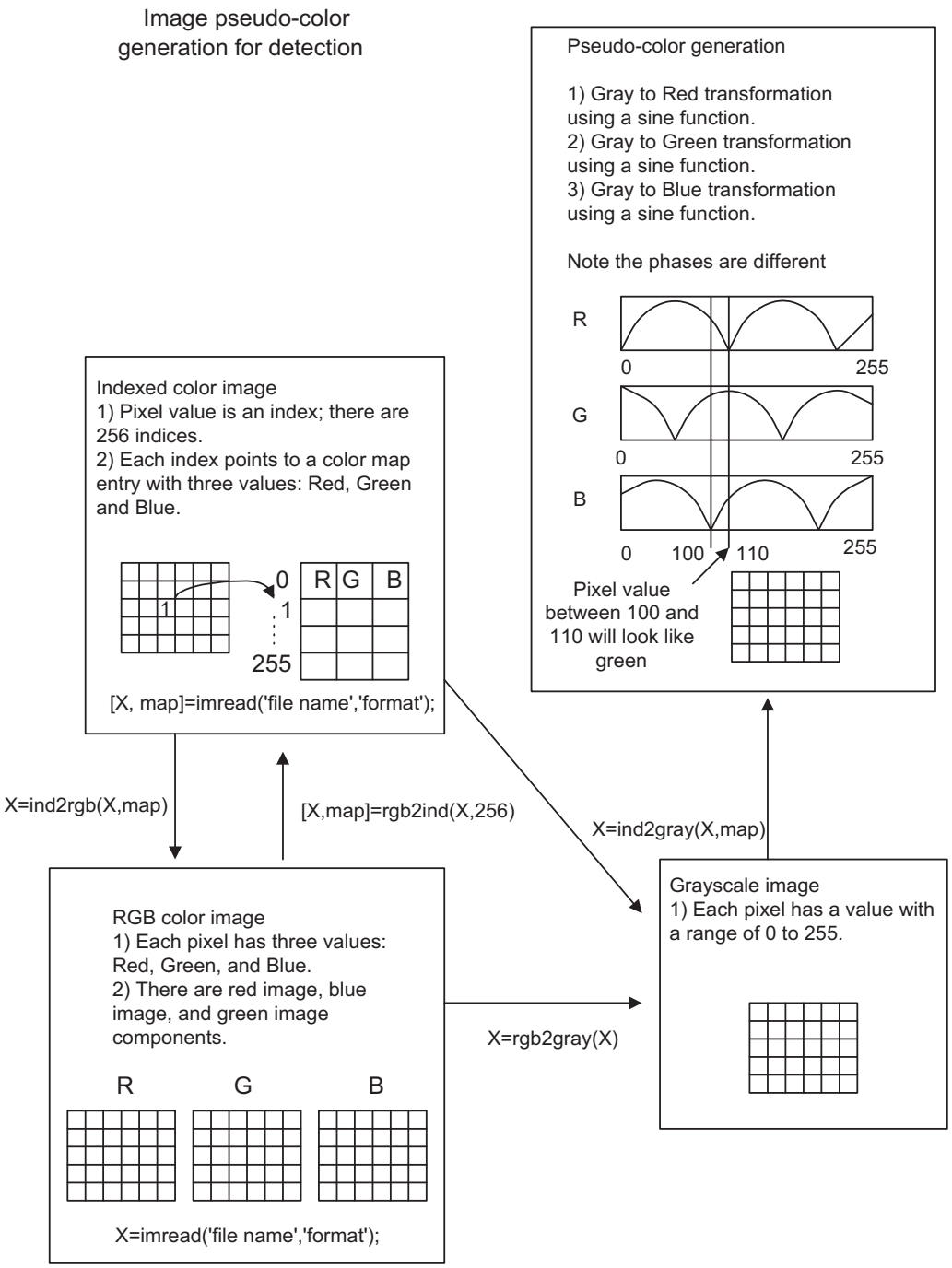


FIGURE 14.34

Illustrative procedure for pseudo-color generation.

Program 14.3 lists the sample MATLAB codes for pseudo-color generation for a grayscale image.
Program 14.3. Program examples for pseudo-color generation.

```
close all; clear all;clc
disp('Convert the grayscale image to the pseudo-color image');
[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
```

14.6 IMAGE SPECTRA

In one-dimensional 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 a one-dimensional (1D) DFT. The spectrum including the magnitude and phase is also in two dimensions. The equations of the 2D-DFT are given by

$$X(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} p(m,n) W_M^{um} W_N^{vn} \tag{14.16}$$

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 the 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

$$A(u,v) = \frac{1}{(N \times M)} |X(u,v)| \tag{14.17}$$

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

EXAMPLE 14.11

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 Equation (14.16) leads to

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

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 the following DFT coefficients:

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

Using Equation (14.17), 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 14.12

Given the 200×200 grayscale image shown in Figure 14.35A with a white rectangle (11×3 pixels) at its center and a black background, we can compute its magnitude spectrum (which ranges from 0 to 255). We can display the spectrum in terms of the grayscale. Figure 14.35B 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. In the spectrum image, the upper left corner area in the left upper quarter is white and hence has a highest scale value. Therefore, the image signal has low-frequency

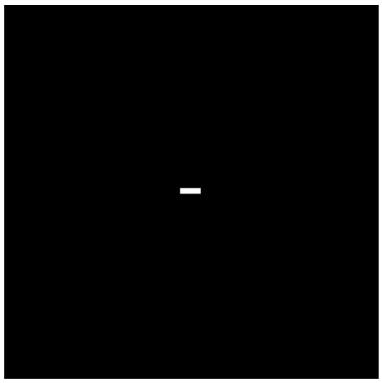


FIGURE 14.35A

A square image.

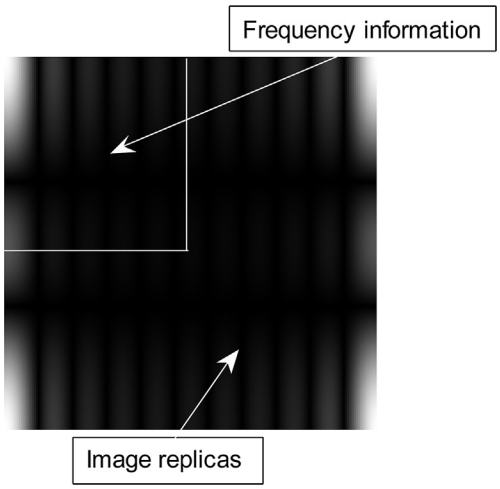


FIGURE 14.35B

Magnitude spectrum for the square image.

dominant components. The spectrum exhibits horizontal and vertical null lines (dark lines). The first vertical null line location can be estimated as 200/11=18 pixels from the left side, while the first horizontal null line happens at 200/3 = 67 pixels from the top. Next, let us apply the 2D spectrum to understand image filtering effects in image enhancement.

