4

Edge, Line and Shape Detection

4.1 Introduction and Overview

The image analysis process requires us to take vast amounts of low-level pixel data and extract useful information. In this chapter, we will explore methods that contribute to the process of dividing the image into meaningful regions by detecting edges, lines, corners and geometric shapes. These shapes represent higher-level information, and these are often precursors to image segmentation, which is explored in the next chapter where we will see that edge and line detection are important steps for one category of image segmentation methods. Edge detection techniques are covered in Section 4.2, as well as metrics to measure edge detector performance. Section 4.3 introduces the Hough transform for line finding and discusses associated postprocessing methods. Section 4.4 will explore corner and shape detection, and this chapter concludes with a discussion of using the extended and generalized Hough transform for shape detection.

4.2 Edge Detection

The edge detection process is based on the idea that if the brightness levels are changing rapidly, there is the potential for an edge to exist at that point. Many different approaches have been explored to facilitate this process. The edge detection operators presented here are representative of the various types that have been developed. Many are implemented with small masks used in a manner similar to the convolution process; by this, we mean they are used to scan the image left to right and top to bottom, and at each pixel, replace the value with the result of a mathematical operation on the pixel's neighborhood. Most edge detectors are based on discrete approximations to differential operators. Differential operations measure the rate of change in a function, in this case, the image brightness function. A large change in image brightness over a short spatial distance indicates the presence of an edge. Some edge detection operators return orientation information – information about the direction of the edge – while others only return information about the magnitude of an edge at each point.

Edge detection methods are used as a first step in the line detection process. Edge detection is also used to find complex object boundaries by marking potential edge points corresponding to places in an image where rapid changes in brightness occur. After these edge points have been marked, they can be merged to form lines and object outlines. Often people are confused about the difference between an edge and a line. This is illustrated in Figure 4.2-1 where we see that an edge occurs at a point, and it is perpendicular to the line. The edge direction is defined as the direction of change; so, on a curve, it will be perpendicular to the tangent line at that point. Note that a line or curve that forms a boundary can be defined as a set of connected edge points.

With many of the edge detection operators, noise in the image can create problems. That is why it is best to preprocess the image to eliminate or at least minimize noise effects. To deal with noise effects, we must make tradeoffs between the sensitivity and the accuracy of an edge detector. For example, if the parameters are adjusted so that the edge detector is very sensitive, it will tend to find many potential edge points that are attributable to noise. If we make it less sensitive, it may miss valid edges. The parameters that we can vary include the size of the edge detection mask and the value of the gray level threshold. A larger mask or a higher gray level threshold will tend to reduce noise effects, but may result in a loss of valid edge points. The tradeoff between sensitivity and accuracy is illustrated in Figure 4.2-2.

Edge detection operators are based on the idea that edge information in an image is found by considering the relationship a pixel has with its neighbors. If a pixel's gray level value is similar to those around it, there is probably not an edge at that point. However, if a pixel has neighbors with widely varying gray levels, it may represent an edge point. In other words, an edge is defined by a discontinuity in gray level values. Ideally, an edge separates two distinct objects. In practice, apparent edges are caused by changes in color, texture or by the specific lighting

DOI: 10.1201/9781003221135-4

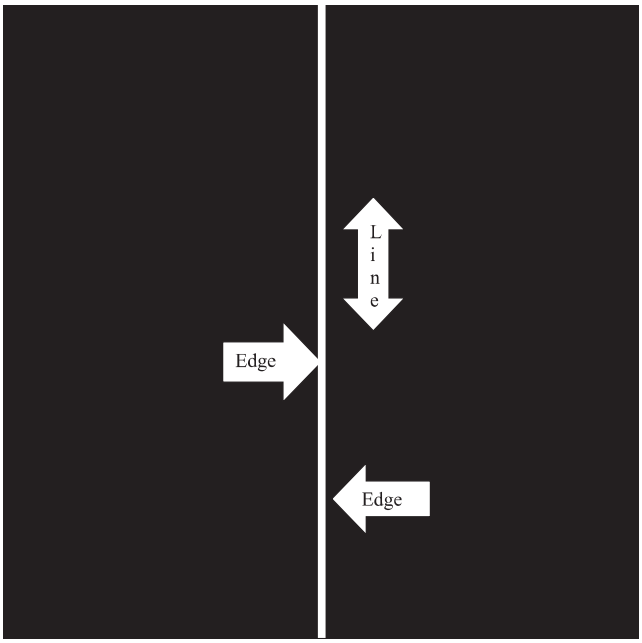

FIGURE 4.2-1
Edges and lines are perpendicular. The line shown here is vertical and the edge direction is horizontal. In this case, the transition from black to white occurs along a row, this is the edge direction, but the line is vertical along a column.

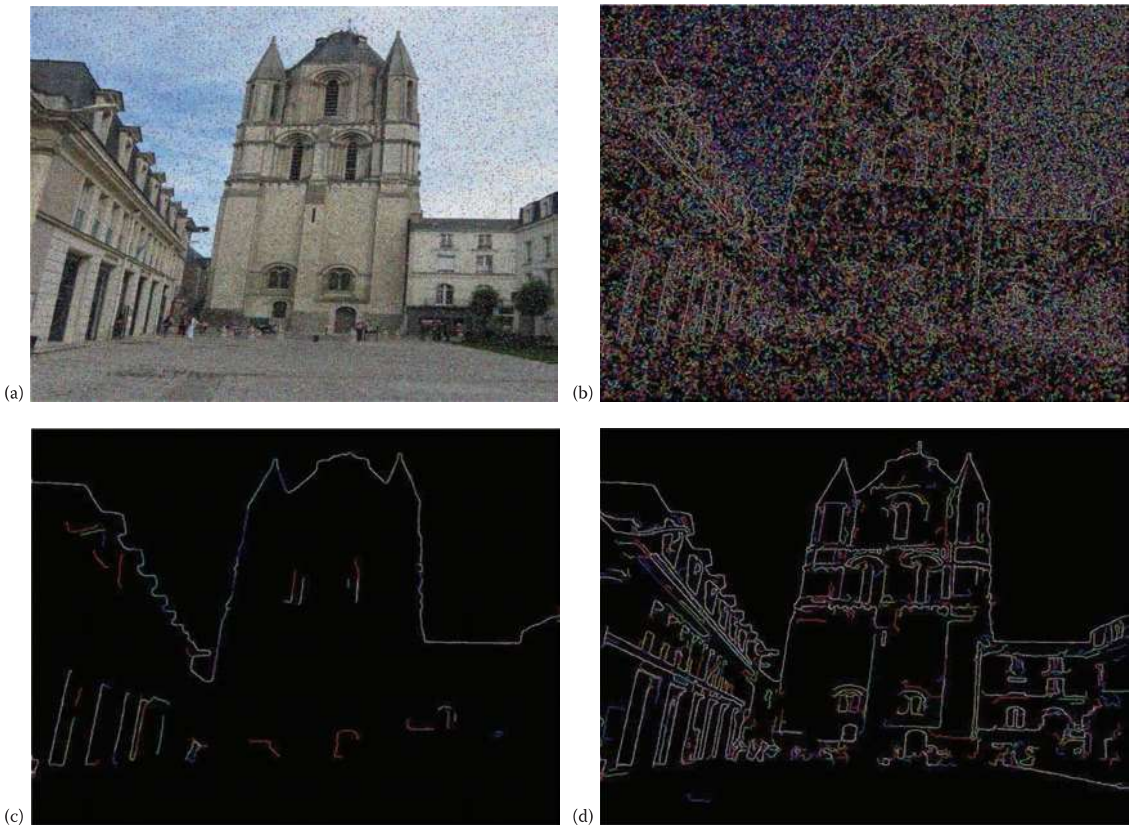

FIGURE 4.2-2
Noise in images requires tradeoffs between sensitivity and accuracy for edge detectors. (a) Noisy image, (b) edge detector too sensitive, many edge points found that are attributable to noise (c) edge detector not sensitive enough, loss of valid edge

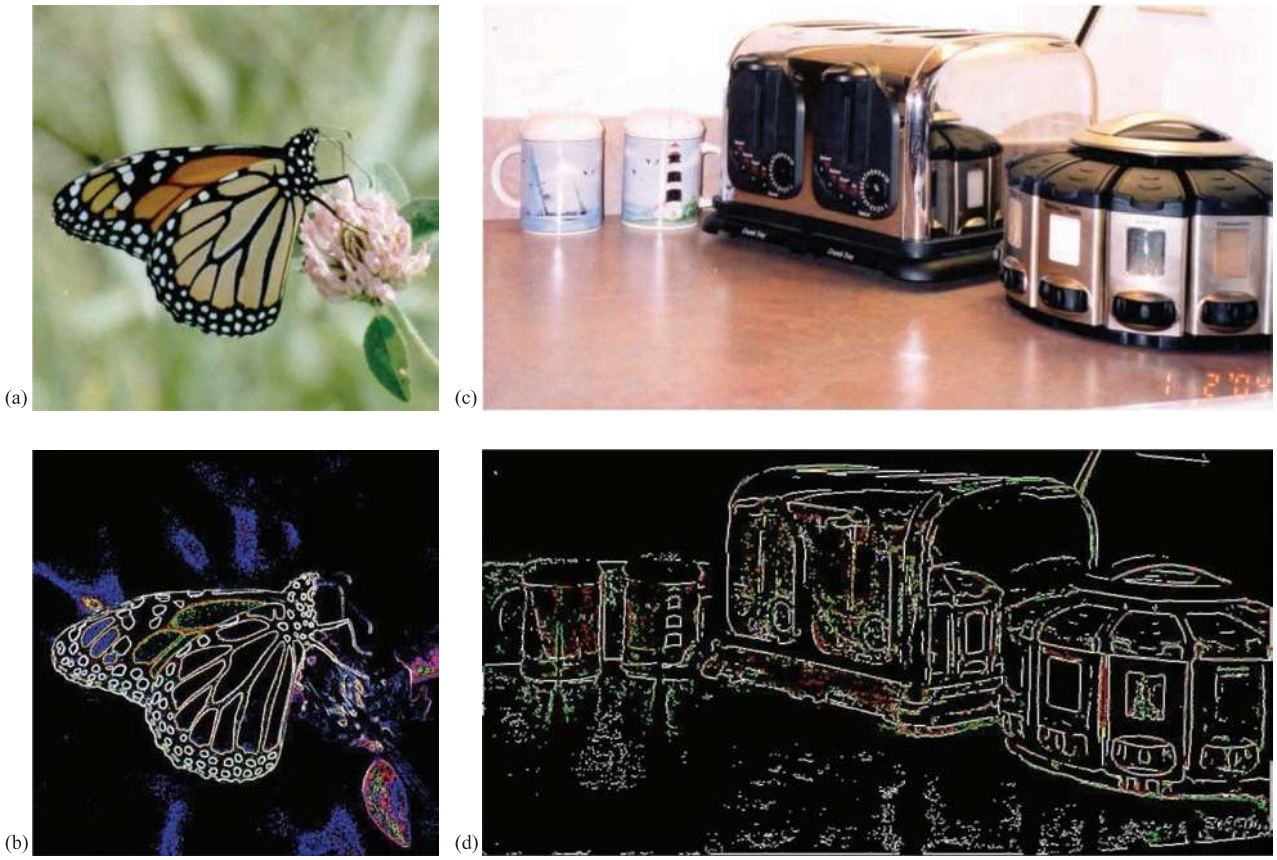

FIGURE 4.2-3
Image objects may be parts of real objects. (a) Butterfly image (original photo courtesy of Mark Zuke), (b) butterfly after edge detection, note that image objects are separated by color and brightness changes, (c) image of objects in kitchen corner, and (d) image after edge detection, note that some image objects are created by reflections in the image due to lighting conditions and object properties.

conditions present during the image acquisition process. This means that what we refer to as image objects may actually be only parts of the objects in the real world; see Figure 4.2-3.

