3 Intensity Transformations and Spatial Filtering

It makes all the difference whether one sees darkness through the light or brightness through the shadows.

David Lindsay

Preview

The term *spatial domain* refers to the image plane itself, and image processing methods in this category are based on direct manipulation of pixels in an image. This is in contrast to image processing in a transform domain which, as we will discuss in Chapters 4 and 6, involves first transforming an image into the transform domain, doing the processing there, and obtaining the inverse transform to bring the results back into the spatial domain. Two principal categories of spatial processing are intensity transformations and spatial filtering. Intensity transformations operate on single pixels of an image for tasks such as contrast manipulation and image thresholding. Spatial filtering performs operations on the neighborhood of every pixel in an image. Examples of spatial filtering include image smoothing and sharpening. In the sections that follow, we discuss a number of "classical" techniques for intensity transformations and spatial filtering.

Upon completion of this chapter, readers should:

- Understand the meaning of spatial domain processing, and how it differs from transform domain processing.
- Be familiar with the principal techniques used for intensity transformations.
- Understand the physical meaning of image histograms and how they can be manipulated for image enhancement.
- Understand the mechanics of spatial filtering, and how spatial filters are formed.
- Understand the principles of spatial convolution and correlation.
- Be familiar with the principal types of spatial filters, and how they are applied.
- Be aware of the relationships between spatial filters, and the fundamental role of lowpass filters.
- Understand how to use combinations of enhancement methods in cases where a single approach is insufficient.

3.1 BACKGROUND

All the image processing techniques discussed in this chapter are implemented in the spatial domain, which we know from the discussion in Section 2.4 is the plane containing the pixels of an image. Spatial domain techniques operate directly on the pixels of an image, as opposed, for example, to the frequency domain (the topic of Chapter 4) in which operations are performed on the Fourier transform of an image, rather than on the image itself. As you will learn in progressing through the book, some image processing tasks are easier or more meaningful to implement in the spatial domain, while others are best suited for other approaches.

THE BASICS OF INTENSITY TRANSFORMATIONS AND SPATIAL FILTERING

The spatial domain processes we discuss in this chapter are based on the expression

$$g(x,y) = T\left[f(x,y)\right] \tag{3-1}$$

where $f(x,y)$ is an input image, $g(x,y)$ is the output image, and T is an operator on f defined over a neighborhood of point (x,y). The operator can be applied to the pixels of a single image (our principal focus in this chapter) or to the pixels of a set of images, such as performing the elementwise sum of a sequence of images for noise reduction, as discussed in Section 2.6. Figure 3.1 shows the basic implementation of Eq. (3-1) on a single image. The point (x_0, y_0) shown is an arbitrary location in the image, and the small region shown is a *neighborhood* of (x_0, y_0), as explained in Section 2.6. Typically, the neighborhood is rectangular, centered on (x_0, y_0), and much smaller in size than the image.

The process that Fig. 3.1 illustrates consists of moving the center of the neighborhood from pixel to pixel, and applying the operator T to the pixels in the neighborhood to yield an output value at that location. Thus, for any specific location (x_0, y_0),

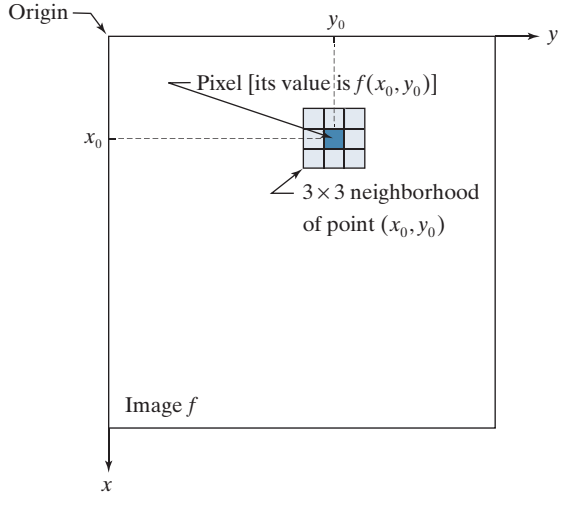


FIGURE 3.1 A 3×3 neighborhood about a point (x_0, y_0) in an image. The neighborhood is moved from pixel to pixel in the image to generate an output image. Recall from Chapter 2 that the value of a pixel at location (x_0, y_0) is $f(x_0, y_0)$, the value of the image at that location.

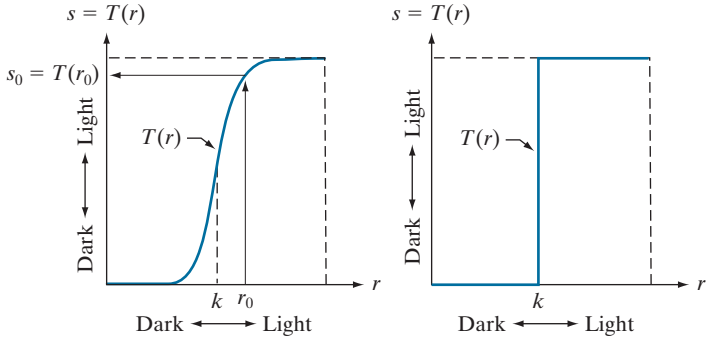
the value of the output image g at those coordinates is equal to the result of applying T to the neighborhood with origin at (x_0, y_0) in f. For example, suppose that the neighborhood is a square of size 3×3 and that operator T is defined as "compute the average intensity of the pixels in the neighborhood." Consider an arbitrary location in an image, say (100,150). The result at that location in the output image, $g(100,150)$, is the sum of $f(100,150)$ and its 8-neighbors, divided by 9. The center of the neighborhood is then moved to the next adjacent location and the procedure is repeated to generate the next value of the output image g. Typically, the process starts at the top left of the input image and proceeds pixel by pixel in a horizontal (vertical) scan, one row (column) at a time. We will discuss this type of neighborhood processing beginning in Section 3.4.

Depending on the size of a neighborhood and its location, part of the neighborhood may lie outside the image. There are two solutions to this: (1) to ignore the values outside the image, or (2) to pad image, as discussed in Section 3.4. The second approach is preferred.

The smallest possible neighborhood is of size 1×1. In this case, g depends only on the value of f at a single point (x, y) and T in Eq. (3-1) becomes an *intensity* (also called a *gray-level,* or *mapping*) *transformation function* of the form

$$s = T(r) \tag{3-2}$$

where, for simplicity in notation, we use s and r to denote, respectively, the intensity of g and f at any point (x, y). For example, if $T(r)$ has the form in Fig. 3.2(a), the result of applying the transformation to every pixel in f to generate the corresponding pixels in g would be to produce an image of higher contrast than the original, by darkening the intensity levels below k and brightening the levels above k. In this technique, sometimes called *contrast stretching* (see Section 3.2), values of r lower than k reduce (darken) the values of s, toward black. The opposite is true for values of r higher than k. Observe how an intensity value r_0 is mapped to obtain the corresponding value s_0. In the limiting case shown in Fig. 3.2(b), $T(r)$ produces a two-level (binary) image. A mapping of this form is called a *thresholding function*. Some fairly simple yet powerful processing approaches can be formulated with intensity transformation functions. In this chapter, we use intensity transformations principally for image enhancement. In Chapter 10, we will use them for image segmentation. Approaches whose results depend only on the intensity at a point sometimes are called *point processing* techniques, as opposed to the *neighborhood processing* techniques discussed in the previous paragraph.



a b

FIGURE 3.2 Intensity transformation functions. (a) Contrast stretching function. (b) Thresholding function.

ABOUT THE EXAMPLES IN THIS CHAPTER

Although intensity transformation and spatial filtering methods span a broad range of applications, most of the examples in this chapter are applications to image enhancement. *Enhancement* is the process of manipulating an image so that the result is more suitable than the original for a specific application. The word *specific* is important, because it establishes at the outset that enhancement techniques are problem-oriented. Thus, for example, a method that is quite useful for enhancing X-ray images may not be the best approach for enhancing infrared images. There is no general "theory" of image enhancement. When an image is processed for visual interpretation, the viewer is the ultimate judge of how well a particular method works. When dealing with machine perception, enhancement is easier to quantify. For example, in an automated character-recognition system, the most appropriate enhancement method is the one that results in the best recognition rate, leaving aside other considerations such as computational requirements of one method versus another. Regardless of the application or method used, image enhancement is one of the most visually appealing areas of image processing. Beginners in image processing generally find enhancement applications interesting and relatively simple to understand. Therefore, using examples from image enhancement to illustrate the spatial processing methods developed in this chapter not only saves having an extra chapter in the book dealing with image enhancement but, more importantly, is an effective approach for introducing newcomers to image processing techniques in the spatial domain. As you progress through the remainder of the book, you will find that the material developed in this chapter has a scope that is much broader than just image enhancement.

3.2 SOME BASIC INTENSITY TRANSFORMATION FUNCTIONS

Intensity transformations are among the simplest of all image processing techniques. As noted in the previous section, we denote the values of pixels, before and after processing, by r and s, respectively. These values are related by a transformation T, as given in Eq. (3-2), that maps a pixel value r into a pixel value s. Because we deal with digital quantities, values of an intensity transformation function typically are stored in a table, and the mappings from r to s are implemented via table lookups. For an 8-bit image, a lookup table containing the values of T will have 256 entries.

As an introduction to intensity transformations, consider Fig. 3.3, which shows three basic types of functions used frequently in image processing: linear (negative and identity transformations), logarithmic (log and inverse-log transformations), and power-law (nth power and nth root transformations). The identity function is the trivial case in which the input and output intensities are identical.

IMAGE NEGATIVES

The negative of an image with intensity levels in the range $[0, L-1]$ is obtained by using the negative transformation function shown in Fig. 3.3, which has the form:

$$s = L - 1 - r \tag{3-3}$$

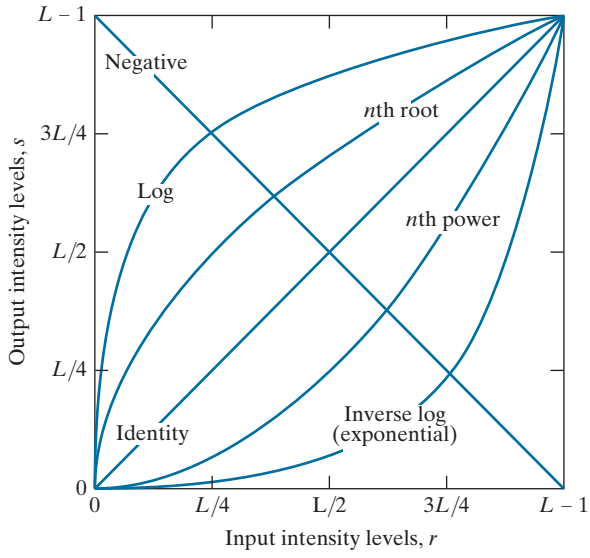
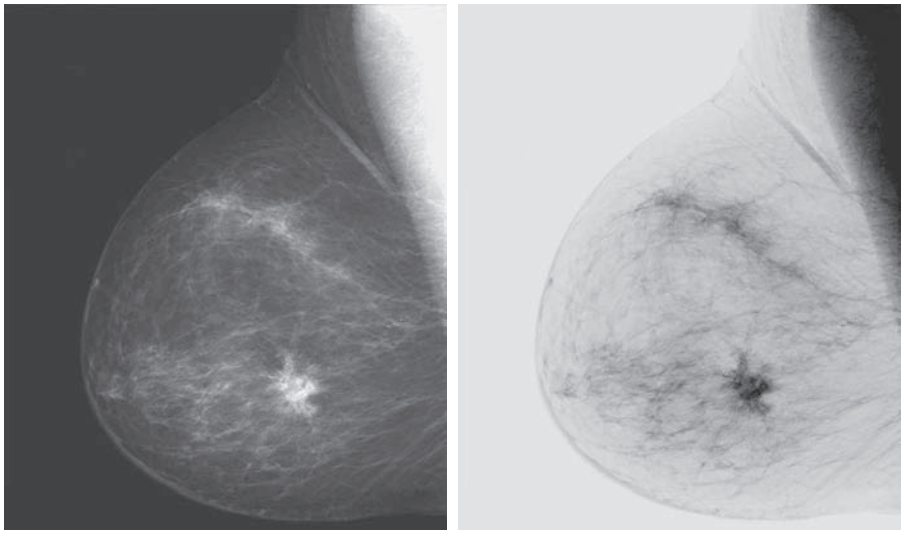


FIGURE 3.3 Some basic intensity transformation functions. Each curve was scaled *independently* so that all curves would fit in the same graph. Our interest here is on the *shapes* of the curves, not on their relative values.

Reversing the intensity levels of a digital image in this manner produces the equivalent of a photographic negative. This type of processing is used, for example, in enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size. Figure 3.4 shows an example. The original image is a digital mammogram showing a small lesion. Despite the fact that the visual content is the same in both images, some viewers find it easier to analyze the fine details of the breast tissue using the negative image.



a b

FIGURE 3.4 (a) A digital mammogram. (b) Negative image obtained using Eq. (3-3). (Image (a) Courtesy of General Electric Medical Systems.)

LOG TRANSFORMATIONS

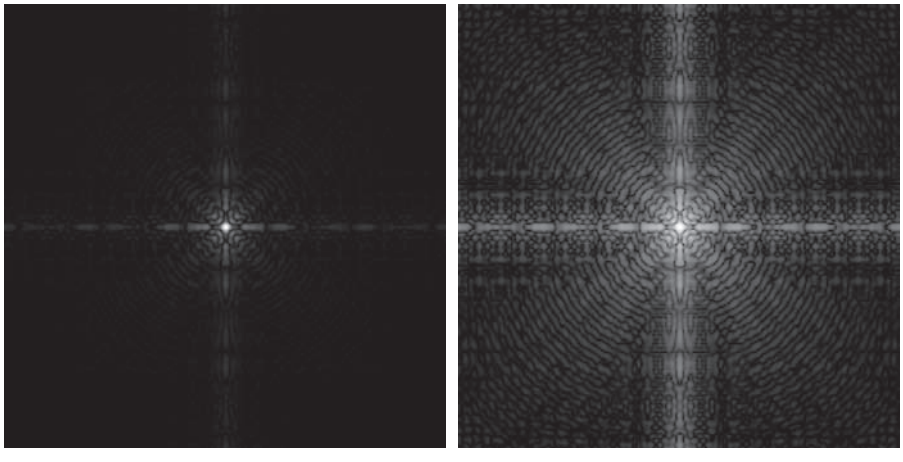
The general form of the log transformation in Fig. 3.3 is

$$s = c\log(1+r) \tag{3-4}$$

where c is a constant and it is assumed that $r \geq 0$. The shape of the log curve in Fig. 3.3 shows that this transformation maps a narrow range of low intensity values in the input into a wider range of output levels. For example, note how input levels in the range $[0, L/4]$ map to output levels to the range $[0, 3L/4]$. Conversely, higher values of input levels are mapped to a narrower range in the output. We use a transformation of this type to expand the values of dark pixels in an image, while compressing the higher-level values. The opposite is true of the inverse log (exponential) transformation.

Any curve having the general shape of the log function shown in Fig. 3.3 would accomplish this spreading/compressing of intensity levels in an image, but the power-law transformations discussed in the next section are much more versatile for this purpose. The log function has the important characteristic that it compresses the dynamic range of pixel values. An example in which pixel values have a large dynamic range is the Fourier spectrum, which we will discuss in Chapter 4. It is not unusual to encounter spectrum values that range from 0 to 10^6 or higher. Processing numbers such as these presents no problems for a computer, but image displays cannot reproduce faithfully such a wide range of values. The net effect is that intensity detail can be lost in the display of a typical Fourier spectrum.

Figure 3.5(a) shows a Fourier spectrum with values in the range 0 to 1.5×10^6. When these values are scaled linearly for display in an 8-bit system, the brightest pixels dominate the display, at the expense of lower (and just as important) values of the spectrum. The effect of this dominance is illustrated vividly by the relatively small area of the image in Fig. 3.5(a) that is not perceived as black. If, instead of displaying the values in this manner, we first apply Eq. (3-4) (with $c = 1$ in this case) to the spectrum values, then the range of values of the result becomes 0 to 6.2. Transforming values in this way enables a greater range of intensities to be shown on the display. Figure 3.5(b) shows the result of scaling the intensity range linearly to the



a b

FIGURE 3.5 (a) Fourier spectrum displayed as a grayscale image. (b) Result of applying the log transformation in Eq. (3-4) with $c = 1$. Both images are scaled to the range $[0, 255]$.

interval [0,255] and showing the spectrum in the same 8-bit display. The level of detail visible in this image as compared to an unmodified display of the spectrum is evident from these two images. Most of the Fourier spectra in image processing publications, including this book, have been scaled in this manner.

POWER-LAW (GAMMA) TRANSFORMATIONS

Power-law transformations have the form

$$s = cr^{\gamma} \tag{3-5}$$

where c and γ are positive constants. Sometimes Eq. (3-5) is written as $s = c(r+\varepsilon)^{\gamma}$ to account for offsets (that is, a measurable output when the input is zero). However, offsets typically are an issue of display calibration, and as a result they are normally ignored in Eq. (3-5). Figure 3.6 shows plots of s as a function of r for various values of γ. As with log transformations, power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels. Note also in Fig. 3.6 that a family of transformations can be obtained simply by varying γ. Curves generated with values of $\gamma > 1$ have exactly the opposite effect as those generated with values of $\gamma < 1$. When $c = \gamma = 1$ Eq. (3-5) reduces to the identity transformation.

The response of many devices used for image capture, printing, and display obey a power law. By convention, the exponent in a power-law equation is referred to as *gamma* [hence our use of this symbol in Eq. (3-5)]. The process used to correct these power-law response phenomena is called *gamma correction* or *gamma encoding*. For example, cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5. As the curve for $\gamma = 2.5$ in Fig. 3.6 shows, such display systems would tend to produce

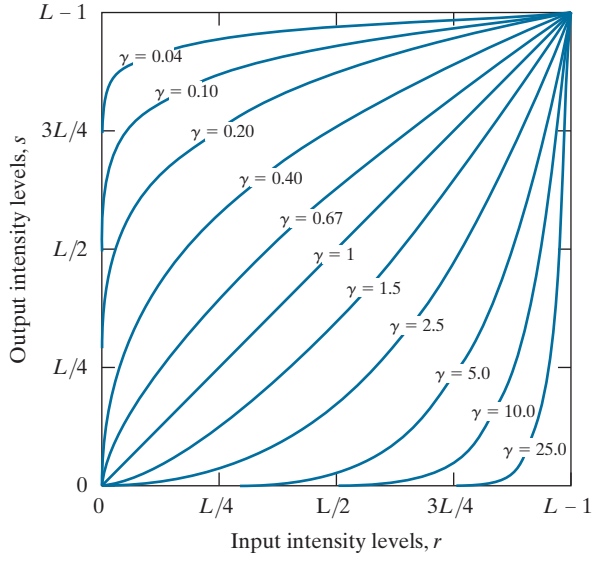
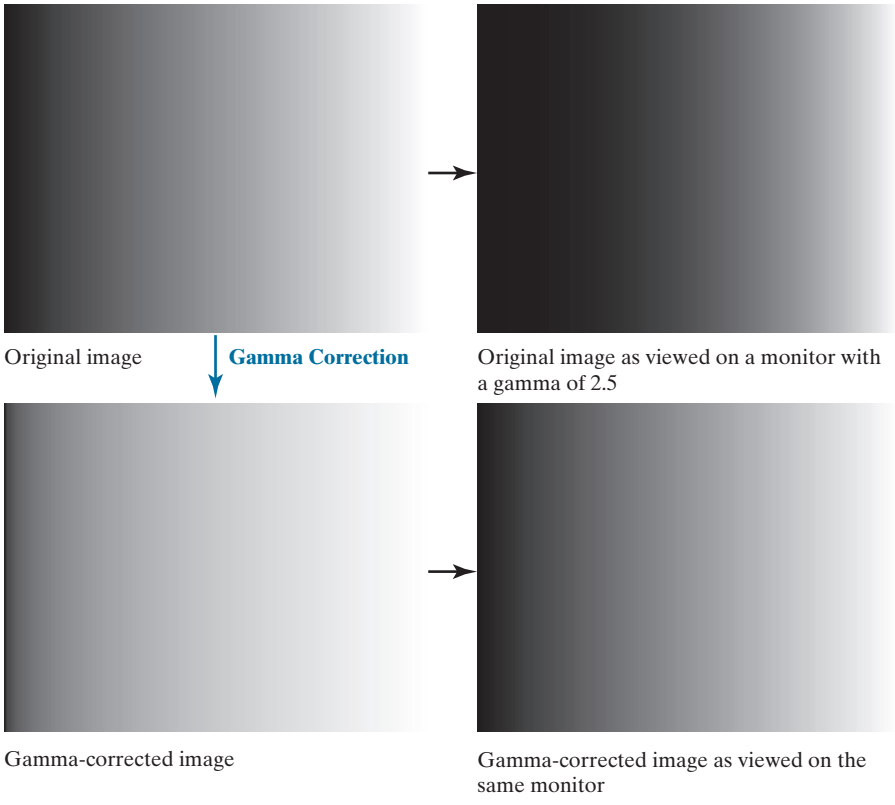


FIGURE 3.6 Plots of the gamma equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases). Each curve was scaled *independently* so that all curves would fit in the same graph. Our interest here is on the *shapes* of the curves, not on their relative values.

a b
c d

FIGURE 3.7
(a) Intensity ramp image. (b) Image as viewed on a simulated monitor with a gamma of 2.5. (c) Gamma-corrected image. (d) Corrected image as viewed on the same monitor. Compare (d) and (a).



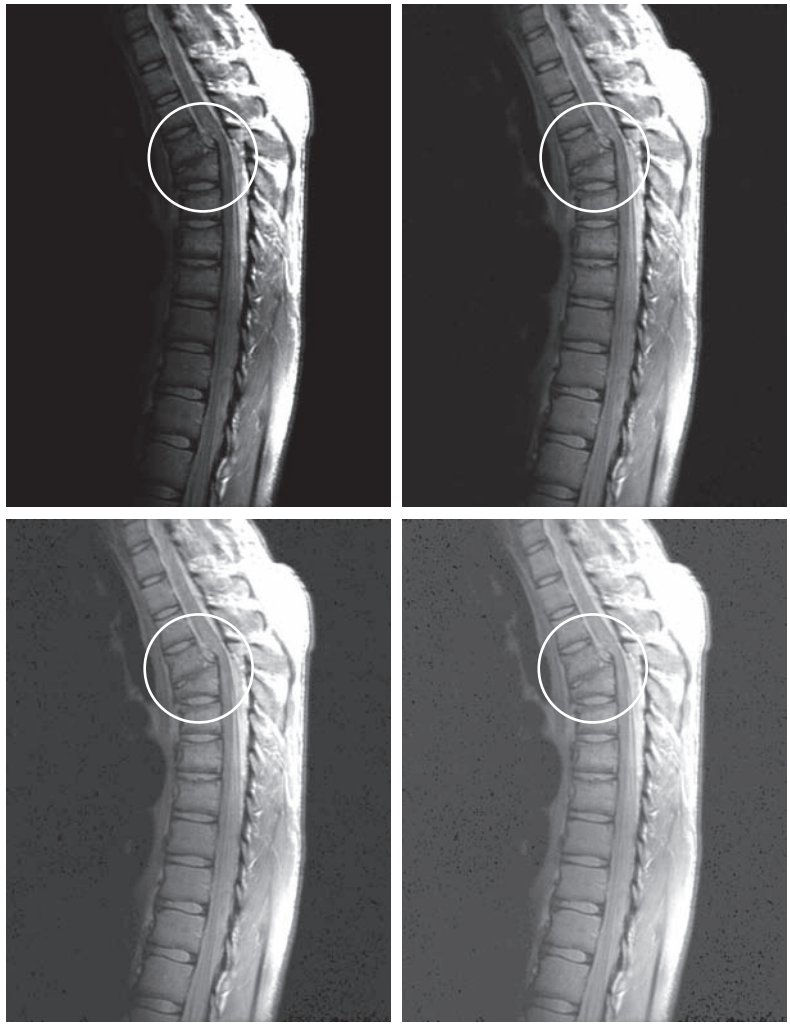
Sometimes, a higher gamma makes the displayed image look better to viewers than the original because of an increase in contrast. However, the objective of gamma correction is to produce a *faithful* display of an input image.

images that are darker than intended. Figure 3.7 illustrates this effect. Figure 3.7(a) is an image of an intensity ramp displayed in a monitor with a gamma of 2.5. As expected, the output of the monitor appears darker than the input, as Fig. 3.7(b) shows.

In this case, gamma correction consists of using the transformation $s = r^{1/2.5} = r^{0.4}$ to preprocess the image before inputting it into the monitor. Figure 3.7(c) is the result. When input into the same monitor, the gamma-corrected image produces an output that is close in appearance to the original image, as Fig. 3.7(d) shows. A similar analysis as above would apply to other imaging devices, such as scanners and printers, the difference being the device-dependent value of gamma (Poynton [1996]).

EXAMPLE 3.1: Contrast enhancement using power-law intensity transformations.

In addition to gamma correction, power-law transformations are useful for general-purpose contrast manipulation. Figure 3.8(a) shows a magnetic resonance image (MRI) of a human upper thoracic spine with a fracture dislocation. The fracture is visible in the region highlighted by the circle. Because the image is predominantly dark, an expansion of intensity levels is desirable. This can be accomplished using a power-law transformation with a fractional exponent. The other images shown in the figure were obtained by processing Fig. 3.8(a) with the power-law transformation function of Eq. (3-5). The values



a b
c d

FIGURE 3.8
(a) Magnetic resonance image (MRI) of a fractured human spine (the region of the fracture is enclosed by the circle). (b)–(d) Results of applying the transformation in Eq. (3-5) with $c = 1$ and $\gamma = 0.6$, 0.4, and 0.3, respectively. (Original image courtesy of Dr. David R. Pickens, Department of Radiology and Radiological Sciences, Vanderbilt University Medical Center.)

of gamma corresponding to images (b) through (d) are 0.6, 0.4, and 0.3, respectively ($c = 1$ in all cases). Observe that as gamma decreased from 0.6 to 0.4, more detail became visible. A further decrease of gamma to 0.3 enhanced a little more detail in the background, but began to reduce contrast to the point where the image started to have a very slight "washed-out" appearance, especially in the background. The best enhancement in terms of contrast and discernible detail was obtained with $\gamma = 0.4$. A value of $\gamma = 0.3$ is an approximate limit below which contrast in this particular image would be reduced to an unacceptable level.

EXAMPLE 3.2: Another illustration of power-law transformations.

Figure 3.9(a) shows the opposite problem of that presented in Fig. 3.8(a). The image to be processed

a b
c d

FIGURE 3.9
(a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3-5) with $\gamma = 3.0$, 4.0, and 5.0, respectively. ($c = 1$ in all cases.) (Original image courtesy of NASA.)

now has a washed-out appearance, indicating that a compression of intensity levels is desirable. This can be accomplished with Eq. (3-5) using values of γ greater than 1. The results of processing Fig. 3.9(a) with $\gamma = 3.0$, 4.0, and 5.0 are shown in Figs. 3.9(b) through (d), respectively. Suitable results were obtained using gamma values of 3.0 and 4.0. The latter result has a slightly more appealing appearance because it has higher contrast. This is true also of the result obtained with $\gamma = 5.0$. For example, the airport runways near the middle of the image appears clearer in Fig. 3.9(d) than in any of the other three images.

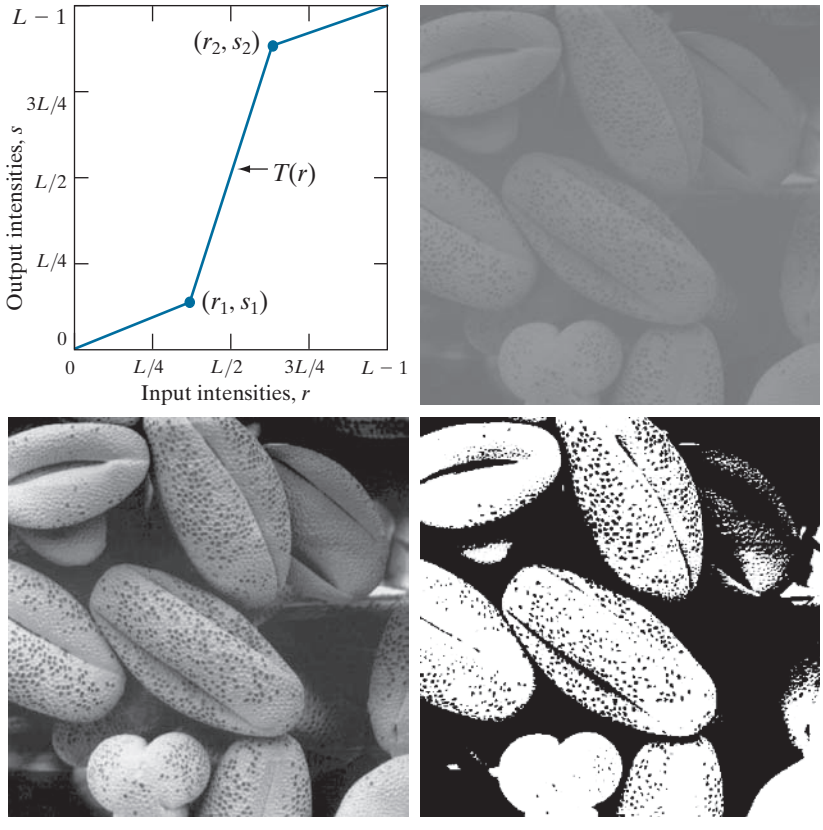
PIECEWISE LINEAR TRANSFORMATION FUNCTIONS

An approach complementary to the methods discussed in the previous three sections is to use piecewise linear functions. The advantage of these functions over those discussed thus far is that the form of piecewise functions can be arbitrarily complex. In fact, as you will see shortly, a practical implementation of some important transformations can be formulated only as piecewise linear functions. The main disadvantage of these functions is that their specification requires considerable user input.