EXAMPLE 14.13

Figure 14.36(a) 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

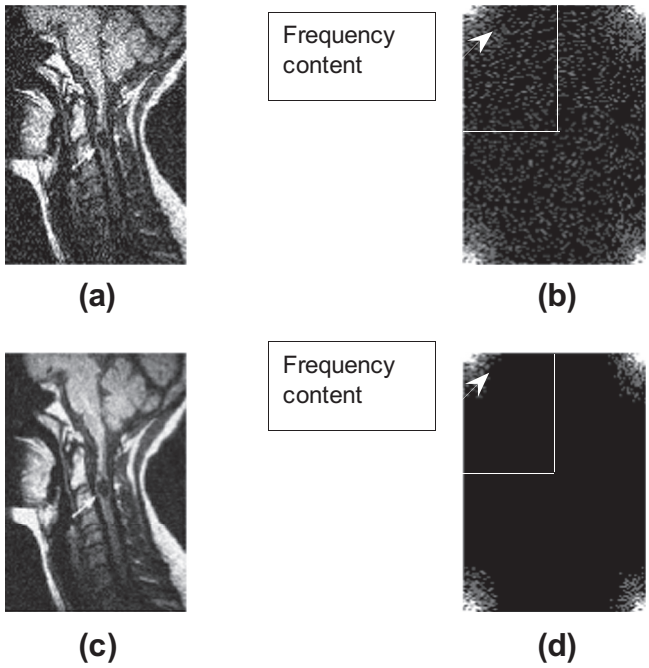


FIGURE 14.36

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.

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 Figure 14.36(b), where we can see that noise occupies the entirety of the image.

To enhance the image, we apply a Gaussian lowpass filter. The enhanced image is shown in Figure 14.36(c), in which the enhancement is easily observed. Figure 14.36(d) displays the spectra for the enhanced image with the same scaling process described just above. As we can see, the noise is significantly reduced compared with Figure 14.36(b).

14.7 IMAGE COMPRESSION BY DISCRETE COSINE TRANSFORM

Image compression is a must 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 lossy compression using the 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 a compression ratio of 20:1 without noticeable quality degradation. JPEG standard image compression is used everyday in real life.

The principle of the DCT is to transform the original image pixels to an identical number of DCT coefficients, where the DCT coefficients have a 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 (with 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 that have small dynamic ranges (most of them do not affect the visual quality of the picture).

14.7.1 Two-Dimensional Discrete Cosine Transform

Image compression uses 2D-DCT, whose transform pairs are defined as follows:

Forward DCT:

$$F(u,v) = \frac{2C(u)C(v)}{\sqrt{MN}} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} p(i,j) \cos\left(\frac{(2i+1)u\pi}{2M}\right) \cos\left(\frac{(2j+1)v\pi}{2N}\right) \tag{14.18}$$

Inverse DCT:

$$p(i,j) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \frac{2C(u)C(v)}{\sqrt{MN}} F(u,v) \cos\left(\frac{(2i+1)u\pi}{2M}\right) \cos\left(\frac{(2j+1)v\pi}{2N}\right) \tag{14.19}$$

where

$$C(m) = \begin{cases} \frac{\sqrt{2}}{2} & \text{if } m = 0 \\ 1 & \text{otherwise} \end{cases} \tag{14.20}$$

$p(i,j)$ = 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 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

$$F(u,v) = \frac{C(u)C(v)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} p(i,j) \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right) \tag{14.21}$$

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

$$p(i,j) = \sum_{u=0}^{7} \sum_{v=0}^{7} \frac{C(u)C(v)}{4} F(u,v) \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right) \tag{14.22}$$

To become familiar with the 2D-DCT formulas, we study Example 14.14.

EXAMPLE 14.14

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 Equation (14.18) yields

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

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

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

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

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

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()** verifies the DCT coefficients as follows:

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

EXAMPLE 14.15

Given the following DCT coefficients from a 2× 2 image, determine the pixel $p(0,0)$:

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

Solution:
Applying Equation (14.19) of the inverse 2D-DCT with $N = M = 2$, $i = 0$, and $j = 0$, it follows that

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

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

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

14.7.2 Two-Dimensional JPEG Grayscale Image Compression Example

To understand JPEG image compression, we examine an 8 × 8 grayscale subblock. Table 14.5 shows a subblock of the grayscale image in Figure 14.37 that is to be compressed. Applying 2D-DCT leads to Table 14.6.

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 14.7.

Table 14.5 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

FIGURE 14.37

Original image.

After normalization, as shown in Table 14.8, 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 14.8, most nonzero coefficients reside in 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 14.9.

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

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

where "EOB" = 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 11, as well as by run-length coding of AC values and Huffman coding, which both belong to lossless compression techniques. We will pursue this in the next section.

Table 14.6 DCT Coefficients for the Subblock Image in Table 14.5

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

Table 14.7 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 14.8 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

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 14.10.

For comparison, the errors between the recovered image and original image are calculated and listed in Table 14.11.

The original and compressed images are displayed in Figures 14.37 and 14.38, respectively. We do not see any noticeable difference between these two grayscale images.

Table 14.9 DCT Coefficient Scan Order

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

Table 14.10 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 14.11 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

FIGURE 14.38

JPEG compressed image.

14.7.3 JPEG Color Image Compression

This section is devoted to reviewing 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 Figure 14.39.

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.