Figure 4.2-4 illustrates the differences between an ideal edge and a real edge. Figure 4.2-4a shows a representation of one row in an image of an ideal edge. The vertical axis represents brightness and the horizontal axis shows the spatial coordinate. The abrupt change in brightness characterizes an ideal edge. In the corresponding image, the edge appears very distinct. In Figure 4.2-4b, we see the representation of a real edge which changes gradually. This gradual change is a minor form of blurring caused by the imaging device, the lenses, and/or the lighting and is typical for real-world (as opposed to computer generated) images. In the figure, where the edge has been exaggerated for illustration purposes, note that from a visual perspective, this image contains the same information as does the ideal image: black on one side and white on the other with a line down the center.

4.2.1 Gradient Operators

Gradient operators are based on the concept of using the first or second derivative of the gray level function as an edge detector. Remember from calculus that the derivative measures the rate of change of a line or the slope of the line. If we model the gray level transition of an edge by a ramp function, which is a reasonable approximation to a real edge, we can see what the first and second derivatives look like in Figure 4.2-5. When the gray level is constant, the first derivative is zero, and when it is linear, it is equal to the slope of the line. With the following operators, we will see that this is approximated with a difference operator, similar to the methods used to derive the definition of the derivative. The second derivative is a positive spike at the change on the dark side of the edge where the brightness is increasing, and it is a negative spike at the change on the light side where the increase stops and zero elsewhere.

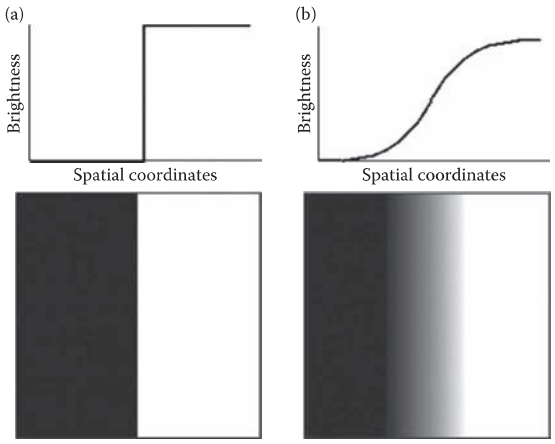

FIGURE 4.2-4
Ideal edge compared to a real edge. (a) Ideal edge and (b) real edge. The ideal edge is an instantaneous change, while a real edge is typically a gradual change. The real edge has been greatly enlarged for illustration purposes, note that from a visual perspective this image contains the same information as does the ideal image: black on one side, white on the other, with a line down the center.

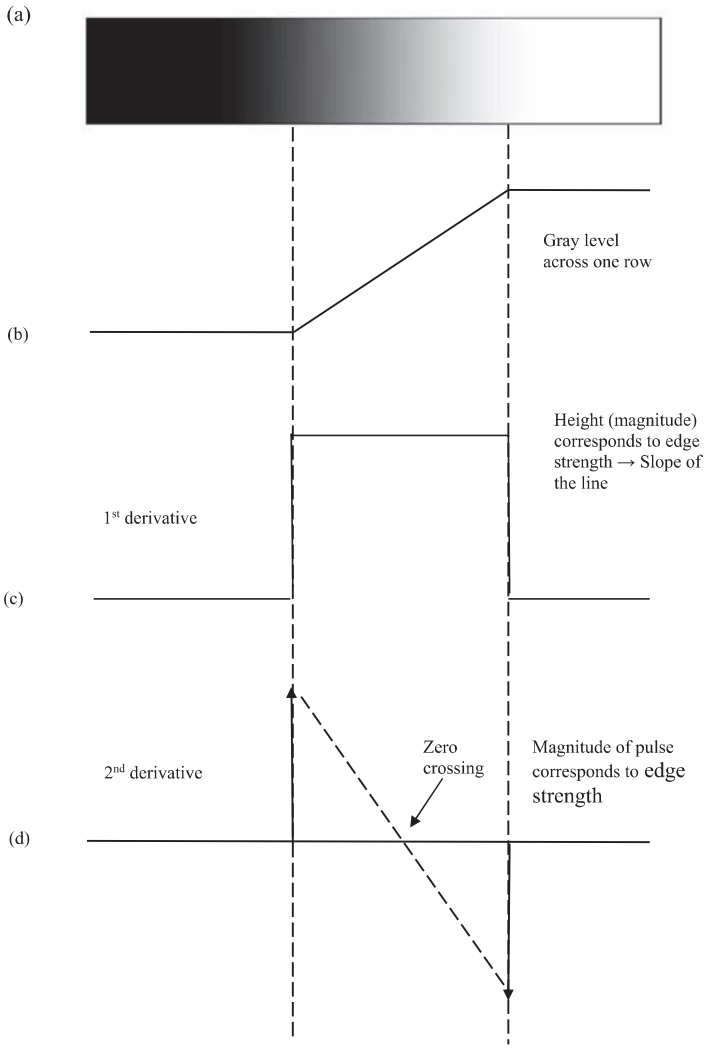

FIGURE 4.2-5
Edge model. (a) A portion of an image with an edge, which has been enlarged to show detail, (b) ramp edge model, (c) first

In Figure 4.2-5c, we can see that the magnitude of the first derivative will mark edge points, with steeper gray level changes corresponding to stronger edges and larger magnitudes from the derivative operators. In Figure 4.2-5d, we can see that applying a second derivative operator to an edge returns two impulses, one on either side of the edge. An advantage of this is that if a line is drawn between the two impulses, the position where this line crosses the zero axis is the center of the edge, which theoretically allows us to measure edge location to sub-pixel accuracy. Sub-pixel accuracy refers to the fact that the zero crossing (*zc*) may be at a fractional pixel distance, for example, halfway between two pixels, so we could say the edge is at, for instance, *zc* = 75.5.

The **Roberts operator** is a simple approximation to the first derivative. It marks edge points only; it does not return any information about the edge orientation. It is the simplest of the edge detection operators, so is the most efficient one for binary images. There are two forms of the Roberts operator. The first consists of the square root of the sum of the differences of the diagonal neighbors squared, as follows:

$$\sqrt{[I(r,c)-I(r-1,c-1)]^2+[I(r,c-1)-I(r-1,c)]^2} \tag{4.2-1}$$

The second form of the Roberts operator is the sum of the magnitude of the differences of the diagonal neighbors, as follows:

$$\left|I(r,c)-I(r-1,c-1)\right|+\left|I(r,c-1)-I(r-1,c)\right| \tag{4.2-2}$$

The second form of the equation is often used in practice due to its computational efficiency – it is typically faster for a computer to find an absolute value than to find square roots.

The **Sobel operator** approximates the gradient by using a row and a column mask, which will approximate the first derivative in each direction. The Sobel edge detection masks find edges in both the horizontal and vertical directions, and then combine this information into two metrics – magnitude and direction. The masks are as follows:

VERTICAL EDGE

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

HORIZONTAL EDGE

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

These masks are each convolved with the image. At each pixel location, we now have two numbers: s_1, corresponding to the result from the vertical edge mask, and s_2, from the horizontal edge mask. We use these numbers to compute two metrics, the edge magnitude and the edge direction, defined as follows:

$$\text{EDGE MAGNITUDE} \sqrt{s_1^2 + s_2^2} \tag{4.2-3}$$

$$\text{EDGE DIRECTION } \text{Tan}^{-1}\left[\frac{s_1}{s_2}\right] \tag{4.2-4}$$

As seen in Figure 4.2-1, the edge direction is perpendicular to the line (or curve), because the direction specified is the direction of the gradient, along which the gray levels are changing.

The **Prewitt** is similar to the Sobel, but with different mask coefficients. The masks are defined as follows:

VERTICAL EDGE

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

HORIZONTAL EDGE

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

These masks are each convolved with the image. At each pixel location, we find two numbers: p_1, corresponding to the result from the vertical edge mask, and p_2, from the horizontal edge mask. We use these results to determine

$$\text{EDGE MAGNITUDE } \sqrt{p_1^{\,2} + p_2^{\,2}} \tag{4.2-5}$$

$$\text{EDGE DIRECTION } \text{Tan}^{-1}\left[\frac{p_1}{p_2}\right] \tag{4.2-6}$$

As with the Sobel edge detector, the direction lies 90° from the apparent direction of the line or curve. The Prewitt is easier to calculate than the Sobel, since the only coefficients are 1's, which makes it easier to implement in hardware. However, the Sobel is defined to place emphasis on the pixels closer to the mask center, which may be desirable for some applications.

The **Laplacian operators** described here are similar to the ones used for preprocessing as described in Section 3.2.3. The three Laplacian masks presented below represent various practical approximations of the Laplacian, which is the two-dimensional version of the second derivative (note that these are masks used in practice and true Laplacians will have all the coefficients negated). Unlike the Sobel and Prewitt edge detection masks, the Laplacian masks are approximately rotationally symmetric, which means edges at all orientations contribute to the result. As that is the case, they are applied by selecting *one* mask and convolving it with the image. The sign of the result (positive or negative) tells us which side of the edge is brighter.

LAPLACIAN MASKS

Filter 1 Filter 2 Filter3

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

These masks differ from the Laplacian-type previously described in that the center coefficients have been decreased by one. With these masks, we are trying to find edges and are not interested in the image itself – if we increase the center coefficient by one, it is equivalent to adding the original image to the edge detected image.

An easy way to picture the difference is to consider the effect each mask has when applied to an area of constant value. The above convolution masks return a value of zero. If we increase the center coefficients by one, each mask returns the original gray level. Therefore, if we are only interested in edge information, the sum of the coefficients should be zero. If we want to retain most of the information that is in the original image, the coefficients should sum to a number greater than zero. The larger this sum, the less the processed image is changed from the original image. Consider an extreme example in which the center coefficient is very large compared with the other coefficients in the mask. The resulting pixel value will depend most heavily upon the current value, with only minimal contribution from the surrounding pixel values.

