CHAPTER

24

Linear Image Processing

Linear image processing is based on the same two techniques as conventional DSP: *convolution* and *Fourier analysis*. Convolution is the more important of these two, since images have their information encoded in the spatial domain rather than the frequency domain. Linear filtering can improve images in many ways: sharpening the edges of objects, reducing random noise, correcting for unequal illumination, deconvolution to correct for blur and motion, etc. These procedures are carried out by convolving the original image with an appropriate filter kernel, producing the filtered image. A serious problem with image convolution is the enormous number of calculations that need to be performed, often resulting in unacceptably long execution times. This chapter presents strategies for designing filter kernels for various image processing tasks. Two important techniques for reducing the execution time are also described: *convolution by separability* and *FFT convolution*.

Convolution

Image convolution works in the same way as one-dimensional convolution. For instance, images can be viewed as a summation of *impulses*, i.e., scaled and shifted delta functions. Likewise, *linear systems* are characterized by how they respond to impulses; that is, by their *impulse responses*. As you should expect, the output image from a system is equal to the input image *convolved* with the system's impulse response.

The two-dimensional delta function is an image composed of all zeros, except for a single pixel at: *row* = 0, *column* = 0, which has a value of *one*. For now, assume that the row and column indexes can have both positive and negative values, such that the *one* is centered in a vast sea of zeros. When the delta function is passed through a linear system, the single nonzero point will be changed into some other two-dimensional pattern. Since the only thing that can happen to a point is that it *spreads out*, the impulse response is often called the **point spread function (PSF)** in image processing jargon.

The human eye provides an excellent example of these concepts. As described in the last chapter, the first layer of the retina transforms an image represented as a pattern of light into an image represented as a pattern of nerve impulses. The second layer of the retina *processes* this neural image and passes it to the third layer, the fibers forming the optic nerve. Imagine that the image being projected onto the retina is a very small spot of light in the center of a dark background. That is, an *impulse* is fed into the eye. Assuming that the system is linear, the image processing taking place in the retina can be determined by inspecting the image appearing at the optic nerve. In other words, we want to find the *point spread function* of the processing. We will revisit the assumption about linearity of the eye later in this chapter.

Figure 24-1 outlines this experiment. Figure (a) illustrates the impulse striking the retina while (b) shows the image appearing at the optic nerve. The middle layer of the eye passes the bright spike, but produces a circular region of increased *darkness*. The eye accomplishes this by a process known as *lateral inhibition*. If a nerve cell in the middle layer is activated, it decreases the ability of its nearby neighbors to become active. When a complete image is viewed by the eye, each point in the image contributes a scaled and shifted version of this impulse response to the image appearing at the optic nerve. In other words, the visual image is *convolved* with this PSF to produce the neural image transmitted to the brain. The obvious question is: how does convolving a viewed image with this PSF improve the ability of the eye to understand the world?

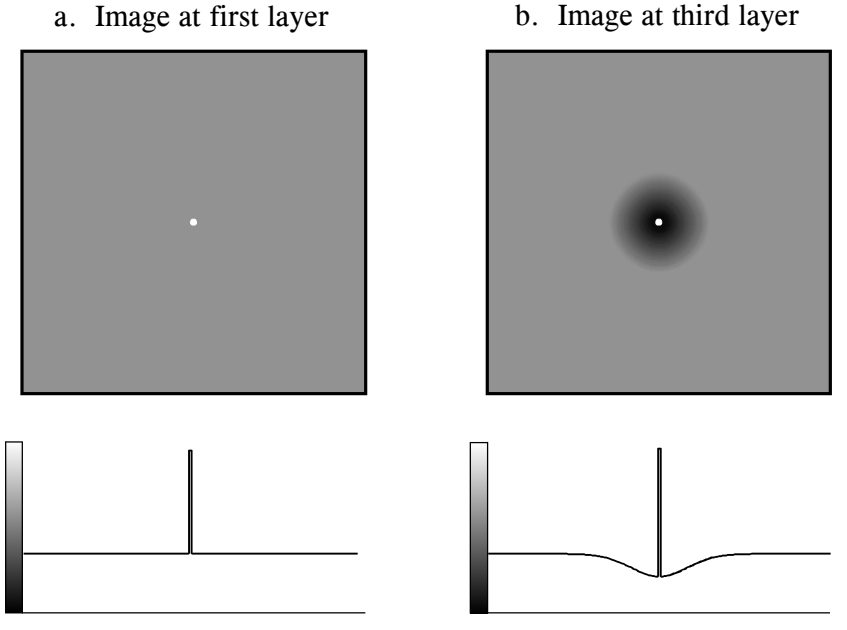


FIGURE 24-1
The PSF of the eye. The middle layer of the retina changes an impulse, shown in (a), into an impulse surrounded by a dark area, shown in (b). This point spread function enhances the edges of objects.

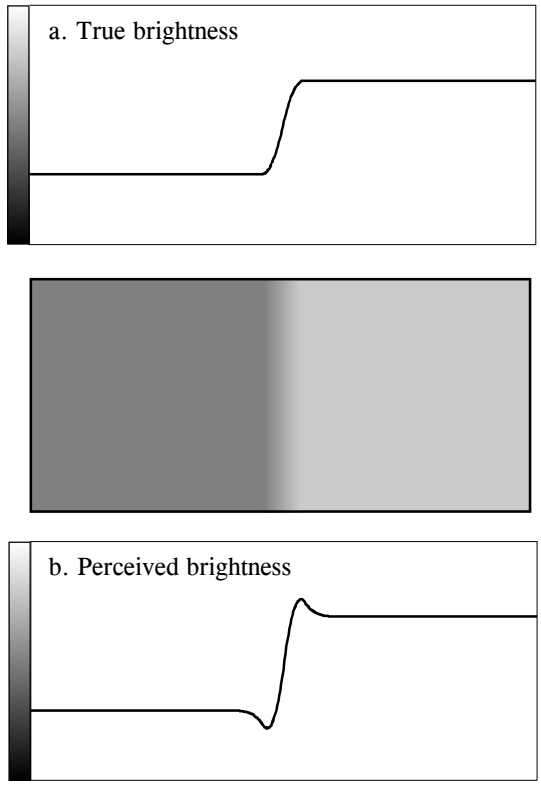


FIGURE 24-2
Mach bands. Image processing in the retina results in a slowly changing edge, as in (a), being sharpened, as in (b). This makes it easier to separate objects in the image, but produces an optical illusion called *Mach bands*. Near the edge, the overshoot makes the dark region look darker, and the light region look lighter. This produces dark and light bands that run parallel to the edge.

Humans and other animals use vision to identify nearby objects, such as enemies, food, and mates. This is done by distinguishing one region in the image from another, based on differences in brightness and color. In other words, the first step in recognizing an object is to identify its *edges*, the discontinuity that separates an object from its background. The middle layer of the retina helps this task by sharpening the edges in the viewed image. As an illustration of how this works, Fig. 24-2 shows an image that slowly changes from dark to light, producing a blurry and poorly defined edge. Figure (a) shows the intensity profile of this image, the pattern of brightness entering the eye. Figure (b) shows the brightness profile appearing on the optic nerve, the image transmitted to the brain. The processing in the retina makes the edge between the light and dark areas appear more abrupt, reinforcing that the two regions are different.

The overshoot in the edge response creates an interesting optical illusion. Next to the edge, the dark region appears to be unusually dark, and the light region appears to be unusually light. The resulting light and dark strips are called **Mach bands**, after Ernst Mach (1838-1916), an Austrian physicist who first described them.

As with one-dimensional signals, image convolution can be viewed in two ways: from the input, and from the output. From the input side, each pixel in

the input image contributes a scaled and shifted version of the point spread function to the output image. As viewed from the output side, each pixel in the output image is influenced by a group of pixels from the input signal. For one-dimensional signals, this region of influence is the impulse response flipped *left-for-right*. For image signals, it is the PSF flipped *left-for-right* and *top-for-bottom*. Since most of the PSFs used in DSP are symmetrical around the vertical and horizonal axes, these flips do nothing and can be ignored. Later in this chapter we will look at nonsymmetrical PSFs that must have the flips taken into account.

Figure 24-3 shows several common PSFs. In (a), the **pillbox** has a circular top and straight sides. For example, if the lens of a camera is not properly focused, each point in the image will be projected to a circular spot on the image sensor (look back at Fig. 23-2 and consider the effect of moving the projection screen toward or away from the lens). In other words, the pillbox is the point spread function of an out-of-focus lens.

The **Gaussian**, shown in (b), is the PSF of imaging systems limited by *random* imperfections. For instance, the image from a telescope is blurred by atmospheric turbulence, causing each point of light to become a Gaussian in the final image. Image sensors, such as the CCD and retina, are often limited by the scattering of light and/or electrons. The Central Limit Theorem dictates that a Gaussian blur results from these types of random processes.

The pillbox and Gaussian are used in image processing the same as the *moving average filter* is used with one-dimensional signals. An image convolved with these PSFs will appear blurry and have less defined edges, but will be lower in random noise. These are called **smoothing filters**, for their action in the time domain, or **low-pass filters**, for how they treat the frequency domain. The **square** PSF, shown in (c), can also be used as a smoothing filter, but it is not circularly symmetric. This results in the blurring being different in the diagonal directions compared to the vertical and horizontal. This may or may not be important, depending on the use.

The opposite of a smoothing filter is an **edge enhancement** or **high-pass filter**. The spectral inversion technique, discussed in Chapter 14, is used to change between the two. As illustrated in (d), an edge enhancement filter kernel is formed by taking the *negative* of a smoothing filter, and adding a delta function in the center. The image processing which occurs in the retina is an example of this type of filter.