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, that is, divided by the corresponding value given in the quantization matrix. In this way, a smaller

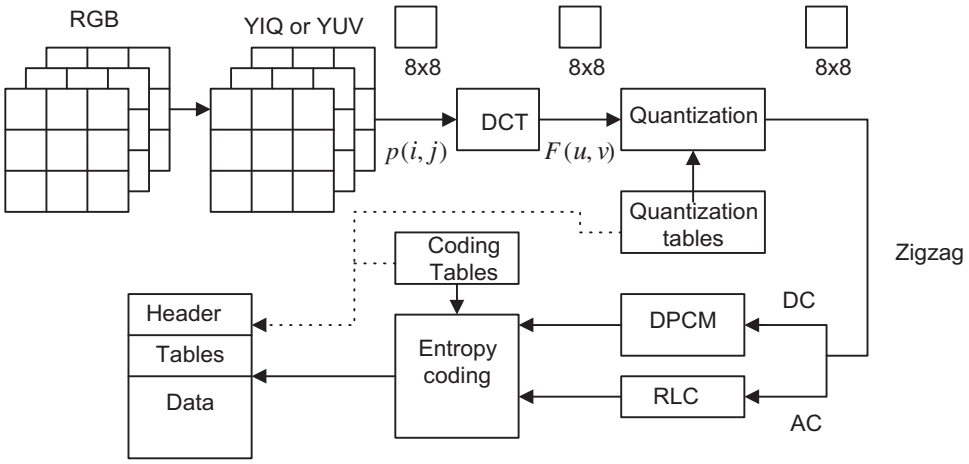


FIGURE 14.39

Block diagram for JPEG encoder.

Table 14.12 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 14.13 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

number of bits can be used for encoding the DCT coefficients. There are two different quantization tables, one for luminance (which is the same as the one in the last section and listed here again for comparison) and the other for chrominance (Tables 14.12 and 14.13).

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

Differential Pulse Code Modulation 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, the 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 of

$$200,\ -10,\ -10,\ -20,\ 10$$

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 of zeros to skip and
- the 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 towards 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 14.14, 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 (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.

Table 14.14 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

FIGURE 14.40

JPEG compressed color image. See color image on book's companion site.

Symbol 1: (run length, size)
Symbol 2: (amplitude)

The 4-bit run length can only tackle 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 14.14, 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:

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

We can produce a bit stream for AC coefficients:

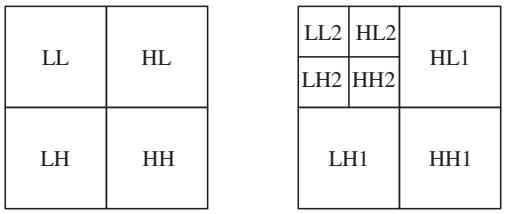
$$(0000, 0001, 0), (0001, 0001, 0), (0010, 0001, 0),$$

$$(0000, 0010, 11), (0011, 0010, 10), (0000, 0000)$$

There are 55 bits in total. Figure 14.40 shows a JPEG compressed color image (included in the color insert). The decompressed image is indistinguishable from the original image after comparison.

14.7.4 Image Compression Using Wavelet Transform Coding

We can extend the one-dimensional discrete wavelet transform (DWT) discussed in Chapter 13 to the two-dimensional DWT. The procedure is described as follows. Given an N $\times$ N image, the 1D-DWT using level 1 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 LL, HL, LH, and HH as shown in Figure 14.41(a). The same procedure repeats for the LL band only and results in the second-level subbands: LL2, HL2, LH2, and HH2



(a) Level-one transformation **(b)** Level-two transformation

FIGURE 14.41

The two-dimensional DWT for level 1 and level 2.

(Figure 14.41(b)). The process proceeds to higher levels as desired. With the obtained wavelet transform, we can quantize coefficients to achieve the compression requirement. For example, for the second-level coefficients, we can omit HL1, LH1, HH1 to simply achieve a 4:1 compression ratio. Decompression reverses the process, that is, we inversely transform columns and then rows of the wavelet coefficients. We can apply the IDWT to the recovered LL band with the column and row inverse transform processes, and continue until the inverse transform at level 1 is completed. Let us look at an illustrative example.

EXAMPLE 14.16

Consider the following 4×4 image:

114	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. The result for the first row is displayed below:

```
>> dwt([1 1]/sqrt(2), [114   135   122   109],1)'   % Row vector coefficients
ans =176.0696   163.3417   −14.8492   9.1924
```

The completed row transform is listed below:

176.0696	163.3417	−14.8492 9	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 presented:
>> dwt([1 1]/sqrt(2),[176.0696 154.1493 178.8980 171.8269],1)'
ans = 233.5000 248.0000 15.5000 5.0000

Applying the transform to each column completes the level-1 transformation:

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

Now, we perform the level-2 transformation. Applying the transform to the first row for the 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

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

Similarly, the first-column MATLAB result is listed below:

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

Finally, we achieve the completed level-2 DWT as

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

b. Recovering the 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

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

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

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

The recovered LL1 band is shown in the following:

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

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

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

176.0696	163.3417	−14.8492	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

Finally, we perform the inverse of the row transform at level 1. The first row result in MATLAB is listed below:

```
>> idwt([1 1]/sqrt(2),[ 176.0696 163.3417 −14.8492 9.1924],1)'
ans = 114.0000 135.0000 122.0000 109.0000
```

The final inverse DWT is obtained as

114.0000	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

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

Figure 14.42 shows 8-bit grayscale image compression by applying the one-level wavelet transform, in which a 16-tap Daubechies wavelet is used. The wavelet coefficients (each is coded using 8 bits) are shown in Figure 14.42(a). By discarding the HL, LH, and HH band coefficients, we can achieve 4:1 compression. The decoded image is displayed in Figure 14.42(c). The MATLAB program is listed in Program 14.4.

Figure 14.43 illustrates two-level wavelet transform and compression results. By discarding the HL2, LH2, HH2, HL1, LH1, and HH1 subbands, we achieve 16:1 compression. However, as shown in