4.2.2 Compass Masks

The Kirsch and Robinson edge detection masks are called compass masks since they are defined by taking a single mask and rotating it to the eight major compass orientations: North, Northwest, West, Southwest, South, Southeast, East and Northeast. The **Kirsch compass masks** are defined as follows:

$$k_0\begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad k_1\begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad k_2\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad k_3\begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

$$k_4\begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad k_5\begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad k_6\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad k_7\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

The edge magnitude is defined as the maximum value found by the convolution of each of the masks with the image. The edge direction is defined by the mask that produces the maximum magnitude; for instance, k_0 corresponds to a horizontal edge, whereas k_5 corresponds to a diagonal edge in the Northeast/Southwest direction

(remember edges are perpendicular to the lines). We also see that the last four masks are actually the same as the first four, but flipped about a central axis.

The **Robinson compass masks** are used in a manner similar to the Kirsch masks, but are easier to implement, as they rely only on coefficients of 0, 1, and 2 and are symmetrical about their directional axis – the axis with the zeros which corresponds to the line direction. We only need to compute the results on four of the masks; the results from the other four can be obtained by negating the results from the first four. The masks are as follows:

$$r_0\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad r_1\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad r_2\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad r_3\begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

$$r_4\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad r_5\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} \quad r_6\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad r_7\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

The edge magnitude is defined as the maximum value found by the convolution of each of the masks with the image. The edge direction is defined by the mask that produces the maximum magnitude. It is interesting to note that masks r_0 and r_6 are the same as the Sobel masks. We can see that any of the edge detection masks can be extended by rotating them in a manner like these compass masks, which will allow us to extract explicit information about edges in any direction.

4.2.3 Thresholds, Noise Mitigation and Edge Linking

The gradient and compass mask edge detectors provide measures for magnitude and direction of the brightness gradient at each pixel point. This is only the first step in marking potential edge points which will be used to ultimately find object boundaries. As illustrated in Figure 4.2-2, consideration must be given to the sensitivity versus the accuracy of the edge detector. By considering any points that we do not want or need as "noise" and the valid, typically stronger edge points as the "real" edge points, the primary methods to control sensitivity and accuracy are the following: (1) control of a threshold, so that only the stronger edge points pass, (2) use of large mask sizes to mitigate noise and (3) use of a preprocessing mean filter.

We have discussed two thresholding methods in the previous chapter, the *Otsu Method* and the *Automatic Thresholding* (K-Means) – these algorithms work well with one object and high contrast with the background – a bimodal histogram as shown in Figure 4.2-6. These types of images are obtained in machine vision applications where the lighting can be controlled. This is not typically the case with the magnitude image that results from an edge detector, these images tend to have unimodal (one peak) histograms.

A method that provides reasonable results for unimodal histograms is to use the *average value* for the threshold, as in Figure 4.2-7. With very noisy images and a unimodal histogram, a good rule of thumb is to use 10%–25% of the maximum value as a threshold. An example of this is shown in Figure 4.2-8.

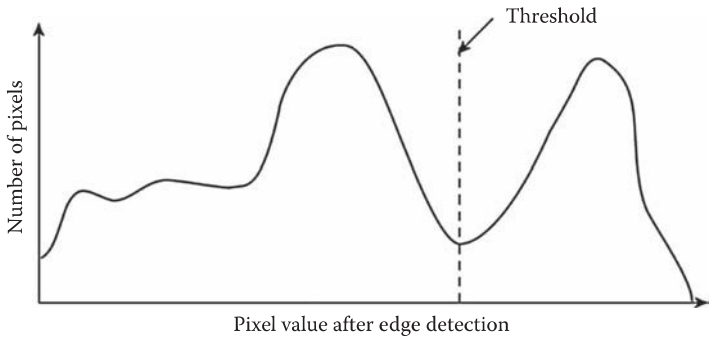

FIGURE 4.2-6
Edge detection thresholding via histogram. The histogram can be examined manually to select a good threshold. This method is easiest with a bimodal (two peaks) histogram. Alternately, the threshold can be found automatically with the Automatic Single Threshold or the Otsu algorithm described in Chapter 3.

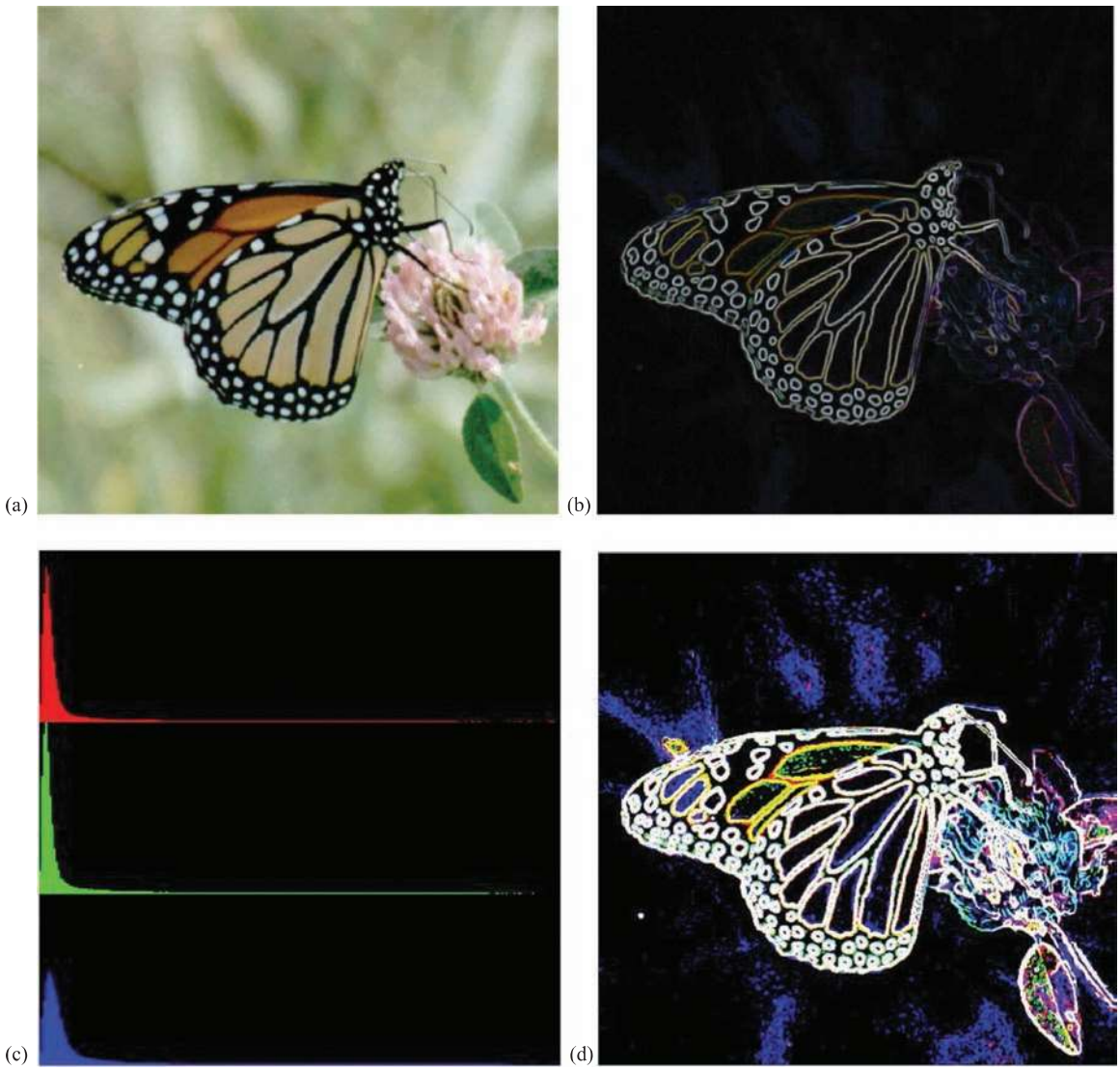

FIGURE 4.2-7
Average value thresholding. (a) Original image, (b) image after Sobel edge detector, (c) unimodal histogram of image after Sobel and (d) Sobel image after thresholding with average value.

If the images contain too much noise, most standard edge detectors perform poorly. The edge detector will tend to find more false edges as a result of the noise. We can preprocess the image with mean, or averaging, spatial filters to mitigate the effects from noise, or we can expand the edge detection operators themselves to mitigate noise effects. One way to do this is to extend the size of the edge detection masks. An example of this method is to extend the Prewitt edge mask as follows:

EXTENDED PREWITT EDGE DETECTION MASK

$$\begin{bmatrix} 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \end{bmatrix}$$

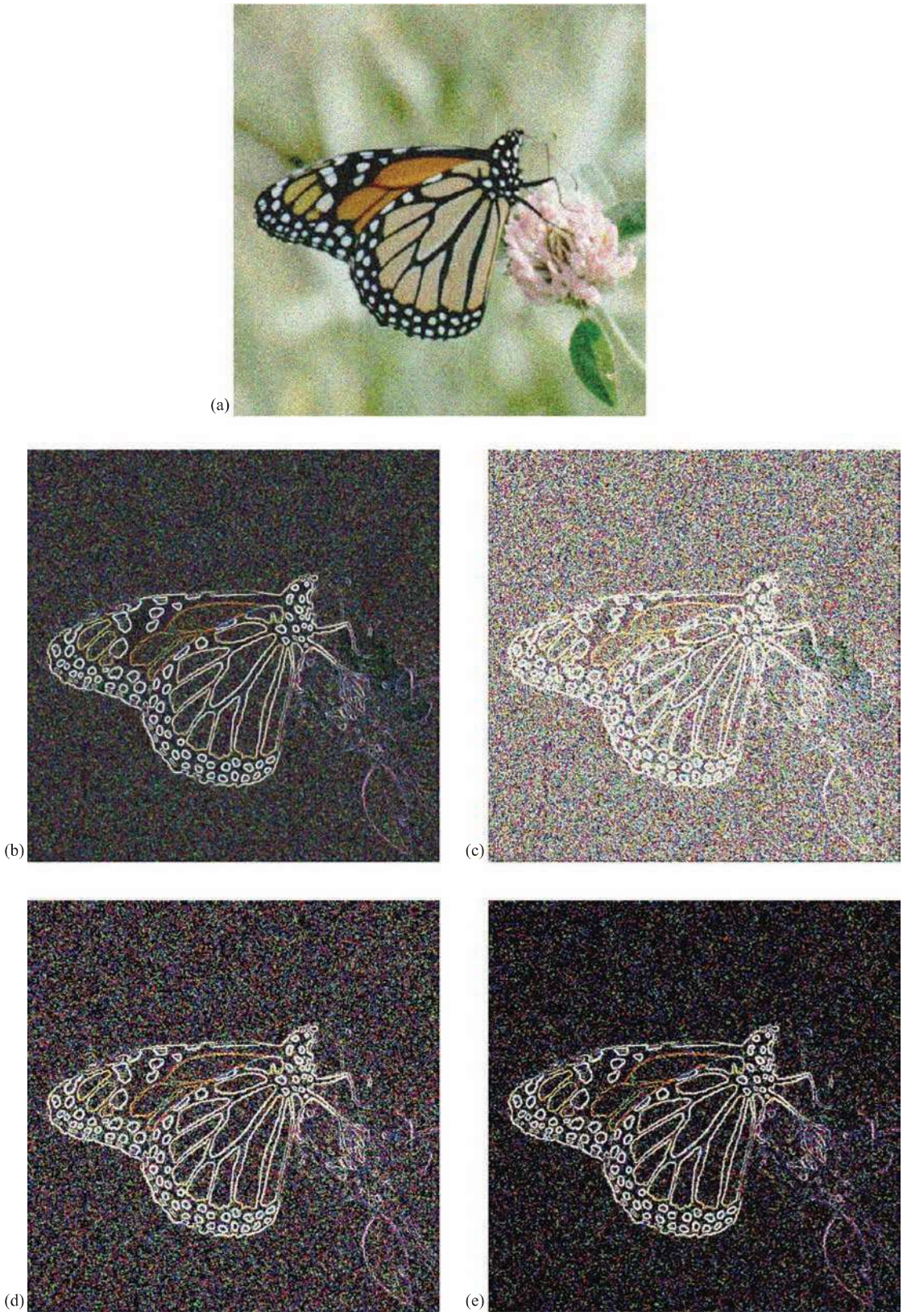

FIGURE 4.2-8
Thresholding noisy images. (a) Original image with Gaussian noise added (zero mean, variance = 800), (b) Sobel edge detector results (remapped), (c) threshold on Sobel at 10% of maximum value, (d) threshold on Sobel at 20% of maximum and (e) threshold on Sobel at 25% of maximum.