Figure (e) shows the two-dimensional sinc function. One-dimensional signal processing uses the windowed-sinc to separate frequency bands. Since images do not have their information encoded in the frequency domain, the sinc function is seldom used as an imaging filter kernel, although it does find use in some theoretical problems. The sinc function can be hard to use because its tails decrease very slowly in amplitude ($1/x$), meaning it must be treated as infinitely wide. In comparison, the Gaussian's tails decrease very rapidly (e^{-x^2}) and can eventually be truncated with no ill effect.

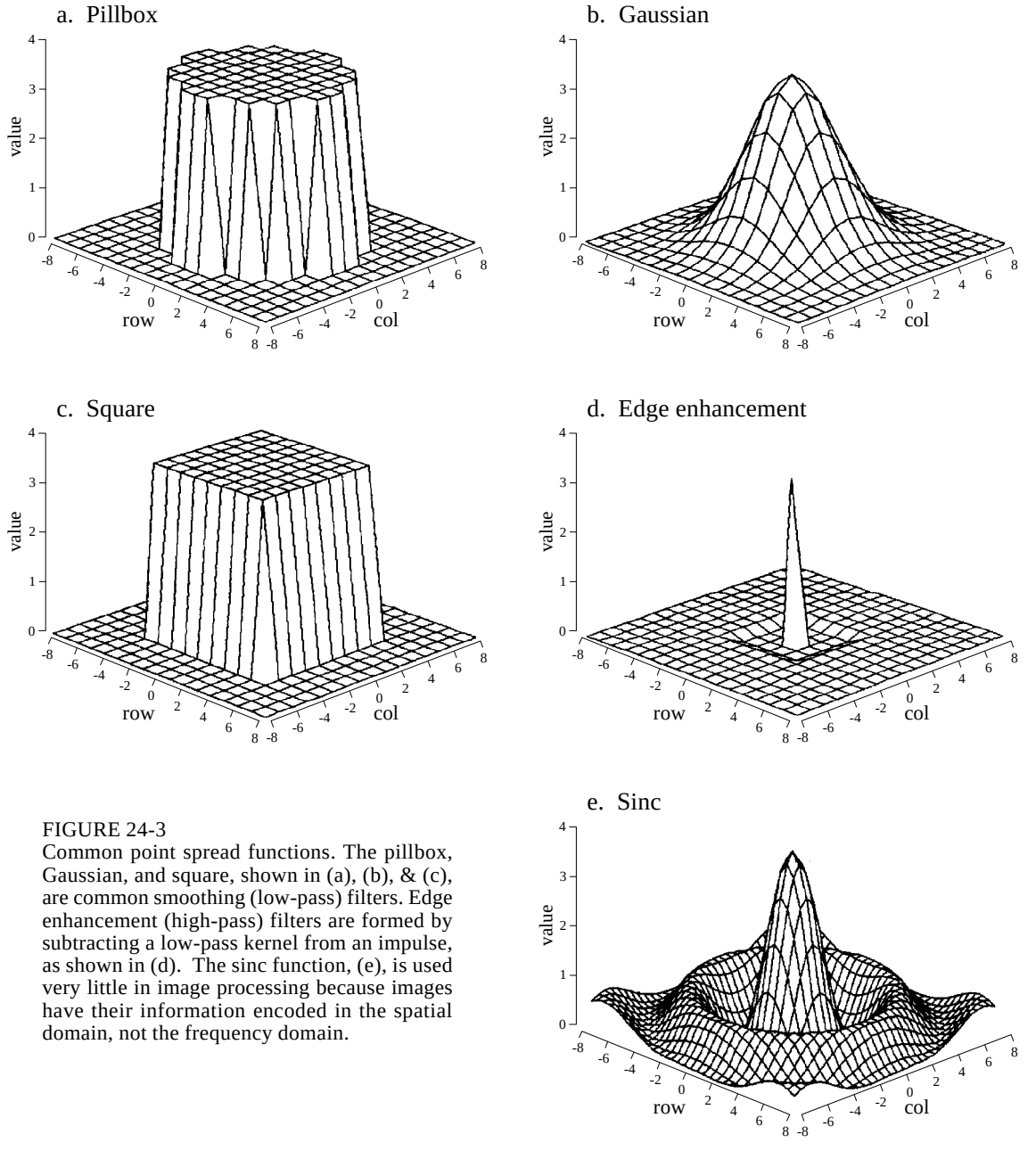


FIGURE 24-3
Common point spread functions. The pillbox, Gaussian, and square, shown in (a), (b), & (c), are common smoothing (low-pass) filters. Edge enhancement (high-pass) filters are formed by subtracting a low-pass kernel from an impulse, as shown in (d). The sinc function, (e), is used very little in image processing because images have their information encoded in the spatial domain, not the frequency domain.

All these filter kernels use *negative* indexes in the rows and columns, allowing the PSF to be centered at *row* = 0 and *column* = 0. Negative indexes are often eliminated in one-dimensional DSP by shifting the filter kernel to the right until all the nonzero samples have a positive index. This shift moves the output signal by an equal amount, which is usually of no concern. In comparison, a shift between the input and output images is generally not acceptable. Correspondingly, negative indexes are the norm for filter kernels in image processing.

A problem with image convolution is that a large number of calculations are involved. For instance, when a 512 by 512 pixel image is convolved with a 64 by 64 pixel PSF, more than a *billion* multiplications and additions are needed (i.e., $64 \times 64 \times 512 \times 512$). The long execution times can make the techniques impractical. Three approaches are used to speed things up.

The first strategy is to use a very small PSF, often only 3×3 pixels. This is carried out by looping through each sample in the output image, using optimized code to multiply and accumulate the corresponding nine pixels from the input image. A surprising amount of processing can be achieved with a mere 3×3 PSF, because it is large enough to affect the *edges* in an image.

The second strategy is used when a large PSF is needed, but its shape isn't critical. This calls for a filter kernel that is *separable*, a property that allows the image convolution to be carried out as a series of one-dimensional operations. This can improve the execution speed by *hundreds* of times.

The third strategy is FFT convolution, used when the filter kernel is large and has a specific shape. Even with the speed improvements provided by the highly efficient FFT, the execution time will be hideous. Let's take a closer look at the details of these three strategies, and examples of how they are used in image processing.

3×3 Edge Modification

Figure 24-4 shows several 3×3 operations. Figure (a) is an image acquired by an airport x-ray baggage scanner. When this image is convolved with a 3×3 delta function (a *one* surrounded by 8 zeros), the image remains unchanged. While this is not interesting by itself, it forms the baseline for the other filter kernels.

Figure (b) shows the image convolved with a 3×3 kernel consisting of a one, a negative one, and 7 zeros. This is called the **shift and subtract** operation, because a *shifted* version of the image (corresponding to the -1) is *subtracted* from the original image (corresponding to the 1). This processing produces the optical illusion that some objects are closer or farther away than the background, making a 3D or embossed effect. The brain interprets images as if the lighting is from *above*, the normal way the world presents itself. If the edges of an object are bright on the top and dark on the bottom, the object is perceived to be poking out from the background. To see another interesting effect, turn the picture upside down, and the objects will be pushed *into* the background.

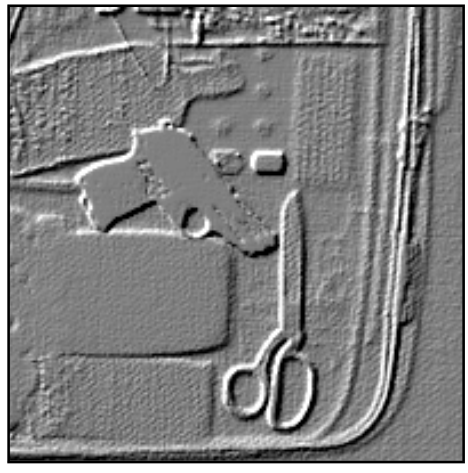
Figure (c) shows an **edge detection** PSF, and the resulting image. Every edge in the original image is transformed into narrow dark and light bands that run parallel to the original edge. Thresholding this image can isolate either the dark or light band, providing a simple algorithm for detecting the edges in an image.

a. Delta function

0	0	0
0	1	0
0	0	0

b. Shift and subtract

0	0	0
0	1	0
0	0	-1



c. Edge detection

-1/8	-1/8	-1/8
-1/8	1	-1/8
-1/8	-1/8	-1/8

d. Edge enhancement

-k/8	-k/8	-k/8
-k/8	k + 1	-k/8
-k/8	-k/8	-k/8

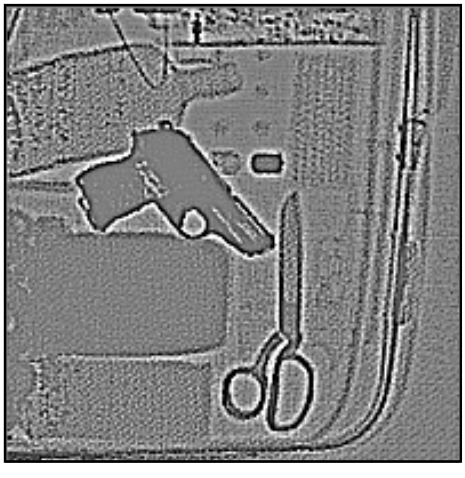


FIGURE 24-4
3×3 edge modification. The original image, (a), was acquired on an airport x-ray baggage scanner. The shift and subtract operation, shown in (b), results in a pseudo three-dimensional effect. The edge detection operator in (c) removes all contrast, leaving only the edge information. The edge enhancement filter, (d), adds various ratios of images (a) and (c), determined by the parameter, *k*. A value of $k = 2$ was used to create this image.