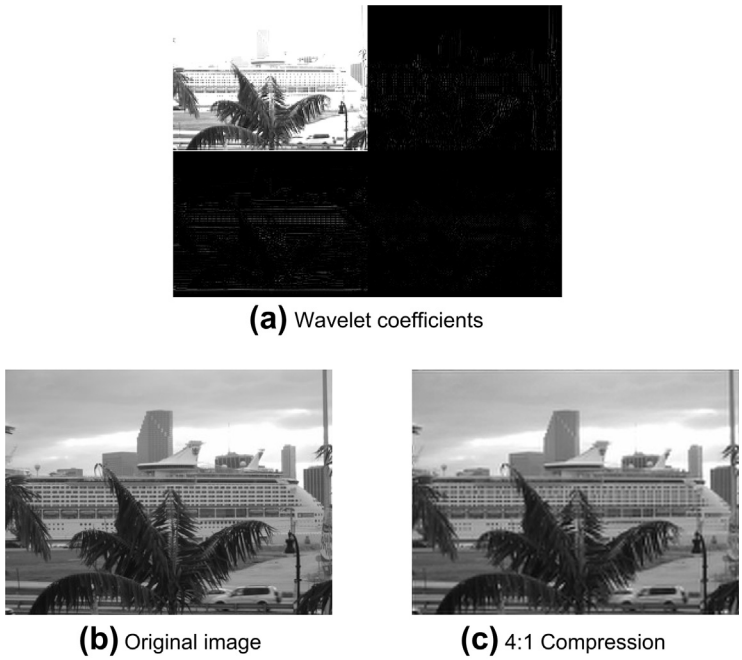


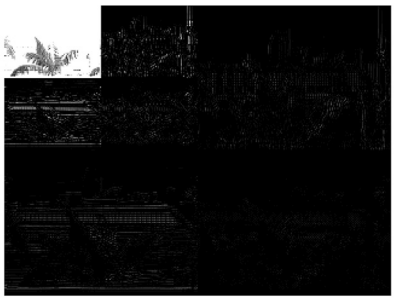
FIGURE 14.42

(a) Wavelet coefficients; (b) original image; (c) 4:1 compression.

Figure 14.43(c), we can observe a noticeable degradation of image quality. Since the high-frequency details are discarded, the compressed image shows a significant smoothing effect. In addition, there are many advanced methods to quantize and compress the wavelet coefficients. Of these compression techniques, the embedded zerotree wavelet (EZW) method is the most efficient one; it can be found in Li and Drew (2004).

Program 14.4. 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);
```



(a) Wavelet coefficients

(b) Original image

(c) 16:1 Compression

FIGURE 14.43

(a) Wavelet coefficients; (b) original image; (c) 16:1 compression.