Contrast Stretching

Low-contrast images can result from poor illumination, lack of dynamic range in the imaging sensor, or even the wrong setting of a lens aperture during image acquisition. *Contrast stretching* expands the range of intensity levels in an image so that it spans the ideal full intensity range of the recording medium or display device.

Figure 3.10(a) shows a typical transformation used for contrast stretching. The locations of points (r_1, s_1) and (r_2, s_2) control the shape of the transformation function. If $r_1 = s_1$ and $r_2 = s_2$ the transformation is a linear function that produces no changes in intensity. If $r_1 = r_2$, $s_1 = 0$, and $s_2 = L - 1$ the transformation becomes a *thresholding function* that creates a binary image [see Fig. 3.2(b)]. Intermediate values of (r_1, s_1) and (s_2, r_2) produce various degrees of spread in the intensity levels of the output image, thus affecting its contrast. In general, $r_1 \leq r_2$ and $s_1 \leq s_2$ is assumed so that the function is single valued and monotonically increasing. This preserves the order of intensity levels, thus preventing the creation of intensity artifacts. Figure 3.10(b) shows an 8-bit image with low contrast. Figure 3.10(c) shows the result of contrast stretching, obtained by setting $(r_1, s_1) = (r_{\min}, 0)$ and $(r_2, s_2) = (r_{\max}, L - 1)$, where $r_{\min}$ and $r_{\max}$ denote the minimum and maximum intensity levels in the input image,



a b
c d

FIGURE 3.10
Contrast stretching. (a) Piecewise linear transformation function. (b) A low-contrast electron microscope image of pollen, magnified 700 times. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

a b

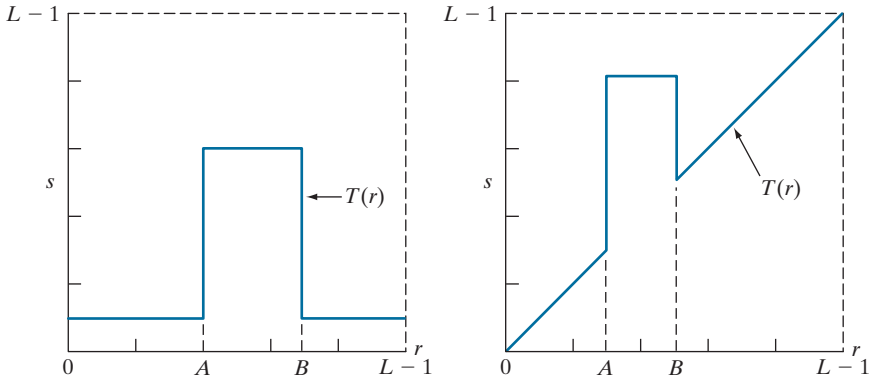


FIGURE 3.11
(a) This transformation function highlights range $[A, B]$ and reduces all other intensities to a lower level. (b) This function highlights range $[A, B]$ and leaves other intensities unchanged.

respectively. The transformation stretched the intensity levels linearly to the full intensity range, $[0, L-1]$. Finally, Fig. 3.10(d) shows the result of using the thresholding function, with $(r_1, s_1) = (m, 0)$ and $(r_2, s_2) = (m, L-1)$, where m is the mean intensity level in the image.

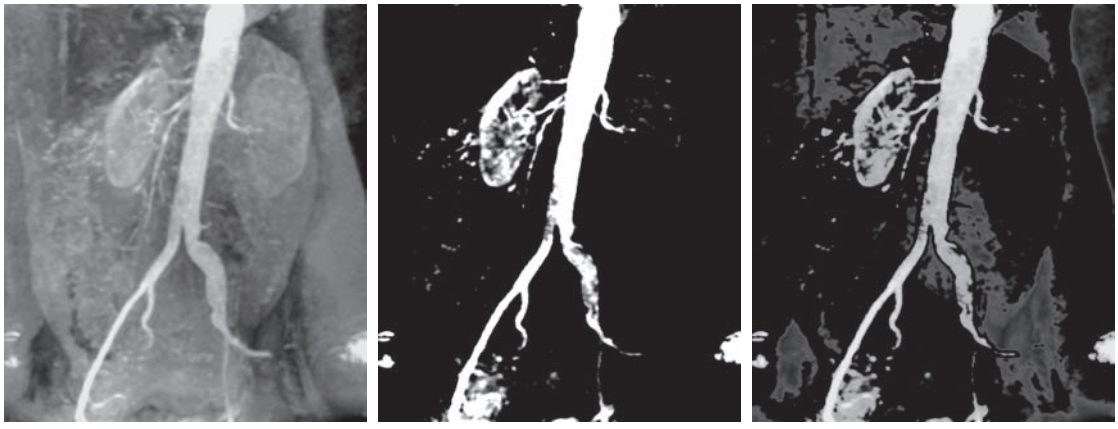
Intensity-Level Slicing

There are applications in which it is of interest to highlight a specific range of intensities in an image. Some of these applications include enhancing features in satellite imagery, such as masses of water, and enhancing flaws in X-ray images. The method, called *intensity-level slicing*, can be implemented in several ways, but most are variations of two basic themes. One approach is to display in one value (say, white) all the values in the range of interest and in another (say, black) all other intensities. This transformation, shown in Fig. 3.11(a), produces a binary image. The second approach, based on the transformation in Fig. 3.11(b), brightens (or darkens) the desired range of intensities, but leaves all other intensity levels in the image unchanged.

EXAMPLE 3.3: Intensity-level slicing.

Figure 3.12(a) is an aortic angiogram near the kidney area (see Section 1.3 for details on this image). The objective of this example is to use intensity-level slicing to enhance the major blood vessels that appear lighter than the background, as a result of an injected contrast medium. Figure 3.12(b) shows the result of using a transformation of the form in Fig. 3.11(a). The selected band was near the top of the intensity scale because the range of interest is brighter than the background. The net result of this transformation is that the blood vessel and parts of the kidneys appear white, while all other intensities are black. This type of enhancement produces a binary image, and is useful for studying the shape characteristics of the flow of the contrast medium (to detect blockages, for example).

If interest lies in the actual intensity values of the region of interest, we can use the transformation of the form shown in Fig. 3.11(b). Figure 3.12(c) shows the result of using such a transformation in which a band of intensities in the mid-gray region around the mean intensity was set to black, while all other intensities were left unchanged. Here, we see that the gray-level tonality of the major blood vessels and part of the kidney area were left intact. Such a result might be useful when interest lies in measuring the actual flow of the contrast medium as a function of time in a sequence of images.



a b c

FIGURE 3.12 (a) Aortic angiogram. (b) Result of using a slicing transformation of the type illustrated in Fig. 3.11(a), with the range of intensities of interest selected in the upper end of the gray scale. (c) Result of using the transformation in Fig. 3.11(b), with the selected range set near black, so that the grays in the area of the blood vessels and kidneys were preserved. (Original image courtesy of Dr. Thomas R. Gest, University of Michigan Medical School.)

Bit-Plane Slicing

Pixel values are integers composed of bits. For example, values in a 256-level grayscale image are composed of 8 bits (one byte). Instead of highlighting intensity-level ranges, as 3.3, we could highlight the contribution made to total image appearance by specific bits. As Fig. 3.13 illustrates, an 8-bit image may be considered as being composed of eight one-bit planes, with plane 1 containing the lowest-order bit of all pixels in the image, and plane 8 all the highest-order bits.

Figure 3.14(a) shows an 8-bit grayscale image and Figs. 3.14(b) through (i) are its eight one-bit planes, with Fig. 3.14(b) corresponding to the highest-order bit. Observe that the four higher-order bit planes, especially the first two, contain a significant amount of the visually-significant data. The lower-order planes contribute to more subtle intensity details in the image. The original image has a gray border whose intensity is 194. Notice that the corresponding borders of some of the bit

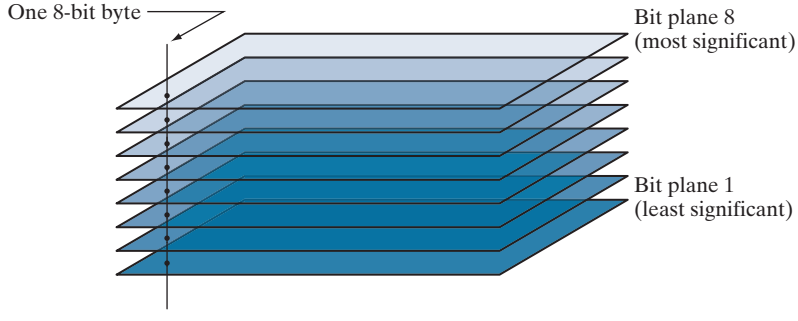
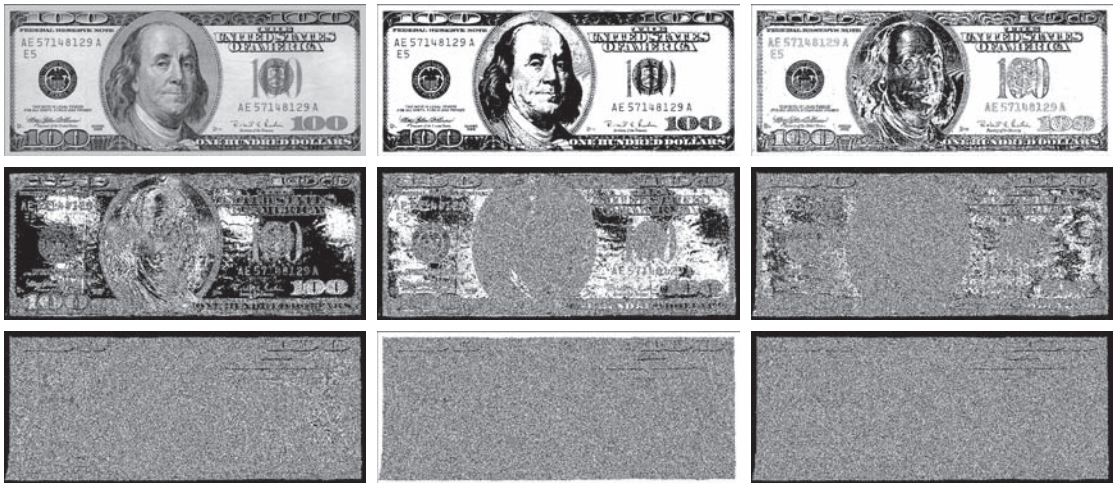


FIGURE 3.13 Bit-planes of an 8-bit image.



a b c
d e f
g h i

FIGURE 3.14 (a) An 8-bit gray-scale image of size 550×1192 pixels. (b) through (i) Bit planes 8 through 1, with bit plane 1 corresponding to the least significant bit. Each bit plane is a binary image..

planes are black (0), while others are white (1). To see why, consider a pixel in, say, the middle of the lower border of Fig. 3.14(a). The corresponding pixels in the bit planes, starting with the highest-order plane, have values 1 1 0 0 0 0 1 0, which is the binary representation of decimal 194. The value of any pixel in the original image can be similarly reconstructed from its corresponding binary-valued pixels in the bit planes by converting an 8-bit binary sequence to decimal.

The binary image for the 8th bit plane of an 8-bit image can be obtained by thresholding the input image with a transformation function that maps to 0 intensity values between 0 and 127, and maps to 1 values between 128 and 255. The binary image in Fig. 3.14(b) was obtained in this manner. It is left as an exercise (see Problem 3.3) to obtain the transformation functions for generating the other bit planes.

Decomposing an image into its bit planes is useful for analyzing the relative importance of each bit in the image, a process that aids in determining the adequacy of the number of bits used to quantize the image. Also, this type of decomposition is useful for image compression (the topic of Chapter 8), in which fewer than all planes are used in reconstructing an image. For example, Fig. 3.15(a) shows an image reconstructed using bit planes 8 and 7 of the preceding decomposition. The reconstruction is done by multiplying the pixels of the nth plane by the constant 2^{n-1}. This converts the nth significant binary bit to decimal. Each bit plane is multiplied by the corresponding constant, and all resulting planes are added to obtain the grayscale image. Thus, to obtain Fig. 3.15(a), we multiplied bit plane 8 by 128, bit plane 7 by 64, and added the two planes. Although the main features of the original image were restored, the reconstructed image appears flat, especially in the background. This

a b c **FIGURE 3.15** Image reconstructed from bit planes: (a) 8 and 7; (b) 8, 7, and 6; (c) 8, 7, 6, and 5.

is not surprising, because two planes can produce only four distinct intensity levels. Adding plane 6 to the reconstruction helped the situation, as Fig. 3.15(b) shows. Note that the background of this image has perceptible false contouring. This effect is reduced significantly by adding the 5th plane to the reconstruction, as Fig. 3.15(c) illustrates. Using more planes in the reconstruction would not contribute significantly to the appearance of this image. Thus, we conclude that, in this example, storing the four highest-order bit planes would allow us to reconstruct the original image in acceptable detail. Storing these four planes instead of the original image requires 50% less storage.

3.3 HISTOGRAM PROCESSING

Let r_k, for $k = 0,1,2,\ldots,L-1$, denote the intensities of an L-level digital image, $f(x,y)$. The *unnormalized histogram* of f is defined as

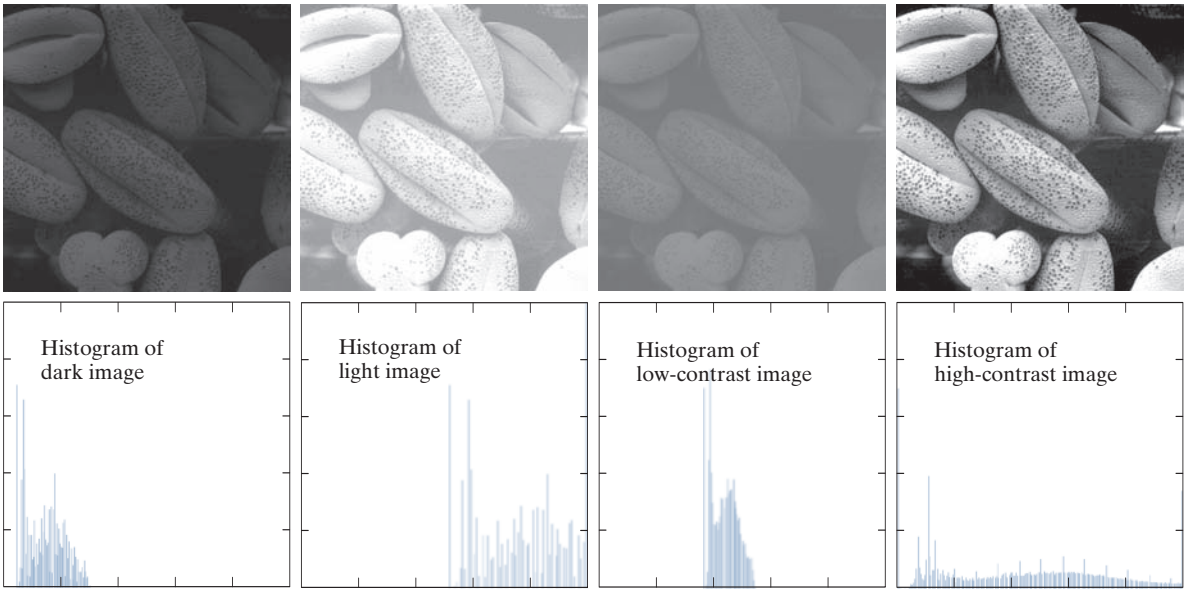
$$h(r_k) = n_k \quad \text{for } k = 0,1,2,\ldots,L-1 \tag{3-6}$$

where n_k is the number of pixels in f with intensity r_k, and the subdivisions of the intensity scale are called *histogram bins*. Similarly, the *normalized histogram* of f is defined as

$$p(r_k) = \frac{h(r_k)}{MN} = \frac{n_k}{MN} \tag{3-7}$$

where, as usual, M and N are the number of image rows and columns, respectively. Mostly, we work with normalized histograms, which we refer to simply as *histograms* or *image histograms*. The sum of $p(r_k)$ for all values of k is always 1. The components of $p(r_k)$ are estimates of the probabilities of intensity levels occurring in an image. As you will learn in this section, histogram manipulation is a fundamental tool in image processing. Histograms are simple to compute and are also suitable for fast hardware implementations, thus making histogram-based techniques a popular tool for real-time image processing.

Histogram shape is related to image appearance. For example, Fig. 3.16 shows images with four basic intensity characteristics: dark, light, low contrast, and high contrast; the image histograms are also shown. We note in the dark image that the most populated histogram bins are concentrated on the lower (dark) end of the intensity scale. Similarly, the most populated bins of the light image are biased toward the higher end of the scale. An image with low contrast has a narrow histo-



a b c d

FIGURE 3.16 Four image types and their corresponding histograms. (a) dark; (b) light; (c) low contrast; (d) high contrast. The horizontal axis of the histograms are values of r_k and the vertical axis are values of $p(r_k)$.

gram located typically toward the middle of the intensity scale, as Fig. 3.16(c) shows. For a monochrome image, this implies a dull, washed-out gray look. Finally, we see that the components of the histogram of the high-contrast image cover a wide range of the intensity scale, and the distribution of pixels is not too far from uniform, with few bins being much higher than the others. Intuitively, it is reasonable to conclude that an image whose pixels tend to occupy the entire range of possible intensity levels and, in addition, tend to be distributed uniformly, will have an appearance of high contrast and will exhibit a large variety of gray tones. The net effect will be an image that shows a great deal of gray-level detail and has a high dynamic range. As you will see shortly, it is possible to develop a transformation function that can achieve this effect automatically, using only the histogram of an input image.

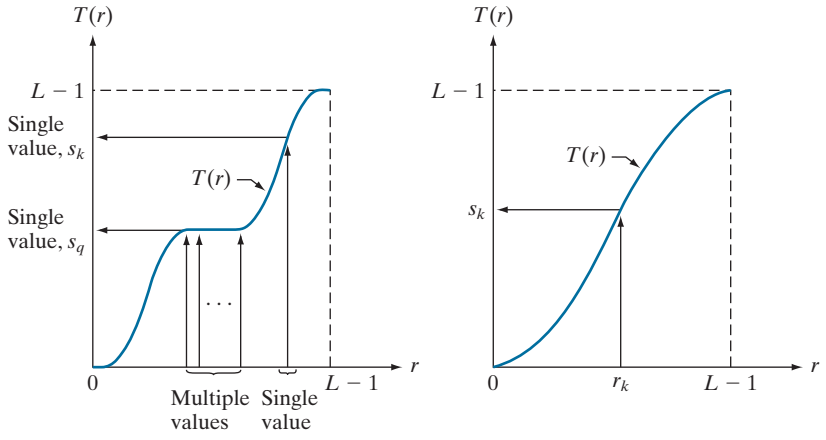
HISTOGRAM EQUALIZATION

Assuming initially continuous intensity values, let the variable r denote the intensities of an image to be processed. As usual, we assume that r is in the range $[0, L-1]$, with $r = 0$ representing black and $r = L-1$ representing white. For r satisfying these conditions, we focus attention on transformations (intensity mappings) of the form

$$s = T(r) \qquad 0 \le r \le L-1 \tag{3-8}$$

a b

FIGURE 3.17
(a) Monotonic increasing function, showing how multiple values can map to a single value. (b) Strictly monotonic increasing function. This is a one-to-one mapping, both ways.



that produce an output intensity value, s, for a given intensity value r in the input image. We assume that

(a) $T(r)$ is a monotonic[†] increasing function in the interval $0 \le r \le L-1$; and
(b) $0 \le T(r) \le L-1$ for $0 \le r \le L-1$.

In some formulations to be discussed shortly, we use the inverse transformation

$$r = T^{-1}(s) \qquad 0 \le s \le L-1 \tag{3-9}$$

in which case we change condition (a) to:

(a′) $T(r)$ is a *strictly* monotonic increasing function in the interval $0 \le r \le L-1$.

The condition in (a) that $T(r)$ be monotonically increasing guarantees that output intensity values will never be less than corresponding input values, thus preventing artifacts created by reversals of intensity. Condition (b) guarantees that the range of output intensities is the same as the input. Finally, condition (a′) guarantees that the mappings from s back to r will be one-to-one, thus preventing ambiguities.

Figure 3.17(a) shows a function that satisfies conditions (a) and (b). Here, we see that it is possible for multiple input values to map to a single output value and still satisfy these two conditions. That is, a monotonic transformation function performs a one-to-one or many-to-one mapping. This is perfectly fine when mapping from r to s. However, Fig. 3.17(a) presents a problem if we wanted to recover the values of r uniquely from the mapped values (inverse mapping can be visualized by reversing the direction of the arrows). This would be possible for the inverse mapping of s_k in Fig. 3.17(a), but the inverse mapping of s_q is a range of values, which, of course, prevents us in general from recovering the original value of r that resulted

[†] A function $T(r)$ is a *monotonic increasing* function if $T(r_2) \ge T(r_1)$ for $r_2 > r_1$. $T(r)$ is a *strictly monotonic increasing* function if $T(r_2) > T(r_1)$ for $r_2 > r_1$. Similar definitions apply to a monotonic decreasing function.

in s_q. As Fig. 3.17(b) shows, requiring that $T(r)$ be strictly monotonic guarantees that the inverse mappings will be *single valued* (i.e., the mapping is one-to-one in both directions).This is a theoretical requirement that will allow us to derive some important histogram processing techniques later in this chapter. Because images are stored using integer intensity values, we are forced to round all results to their nearest integer values. This often results in strict monotonicity not being satisfied, which implies inverse transformations that may not be unique. Fortunately, this problem is not difficult to handle in the discrete case, as Example 3.7 in this section illustrates.

The intensity of an image may be viewed as a random variable in the interval $[0, L-1]$. Let $p_r(r)$ and $p_s(s)$ denote the PDFs of intensity values r and s in two different images. The subscripts on p indicate that p_r and p_s are different functions. A fundamental result from probability theory is that if $p_r(r)$ and $T(r)$ are known, and $T(r)$ is continuous and differentiable over the range of values of interest, then the PDF of the transformed (mapped) variable s can be obtained as

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| \tag{3-10}$$

Thus, we see that the PDF of the output intensity variable, s, is determined by the PDF of the input intensities and the transformation function used [recall that r and s are related by $T(r)$].

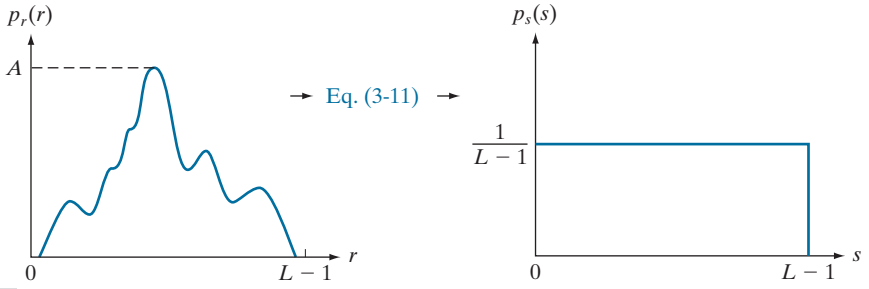
A transformation function of particular importance in image processing is

$$s = T(r) = (L-1)\int_0^r p_r(w)\,dw \tag{3-11}$$

where w is a dummy variable of integration. The integral on the right side is the *cumulative distribution function* (CDF) of random variable r. Because PDFs always are positive, and the integral of a function is the area under the function, it follows that the transformation function of Eq. (3-11) satisfies condition (a). This is because the area under the function cannot decrease as r increases. When the upper limit in this equation is $r = (L-1)$ the integral evaluates to 1, as it must for a PDF. Thus, the maximum value of s is $L-1$, and condition (b) is satisfied also.

We use Eq. (3-10) to find the $p_s(s)$ corresponding to the transformation just discussed. We know from Leibniz's rule in calculus that the derivative of a definite integral with respect to its upper limit is the integrand evaluated at the limit. That is,

$$\begin{aligned}\frac{ds}{dr} &= \frac{dT(r)}{dr} \\ &= (L-1)\frac{d}{dr}\left[\int_0^r p_r(w)\,dw\right] \\ &= (L-1)p_r(r)\end{aligned} \tag{3-12}$$



a b

FIGURE 3.18 (a) An arbitrary PDF. (b) Result of applying Eq. (3-11) to the input PDF. The resulting PDF is always uniform, independently of the shape of the input.

Substituting this result for dr/ds in Eq. (3-10), and noting that all probability values are positive, gives the result

$$\begin{aligned} p_s(s) &= p_r(r)\left|\frac{dr}{ds}\right| \\ &= p_r(r)\left|\frac{1}{(L-1)p_r(r)}\right| \\ &= \frac{1}{L-1} \qquad 0 \le s \le L-1 \end{aligned} \tag{3-13}$$

We recognize the form of $p_s(s)$ in the last line of this equation as a *uniform* probability density function. Thus, performing the intensity transformation in Eq. (3-11) yields a random variable, s, characterized by a uniform PDF. What is important is that $p_s(s)$ in Eq. (3-13) will *always* be uniform, *independently* of the form of $p_r(r)$. Figure 3.18 and the following example illustrate these concepts.

EXAMPLE 3.4: Illustration of Eqs. (3-11) and (3-13).

Suppose that the (continuous) intensity values in an image have the PDF

$$p_r(r) = \begin{cases} \dfrac{2r}{(L-1)^2} & \text{for } 0 \le r \le L-1 \\ 0 & \text{otherwise} \end{cases}$$

From Eq. (3-11)

$$s = T(r) = (L-1)\int_0^r p_r(w)\,dw = \frac{2}{L-1}\int_0^r w\,dw = \frac{r^2}{L-1}$$

Suppose that we form a new image with intensities, s, obtained using this transformation; that is, the s values are formed by squaring the corresponding intensity values of the input image, then dividing them by $L-1$. We can verify that the PDF of the intensities in the new image, $p_s(s)$, is uniform by substituting $p_r(r)$ into Eq. (3-13), and using the fact that $s = r^2/(L-1)$; that is,

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| = \frac{2r}{(L-1)^2}\left|\left[\frac{ds}{dr}\right]^{-1}\right|$$

$$= \frac{2r}{(L-1)^2}\left|\left[\frac{d}{dr}\frac{r^2}{L-1}\right]^{-1}\right| = \frac{2r}{(L-1)^2}\left|\frac{(L-1)}{2r}\right| = \frac{1}{L-1}$$

The last step follows because r is nonnegative and $L > 1$. As expected, the result is a uniform PDF.

For discrete values, we work with probabilities and summations instead of probability density functions and integrals (but the requirement of monotonicity stated earlier still applies). Recall that the probability of occurrence of intensity level r_k in a digital image is approximated by

$$p_r(r_k) = \frac{n_k}{MN} \tag{3-14}$$

where MN is the total number of pixels in the image, and n_k denotes the number of pixels that have intensity r_k. As noted in the beginning of this section, $p_r(r_k)$, with $r_k \in [0, L-1]$, is commonly referred to as a normalized image histogram.

The discrete form of the transformation in Eq. (3-11) is

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) \qquad k = 0, 1, 2, \ldots, L-1 \tag{3-15}$$

where, as before, L is the number of possible intensity levels in the image (e.g., 256 for an 8-bit image). Thus, a processed (output) image is obtained by using Eq. (3-15) to map each pixel in the input image with intensity r_k into a corresponding pixel with level s_k in the output image, This is called a *histogram equalization* or *histogram linearization* transformation. It is not difficult to show (see Problem 3.9) that this transformation satisfies conditions (a) and (b) stated previously in this section.

EXAMPLE 3.5: Illustration of the mechanics of histogram equalization.

It will be helpful to work through a simple example. Suppose that a 3-bit image ($L = 8$) of size 64×64 pixels ($MN = 4096$) has the intensity distribution in Table 3.1, where the intensity levels are integers in the range $[0, L-1] = [0, 7]$. The histogram of this image is sketched in Fig. 3.19(a). Values of the histogram equalization transformation function are obtained using Eq. (3-15). For instance,

$$s_0 = T(r_0) = 7\sum_{j=0}^{0} p_r(r_j) = 7p_r(r_0) = 1.33$$

TABLE 3.1
Intensity distribution and histogram values for a 3-bit, 64×64 digital image.

r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

Similarly, $s_1 = T(r_1) = 3.08$, $s_2 = 4.55$, $s_3 = 5.67$, $s_4 = 6.23$, $s_5 = 6.65$, $s_6 = 6.86$, and $s_7 = 7.00$. This transformation function has the staircase shape shown in Fig. 3.19(b).

At this point, the s values are fractional because they were generated by summing probability values, so we round them to their nearest integer values in the range $[0, 7]$:

$$s_0 = 1.33 \rightarrow 1 \qquad s_2 = 4.55 \rightarrow 5 \qquad s_4 = 6.23 \rightarrow 6 \qquad s_6 = 6.86 \rightarrow 7$$
$$s_1 = 3.08 \rightarrow 3 \qquad s_3 = 5.67 \rightarrow 6 \qquad s_5 = 6.65 \rightarrow 7 \qquad s_7 = 7.00 \rightarrow 7$$

These are the values of the equalized histogram. Observe that the transformation yielded only five distinct intensity levels. Because $r_0 = 0$ was mapped to $s_0 = 1$, there are 790 pixels in the histogram equalized image with this value (see Table 3.1). Also, there are 1023 pixels with a value of $s_1 = 3$ and 850 pixels with a value of $s_2 = 5$. However, both r_3 and r_4 were mapped to the same value, 6, so there are $(656 + 329) = 985$ pixels in the equalized image with this value. Similarly, there are $(245 + 122 + 81) = 448$ pixels with a value of 7 in the histogram equalized image. Dividing these numbers by $MN = 4096$ yielded the equalized histogram in Fig. 3.19(c).