A common image processing technique is shown in (d): **edge enhancement**. This is sometimes called a *sharpening* operation. In (a), the objects have good contrast (an appropriate level of darkness and lightness) but very blurry edges. In (c), the objects have absolutely no contrast, but very sharp edges. The

strategy is to multiply the image with *good edges* by a constant, *k*, and add it to the image with *good contrast*. This is equivalent to convolving the original image with the 3×3 PSF shown in (d). If *k* is set to 0, the PSF becomes a delta function, and the image is left unchanged. As *k* is made larger, the image shows better edge definition. For the image in (d), a value of $k = 2$ was used: two parts of image (c) to one part of image (a). This operation mimics the eye's ability to sharpen edges, allowing objects to be more easily separated from the background.

Convolution with any of these PSFs can result in negative pixel values appearing in the final image. Even if the program can handle negative values for pixels, the image display cannot. The most common way around this is to add an offset to each of the calculated pixels, as is done in these images. An alternative is to truncate out-of-range values.

Convolution by Separability

This is a technique for fast convolution, as long as the PSF is **separable**. A PSF is said to be *separable* if it can be broken into two one-dimensional signals: a vertical and a horizontal projection. Figure 24-5 shows an example of a separable image, the square PSF. Specifically, the value of each pixel in the image is equal to the corresponding point in the horizontal projection multiplied by the corresponding point in the vertical projection. In mathematical form:

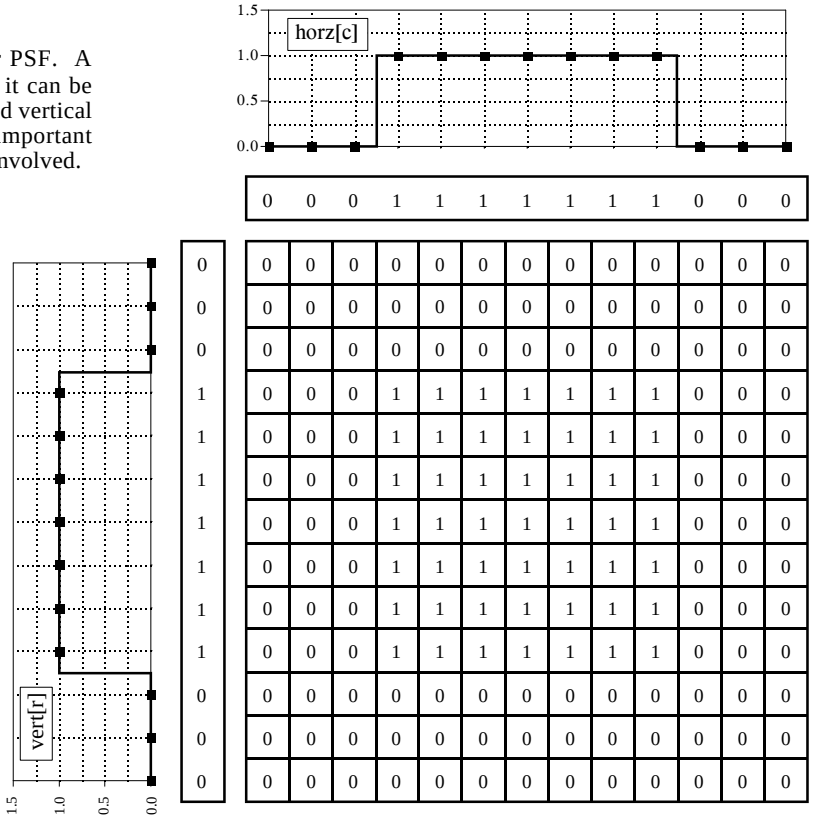
EQUATION 24-1
Image separation. An image is referred to as *separable* if it can be decomposed into horizontal and vertical projections.

$$x[r,c] = vert[r] \times horz[c]$$

where $x[r,c]$ is the two-dimensional image, and $vert[r]$ & $horz[c]$ are the one-dimensional projections. Obviously, most images do not satisfy this requirement. For example, the pillbox is not separable. There are, however, an *infinite* number of separable images. This can be understood by generating arbitrary horizontal and vertical projections, and finding the image that corresponds to them. For example, Fig. 24-6 illustrates this with profiles that are double-sided exponentials. The image that corresponds to these profiles is then found from Eq. 24-1. When displayed, the image appears as a diamond shape that exponentially decays to zero as the distance from the origin increases.

In most image processing tasks, the ideal PSF is *circularly symmetric*, such as the pillbox. Even though digitized images are usually stored and processed in the rectangular format of rows and columns, it is desired to modify the image the same in all directions. This raises the question: is there a PSF that is circularly symmetric *and* separable? The answer is, yes,

FIGURE 24-5
Separation of the rectangular PSF. A PSF is said to be *separable* if it can be decomposed into horizontal and vertical profiles. Separable PSFs are important because they can be rapidly convolved.



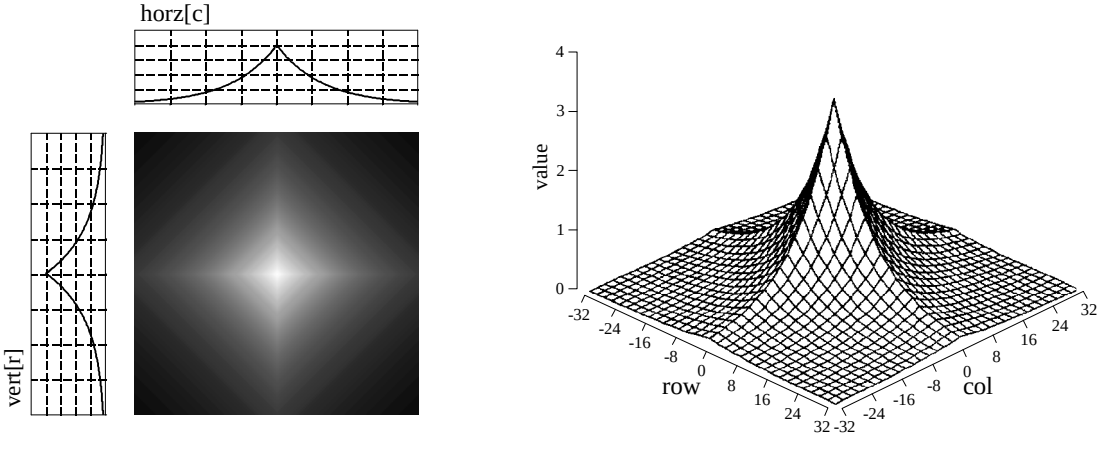


FIGURE 24-6
Creation of a separable PSF. An infinite number of separable PSFs can be generated by defining arbitrary projections, and then calculating the two-dimensional function that corresponds to them. In this example, the profiles are chosen to be double-sided exponentials, resulting in a diamond shaped PSF.

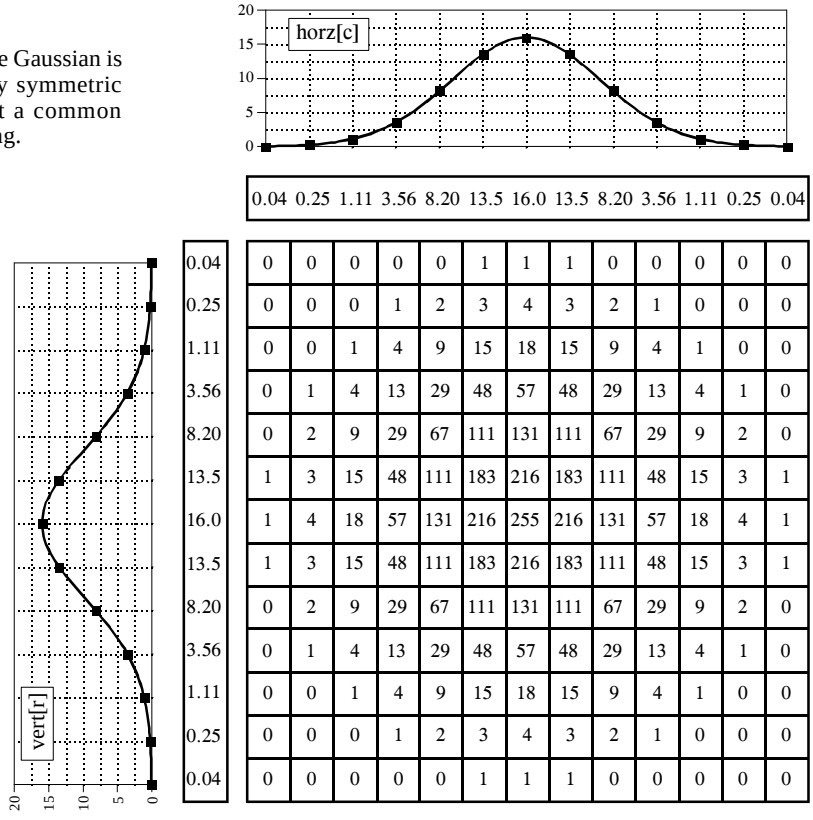


FIGURE 24-7
Separation of the Gaussian. The Gaussian is the only PSF that is circularly symmetric *and* separable. This makes it a common filter kernel in image processing.

but there is only one, the *Gaussian*. As is shown in Fig. 24-7, a two-dimensional Gaussian image has projections that are also Gaussians. The image and projection Gaussians have the *same* standard deviation.