```
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 bits
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 14.5. 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); figure (3),imshow(Y11);xlabel('16:1 compression');
```

14.8 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:

$$\text{Mixed image} = (1-\alpha) \times \text{image}_1 + \alpha \times \text{image}_2 \tag{14.23}$$

where $\alpha =$ 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 Figure 14.44A consisting of six frames is generated using $\alpha = 0$, $\alpha = 0.2$, $\alpha = 0.4$, $\alpha = 0.6$, $\alpha = 0.8$, and $\alpha = 1.0$, respectively, for two images. The equations for generating these frames are listed as follows:

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

The sequence begins with the Grand Canyon image and fades in with the cruise ship image. At frame 4, 60% of the cruise ship is faded in, and the image begins to be discernible as such. The sequence ends

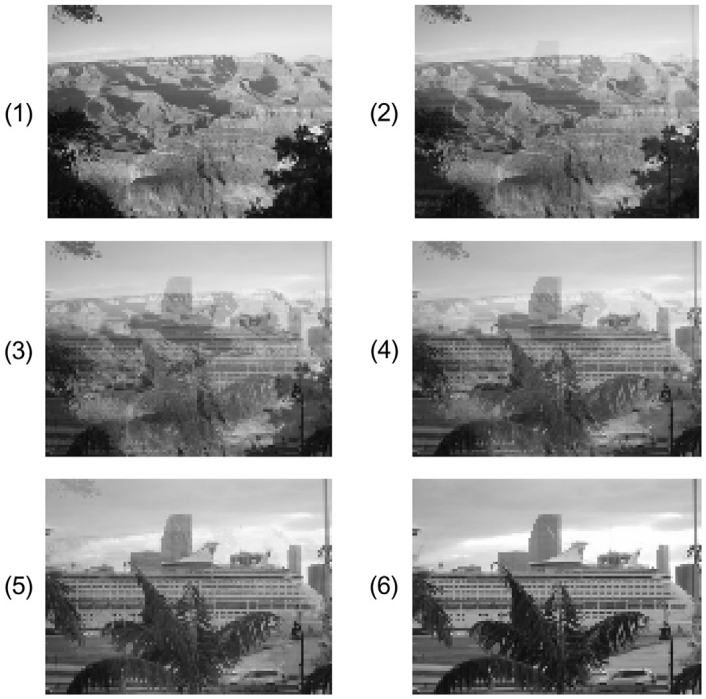


FIGURE 14.44A

Grayscale video sequence.

with the cruise ship in 100% fade-in. Figure 14.44A displays the generated grayscale sequence. Figure 14.44B shows the RGB color video sequence (also given in the color insert).

14.9 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 red, green, and blue 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 intensity (luminance) and two-color (chrominance) components that modulate the carrier wave. This signal is used in 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.

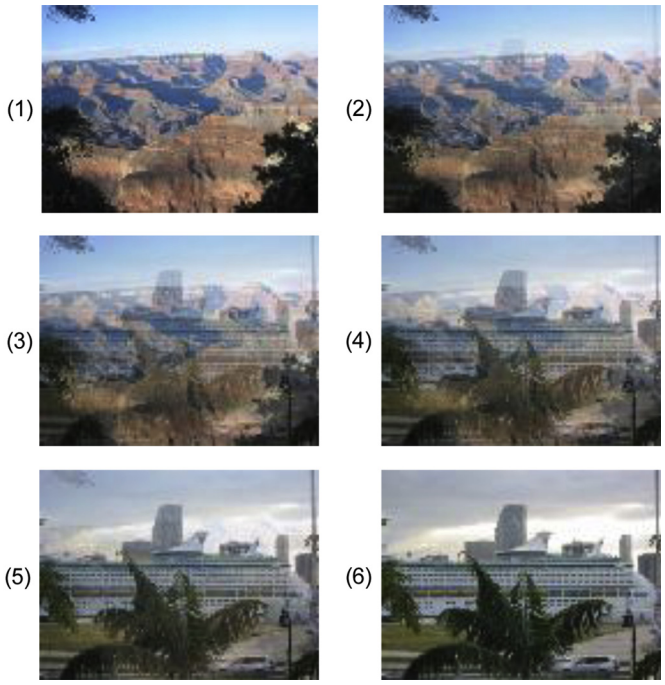


FIGURE 14.44B

The RGB color video sequence.

14.9.1 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 frames 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 Figure 14.45, 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 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 Figure 14.46. In the United States, negative modulation is used, meaning that the bright and dark intensities are inverted before modulation. In the negative modulated video signal, less amplitude comes from a brighter scene while more amplitude comes from a darker one. Since most pictures contain more white than black levels, the video signal level is reduced. Hence, with negative modulation, possible power efficiency can be achieved for transmission. The reverse process will apply for display at the receiver.

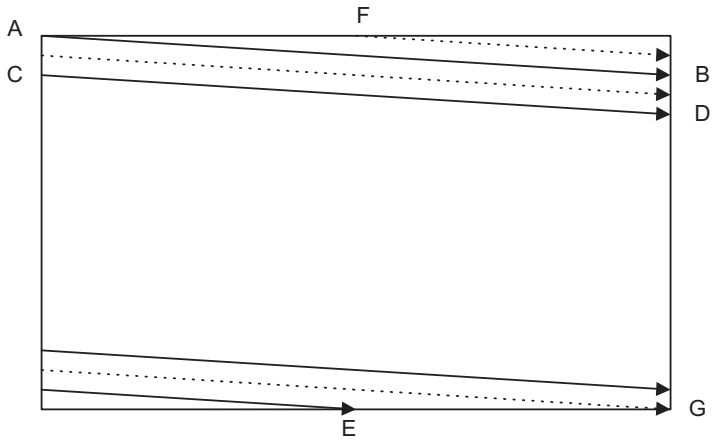


FIGURE 14.45

Interlaced raster scanning.

The horizontal synchronizing pulse controls the timing for the horizontal retrace. The blanking levels are used for synchronizing as well. The "back porch" (Figure 14.46) of the blanking also contains the color subcarrier burst for color demodulation.

The demodulated electrical signal can be seen in Figure 14.47, where a typical electronic signal for one scan line is depicted. The white intensity has a peak value of 0.714 volt, and the black has a voltage level of 0.055 volt, which is close to zero. The blank corresponds to zero voltage, and the synchronizing pulse is at the level of -0.286 volt. Synchronizing takes 10.9 microseconds, while video