Because a histogram is an approximation to a PDF, and no new allowed intensity levels are created in the process, perfectly flat histograms are rare in practical applications of histogram equalization using the method just discussed. Thus, unlike its continuous counterpart, it cannot be proved in general that discrete histogram equalization using Eq. (3-15) results in a uniform histogram (we will introduce later in

a b c

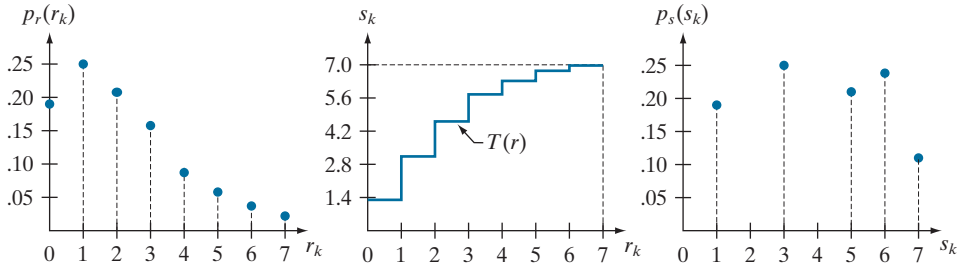


FIGURE 3.19
Histogram equalization. (a) Original histogram. (b) Transformation function. (c) Equalized histogram.

this section an approach for removing this limitation). However, as you will see shortly, using Eq. (3-15) has the general tendency to spread the histogram of the input image so that the intensity levels of the equalized image span a wider range of the intensity scale. The net result is contrast enhancement.

We discussed earlier the advantages of having intensity values that span the entire gray scale. The method just derived produces intensities that have this tendency, and also has the advantage that it is fully automatic. In other words, the process of histogram equalization consists entirely of implementing Eq. (3-15), which is based on information that can be extracted directly from a given image, without the need for any parameter specifications. This automatic, "hands-off" characteristic is important.

The inverse transformation from s back to r is denoted by

$$r_k = T^{-1}(s_k) \tag{3-16}$$

It can be shown (see Problem 3.9) that this inverse transformation satisfies conditions (a′) and (b) defined earlier *only* if *all* intensity levels are present in the input image. This implies that none of the bins of the image histogram are empty. Although the inverse transformation is not used in histogram equalization, it plays a central role in the histogram-matching scheme developed after the following example.

EXAMPLE 3.6: Histogram equalization.

The left column in Fig. 3.20 shows the four images from Fig. 3.16, and the center column shows the result of performing histogram equalization on each of these images. The first three results from top to bottom show significant improvement. As expected, histogram equalization did not have much effect on the fourth image because its intensities span almost the full scale already. Figure 3.21 shows the transformation functions used to generate the equalized images in Fig. 3.20. These functions were generated using Eq. (3-15). Observe that transformation (4) is nearly linear, indicating that the inputs were mapped to nearly equal outputs. Shown is the mapping of an input value r_k to a corresponding output value s_k. In this case, the mapping was for image 1 (on the top left of Fig. 3.21), and indicates that a dark value was mapped to a much lighter one, thus contributing to the brightness of the output image.

The third column in Fig. 3.20 shows the histograms of the equalized images. While all the histograms are different, the histogram-equalized images themselves are visually very similar. This is not totally unexpected because the basic difference between the images on the left column is one of contrast, not content. Because the images have the same content, the increase in contrast resulting from histogram equalization was enough to render any intensity differences between the equalized images visually indistinguishable. Given the significant range of contrast differences in the original images, this example illustrates the power of histogram equalization as an adaptive, autonomous contrast-enhancement tool.

HISTOGRAM MATCHING (SPECIFICATION)

As explained in the last section, histogram equalization produces a transformation function that seeks to generate an output image with a uniform histogram. When automatic enhancement is desired, this is a good approach to consider because the

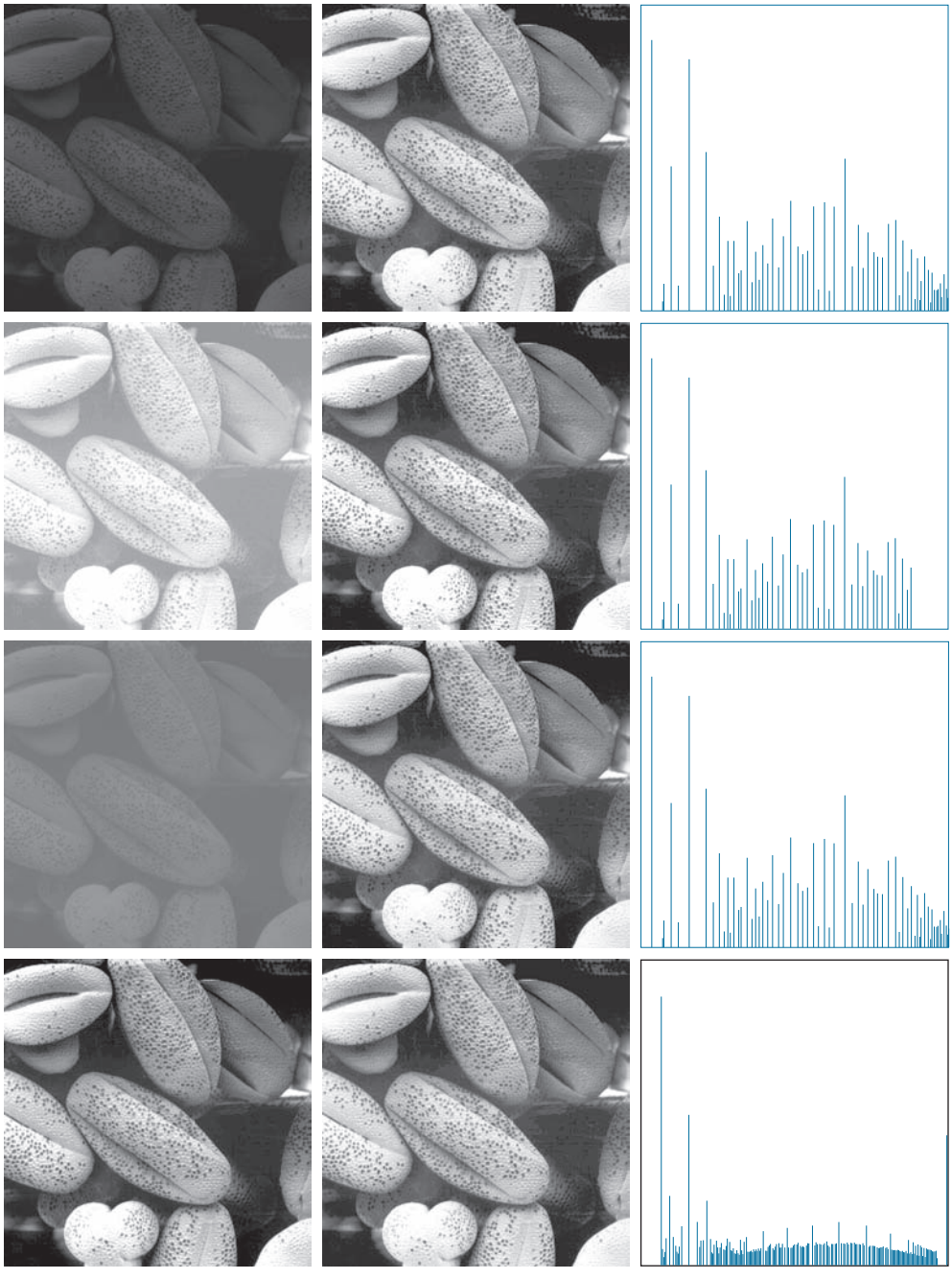


FIGURE 3.20 Left column: Images from Fig. 3.16. Center column: Corresponding histogram-equalized images. Right column: histograms of the images in the center column (compare with the histograms in Fig. 3.16).

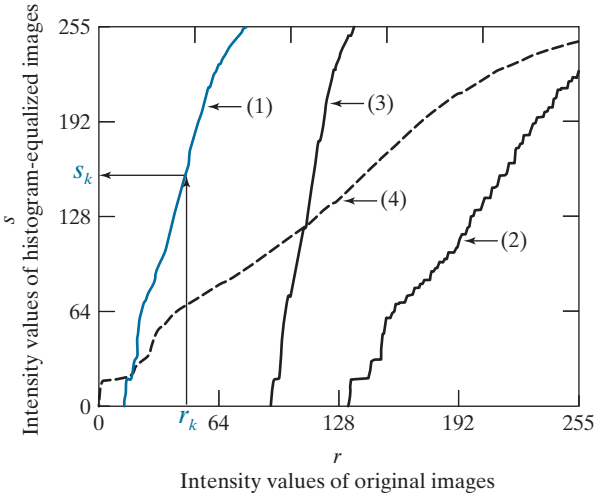


FIGURE 3.21 Transformation functions for histogram equalization. Transformations (1) through (4) were obtained using Eq. (3-15) and the histograms of the images on the left column of Fig. 3.20. Mapping of one intensity value r_k in image 1 to its corresponding value s_k is shown.

results from this technique are predictable and the method is simple to implement. However, there are applications in which histogram equalization is not suitable. In particular, it is useful sometimes to be able to specify the shape of the histogram that we wish the processed image to have. The method used to generate images that have a specified histogram is called *histogram matching* or *histogram specification*.

Consider for a moment continuous intensities r and z which, as before, we treat as random variables with PDFs $p_r(r)$ and $p_z(z)$, respectively. Here, r and z denote the intensity levels of the input and output (processed) images, respectively. We can estimate $p_r(r)$ from the given input image, and $p_z(z)$ is the *specified* PDF that we wish the output image to have.

Let s be a random variable with the property

$$s = T(r) = (L-1)\int_0^r p_r(w)\,dw \tag{3-17}$$

where w is dummy variable of integration. This is the same as Eq. (3-11), which we repeat here for convenience.

Define a function G on variable z with the property

$$G(z) = (L-1)\int_0^z p_z(v)\,dv = s \tag{3-18}$$

where v is a dummy variable of integration. It follows from the preceding two equations that $G(z) = s = T(r)$ and, therefore, that z must satisfy the condition

$$z = G^{-1}(s) = G^{-1}\left[T(r)\right] \tag{3-19}$$

The transformation function $T(r)$ can be obtained using Eq. (3-17) after $p_r(r)$ has been estimated using the input image. Similarly, function $G(z)$ can be obtained from Eq. (3-18) because $p_z(z)$ is given.

Equations (3-17) through (3-19) imply that an image whose intensity levels have a specified PDF can be obtained using the following procedure:

1. Obtain $p_r(r)$ from the input image to use in Eq. (3-17).
2. Use the specified PDF, $p_z(z)$, in Eq. (3-18) to obtain the function $G(z)$.
3. Compute the inverse transformation $z = G^{-1}(s)$; this is a mapping from s to z, the latter being the values that have the specified PDF.
4. Obtain the output image by first equalizing the input image using Eq. (3-17); the pixel values in this image are the s values. For each pixel with value s in the equalized image, perform the inverse mapping $z = G^{-1}(s)$ to obtain the corresponding pixel in the output image. When all pixels have been processed with this transformation, the PDF of the output image, $p_z(z)$, will be equal to the specified PDF.

Because s is related to r by $T(r)$, it is possible for the mapping that yields z from s to be expressed directly in terms of r. In general, however, finding analytical expressions for G^{-1} is not a trivial task. Fortunately, this is not a problem when working with discrete quantities, as you will see shortly.

As before, we have to convert the continuous result just derived into a discrete form. This means that we work with histograms instead of PDFs. As in histogram equalization, we lose in the conversion the ability to be able to guarantee a result that will have the exact specified histogram. Despite this, some very useful results can be obtained even with approximations.

The discrete formulation of Eq. (3-17) is the histogram equalization transformation in Eq. (3-15), which we repeat here for convenience:

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) \qquad k = 0, 1, 2, \ldots, L-1 \tag{3-20}$$

where the components of this equation are as before. Similarly, given a specific value of s_k, the discrete formulation of Eq. (3-18) involves computing the transformation function

$$G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i) \tag{3-21}$$

for a value of q so that

$$G(z_q) = s_k \tag{3-22}$$

where $p_z(z_i)$ is the ith value of the specified histogram. Finally, we obtain the desired value z_q from the inverse transformation:

$$z_q = G^{-1}(s_k) \tag{3-23}$$

When performed over all pixels, this is a mapping from the s values in the histogram-equalized image to the corresponding z values in the output image.

In practice, there is no need to compute the inverse of G. Because we deal with intensity levels that are integers, it is a simple matter to compute all the possible values of G using Eq. (3-21) for $q = 0, 1, 2, \ldots, L-1$. These values are rounded to their nearest integer values spanning the range $[0, L-1]$ and stored in a lookup table. Then, given a particular value of s_k, we look for the closest match in the table. For example, if the 27th entry in the table is the closest value to s_k, then $q = 26$ (recall that we start counting intensities at 0) and z_{26} is the best solution to Eq. (3-23). Thus, the given value s_k would map to z_{26}. Because the z's are integers in the range $[0, L-1]$, it follows that $z_0 = 0$, $z_{L-1} = L-1$, and, in general, $z_q = q$. Therefore, z_{26} would equal intensity value 26. We repeat this procedure to find the mapping from each value s_k to the value z_q that is its closest match in the table. These mappings are the solution to the histogram-specification problem.

Given an input image, a specified histogram, $p_z(z_i)$, $i = 0, 1, 2, \ldots, L-1$, and recalling that the s_k's are the values resulting from Eq. (3-20), we may summarize the procedure for discrete histogram specification as follows:

1. Compute the histogram, $p_r(r)$, of the input image, and use it in Eq. (3-20) to map the intensities in the input image to the intensities in the histogram-equalized image. Round the resulting values, s_k, to the integer range $[0, L-1]$.
2. Compute all values of function $G(z_q)$ using the Eq. (3-21) for $q = 0, 1, 2, \ldots, L-1$, where $p_z(z_i)$ are the values of the specified histogram. Round the values of G to integers in the range $[0, L-1]$. Store the rounded values of G in a lookup table.
3. For every value of s_k, $k = 0, 1, 2, \ldots, L-1$, use the stored values of G from Step 2 to find the corresponding value of z_q so that $G(z_q)$ is closest to s_k. Store these mappings from s to z. When more than one value of z_q gives the same match (i.e., the mapping is not unique), choose the smallest value by convention.
4. Form the histogram-specified image by mapping every equalized pixel with value s_k to the corresponding pixel with value z_q in the histogram-specified image, using the mappings found in Step 3.

As in the continuous case, the intermediate step of equalizing the input image is conceptual. It can be skipped by combining the two transformation functions, T and G^{-1}, as Example 3.7 below shows.

We mentioned at the beginning of the discussion on histogram equalization that, in addition to condition (b), inverse functions (G^{-1} in the present discussion) have to be strictly monotonic to satisfy condition (a′). In terms of Eq. (3-21), this means that none of the values $p_z(z_i)$ in the specified histogram can be zero (see Problem 3.9). When this condition is not satisfied, we use the "work-around" procedure in Step 3. The following example illustrates this numerically.

EXAMPLE 3.7: Illustration of the mechanics of histogram specification.

Consider the 64×64 hypothetical image from Example 3.5, whose histogram is repeated in Fig. 3.22(a). It is desired to transform this histogram so that it will have the values specified in the second column of Table 3.2. Figure 3.22(b) shows this histogram.

The first step is to obtain the histogram-equalized values, which we did in Example 3.5:

$$s_0 = 1;\ s_1 = 3;\ s_2 = 5;\ s_3 = 6;\ s_4 = 6;\ s_5 = 7;\ s_6 = 7;\ s_7 = 7$$

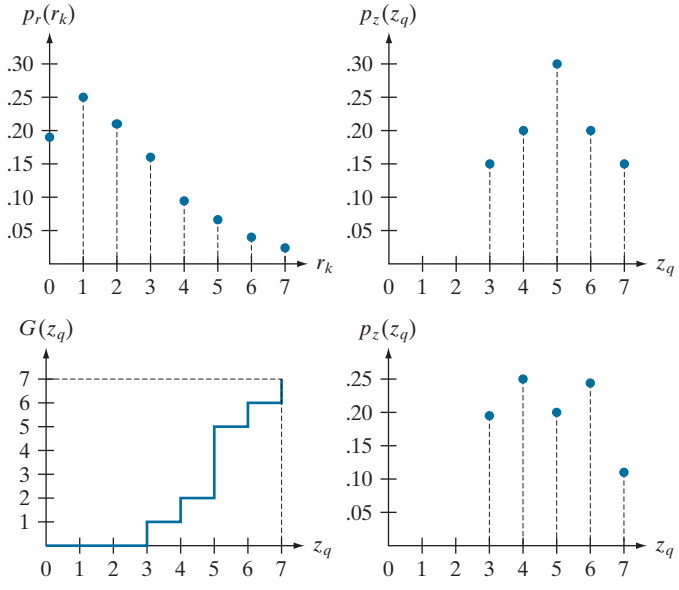
In the next step, we compute the values of $G(z_q)$ using the values of $p_z(z_q)$ from Table 3.2 in Eq. (3-21):

$$G(z_0) = 0.00 \quad G(z_2) = 0.00 \quad G(z_4) = 2.45 \quad G(z_6) = 5.95$$
$$G(z_1) = 0.00 \quad G(z_3) = 1.05 \quad G(z_5) = 4.55 \quad G(z_7) = 7.00$$

As in Example 3.5, these fractional values are rounded to integers in the range $[0, 7]$:

$$G(z_0) = 0.00 \rightarrow 0 \qquad G(z_4) = 2.45 \rightarrow 2$$
$$G(z_1) = 0.00 \rightarrow 0 \qquad G(z_5) = 4.55 \rightarrow 5$$
$$G(z_2) = 0.00 \rightarrow 0 \qquad G(z_6) = 5.95 \rightarrow 6$$
$$G(z_3) = 1.05 \rightarrow 1 \qquad G(z_7) = 7.00 \rightarrow 7$$

These results are summarized in Table 3.3. The transformation function, $G(z_q)$, is sketched in Fig. 3.23(c). Because its first three values are equal, G is not strictly monotonic, so condition (a′) is violated. Therefore, we use the approach outlined in Step 3 of the algorithm to handle this situation. According to this step, we find the smallest value of z_q so that the value $G(z_q)$ is the closest to s_k. We do this for every value of



a b
c d

FIGURE 3.22
(a) Histogram of a 3-bit image. (b) Specified histogram. (c) Transformation function obtained from the specified histogram. (d) Result of histogram specification. Compare the histograms in (b) and (d).

TABLE 3.2
Specified and actual histograms (the values in the third column are computed in Example 3.7).

z_q	Specified $p_z(z_q)$	Actual $p_z(z_q)$
$z_0 = 0$	0.00	0.00
$z_1 = 1$	0.00	0.00
$z_2 = 2$	0.00	0.00
$z_3 = 3$	0.15	0.19
$z_4 = 4$	0.20	0.25
$z_5 = 5$	0.30	0.21
$z_6 = 6$	0.20	0.24
$z_7 = 7$	0.15	0.11

s_k to create the required mappings from s to z. For example, $s_0 = 1$, and we see that $G(z_3) = 1$, which is a perfect match in this case, so we have the correspondence $s_0 \rightarrow z_3$. Every pixel whose value is 1 in the histogram equalized image would map to a pixel valued 3 in the histogram-specified image. Continuing in this manner, we arrive at the mappings in Table 3.4.

In the final step of the procedure, we use the mappings in Table 3.4 to map every pixel in the histogram equalized image into a corresponding pixel in the newly created histogram-specified image. The values of the resulting histogram are listed in the third column of Table 3.2, and the histogram is shown in Fig. 3.22(d). The values of $p_z(z_q)$ were obtained using the same procedure as in Example 3.5. For instance, we see in Table 3.4 that $s_k = 1$ maps to $z_q = 3$, and there are 790 pixels in the histogram-equalized image with a value of 1. Therefore, $p_z(z_3) = 790/4096 = 0.19$.

Although the final result in Fig. 3.22(d) does not match the specified histogram exactly, the general trend of moving the intensities toward the high end of the intensity scale definitely was achieved. As mentioned earlier, obtaining the histogram-equalized image as an intermediate step is useful for

TABLE 3.3
Rounded values of the transformation function $G(z_q)$.

z_q	$G(z_q)$
$z_0 = 0$	0
$z_1 = 1$	0
$z_2 = 2$	0
$z_3 = 3$	1
$z_4 = 4$	2
$z_5 = 5$	5
$z_6 = 6$	6
$z_7 = 7$	7

TABLE 3.4
Mapping of values s_k into corresponding values z_q.

s_k	→	z_q
1	→	3
3	→	4
5	→	5
6	→	6
7	→	7

explaining the procedure, but this is not necessary. Instead, we could list the mappings from the r's to the s's and from the s's to the z's in a three-column table. Then, we would use those mappings to map the original pixels directly into the pixels of the histogram-specified image.

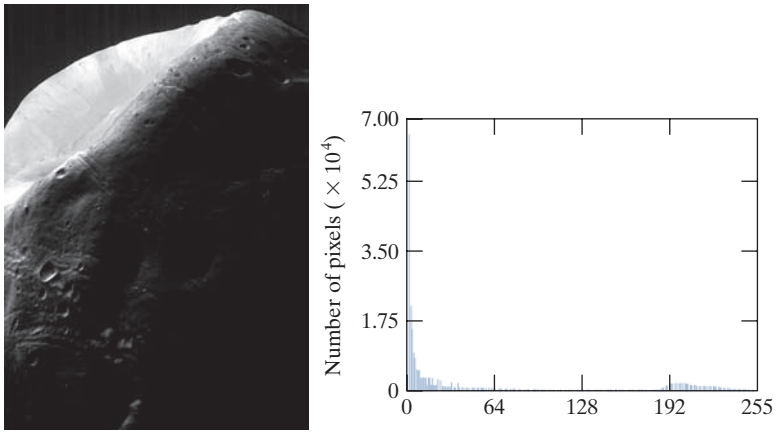
EXAMPLE 3.8: Comparison between histogram equalization and histogram specification.

Figure 3.23(a) shows an image of the Mars moon, Phobos, taken by NASA's Mars Global Surveyor. Figure 3.23(b) shows the histogram of Fig. 3.23(a). The image is dominated by large, dark areas, resulting in a histogram characterized by a large concentration of pixels in the dark end of the gray scale. At first glance, one might conclude that histogram equalization would be a good approach to enhance this image, so that details in the dark areas become more visible. It is demonstrated in the following discussion that this is not so.

Figure 3.24(a) shows the histogram equalization transformation [Eq. (3-20)] obtained using the histogram in Fig. 3.23(b). The most relevant characteristic of this transformation function is how fast it rises from intensity level 0 to a level near 190. This is caused by the large concentration of pixels in the input histogram having levels near 0. When this transformation is applied to the levels of the input image to obtain a histogram-equalized result, the net effect is to map a very narrow interval of dark pixels into the

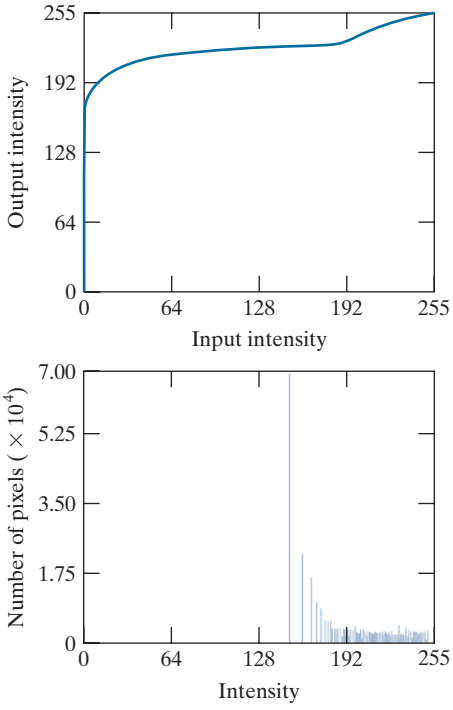
a b

FIGURE 3.23
(a) An image, and (b) its histogram.



a b
c

FIGURE 3.24
(a) Histogram equalization transformation obtained using the histogram in Fig. 3.23(b). (b) Histogram equalized image. (c) Histogram of equalized image.



upper end of the gray scale of the output image. Because numerous pixels in the input image have levels precisely in this interval, we would expect the result to be an image with a light, washed-out appearance. As Fig. 3.24(b) shows, this is indeed the case. The histogram of this image is shown in Fig. 3.24(c). Note how all the intensity levels are biased toward the upper one-half of the gray scale.

Because the problem with the transformation function in Fig. 3.24(a) was caused by a large concentration of pixels in the original image with levels near 0, a reasonable approach is to modify the histogram of that image so that it does not have this property. Figure 3.25(a) shows a manually specified function that preserves the general shape of the original histogram, but has a smoother transition of levels in the dark region of the gray scale. Sampling this function into 256 equally spaced discrete values produced the desired specified histogram. The transformation function, $G(z_q)$, obtained from this histogram using Eq. (3-21) is labeled transformation (1) in Fig. 3.25(b). Similarly, the inverse transformation $G^{-1}(s_k)$, from Eq. (3-23) (obtained using the step-by-step procedure discussed earlier) is labeled transformation (2) in Fig. 3.25(b). The enhanced image in Fig. 3.25(c) was obtained by applying transformation (2) to the pixels of the histogram-equalized image in Fig. 3.24(b). The improvement of the histogram-specified image over the result obtained by histogram equalization is evident by comparing these two images. It is of interest to note that a rather modest change in the original histogram was all that was required to obtain a significant improvement in appearance. Figure 3.25(d) shows the histogram of Fig. 3.25(c). The most distinguishing feature of this histogram is how its low end has shifted right toward the lighter region of the gray scale (but not excessively so), as desired.

a c
b
d

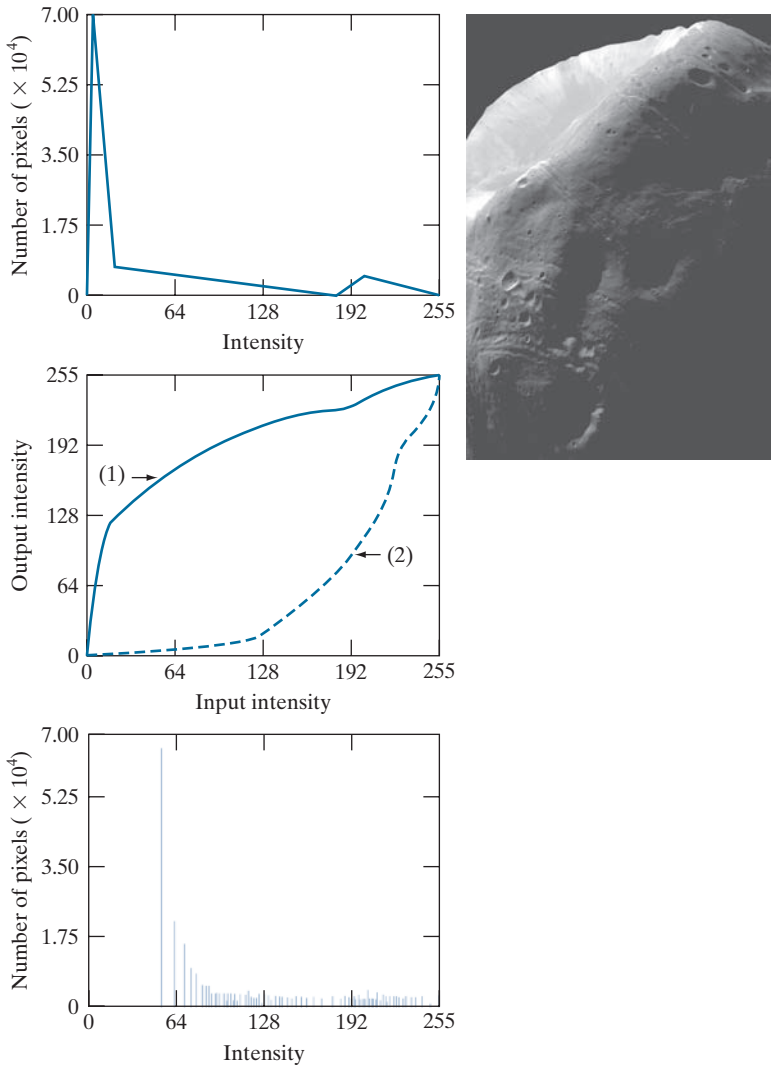


FIGURE 3.25
Histogram specification. (a) Specified histogram. (b) Transformation $G(z_q)$, labeled (1), and $G^{-1}(s_k)$, labeled (2). (c) Result of histogram specification. (d) Histogram of image (c).

LOCAL HISTOGRAM PROCESSING

The histogram processing methods discussed thus far are *global*, in the sense that pixels are modified by a transformation function based on the intensity distribution of an entire image. This global approach is suitable for overall enhancement, but generally fails when the objective is to enhance details over small areas in an image. This is because the number of pixels in small areas have negligible influence on the computation of global transformations. The solution is to devise transformation functions based on the intensity distribution of pixel neighborhoods.

