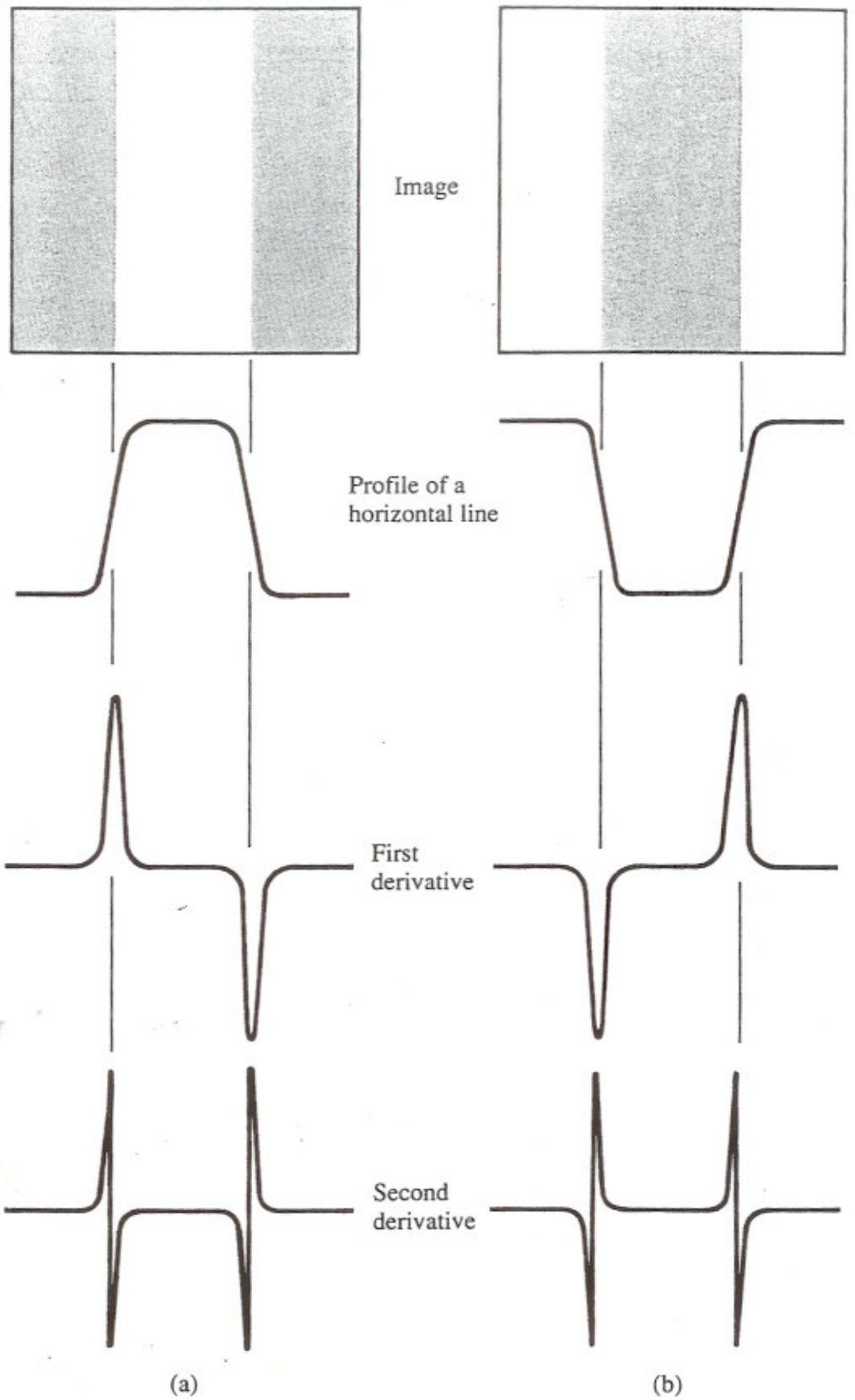


Figure 7.4 *Edge detection by derivative operators: (a) light stripe on a dark background; (b) dark stripe on a light background. Note that the second derivative has a zero crossing at the location of each edge.*

10.10 Laplace-Based Edge Detection

10.10.1 Laplace Filter

With second derivatives, we can easily form an isotropic linear operator, the *Laplace operator* with the transfer function $-(\pi\tilde{k})^2$. We can directly derive second-order derivative operators by a twofold application of first-order operators

$$\mathcal{D}_x^2 = {}^-\mathcal{D}_x\,{}^+\mathcal{D}_x. \tag{10.92}$$

The discrete Laplace operator $\mathcal{L} = \mathcal{D}_x^2 + \mathcal{D}_y^2$ for 2-D images thus has the filter mask

$$L = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{10.94}$$

and the transfer function

$$\hat{L}(\tilde{k}) = -4\sin^2(\pi\tilde{k}_x/2) - 4\sin^2(\pi\tilde{k}_y/2). \tag{10.95}$$

Like other discrete approximations of operators, the Laplace operator is only isotropic for small wave numbers (Fig. 10.28a):

$$\hat{L}(\tilde{k},\theta) = -(\pi\tilde{k})^2 + \frac{3}{48}(\pi\tilde{k})^4 + \frac{1}{48}\cos 4\phi\,(\pi\tilde{k})^4 + O(\tilde{k}^6). \tag{10.96}$$

There are many other ways to construct a discrete approximation for the Laplace operator. An interesting possibility is the use of binomial masks. With (10.23) we can approximate all binomial masks for sufficiently small wave numbers by

$$\hat{B}^{2R}(\tilde{k}) \approx 1 - R\frac{\pi^2}{4}\tilde{k}^2 + O(\tilde{k}^4). \tag{10.97}$$

2.3.6 Laplacian Operators

The laplacian operators described here are similar to the ones used for preprocessing as described in Section 2.2.3. The three laplacian masks that follow represent different approximations of the laplacian operator. Unlike compass masks, the laplacian masks are rotationally symmetric, which means edges at all orientations contribute to the result. They are applied by selecting one mask and convolving it with the image. The sign of the result (positive or negative) from two adjacent pixel locations provides directional information, and tells us which side of the edge is brighter.

LAPLACIAN MASKS

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

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. An easy way to picture the difference is to consider the effect each mask would have when applied to an area of constant value. The preceding convolution masks would return a value of zero. If we increase the cen-

have some negative gray

$$\frac{1}{9} \times \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

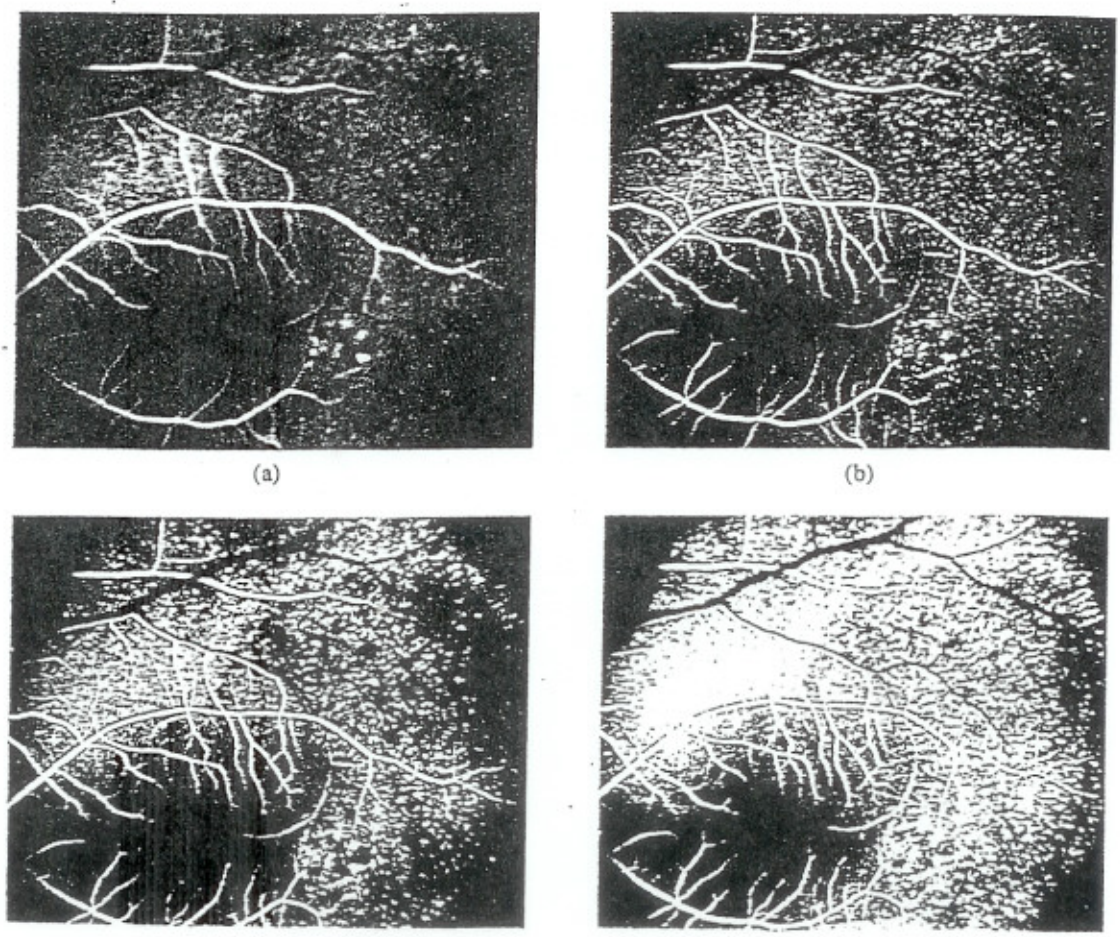
Figure 4.24 A basic highpass spatial filter.