We then can rotate this by 90°, and we have both row and column masks which can be used like the Prewitt operators to return the edge magnitude and gradient. These types of operators are called boxcar operators and can be extended to any size, although 7×7, 9×9 and 11×11 are typical. The Sobel operator can be extended in a similar manner:

EXTENDED SOBEL EDGE DETECTION MASK

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

If we approximate a linear distribution, the *truncated pyramid* operator is created as follows:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 2 & 2 & 0 & -2 & -2 & -1 \\ 1 & 2 & 3 & 0 & -3 & -2 & -1 \\ 1 & 2 & 3 & 0 & -3 & -2 & -1 \\ 1 & 2 & 3 & 0 & -3 & -2 & -1 \\ 1 & 2 & 2 & 0 & -2 & -2 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \end{bmatrix}$$

This operator provides weights that decrease from the center pixel outward, which will smooth the result in a more natural manner. These operators are used in the same manner as the Prewitt and Sobel – we define a row and column mask and then find a magnitude and direction at each point. A comparison of applying the extended Prewitt operators with the standard 3×3 operators to an image is shown in Figure 4.2-9. We see that as the mask size increases, the lines thicken and small ones disappear. Figure 4.2-10 shows various edge detectors and the effect of larger masks on an image with Gaussian noise. Comparing Figure 4.2-10c and d and Figure 4.2-10e and f, we see that the extended operators exhibit better performance than the smaller masks. However, they require more computations and will smear the edges, which can be alleviated by postprocessing to thin the smeared edges and remove any leftover noise. Figure 4.2-11 shows results from changing Sobel mask sizes on an image with salt-and-pepper noise.

After a threshold for the edge detected image is determined, we need to merge the selected edge points into boundaries. This is done by *edge linking*. The simplest approach to edge linking involves considering each point that has passed the threshold test and connecting it to all other such points that are within a maximum distance. This method tends to connect many points and is not useful for images where too many points have been marked; it is most applicable to simple images.

Instead of thresholding and then edge linking, we can perform edge linking on the edge detected image before we threshold it. If this approach is used, we look at small neighborhoods (3×3 or 5×5) and link similar points. Similar points are defined as being spatially adjacent and having close values for both magnitude and direction. How close they need to be will be dependent on the application and the types of lines or curves we are wanting to identify. The entire image undergoes this process while keeping a list of the linked points. When the process is complete, the boundaries are determined by the linked points.

4.2.4 Advanced Edge Detectors

The advanced edge detectors are algorithms that use the basic edge detectors and combine them with the noise mitigation concepts and also introduce new ideas such as that of hysteresis thresholding. The edge detectors considered here include the Marr–Hildreth algorithm, the Canny algorithm, the Boie–Cox algorithm, the Shen–Castan

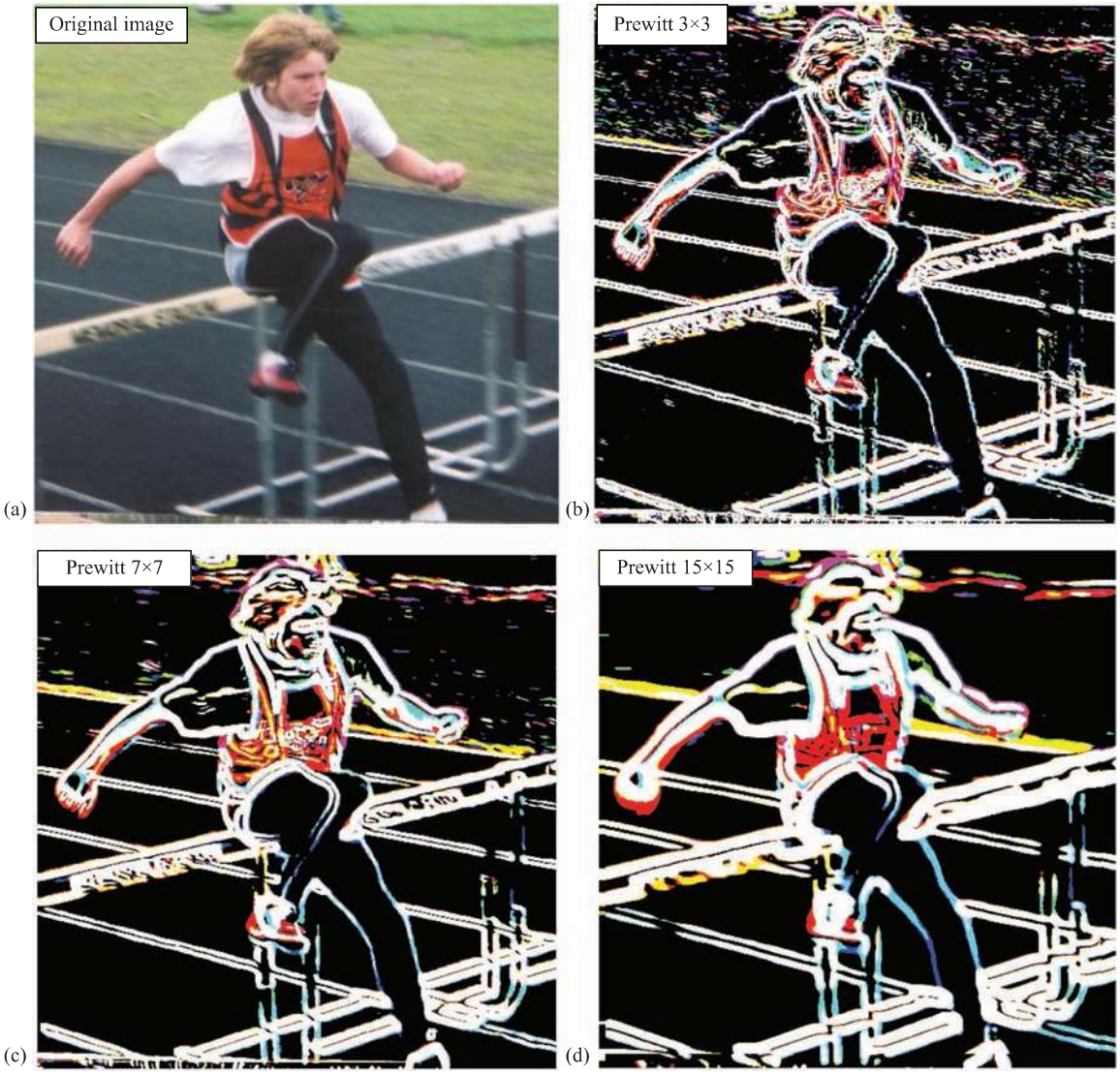

FIGURE 4.2-9
Edge detection example with various mask sizes. (a) Original image, (b) Prewitt magnitude with a 3×3 mask (c) Prewitt magnitude with a 7×7 mask and (d) Prewitt magnitude with a 15×15 mask. The images have undergone a threshold with the average value. Notice that a larger mask thickens the lines and eliminates small ones.

algorithm and the Frei–Chen masks. They are considered to be advanced because they are algorithmic in nature, which basically means they require multiple steps. Except for the Frei–Chen masks, these algorithms begin with the idea that, in general, most edge detectors are too sensitive to noise, and by blurring the image prior to edge detection, we can mitigate these noise effects. The noise considered here includes irrelevant image detail, as well as a combination of blurring from camera optics and signal corruption by camera electronics.

The simplest of these is the **Marr–Hildreth algorithm**, based on a model of the human visual system's response first developed by neuroscientist David Marr in the 1960s. The algorithm requires three steps:

1. Convolve the image with a Gaussian smoothing filter.
2. Convolve the image with a Laplacian mask.
3. Find the zero crossings of the image from Step 2.

By preprocessing with a smoothing filter, we can mitigate noise effects and then use the Laplacian to enhance the edges. By adjusting the spread, or variance, of the Gaussian, we can adjust the filter for different amounts of noise