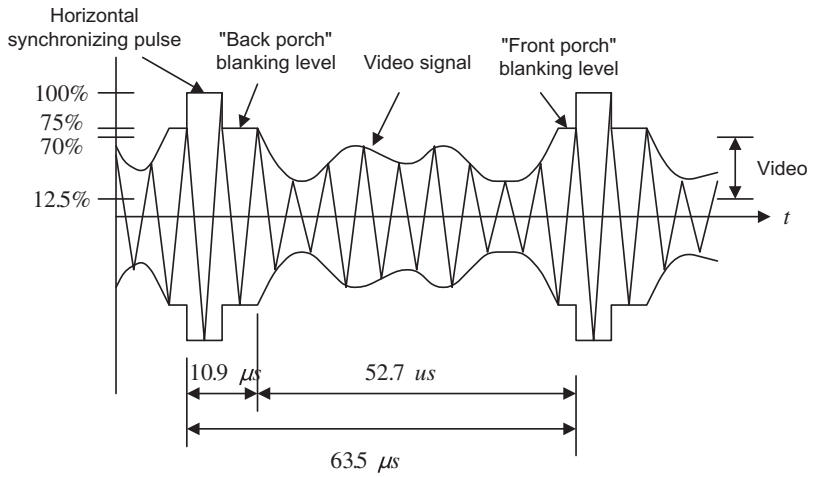


FIGURE 14.46

Video-modulated waveform.

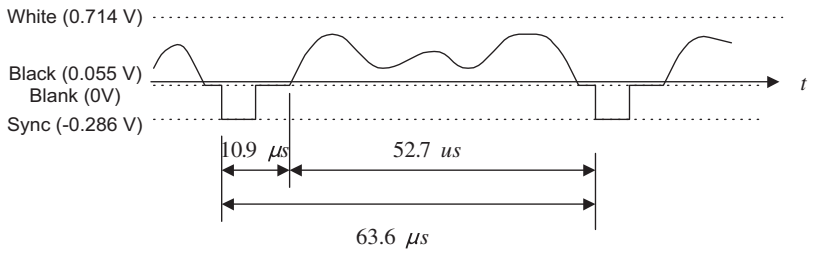


FIGURE 14.47

The demodulated signal level for one NTSC scan line.

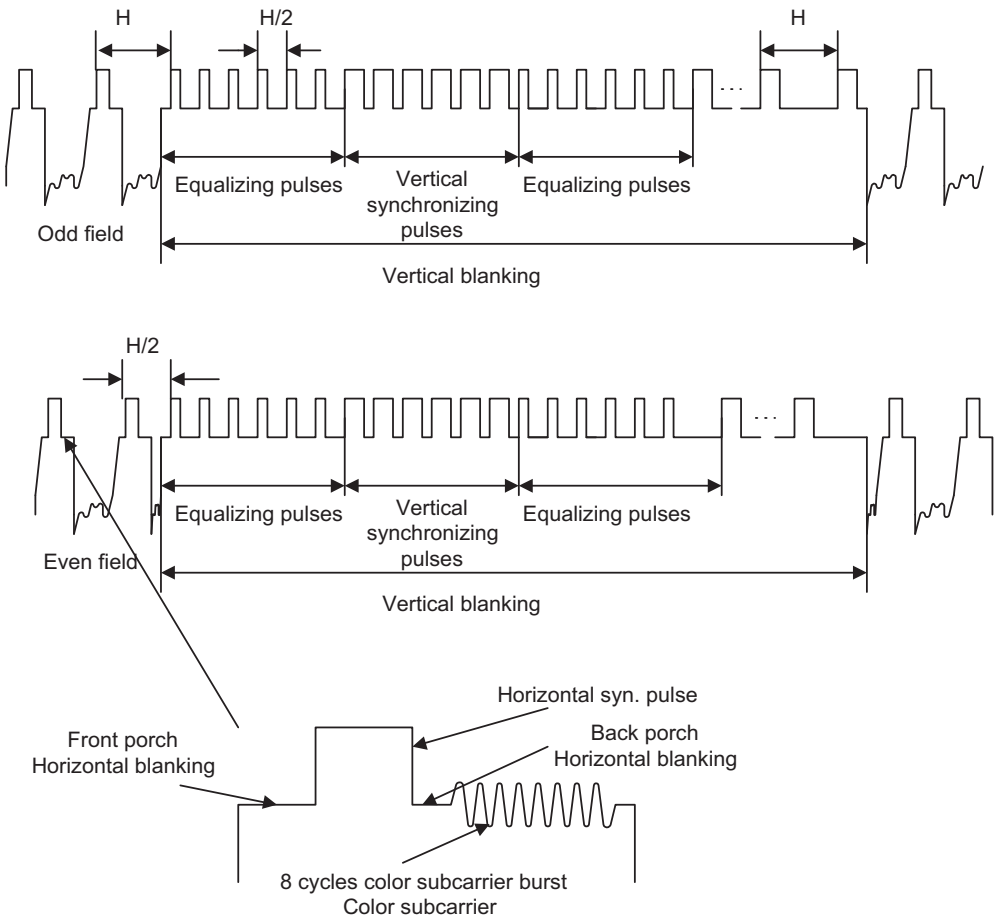


FIGURE 14.48

Vertical synchronization for each field and the color subcarrier burst.

occupies 52.7 microseconds, and one entire scan line occupies 63.6 microseconds. Hence, the line scan rate can be determined as 15.75 kHz.

Figure 14.48 describes vertical synchronization. A pulse train is generated at the end of each field. The pulse train contains 6 equalizing pulses, 6 vertical synchronizing pulses, and another 6 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 a time interval that is equivalent to 9 lines. This leaves lines 10 to 20 for other uses.

A color subcarrier resides on the back porch, as shown in Figure 14.48. 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. The active video lines per frame are 485. The layout of the video data, retrace, and sync data is shown in Figure 14.49.

Blanking regions can be used for V-chip information, stereo audio channel data, and subtitles in various languages. The active line is then 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, while Super VHS uses 400–425 samples per line.

Figure 14.50 shows the NTSC video signal spectra. The NTSC standard assigns a bandwidth of 4.2 MHz for luminance Y, 1.5 MHz for I, and 0.5 MHz for Q, due to the human perception of color

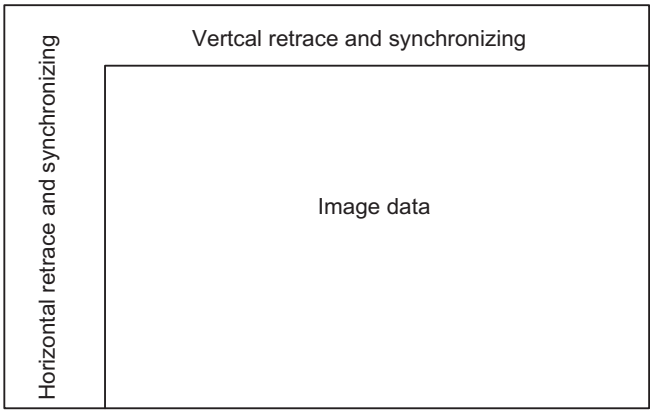


FIGURE 14.49

Video data, retrace, and sync layout.

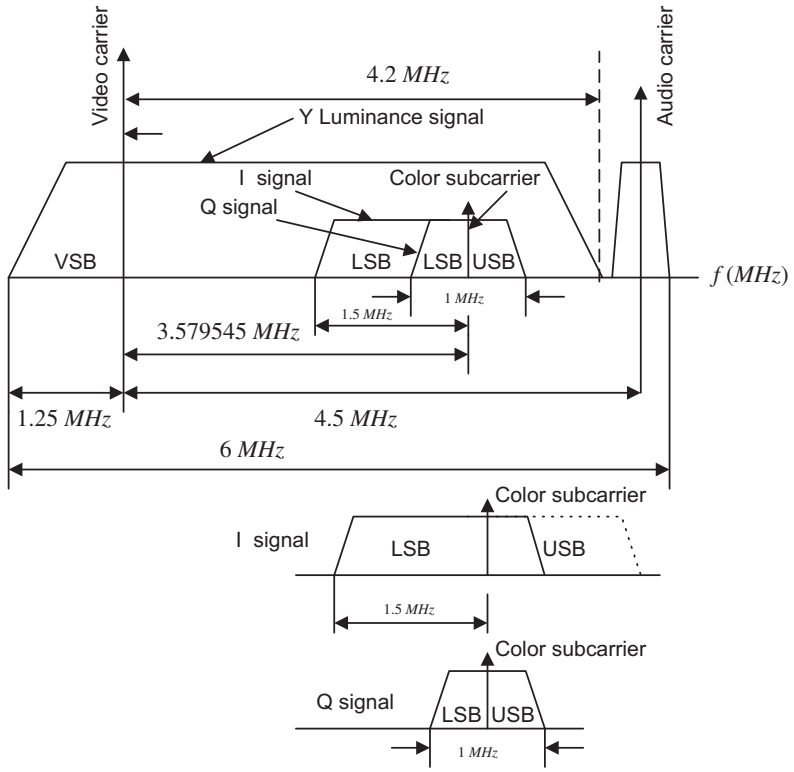


FIGURE 14.50

NTSC Y, I, and Q spectra.

information. There is a wider bandwidth for I because the human eye has higher resolution to the I color component than the Q color component.

As shown in Figure 14.50, *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, 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 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

$$C = I\cos(2\pi f_{sc}t) + Q\sin(2\pi f_{sc}t) \tag{14.24}$$

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

$$Composite = Y + C = Y + I\cos(2\pi f_{sc}t) + Q\sin(2\pi f_{sc}t) \tag{14.25}$$

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. Comb filters may be employed to cancel interferences between the modulated luminance signal and the chroma signal (Li and Drew, 2004). Then we perform demodulation for I and Q as follows:

$$\begin{aligned} C \times 2\cos(2\pi f_{sc}t) &= I2\cos^2(2\pi f_{sc}t) + Q \times 2\sin(2\pi f_{sc}t)\cos(2\pi f_{sc}t) \\ &= I + I \times \cos(2 \times 2\pi f_{sc}t) + Q\sin(2 \times 2\pi f_{sc}t) \end{aligned} \tag{14.26}$$