Derivative filters

Averaging of pixels over a region tends to blur detail in an image. As averaging is analogous to integration, differentiation can be expected to have the opposite effect and thus sharpen an image.

$$\frac{1}{9} \times \begin{bmatrix} -1 & -1 & -1 \\ -1 & w & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 4.26 Mask used for high-boost spatial filtering. The value of the center weight is $w = 9A - 1$, with $A \geq 1$.



(a) (b)

$$\left[-1 \;\; 2 \;\; -1\right] \rightarrow -e^{-\frac{2\pi in}{N}} + 2 - e^{\frac{2\pi in}{N}}$$

$$= 2 - 2\cos\left(\frac{2\pi in}{N}\right)$$

So

$$H_n = 2\left(1 - \cos\left(\frac{2\pi in}{N}\right)\right)$$

$$\sim 2\left\{1 - \left[1 - \frac{1}{2}\left(\frac{2\pi n}{N}\right)^2\right]\right\} = \left(\frac{2\pi n}{N}\right)^2 \qquad n \ll N$$

$$H_{\frac{n}{2}} = 2\left[1 - \cos\left(\frac{\pi}{4}\right)\right] = 4$$

Similarly

$$\left[-1 \;\; w \;\; -1\right] \rightarrow -e^{-\frac{2\pi in}{N}} + w - e^{\frac{2\pi in}{N}}$$

$$= w - 2 + 2\left[1 - \cos\left(\frac{2\pi in}{N}\right)\right]$$

So

$$H_n \sim w - 2 + \left(\frac{2\pi n}{N}\right)^2 \qquad n \ll N$$

$$H_{\frac{n}{2}} = w + 2$$

$$G(\vartheta,\varphi)=\begin{pmatrix}-1 & -1 & -1\\ -1 & w & -1\\ -1 & -1 & a-1\end{pmatrix}\rightarrow$$

$$-\begin{bmatrix}e^{i(\vartheta+\varphi)}+e^{i\varphi}+e^{i(-\vartheta+\varphi)}\\ e^{i\varphi}-w+e^{-i\varphi}\\ e^{i(\vartheta-\varphi)}+e^{i\varphi}+e^{-i(\vartheta+\varphi)}\end{bmatrix}$$

$$=-\begin{bmatrix}\left(e^{i\varphi}+e^{-i\varphi}\right)\left(e^{i\vartheta}+e^{-i\vartheta}\right)+\\ e^{i\vartheta}+e^{-i\vartheta}+e^{i\varphi}+e^{-i\varphi}\end{bmatrix}+w$$

$$=-\left[2\cos\theta\bullet 2\cos\varphi+2\cos\theta+2\cos\varphi\right]+w$$

$$=w-2\left[2\cos\theta\cos\varphi+\cos\theta+\cos\varphi\right]$$

So

$$G(0,0)=w-8$$

$$G(\pi,\pi)=w$$

$$G(\frac{\pi}{2},\frac{\pi}{2})=w$$

$$G(\frac{\pi}{2},\pi)=w+2$$

same properties in all directions and is therefore invariant to rotation in the image. It is defined as

$$\nabla^2 g(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2} \tag{4.39}$$

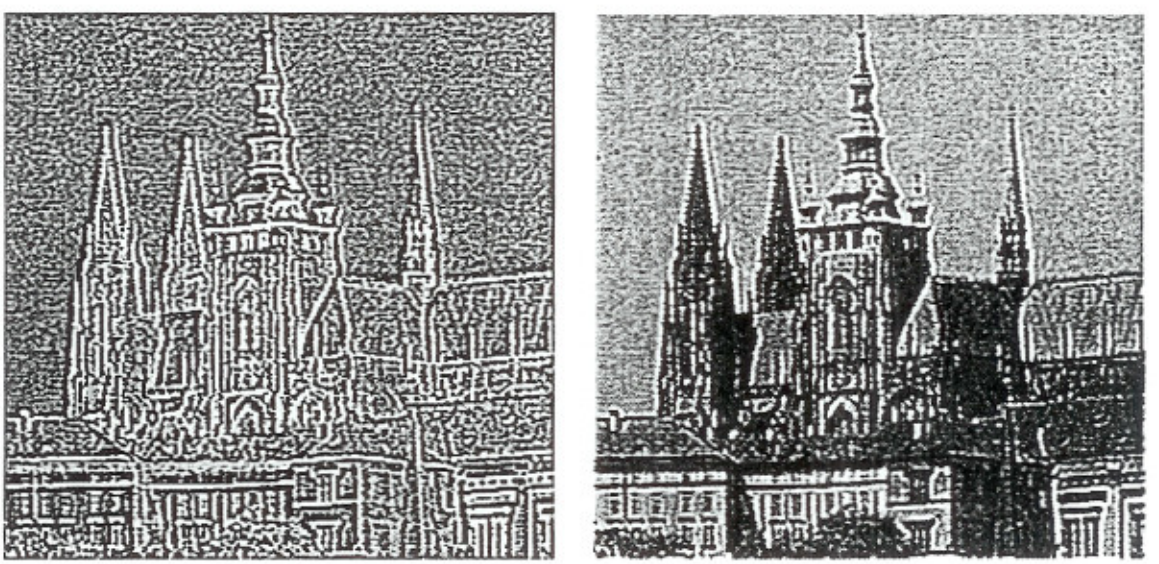
Image **sharpening** [Rosenfeld and Kak 82] has the objective of making edges steeper—the sharpened image is intended to be observed by a human. The sharpened output image f is obtained from the input image g as

$$f(i,j) = g(i,j) - CS(i,j) \tag{4.40}$$

where C is a positive coefficient which gives the strength of sharpening and $S(i,j)$ is a measure of the image function sheerness, calculated using a gradient operator. The Laplacian is very often used for this purpose. Figure 4.18 gives an example of image sharpening using a Laplacian.

Image sharpening can be interpreted in the frequency domain as well. We already know that the result of the Fourier transform is a combination of harmonic functions. The derivative of the harmonic function $\sin(nx)$ is $n\cos(nx)$; thus the higher the frequency, the higher the magnitude of its derivative. This is another explanation of why gradient operators enhance edges.

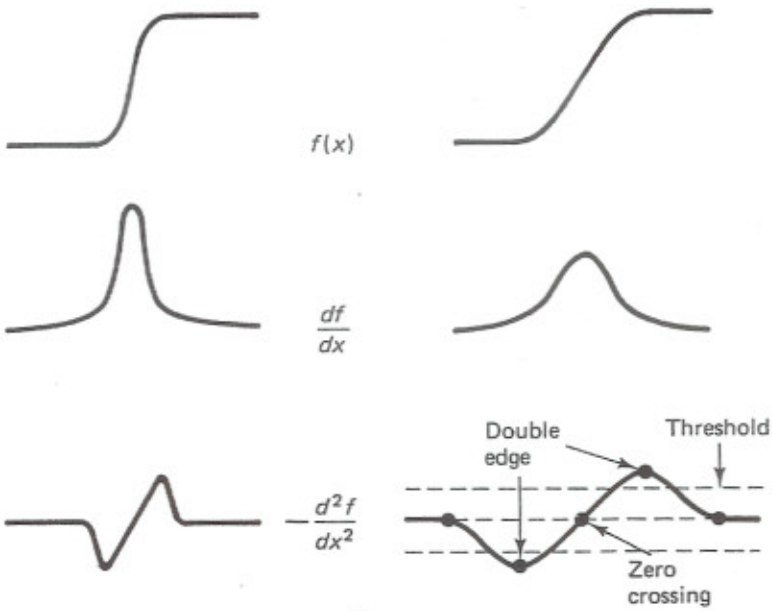
A similar image sharpening technique to that given in equation (4.40), called **unsharp masking**, is often used in printing industry applications [Jain 89]. A signal proportional to an unsharp image (e.g., heavily blurred by a smoothing operator) is subtracted from the original image.



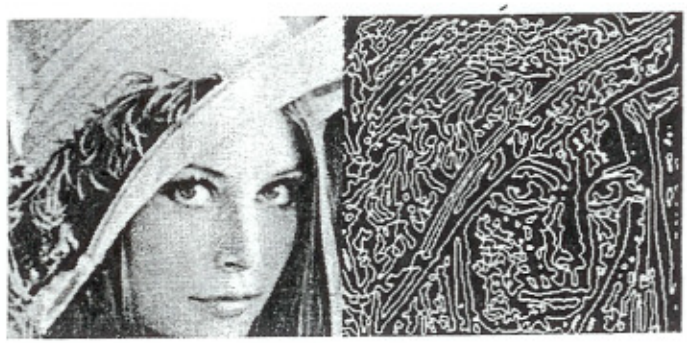
(a) (b)

Figure 4.18: *Laplace gradient operator: (a) Laplace edge image using the 8-connectivity mask; (b) sharpening using the Laplace operator [equation (4.40), $C = 0.7$]. Compare the sharpening effect with the original image in Figure 4.10a.*

A digital image is discrete in nature and so equations (4.37) and (4.38), containing derivatives, must be approximated by **differences**. The first differences of the image g in the vertical



(a) First and second derivatives for edge detection



(b) An image and its zero-crossings.

Figure 9.10 Edge detection via zero-crossings.

Table 9.4 gives three different discrete approximations of this operator. Figure 9.8d shows the edge extraction ability of the Laplace mask (2). Because of the second-order derivatives, this gradient operator is more sensitive to noise than those previously defined. Also, the thresholded magnitude of $\nabla^2 f$ produces double edges. For these reasons, together with its inability to detect the edge direction, the Laplacian as such is not a good edge detection operator. A better utilization of the Laplacian is to use its zero-crossings to detect the edge locations (Fig. 9.10). A generalized Laplacian operator, which approximates the Laplacian of Gaussian functions, is a powerful zero-crossing detector [13]. It is defined as

$$h(m, n) \triangleq c\left[1 - \frac{(m^2 + n^2)}{\sigma^2}\right] \exp\left(-\frac{m^2 + n^2}{2\sigma^2}\right) \tag{9.21}$$

where σ controls the width of the Gaussian kernel and c normalizes the sum of the elements of a given size mask to unity. Zero-crossings of a given image convolved with $h(m, n)$ give its edge locations. On a two-dimensional grid, a zero-crossing is said to occur wherever there is a zero-crossing in at least one direction.