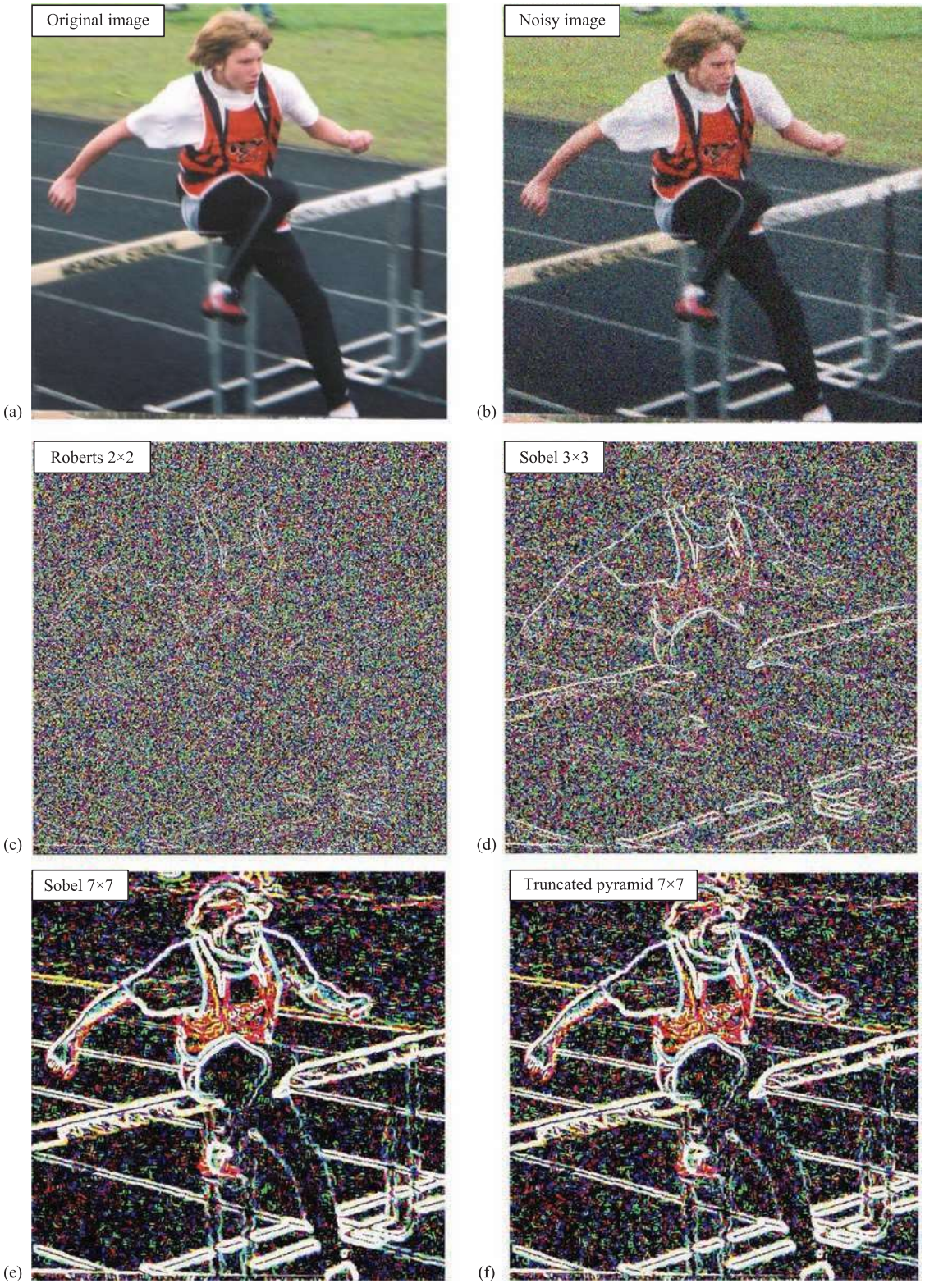

FIGURE 4.2-10
Edge detection examples with Gaussian noise – larger masks mitigate noise effects. (a) Original image, (b) image with added Gaussian noise (c) Robert's edge detection, 2×2, (d) Sobel with a 3×3 mask (e) Sobel with a 7×7 mask and (f) a 7×7 truncated pyramid. The images have undergone a threshold with the average value. In (c), with the 2×2 Roberts, the noise conceals almost all the edges. In (d), with a 3×3 Sobel mask, the edges are visible, but the resultant image is very noisy. With the 7×7 mask, shown in (e) and (f), the edges are much more prominent and the noise is much less noticeable.

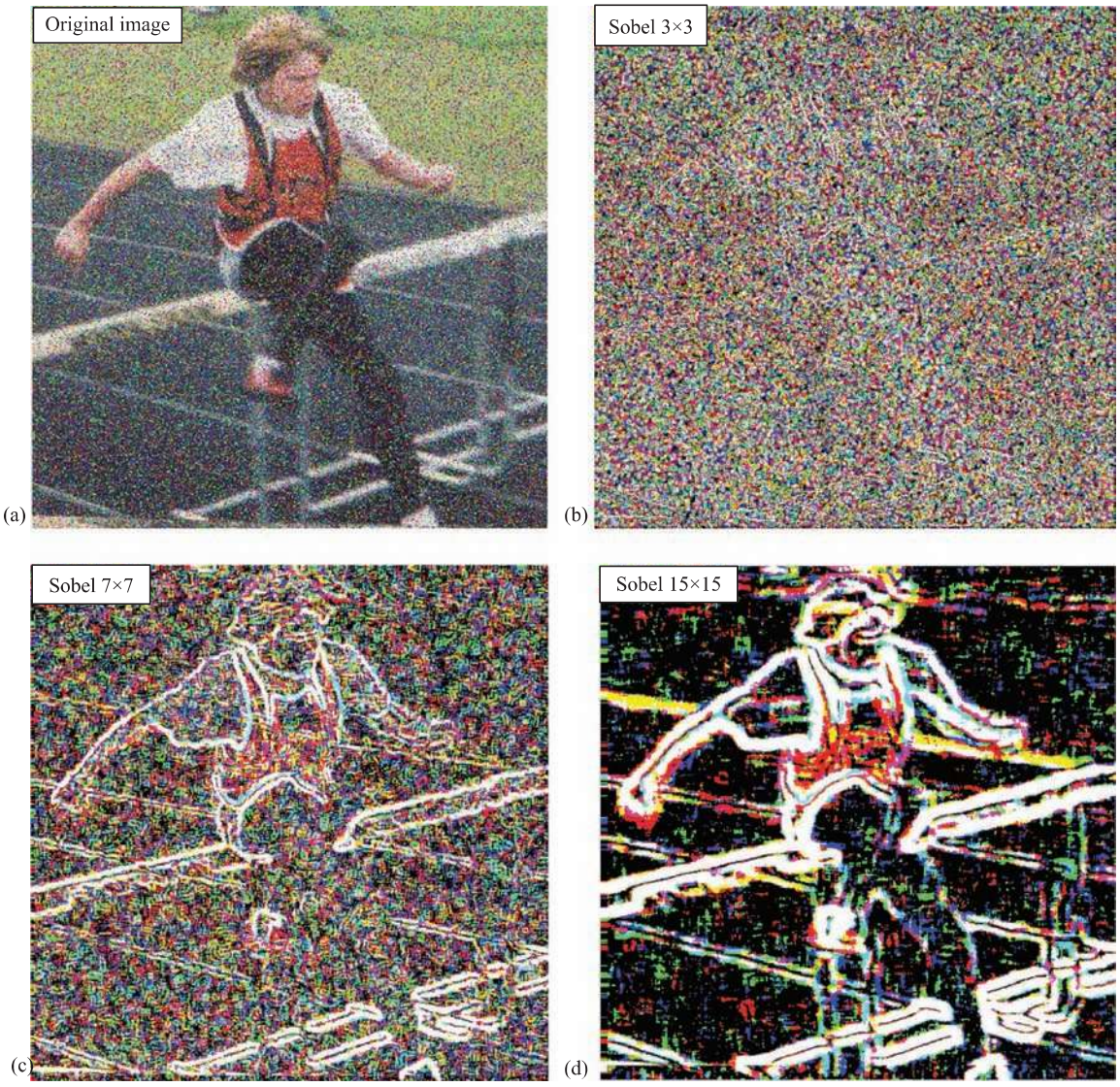

FIGURE 4.2-11
Edge detection with salt-and-pepper noise, Sobel example with various mask sizes. (a) Original image, (b) Sobel magnitude with a 3×3 mask (c) Sobel magnitude with a 7×7 mask and (d) Sobel magnitude with a 15×15 mask. The images have undergone a threshold with the average value. Notice that a larger mask helps mitigate the noise substantially, with a 3×3 the image is not even visible.

and various amounts of blurring. The combination of the Gaussian followed by a Laplacian is called a **Laplacian of a Gaussian (*LoG*)** or the *Mexican hat* operator since the function resembles a sombrero (see Figure 4.2-12). Since the process requires the successive convolution of two masks, they can be combined into one *LoG* mask. Commonly used 5×5 and 17×17 masks that approximate the combination of the Gaussian and Laplacian into one convolution mask are as follows:

5×5 Laplacian of a Gaussian mask:

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

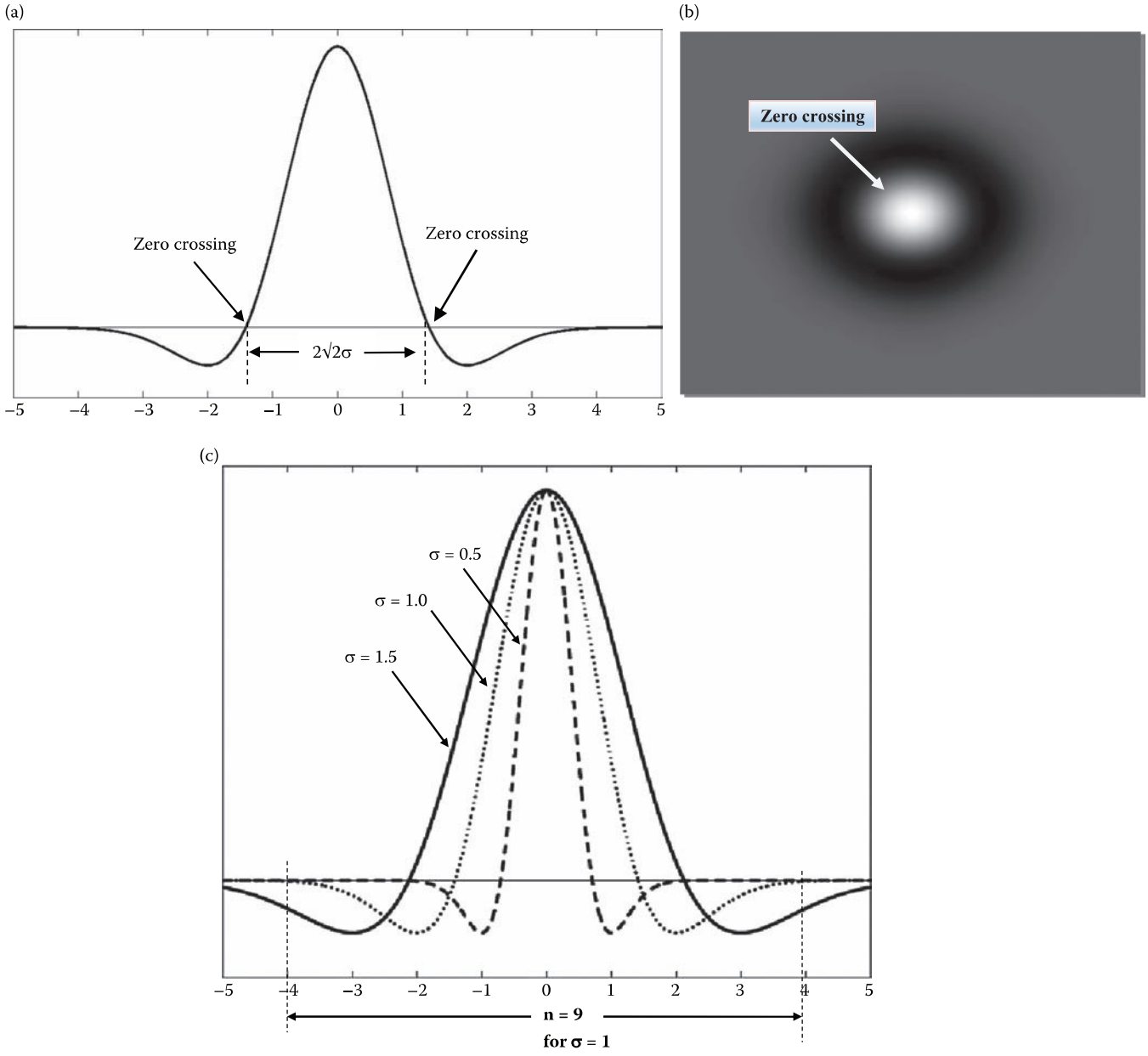

FIGURE 4.2-12

The *inverted* Laplacian of a Gaussian (*LoG*). (a) One-dimensional plot of the *LoG* function; (b) the *LoG* as an image with white representing positive numbers, black negative numbers and gray representing zero; and (c) three *LoG* plots with $\sigma = 0.5$, 1.0 and 1.5. Note for $\sigma = 0.5$, the mask size, n, should be about 5×5; for $\sigma = 1$, 9×9, and so on. This is done so that the mask covers the entire function as it goes negative and then goes back up to zero. Note this is 4σ to the left, 4σ to the right and the center term corresponding to the term at the 0 point on the graph.