The histogram processing techniques previously described can be adapted to local enhancement. The procedure is to define a neighborhood and move its center from

pixel to pixel in a horizontal or vertical direction. At each location, the histogram of the points in the neighborhood is computed, and either a histogram equalization or histogram specification transformation function is obtained. This function is used to map the intensity of the pixel centered in the neighborhood. The center of the neighborhood is then moved to an adjacent pixel location and the procedure is repeated. Because only one row or column of the neighborhood changes in a one-pixel translation of the neighborhood, updating the histogram obtained in the previous location with the new data introduced at each motion step is possible (see Problem 3.14). This approach has obvious advantages over repeatedly computing the histogram of all pixels in the neighborhood region each time the region is moved one pixel location. Another approach used sometimes to reduce computation is to utilize nonoverlapping regions, but this method usually produces an undesirable "blocky" effect.

EXAMPLE 3.9: Local histogram equalization.

Figure 3.26(a) is an 8-bit, 512×512 image consisting of five black squares on a light gray background. The image is slightly noisy, but the noise is imperceptible. There are objects embedded in the dark squares, but they are invisible for all practical purposes. Figure 3.26(b) is the result of global histogram equalization. As is often the case with histogram equalization of smooth, noisy regions, this image shows significant enhancement of the noise. However, other than the noise, Fig. 3.26(b) does not reveal any new significant details from the original. Figure 3.26(c) was obtained using local histogram equalization of Fig. 3.26(a) with a neighborhood of size 3×3. Here, we see significant detail within all the dark squares. The intensity values of these objects are too close to the intensity of the dark squares, and their sizes are too small, to influence global histogram equalization significantly enough to show this level of intensity detail.

USING HISTOGRAM STATISTICS FOR IMAGE ENHANCEMENT

Statistics obtained directly from an image histogram can be used for image enhancement. Let r denote a discrete random variable representing intensity values in the range $[0, L-1]$, and let $p(r_i)$ denote the normalized histogram component corresponding to intensity value r_i. As indicated earlier, we may view $p(r_i)$ as an estimate of the probability that intensity r_i occurs in the image from which the histogram was obtained.

a b c

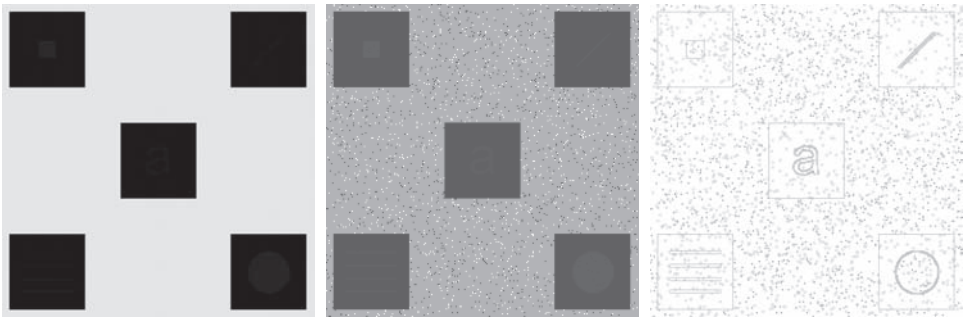


FIGURE 3.26 (a) Original image. (b) Result of global histogram equalization. (c) Result of local histogram equalization.

See the tutorials section in the book website for a review of probability.

For an image with intensity levels in the range $[0, L-1]$, the nth moment of r about its mean, m, is defined as

$$\mu_n = \sum_{i=0}^{L-1} (r_i - m)^n p(r_i) \tag{3-24}$$

We follow convention in using m for the mean value. Do not confuse it with our use of the same symbol to denote the number of rows in an $m \times n$ neighborhood.

where m is given by

$$m = \sum_{i=0}^{L-1} r_i p(r_i) \tag{3-25}$$

The mean is a measure of average intensity and the variance (or standard deviation, σ), given by

$$\sigma^2 = \mu_2 = \sum_{i=0}^{L-1} (r_i - m)^2 p(r_i) \tag{3-26}$$

is a measure of image contrast.

We consider two uses of the mean and variance for enhancement purposes. The *global* mean and variance [Eqs. (3-25) and (3-26)] are computed over an entire image and are useful for gross adjustments in overall intensity and contrast. A more powerful use of these parameters is in local enhancement, where the *local* mean and variance are used as the basis for making changes that depend on image characteristics in a neighborhood about each pixel in an image.

Let (x, y) denote the coordinates of any pixel in a given image, and let S_{xy} denote a neighborhood of specified size, centered on (x, y). The mean value of the pixels in this neighborhood is given by the expression

$$m_{S_{xy}} = \sum_{i=0}^{L-1} r_i p_{S_{xy}}(r_i) \tag{3-27}$$

where $p_{S_{xy}}$ is the histogram of the pixels in region S_{xy}. This histogram has L bins, corresponding to the L possible intensity values in the input image. However, many of the bins will have 0 counts, depending on the size of S_{xy}. For example, if the neighborhood is of size 3×3 and $L = 256$, only between 1 and 9 of the 256 bins of the histogram of the neighborhood will be nonzero (the maximum number of possible *different* intensities in a 3×3 region is 9, and the minimum is 1). These non-zero values will correspond to the number of different intensities in S_{xy}.

The variance of the pixels in the neighborhood is similarly given by

$$\sigma^2_{S_{xy}} = \sum_{i=0}^{L-1} (r_i - m_{S_{xy}})^2 p_{S_{xy}}(r_i) \tag{3-28}$$

As before, the local mean is a measure of average intensity in neighborhood S_{xy}, and the local variance (or standard deviation) is a measure of intensity contrast in that neighborhood.

As the following example illustrates, an important aspect of image processing using the local mean and variance is the flexibility these parameters afford in developing simple, yet powerful enhancement rules based on statistical measures that have a close, predictable correspondence with image appearance.

EXAMPLE 3.10: Local enhancement using histogram statistics.

Figure 3.27(a) is the same image as Fig. 3.26(a), which we enhanced using local histogram equalization. As noted before, the dark squares contain embedded symbols that are almost invisible. As before, we want to enhance the image to bring out these hidden features.

We can use the concepts presented in this section to formulate an approach for enhancing low-contrast details embedded in a background of similar intensity. The problem at hand is to enhance the low-contrast detail in the dark areas of the image, while leaving the light background unchanged.

A method used to determine whether an area is relatively light or dark at a point (x, y) is to compare the average local intensity, $m_{S_{xy}}$, to the average image intensity (the global mean), denoted by m_G. We obtain m_G using Eq. (3-25) with the histogram of the entire image. Thus, we have the first element of our enhancement scheme: We will consider the pixel at (x, y) as a candidate for processing if $k_0 m_G \leq m_{S_{xy}} \leq k_1 m_G$, where k_0 and k_1 are nonnegative constants and $k_0 < k_1$. For example, if our focus is on areas that are darker than one-quarter of the mean intensity, we would choose $k_0 = 0$ and $k_1 = 0.25$.

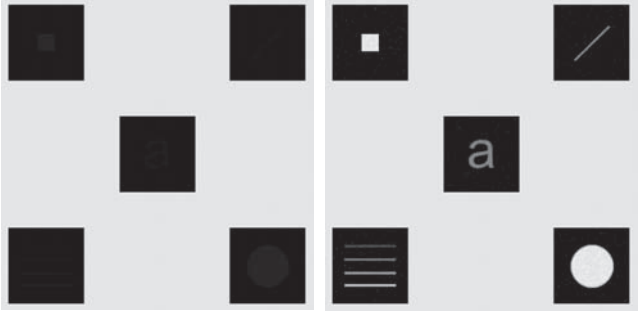
Because we are interested in enhancing areas that have low contrast, we also need a measure to determine whether the contrast of an area makes it a candidate for enhancement. We consider the pixel at (x, y) as a candidate if $k_2 \sigma_G \leq \sigma_{S_{xy}} \leq k_3 \sigma_G$, where σ_G is the global standard deviation obtained with Eq. (3-26) using the histogram of the entire image, and k_2 and k_3 are nonnegative constants, with $k_2 < k_3$. For example, to enhance a dark area of low contrast, we might choose $k_2 = 0$ and $k_3 = 0.1$. A pixel that meets all the preceding conditions for local enhancement is processed by multiplying it by a specified constant, C, to increase (or decrease) the value of its intensity level relative to the rest of the image. Pixels that do not meet the enhancement conditions are not changed.

We summarize the preceding approach as follows. Let $f(x, y)$ denote the value of an image at any image coordinates (x, y), and let $g(x, y)$ be the corresponding value in the enhanced image at those coordinates. Then,

$$g(x,y) = \begin{cases} C f(x,y) & \text{if } k_0 m_G \leq m_{S_{xy}} \leq k_1 m_G \text{ AND } k_2 \sigma_G \leq \sigma_{S_{xy}} \leq k_3 \sigma_G \\ f(x,y) & \text{otherwise} \end{cases} \tag{3-29}$$

a b

FIGURE 3.27 (a) Original image. (b) Result of local enhancement based on local histogram statistics. Compare (b) with Fig. 3.26(c).



for $x = 0, 1, 2, \ldots, M-1$ and $y = 0, 1, 2, \ldots, N-1$, where, as indicated above, C, k_0, k_1, k_2, and k_3 are specified constants, m_G is the global mean of the input image, and σ_G is its standard deviation. Parameters $m_{S_{xy}}$ and $\sigma_{S_{xy}}$ are the local mean and standard deviation, respectively, which change for every location (x, y). As usual, M and N are the number of rows and columns in the input image.

Factors such as the values of the global mean and variance relative to values in the areas to be enhanced play a key role in selecting the parameters in Eq. (3-29), as does the range of differences between the intensities of the areas to be enhanced and their background. In the case of Fig. 3.27(a), $m_G = 161$, $\sigma_G = 103$, the maximum intensity values of the image and areas to be enhanced are 228 and 10, respectively, and the minimum values are 0 in both cases.

We would like for the maximum value of the enhanced features to be the same as the maximum value of the image, so we select $C = 22.8$. The areas to be enhanced are quite dark relative to the rest of the image, and they occupy less than a third of the image area; thus, we expect the mean intensity in the dark areas to be much less than the global mean. Based on this, we let $k_0 = 0$ and $k_1 = 0.1$. Because the areas to be enhanced are of very low contrast, we let $k_2 = 0$. For the upper limit of acceptable values of standard deviation we set $k_3 = 0.1$, which gives us one-tenth of the global standard deviation. Figure 3.27(b) is the result of using Eq. (3-29) with these parameters. By comparing this figure with Fig. 3.26(c), we see that the method based on local statistics detected the same hidden features as local histogram equalization. But the present approach extracted significantly more detail. For example, we see that all the objects are solid, but only the boundaries were detected by local histogram equalization. In addition, note that the intensities of the objects are not the same, with the objects in the top-left and bottom-right being brighter than the others. Also, the horizontal rectangles in the lower left square evidently are of different intensities. Finally, note that the background in both the image and dark squares in Fig. 3.27(b) is nearly the same as in the original image; by comparison, the same regions in Fig. 3.26(c) exhibit more visible noise and have lost their gray-level content. Thus, the additional complexity required to use local statistics yielded results in this case that are superior to local histogram equalization.

3.4 FUNDAMENTALS OF SPATIAL FILTERING

In this section, we discuss the use of spatial filters for image processing. Spatial filtering is used in a broad spectrum of image processing applications, so a solid understanding of filtering principles is important. As mentioned at the beginning of this chapter, the filtering examples in this section deal mostly with image enhancement. Other applications of spatial filtering are discussed in later chapters.

The name *filter* is borrowed from frequency domain processing (the topic of Chapter 4) where "filtering" refers to passing, modifying, or rejecting specified frequency components of an image. For example, a filter that passes low frequencies is called a *lowpass filter*. The net effect produced by a lowpass filter is to smooth an image by blurring it. We can accomplish similar smoothing directly on the image itself by using *spatial filters*.

Spatial filtering modifies an image by replacing the value of each pixel by a function of the values of the pixel and its neighbors. If the operation performed on the image pixels is linear, then the filter is called a *linear spatial filter*. Otherwise, the filter is a *nonlinear spatial filter*. We will focus attention first on linear filters and then introduce some basic nonlinear filters. Section 5.3 contains a more comprehensive list of nonlinear filters and their application.

See Section 2.6 regarding linearity.

THE MECHANICS OF LINEAR SPATIAL FILTERING

A linear spatial filter performs a sum-of-products operation between an image f and a *filter kernel*, w. The kernel is an array whose size defines the neighborhood of operation, and whose coefficients determine the nature of the filter. Other terms used to refer to a spatial filter kernel are *mask*, *template*, and *window*. We use the term *filter kernel* or simply *kernel*.

Figure 3.28 illustrates the mechanics of linear spatial filtering using a 3×3 kernel. At any point (x, y) in the image, the response, $g(x, y)$, of the filter is the sum of products of the kernel coefficients and the image pixels encompassed by the kernel:

$$\begin{aligned} g(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \ldots \\ + w(0,0)f(x,y) + \ldots + w(1,1)f(x+1,y+1) \end{aligned} \tag{3-30}$$

As coordinates x and y are varied, the center of the kernel moves from pixel to pixel, generating the filtered image, g, in the process.[†]

Observe that the center coefficient of the kernel, $w(0,0)$, aligns with the pixel at location (x, y). For a kernel of size $m \times n$, we assume that $m = 2a + 1$ and $n = 2b + 1$, where a and b are nonnegative integers. This means that our focus is on kernels of odd size in both coordinate directions. In general, linear spatial filtering of an image of size $M \times N$ with a kernel of size $m \times n$ is given by the expression

It certainly is possible to work with kernels of even size, or mixed even and odd sizes. However, working with odd sizes simplifies indexing and is also more intuitive because the kernels have centers falling on integer values, and they are spatially symmetric.

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{3-31}$$

where x and y are varied so that the center (origin) of the kernel visits every pixel in f once. For a fixed value of (x, y), Eq. (3-31) implements the *sum of products* of the form shown in Eq. (3-30), but for a kernel of arbitrary odd size. As you will learn in the following section, this equation is a central tool in linear filtering.

SPATIAL CORRELATION AND CONVOLUTION

Spatial correlation is illustrated graphically in Fig. 3.28, and it is described mathematically by Eq. (3-31). Correlation consists of moving the center of a kernel over an image, and computing the sum of products at each location. The mechanics of *spatial convolution* are the same, except that the correlation kernel is rotated by 180°. Thus, when the values of a kernel are symmetric about its center, correlation and convolution yield the same result. The reason for rotating the kernel will become clear in the following discussion. The best way to explain the differences between the two concepts is by example.

We begin with a 1-D illustration, in which case Eq. (3-31) becomes

$$g(x) = \sum_{s=-a}^{a} w(s) f(x+s) \tag{3-32}$$

[†] A filtered pixel value typically is assigned to a corresponding location in a new image created to hold the results of filtering. It is seldom the case that filtered pixels replace the values of the corresponding location in the original image, as this would change the content of the image while filtering is being performed.

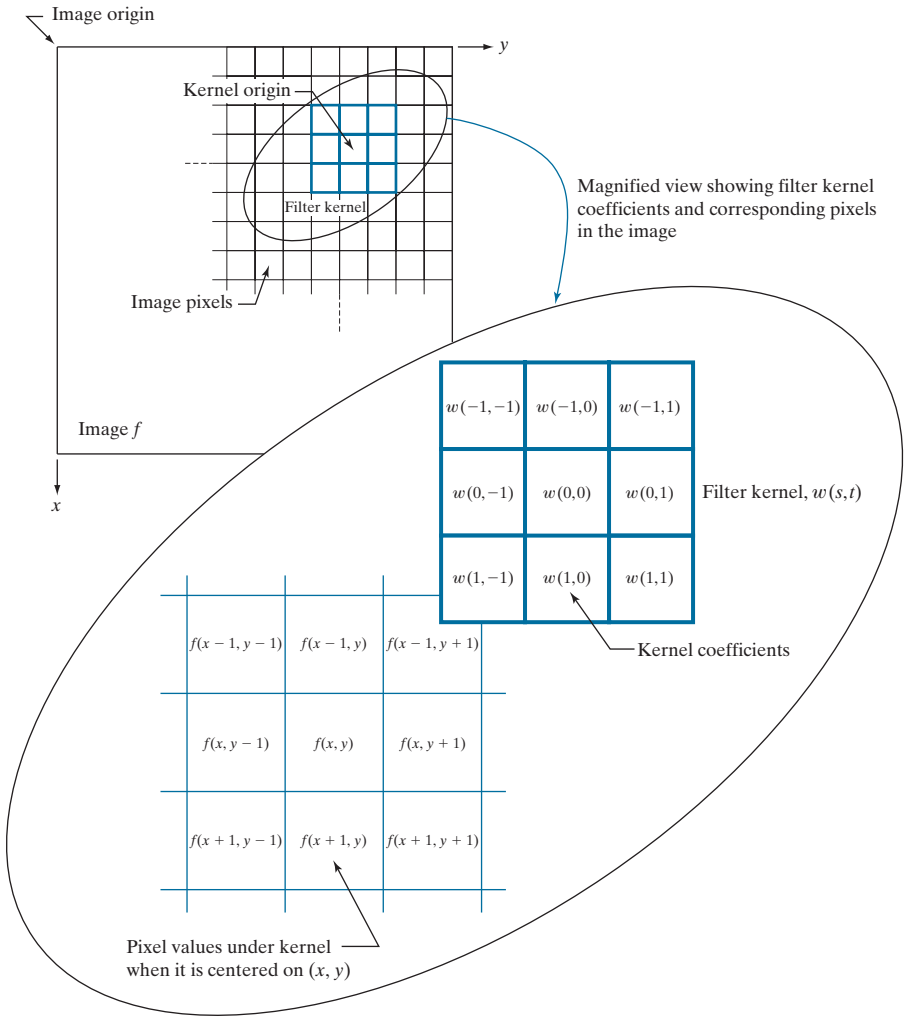


FIGURE 3.28 The mechanics of linear spatial filtering using a 3×3 kernel. The pixels are shown as squares to simplify the graphics. Note that the origin of the image is at the top left, but the origin of the kernel is at its center. Placing the origin at the center of spatially symmetric kernels simplifies writing expressions for linear filtering.

Figure 3.29(a) shows a 1-D function, f, and a kernel, w. The kernel is of size 1×5, so $a = 2$ and $b = 0$ in this case. Figure 3.29(b) shows the starting position used to perform correlation, in which w is positioned so that its center coefficient is coincident with the origin of f.

The first thing we notice is that part of w lies outside f, so the summation is undefined in that area. A solution to this problem is to *pad* function f with enough 0's on either side. In general, if the kernel is of size $1 \times m$, we need $(m-1)/2$ zeros on either side of f in order to handle the beginning and ending configurations of w with respect to f. Figure 3.29(c) shows a properly padded function. In this starting configuration, all coefficients of the kernel overlap valid values.

Zero padding is not the only padding option, as we will discuss in detail later in this chapter.

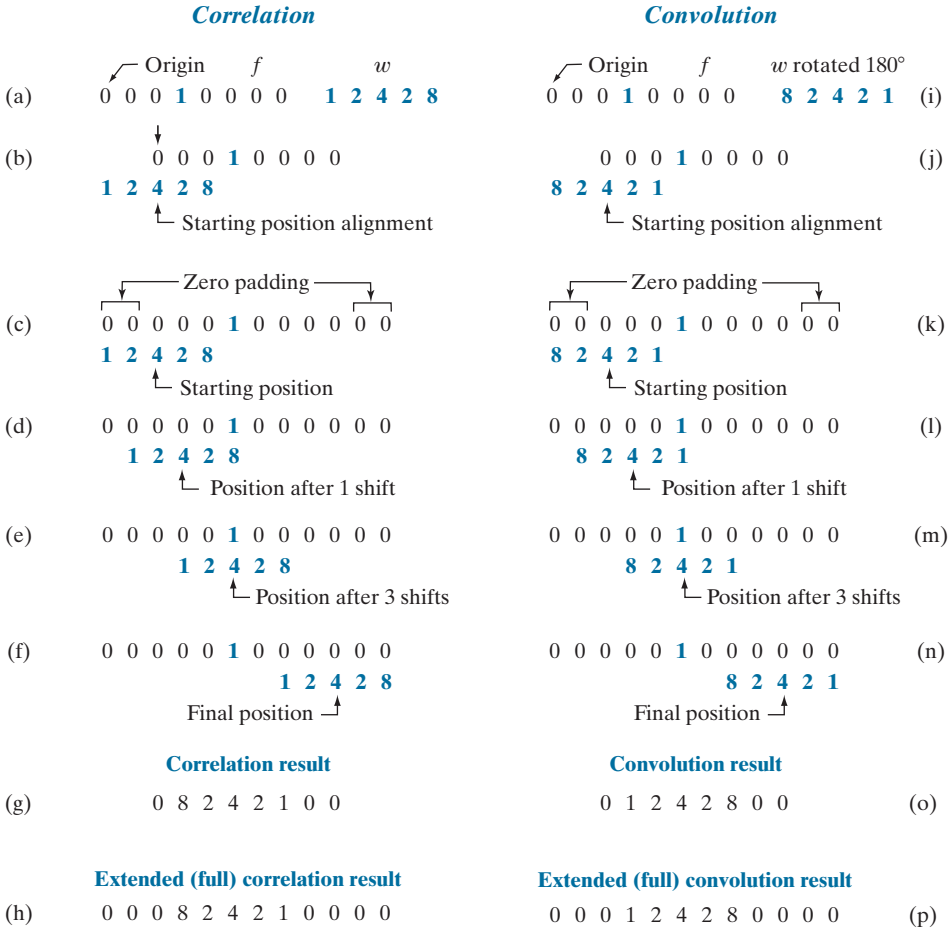


FIGURE 3.29 Illustration of 1-D correlation and convolution of a kernel, w, with a function f consisting of a discrete unit impulse. Note that correlation and convolution are functions of the variable x, which acts to *displace* one function with respect to the other. For the extended correlation and convolution results, the starting configuration places the rightmost element of the kernel to be coincident with the origin of f. Additional padding must be used.

The first correlation value is the sum of products in this initial position, computed using Eq. (3-32) with $x = 0$:

$$g(0) = \sum_{s=-2}^{2} w(s)f(s+0) = 0$$

This value is in the leftmost location of the correlation result in Fig. 3.29(g).

To obtain the second value of correlation, we shift the relative positions of w and f one pixel location to the right [i.e., we let $x = 1$ in Eq. (3-32)] and compute the sum of products again. The result is $g(1) = 8$, as shown in the leftmost, nonzero location in Fig. 3.29(g). When $x = 2$, we obtain $g(2) = 2$. When $x = 3$, we get $g(3) = 4$ [see Fig. 3.29(e)]. Proceeding in this manner by varying x one shift at a time, we "build" the correlation result in Fig. 3.29(g). Note that it took 8 values of x (i.e., $x = 0, 1, 2, \ldots, 7$) to fully shift w past f so the *center* coefficient in w visited *every* pixel in f. Sometimes, it is useful to have every element of w visit every pixel in f. For this, we have to start

with the rightmost element of w coincident with the origin of f, and end with the leftmost element of w being coincident the last element of f (additional padding would be required). Figure Fig. 3.29(h) shows the result of this *extended*, or *full*, correlation. As Fig. 3.29(g) shows, we can obtain the "standard" correlation by cropping the full correlation in Fig. 3.29(h).

There are two important points to note from the preceding discussion. First, correlation is a function of *displacement* of the filter kernel relative to the image. In other words, the first value of correlation corresponds to zero displacement of the kernel, the second corresponds to one unit displacement, and so on.[†] The second thing to notice is that correlating a kernel w with a function that contains all 0's and a single 1 yields a *copy* of w, but *rotated* by 180°. A function that contains a single 1 with the rest being 0's is called a *discrete unit impulse*. Correlating a kernel with a discrete unit impulse yields a *rotated* version of the kernel at the location of the impulse.

Rotating a 1-D kernel by 180° is equivalent to flipping the kernel about its axis.

The right side of Fig. 3.29 shows the sequence of steps for performing convolution (we will give the equation for convolution shortly). The only difference here is that the kernel is *pre-rotated* by 180° prior to performing the shifting/sum of products operations. As the convolution in Fig. 3.29(o) shows, the result of pre-rotating the kernel is that now we have an *exact* copy of the kernel at the location of the unit impulse. In fact, a foundation of linear system theory is that convolving a function with an impulse yields a copy of the function at the location of the impulse. We will use this property extensively in Chapter 4.

The 1-D concepts just discussed extend easily to images, as Fig. 3.30 shows. For a kernel of size $m \times n$, we pad the image with a minimum of $(m-1)/2$ rows of 0's at the top and bottom and $(n-1)/2$ columns of 0's on the left and right. In this case, m and n are equal to 3, so we pad f with one row of 0's above and below and one column of 0's to the left and right, as Fig. 3.30(b) shows. Figure 3.30(c) shows the initial position of the kernel for performing correlation, and Fig. 3.30(d) shows the final result after the center of w visits every pixel in f, computing a sum of products at each location. As before, the result is a copy of the kernel, rotated by 180°. We will discuss the extended correlation result shortly.

In 2-D, rotation by 180° is equivalent to flipping the kernel about one axis and then the other.

For convolution, we pre-rotate the kernel as before and repeat the sliding sum of products just explained. Figures 3.30(f) through (h) show the result. You see again that convolution of a function with an impulse copies the function to the location of the impulse. As noted earlier, correlation and convolution yield the same result if the kernel values are symmetric about the center.

The concept of an impulse is fundamental in linear system theory, and is used in numerous places throughout the book. A *discrete impulse of strength* (*amplitude*) A located at coordinates (x_0, y_0) is defined as

$$\delta(x - x_0, y - y_0) = \begin{cases} A & \text{if } x = x_0 \text{ and } y = y_0 \\ 0 & \text{otherwise} \end{cases} \tag{3-33}$$

[†] In reality, we are shifting f to the left of w every time we increment x in Eq. (3-32). However, it is more intuitive to think of the smaller kernel moving right over the larger array f. The motion of the two is relative, so either way of looking at the motion is acceptable. The reason we increment f and not w is that indexing the equations for correlation and convolution is much easier (and clearer) this way, especially when working with 2-D arrays.

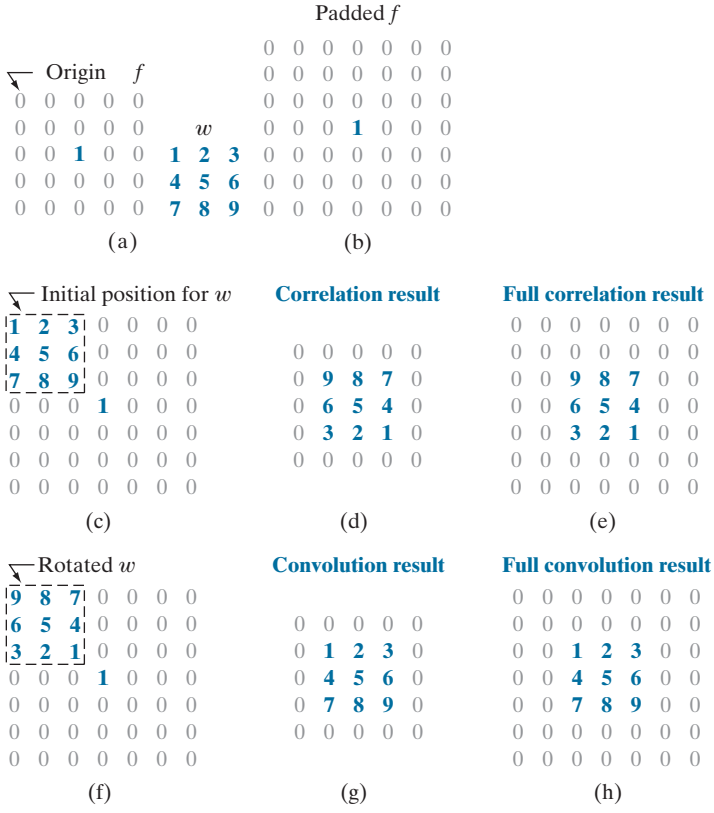


FIGURE 3.30 Correlation (middle row) and convolution (last row) of a 2-D kernel with an image consisting of a discrete unit impulse. The 0's are shown in gray to simplify visual analysis. Note that correlation and convolution are functions of x and y. As these variable change, they *displace* one function with respect to the other. See the discussion of Eqs. (3-36) and (3-37) regarding full correlation and convolution.

Recall that $A = 1$ for a unit impulse.

For example, the unit impulse in Fig. 3.29(a) is given by $\delta(x-3)$ in the 1-D version of the preceding equation. Similarly, the impulse in Fig. 3.30(a) is given by $\delta(x-2, y-2)$ [remember, the origin is at $(0,0)$].

Summarizing the preceding discussion in equation form, the correlation of a kernel w of size $m \times n$ with an image $f(x,y)$, denoted as $(w \star f)(x,y)$, is given by Eq. (3-31), which we repeat here for convenience:

$$(w \star f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{3-34}$$

Because our kernels do not depend on (x,y), we will sometimes make this fact explicit by writing the left side of the preceding equation as $w \star f(x,y)$. Equation (3-34) is evaluated for all values of the displacement variables x and y so that the center point of w visits every pixel in f,[†] where we assume that f has been padded appropriately.

[†] As we mentioned earlier, the *minimum* number of required padding elements for a 2-D correlation is $(m-1)/2$ rows above and below f, and $(n-1)/2$ columns on the left and right. With this padding, and assuming that f is of size $M \times N$, the values of x and y required to obtain a complete correlation are $x = 0, 1, 2, \ldots, M-1$ and $y = 0, 1, 2, \ldots, N-1$. This assumes that the starting configuration is such that the *center* of the kernel coincides with the *origin* of the image, which we have defined to be at the top, left (see Fig. 2.19).

As explained earlier, $a = (m-1)/2$, $b = (n-1)/2$, and we assume that m and n are odd integers.