LAPLACIAN

Find the zeros of the Laplacian

This gives sharp edges

The edges are closed loops

It doesn't give the magnitude of the edge

Hence, a jump from $-\varepsilon$ to ε is found to be an edge

$\Delta \to k^2$ So the Laplacian is more sensitive to noise than the gradient

Possible fix: Use the gradient or variance to eliminate false edges

is constant. Since $\nabla^2 f(n_1, n_2)$ is zero and we detect edges from zero-crossing points of $\nabla^2 f(n_1, n_2)$, any small perturbation of $f(n_1, n_2)$ is likely to cause false edge contours. One method to remove many of these false contours is to require that the local variance is sufficiently large at an edge point, as shown in Figure 8.34. The local variance $\sigma_f^2(n_1, n_2)$ can be estimated by

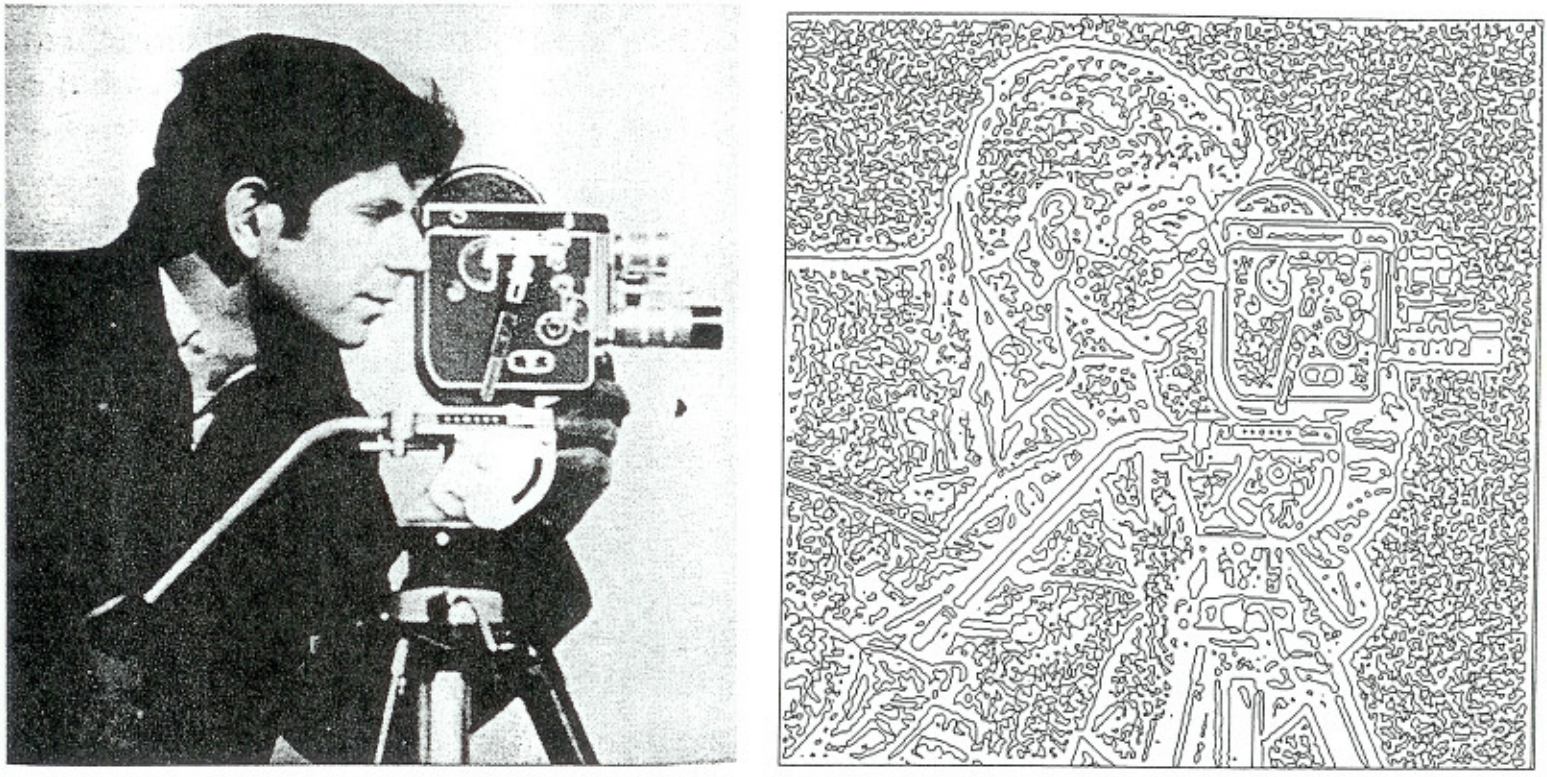
$$\sigma_f^2(n_1, n_2) = \frac{1}{(2M + 1)^2} \sum_{k_1=n_1-M}^{n_1+M} \sum_{k_2=n_2-M}^{n_2+M} [f(k_1, k_2) - m_f(k_1, k_2)]^2 \tag{8.16a}$$

where

$$m_f(n_1, n_2) = \frac{1}{(2M + 1)^2} \sum_{k_1=n_1-M}^{n_1+M} \sum_{k_2=n_2-M}^{n_2+M} f(k_1, k_2) \tag{8.16b}$$

with M typically chosen around 2. Since $\sigma_f^2(n_1, n_2)$ is compared with a threshold, the scaling factor $1/(2M + 1)^2$ in (8.16a) can be eliminated. In addition, the local variance σ_f^2 needs to be computed only for (n_1, n_2) which are zero-crossing points of $\nabla^2 f(n_1, n_2)$. Figure 8.35 shows the result of applying the system in Figure 8.34 to the image in Figure 8.33(a). Comparison of Figures 8.33(b) and 8.35 shows considerable reduction in the "false" edge contours.

The system in Figure 8.34 can be interpreted as a gradient-based method.



(a) (b)

Figure 8.33 Edge map obtained by a Laplacian-based edge detector. (a) Image of 512 × 512 pixels; (b) result of convolving the image in (a) with $h(n_1, n_2)$ in Figure 8.32(a) and then finding zero-crossing points.

ıre 2.3-5 Edge Detection Examples

a. Original image.

b. Sobel operator .

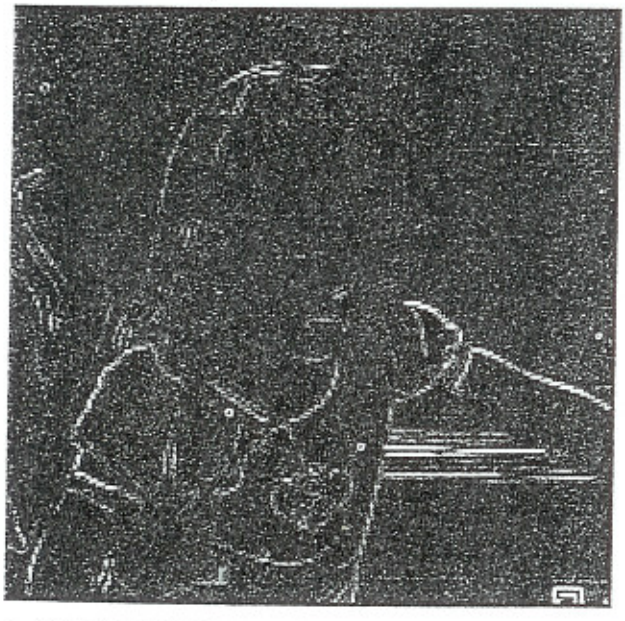
c. Prewitt operator.

d. Frei-Chen operator, edge subspace.

Figure 2.3-5 (Continued)

e. Laplacian operator.

f. Kirsch operator.



g. Roberts operator.

h. Robinson operator.

As previously mentioned, the objective metrics are often of limited use in pra applications, so we will take a subjective look at the results of the edge detectors human visual system is still superior, by far, to any computer vision system that ha been devised and is often used as the final judge in application development. Figur 5 shows the results of the edge detection operators. Here we see similar results fro the operators but the laplacian. This occurs because the laplacian returns positive negative numbers that get linearly remapped to 0 to 255 (for 8-bit display), w means that the background value of 0 is mapped to some intermediate gray level