17×17 Laplacian of a Gaussian mask:

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & -1 & 0 & 0 \\
0 & 0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 & 0 \\
-1 & -1 & -1 & -2 & -3 & -3 & -3 & -2 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 \\
-1 & -1 & -2 & -3 & -3 & -3 & 0 & 2 & 4 & 2 & 0 & -3 & -3 & -3 & -2 & -1 & 0 \\
-1 & -1 & -3 & -3 & -3 & 0 & 4 & 10 & 12 & 10 & 4 & 0 & -3 & -3 & -3 & -1 & -1 \\
-1 & -1 & -3 & -3 & -3 & 2 & 10 & 18 & 21 & 18 & 10 & 2 & -2 & -3 & -3 & -1 & -1 \\
-1 & -1 & -3 & -3 & -3 & 4 & 12 & 21 & 24 & 21 & 12 & 4 & -3 & -3 & -3 & -1 & -1 \\
0 & -1 & -3 & -3 & -3 & 2 & 10 & 18 & 21 & 18 & 10 & 2 & -2 & -3 & -3 & -1 & -1 \\
0 & -1 & -3 & -3 & -3 & 0 & 4 & 10 & 12 & 10 & 4 & 0 & -3 & -3 & -3 & -1 & -1 \\
0 & -1 & -2 & -3 & -3 & -3 & 0 & 2 & 4 & 2 & 0 & -3 & -3 & -3 & -3 & -1 & 0 \\
0 & -1 & -1 & -2 & -3 & -3 & -3 & -2 & -3 & -2 & -3 & -3 & -3 & -2 & -2 & -1 & 0 \\
0 & 0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 & 0 \\
0 & 0 & -1 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

The equation for the LoG filter is:

$$\text{LoG} = \left[\frac{r^2 + c^2 - 2\sigma^2}{\sigma^4}\right] e^{-\left(\frac{r^2+c^2}{2\sigma^2}\right)} \tag{4.2-7}$$

where (r, c) are the row and column coordinates, and σ^2 is the Gaussian variance and σ is the standard deviation. From the equation, we can see that zero crossings occur at $(r^2 + c^2) = 2\sigma^2$, or $\sqrt{2}\sigma$ from the mean, as shown in Figure 4.2-12a. Note that, in practice, if creating the *LoG* filter mask by convolving a Gaussian and a Laplacian mask, we need to be sure that the Gaussian is normalized to one and that the Laplacian coefficients sum to zero. This is done to avoid biasing the mask with a DC term, which will shift the zero crossings and defeat the filter's purpose.

To determine the size of the mask to use, we consider that 99.7% of the area under a Gaussian curve is within $\pm 3\sigma$ of the mean. Keeping in mind that the sampling grid is fixed by the pixel spacing, the variance and the mask size must be related (see Figure 4.2-12c). So, we want to select a value of n for the $n \times n$ convolution mask that is an odd integer greater than or equal to 6σ, or we will get only a portion of the curve with our sampled filter mask. In CVIPtools, we use the following equation to determine n, based on the standard deviation, σ:

$$n = \left[2 * \text{TRUNCATE}(3.35\sigma + 0.33) + 1\right] \tag{4.2-8}$$

This equation assures us that we have the complete spread of the *LoG* filter and actually provides us with an n that corresponds to about $\pm 4\sigma$. Note that for positive numbers, the *truncate* operation is the same as the *floor* operation.

The third step for the Marr–Hildreth algorithm is to find the zero crossings after the *LoG* is performed. This can be accomplished by considering a pixel and its surrounding pixels, thus a 3×3 subimage, and looking for sign changes between two of the opposing neighbors. That is, we check the left/right, up/down and the two diagonal neighboring pairs. Figure 4.2-13 illustrates the results from the standard Marr–Hildreth algorithm. The disadvantage of the Marr–Hildreth algorithm, or any second derivative/zero-crossing method, is that it tends to smooth shapes too much which has the effect of eliminating corners and creating closed loops in the resulting lines/curves. The Marr–Hildreth results are often referred to as a "plate of spaghetti", as seen in Figure 4.2-13b.

In practice, we may want to set a threshold to use before a pixel is classified as an edge. The threshold is tested against the absolute value of the difference between the two pixels that have the sign changes. If this value exceeds the threshold, it is classified as an edge pixel.

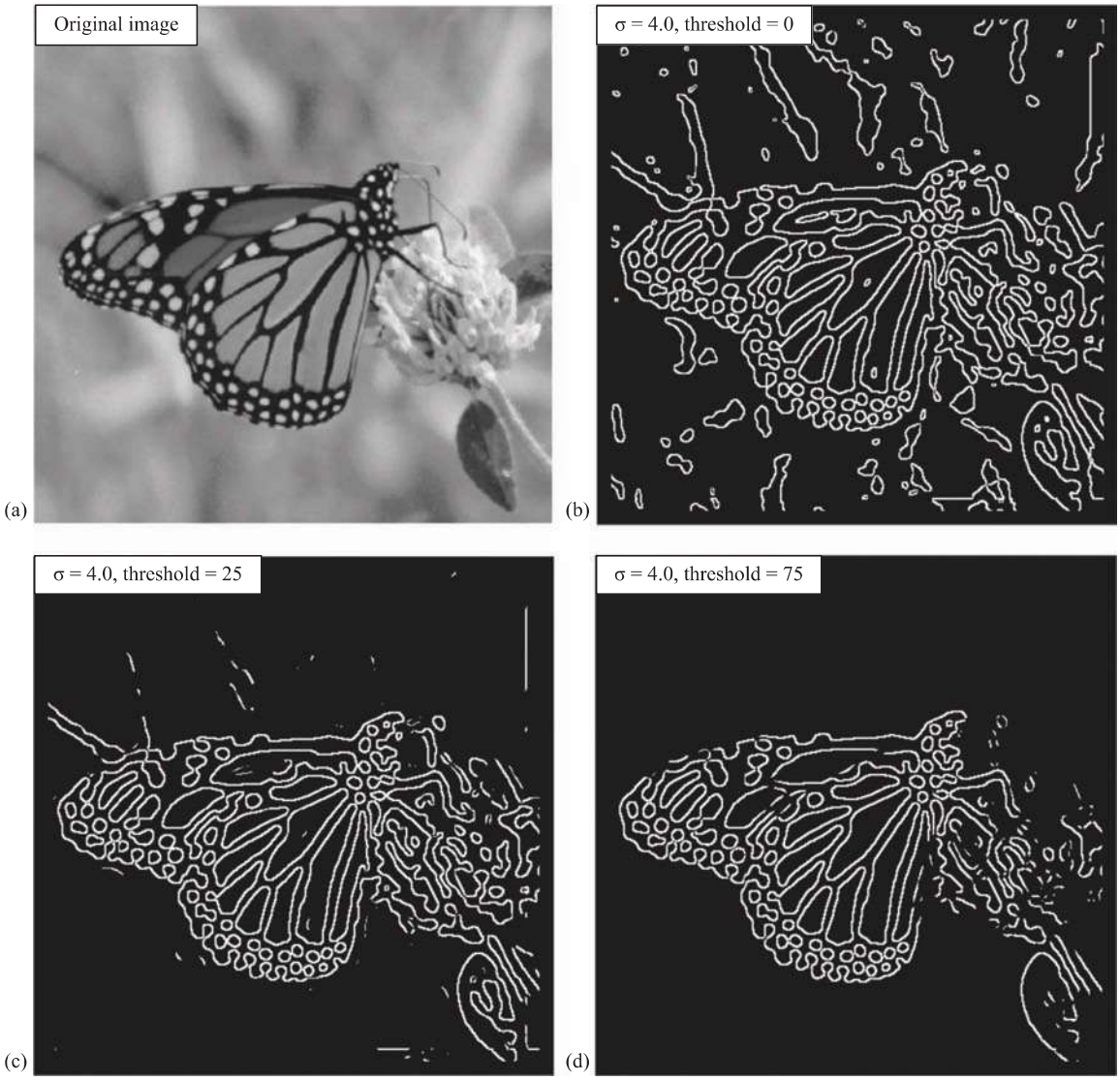

FIGURE 4.2-13
Marr–Hildreth algorithm results. (a) Original image, (b) Sigma (σ) = 4, threshold = 0, note the algorithm creating closed loops Marr–Hildreth algorithm results. (c) Sigma (σ) = 4, threshold = 25 and (d) Sigma (σ) = 4, threshold = 75; these values eliminate most of the background noise, but keep the butterfly.

EXAMPLE 4.2.1: APPLYING A THRESHOLD TO STEP 3 OF THE MARR–HILDRETH ALGORITHM

Suppose, after the *LoG*, we have a 3×3 subimage as follows:

$$\begin{bmatrix} -10 & 11 & 17 \\ 18 & 2 & 15 \\ 21 & 33 & 28 \end{bmatrix}$$

the only pair that has a sign change is the NW/SE diagonal. So, the center pixel may be considered an edge pixel. If we apply a threshold, we calculate the absolute value of the difference of this pair:

$$|-10 - 28| = 38$$

Now, if this value exceeds the threshold we have set, then the center pixel is determined to be an edge pixel.

The Marr–Hildreth as implemented in CVIPtools has a parameter to allow the user to select *single variance* or *dual variance*. If dual variance is selected, the user specifies a *sigma* (σ, standard deviation) value and a *delta* value. CVIPtools then computes the Marr–Hildreth results using two variances – the specified sigma plus the delta value and the specified sigma minus the delta value. These results are then combined into a single image with a logical AND function. For color images, the user can also select to combine the bands, which performs a logical AND of the red, green and blue band results.