To convolve an image with a separable filter kernel, convolve each *row* in the image with the *horizontal projection*, resulting in an intermediate image. Next, convolve each *column* of this intermediate image with the *vertical projection* of the PSF. The resulting image is identical to the direct convolution of the original image and the filter kernel. If you like, convolve the columns first and then the rows; the result is the same.

The convolution of an $N \times N$ image with an $M \times M$ filter kernel requires a time proportional to N^2M^2. In other words, each pixel in the output image depends on *all* the pixels in the filter kernel. In comparison, convolution by separability only requires a time proportional to N^2M. For filter kernels that are hundreds of pixels wide, this technique will reduce the execution time by a factor of *hundreds*.

Things can get even better. If you are willing to use a *rectangular* PSF (Fig. 24-5) or a *double-sided exponential* PSF (Fig. 24-6), the calculations are even more efficient. This is because the one-dimensional convolutions are the *moving average* filter (Chapter 15) and the *bidirectional single pole* filter

(Chapter 19), respectively. Both of these one-dimensional filters can be rapidly carried out by recursion. This results in an image convolution time proportional to only N^2, completely independent of the size of the PSF. In other words, an image can be convolved with as large a PSF as needed, with only a few integer operations per pixel. For example, the convolution of a 512×512 image requires only a few hundred milliseconds on a personal computer. That's fast! Don't like the shape of these two filter kernels? Convolve the image with one of them *several times* to approximate a Gaussian PSF (guaranteed by the Central Limit Theorem, Chapter 7). These are great algorithms, capable of snatching success from the jaws of failure. They are well worth remembering.

Example of a Large PSF: Illumination Flattening

A common application requiring a large PSF is the enhancement of images with unequal illumination. Convolution by separability is an ideal algorithm to carry out this processing. With only a few exceptions, the images seen by the eye are formed from *reflected* light. This means that a viewed image is equal to the reflectance of the objects multiplied by the ambient illumination. Figure 24-8 shows how this works. Figure (a) represents the *reflectance* of a scene being viewed, in this case, a series of light and dark bands. Figure (b) illustrates an example illumination signal, the pattern of light falling on (a). As in the real world, the illumination slowly varies over the imaging area. Figure (c) is the image seen by the eye, equal to the reflectance image, (a), multiplied by the illumination image, (b). The regions of poor illumination are difficult to view in (c) for two reasons: they are too dark and their contrast is too low (the difference between the peaks and the valleys).

To understand how this relates to the problem of every day vision, imagine you are looking at two identically dressed men. One of them is standing in the bright sunlight, while the other is standing in the shade of a nearby tree. The percent of the incident light reflected from both men is the same. For instance, their faces might reflect 80% of the incident light, their gray shirts 40% and their dark pants 5%. The problem is, the illumination of the two might be, say, ten times different. This makes the image of the man in the shade ten times darker than the person in the sunlight, and the contrast (between the face, shirt, and pants) ten times less.

The goal of the image processing is to *flatten* the illumination component in the acquired image. In other words, we want the final image to be representative of the objects' reflectance, not the lighting conditions. In terms of Fig. 24-8, given (c), find (a). This is a nonlinear filtering problem, since the component images were combined by multiplication, not addition. While this separation cannot be performed perfectly, the improvement can be dramatic.

To start, we will convolve image (c) with a large PSF, one-fifth the size of the entire image. The goal is to eliminate the sharp features in (c), resulting

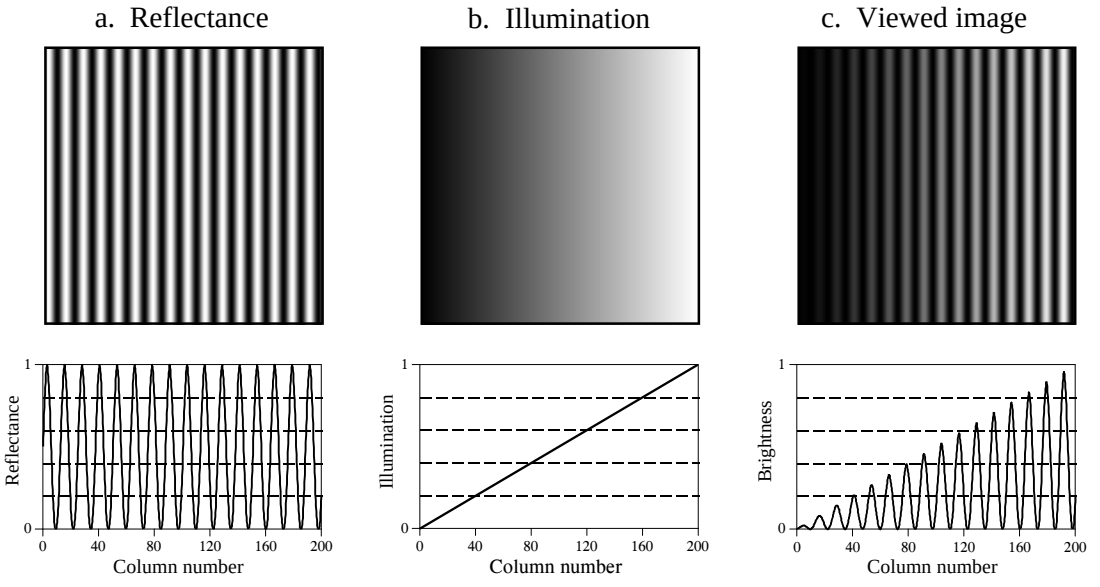


FIGURE 24-8
Model of image formation. A viewed image, (c), results from the multiplication of an illumination pattern, (b), by a reflectance pattern, (a). The goal of the image processing is to modify (c) to make it look more like (a). This is performed in Figs. (d), (e) and (f) on the opposite page.

in an approximation to the original illumination signal, (b). This is where convolution by separability is used. The exact shape of the PSF is not important, only that it is much wider than the features in the reflectance image. Figure (d) is the result, using a Gaussian filter kernel.

Since a smoothing filter provides an estimate of the illumination image, we will use an edge enhancement filter to find the reflectance image. That is, image (c) will be convolved with a filter kernel consisting of a delta function minus a Gaussian. To reduce execution time, this is done by subtracting the smoothed image in (d) from the original image in (c). Figure (e) shows the result. It doesn't work! While the dark areas have been properly lightened, the contrast in these areas is still terrible.

Linear filtering performs poorly in this application because the reflectance and illumination signals were original combined by multiplication, not addition. Linear filtering cannot correctly separate signals combined by a nonlinear operation. To separate these signals, they must be *unmultiplied*. In other words, the original image should be *divided* by the smoothed image, as is shown in (f). This corrects the brightness and restores the contrast to the proper level.

This procedure of dividing the images is closely related to **homomorphic processing**, previously described in Chapter 22. Homomorphic processing is a way of handling signals combined through a nonlinear operation. The strategy is to change the nonlinear problem into a linear one, through an appropriate mathematical operation. When two signals are combined by

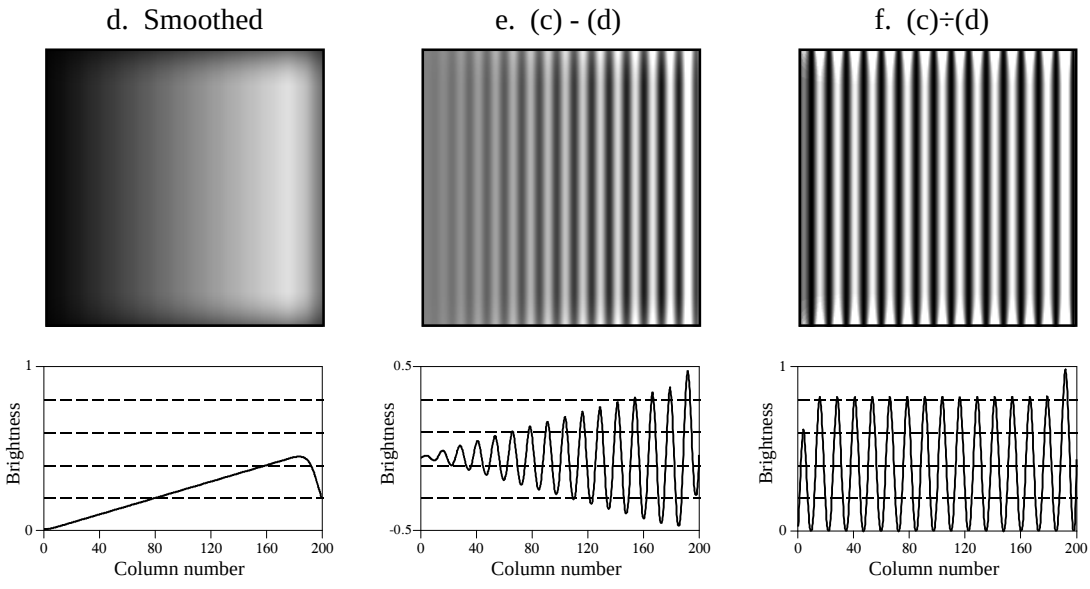


FIGURE 24-8 (continued)
Figure (d) is a smoothed version of (c), used as an approximation to the illumination signal. Figure (e) shows an approximation to the reflectance image, created by *subtracting* the smoothed image from the viewed image. A better approximation is shown in (f), obtained by the nonlinear process of *dividing* the two images.

multiplication, homomorphic processing starts by taking the *logarithm* of the acquired signal. With the identity: $\log(a \times b) = \log(a) + \log(b)$, the problem of separating *multiplied* signals is converted into the problem of separating *added* signals. For example, after taking the logarithm of the image in (c), a linear high-pass filter can be used to isolate the logarithm of the reflectance image. As before, the quickest way to carry out the high-pass filter is to subtract a smoothed version of the image. The antilogarithm (exponent) is then used to undo the logarithm, resulting in the desired approximation to the reflectance image.