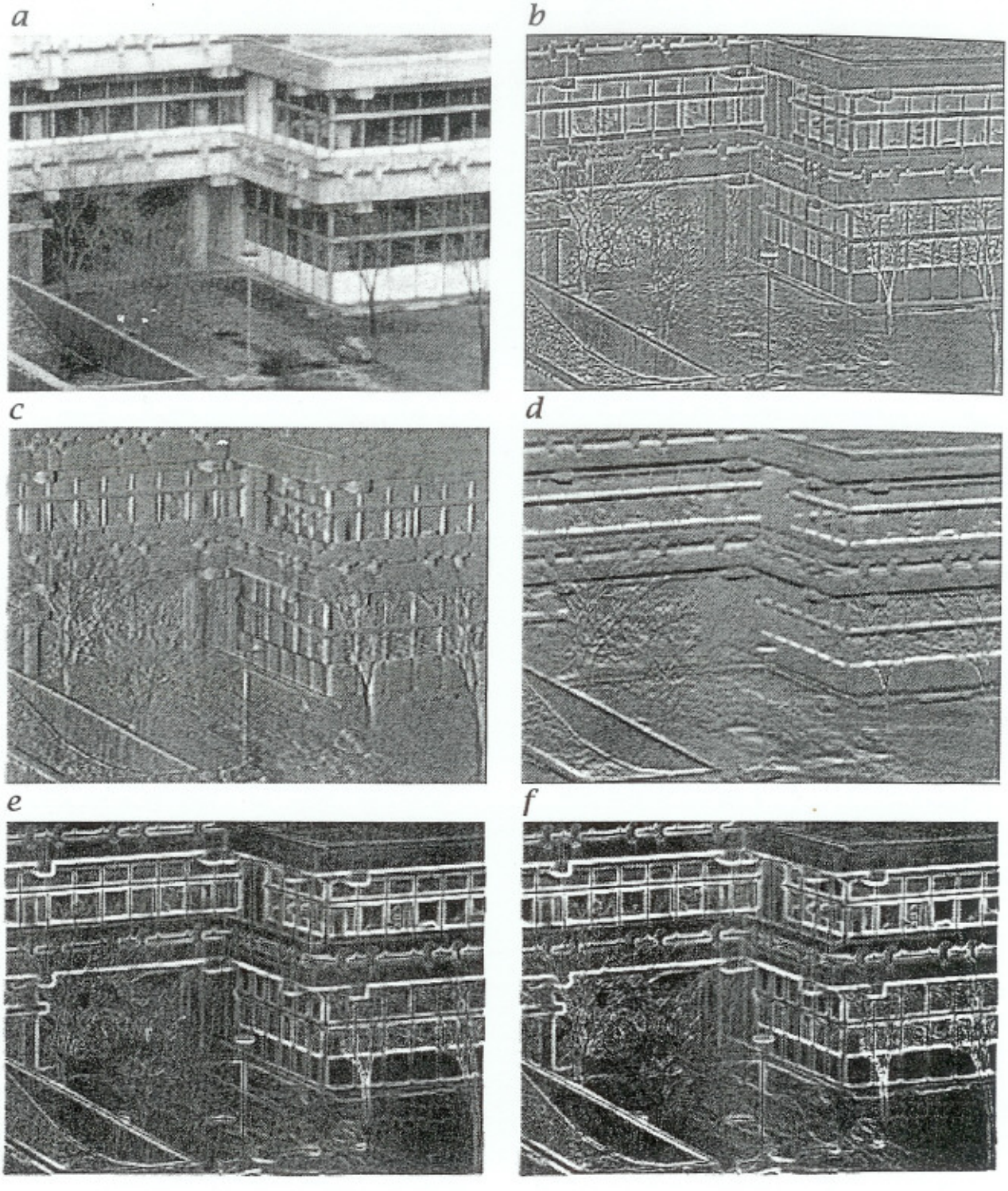
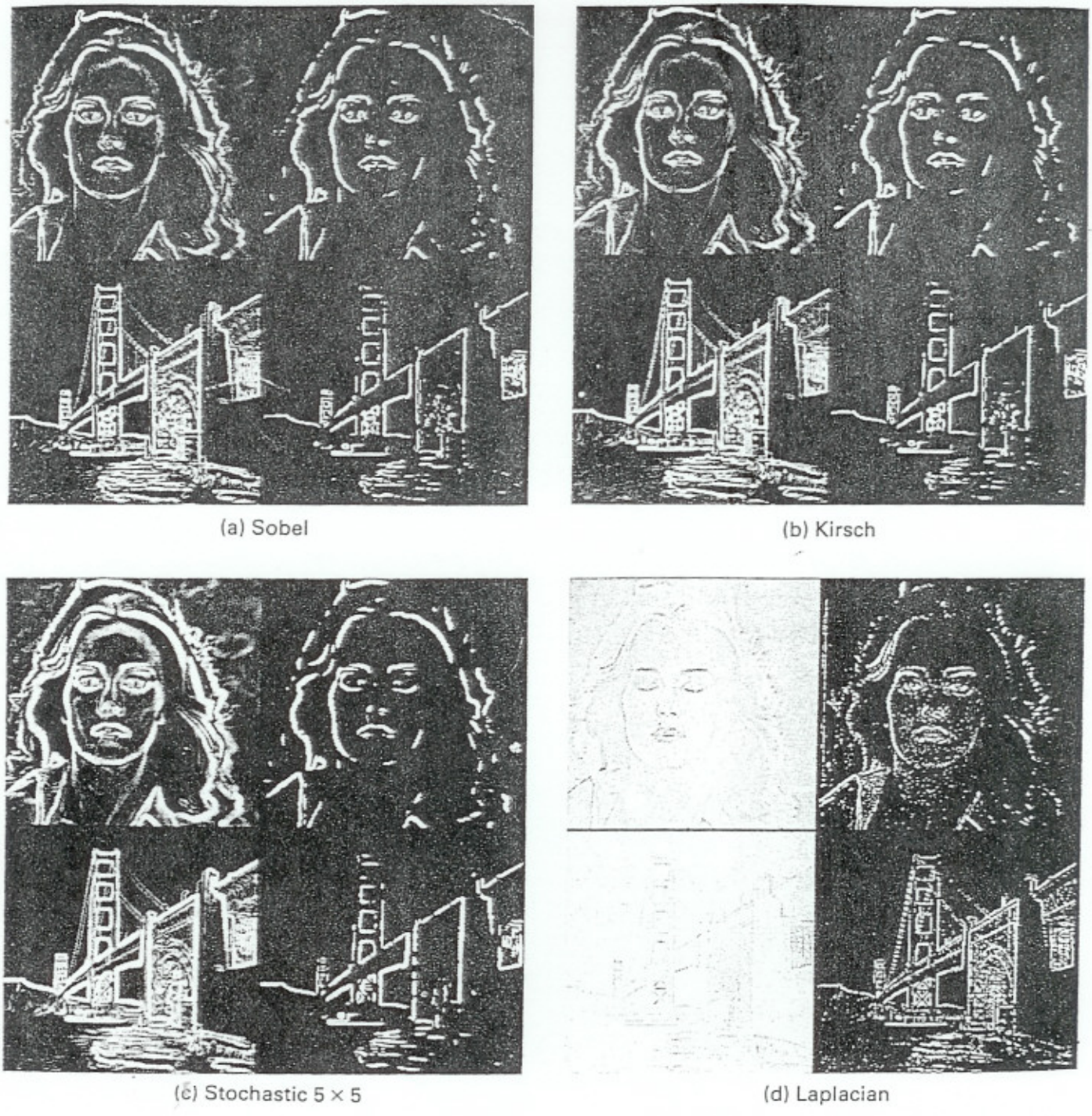


Figure 10.20: *Detection of edges by derivative filters (**exercise 10.7**): **a** Original image, **b** Laplacian operator $\mathcal{L}$, **c** horizontal derivative $\mathcal{D}_{2x}$, **d** vertical derivative $\mathcal{D}_{2y}$, **e** magnitude of the gradient $(\mathcal{D}_{2x} \cdot \mathcal{D}_{2x} + \mathcal{D}_{2x} \cdot \mathcal{D}_{2x})^{1/2}$, and **f** sum of the magnitudes of **c** and **d** after (10.70).*



(a) Sobel (b) Kirsch

(c) Stochastic 5 × 5 (d) Laplacian

Figure 9.8 Edge detection examples. In each case, gradient images (left), edge maps (right).

that 5 to 10% of pixels with largest gradients are declared as edges. Figure 9.8a shows the gradients and edge maps using the Sobel operator on two different images.