The **Canny algorithm**, developed by John Canny in 1986, is an optimal edge detection method based on a specific mathematical model for edges. The edge model is a step edge corrupted by Gaussian noise. The algorithm consists of four primary steps:

1. **Apply a Gaussian filter mask to smooth the image** to mitigate noise effects. This can be performed at different scales by varying the size of the filter mask which corresponds to the variance of the Gaussian function. A larger mask will blur the image more and will find fewer, but more prominent, edges.
2. **Find the magnitude and direction of the gradient** using equations similar to the Sobel or Prewitt edge detectors, for example:

$$\begin{array}{cc} \text{VERTICAL} & \text{HORIZONTAL} \\ 1/2\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} & 1/2\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \end{array}$$

   These masks are each convolved with the image. At each pixel location, we find two numbers: c_1, corresponding to the result from the vertical edge mask, and c_2, from the horizontal edge mask. We use these results to determine two metrics, the edge magnitude and the edge direction, which are defined as follows:

$$\text{EDGE MAGNITUDE} \sqrt{c_1^{\,2} + c_2^{\,2}} \tag{4.2-9}$$

$$\text{EDGE DIRECTION Tan}^{-1}\left[\frac{c_1}{c_2}\right] \tag{4.2-10}$$

3. **Apply nonmaxima suppression** which results in thinned edges. This is done by considering small neighborhoods in the magnitude image, for example, 3×3, and comparing the center value to its neighbors in the direction of the gradient. If the center value is not larger than the neighboring pixels along the gradient direction, then set it to zero. Otherwise, it is a local maximum, so we keep it. In Figure 4.2-14, we see an example of a 3×3 neighborhood showing the magnitude at each location and using an arrow to show the gradient direction. The center pixel has a value of 100 and the gradient direction is horizontal (corresponding to a vertical line), so it is compared to the pixels to the right and left which are 40 and 91. Since it is greater than both, it is retained as an edge pixel; if it was less than either one, it would be removed as an edge point. Note that this will have the effect of making thick edges thinner by selecting the "best" point along a gradient direction.
4. **Apply two thresholds** to obtain the final result. This technique, known as *hysteresis thresholding*, helps to avoid false edges caused by too low a threshold value or missing edges caused by too high a value. It is a two-step thresholding method, which first marks edge pixels above a high threshold and then applies a low threshold to pixels connected to the pixels found with the high threshold. This can be performed multiple times, as either a recursive or an iterative process.

$$\begin{bmatrix} \leftarrow 50 & 112 \rightarrow & 20 \rightarrow \\ \leftarrow 40 & 100 \rightarrow & 91 \rightarrow \\ \leftarrow 88 & 95 \rightarrow & 92 \rightarrow \end{bmatrix}$$

FIGURE 4.2-14
Nonmaxima suppression. A 3×3 subimage of the magnitude image, which consists of the magnitude results in an image grid. The arrows show the gradient directions. This particular subimage has a vertical line (a horizontal edge). To apply nonmaxima suppression, we compare the center pixel magnitude along the gradient direction. Here, the 100 is compared with the 40 and the 91. Since it is a local maximum, it is retained as an edge pixel.

In CVIPtools, the high threshold is computed from the image by finding the value which is greater than 90% of the pixels after applying nonmaxima suppression to the magnitude images. The high threshold is multiplied with the high threshold factor to obtain the final high threshold for hysteresis. The low threshold is computed from the image by averaging the high threshold and minimum value in the image after applying the nonmaxima suppression to the magnitude images. The low threshold is then multiplied with the low threshold factor to obtain the final low threshold for hysteresis. CVIPtools also allows the variance of the Gaussian filter as an input parameter. Figures 4.2-15 and 4.2-16 show results from varying these parameters.

The **Boie–Cox algorithm**, developed in 1986 and 1987, is a generalization of the Canny algorithm. It consists of similar steps, but uses matched filters and Wiener filters (further explored in the references) to allow for a more generalized edge model. The **Shen–Castan algorithm**, developed in 1992, uses an optimal filter function they derived called an infinite symmetric exponential filter. Shen and Castan claim that their filter does better than the Canny at finding the precise location of the edge pixels. Like the Canny, it uses a smoothing filter followed by a similar multistep algorithm to find edge pixels. The search includes steps similar to the Canny, but with modifications and extensions (for more details, see the references). Figure 4.2-17 shows results from these algorithms.

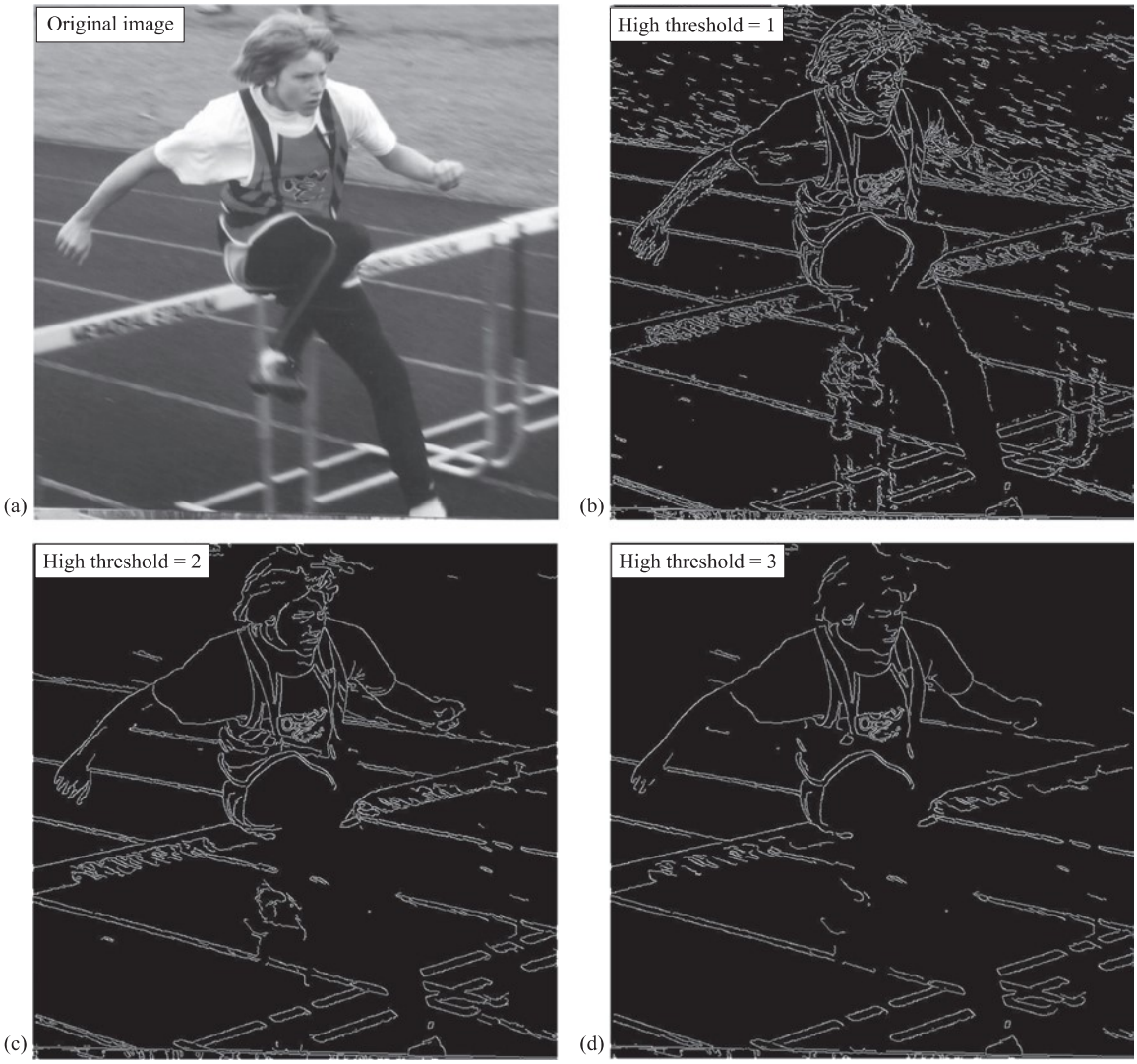

FIGURE 4.2-15
Results from changing high threshold with Canny algorithm. As the high threshold is increased, small details are removed. In the results, the Gaussian = 0.5 and the low threshold = 1. (a) Original image, (b) results high threshold factor = 1 (c) high threshold factor = 2 and (d) high threshold factor = 3.

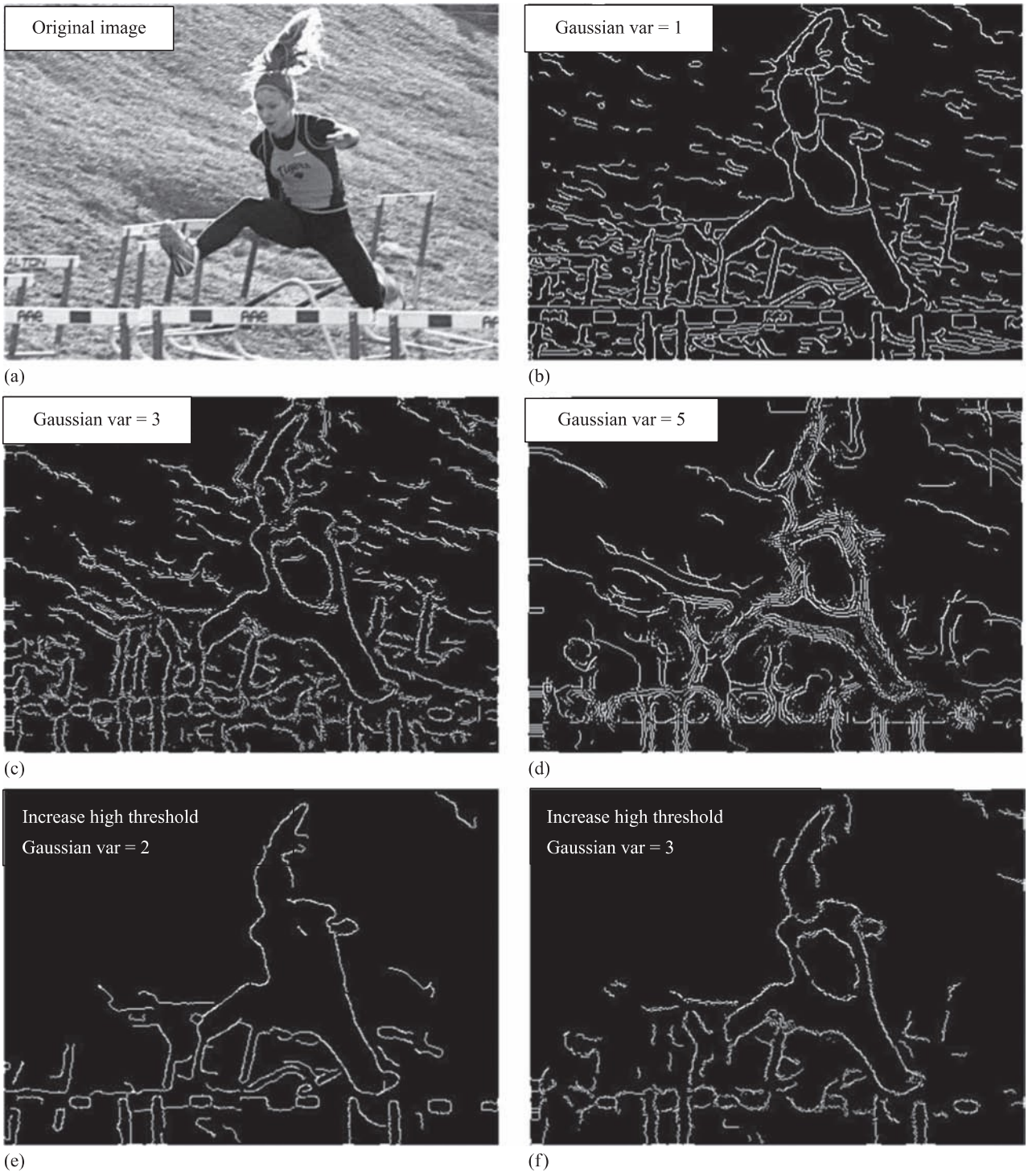

FIGURE 4.2-16

Results from changing Gaussian variance with the Canny algorithm. As the Gaussian variance is increased, the Canny should find fewer, but more prominent edges. In these results, the low and high threshold factors = 1. (a) Original image, (b) Canny results with Gaussian variance = 1, (c) Gaussian variance = 3 and (d) Gaussian variance = 5. Note that if the variance is too large, the image is blurred too much, and instead of finding "fewer, more prominent edges", we find fuzzy and then multiple edges. To avoid this, we need to also increase the high threshold. In the next two images, the low threshold is 1, but the high threshold has been increased to 2. Now we do see fewer, more prominent edges. (e) Gaussian variance = 2 and (f) Gaussian variance = 3. Results with noise in the image. Note that as the variance is increased, the false edges from the salt-and-pepper noise are mitigated.