Which is better, dividing or going along the homomorphic path? They are nearly the same, since taking the logarithm and subtracting is *equal* to dividing. The only difference is the approximation used for the illumination image. One method uses a smoothed version of the acquired image, while the other uses a smoothed version of the logarithm of the acquired image.

This technique of flattening the illumination signal is so useful it has been incorporated into the neural structure of the eye. The processing in the middle layer of the retina was previously described as an edge enhancement or high-pass filter. While this is true, it doesn't tell the whole story. The first layer of the eye is nonlinear, approximately taking the *logarithm* of the incoming image. This makes the eye a homomorphic processor. Just as described above, the logarithm followed by a linear edge enhancement filter flattens the illumination component, allowing the eye to see under poor lighting conditions. Another interesting use of homomorphic processing

occurs in photography. The density (darkness) of a negative is equal to the logarithm of the brightness in the final photograph. This means that any manipulation of the negative during the development stage is a type of homomorphic processing.

Before leaving this example, there is a nuisance that needs to be mentioned. As discussed in Chapter 6, when an N point signal is convolved with an M point filter kernel, the resulting signal is $N+M-1$ points long. Likewise, when an $M \times M$ image is convolved with an $N \times N$ filter kernel, the result is an $(M+N-1) \times (M+N-1)$ image. The problem is, it is often difficult to manage a changing image size. For instance, the allocated memory must change, the video display must be adjusted, the array indexing may need altering, etc. The common way around this is to *ignore* it; if we start with a 512×512 image, we want to end up with a 512×512 image. The pixels that do not fit within the original boundaries are discarded.

While this keeps the image size the same, it doesn't solve the whole problem; these is still the *boundary condition* for convolution. For example, imagine trying to calculate the pixel at the upper-right corner of (d). This is done by centering the Gaussian PSF on the upper-right corner of (c). Each pixel in (c) is then multiplied by the corresponding pixel in the overlaying PSF, and the products are added. The problem is, three-quarters of the PSF lies outside the defined image. The easiest approach is to assign the undefined pixels a value of zero. This is how (d) was created, accounting for the dark band around the perimeter of the image. That is, the brightness smoothly decreases to the pixel value of zero, exterior to the defined image.

Fortunately, this dark region around the boarder can be corrected (although it hasn't been in this example). This is done by dividing each pixel in (d) by a correction factor. The correction factor is the fraction of the PSF that was immersed in the input image when the pixel was calculated. That is, to correct an individual pixel in (d), imagine that the PSF is centered on the corresponding pixel in (c). For example, the upper-right pixel in (c) results from only 25% of the PSF overlapping the input image. Therefore, correct this pixel in (d) by dividing it by a factor of 0.25. This means that the pixels in the center of (d) will not be changed, but the dark pixels around the perimeter will be brightened. To find the correction factors, imagine convolving the filter kernel with an image having all the pixel values equal to *one*. The pixels in the resulting image are the correction factors needed to eliminate the edge effect.

Fourier Image Analysis

Fourier analysis is used in image processing in much the same way as with one-dimensional signals. However, images do not have their information encoded in the frequency domain, making the techniques much less useful. For example, when the Fourier transform is taken of an *audio* signal, the confusing time domain waveform is converted into an easy to understand frequency

spectrum. In comparison, taking the Fourier transform of an image converts the straightforward information in the spatial domain into a scrambled form in the frequency domain. In short, don't expect the Fourier transform to help you understand the information encoded in images.

Likewise, don't look to the frequency domain for filter design. The basic feature in images is the edge, the line separating one *object* or *region* from another *object* or *region*. Since an edge is composed of a wide range of frequency components, trying to modify an image by manipulating the frequency spectrum is generally not productive. Image filters are normally designed in the spatial domain, where the information is encoded in its simplest form. Think in terms of *smoothing* and *edge enhancement* operations (the spatial domain) rather than *high-pass* and *low-pass* filters (the frequency domain).

In spite of this, Fourier image analysis does have several useful properties. For instance, *convolution* in the spatial domain corresponds to *multiplication* in the frequency domain. This is important because multiplication is a simpler mathematical operation than convolution. As with one-dimensional signals, this property enables FFT convolution and various deconvolution techniques. Another useful property of the frequency domain is the *Fourier Slice Theorem*, the relationship between an image and its projections (the image viewed from its sides). This is the basis of *computed tomography*, an x-ray imaging technique widely used medicine and industry.

The frequency spectrum of an image can be calculated in several ways, but the FFT method presented here is the only one that is practical. The original image must be composed of N rows by N columns, where N is a power of two, i.e., 256, 512, 1024, etc. If the size of the original image is not a power of two, pixels with a value of zero are added to make it the correct size. We will call the two-dimensional array that holds the image the **real array**. In addition, another array of the same size is needed, which we will call the **imaginary array**.

The recipe for calculating the Fourier transform of an image is quite simple: take the one-dimensional FFT of each of the rows, followed by the one-dimensional FFT of each of the columns. Specifically, start by taking the FFT of the N pixel values in row 0 of the real array. The real part of the FFT's output is placed back into row 0 of the real array, while the imaginary part of the FFT's output is placed into row 0 of the imaginary array. After repeating this procedure on rows 1 through $N-1$, both the real and imaginary arrays contain an intermediate image. Next, the procedure is repeated on each of the *columns* of the intermediate data. Take the N pixel values from column 0 of the real array, *and* the N pixel values from column 0 of the imaginary array, and calculate the FFT. The real part of the FFT's output is placed back into column 0 of the real array, while the imaginary part of the FFT's output is placed back into column 0 of the imaginary array. After this is repeated on columns 1 through $N-1$, both arrays have been overwritten with the image's frequency spectrum.

Since the vertical and horizontal directions are equivalent in an image, this algorithm can also be carried out by transforming the columns first and then transforming the rows. Regardless of the order used, the result is the same. From the way that the FFT keeps track of the data, the amplitudes of the low frequency components end up being at the corners of the two-dimensional spectrum, while the high frequencies are at the center. The inverse Fourier transform of an image is calculated by taking the inverse FFT of each row, followed by the inverse FFT of each column (or vice versa).

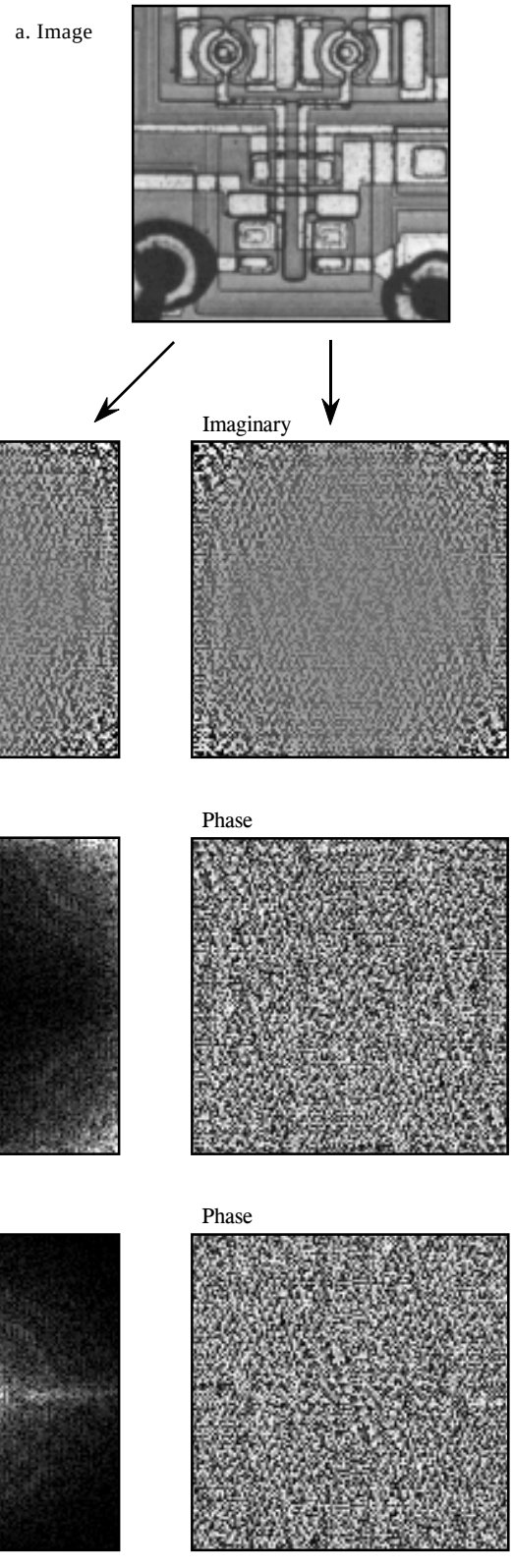
Figure 24-9 shows an example Fourier transform of an image. Figure (a) is the original image, a microscopic view of the input stage of a 741 op amp integrated circuit. Figure (b) shows the real and imaginary parts of the frequency spectrum of this image. Since the frequency domain can contain negative pixel values, the grayscale values of these images are offset such that negative values are dark, zero is gray, and positive values are light. The low-frequency components in an image are normally much larger in amplitude than the high-frequency components. This accounts for the very bright and dark pixels at the four corners of (b). Other than this, the spectra of *typical* images have no discernable order, appearing random. Of course, images can be *contrived* to have any spectrum you desire.