In a similar manner, the *convolution* of a kernel w of size $m \times n$ with an image $f(x, y)$, denoted by $(w \star f)(x, y)$, is defined as

$$(w \star f)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t) \tag{3-35}$$

where the minus signs align the coordinates of f and w when one of the functions is rotated by 180° (see Problem 3.17). This equation implements the sum of products process to which we refer throughout the book as *linear spatial filtering*. That is, linear spatial filtering and spatial convolution are synonymous.

Because convolution is commutative (see Table 3.5), it is immaterial whether w or f is rotated, but rotation of the kernel is used by convention. Our kernels do not depend on (x, y), a fact that we sometimes make explicit by writing the left side of Eq. (3-35) as $w \star f(x, y)$. When the meaning is clear, we let the dependence of the previous two equations on x and y be implied, and use the simplified notation $w \star f$ and $w \star f$. As with correlation, Eq. (3-35) is evaluated for all values of the displacement variables x and y so that the center of w visits every pixel in f, which we assume has been padded. The values of x and y needed to obtain a full convolution are $x = 0, 1, 2, \ldots, M-1$ and $y = 0, 1, 2, \ldots, N-1$. The size of the result is $M \times N$.

We can define correlation and convolution so that *every* element of w (instead of just its center) visits *every* pixel in f. This requires that the starting configuration be such that the right, lower corner of the kernel coincides with the origin of the image. Similarly, the ending configuration will be with the top left corner of the kernel coinciding with the lower right corner of the image. If the kernel and image are of sizes $m \times n$ and $M \times N$, respectively, the padding would have to increase to $(m-1)$ padding elements above and below the image, and $(n-1)$ elements to the left and right. Under these conditions, the size of the resulting full correlation or convolution array will be of size $S_v \times S_h$, where (see Figs. 3.30(e) and (h), and Problem 3.19),

$$S_v = m + M - 1 \tag{3-36}$$

and

$$S_h = n + N - 1 \tag{3-37}$$

Often, spatial filtering algorithms are based on correlation and thus implement Eq. (3-34) instead. To use the algorithm for correlation, we input w into it; for convolution, we input w rotated by 180°. The opposite is true for an algorithm that implements Eq. (3-35). Thus, either Eq. (3-34) or Eq. (3-35) can be made to perform the function of the other by rotating the filter kernel. Keep in mind, however, that the *order* of the functions input into a correlation algorithm *does* make a difference, because correlation is neither commutative nor associative (see Table 3.5).

TABLE 3.5
Some fundamental properties of convolution and correlation. A dash means that the property does not hold.

Property	Convolution	Correlation
Commutative	$f \star g = g \star f$	—
Associative	$f \star (g \star h) = (f \star g) \star h$	—
Distributive	$f \star (g + h) = (f \star g) + (f \star h)$	$f \star\!\!\star (g + h) = (f \star\!\!\star g) + (f \star\!\!\star h)$

Because the values of these kernels are symmetric about the center, no rotation is required before convolution.

Figure 3.31 shows two kernels used for smoothing the intensities of an image. To filter an image using one of these kernels, we perform a convolution of the kernel with the image in the manner just described. When talking about filtering and kernels, you are likely to encounter the terms *convolution filter*, *convolution mask,* or *convolution kernel* to denote filter kernels of the type we have been discussing. Typically, these terms are used in the literature to denote a spatial filter kernel, and not to imply necessarily that the kernel is used for convolution. Similarly, "convolving a kernel with an image" often is used to denote the sliding, sum-of-products process we just explained, and does not necessarily differentiate between correlation and convolution. Rather, it is used generically to denote either of the two operations. This imprecise terminology is a frequent source of confusion. In this book, when we use the term *linear spatial filtering*, we mean *convolving a kernel with an image*.

Sometimes an image is filtered (i.e., convolved) sequentially, in stages, using a different kernel in each stage. For example, suppose than an image f is filtered with a kernel w_1, the result filtered with kernel w_2, that result filtered with a third kernel, and so on, for Q stages. Because of the commutative property of convolution, this multistage filtering can be done in a single filtering operation, $w \star f$, where

We could not write a similar equation for correlation because it is not commutative.

$$w = w_1 \star w_2 \star w_3 \star \cdots \star w_Q \tag{3-38}$$

The size of w is obtained from the sizes of the individual kernels by successive applications of Eqs. (3-36) and (3-37). If all the individual kernels are of size $m \times n$, it follows from these equations that w will be of size $W_v \times W_h$, where

$$W_v = Q \times (m - 1) + m \tag{3-39}$$

and

$$W_h = Q \times (n - 1) + n \tag{3-40}$$

These equations assume that every value of a kernel visits every value of the array resulting from the convolution in the previous step. That is, the initial and ending configurations, are as described in connection with Eqs. (3-36) and (3-37).

a b

FIGURE 3.31
Examples of smoothing kernels: (a) is a *box* kernel; (b) is a *Gaussian* kernel.

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{4.8976} \times \begin{bmatrix} 0.3679 & 0.6065 & 0.3679 \\ 0.6065 & 1.0000 & 0.6065 \\ 0.3679 & 0.6065 & 0.3679 \end{bmatrix}$$

SEPARABLE FILTER KERNELS

As noted in Section 2.6, a 2-D function $G(x, y)$ is said to be *separable* if it can be written as the product of two 1-D functions, $G_1(x)$ and $G_2(x)$; that is, $G(x, y) = G_1(x)G_2(y)$. A spatial filter kernel is a matrix, and a separable kernel is a matrix that can be expressed as the outer product of two vectors. For example, the 2×3 kernel

To be strictly consistent in notation, we should use uppercase, bold symbols for kernels when we refer to them as matrices. However, kernels are mostly treated in the book as 2-D functions, which we denote in italics. To avoid confusion, we continue to use italics for kernels in this short section, with the understanding that the two notations are intended to be equivalent in this case.

$$w = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

is separable because it can be expressed as the outer product of the vectors

$$\mathbf{c} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{r} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

That is,

$$\mathbf{c}\,\mathbf{r}^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = w$$

A separable kernel of size $m \times n$ can be expressed as the outer product of two vectors, $\mathbf{v}$ and $\mathbf{w}$:

$$w = \mathbf{v}\mathbf{w}^T \tag{3-41}$$

where $\mathbf{v}$ and $\mathbf{w}$ are vectors of size $m \times 1$ and $n \times 1$, respectively. For a square kernel of size $m \times m$, we write

$$w = \mathbf{v}\mathbf{v}^T \tag{3-42}$$

It turns out that the product of a column vector and a row vector is the same as the 2-D convolution of the vectors (see Problem 3.24).

The importance of separable kernels lies in the computational advantages that result from the associative property of convolution. If we have a kernel w that can be decomposed into two simpler kernels, such that $w = w_1 \star w_2$, then it follows from the commutative and associative properties in Table 3.5 that

$$w \star f = (w_1 \star w_2) \star f = (w_2 \star w_1) \star f = w_2 \star (w_1 \star f) = (w_1 \star f) \star w_2 \tag{3-43}$$

This equation says that convolving a separable kernel with an image is the same as convolving w_1 with f first, and then convolving the result with w_2.

We assume that the values of M and N include any padding of f prior to performing convolution.

For an image of size $M \times N$ and a kernel of size $m \times n$, implementation of Eq. (3-35) requires on the order of $MNmn$ multiplications and additions. This is because it follows directly from that equation that *each* pixel in the output (filtered) image depends on *all* the coefficients in the filter kernel. But, if the kernel is separable and we use Eq. (3-43), then the first convolution, $w_1 \star f$, requires on the order of MNm

multiplications and additions because w_1 is of size $m \times 1$. The result is of size $M \times N$, so the convolution of w_2 with the result requires MNn such operations, for a total of $MN(m+n)$ multiplication and addition operations. Thus, the *computational advantage* of performing convolution with a separable, as opposed to a nonseparable, kernel is defined as

$$C = \frac{MNmn}{MN(m+n)} = \frac{mn}{m+n} \tag{3-44}$$

For a kernel of modest size, say 11×11, the computational advantage (and thus execution-time advantage) is a respectable 5.2. For kernels with hundreds of elements, execution times can be reduced by a factor of a hundred or more, which is significant. We will illustrate the use of such large kernels in Example 3.16.

We know from matrix theory that a matrix resulting from the product of a column vector and a row vector *always* has a rank of 1. By definition, a separable kernel is formed by such a product. Therefore, to determine if a kernel is separable, all we have to do is determine if its rank is 1. Typically, we find the rank of a matrix using a pre-programmed function in the computer language being used. For example, if you use MATLAB, function **rank** will do the job.

Once you have determined that the rank of a kernel matrix is 1, it is not difficult to find two vectors $\mathbf{v}$ and $\mathbf{w}$ such that their outer product, $\mathbf{v}\mathbf{w}^T$, is equal to the kernel. The approach consists of only three steps:

1. Find any nonzero element in the kernel and let E denote its value.
2. Form vectors $\mathbf{c}$ and $\mathbf{r}$ equal, respectively, to the column and row in the kernel containing the element found in Step 1.
3. With reference to Eq. (3-41), let $\mathbf{v} = \mathbf{c}$ and $\mathbf{w}^T = \mathbf{r}/E$.

The reason why this simple three-step method works is that the rows and columns of a matrix whose rank is 1 are linearly dependent. That is, the rows differ only by a constant multiplier, and similarly for the columns. It is instructive to work through the mechanics of this procedure using a small kernel (see Problems 3.20 and 3.22).

As we will discuss later in this chapter, the only kernels that are separable *and* whose values are circularly symmetric about the center are Gaussian kernels, which have a nonzero center coefficient (i.e., $c > 0$ for these kernels).

As we explained above, the objective is to find two 1-D kernels, w_1 and w_2, in order to implement 1-D convolution. In terms of the preceding notation, $w_1 = \mathbf{c} = \mathbf{v}$ and $w_2 = \mathbf{r}/E = \mathbf{w}^T$. For circularly symmetric kernels, the column through the center of the kernel describes the entire kernel; that is, $w = \mathbf{v}\mathbf{v}^T/c$, where c is the value of the center coefficient. Then, the 1-D components are $w_1 = \mathbf{v}$ and $w_2 = \mathbf{v}^T/c$.

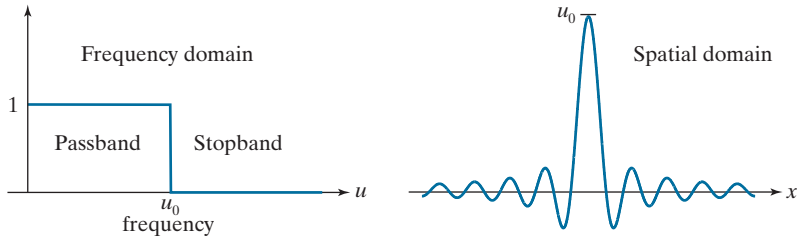
SOME IMPORTANT COMPARISONS BETWEEN FILTERING IN THE SPATIAL AND FREQUENCY DOMAINS

Although filtering in the frequency domain is the topic of Chapter 4, we introduce at this junction some important concepts from the frequency domain that will help you master the material that follows.

The tie between spatial- and frequency-domain processing is the *Fourier transform*. We use the Fourier transform to go from the spatial to the frequency domain;

a b

FIGURE 3.32
(a) Ideal 1-D lowpass filter transfer function in the frequency domain. (b) Corresponding filter kernel in the spatial domain.



to return to the spatial domain we use the *inverse Fourier transform*. This will be covered in detail in Chapter 4. The focus here is on two fundamental properties relating the spatial and frequency domains:

1. Convolution, which is the basis for filtering in the spatial domain, is equivalent to multiplication in the frequency domain, and vice versa.
2. An impulse of strength A in the spatial domain is a constant of value A in the frequency domain, and vice versa.

See the explanation of Eq. (3-33) regarding impulses.

As explained in Chapter 4, a function (e.g., an image) satisfying some mild conditions can be expressed as the sum of sinusoids of different frequencies and amplitudes. Thus, the *appearance* of an image depends on the frequencies of its sinusoidal components—change the frequencies of those components, and you will change the appearance of the image. What makes this a powerful concept is that it is possible to associate certain frequency bands with image characteristics. For example, regions of an image with intensities that vary slowly (e.g., the walls in an image of a room) are characterized by sinusoids of low frequencies. Similarly, edges and other sharp intensity transitions are characterized by high frequencies. Thus, reducing the high-frequency components of an image will tend to blur it.

Linear filtering is concerned with finding suitable ways to modify the frequency content of an image. In the spatial domain we do this via convolution filtering. In the frequency domain we do it with multiplicative filters. The latter is a much more intuitive approach, which is one of the reasons why it is virtually impossible to truly understand spatial filtering without having at least some rudimentary knowledge of the frequency domain.

An example will help clarify these ideas. For simplicity, consider a 1-D function (such as an intensity scan line through an image) and suppose that we want to eliminate all its frequencies above a cutoff value, u_0, while "passing" all frequencies below that value. Figure 3.32(a) shows a frequency-domain filter function for doing this. (The term *filter transfer function* is used to denote filter functions in the frequency domain—this is analogous to our use of the term "filter kernel" in the spatial domain.) Appropriately, the function in Fig. 3.32(a) is called a *lowpass* filter transfer function. In fact, this is an *ideal* lowpass filter function because it eliminates *all* frequencies above u_0, while passing all frequencies below this value.[†] That is, the

As we did earlier with spatial filters, when the meaning is clear we use the term *filter* interchangeably with *filter transfer function* when working in the frequency domain.

[†]All the frequency domain filters in which we are interested are symmetrical about the origin and encompass both positive and negative frequencies, as we will explain in Section 4.3 (see Fig. 4.8). For the moment, we show only the right side (positive frequencies) of 1-D filters for simplicity in this short explanation.

transition of the filter between low and high frequencies is instantaneous. Such filter functions are not realizable with physical components, and have issues with "ringing" when implemented digitally. However, ideal filters are very useful for illustrating numerous filtering phenomena, as you will learn in Chapter 4.

To lowpass-filter a spatial signal in the frequency domain, we first convert it to the frequency domain by computing its Fourier transform, and then *multiply* the result by the filter transfer function in Fig. 3.32(a) to eliminate frequency components with values higher than u_0. To return to the spatial domain, we take the inverse Fourier transform of the filtered signal. The result will be a blurred spatial domain function.

Because of the duality between the spatial and frequency domains, we can obtain the same result in the spatial domain by *convolving* the equivalent spatial domain filter kernel with the input spatial function. The equivalent spatial filter kernel is the inverse Fourier transform of the frequency-domain filter transfer function. Figure 3.32(b) shows the spatial filter kernel corresponding to the frequency domain filter transfer function in Fig. 3.32(a). The ringing characteristics of the kernel are evident in the figure. A central theme of digital filter design theory is obtaining faithful (and practical) approximations to the sharp cut off of ideal frequency domain filters while reducing their ringing characteristics.

A WORD ABOUT HOW SPATIAL FILTER KERNELS ARE CONSTRUCTED

We consider three basic approaches for constructing spatial filters in the following sections of this chapter. One approach is based on formulating filters based on mathematical properties. For example, a filter that computes the average of pixels in a neighborhood blurs an image. Computing an average is analogous to integration. Conversely, a filter that computes the local derivative of an image sharpens the image. We give numerous examples of this approach in the following sections.

A second approach is based on sampling a 2-D spatial function whose shape has a desired property. For example, we will show in the next section that samples from a Gaussian function can be used to construct a weighted-average (lowpass) filter. These 2-D spatial functions sometimes are generated as the inverse Fourier transform of 2-D filters specified in the frequency domain. We will give several examples of this approach in this and the next chapter.

A third approach is to design a spatial filter with a specified frequency response. This approach is based on the concepts discussed in the previous section, and falls in the area of digital filter design. A 1-D spatial filter with the desired response is obtained (typically using filter design software). The 1-D filter values can be expressed as a vector $\mathbf{v}$, and a 2-D separable kernel can then be obtained using Eq. (3-42). Or the 1-D filter can be rotated about its center to generate a 2-D kernel that approximates a circularly symmetric function. We will illustrate these techniques in Section 3.7.

3.5 SMOOTHING (LOWPASS) SPATIAL FILTERS

Smoothing (also called *averaging*) spatial filters are used to reduce sharp transitions in intensity. Because random noise typically consists of sharp transitions in

intensity, an obvious application of smoothing is noise reduction. Smoothing prior to image resampling to reduce aliasing, as will be discussed in Section 4.5, is also a common application. Smoothing is used to reduce irrelevant detail in an image, where "irrelevant" refers to pixel regions that are small with respect to the size of the filter kernel. Another application is for smoothing the false contours that result from using an insufficient number of intensity levels in an image, as discussed in Section 2.4. Smoothing filters are used in combination with other techniques for image enhancement, such as the histogram processing techniques discussed in Section 3.3, and unsharp masking, as discussed later in this chapter. We begin the discussion of smoothing filters by considering linear smoothing filters in some detail. We will introduce nonlinear smoothing filters later in this section.

As we discussed in Section 3.4, linear spatial filtering consists of convolving an image with a filter kernel. Convolving a smoothing kernel with an image blurs the image, with the degree of blurring being determined by the size of the kernel and the values of its coefficients. In addition to being useful in countless applications of image processing, lowpass filters are fundamental, in the sense that other important filters, including sharpening (highpass), bandpass, and bandreject filters, can be derived from lowpass filters, as we will show in Section 3.7.

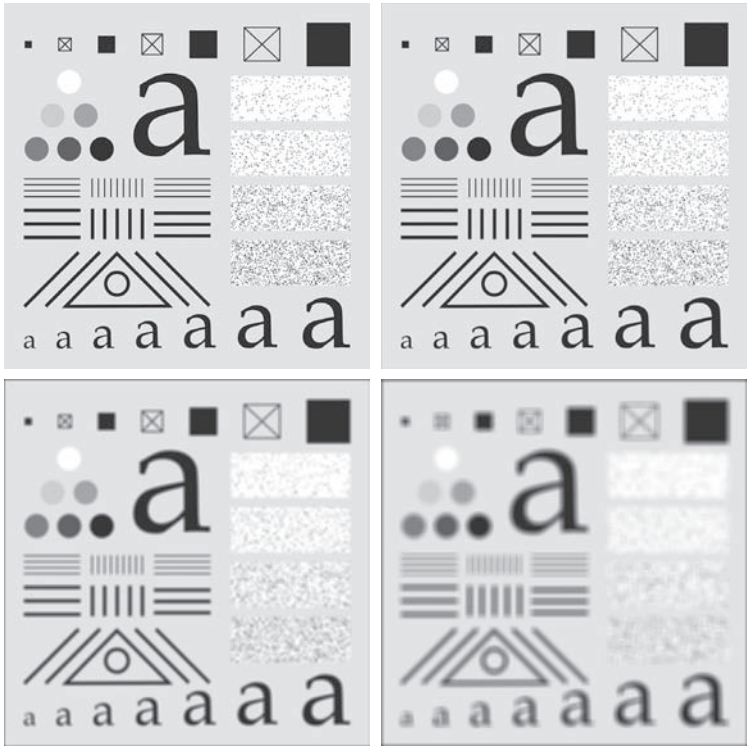
We discuss in this section lowpass filters based on *box* and *Gaussian* kernels, both of which are separable. Most of the discussion will center on Gaussian kernels because of their numerous useful properties and breadth of applicability. We will introduce other smoothing filters in Chapters 4 and 5.

BOX FILTER KERNELS

The simplest, separable lowpass filter kernel is the *box kernel*, whose coefficients have the same value (typically 1). The name "box kernel" comes from a constant kernel resembling a box when viewed in 3-D. We showed a 3×3 box filter in Fig. 3.31(a). An $m \times n$ box filter is an $m \times n$ array of 1's, with a normalizing constant in front, whose value is 1 divided by the sum of the values of the coefficients (i.e., $1/mn$ when all the coefficients are 1's). This normalization, which we apply to all lowpass kernels, has two purposes. First, the average value of an area of constant intensity would equal that intensity in the filtered image, as it should. Second, normalizing the kernel in this way prevents introducing a *bias* during filtering; that is, the sum of the pixels in the original and filtered images will be the same (see Problem 3.31). Because in a box kernel all rows and columns are identical, the rank of these kernels is 1, which, as we discussed earlier, means that they are separable.

EXAMPLE 3.11: Lowpass filtering with a box kernel.

Figure 3.33(a) shows a test pattern image of size 1024×1024 pixels. Figures 3.33(b)-(d) are the results obtained using box filters of size $m \times m$ with $m = 3, 11$, and 21, respectively. For $m = 3$, we note a slight overall blurring of the image, with the image features whose sizes are comparable to the size of the kernel being affected significantly more. Such features include the thinner lines in the image and the noise pixels contained in the boxes on the right side of the image. The filtered image also has a thin gray border, the result of zero-padding the image prior to filtering. As indicated earlier, padding extends the boundaries of an image to avoid undefined operations when parts of a kernel lie outside the border of



a b
c d

FIGURE 3.33
(a) Test pattern of size 1024×1024 pixels.
(b)-(d) Results of lowpass filtering with box kernels of sizes 3×3, 11×11, and 21×21, respectively.

the image during filtering. When zero (black) padding is used, the net result of smoothing at or near the border is a dark gray border that arises from including black pixels in the averaging process. Using the 11×11 kernel resulted in more pronounced blurring throughout the image, including a more prominent dark border. The result with the 21×21 kernel shows significant blurring of all components of the image, including the loss of the characteristic shape of some components, including, for example, the small square on the top left and the small character on the bottom left. The dark border resulting from zero padding is proportionally thicker than before. We used zero padding here, and will use it a few more times, so that you can become familiar with its effects. In Example 3.14 we discuss two other approaches to padding that eliminate the dark-border artifact that usually results from zero padding.

LOWPASS GAUSSIAN FILTER KERNELS

Because of their simplicity, box filters are suitable for quick experimentation and they often yield smoothing results that are visually acceptable. They are useful also when it is desired to reduce the effect of smoothing on edges (see Example 3.13). However, box filters have limitations that make them poor choices in many applications. For example, a defocused lens is often modeled as a lowpass filter, but box filters are poor approximations to the blurring characteristics of lenses (see Problem 3.33). Another limitation is the fact that box filters favor blurring along perpendicular directions. In applications involving images with a high level of detail,

or with strong geometrical components, the directionality of box filters often produces undesirable results. (Example 3.13 illustrates this issue.) These are but two applications in which box filters are not suitable.

The kernels of choice in applications such as those just mentioned are *circularly symmetric* (also called *isotropic*, meaning their response is independent of orientation). As it turns out, Gaussian kernels of the form

Our interest here is strictly on the bell shape of the Gaussian function; thus, we dispense with the traditional multiplier of the Gaussian PDF and use a general constant, K, instead. Recall that σ controls the "spread" of a Gaussian function about its mean.

$$w(s,t) = G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}} \tag{3-45}$$

are the *only* circularly symmetric kernels that are also separable (Sahoo [1990]). Thus, because Gaussian kernels of this form are separable, Gaussian filters enjoy the same computational advantages as box filters, but have a host of additional properties that make them ideal for image processing, as you will learn in the following discussion. Variables s and t in Eq. (3-45), are real (typically discrete) numbers.

By letting $r = [s^2 + t^2]^{1/2}$ we can write Eq. (3-45) as

$$G(r) = Ke^{-\frac{r^2}{2\sigma^2}} \tag{3-46}$$

This equivalent form simplifies derivation of expressions later in this section. This form also reminds us that the function is circularly symmetric. Variable r is the distance from the center to any point on function G. Figure 3.34 shows values of r for several kernel sizes using integer values for s and t. Because we work generally with odd kernel sizes, the centers of such kernels fall on integer values, and it follows that all values of r^2 are integers also. You can see this by squaring the values in Fig. 3.34

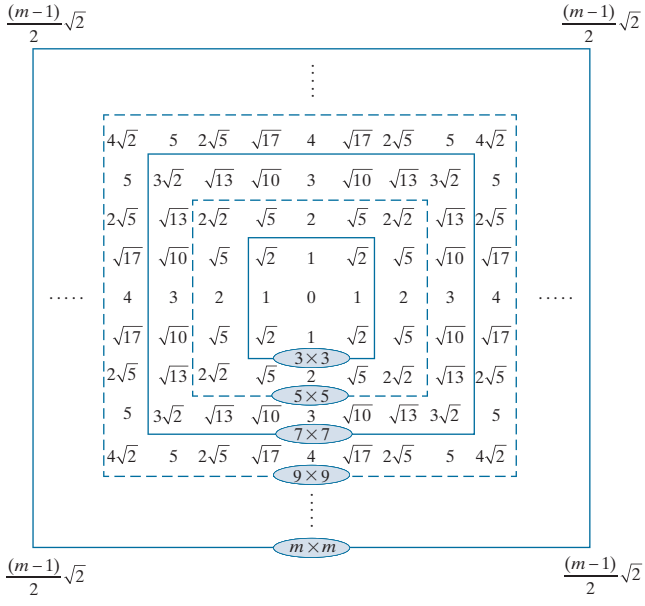


FIGURE 3.34 Distances from the center for various sizes of square kernels.

(for a formal proof, see Padfield [2011]). Note in particular that the distance squared to the corner points for a kernel of size $m \times m$ is

$$r_{\max}^2 = \left[\frac{(m-1)}{2}\sqrt{2}\right]^2 = \frac{(m-1)^2}{2} \tag{3-47}$$

Small Gaussian kernels cannot capture the characteristic Gaussian bell shape, and thus behave more like box kernels. As we discuss below, a practical size for Gaussian kernels is on the order of $6\sigma \times 6\sigma$.

The kernel in Fig. 3.31(b) was obtained by sampling Eq. (3-45) (with $K = 1$ and $\sigma = 1$). Figure 3.35(a) shows a perspective plot of a Gaussian function, and illustrates that the samples used to generate that kernel were obtained by specifying values of s and t, then "reading" the values of the function at those coordinates. These values are the coefficients of the kernel. Normalizing the kernel by dividing its coefficients by the sum of the coefficients completes the specification of the kernel. The reasons for normalizing the kernel are as discussed in connection with box kernels. Because Gaussian kernels are separable, we could simply take samples along a cross section through the center and use the samples to form vector $\mathbf{v}$ in Eq. (3-42), from which we obtain the 2-D kernel.

As we explained in Section 2.6, the symbols $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the *ceiling* and *floor* functions. That is, the ceiling and floor functions map a real number to the smallest following, or the largest previous, integer, respectively.

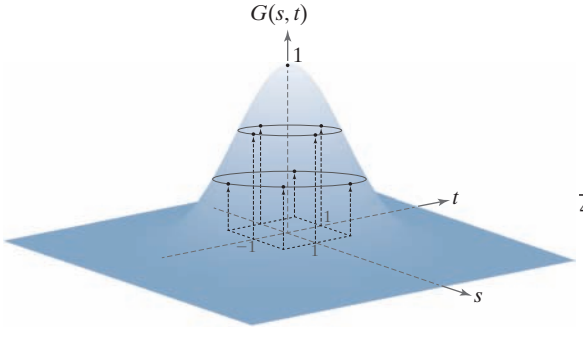
Separability is one of many fundamental properties of circularly symmetric Gaussian kernels. For example, we know that the values of a Gaussian function at a distance larger than 3σ from the mean are small enough that they can be ignored. This means that if we select the size of a Gaussian kernel to be $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$ (the notation $\lceil c \rceil$ is used to denote the *ceiling* of c; that is, the smallest integer not less than c), we are assured of getting essentially the same result as if we had used an arbitrarily large Gaussian kernel. Viewed another way, this property tells us that there is nothing to be gained by using a Gaussian kernel larger than $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$ for image processing. Because typically we work with kernels of odd dimensions, we would use the smallest *odd* integer that satisfies this condition (e.g., a 43×43 kernel if $\sigma = 7$).

Proofs of the results in Table 3.6 are simplified by working with the Fourier transform and the frequency domain, both of which are topics in Chapter 4.

Two other fundamental properties of Gaussian functions are that the product and convolution of two Gaussians are Gaussian functions also. Table 3.6 shows the mean and standard deviation of the product and convolution of two 1-D Gaussian functions, f and g (remember, because of separability, we only need a 1-D Gaussian to form a circularly symmetric 2-D function). The mean and standard deviation

a b

FIGURE 3.35
(a) Sampling a Gaussian function to obtain a discrete Gaussian kernel. The values shown are for $K = 1$ and $\sigma = 1$. (b) Resulting 3×3 kernel [this is the same as Fig. 3.31(b)].



$$\frac{1}{4.8976} \times \begin{bmatrix} 0.3679 & 0.6065 & 0.3679 \\ 0.6065 & 1.0000 & 0.6065 \\ 0.3679 & 0.6065 & 0.3679 \end{bmatrix}$$

TABLE 3.6 Mean and standard deviation of the product ($\times$) and convolution ($\star$) of two 1-D Gaussian functions, f and g. These results generalize directly to the product and convolution of more than two 1-D Gaussian functions (see Problem 3.25).

	f	g	$f \times g$	$f \star g$
Mean	m_f	m_g	$m_{f\times g} = \dfrac{m_f\sigma_g^2 + m_g\sigma_f^2}{\sigma_f^2 + \sigma_g^2}$	$m_{f\star g} = m_f + m_g$
Standard deviation	σ_f	σ_g	$\sigma_{f\times g} = \sqrt{\dfrac{\sigma_f^2\sigma_g^2}{\sigma_f^2 + \sigma_g^2}}$	$\sigma_{f\star g} = \sqrt{\sigma_f^2 + \sigma_g^2}$

completely define a Gaussian, so the parameters in Table 3.6 tell us all there is to know about the functions resulting from multiplication and convolution of Gaussians. As indicated by Eqs. (3-45) and (3-46), Gaussian kernels have zero mean, so our interest here is in the standard deviations.