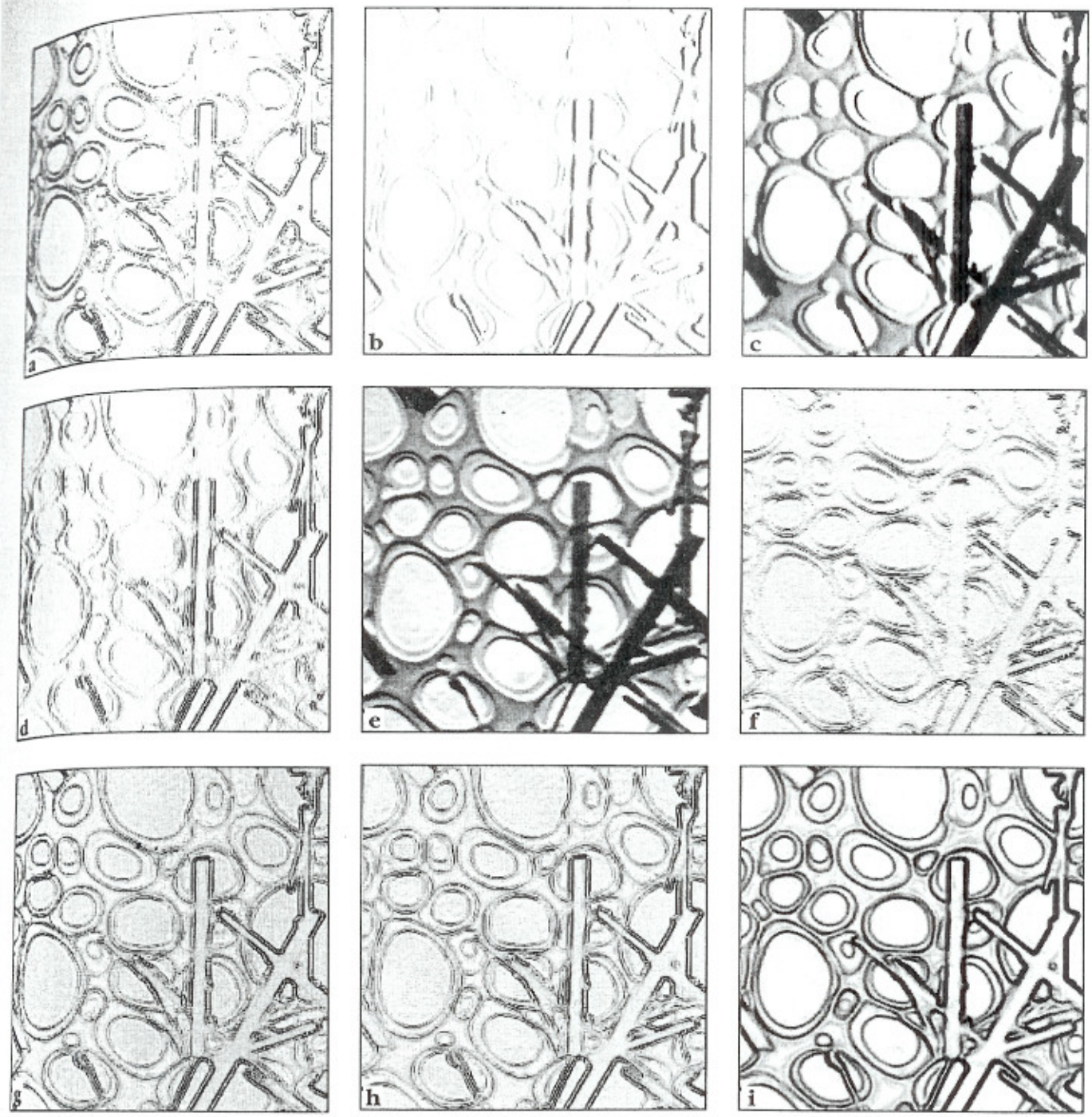


Figure 27. ***Edge enhancement of the image in Figure 26:***

a) Laplacian operator;
b) Roberts' Cross operator;
c) horizontal derivative, scaled to full grey scale range;
d) absolute value of image ***c****;*
e) vertical derivative, scaled to full grey scale range;
f) absolute value of image ***e****;*
g) sum of absolute values from images ***d*** *and* ***f****;*
h) maximum of values in images ***d*** *and* ***f****, pixel by pixel;*
i) Sobel operator (square root of sum of squares of values);
j) Kirsch operator.

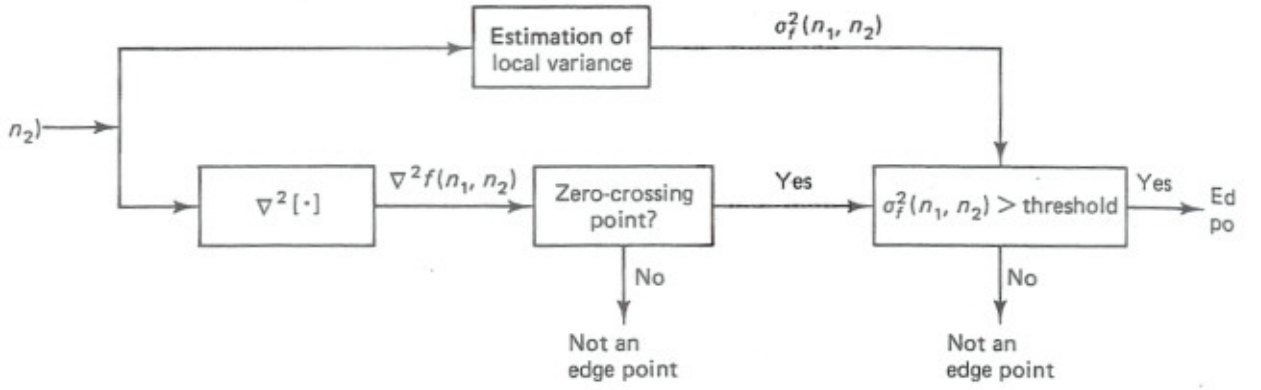


Figure 8.34 Laplacian-based edge detection system that does not produce many false edge contours.

The local variance $\sigma_f^2(n_1, n_2)$ is closely related to the gradient magnitude. Com paring $\sigma_f^2(n_1, n_2)$ with a threshold is similar to comparing the gradient magnitud with a threshold. Requiring that $\nabla^2 f(n_1, n_2)$ crosses zero at an edge can be inter preted as edge thinning. With this interpretation, we can implement the syster in Figure 8.34 by computing $\sigma_f^2(n_1, n_2)$ first and then by detecting the zero-crossing points of $\nabla^2 f(n_1, n_2)$ only at those points where $\sigma_f^2(n_1, n_2)$ is above the chosen threshold.

8.3.3 Edge Detection by Marr and Hildreth's Method

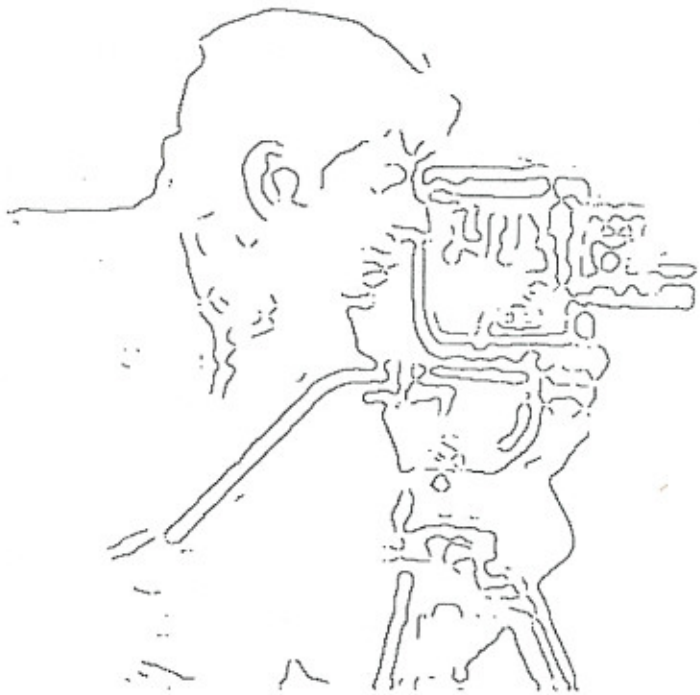
In the previous two sections, we discussed edge detection algorithms that produce one edge map from an input image. Marr and Hildreth [Marr and Hildreth; Marr] observed that significant intensity changes occur at different scales (resolution) in an image. For example, blurry shadow regions and sharply focused fine-detail

Figure 8.35 Edge map obtained by applying the system in Figure 8.34 to the image in Figure 8.33(a).

(a)

(b)

(c)



(d)

FIGURE 8 Zero crossings of $f * \nabla^2 g$ for several values of σ, with (d) also thresholded: (a) $\sigma = 1.0$, (b) $\sigma = 1.5$, (c) $\sigma = 2.0$, (d) $\sigma = 2.0$ and $T = 20$.

Gaussian Filter

$$g_c = e^{-\frac{x^2+y^2}{2\sigma^2}} = e^{-\frac{x^2}{2\sigma^2}} \bullet e^{-\frac{y^2}{2\sigma^2}}$$

Taking the Fourier transform we get

$$G = \mathbb{F}(g) = 2\pi e^{-\frac{\sigma^2}{2}\left(\omega^2{}_1+\omega^2{}_2\right)}$$

We now consier the Laplacian of the Gaussian (LOG)

$$h_c = \Delta g_c = \frac{x^2+y^2-2\sigma^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The Gaussian is a separable function

The Laplacian of the Gaussian is a sum of two separable functions

g_c has an infinite stencil

One can approximate g_c on a finite stencil

If the stencil is small use convolutions

If the stencil is large use FFTs

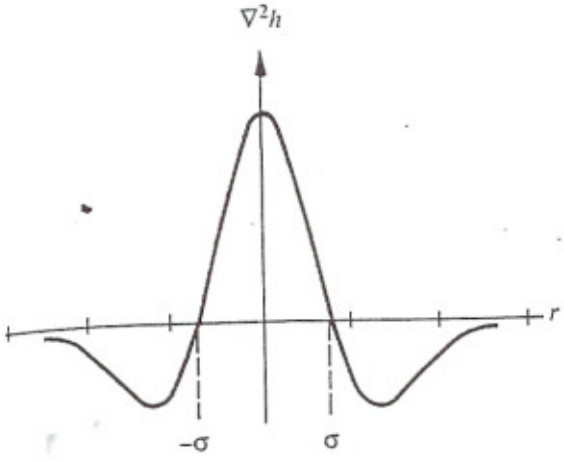
$\Delta(G*I)=G*\Delta(I)=\left(\Delta G\right)*I$

Where we find the Laplacian of G analytically

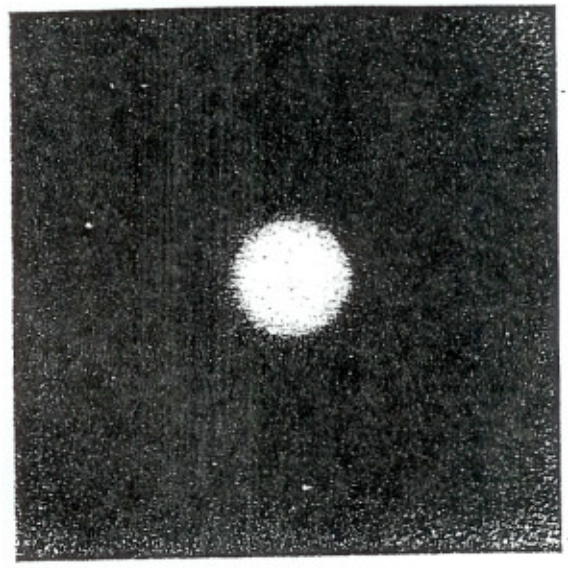
Example: Consider Fig. 7.9(a), which is a simple image of resolution 320 × 320 pixels. Figure 7.9(b) shows the result of convolving this image with the function $\nabla^2 h$. The value of σ in this case is 4. In this result black represents the most negative values, and white represents the most positive values; thus mid grays represent zeros. Figure 7.9(c) shows a binary image created by setting all negative values in Fig. 7.9(b) to black and all positive values to white. From

0	−1	0
−1	4	−1
0	−1	0

Figure 7.7 Mask used to compute the Laplacian.