As shown in (c), the polar form of an image spectrum is only slightly easier to understand. The low-frequencies in the magnitude have large positive values (the white corners), while the high-frequencies have small positive values (the black center). The phase looks the same at low and high-frequencies, appearing to run randomly between $-\pi$ and π radians.

Figure (d) shows an alternative way of displaying an image spectrum. Since the spatial domain contains a *discrete* signal, the frequency domain is *periodic*. In other words, the frequency domain arrays are duplicated an infinite number of times to the left, right, top and bottom. For instance, imagine a tile wall, with each tile being the $N \times N$ magnitude shown in (c). Figure (d) is also an $N \times N$ section of this tile wall, but it straddles four tiles; the center of the image being where the four tiles touch. In other words, (c) is the same image as (d), except it has been shifted $N/2$ pixels horizontally (either left or right) and $N/2$ pixels vertically (either up or down) in the periodic frequency spectrum. This brings the bright pixels at the four corners of (c) together in the center of (d).

Figure 24-10 illustrates how the two-dimensional frequency domain is organized (with the low-frequencies placed at the corners). Row $N/2$ and column $N/2$ break the frequency spectrum into four quadrants. For the real part and the magnitude, the upper-right quadrant is a mirror image of the lower-left, while the upper-left is a mirror image of the lower-right. This symmetry also holds for the imaginary part and the phase, except that the values of the mirrored pixels are opposite in sign. In other words, every point in the frequency spectrum has a matching point placed symmetrically on the other side of the center of the image (row $N/2$ and column $N/2$). One of the points is the *positive* frequency, while the other is the matching

FIGURE 24-9
Frequency spectrum of an image. The example image, shown in (a), is a microscopic photograph of the silicon surface of an integrated circuit. The frequency spectrum can be displayed as the real and imaginary parts, shown in (b), or as the magnitude and phase, shown in (c). Figures (b) & (c) are displayed with the low-frequencies at the corners and the high-frequencies at the center. Since the frequency domain is periodic, the display can be rearranged to reverse these positions. This is shown in (d), where the magnitude and phase are displayed with the low-frequencies located at the center and the high-frequencies at the corners.



b. Frequency spectrum displayed in rectangular form (as the real and imaginary parts).

c. Frequency spectrum displayed in polar form (as the magnitude and phase).

d. Frequency spectrum displayed in polar form, with the spectrum shifted to place zero frequency at the center.

negative frequency, as discussed in Chapter 10 for one-dimensional signals. In equation form, this symmetry is expressed as:

EQUATION 24-2
Symmetry of the two-dimensional frequency domain. These equations can be used in both formats, when the low-frequencies are displayed at the corners, or when shifting places them at the center. In polar form, the magnitude has the same symmetry as the real part, while the phase has the same symmetry as the imaginary part.

$$Re\,X[r,c] = Re\,X[N-r,N-c]$$

$$Im\,X[r,c] = -Im\,X[N-r,N-c]$$

These equations take into account that the frequency spectrum is periodic, repeating itself every N samples with indexes running from 0 to $N-1$. In other words, $X[r,N]$ should be interpreted as $X[r,0]$, $X[N,c]$ as $X[0,c]$, and $X[N,N]$ as $X[0,0]$. This symmetry makes four points in the spectrum match with *themselves*. These points are located at: $[0,0]$, $[0,N/2]$, $[N/2,0]$ and $[N/2,N/2]$.

Each pair of points in the frequency domain corresponds to a sinusoid in the spatial domain. As shown in (a), the value at $[0,0]$ corresponds to the *zero frequency* sinusoid in the spatial domain, i.e., the DC component of the image. There is only one point shown in this figure, because this is one of the points that is its own match. As shown in (b), (c), and (d), other pairs of points correspond to two-dimensional sinusoids that look like waves on the ocean. One-dimensional sinusoids have a *frequency*, *phase*, and *amplitude*. Two dimensional sinusoids also have a *direction*.

The frequency and direction of each sinusoid is determined by the location of the pair of points in the frequency domain. As shown, draw a line from each point to the *zero frequency* location at the outside corner of the quadrant that the point is in, i.e., $[0,0]$, $[0,N/2]$, $[N/2,0]$, or $[N/2,N/2]$ (as indicated by the circles in this figure). The direction of this line determines the direction of the spatial sinusoid, while its length is proportional to the frequency of the wave. This results in the low frequencies being positioned near the corners, and the high frequencies near the center.

When the spectrum is displayed with zero frequency at the center (Fig. 24-9d), the line from each pair of points is drawn to the DC value at the *center* of the image, i.e., $[N/2,N/2]$. This organization is simpler to understand and work with, since all the lines are drawn to the same point. Another advantage of placing zero at the center is that it matches the frequency spectra of *continuous* images. When the spatial domain is continuous, the frequency domain is *aperiodic*. This places zero frequency at the center, with the frequency becoming higher in all directions out to infinity. In general, the frequency spectra of discrete images are displayed with zero frequency at the center whenever people will view the data, in textbooks, journal articles, and algorithm documentation. However, most calculations are carried out with the computer arrays storing data in the other format (low-frequencies at the corners). This is because the FFT has this format.

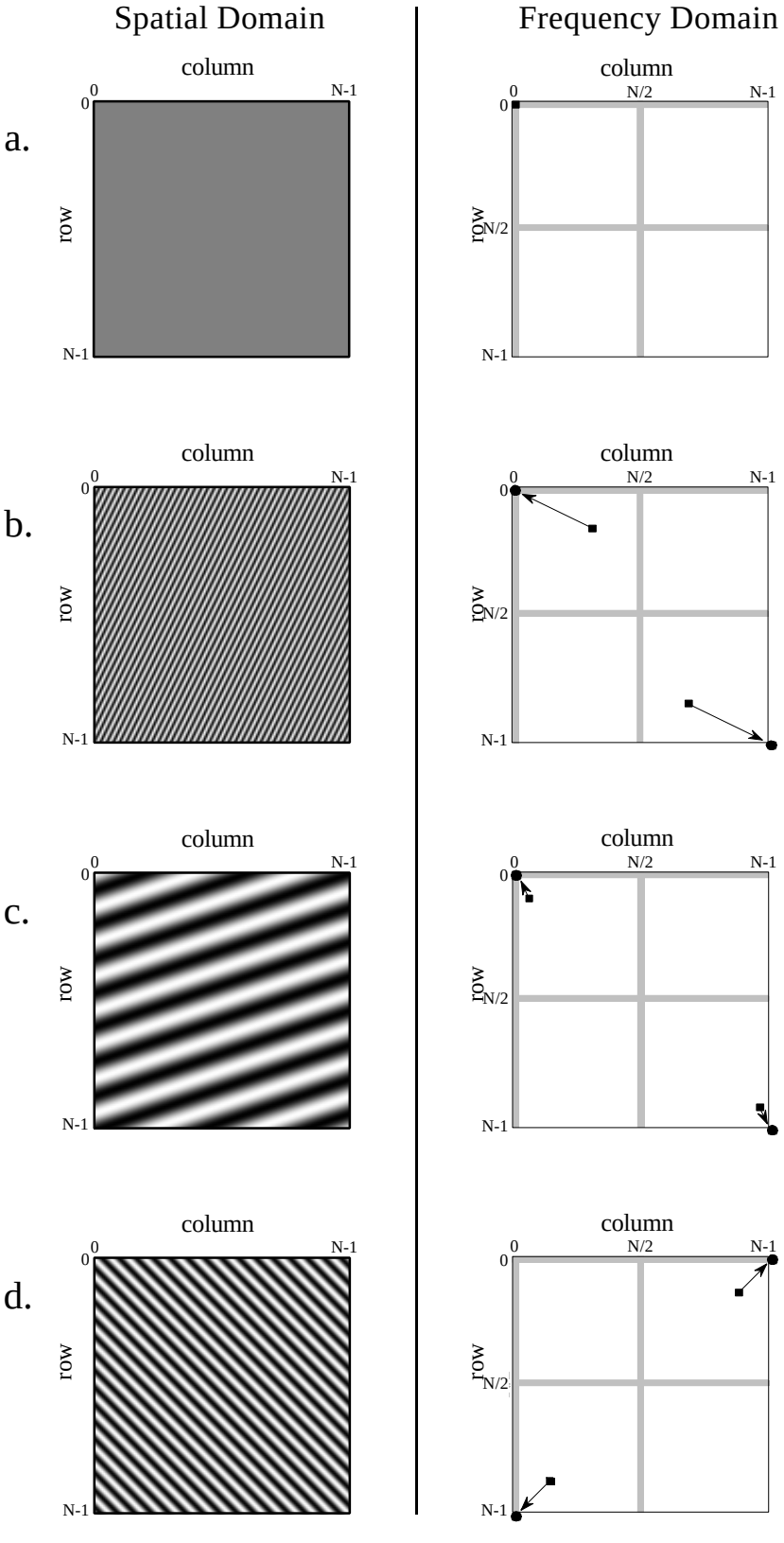


FIGURE 24-10
Two-dimensional sinusoids. Image sine and cosine waves have both a *frequency* and a *direction*. Four examples are shown here. These spectra are displayed with the low-frequencies at the corners. The circles in these spectra show the location of zero frequency.

Even with the FFT, the time required to calculate the Fourier transform is a tremendous bottleneck in image processing. For example, the Fourier transform of a 512×512 image requires several minutes on a personal computer. This is roughly 10,000 times slower than needed for real time image processing, 30 frames per second. This long execution time results from the massive amount of information contained in images. For comparison, there are about the same number of *pixels* in a typical image, as there are *words* in this book. Image processing via the frequency domain will become more popular as computers become faster. This is a twenty-first century technology; watch it emerge!