The convolution result is of particular importance in filtering. For example, we mentioned in connection with Eq. (3-43) that filtering sometimes is done in successive stages, and that the same result can be obtained by one stage of filtering with a composite kernel formed as the convolution of the individual kernels. If the kernels are Gaussian, we can use the result in Table 3.6 (which, as noted, generalizes directly to more than two functions) to compute the standard deviation of the composite kernel (and thus completely define it) without actually having to perform the convolution of all the individual kernels.

EXAMPLE 3.12: Lowpass filtering with a Gaussian kernel.

To compare Gaussian and box kernel filtering, we repeat Example 3.11 using a Gaussian kernel. Gaussian kernels have to be larger than box filters to achieve the same degree of blurring. This is because, whereas a box kernel assigns the same weight to all pixels, the values of Gaussian kernel coefficients (and hence their effect) decreases as a function of distance from the kernel center. As explained earlier, we use a size equal to the closest odd integer to $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$. Thus, for a Gaussian kernel of size 21×21, which is the size of the kernel we used to generate Fig. 3.33(d), we need $\sigma = 3.5$. Figure 3.36(b) shows the result of lowpass filtering the test pattern with this kernel. Comparing this result with Fig. 3.33(d), we see that the Gaussian kernel resulted in significantly less blurring. A little experimentation would show that we need $\sigma = 7$ to obtain comparable results. This implies a Gaussian kernel of size 43×43. Figure 3.36(c) shows the results of filtering the test pattern with this kernel. Comparing it with Fig. 3.33(d), we see that the results indeed are very close.

We mentioned earlier that there is little to be gained by using a Gaussian kernel larger than $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$. To demonstrate this, we filtered the test pattern in Fig. 3.36(a) using a Gaussian kernel with $\sigma = 7$ again, but of size 85×85. Figure 3.37(a) is the same as Fig. 3.36(c), which we generated using the smallest odd kernel satisfying the $\lceil 6 \rceil \times \lceil 6 \rceil$ condition (43×43, for $\sigma = 7$). Figure 3.37(b) is the result of using the 85×85 kernel, which is double the size of the other kernel. As you can see, not discernible additional

a b c

FIGURE 3.36 (a)A test pattern of size 1024×1024. (b) Result of lowpass filtering the pattern with a Gaussian kernel of size 21×21, with standard deviations $\sigma = 3.5$. (c) Result of using a kernel of size 43×43, with $\sigma = 7$. This result is comparable to Fig. 3.33(d). We used $K = 1$ in all cases.

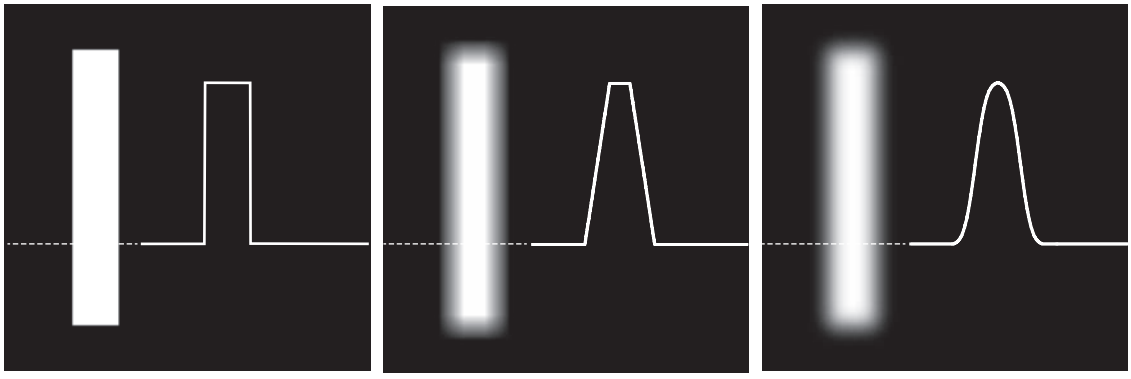
blurring occurred. In fact, the difference image in Fig 3.37(c) indicates that the two images are nearly identical, their maximum difference being 0.75, which is less than one level out of 256 (these are 8-bit images).

EXAMPLE 3.13: Comparison of Gaussian and box filter smoothing characteristics.

The results in Examples 3.11 and 3.12 showed little visual difference in blurring. Despite this, there are some subtle differences that are not apparent at first glance. For example, compare the large letter "a" in Figs. 3.33(d) and 3.36(c); the latter is much smoother around the edges. Figure 3.38 shows this type of different behavior between box and Gaussian kernels more clearly. The image of the rectangle was

a b c

FIGURE 3.37 (a) Result of filtering Fig. 3.36(a) using a Gaussian kernels of size 43×43, with $\sigma = 7$. (b) Result of using a kernel of 85×85, with the same value of σ. (c) Difference image.



a b c

FIGURE 3.38 (a) Image of a white rectangle on a black background, and a horizontal intensity profile along the scan line shown dotted. (b) Result of smoothing this image with a box kernel of size 71×71, and corresponding intensity profile. (c) Result of smoothing the image using a Gaussian kernel of size 151×151, with $K = 1$ and $\sigma = 25$. Note the smoothness of the profile in (c) compared to (b). The image and rectangle are of sizes 1024×1024 and 768×128 pixels, respectively.

smoothed using a box and a Gaussian kernel with the sizes and parameters listed in the figure. These parameters were selected to give blurred rectangles of approximately the same width and height, in order to show the effects of the filters on a comparable basis. As the intensity profiles show, the box filter produced linear smoothing, with the transition from black to white (i.e., at an edge) having the shape of a ramp. The important features here are hard transitions at the onset and end of the ramp. We would use this type of filter when less smoothing of edges is desired. Conversely, the Gaussian filter yielded significantly smoother results around the edge transitions. We would use this type of filter when generally uniform smoothing is desired.

As the results in Examples 3.11, 3.12, and 3.13 show, zero padding an image introduces dark borders in the filtered result, with the thickness of the borders depending on the size and type of the filter kernel used. Earlier, when discussing correlation and convolution, we mentioned two other methods of image padding: *mirror* (also called *symmetric*) *padding*, in which values outside the boundary of the image are obtained by mirror-reflecting the image across its border; and *replicate padding*, in which values outside the boundary are set equal to the nearest image border value. The latter padding is useful when the areas near the border of the image are constant. Conversely, mirror padding is more applicable when the areas near the border contain image details. In other words, these two types of padding attempt to "extend" the characteristics of an image past its borders.

Figure 3.39 illustrates these padding methods, and also shows the effects of more aggressive smoothing. Figures 3.39(a) through 3.39(c) show the results of filtering Fig. 3.36(a) with a Gaussian kernel of size 187×187 elements with $K = 1$ and $\sigma = 31$, using zero, mirror, and replicate padding, respectively. The differences between the borders of the results with the zero-padded image and the other two are obvious,

a b c

FIGURE 3.39 Result of filtering the test pattern in Fig. 3.36(a) using (a) zero padding, (b) mirror padding, and (c) replicate padding. A Gaussian kernel of size 187×187, with $K = 1$ and $\sigma = 31$ was used in all three cases.

and indicate that mirror and replicate padding yield more visually appealing results by eliminating the dark borders resulting from zero padding.

EXAMPLE 3.14: Smoothing performance as a function of kernel and image size.

The amount of relative blurring produced by a smoothing kernel of a given size depends directly on image size. To illustrate, Fig. 3.40(a) shows the same test pattern used earlier, but of size 4096×4096 pixels, four times larger in each dimension than before. Figure 3.40(b) shows the result of filtering this image with the same Gaussian kernel and padding used in Fig. 3.39(b). By comparison, the former image shows considerably less blurring for the same size filter. In fact, Fig. 3.40(b) looks more like the

a b c

FIGURE 3.40 (a) Test pattern of size 4096×4096 pixels. (b) Result of filtering the test pattern with the same Gaussian kernel used in Fig. 3.39. (c) Result of filtering the pattern using a Gaussian kernel of size 745×745 elements, with $K = 1$ and $\sigma = 124$. Mirror padding was used throughout.

image in Fig. 3.36(d), which was filtered using a 43×43 Gaussian kernel. In order to obtain results that are comparable to Fig. 3.39(b) we have to increase the size and standard deviation of the Gaussian kernel by four, the same factor as the increase in image dimensions. This gives a kernel of (odd) size 745×745 (with $K = 1$ and $\sigma = 124$). Figure 3.40(c) shows the result of using this kernel with mirror padding. This result is quite similar to Fig. 3.39(b). After the fact, this may seem like a trivial observation, but you would be surprised at how frequently not understanding the relationship between kernel size and the size of objects in an image can lead to ineffective performance of spatial filtering algorithms.

EXAMPLE 3.15: Using lowpass filtering and thresholding for region extraction.

Figure 3.41(a) is a 2566×2758 Hubble Telescope image of the *Hickson Compact Group* (see figure caption), whose intensities were scaled to the range $[0, 1]$. Our objective is to illustrate lowpass filtering combined with intensity thresholding for eliminating irrelevant detail in this image. In the present context, "irrelevant" refers to pixel regions that are small compared to kernel size.

Figure 3.41(b) is the result of filtering the original image with a Gaussian kernel of size 151×151 (approximately 6% of the image width) and standard deviation $\sigma = 25$. We chose these parameter values in order generate a sharper, more selective Gaussian kernel shape than we used in earlier examples. The filtered image shows four predominantly bright regions. We wish to extract only those regions from the image. Figure 3.41(c) is the result of thresholding the filtered image with a threshold $T = 0.4$ (we will discuss threshold selection in Chapter 10). As the figure shows, this approach effectively extracted the four regions of interest, and eliminated details deemed irrelevant in this application.

EXAMPLE 3.16: Shading correction using lowpass filtering.

One of the principal causes of image shading is nonuniform illumination. *Shading correction* (also called *flat-field correction*) is important because shading is a common cause of erroneous measurements, degraded performance of automated image analysis algorithms, and difficulty of image interpretation

a b c

FIGURE 3.41 (a) A 2566×2758 Hubble Telescope image of the *Hickson Compact Group*. (b) Result of lowpass filtering with a Gaussian kernel. (c) Result of thresholding the filtered image (intensities were scaled to the range $[0, 1]$). The Hickson Compact Group contains dwarf galaxies that have come together, setting off thousands of new star clusters. (Original image courtesy of NASA.)

a b c

FIGURE 3.42 (a) Image shaded by a shading pattern oriented in the −45° direction. (b) Estimate of the shading patterns obtained using lowpass filtering. (c) Result of dividing (a) by (b). (See Section 9.8 for a morphological approach to shading correction).

by humans. We introduced shading correction in Example 2.7, where we corrected a shaded image by dividing it by the shading pattern. In that example, the shading pattern was given. Often, that is not the case in practice, and we are faced with having to estimate the pattern directly from available samples of shaded images. Lowpass filtering is a rugged, simple method for estimating shading patterns.

Consider the 2048×2048 checkerboard image in Fig. 3.42(a), whose inner squares are of size 128×128 pixels. Figure 3.42(b) is the result of lowpass filtering the image with a 512×512 Gaussian kernel (four times the size of the squares), $K = 1$, and $\sigma = 128$ (equal to the size of the squares). This kernel is just large enough to blur-out the squares (a kernel three times the size of the squares is too small to blur them out sufficiently). This result is a good approximation to the shading pattern visible in Fig. 3.42(a). Finally, Fig. 3.42(c) is the result of dividing (a) by (b). Although the result is not perfectly flat, it definitely is an improvement over the shaded image.

In the discussion of separable kernels in Section 3.4, we pointed out that the computational advantage of separable kernels can be significant for large kernels. It follows from Eq. (3-44) that the computational advantage of the kernel used in this example (which of course is separable) is 262 to 1. Thinking of computation time, if it took 30 sec to process a set of images similar to Fig. 3.42(b) using the two 1-D separable components of the Gaussian kernel, it would have taken 2.2 hrs to achieve the same result using a nonseparable lowpass kernel, or if we had used the 2-D Gaussian kernel directly, without decomposing it into its separable parts.

ORDER-STATISTIC (NONLINEAR) FILTERS

Order-statistic filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the region encompassed by the filter. Smoothing is achieved by replacing the value of the center pixel with the value determined by the ranking result. The best-known filter in this category is the *median filter*, which, as its name implies, replaces the value of the center pixel by the median of the intensity values in the neighborhood of that pixel (the value of the center pixel is included

in computing the median). Median filters provide excellent noise reduction capabilities for certain types of random noise, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of *impulse noise* (sometimes called *salt-and-pepper noise,* when it manisfests itself as white and black dots superimposed on an image).

The *median*, ξ, of a set of values is such that half the values in the set are less than or equal to ξ and half are greater than or equal to ξ. In order to perform median filtering at a point in an image, we first sort the values of the pixels in the neighborhood, determine their median, and assign that value to the pixel in the filtered image corresponding to the center of the neighborhood. For example, in a 3×3 neighborhood the median is the 5th largest value, in a 5×5 neighborhood it is the 13th largest value, and so on. When several values in a neighborhood are the same, all equal values are grouped. For example, suppose that a 3×3 neighborhood has values (10, 20, 20, 20, 15, 20, 20, 25, 100). These values are sorted as (10, 15, 20, 20, 20, 20, 20, 25, 100), which results in a median of 20. Thus, the principal function of median filters is to force points to be more like their neighbors. Isolated clusters of pixels that are light or dark with respect to their neighbors, and whose area is less than $m^2/2$ (one-half the filter area), are forced by an $m \times m$ median filter to have the value of the median intensity of the pixels in the neighborhood (see Problem 3.36).

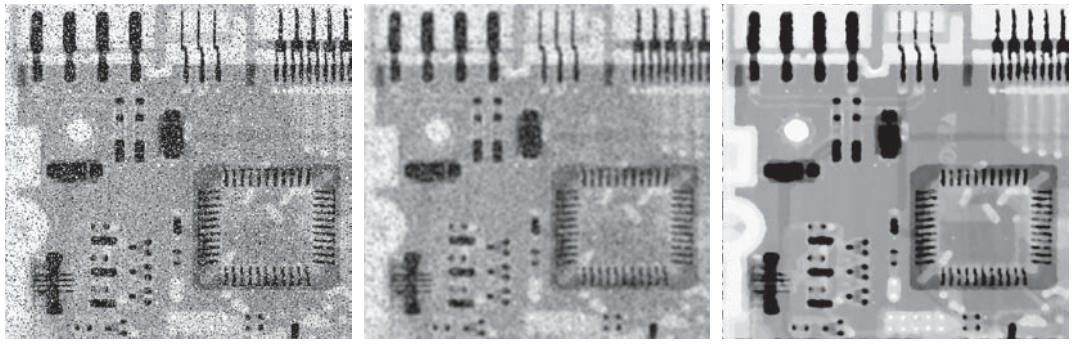
The median filter is by far the most useful order-statistic filter in image processing, but is not the only one. The median represents the 50th percentile of a ranked set of numbers, but ranking lends itself to many other possibilities. For example, using the 100th percentile results in the so-called *max filter*, which is useful for finding the brightest points in an image or for eroding dark areas adjacent to light regions. The response of a 3×3 max filter is given by $R = \max\{z_k \mid k = 1, 2, 3, \ldots, 9\}$. The 0th percentile filter is the *min filter*, used for the opposite purpose. Median, max, min, and several other nonlinear filters will be considered in more detail in Section 5.3.

EXAMPLE 3.17: Median filtering.

Figure 3.43(a) shows an X-ray image of a circuit board heavily corrupted by salt-and-pepper noise. To illustrate the superiority of median filtering over lowpass filtering in situations such as this, we show in Fig. 3.43(b) the result of filtering the noisy image with a Gaussian lowpass filter, and in Fig. 3.43(c) the result of using a median filter. The lowpass filter blurred the image and its noise reduction performance was poor. The superiority in all respects of median over lowpass filtering in this case is evident.

3.6 SHARPENING (HIGHPASS) SPATIAL FILTERS

Sharpening highlights transitions in intensity. Uses of image sharpening range from electronic printing and medical imaging to industrial inspection and autonomous guidance in military systems. In Section 3.5, we saw that image blurring could be accomplished in the spatial domain by pixel averaging (smoothing) in a neighborhood. Because averaging is analogous to integration, it is logical to conclude that sharpening can be accomplished by spatial differentiation. In fact, this is the case, and the following discussion deals with various ways of defining and implementing operators for sharpening by digital differentiation. The strength of the response of



a b c

FIGURE 3.43 (a) X-ray image of a circuit board, corrupted by salt-and-pepper noise. (b) Noise reduction using a 19×19 Gaussian lowpass filter kernel with $\sigma = 3$. (c) Noise reduction using a 7×7 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

a derivative operator is proportional to the magnitude of the intensity discontinuity at the point at which the operator is applied. Thus, image differentiation enhances edges and other discontinuities (such as noise) and de-emphasizes areas with slowly varying intensities. As noted in Section 3.5, smoothing is often referred to as lowpass filtering, a term borrowed from frequency domain processing. In a similar manner, sharpening is often referred to as *highpass* filtering. In this case, high frequencies (which are responsible for fine details) are passed, while low frequencies are attenuated or rejected.

FOUNDATION

In the two sections that follow, we will consider in some detail sharpening filters that are based on first- and second-order derivatives, respectively. Before proceeding with that discussion, however, we stop to look at some of the fundamental properties of these derivatives in a digital context. To simplify the explanation, we focus attention initially on one-dimensional derivatives. In particular, we are interested in the behavior of these derivatives in areas of constant intensity, at the onset and end of discontinuities (*step* and *ramp discontinuities*), and along *intensity ramps*. As you will see in Chapter 10, these types of discontinuities can be used to model noise points, lines, and edges in an image.

Derivatives of a digital function are defined in terms of differences. There are various ways to define these differences. However, we require that any definition we use for a *first derivative:*

1. Must be zero in areas of constant intensity.
2. Must be nonzero at the onset of an intensity step or ramp.
3. Must be nonzero along intensity ramps.

Similarly, any definition of a *second derivative*

1. Must be zero in areas of constant intensity.
2. Must be nonzero at the onset *and* end of an intensity step or ramp.
3. Must be zero along intensity ramps.

We are dealing with digital quantities whose values are finite. Therefore, the maximum possible intensity change also is finite, and the shortest distance over which that change can occur is between adjacent pixels.

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

We will return to Eq. (3-48) in Section 10.2 and show how it follows from a Taylor series expansion. For now, we accept it as a definition.

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \tag{3-48}$$

We used a partial derivative here in order to keep the notation consistent when we consider an image function of two variables, $f(x, y)$, at which time we will be dealing with partial derivatives along the two spatial axes. Clearly, $\partial f/\partial x = df/dx$ when there is only one variable in the function; the same is true for the second derivative.

We define the *second-order* derivative of $f(x)$ as the difference

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x) \tag{3-49}$$

These two definitions satisfy the conditions stated above, as we illustrate in Fig. 3.44, where we also examine the similarities and differences between first- and second-order derivatives of a digital function.

The values denoted by the small squares in Fig. 3.44(a) are the intensity values along a horizontal intensity profile (the dashed line connecting the squares is included to aid visualization). The actual numerical values of the scan line are shown inside the small boxes in 3.44(b). As Fig. 3.44(a) shows, the scan line contains three sections of constant intensity, an intensity ramp, and an intensity step. The circles indicate the onset or end of intensity transitions. The first- and second-order derivatives, computed using the two preceding definitions, are shown below the scan line values in Fig. 3.44(b), and are plotted in Fig. 3.44(c). When computing the first derivative at a location x, we subtract the value of the function at that location from the next point, as indicated in Eq. (3-48), so this is a "look-ahead" operation. Similarly, to compute the second derivative at x, we use the previous and the next points in the computation, as indicated in Eq. (3-49). To avoid a situation in which the previous or next points are outside the range of the scan line, we show derivative computations in Fig. 3.44 from the second through the penultimate points in the sequence.

As we traverse the profile from left to right we encounter first an area of constant intensity and, as Figs. 3.44(b) and (c) show, both derivatives are zero there, so condition (1) is satisfied by both. Next, we encounter an intensity ramp followed by a step, and we note that the first-order derivative is nonzero at the onset of the ramp and the step; similarly, the second derivative is nonzero at the onset and end of both the ramp and the step; therefore, property (2) is satisfied by both derivatives. Finally, we

a
b
c

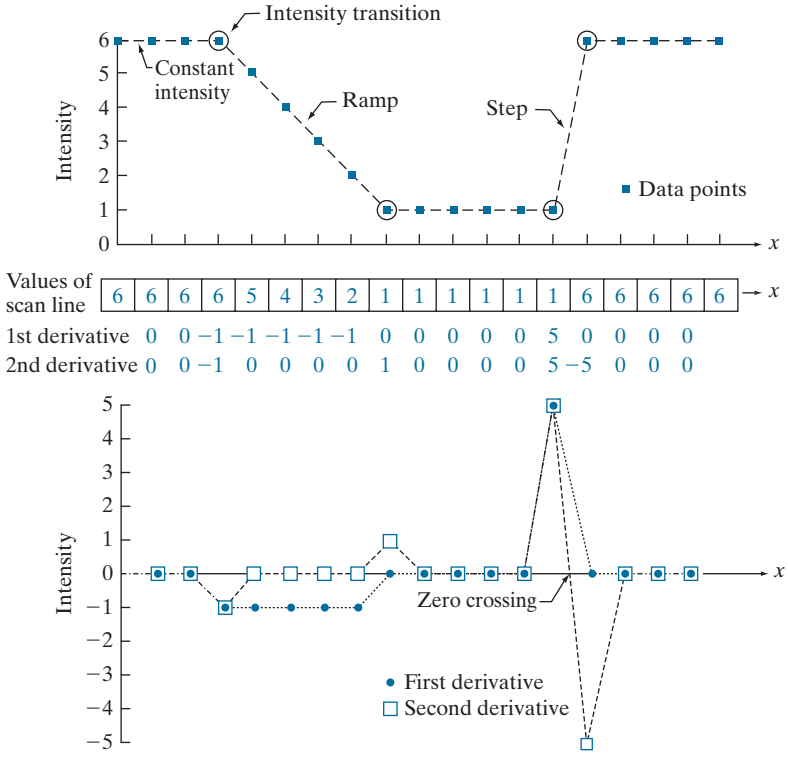


FIGURE 3.44
(a) A section of a horizontal scan line from an image, showing ramp and step edges, as well as constant segments. (b)Values of the scan line and its derivatives. (c) Plot of the derivatives, showing a zero crossing. In (a) and (c) points were joined by dashed lines as a visual aid.

see that property (3) is satisfied also by both derivatives because the first derivative is nonzero and the second is zero along the ramp. Note that the sign of the second derivative changes at the onset and end of a step or ramp. In fact, we see in Fig. 3.44(c) that in a step transition a line joining these two values crosses the horizontal axis midway between the two extremes. This *zero crossing* property is quite useful for locating edges, as you will see in Chapter 10.

Edges in digital images often are ramp-like transitions in intensity, in which case the first derivative of the image would result in thick edges because the derivative is nonzero along a ramp. On the other hand, the second derivative would produce a double edge one pixel thick, separated by zeros. From this, we conclude that the second derivative enhances fine detail much better than the first derivative, a property ideally suited for sharpening images. Also, second derivatives require fewer operations to implement than first derivatives, so our initial attention is on the former.

USING THE SECOND DERIVATIVE FOR IMAGE SHARPENING—THE LAPLACIAN

We will return to the second derivative in Chapter 10, where we use it extensively for image segmentation.

In this section we discuss the implementation of 2-D, second-order derivatives and their use for image sharpening. The approach consists of defining a discrete formulation of the second-order derivative and then constructing a filter kernel based on

that formulation. As in the case of Gaussian lowpass kernels in Section 3.5, we are interested here in isotropic kernels, whose response is independent of the direction of intensity discontinuities in the image to which the filter is applied.

It can be shown (Rosenfeld and Kak [1982]) that the simplest isotropic derivative operator (kernel) is the *Laplacian*, which, for a function (image) $f(x, y)$ of two variables, is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3-50}$$

Because derivatives of any order are linear operations, the Laplacian is a linear operator. To express this equation in discrete form, we use the definition in Eq. (3-49), keeping in mind that we now have a second variable. In the x-direction, we have

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{3-51}$$

and, similarly, in the y-direction, we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \tag{3-52}$$

It follows from the preceding three equations that the discrete Laplacian of two variables is

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \tag{3-53}$$

This equation can be implemented using convolution with the kernel in Fig. 3.45(a); thus, the filtering mechanics for image sharpening are as described in Section 3.5 for lowpass filtering; we are simply using different coefficients here.

The kernel in Fig. 3.45(a) is isotropic for rotations in increments of 90° with respect to the x- and y-axes. The diagonal directions can be incorporated in the definition of the digital Laplacian by adding four more terms to Eq. (3-53). Because each diagonal term would contains a $-2f(x, y)$ term, the total subtracted from the difference terms

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

a b c d

FIGURE 3.45 (a) Laplacian kernel used to implement Eq. (3-53). (b) Kernel used to implement an extension of this equation that includes the diagonal terms. (c) and (d) Two other Laplacian kernels.

now would be $-8f(x, y)$. Figure 3.45(b) shows the kernel used to implement this new definition. This kernel yields isotropic results in increments of 45°. The kernels in Figs. 3.45(c) and (d) also are used to compute the Laplacian. They are obtained from definitions of the second derivatives that are the negatives of the ones we used here. They yield equivalent results, but the difference in sign must be kept in mind when combining a Laplacian-filtered image with another image.

Because the Laplacian is a derivative operator, it highlights sharp intensity transitions in an image and de-emphasizes regions of slowly varying intensities. This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be "recovered" while still preserving the sharpening effect of the Laplacian by adding the Laplacian image to the original. As noted in the previous paragraph, it is important to keep in mind which definition of the Laplacian is used. If the definition used has a negative center coefficient, then we *subtract* the Laplacian image from the original to obtain a sharpened result. Thus, the basic way in which we use the Laplacian for image sharpening is

$$g(x, y) = f(x, y) + c\left[\nabla^2 f(x, y)\right] \tag{3-54}$$

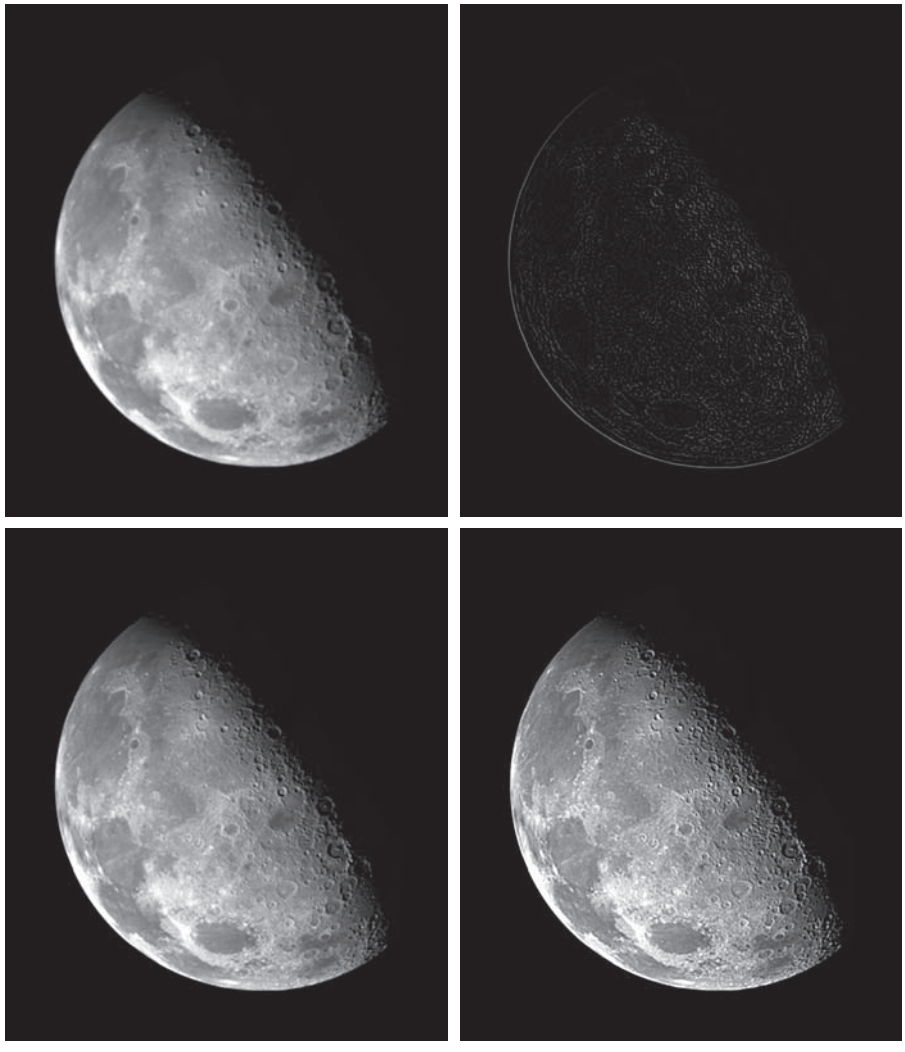
where $f(x, y)$ and $g(x, y)$ are the input and sharpened images, respectively. We let $c = -1$ if the Laplacian kernels in Fig. 3.45(a) or (b) is used, and $c = 1$ if either of the other two kernels is used.

EXAMPLE 3.18: Image sharpening using the Laplacian.

Figure 3.46(a) shows a slightly blurred image of the North Pole of the moon, and Fig. 3.46(b) is the result of filtering this image with the Laplacian kernel in Fig. 3.45(a) directly. Large sections of this image are black because the Laplacian image contains both positive and negative values, and all negative values are clipped at 0 by the display.