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. 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, in consecutive lines, the signal and its sign-reversed counterpart are averaged to cancel out phase errors that could be displayed as 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, quadrature multiplexing and the possible cross-coupling of color signals can be avoided by halving the color resolution in the vertical dimension. Table 14.15 includes a summary of analog broadband TV systems.

Table 14.15 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 and Drew, 2004.

14.9.2 Digital Video

Digital video has become dominant over the long-standing analog method in 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). One of the most important standards is CCIR-601, which became ITU-R-601, an international standard for professional video applications.

In CCIR-601, chroma subsampling is carried for digital video. Each pixel is in the 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 Figure 14.51.

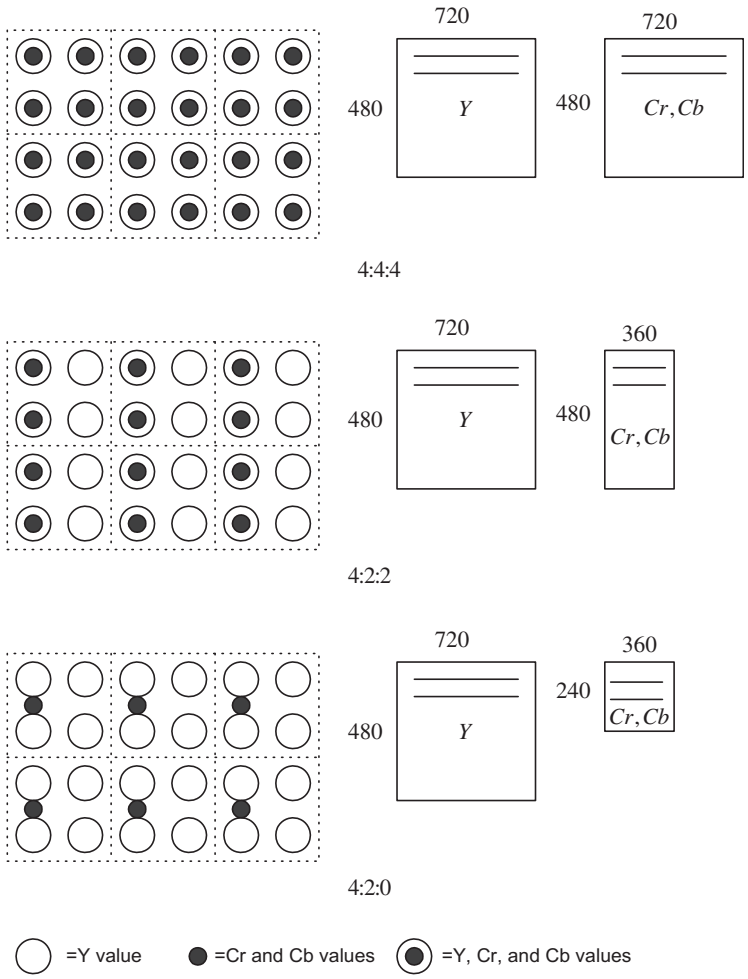


FIGURE 14.51

Chroma subsampling.

Table 14.16 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	4:2:2	4:2:2	4:2:0	4:2:0
Aspect ratio	4:3	4:3	4:3	4:3
Fields/sec	60	50	30	30
Interlaced	Yes	Yes	No	No

CCR, comparison category rating; CIF, common intermediate format; QCIF, quarter-CIF.

Source: Li and Drew, 2004.

In each frame in a 4:4:4 video format, 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 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 Figure 14.51, 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 values for both the 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 Figure 14.51. Digital video specifications are given in Table 14.16.

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

Table 14.17 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.

Table 14.17 High-Definition TV (HDTV) Formats

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

Source: Li and Drew, 2004.

14.10 MOTION ESTIMATION IN VIDEO

In this section, we study motion estimation since this technique is widely used in 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 conditions. 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 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 Figure 14.52.

As shown in Figure 14.52, 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 finding the best match can be chosen using the mean absolute difference (MAD) between the reference frame and the target frame:

$$MAD(i,j) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}|T(m+k,n+l) - R(m+k+i,\, n+l+j)| \tag{14.27}$$

$$u = i, v = j \, for \, MAD(i,j) = \text{minimum, and} - p \le i, j \le p \tag{14.28}$$

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 sequential search to understand the basic idea.

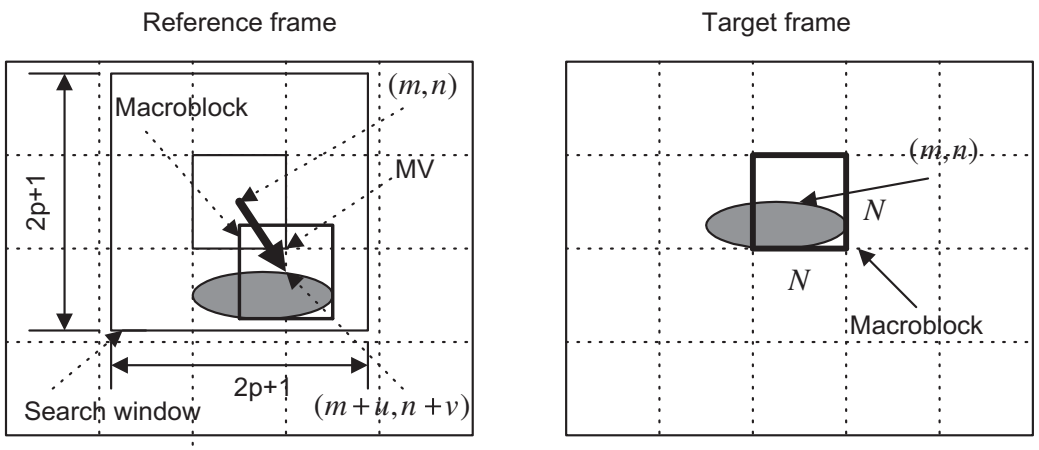


FIGURE 14.52

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 in the reference frame. Comparison using Equation (14.27) 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
```