(*Continued*)

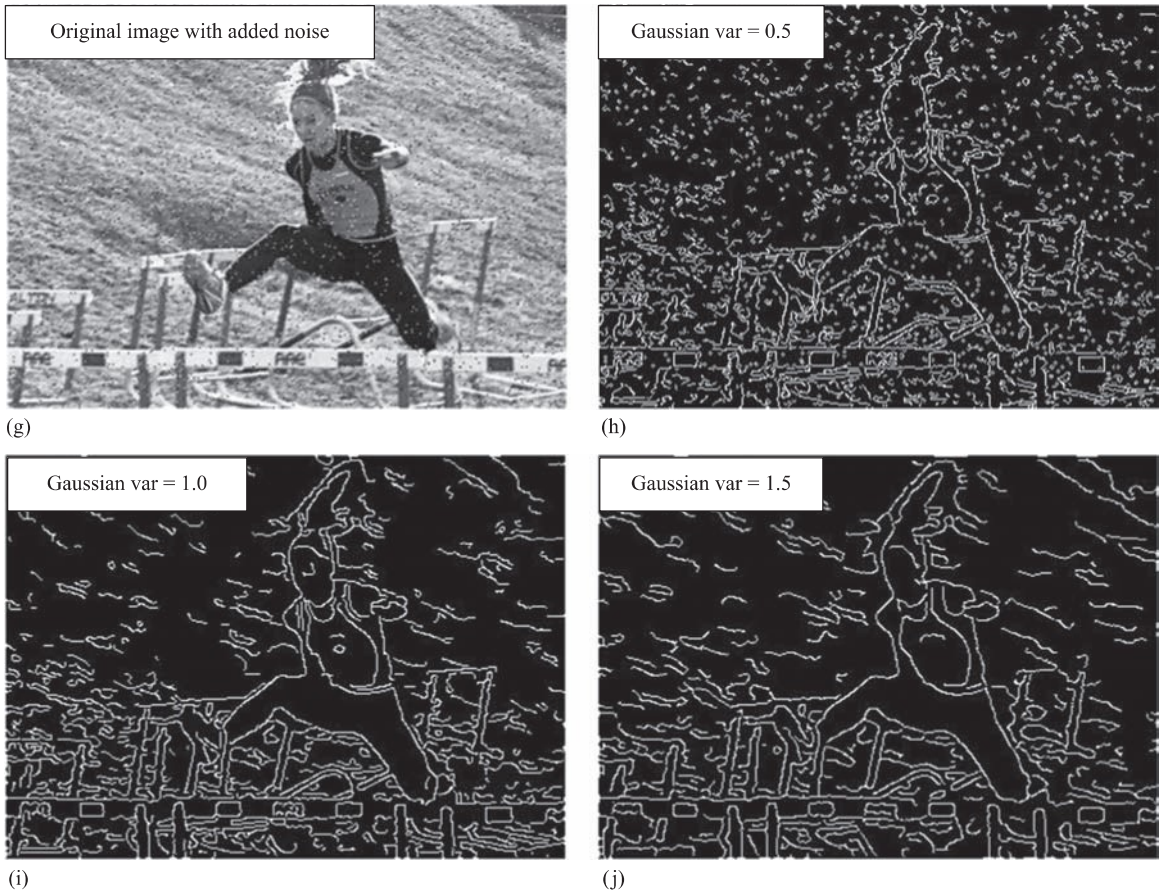

FIGURE 4.2-16 (*Continued*)
(g) original image with salt-and-pepper noise added, 2% each, (h) Canny with variance = 0.5, (i) Canny with variance = 1.0 and (j) Canny with variance = 1.5.

The **Frei–Chen masks** are unique in that they form a complete set of *basis vectors*. This means we can represent any 3×3 subimage as a weighted sum of the nine Frei–Chen masks (Figure 4.2-18). These weights are found by projecting a 3×3 subimage onto each of these masks. This projection process is similar to the convolution process in that both overlay the mask on the image, multiply coincident terms and sum the results (also called a *vector inner product*). This is best illustrated by example.

EXAMPLE 4.2.2

Suppose we have the following subimage, I_s:

$$I_s = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

To project this subimage onto the Frei–Chen masks, start by finding the projection onto f_1. Overlay the subimage on the mask and consider the first row. The 1 in the upper left corner of the subimage coincides with the 1 in the upper left corner of the mask, the 0 is over the $\sqrt{2}$, and the 1 on the upper right corner of the subimage coincides with the 1 in the mask. Note that all these must be summed and then multiplied by the $\frac{1}{2\sqrt{2}}$ factor to normalize the masks. The projection of I_s onto f_1 is equal to:

$$\frac{1}{2\sqrt{2}}\left[1(1)+0\left(\sqrt{2}\right)+1(1)+1(0)+0(0)+1(0)+1(-1)+0\left(-\sqrt{2}\right)+1(-1)\right]=0$$

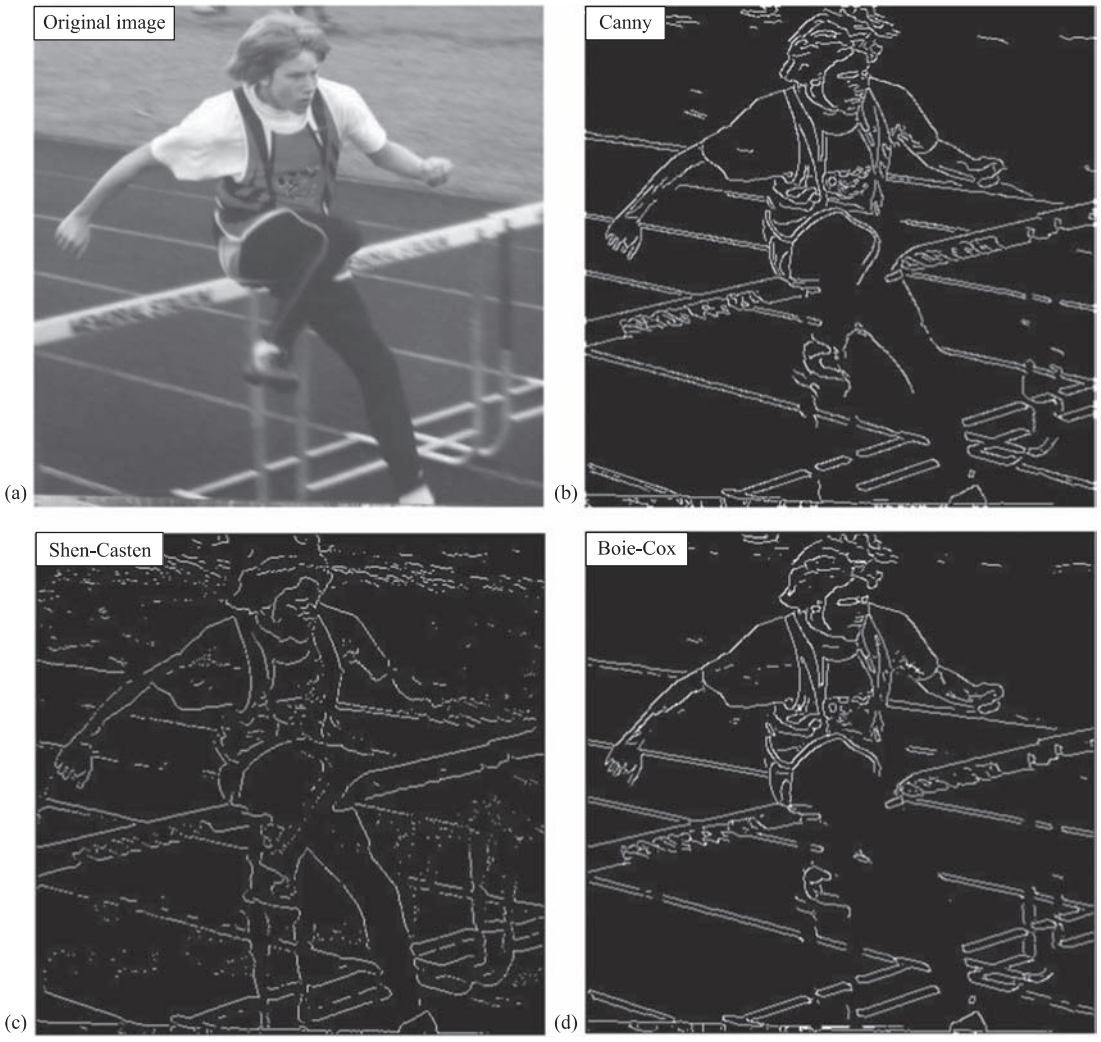

FIGURE 4.2-17
Comparison of the Canny, Shen–Castan and Boie–Cox algorithms. These results used the default parameters in CVIPtools. (a) Original image, (b) Canny (c) Shen–Castan and (d) Boie–Cox.

If we follow this process and project the subimage, I_s, onto each of the Frei–Chen masks, we get the following:

$$f_1 \to 0,\ f_2 \to 0,\ f_3 \to 0,\ f_4 \to 0,\ f_5 \to -1,\ f_6 \to 0,\ f_7 \to 0,\ f_8 \to -1,\ f_9 \to 2.$$

We can now see what is meant by a complete set of basis vectors allowing us to represent a subimage by a weighted sum. The basis vectors in this case are the Frei–Chen masks, and the weights are the projection values. Take the weights and multiply them by each mask, then sum the corresponding values. For this example, the only nonzero terms correspond to masks f_5, f_8 and f_9, and we find the following:

$$(-1)\left(\frac{1}{2}\right)\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} + (-1)\left(\frac{1}{6}\right)\begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix} + (2)\left(\frac{1}{3}\right)\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = I_S$$

We have seen how the Frei–Chen masks can be used to represent a subimage as a weighted sum, but how are they used for edge detection? The Frei–Chen masks can be grouped into a set of four masks for an *edge* subspace, four