Figure 3.46(c) shows the result obtained using Eq. (3-54), with $c = -1$, because we used the kernel in Fig. 3.45(a) to compute the Laplacian. The detail in this image is unmistakably clearer and sharper than in the original image. Adding the Laplacian to the original image restored the overall intensity variations in the image. Adding the Laplacian increased the contrast at the locations of intensity discontinuities. The net result is an image in which small details were enhanced and the background tonality was reasonably preserved. Finally, Fig. 3.46(d) shows the result of repeating the same procedure but using the kernel in Fig. 3.45(b). Here, we note a significant improvement in sharpness over Fig. 3.46(c). This is not unexpected because using the kernel in Fig. 3.45(b) provides additional differentiation (sharpening) in the diagonal directions. Results such as those in Figs. 3.46(c) and (d) have made the Laplacian a tool of choice for sharpening digital images.

Because Laplacian images tend to be dark and featureless, a typical way to scale these images for display is to use Eqs. (2-31) and (2-32). This brings the most negative value to 0 and displays the full range of intensities. Figure 3.47 is the result of processing Fig. 3.46(b) in this manner. The dominant features of the image are edges and sharp intensity discontinuities. The background, previously black, is now gray as a result of scaling. This grayish appearance is typical of Laplacian images that have been scaled properly.



a b
c d

FIGURE 3.46
(a) Blurred image of the North Pole of the moon. (b) Laplacian image obtained using the kernel in Fig. 3.45(a). (c) Image sharpened using Eq. (3-54) with $c = -1$. (d) Image sharpened using the same procedure, but with the kernel in Fig. 3.45(b). (Original image courtesy of NASA.)

Observe in Fig. 3.45 that the coefficients of each kernel sum to zero. Convolution-based filtering implements a sum of products, so when a derivative kernel encompasses a constant region in a image, the result of convolution in that location must be zero. Using kernels whose coefficients sum to zero accomplishes this.

In Section 3.5, we normalized smoothing kernels so that the sum of their coefficients would be one. Constant areas in images filtered with these kernels would be constant also in the filtered image. We also found that the sum of the pixels in the original and filtered images were the same, thus preventing a bias from being introduced by filtering (see Problem 3.31). When convolving an image with a kernel

FIGURE 3.47 The Laplacian image from Fig. 3.46(b), scaled to the full [0, 255] range of intensity values. Black pixels correspond to the most negative value in the unscaled Laplacian image, grays are intermediate values, and white pixels corresponds to the highest positive value.

whose coefficients sum to zero, it turns out that the pixels of the filtered image *will sum to zero also* (see Problem 3.32). This implies that images filtered with such kernels will have negative values, and sometimes will require additional processing to obtain suitable visual results. Adding the filtered image to the original, as we did in Eq. (3-54), is an example of such additional processing.

UNSHARP MASKING AND HIGHBOOST FILTERING

The photographic process of unsharp masking is based on creating a blurred positive and using it along with the original negative to create a sharper image. Our interest is in the *digital* equivalent of this process.

Subtracting an unsharp (smoothed) version of an image from the original image is process that has been used since the 1930s by the printing and publishing industry to sharpen images. This process, called *unsharp masking*, consists of the following steps:

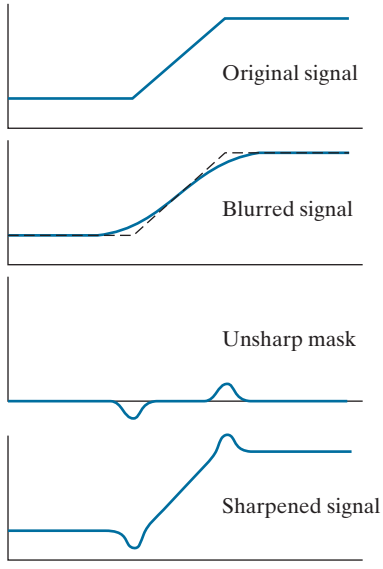
1. Blur the original image.
2. Subtract the blurred image from the original (the resulting difference is called the *mask*.)
3. Add the mask to the original.

Letting $\bar{f}(x, y)$ denote the blurred image, the mask in equation form is given by:

$$g_{\text{mask}}(x, y) = f(x, y) - \bar{f}(x, y) \tag{3-55}$$

Then we add a weighted portion of the mask back to the original image:

$$g(x, y) = f(x, y) + k\, g_{\text{mask}}(x, y) \tag{3-56}$$



a
b
c
d

FIGURE 3.48
1-D illustration of the mechanics of unsharp masking. (a) Original signal. (b) Blurred signal with original shown dashed for reference. (c) Unsharp mask. (d) Sharpened signal, obtained by adding (c) to (a).

where we included a weight, k ($k \geq 0$), for generality. When $k = 1$ we have unsharp masking, as defined above. When $k > 1$, the process is referred to as *highboost filtering*. Choosing $k < 1$ reduces the contribution of the unsharp mask.

Figure 3.48 illustrates the mechanics of unsharp masking. Part (a) is a horizontal intensity profile across a vertical ramp edge that transitions from dark to light. Figure 3.48(b) shows the blurred scan line superimposed on the original signal (shown dashed). Figure 3.48(c) is the mask, obtained by subtracting the blurred signal from the original. By comparing this result with the section of Fig. 3.44(c) corresponding to the ramp in Fig. 3.44(a), we note that the unsharp mask in Fig. 3.48(c) is similar to what we would obtain using a second-order derivative. Figure 3.48(d) is the final sharpened result, obtained by adding the mask to the original signal. The points at which a change of slope occurs in the signal are now emphasized (sharpened). Observe that negative values were added to the original. Thus, it is possible for the final result to have negative intensities if the original image has any zero values, or if the value of k is chosen large enough to emphasize the peaks of the mask to a level larger than the minimum value in the original signal. Negative values cause dark halos around edges that can become objectionable if k is too large.

EXAMPLE 3.19: Unsharp masking and highboost filtering.

Figure 3.49(a) shows a slightly blurred image of white text on a dark gray background. Figure 3.49(b) was obtained using a Gaussian smoothing filter of size 31×31 with $\sigma = 5$. As explained in our earlier discussion of Gaussian lowpass kernels, the size of the kernel we used here is the smallest odd integer no less than $6\sigma \times 6\sigma$. Figure 3.49(c) is the unsharp mask, obtained using Eq. (3-55). To obtain the im-

a b c
d e

FIGURE 3.49 (a) Original image of size 600×259 pixels. (b) Image blurred using a 31×31 Gaussian lowpass filter with $\sigma = 5$. (c) Mask. (d) Result of unsharp masking using Eq. (3-56) with $k = 1$. (e) Result of highboost filtering with $k = 4.5$.

age in Fig. 3.49(d) was used the unsharp masking expression room Eq. (3-56) with $k = 1$. This image is significantly sharper than the original image in Fig. 3.49(a), but we can do better, as we show in the following paragraph.

Figure 3.49(e) shows the result of using Eq. (3-56) with $k = 4.5$. This value is almost at the extreme of what we can use without introducing some serious artifacts in the image. The artifacts are dark, almost black, halos around the border of the characters. This is caused by the lower "blip" in Fig. 3.48(d) becoming negative, as we explained earlier. When scaling the image so that it only has positive values for display, the negative values are either clipped at 0, or scaled so that the most negative values become 0, depending on the scaling method used. In either case, the blips will be the darkest values in the image.

The results in Figs. 3.49(d) and 3.49(e) would be difficult to generate using the traditional film photography explained earlier, and it illustrates the power and versatility of image processing in the context of digital photography.

USING FIRST-ORDER DERIVATIVES FOR IMAGE SHARPENING—THE GRADIENT

We will discuss the gradient in more detail in Section 10.2. Here, we are interested only in using it for image sharpening.

First derivatives in image processing are implemented using the magnitude of the gradient. The *gradient* of an image f at coordinates (x, y) is defined as the two-dimensional column vector

$$\nabla f \equiv \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{3-57}$$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x, y).

The *magnitude* (*length*) of vector ∇f, denoted as $M(x, y)$ (the vector norm notation $\|\nabla f\|$ is also used frequently), where

$$M(x, y) = \|\nabla f\| = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \tag{3-58}$$

is the *value* at (x, y) of the *rate of change* in the direction of the gradient vector. Note that $M(x, y)$ is an image of the same size as the original, created when x and y are allowed to vary over all pixel locations in f. It is common practice to refer to this image as the *gradient image* (or simply as the *gradient* when the meaning is clear).

Because the components of the gradient vector are derivatives, they are linear operators. However, the magnitude of this vector is not, because of the squaring and square root operations. On the other hand, the partial derivatives in Eq. (3-57) are not rotation invariant, but the magnitude of the gradient vector is.

In some implementations, it is more suitable computationally to approximate the squares and square root operations by absolute values:

The vertical bars denote absolute values.

$$M(x, y) \approx |g_x| + |g_y| \tag{3-59}$$

This expression still preserves the relative changes in intensity, but the isotropic property is lost in general. However, as in the case of the Laplacian, the isotropic properties of the discrete gradient defined in the following paragraph are preserved only for a limited number of rotational increments that depend on the kernels used to approximate the derivatives. As it turns out, the most popular kernels used to approximate the gradient are isotropic at multiples of 90°. These results are independent of whether we use Eq. (3-58) or (3-59), so nothing of significance is lost in using the latter equation if we choose to do so.

As in the case of the Laplacian, we now define discrete approximations to the preceding equations, and from these formulate the appropriate kernels. In order to simplify the discussion that follows, we will use the notation in Fig. 3.50(a) to denote the intensities of pixels in a 3×3 region. For example, the value of the center point, z_5, denotes the value of $f(x, y)$ at an arbitrary location, (x, y); z_1 denotes the value of $f(x - 1, y - 1)$; and so on. As indicated in Eq. (3-48), the simplest approximations to a first-order derivative that satisfy the conditions stated at the beginning of this section are $g_x = (z_8 - z_5)$ and $g_y = (z_6 - z_5)$. Two other definitions, proposed by Roberts [1965] in the early development of digital image processing, use cross differences:

$$g_x = (z_9 - z_5) \quad \text{and} \quad g_y = (z_8 - z_6) \tag{3-60}$$

If we use Eqs. (3-58) and (3-60), we compute the gradient image as

$$M(x, y) = \left[(z_9 - z_5)^2 + (z_8 - z_6)^2\right]^{1/2} \tag{3-61}$$

a
b c
d e

FIGURE 3.50
(a) A 3×3 region of an image, where the zs are intensity values. (b)–(c) Roberts cross-gradient operators. (d)–(e) Sobel operators. All the kernel coefficients sum to zero, as expected of a derivative operator.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

-1	0
0	1

0	-1
1	0

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

If we use Eqs. (3-59) and (3-60), then

$$M(x, y) \approx |z_9 - z_5| + |z_8 - z_6| \tag{3-62}$$

where it is understood that x and y vary over the dimensions of the image in the manner described earlier. The difference terms needed in Eq. (3-60) can be implemented using the two kernels in Figs. 3.50(b) and (c). These kernels are referred to as the *Roberts cross-gradient operators*.

As noted earlier, we prefer to use kernels of odd sizes because they have a unique, (integer) center of spatial symmetry. The smallest kernels in which we are interested are of size 3×3. Approximations to g_x and g_y using a 3×3 neighborhood centered on z_5 are as follows:

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{3-63}$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{3-64}$$

These equations can be implemented using the kernels in Figs. 3.50(d) and (e). The difference between the third and first rows of the 3×3 image region approximates the partial derivative in the x-direction, and is implemented using the kernel in Fig. 3.50(d).

The difference between the third and first columns approximates the partial derivative in the y-direction and is implemented using the kernel in Fig. 3.50(e). The partial derivatives at all points in an image are obtained by convolving the image with these kernels. We then obtain the magnitude of the gradient as before. For example, substituting g_x and g_y into Eq. (3-59) yields

$$M(x,y) = \left[g_x^2 + g_y^2\right]^{\frac{1}{2}} = \Big[\left[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)\right]^2 + \left[(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)\right]^2\Big]^{\frac{1}{2}} \tag{3-65}$$

This equation indicates that the value of M at any image coordinates (x, y) is given by squaring values of the convolution of the two kernels with image f at those coordinates, summing the two results, and taking the square root.

The kernels in Figs. 3.50(d) and (e) are called the *Sobel operators*. The idea behind using a weight value of 2 in the center coefficient is to achieve some smoothing by giving more importance to the center point (we will discuss this in more detail in Chapter 10). The coefficients in all the kernels in Fig. 3.50 sum to zero, so they would give a response of zero in areas of constant intensity, as expected of a derivative operator. As noted earlier, when an image is convolved with a kernel whose coefficients sum to zero, the elements of the resulting filtered image sum to zero also, so images convolved with the kernels in Fig. 3.50 will have negative values in general.

The computations of g_x and g_y are linear operations and are implemented using convolution, as noted above. The nonlinear aspect of sharpening with the gradient is the computation of $M(x, y)$ involving squaring and square roots, or the use of absolute values, all of which are nonlinear operations. These operations are performed after the linear process (convolution) that yields g_x and g_y.

EXAMPLE 3.20: Using the gradient for edge enhancement.

The gradient is used frequently in industrial inspection, either to aid humans in the detection of defects or, what is more common, as a preprocessing step in automated inspection. We will have more to say about this in Chapter 10. However, it will be instructive now to consider a simple example to illustrate how the gradient can be used to enhance defects and eliminate slowly changing background features.

Figure 3.51(a) is an optical image of a contact lens, illuminated by a lighting arrangement designed to highlight imperfections, such as the two edge defects in the lens boundary seen at 4 and 5 o'clock. Figure 3.51(b) shows the gradient obtained using Eq. (3-65) with the two Sobel kernels in Figs. 3.50(d) and (e). The edge defects are also quite visible in this image, but with the added advantage that constant or slowly varying shades of gray have been eliminated, thus simplifying considerably the computational task required for automated inspection. The gradient can be used also to highlight small specs that may not be readily visible in a gray-scale image (specs like these can be foreign matter, air pockets in a supporting solution, or miniscule imperfections in the lens). The ability to enhance small discontinuities in an otherwise flat gray field is another important feature of the gradient.

a b

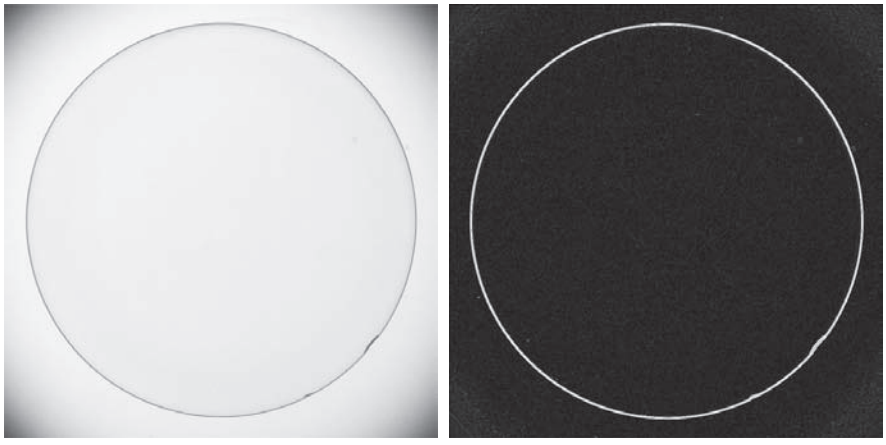


FIGURE 3.51
(a) Image of a contact lens (note defects on the boundary at 4 and 5 o'clock). (b) Sobel gradient. (Original image courtesy of Perceptics Corporation.)

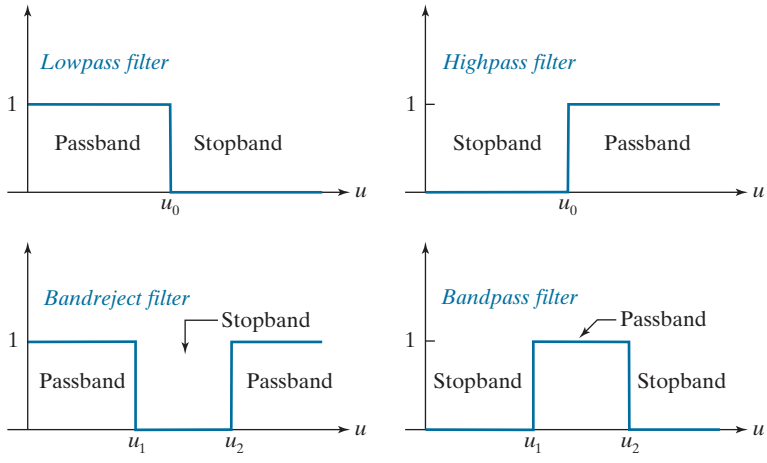
3.7 HIGHPASS, BANDREJECT, AND BANDPASS FILTERS FROM LOWPASS FILTERS

Spatial and frequency-domain linear filters are classified into four broad categories: lowpass and highpass filters, which we introduced in Sections 3.5 and 3.6, and *bandpass* and *bandreject* filters, which we introduce in this section. We mentioned at the beginning of Section 3.5 that the other three types of filters can be constructed from lowpass filters. In this section we explore methods for doing this. Also, we illustrate the third approach discussed at the end of Section 3.4 for obtaining spatial filter kernels. That is, we use a filter design software package to generate 1-D filter functions. Then, we use these to generate 2-D separable filters functions either via Eq.(3-42), or by rotating the 1-D functions about their centers to generate 2-D kernels. The rotated versions are approximations of circularly symmetric (isotropic) functions.

Figure 3.52(a) shows the transfer function of a 1-D ideal lowpass filter in the frequency domain [this is the same as Fig. 3.32(a)]. We know from earlier discussions in this chapter that lowpass filters attenuate or delete high frequencies, while passing low frequencies. A *highpass filter* behaves in exactly the opposite manner. As Fig. 3.52(b) shows, a highpass filter deletes or attenuates all frequencies below a cut-off value, u_0, and passes all frequencies above this value. Comparing Figs. 3.52(a) and (b), we see that a highpass filter transfer function is obtained by subtracting a lowpass function from 1. This operation is in the frequency domain. As you know from Section 3.4, a constant in the frequency domain is an impulse in the spatial domain. Thus, we obtain a highpass filter kernel in the spatial domain by subtracting a lowpass filter kernel from a unit impulse with the same center as the kernel. An image filtered with this kernel is the same as an image obtained by subtracting a lowpass-filtered image from the original image. The unsharp mask defined by Eq. (3-55) is precisely this operation. Therefore, Eqs. (3-54) and (3-56) implement equivalent operations (see Problem 3.42).

Recall from the discussion of Eq. (3-33) that a unit impulse is an array of 0's with a single 1.

Figure 3.52(c) shows the transfer function of a bandreject filter. This transfer function can be constructed from the sum of a lowpass and a highpass function with



a b
c d

FIGURE 3.52 Transfer functions of ideal 1-D filters in the frequency domain (u denotes frequency). (a) Lowpass filter. (b) Highpass filter. (c) Bandreject filter. (d) Bandpass filter. (As before, we show only positive frequencies for simplicity.)

different cut-off frequencies (the highpass function can be constructed from a *different* lowpass function). The bandpass filter transfer function in Fig. 3.52(d) can be obtained by subtracting the bandreject function from 1 (a unit impulse in the spatial domain). Bandreject filters are also referred to as *notch* filters, but the latter tend to be more locally oriented, as we will show in Chapter 4. Table 3.7 summarizes the preceding discussion.

The key point in Fig. 3.52 and Table 3.7 is that all transfer functions shown can be obtained starting with a lowpass filter transfer function. This is important. It is important also to realize that we arrived at this conclusion via simple graphical interpretations in the frequency domain. To arrive at the same conclusion based on convolution in the spatial domain would be a much harder task.

EXAMPLE 3.21: Lowpass, highpass, bandreject, and bandpass filtering.

In this example we illustrate how we can start with a 1-D lowpass filter transfer function generated using a software package, and then use that transfer function to generate spatial filter kernels based on the concepts introduced in this section. We also examine the spatial filtering properties of these kernels.

TABLE 3.7 Summary of the four principal spatial filter types expressed in terms of lowpass filters. The centers of the unit impulse and the filter kernels coincide.

Filter type	Spatial kernel in terms of lowpass kernel, *lp*
Lowpass	$lp(x,y)$
Highpass	$hp(x,y) = \delta(x,y) - lp(x,y)$
Bandreject	$br(x,y) = lp_1(x,y) + hp_2(x,y)$ $= lp_1(x,y) + [\delta(x,y) - lp_2(x,y)]$
Bandpass	$bp(x,y) = \delta(x,y) - br(x,y)$ $= \delta(x,y) - [lp_1(x,y) + [\delta(x,y) - lp_2(x,y)]]$

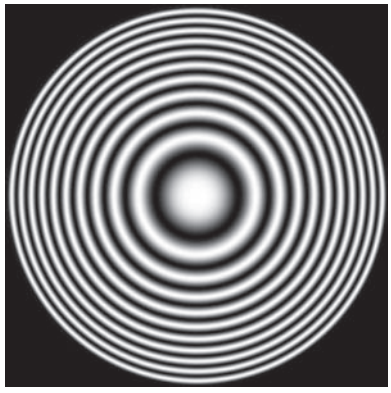


FIGURE 3.53
A zone plate image of size 597×597 pixels.

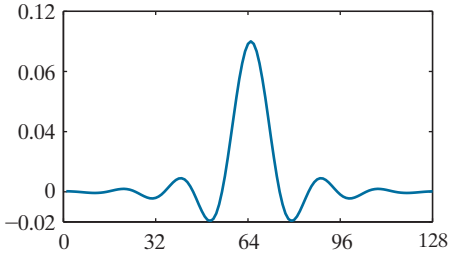
Figure 3.53 shows a so-called *zone plate* image that is used frequently for testing the characteristics of filtering approaches. There are various versions of zone plates; the one in Fig. 3.53 was generated using the equation

$$z(x, y) = \frac{1}{2}\left[1 + \cos\left(x^2 + y^2\right)\right] \tag{3-66}$$

with x and y varying in the range $[-8.2, 8.2]$, in increments of 0.0275. This resulted in an image of size 597×597 pixels. The bordering black region was generated by setting to 0 all pixels with distance greater than 8.2 from the image center. The key characteristic of a zone plate is that its spatial frequency increases as a function of distance from the center, as you can see by noting that the rings get narrower the further they are from the center. This property makes a zone plate an ideal image for illustrating the behavior of the four filter types just discussed.

Figure 3.54(a) shows a 1-D, 128-element spatial lowpass filter function designed using MATLAB [compare with Fig. 3.32(b)]. As discussed earlier, we can use this 1-D function to construct a 2-D, separable lowpass filter kernel based on Eq. (3-42), or we can rotate it about its center to generate a 2-D, isotropic kernel. The kernel in Fig. 3.54(b) was obtained using the latter approach. Figures 3.55(a) and (b) are the results of filtering the image in Fig. 3.53 with the separable and isotropic kernels, respectively. Both filters passed the low frequencies of the zone plate while attenuating the high frequencies significantly. Observe, however, that the separable filter kernel produced a "squarish" (non-radially symmetric) result in the passed frequencies. This is a consequence of filtering the image in perpendicular directions with a separable kernel that is not isotropic. Using the isotropic kernel yielded a result that is uniform in all radial directions. This is as expected, because both the filter and the image are isotropic.

a b



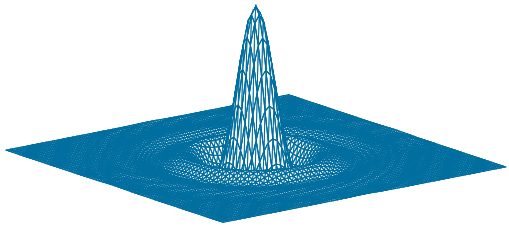


FIGURE 3.54
(a) A 1-D spatial lowpass filter function. (b) 2-D kernel obtained by rotating the 1-D profile about its center.

a b

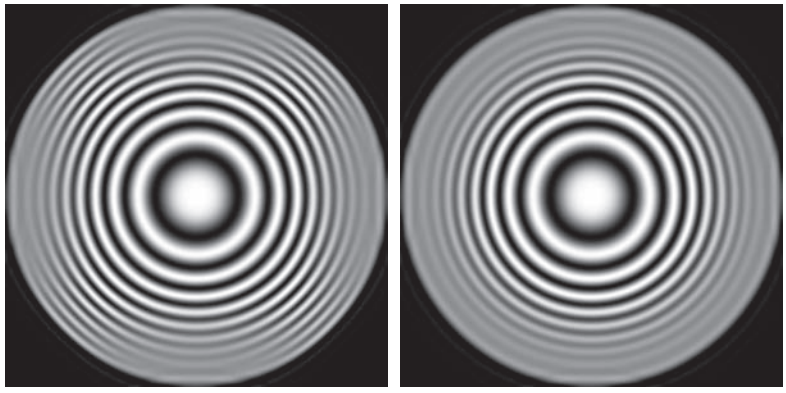


FIGURE 3.55
(a) Zone plate image filtered with a separable lowpass kernel. (b) Image filtered with the isotropic lowpass kernel in Fig. 3.54(b).

Figure 3.56 shows the results of filtering the zone plate with the four filters described in Table 3.7. We used the 2-D lowpass kernel in Fig. 3.54(b) as the basis for the highpass filter, and similar lowpass kernels for the bandreject filter. Figure 3.56(a) is the same as Fig. 3.55(b), which we repeat for convenience. Figure 3.56(b) is the highpass-filtered result. Note how effectively the low frequencies were filtered out. As is true of highpass-filtered images, the black areas were caused by negative values being clipped at 0 by the display. Figure 3.56(c) shows the same image scaled using Eqs. (2-31) and (2-32). Here we see clearly that only high frequencies were passed by the filter. Because the highpass kernel was constructed using the same lowpass kernel that we used to generate Fig. 3.56(a), it is evident by comparing the two results that the highpass filter passed the frequencies that were attenuated by the lowpass filter.

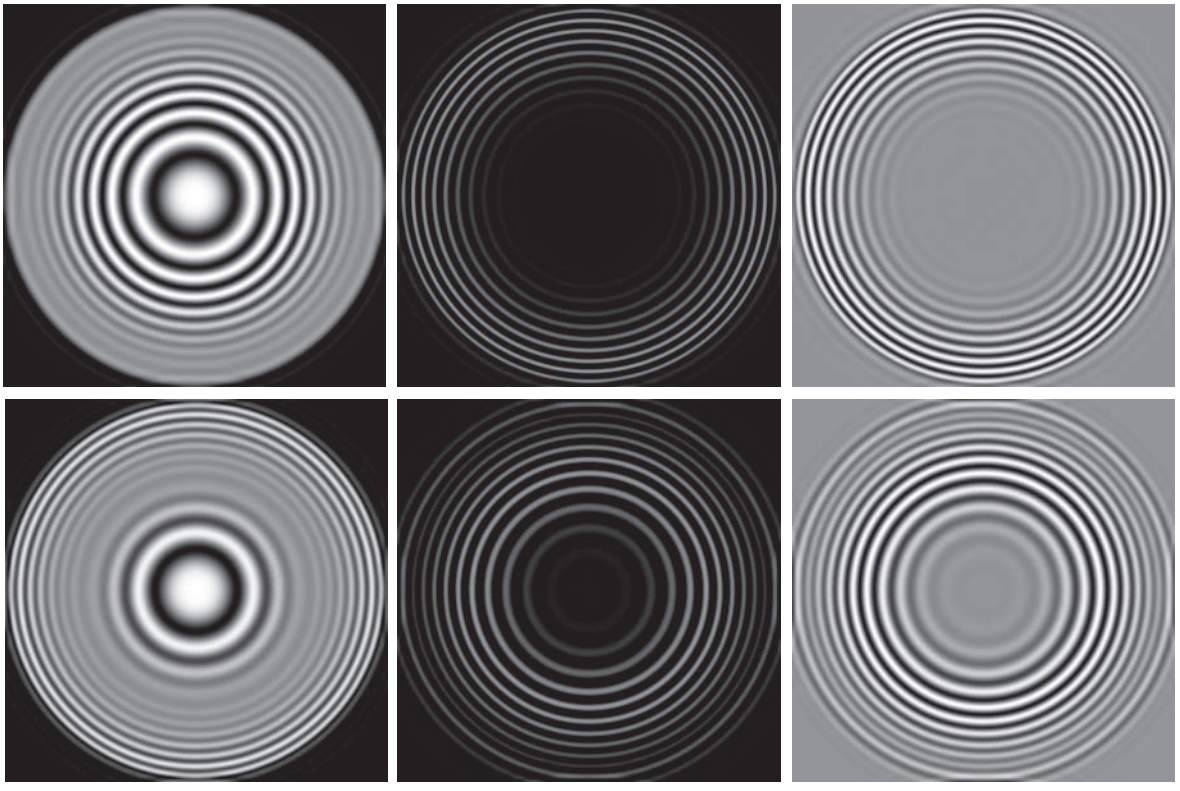
Figure 3.56(d) shows the bandreject-filtered image, in which the attenuation of the mid-band of frequencies is evident. Finally, Fig. 33.56(e) shows the result of bandpass filtering. This image also has negative values, so it is shown scaled in Fig. 3.56(f). Because the bandpass kernel was constructed by subtracting the bandreject kernel from a unit impulse, we see that the bandpass filter passed the frequencies that were attenuated by the bandreject filter. We will give additional examples of bandpass and bandreject filtering in Chapter 4.