Sequential search provides the best match 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 14.17

An 80×80 reference frame, target frame, and their difference are displayed in Figure 14.53. 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 Figure 14.53.

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

Horizontal direction =

$$\begin{matrix} -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 \\ -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 & -6 \end{matrix} \tag{14.29}$$

Vertical direction =

$$\begin{matrix} -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\ -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \end{matrix} \tag{14.30}$$

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 towards the target frame. As indicated by the obtained motion vectors, the target image is a version of the reference image moving to the right by 6 pixels and down by 4 pixels.

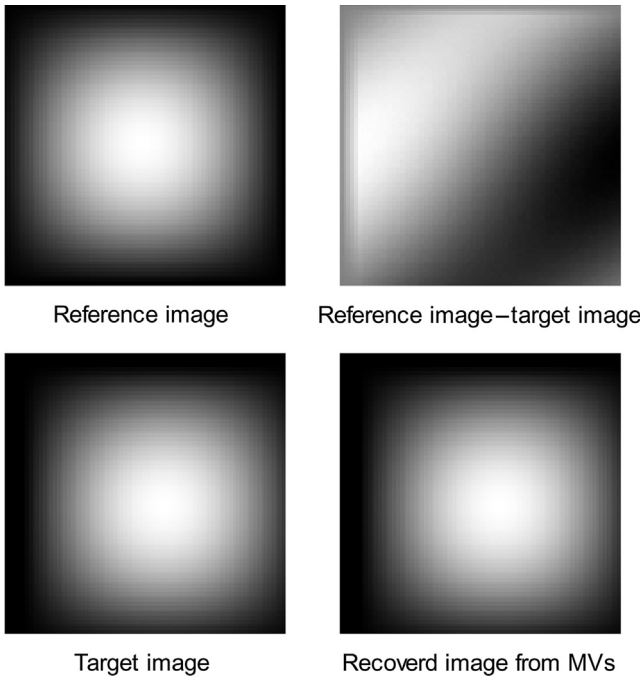


FIGURE 14.53

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

14.11 SUMMARY

1. A digital image consists of pixels. For a grayscale image, each pixel is assigned a grayscale level that presents the 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 red, green, and blue 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. RGB color pixels can be converted to YIQ color pixels. The 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 removes the "pepper and salt" noise in an image.
8. The edge detection filter with Sobel convolution, Laplacian, and Laplacian of Gaussian kernels can detect image boundaries.
9. The grayscale image can be made into a facsimile of the color image by pseudo-color image generation, using the red, green, and blue 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, keeping 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 the 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 of 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, SECAM.
15. Digital video carries the modulated information for each pixel in the YCbCr color space, where Y is the luminance and Cb and Cr are the chrominance. Chroma subsampling creates various digital video formats. The industry standards include CCIR601, CCR601, CIF, and QCIF.
16. Motion compensation of a video sequence produces motion vectors 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.

14.12 PROBLEMS

14.1. 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

14.2. 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

14.3. Given a pixel in an RGB image

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

convert the RGB values to YIQ values.

14.4. Given a pixel of an image in YIQ color format

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

convert the YIQ values back to RGB values.

14.5. Given the 2×2 RGB image,

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

convert the image into grayscale.

14.6. 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.

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

14.7. Consider the following image with a grayscale value ranging from 0 to 7, that is, each pixel is encoded in 3 bits:

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

Perform equalization using the histogram in Problem 14.6, and plot the histogram for the equalized image.

14.8. Consider the following image with a grayscale value ranging from 0 to 7, that is, each pixel is 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.

14.9. Consider 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.

14.10. Consider 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.

14.11. Given the 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

Scale the resultant image pixel value to the range of 0 to 255.

14.12. In Example 14.10, 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?

14.13. In Example 14.10, 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?

14.14. Consider 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 $A(1, 2)$.

14.15. Consider 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 $A(u, v)$.

14.16. Consider the following grayscale image $p(i, j)$:

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

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

14.17. Consider the following DCT coefficients $F(u, v)$:

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

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

14.18. 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.

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

$$[175,\ -2,\ 0,\ 0,\ 0,\ 4,\ 0,\ 0\ ,\ -3\ 7,\ 0,\ 0,\ 0,\ 0,\ -2,\ 0,\ 0,\ \ldots,\ 0]$$

a. Generate the run-length codes for AC coefficients.

b. Perform entropy coding for the run-length codes using the Huffman table.

14.20. Consider 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.

14.21. Consider the following level-1 IDWT coefficients $W(u, v)$ obtained using the Haar wavelet:

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

Apply the IDWT to determine the 2D data.

14.22. Consider 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.

14.23. Consider the following level-1 IDWT coefficients $W(u, v)$ obtained using the Haar 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 the 2D data.

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

14.25. What is the purpose of using interlaced scanning in a traditional NTSC TV system?

14.26. 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?

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

14.28. 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.

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

14.30. 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.

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

14.32. Explain which of the following digital video formats achieves the most data transmission efficiency:

a. 4:4:4

b. 4:2:2

c. 4:2:0

14.33. What is the difference between interlaced scan and progressive scan? Which of the following video systems use progressive scan?

a. CCIR-601

b. CIF

14.34. In motion compensation, which of the following would require more computation? Explain.

a. Finding the motion vector using sequential search

b. Recovering the target frame with the motion vectors and reference frame

14.35. 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 number of subtractions, absolute value calculations, and additions for searching all the motion vectors using the sequential search method.

14.12.1 MATLAB Problems

Use MATLAB to solve Problems 14.36 to 14.42.

14.36. Given the image data "trees.jpg", use MATLAB functions to perform each of the following processes:

a. Use MATLAB to read and display the image.

b. Convert the image to grayscale.

c. Perform histogram equalization for the grayscale image in (b) and display the histogram plots for both the original grayscale image and equalized grayscale image.

d. Perform histogram equalization for the color image in (a) and display the histogram plots of the Y channel for both the original color image and equalized color image.

14.37. Given the image data "cruise.jpg", perform the following linear filtering:

a. Convert the image to grayscale and then create an 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 a Gaussian filter with the following parameters: convolution kernel size = 4, SIGMA = 0.8. Compare the filtered image with the noisy image.

14.38. Given the image data "cruise.jpg", perform the following filtering process:

a. Convert the image to grayscale and then create an 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 median filtering with a convolution kernel size of 4×4.

14.39. Given the image data "cruise.jpg", convert the image to the grayscale 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.

14.40. Given the image data "clipim2.gif', perform the following process:

a. Convert the indexed image to grayscale.

b. Adjust the color transformation functions (sine functions) to make the object indicated by the arrow in the image red and the background color green.

14.41. Given the image data "cruiseorg.tiff", perform JPEG compression by completing the following steps:

a. Convert the image to grayscale.

b. Write a MATLAB program for encoding with the following features: (1) divide the image into 8×8 blocks; (2) transform each block using the discrete-cosine transform; (3) scale and round DCT coefficients with the standard quality factor. Note that using lossless compression with the quantized DCT coefficients is omitted here for a simple simulation.

c. Continue and write a MATLAB program for decoding with the following features: (1) invert the scaling process for quantized DCT coefficients; (2) perform the inverse DCT for each 8×8 image block; (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

14.42. Given the image data "cruiseorg.tiff", perform wavelet-based compression by completing the following steps:

a. Convert the image to grayscale.

b. Write a MATLAB program for encoding with the following features: (1) use an 8-tap Daubechies filter; (2) apply the two-level DWT; (3) perform 8-bit quantization for subbands LL2, LH2, HL2, HH2, LH1, HL1, and HH1 for simulation.

c. Write the MATLAB program for the decoding process.

d. Run the developed MATLAB program to examine the image quality using the following methods:

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.