FFT Convolution

Even though the Fourier transform is slow, it is still the fastest way to convolve an image with a large filter kernel. For example, convolving a 512×512 image with a 50×50 PSF is about 20 times faster using the FFT compared with conventional convolution. Chapter 18 discusses how FFT convolution works for one-dimensional signals. The two-dimensional version is a simple extension.

We will demonstrate FFT convolution with an example, an algorithm to locate a predetermined pattern in an image. Suppose we build a system for inspecting one-dollar bills, such as might be used for printing quality control, counterfeiting detection, or payment verification in a vending machine. As shown in Fig. 24-11, a 100×100 pixel image is acquired of the bill, centered on the portrait of George Washington. The goal is to search this image for a known pattern, in this example, the 29×29 pixel image of the face. The problem is this: given an acquired image and a known pattern, what is the most effective way to locate where (or if) the pattern appears in the image? If you paid attention in Chapter 6, you know that the solution to this problem is *correlation* (a matched filter) and that it can be implemented by using *convolution*.

Before performing the actual convolution, there are two modifications that need to be made to turn the target image into a PSF. These are illustrated in Fig. 24-12. Figure (a) shows the target signal, the pattern we are trying to detect. In (b), the image has been rotated by 180°, the same as being flipped left-for-right and then flipped top-for-bottom. As discussed in Chapter 7, when performing *correlation* by using *convolution*, the target signal needs to be reversed to counteract the reversal that occurs during convolution. We will return to this issue shortly.

The second modification is a trick for improving the effectiveness of the algorithm. Rather than trying to detect the face in the original image, it is more effective to detect the *edges of the face* in the *edges of the original image*. This is because the edges are sharper than the original features, making the correlation have a sharper peak. This step isn't required, but it makes the results significantly better. In the simplest form, a 3×3 edge detection filter is applied to both the original image and the target signal

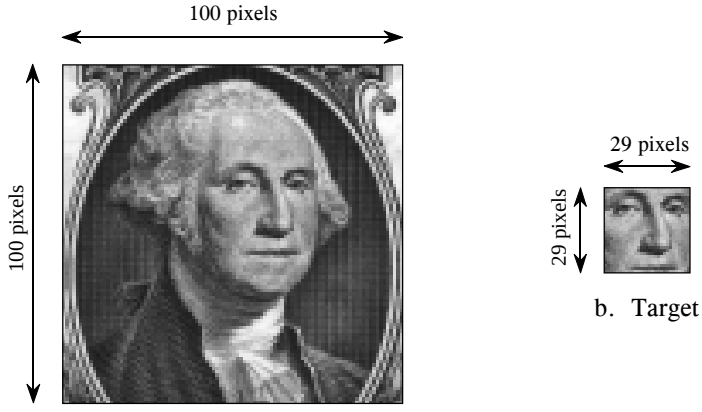


FIGURE 24-11
Target detection example. The problem is to search the 100×100 pixel image of George Washington, (a), for the target pattern, (b), the 29×29 pixel face. The optimal solution is *correlation*, which can be carried out by *convolution*.

before the correlation is performed. From the associative property of convolution, this is the same as applying the edge detection filter to the target signal *twice*, and leaving the original image alone. In actual practice, applying the edge detection 3×3 kernel only once is generally sufficient. This is how (b) is changed into (c) in Fig. 24-12. This makes (c) the PSF to be used in the convolution

Figure 24-13 illustrates the details of FFT convolution. In this example, we will convolve image (a) with image (b) to produce image (c). The fact that these images have been chosen and preprocessed to implement *correlation* is irrelevant; this is a flow diagram of *convolution.* The first step is to pad both signals being convolved with enough zeros to make them a power of two in size, and big enough to hold the final image. For instance, when images of 100×100 and 29×29 pixels are convolved, the resulting image will be 128×128 pixels. Therefore, enough zeros must be added to (a) and (b) to make them each 128×128 pixels in size. If this isn't done, circular

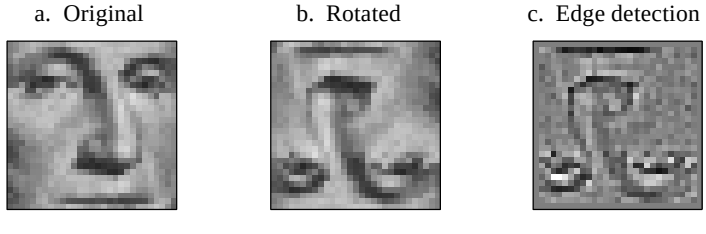


FIGURE 24-12
Development of a correlation filter kernel. The target signal is shown in (a). In (b) it is rotated by 180° to undo the rotation inherent in convolution, allowing correlation to be performed. Applying an edge detection filter results in (c), the filter kernel used for this example.

convolution takes place and the final image will be distorted. If you are having trouble understanding these concepts, go back and review Chapter 18, where the one-dimensional case is discussed in more detail.

The FFT algorithm is used to transform (a) and (b) into the frequency domain. This results in *four* 128×128 arrays, the real and imaginary parts of the two images being convolved. Multiplying the real and imaginary parts of (a) with the real and imaginary parts of (b), generates the real and imaginary parts of (c). (If you need to be reminded how this is done, see Eq. 9-1). Taking the Inverse FFT completes the algorithm by producing the final convolved image.

The value of each pixel in a correlation image is a measure of how well the target image matches the searched image *at that point*. In this particular example, the correlation image in (c) is composed of noise plus a single bright peak, indicating a good match to the target signal. Simply locating the brightest pixel in this image would specify the detected coordinates of the face. If we had not used the edge detection modification on the target signal, the peak would still be present, but much less distinct.

While correlation is a powerful tool in image processing, it suffers from a significant limitation: the target image must be exactly the same *size* and *rotational orientation* as the corresponding area in the searched image. Noise and other variations in the amplitude of each pixel are relatively unimportant, but an exact spatial match is critical. For example, this makes the method almost useless for finding enemy tanks in military reconnaissance photos, tumors in medical images, and handguns in airport baggage scans. One approach is to correlate the image *multiple times* with a variety of shapes and rotations of the target image. This works in principle, but the execution time will make you loose interest in a hurry.

A Closer Look at Image Convolution

Let's use this last example to explore two-dimensional convolution in more detail. Just as with one dimensional signals, image convolution can be viewed from either the *input side* or the *output side*. As you recall from Chapter 6, the input viewpoint is the best description of how convolution works, while the output viewpoint is how most of the mathematics and algorithms are written. You need to become comfortable with both these ways of looking at the operation.

Figure 24-14 shows the input side description of image convolution. Every pixel in the input image results in a scaled and shifted PSF being added to the output image. The output image is then calculated as the sum of all the contributing PSFs. This illustration show the contribution to the output image from the point at location [r,c] in the input image. The PSF is shifted such that pixel [0,0] in the PSF aligns with pixel [r,c] in the output image. If the PSF is defined with only positive indexes, such as in this example, the shifted PSF will be entirely to the lower-right of [r,c]. Don't