3.8 COMBINING SPATIAL ENHANCEMENT METHODS

With a few exceptions, such as combining blurring with thresholding (Fig. 3.41), we have focused attention thus far on individual spatial-domain processing approaches. Frequently, a given task will require application of several complementary techniques in order to achieve an acceptable result. In this section, we illustrate how to combine several of the approaches developed thus far in this chapter to address a difficult image enhancement task.

The image in Fig. 3.57(a) is a nuclear whole body bone scan, used to detect diseases such as bone infections and tumors. Our objective is to enhance this image by sharpening it and by bringing out more of the skeletal detail. The narrow dynamic range of the intensity levels and high noise content make this image difficult to enhance. The strategy we will follow is to utilize the Laplacian to highlight fine detail, and the gradient to enhance prominent edges. For reasons that will be explained shortly, a smoothed version of the gradient image will be used to mask the Laplacian

In this context, masking refers to multiplying two images, as in Fig. 2.34. This is not be confused with the mask used in unsharp masking.



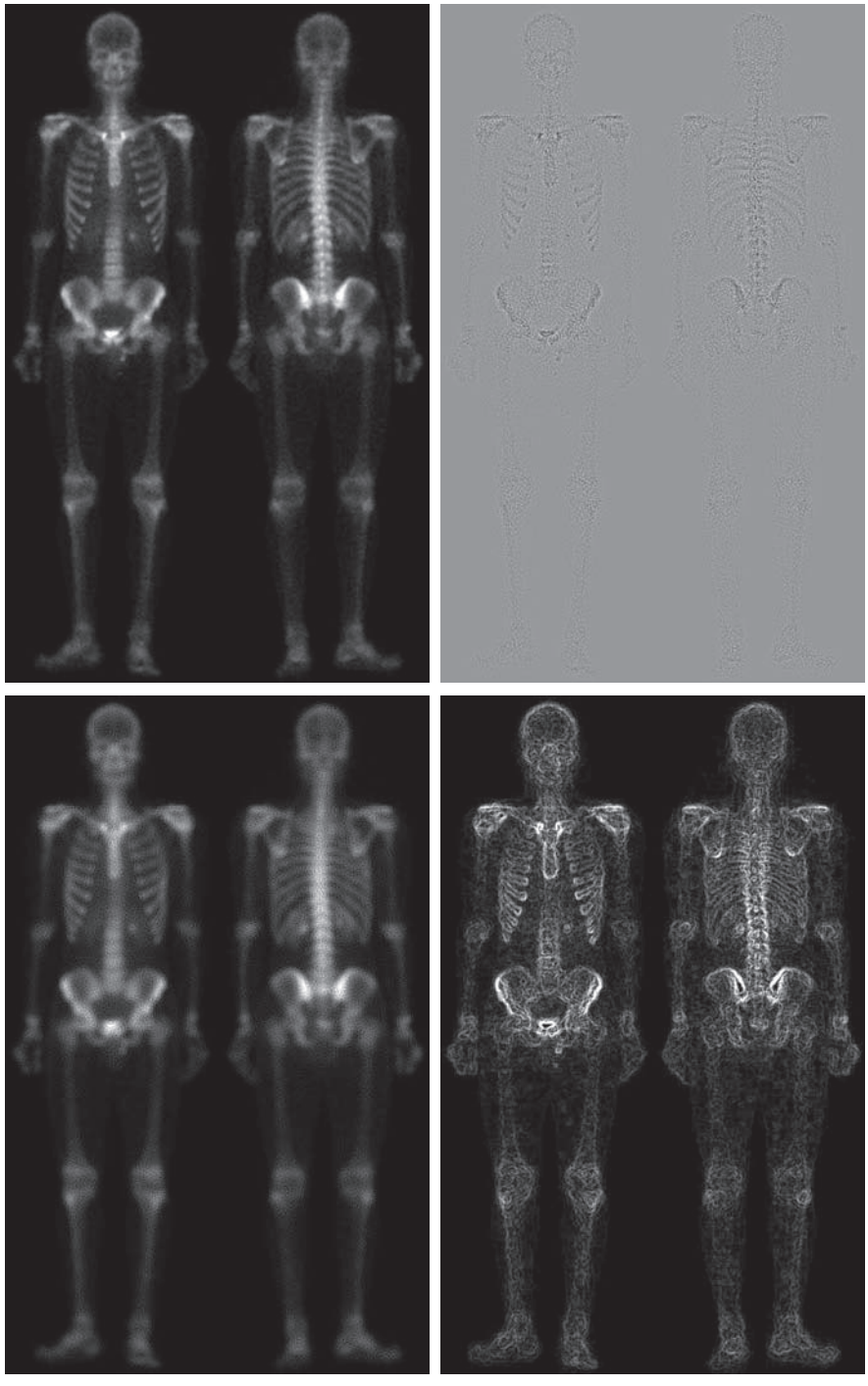
a b c
d e f

FIGURE 3.56
Spatial filtering of the zone plate image. (a) Lowpass result; this is the same as Fig. 3.55(b). (b) Highpass result. (c) Image (b) with intensities scaled. (d) Bandreject result. (e) Bandpass result. (f) Image (e) with intensities scaled.

image. Finally, we will attempt to increase the dynamic range of the intensity levels by using an intensity transformation.

Figure 3.57(b) shows the Laplacian of the original image, obtained using the kernel in Fig. 3.45(d). This image was scaled (for display only) using the same technique as in Fig. 3.47. We can obtain a sharpened image at this point simply by adding Figs. 3.57(a) and (b), according to Eq. (3-54). Just by looking at the noise level in Fig. 3.57(b), we would expect a rather noisy sharpened image if we added Figs. 3.57(a) and (b). This is confirmed by the result in Fig. 3.57(c). One way that comes immediately to mind to reduce the noise is to use a median filter. However, median filtering is an aggressive nonlinear process capable of removing image features. This is unacceptable in medical image processing.

An alternate approach is to use a mask formed from a smoothed version of the gradient of the original image. The approach is based on the properties of first- and



a b
c d

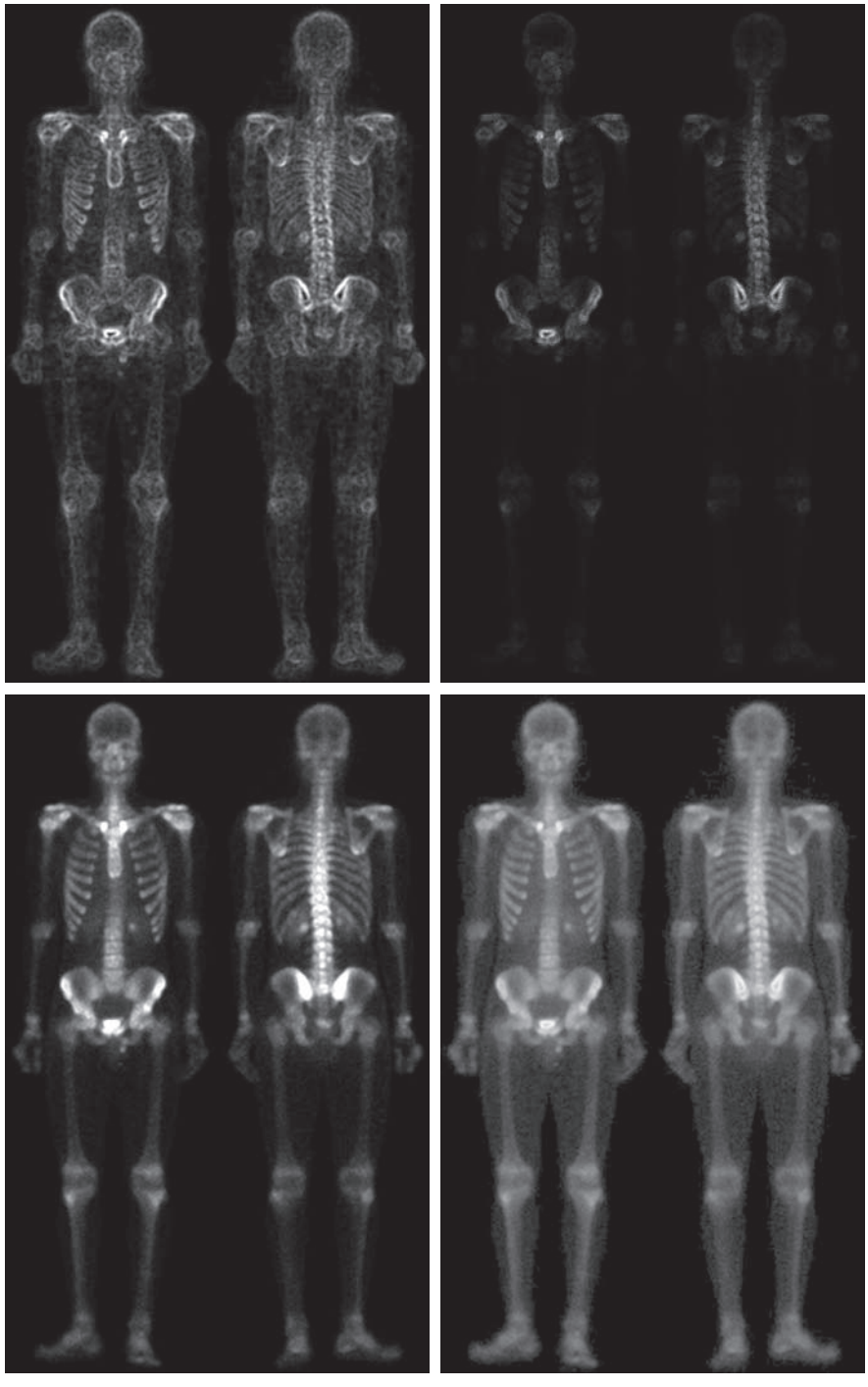
FIGURE 3.57
(a) Image of whole body bone scan. (b) Laplacian of (a). (c) Sharpened image obtained by adding (a) and (b). (d) Sobel gradient of image (a). (Original image courtesy of G.E. Medical Systems.)

second-order derivatives we discussed when explaining Fig. 3.44. The Laplacian, is a second-order derivative operator and has the definite advantage that it is superior for enhancing fine detail. However, this causes it to produce noisier results than the gradient. This noise is most objectionable in smooth areas, where it tends to be more visible. The gradient has a stronger response in areas of significant intensity transitions (ramps and steps) than does the Laplacian. The response of the gradient to noise and fine detail is lower than the Laplacian's and can be lowered further by smoothing the gradient with a lowpass filter. The idea, then, is to smooth the gradient and multiply it by the Laplacian image. In this context, we may view the smoothed gradient as a mask image. The product will preserve details in the strong areas, while reducing noise in the relatively flat areas. This process can be interpreted roughly as combining the best features of the Laplacian and the gradient. The result is added to the original to obtain a final sharpened image.

Figure 3.57(d) shows the Sobel gradient of the original image, computed using Eq. (3-59). Components g_x and g_y were obtained using the kernels in Figs. 3.50(d) and (e), respectively. As expected, the edges are much more dominant in this image than in the Laplacian image. The smoothed gradient image in Fig. 3.57(e) was obtained by using a box filter of size 5×5. The fact that Figs. 3.57(d) and (e) are much brighter than Fig. 3.57(b) is further evidence that the gradient of an image with significant edge content has values that are higher in general than in a Laplacian image.

Figure 3.57(f) shows the product of the Laplacian and smoothed gradient image. Note the dominance of the strong edges and the relative lack of visible noise, which is the reason for masking the Laplacian with a smoothed gradient image. Adding the product image to the original resulted in the sharpened image in Fig. 3.57(g). The increase in sharpness of detail in this image over the original is evident in most parts of the image, including the ribs, spinal cord, pelvis, and skull. This type of improvement would not have been possible by using the Laplacian or the gradient alone.

The sharpening procedure just discussed did not affect in an appreciable way the dynamic range of the intensity levels in an image. Thus, the final step in our enhancement task is to increase the dynamic range of the sharpened image. As we discussed in some detail in Sections 3.2 and 3.3, there are several intensity transformation functions that can accomplish this objective. Histogram processing is not a good approach on images whose histograms are characterized by dark and light components, which is the case here. The dark characteristics of the images with which we are dealing lend themselves much better to a power-law transformation. Because we wish to spread the intensity levels, the value of γ in Eq. (3-5) has to be less than 1. After a few trials with this equation, we arrived at the result in Fig. 3.57(h), obtained with $\gamma = 0.5$ and $c = 1$. Comparing this image with Fig. 3.57(g), we note that significant new detail is visible in Fig. 3.57(h). The areas around the wrists, hands, ankles, and feet are good examples of this. The skeletal bone structure also is much more pronounced, including the arm and leg bones. Note the faint definition of the outline of the body, and of body tissue. Bringing out detail of this nature by expanding the dynamic range of the intensity levels also enhanced noise, but Fig. 3.57(h) is a significant visual improvement over the original image.



e f
g h

FIGURE 3.57
(Continued)
(e) Sobel image smoothed with a 5×5 box filter. (f) Mask image formed by the product of (b) and (e). (g) Sharpened image obtained by the adding images (a) and (f). (h) Final result obtained by applying a power-law transformation to (g). Compare images (g) and (h) with (a). (Original image courtesy of G.E. Medical Systems.)

Summary, References, and Further Reading

The material in this chapter is representative of current techniques used for intensity transformations and spatial filtering. The topics were selected for their value as fundamental material that would serve as a foundation in an evolving field. Although most of the examples used in this chapter deal with image enhancement, the techniques presented are perfectly general, and you will encounter many of them again throughout the remaining chapters in contexts unrelated to enhancement.

The material in Section 3.1 is from Gonzalez [1986]. For additional reading on the material in Section 3.2, see Schowengerdt [2006] and Poyton [1996]. Early references on histogram processing (Section 3.3) are Gonzalez and Fittes [1977], and Woods and Gonzalez [1981]. Stark [2000] gives some interesting generalizations of histogram equalization for adaptive contrast enhancement.

For complementary reading on linear spatial filtering (Sections 3.4-3.7), see Jain [1989], Rosenfeld and Kak [1982], Schowengerdt [2006], Castleman [1996], and Umbaugh [2010]. For an interesting approach for generating Gaussian kernels with integer coefficients see Padfield [2011]. The book by Pitas and Venetsanopoulos [1990] is a good source for additional reading on median and other nonlinear spatial filters.

For details on the software aspects of many of the examples in this chapter, see Gonzalez, Woods, and Eddins [2009].

Problems

Solutions to the problems marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

3.1 Give a single intensity transformation function for spreading the intensities of an image so the lowest intensity is 0 and the highest is $L-1$.

3.2 Do the following:

(a)* Give a continuous function for implementing the contrast stretching transformation in Fig. 3.2(a). In addition to m, your function must include a parameter, E, for controlling the slope of the function as it transitions from low to high intensity values. Your function should be normalized so that its minimum and maximum values are 0 and 1, respectively.

(b) Sketch a family of transformations as a function of parameter E, for a fixed value $m = L/2$, where L is the number of intensity levels in the image..

3.3 Do the following:

(a)* Propose a *set* of intensity-slicing transformation functions capable of producing all the individual bit planes of an 8-bit monochrome image. For example, applying to an image a transformation function with the property $T(r) = 0$ if r is 0 or even, and $T(r) = 1$ if r is odd, produces an image of the least significant bit plane (see Fig. 3.13). (*Hint:* Use an 8-bit truth table to determine the form of each transformation function.)

(b) How many intensity transformation functions would there be for 16-bit images?

(c) Is the basic approach in (a) limited to images in which the number of intensity levels is an integer power of 2, or is the method general for any number of *integer* intensity levels?

(d) If the method is general, how would it be different from your solution in (a)?

3.4 Do the following:

(a) Propose a method for extracting the bit planes of an image based on converting the value of its pixels to binary.

(b) Find all the bit planes of the following 4-bit image:

0	1	8	6
2	2	1	1
1	15	14	12
3	6	9	10

3.5 In general:

(a)* What effect would setting to zero the lower-

order bit planes have on the histogram of an image?

(b) What would be the effect on the histogram if we set to zero the higher-order bit planes instead?

3.6 Explain why the discrete histogram equalization technique does not yield a flat histogram in general.

3.7 Suppose that a digital image is subjected to histogram equalization. Show that a second pass of histogram equalization (on the histogram-equalized image) will produce exactly the same result as the first pass.

3.8 Assuming continuous values, show by an example that it is possible to have a case in which the transformation function given in Eq. (3-11) satisfies conditions (a) and (b) discussed in Section 3.3, but its inverse may fail condition (a′).

3.9 Do the following:

(a) Show that the discrete transformation function given in Eq. (3-15) for histogram equalization satisfies conditions (a) and (b) stated at the beginning of Section 3.3.

(b)* Show that the inverse discrete transformation in Eq. (3-16) satisfies conditions (a′) and (b) in Section 3.3 *only if* none of the intensity levels r_k, $k = 0,1,2,\ldots,L-1$, are missing in the original image.

3.10 Two images, $f(x,y)$ and $g(x,y)$ have unnormalized histograms h_f and h_g. Give the conditions (on the values of the pixels in f and g) under which you can determine the histograms of images formed as follows:

(a)* $f(x,y) + g(x,y)$

(b) $f(x,y) - g(x,y)$

(c) $f(x,y) \times g(x,y)$

(d) $f(x,y) \div g(x,y)$

Show how the histograms would be formed in each case. The arithmetic operations are elementwise operations, as defined in Section 2.6.

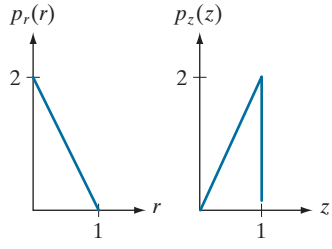
3.11 Assume continuous intensity values, and suppose that the intensity values of an image have the PDF $p_r(r) = 2r/(L-1)^2$ for $0 \le r \le L-1$, and $p_r(r) = 0$ for other values of r.

(a)* Find the transformation function that will map the input intensity values, r, into values, s, of a histogram-equalized image.

(b)* Find the transformation function that (when applied to the histogram-equalized intensities, s) will produce an image whose intensity PDF is $p_z(z) = 3z^2/(L-1)^3$ for $0 \le z \le L-1$ and $p_z(z) = 0$ for other values of z.

(c) Express the transformation function from (b) directly in terms of r, the intensities of the input image.

3.12 An image with intensities in the range [0,1] has the PDF, $p_r(r)$, shown in the following figure. It is desired to transform the intensity levels of this image so that they will have the specified $p_z(z)$ shown in the figure. Assume continuous quantities, and find the transformation (expressed in terms of r and z) that will accomplish this.



3.13* In Fig. 3.25(b), the transformation function labeled (2) $[G^{-1}(s_k)$ from Eq. (3-23)] is the mirror image of (1) $[G(z_q)$ in Eq. (3-21)] about a line joining the two end points. Does this property always hold for these two transformation functions? Explain.

3.14* The local histogram processing method discussed in Section 3.3 requires that a histogram be computed at each neighborhood location. Propose a method for updating the histogram from one neighborhood to the next, rather than computing a new histogram each time.

3.15 What is the behavior of Eq. (3-35) when $a = b = 0$? Explain.

3.16 You are given a computer chip that is capable of performing linear filtering in real time, but you are not told whether the chip performs correlation or convolution. Give the details of a test you would perform to determine which of the two operations the chip performs.

3.17* We mentioned in Section 3.4 that to perform con-

volution we rotate the kernel by 180°. The rotation is "built" into Eq. (3-35). Figure 3.28 corresponds to correlation. Draw the part of the figure enclosed by the large ellipse, but with w rotated 180°. Expand Eq. (3-35) for a general 3×3 kernel and show that the result of your expansion corresponds to your figure. This shows graphically that convolution and correlation differ by the rotation of the kernel.

3.18 You are given the following kernel and image:

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a)* Give a sketch of the area encircled by the large ellipse in Fig. 3.28 when the kernel is centered at point (2,3) (2nd row, 3rd col) of the image shown above. Show specific values of w and f.

(b)* Compute the convolution $w \star f$ using the *minimum* zero padding needed. Show the details of your computations when the kernel is centered on point (2,3) of f; and then show the final full convolution result.

(c) Repeat (b), but for correlation, $w \star f$.

3.19* Prove the validity of Eqs. (3-36) and (3-37).

3.20 The kernel, w, in Problem 3.18 is separable.

(a)* By inspection, find two kernels, w_1 and w_2 so that $w = w_1 \star w_2$.

(b) Using the image in Problem 3.18, compute $w_1 \star f$ using the *minimum* zero padding (see Fig. 3.30). Show the details of your computation when the kernel is centered at point (2,3) (2nd row, 3rd col) of f and then show the full convolution.

(c) Compute the convolution of w_2 with the result from (b). Show the details of your computation when the kernel is centered at point (3,3) of the result from (b), and then show the full convolution. Compare with the result in Problem 3.18(b).

3.21 Given the following kernel and image:

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad f = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

(a) Give the convolution of the two.

(b) Does your result have a bias?

3.22 Answer the following:

(a)* If $\mathbf{v} = [1 \quad 2 \quad 1]^T$ and $\mathbf{w}^T = [2 \quad 1 \quad 1 \quad 3]$, is the kernel formed by $\mathbf{v}\mathbf{w}^T$ separable?

(b) The following kernel is separable. Find w_1 and w_2 such that $w = w_1 \star w_2$.

$$w = \begin{bmatrix} 1 & 3 & 1 \\ 2 & 6 & 2 \end{bmatrix}$$

3.23 Do the following:

(a)* Show that the Gaussian kernel, $G(s,t)$, in Eq. (3-45) is separable. (*Hint:* Read the first paragraph in the discussion of separable filter kernels in Section 3.4.)

(b) Because G is separable and circularly symmetric, it can be expressed in the form $G = \mathbf{v}\mathbf{v}^T$. Assume that the kernel form in Eq. (3-46) is used, and that the function is sampled to yield an $m \times m$ kernel. What is $\mathbf{v}$ in this case?

3.24* Show that the product of a column vector with a row vector is equivalent to the 2-D convolution of the two vectors. The vectors do not have to be of the same length. You may use a graphical approach (as in Fig. 3.30) to support the explanation of your proof.

3.25 Given K, 1-D Gaussian kernels, $g_1, g_2, \ldots, g_K$, with arbitrary means and standard deviations:

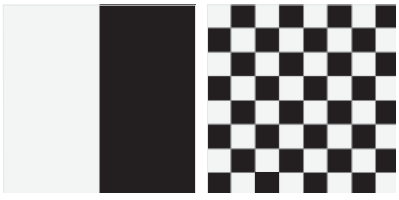
(a)* Determine what the entries in the third column of Table 3.6 would be for the product $g_1 \times g_2 \times \cdots \times g_K$.

(b) What would the fourth column look like for the convolution $g_1 \star g_2 \star \cdots \star g_K$?

(*Hint:* It is easier to work with the variance; the standard deviation is just the square root of your result.)

3.26 The two images shown in the following figure are quite different, but their histograms are the same. Suppose that each image is blurred using a 3×3 box kernel.

(a)* Would the histograms of the blurred images still be equal? Explain.



(b) If your answer is no, either sketch the two histograms or give two tables detailing the histogram components.

3.27 An image is filtered four times using a Gaussian kernel of size 3×3 with a standard deviation of 1.0. Because of the associative property of convolution, we know that equivalent results can be obtained using a single Gaussian kernel formed by convolving the individual kernels.

(a)* What is the size of the single Gaussian kernel?

(b) What is its standard deviation?

3.28 An image is filtered with three Gaussian lowpass kernels of sizes 3×3, 5×5, and 7×7, and standard deviations 1.5, 2, and 4, respectively. A composite filter, w, is formed as the convolution of these three filters.

(a)* Is the resulting filter Gaussian? Explain.

(b) What is its standard deviation?

(c) What is its size?

3.29* Discuss the limiting effect of repeatedly filtering an image with a 3×3 lowpass filter kernel. You may ignore border effects.

3.30 In Fig. 3.42(b) the corners of the estimated shading pattern appear darker or lighter than their surrounding areas. Explain the reason for this.

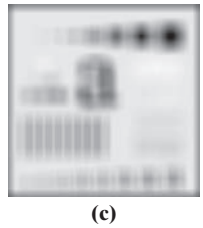
3.31* An image is filtered with a kernel whose coefficients sum to 1. Show that the sum of the pixel values in the original and filtered images is the same.

3.32 An image is filtered with a kernel whose coefficients sum to 0. Show that the sum of the pixel values in the filtered image also is 0.

3.33 A single point of light can be modeled by a digital image consisting of all 0's, with a 1 in the location of the point of light. If you view a single point of light through a defocused lens, it will appear as a fuzzy blob whose size depends on the amount by which the lens is defocused. We mentioned in Section 3.5 that filtering an image with a box kernel is a poor model for a defocused lens, and that a better approximation is obtained with a Gaussian kernel. Using the single-point-of-light analogy, explain why this is so.

3.34 In the original image used to generate the three blurred images shown, the vertical bars are 5 pixels wide, 100 pixels high, and their separation is 20 pixels. The image was blurred using square box kernels of sizes 23, 25, and 45 elements on the side, respectively. The vertical bars on the left, lower part of (a) and (c) are blurred, but a clear separation exists between them.

(a) **(b)**



(c)

However, the bars have merged in image (b), despite the fact that the kernel used to generate this image is much smaller than the kernel that produced image (c). Explain the reason for this.

3.35 Consider an application such as in Fig. 3.41, in which it is desired to eliminate objects smaller than those enclosed by a square of size $q \times q$ pixels. Suppose that we want to reduce the average

intensity of those objects to one-tenth of their original average value. In this way, their intensity will be closer to the intensity of the background and they can be eliminated by thresholding. Give the (odd) size of the smallest box kernel that will yield the desired reduction in average intensity in only one pass of the kernel over the image.

3.36 With reference to order-statistic filters (see Section 3.5):

(a)* We mentioned that isolated clusters of dark or light (with respect to the background) pixels whose area is less than one-half the area of a median filter are forced to the median value of the neighbors by the filter. Assume a filter of size $n \times n$ (n odd) and explain why this is so.

(b) Consider an image having various sets of pixel clusters. Assume that all points in a cluster are lighter or darker than the background (but not both simultaneously in the same cluster), and that the area of each cluster is less than or equal to $n^2/2$. In terms of n, under what condition would one or more of these clusters cease to be isolated in the sense described in part (a)?

3.37 Do the following:

(a)* Develop a procedure for computing the median of an $n \times n$ neighborhood.

(b) Propose a technique for updating the median as the center of the neighborhood is moved from pixel to pixel.

3.38 In a given application, a smoothing kernel is applied to input images to reduce noise, then a Laplacian kernel is applied to enhance fine details. Would the result be the same if the order of these operations is reversed?

3.39* Show that the Laplacian defined in Eq. (3-50) is isotropic (invariant to rotation). Assume continuous quantities. From Table 2.3, coordinate rotation by an angle θ is given by

$$x' = x\cos\theta - y\sin\theta \quad \text{and} \quad y' = x\sin\theta + y\cos\theta$$

where (x, y) and (x', y') are the unrotated and rotated coordinates, respectively.

3.40* You saw in Fig. 3.46 that the Laplacian with a –8 in the center yields sharper results than the one with a –4 in the center. Explain the reason why.

3.41* Give a 3×3 kernel for performing unsharp masking in a single pass through an image. Assume that the average image is obtained using a box filter of size 3×3.

3.42 Show that subtracting the Laplacian from an image gives a result that is proportional to the unsharp mask in Eq. (3-55). Use the definition for the Laplacian given in Eq. (3-53).

3.43 Do the following:

(a)* Show that the magnitude of the gradient given in Eq. (3-58) is an isotropic operation (see the statement of Problem 3.39).

(b) Show that the isotropic property is lost in general if the gradient is computed using Eq. (3-59).

3.44 Are any of the following highpass (sharpening) kernels separable? For those that are, find vectors $\mathbf{v}$ and $\mathbf{w}$ such that $\mathbf{v}\mathbf{w}^T$ equals the kernel(s).

(a) The Laplacian kernels in Figs. 3.45(a) and (b).

(b) The Roberts cross-gradient kernels shown in Figs. 3.50(b) and (c).

(c)* The Sobel kernels in Figs. 3.50(d) and (e).

3.45 In a character recognition application, text pages are reduced to binary using a thresholding transformation function of the form in Fig. 3.2(b). This is followed by a procedure that thins the characters until they become strings of binary 1's on a background of 0's. Due to noise, binarization and thinning result in broken strings of characters with gaps ranging from 1 to 3 pixels. One way to "repair" the gaps is to run a smoothing kernel over the binary image to blur it, and thus create bridges of nonzero pixels between gaps.

(a)* Give the (odd) size of the smallest box kernel capable of performing this task.

(b) After bridging the gaps, the image is thresholded to convert it back to binary form. For your answer in (a), what is the minimum value of the threshold required to accomplish this, without causing the segments to break up again?

3.46 A manufacturing company purchased an imaging system whose function is to either smooth or sharpen images. The results of using the system on the manufacturing floor have been poor, and the plant manager suspects that the system is not smoothing and sharpening images the way it should. You are hired as a consultant to determine if the system is performing these functions properly. How would you determine if the system is working correctly? (*Hint:* Study the statements of Problems 3.31 and 3.32).

3.47 A CCD TV camera is used to perform a long-term study by observing the same area 24 hours a day, for 30 days. Digital images are captured and transmitted to a central location every 5 minutes. The illumination of the scene changes from natural daylight to artificial lighting. At no time is the scene without illumination, so it is always possible to obtain an acceptable image. Because the range of illumination is such that it is always in the linear operating range of the camera, it is decided not to employ any compensating mechanisms on the camera itself. Rather, it is decided to use image processing techniques to post-process, and thus normalize, the images to the equivalent of constant illumination. Propose a method to do this. You are at liberty to use any method you wish, but state clearly all the assumptions you made in arriving at your design.