(a)



(b)

Figure 7.8 (a) Cross section of $\nabla^2 h$; (b) $\nabla^2 h$ shown as an intensity function (image). (From

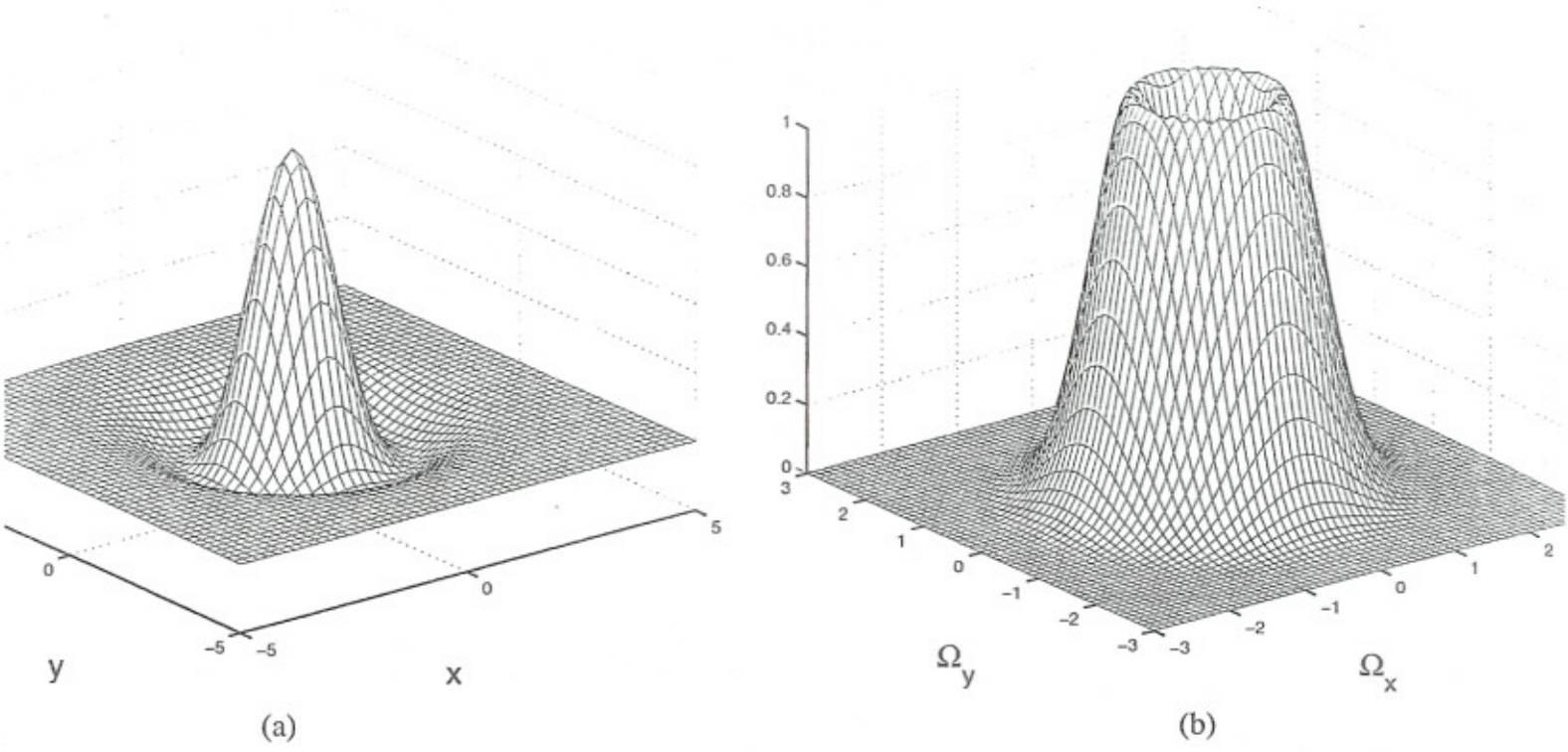


FIGURE 9 Plots of the LoG and its frequency response for $\sigma = 1$: (a) $-\nabla^2 g_c(x, y)$, the negative of Eq. (18); (b) $F\{\nabla^2 g_c(x, y)\}$, the bandpass-shaped frequency response of Eq. (18).

; goals: low rate of detection errors, good edge lo- d only a single detection response per edge. Canny : false-positive and false-negative detection errors ndesirable and so gave them equal weight. He fur- ! that each edge has nearly constant cross section on, but his general method includes a way to ef- with the cases of curved edges and corners. With ints, Canny determined the optimal 1-D edge de- step edge and showed that its impulse response can ıted fairly well by the derivative of a Gaussian.

ant action of Canny's edge detector is to prevent onses per true edge. Without this criterion, the edge detector would have an impulse response in truncated signum function. (The signum function for any positive argument and −1 for any negative ut this type of filter has high bandwidth, allowing ıre to produce several local maxima in the vicinity edge. The effect of the derivative of Gaussian is to iple responses by smoothing the truncated signum rmit only one response peak in the edge neighbor- ıoice of variance for the Gaussian kernel controls th and the amount of smoothing. This defines the neighborhood in which only a single peak is to he variance selected should be proportional to the

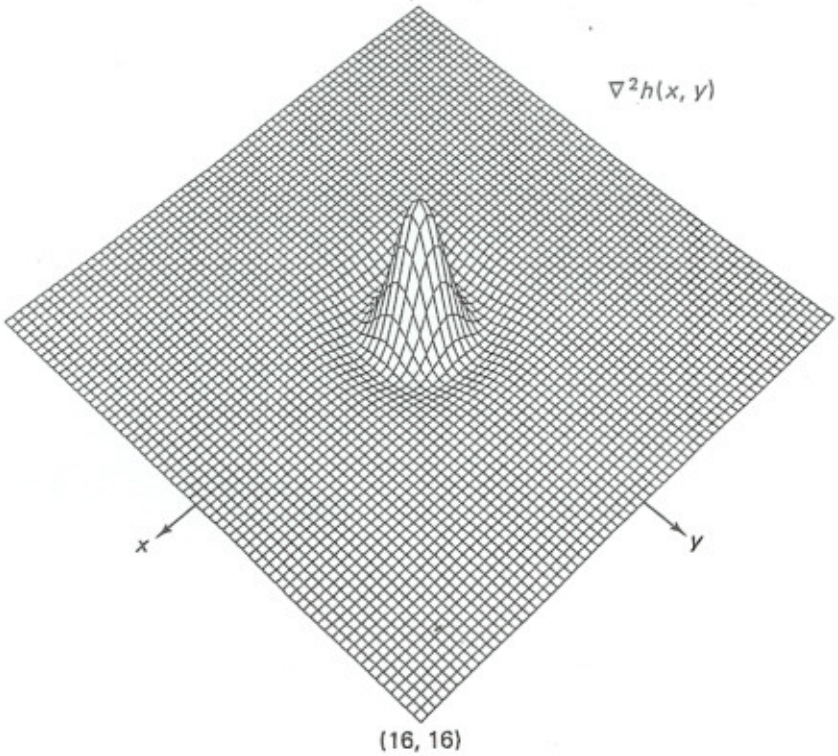
Canny's approach begins by smoothing the image Gaussian filter:

$$g_c(x, y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right).$$

One may sample and truncate Eq. (19) to produce a finite- filter, $g(n_1, n_2)$. At each pixel, Eq. (8) is used to estim gradient direction. From a set of prepared edge detectio masks having various orientations, the one oriented nea the gradient direction for the targeted pixel is then chosen. applied to the Gaussian-smoothed image, this filter produ estimate of gradient magnitude at that pixel. Next, the to suppress non-maxima of the gradient magnitude by a 3 × 3 neighborhood, comparing the magnitude at the pixel with those at interpolated positions to either side alc gradient direction.

The pixels that survive to this point are candidates f edge map. To produce an edge map from these candida els, Canny applies thresholding by gradient magnitude adaptive manner with hysteresis. An estimate of the noise image determines the values of a pair of thresholds, w upper threshold typically two or three times that of the

A candidate edge segment



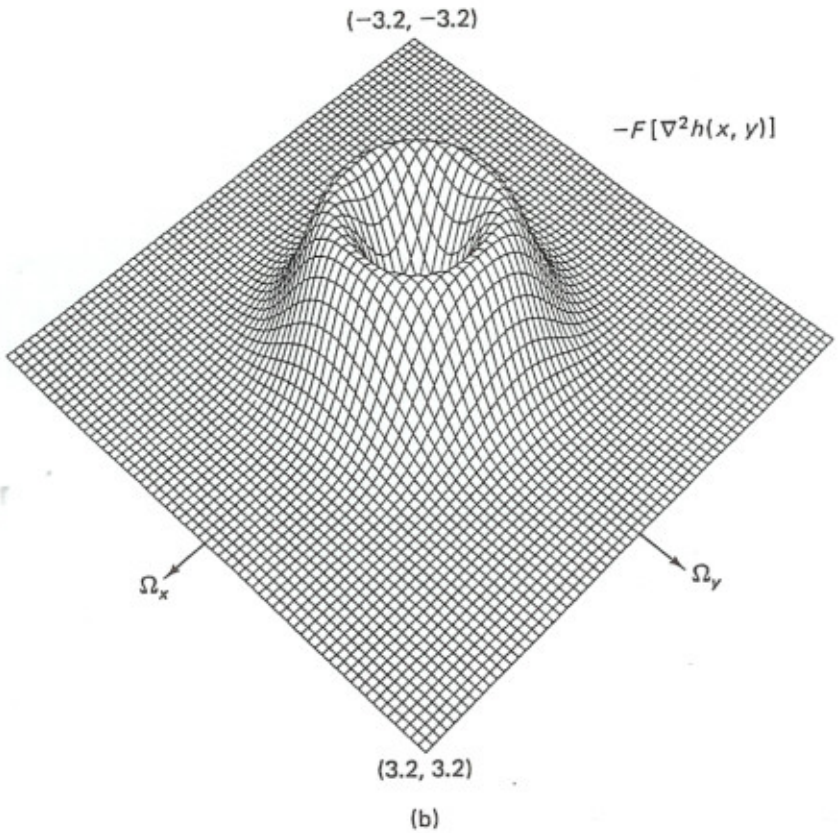


Figure 8.36 Sketch of (a) $\nabla^2 h(x, y)$ and (b) $-F[\nabla^2 h(x, y)]$ in Equation (8.19) for $\sigma^2 = 1$.