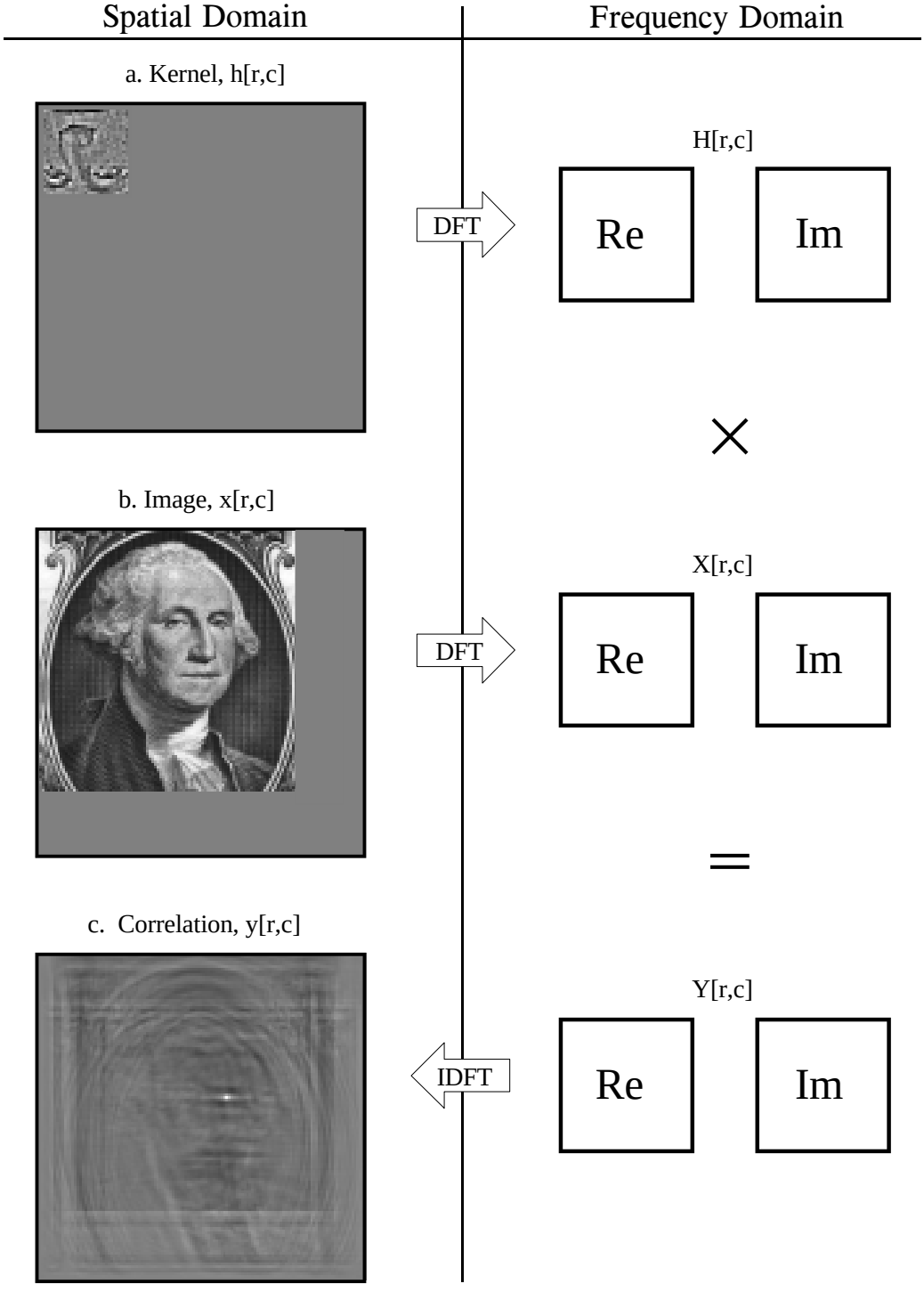


FIGURE 24-13
Flow diagram of FFT image convolution. The images in (a) and (b) are transformed into the frequency domain by using the FFT. These two frequency spectra are multiplied, and the Inverse FFT is used to move back into the spatial domain. In this example, the original images have been chosen and preprocessed to implement *correlation* through the action of convolution.

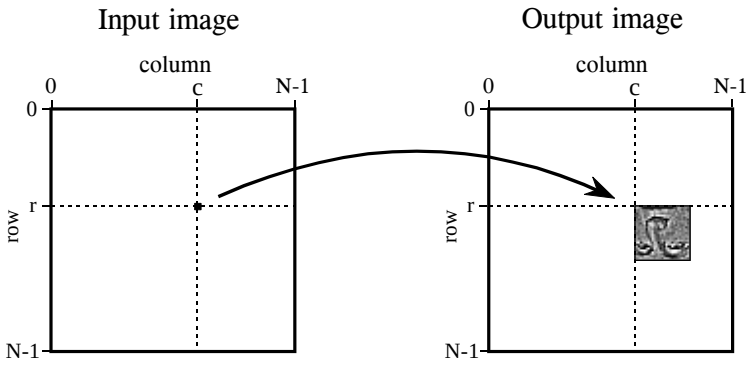


FIGURE 24-14
Image convolution viewed from the input side. Each pixel in the input image contributes a scaled and shifted PSF to the output image. The output image is the sum of these contributions. The face is inverted in this illustration because this is the PSF we are using.

be confused by the face appearing upside down in this figure; this upside down face *is* the PSF we are using in this example (Fig. 24-13a). In the input side view, there is no rotation of the PSF, it is simply shifted.

Image convolution viewed from the output is illustrated in Fig. 24-15. Each pixel in the output image, such as shown by the sample at [r,c], receives a contribution from many pixels in the input image. The PSF is rotated by 180° around pixel [0,0], and then shifted such that pixel [0,0] in the PSF is aligned with pixel [r,c] in the input image. If the PSF only uses positive indexes, it will be to the upper-left of pixel [r,c] in the input image. The value of the pixel at [r,c] in the output image is found by multiplying the pixels in the rotated PSF with the corresponding pixels in the input image, and summing the products. This procedure is given by Eq. 24-3, and in the program of Table 24-1.

EQUATION 24-3
Image convolution. The images $x[\ ,\ ]$ and $h[\ ,\ ]$ are convolved to produce image, $y[\ ,\ ]$. The size of $h[\ ,\ ]$ is $M \times M$ pixels, with the indexes running from 0 to $M-1$. In this equation, an individual pixel in the output image, $y[r,c]$, is calculated according to the output side view. The indexes j and k are used to loop through the rows and columns of $h[\ ,\ ]$ to calculate the sum-of-products.

$$y[r,c] = \sum_{k=0}^{M-1} \sum_{j=0}^{M-1} h[k,j]\, x[r-k, c-j]$$

Notice that the PSF rotation resulting from the convolution has undone the rotation made in the design of the PSF. This makes the face appear upright in Fig. 24-15, allowing it to be in the same orientation as the pattern being detected in the input image. That is, we have successfully used *convolution* to implement *correlation*. Compare Fig. 24-13c with Fig. 24-15 to see how the bright spot in the correlation image signifies that the target has been detected.

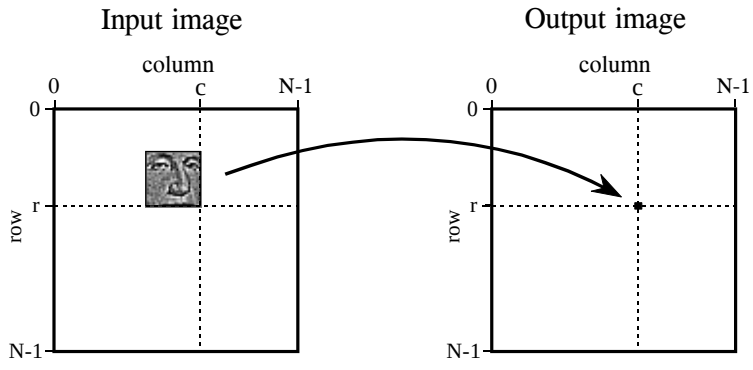


FIGURE 24-15
Image convolution viewed from the output side. Each pixel in the output signal is equal to the sum of the pixels in the rotated PSF multiplied by the corresponding pixels in the input image.

FFT convolution provides the same output image as the conventional convolution program of Table 24-1. Is the reduced execution time provided by FFT convolution really worth the additional program complexity? Let's take a closer look. Figure 24-16 shows an execution time comparison between conventional convolution using *floating point* (labeled FP), conventional convolution using *integers* (labeled INT), and FFT convolution using floating point (labeled FFT). Data for two different image sizes are presented, 512×512 and 128×128.

First, notice that the execution time required for FFT convolution does not depend on the size of the kernel, resulting in flat lines in this graph. On a 100 MHz Pentium personal computer, a 128×128 image can be convolved

```
100 CONVENTIONAL IMAGE CONVOLUTION
110 '
120 DIM X[99,99]                    'holds the input image,   100×100 pixels
130 DIM H[28,28]                    'holds the filter kernel,   29×29 pixels
140 DIM Y[127,127]                  'holds the output image, 128×128 pixels
150 '
160 FOR R% = 0 TO 127               'loop through each row and column in the output
170 FOR C% = 0 TO 127               'image calculating the pixel value via Eq. 24-3
180 '
190   Y[R%,C%] = 0                  'zero the pixel so it can be used as an accumulator
200 '
210   FOR J% = 0 TO 28              'multiply each pixel in the kernel by the corresponding
220   FOR K% = 0 TO 28              'pixel in the input image, and add to the accumulator
230     Y[R%,C%] = Y[R%,C%] + H[J%,K%] * X[R%-J%,C%-J%]
240   NEXT K%
250   NEXT J%
260 '
270 NEXT C%
280 NEXT R%
290 '
300 END
```

TABLE 24-1

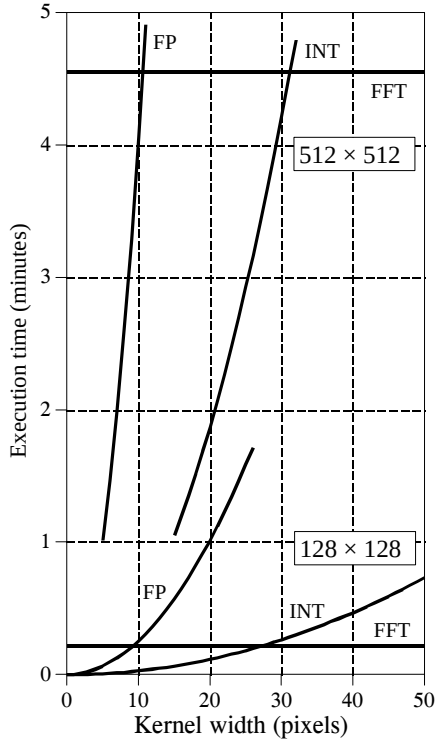


FIGURE 24-16
Execution time for image convolution. This graph shows the execution time on a 100 MHz Pentium processor for three image convolution methods: conventional convolution carried out with floating point math (FP), conventional convolution using integers (INT), and FFT convolution using floating point (FFT). The two sets of curves are for input image sizes of 512×512 and 128×128 pixels. Using FFT convolution, the time depends only on the image size, and not the size of the kernel. In contrast, conventional convolution depends on both the image and the kernel size.

in about 15 seconds using FFT convolution, while a 512×512 image requires more than 4 minutes. Adding up the number of calculations shows that the execution time for FFT convolution is proportional to $N^2 Log_2(N)$, for an $N \times N$ image. That is, a 512×512 image requires about 20 times as long as a 128×128 image.

Conventional convolution has an execution time proportional to $N^2 M^2$ for an $N \times N$ image convolved with an $M \times M$ kernel. This can be understood by examining the program in Table 24-1. In other words, the execution time for conventional convolution depends *very strongly* on the size of the kernel used. As shown in the graph, FFT convolution is faster than conventional convolution using floating point if the kernel is larger than about 10×10 pixels. In most cases, integers can be used for conventional convolution, increasing the break-even point to about 30×30 pixels. These break-even points depend slightly on the size of the image being convolved, as shown in the graph. The concept to remember is that FFT convolution is only useful for *large* filter kernels.