`h = fspecial('prewitt')` returns this 3-by-3 horizontal edge-finding and *y*-derivative approximation filter:

```
[  1   1   1
   0   0   0
  -1  -1  -1  ]
```

To find vertical edges, or for *x*-derivatives, use `-h'`.

`h = fspecial('laplacian',alpha)` returns a 3-by-3 filter approximating the two-dimensional Laplacian operator. The parameter `alpha` controls the shape of the Laplacian and must be in the range 0 to 1.0. `fspecial` uses the default value of 0.2 if you do not specify `alpha`.

`h = fspecial('log',n,sigma)` returns a rotationally symmetric Laplacian of Gaussian filter with standard deviation `sigma` (in pixels). `n` is a 1-by-2 vector specifying the number of rows and columns in `h`. (`n` can also be a scalar, in which case `h` is `n`-by-`n`.) If you do not specify the parameters, `fspecial` uses the default values of [5 5] for `n` and 0.5 for `sigma`.

`h = fspecial('average',n)` returns an averaging filter. `n` is a 1-by-2 vector specifying the number of rows and columns in `h`. (`n` can also be a scalar, in which case `h` is `n`-by-`n`.) If you do not specify `n`, `fspecial` uses the default value of [3 3].

`h = fspecial('unsharp',alpha)` returns a 3-by-3 unsharp contrast enhancement filter. `fspecial` creates the unsharp filter from the negative of the Laplacian filter with parameter `alpha`. `alpha` controls the shape of the Laplacian and must be in the range 0 to 1.0. `fspecial` uses the default value of 0.2 if you do not specify `alpha`.

Figure 34 shows an example at a very different scale. The specimen is a polished aluminum metal. The individual grains exhibit different brightnesses because their crystallographic lattices are randomly oriented in space so that the etching procedure used darkens some grains more than others. It is the grain boundaries that are usually important in studying metal structures, since the configuration of grain boundaries results from prior heat treatment and controls many mechanical properties. The human visual process detects the grain boundaries using its sensitivity to boundaries and edges. Most image analysis systems use a gradient operation, such as a Sobel, to perform a similar enhancement prior to measuring the grain boundaries.

In the example in **Figure 34**, another method has been employed. This is a variance operator, which calculates the sum

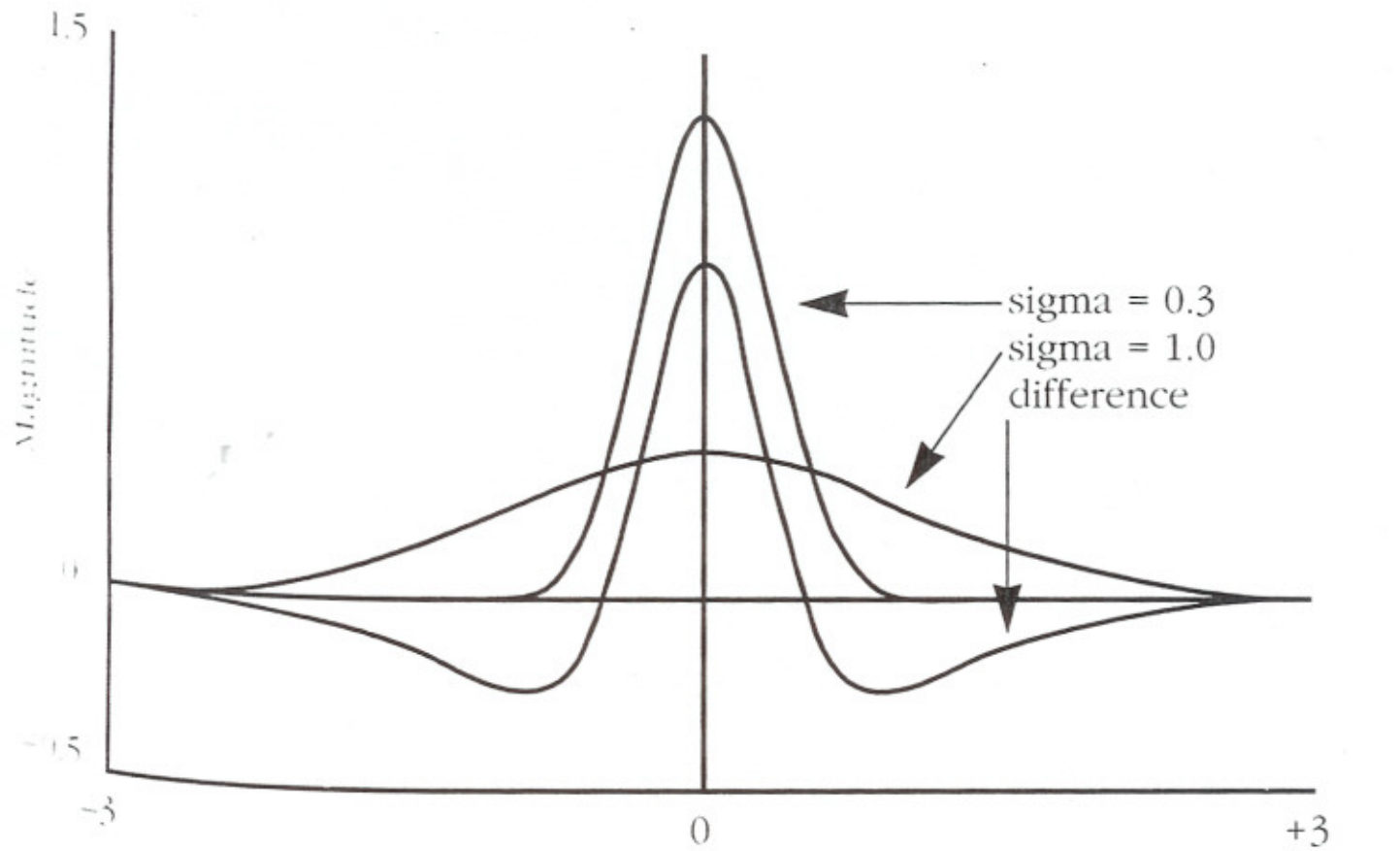


Figure 36. Application of Laplacian difference operations to the same image as in Figure 35:

a) *sharpening (addition of the 3×3 Laplacian to the original grey scale image);*

b) *5×5 Laplacian;*

c) *7×7 Laplacian;*

d) *9×9 Laplacian.*

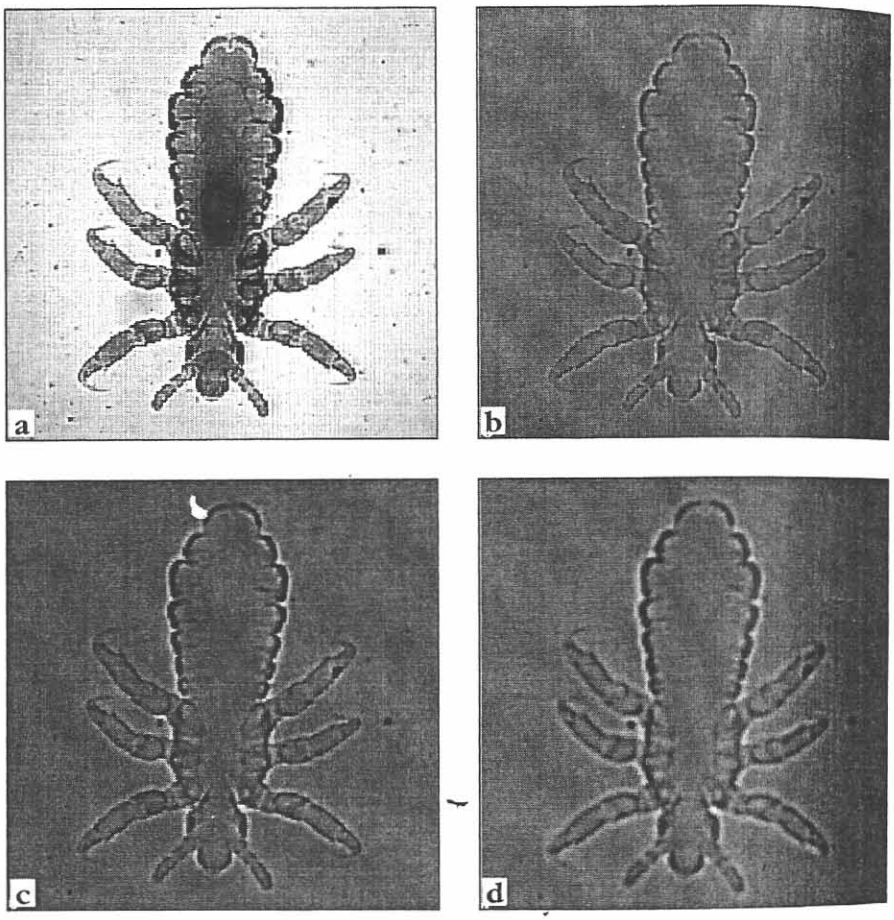


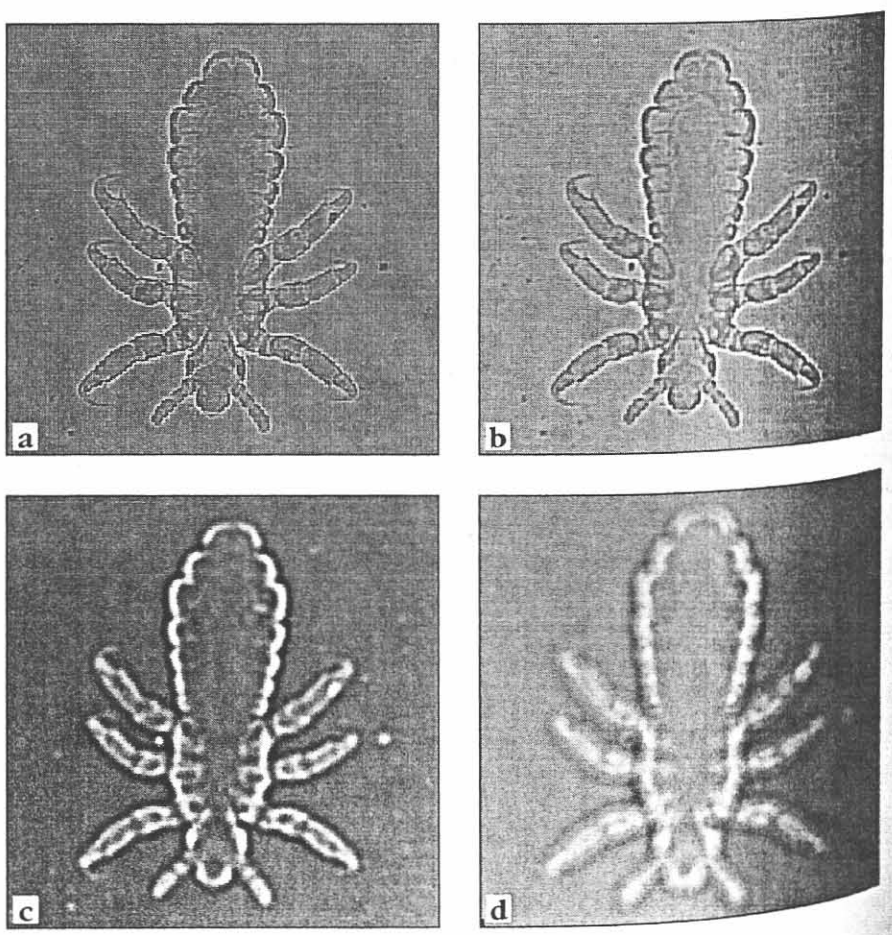
Figure 37. Difference of Gaussians applied to the same image as Figure 35:

a) *original image minus 3×3 Gaussian smooth;*

b) *difference between smoothing with σ = 0.625 and σ = 1.0 pixels;*

c) *difference between smoothing with σ = 1.6 and σ = 1.0 pixels;*

d) *difference between smoothing with σ = 4.1 and σ = 2.6 pixels.*



the upper left, if LoG/DoG image values of both polarities occur in the 2×2 window; no edge label would be given if values within the window are either all positive or all negative. Another post-processing step to avoid detection of zero-crossings corresponding to nonsignificant edges in regions of almost constant gray-level would admit only those zero-crossings for which there is sufficient edge evidence from a first-derivative edge detector. Figure 4.21 provides several examples of edge detection using zero crossings of the second derivative.

Many other approaches improving zero-crossing performance can be found in the literature [Qian and Huang 94, Mehrotra and Shiming 96]; some of them are used in pre-processing [Hardie and Boncelet 95] or post-processing steps [Alparone et al. 96].

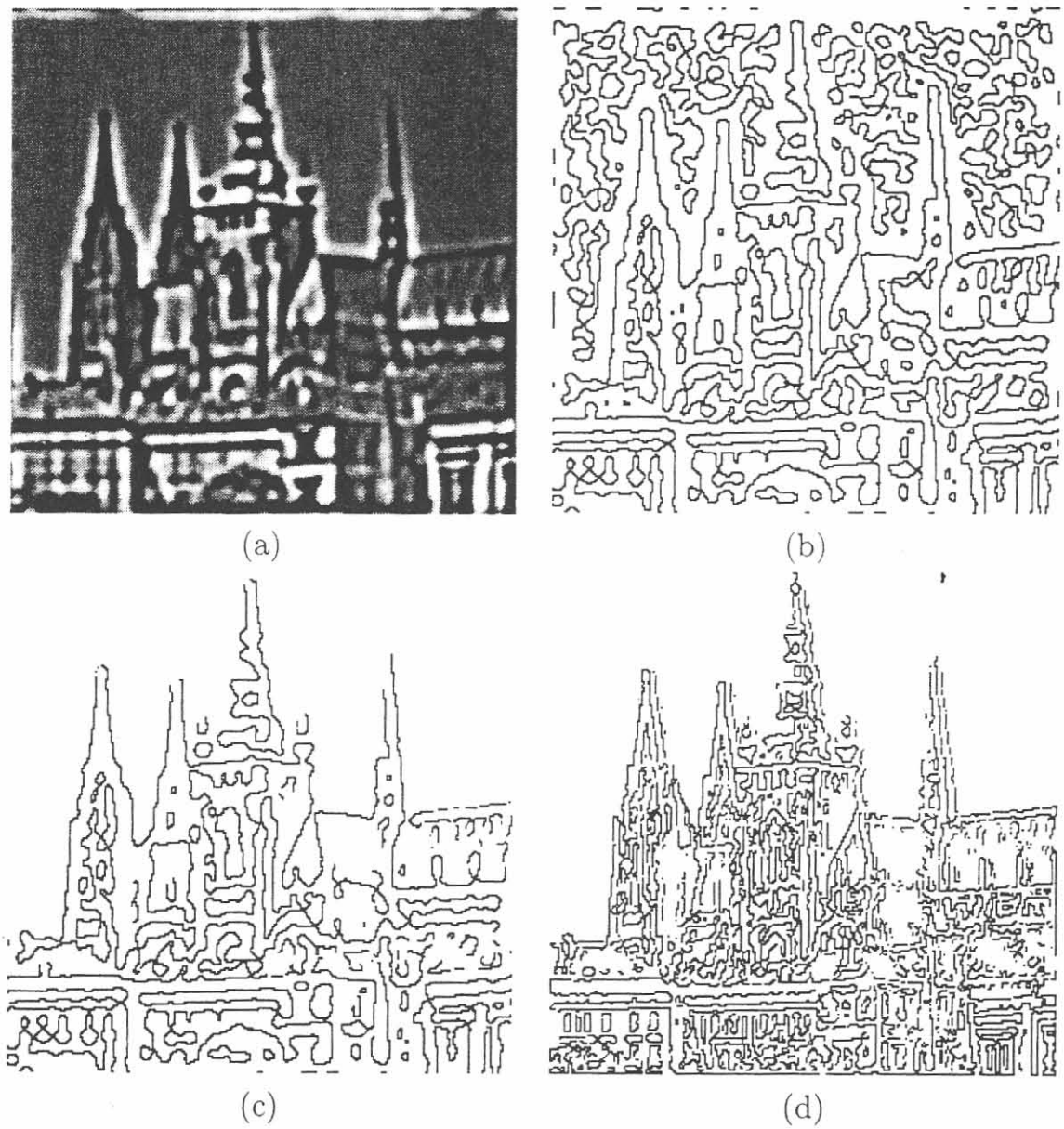


Figure 4.21: *Zero-crossings of the second derivative, see Figure 4.10a for the original image: (a) DoG image ($\sigma_1 = 0.10, \sigma_2 = 0.09$), dark pixels correspond to negative DoG values, bright pixels represent positive DoG values; (b) zero-crossings of the DoG image; (c) DoG zero-crossing edges after removing edges lacking first-derivative support; (d) LoG zero-crossing edges ($\sigma = 0.20$) after removing edges lacking first-derivative support—note different scale of edges due to different Gaussian smoothing parameters.*

Marr-Hildreth

This is based on a biological model.

1. convolve picture with a Gaussian
2. take the Laplacian of (1

$$\Delta\left(G * f\right) = \Delta\left(G\right) * f$$

i.e. take LOG convolved with the image

3. Find the zeros of (2)
   Note the Gaussian depends on σ

   Instead of the Laplacian of the Gaussian one can use the difference of Gaussians with different σ

   A large $\sigma \Rightarrow$ a large stencil and coarse features
   A small $\sigma \Rightarrow$ a small stencil and fine features

Marr studied the literature on mammalian visual systems and summarized these in five major points:

1. In natural images, features of interest occur at a variety of scales. No single operator can function at all of these scales, so the result of operators at each of many scales should be combined.
2. A natural scene does not appear to consist of diffraction patterns or other wavelike effects, and so some form of local averaging (*smoothing*) must take place.
3. The optimal smoothing filter that matches the observed requirements of biological vision (smooth and localized in the spatial domain and smooth and band-limited in the frequency domain) is the *Gaussian*.
4. When a change in intensity (an edge) occurs, there is an extreme value in the first derivative or intensity. This corresponds to a *zero crossing* in the second derivative.
5. The orientation-independent differential operator of lowest order is the *Laplacian*.

Each of these points is either supported by the observation of vision systems or derived mathematically, but the overall grounding of the resulting edge detector is still a little loose. However, based on the five points above, an edge-detection algorithm can be stated as follows:

1. Convolve the image I with a two-dimensional Gaussian function.
2. Complete the Laplacian of the convolved image; call this L.
3. Edges pixels are those for which there is a zero crossing in L.

The results of convolutions with Gaussians having a variety of standard deviations are combined to form a single edge image. The standard deviation is a measure of scale in this instance.

The algorithm is not difficult to implement, although it is more difficult than the methods seen so far. A convolution in two dimensions can be expressed as:

$$I^*G(i,j) = \sum_n \sum_m I(n,m)G(i - n, j - m) \qquad \text{(EQ 1.13)}$$

The function G being convolved with the image is a two-dimensional Gaussian, which is:

$$G_\sigma(x,y) = \sigma^2 e^{\frac{-(x2+y2)}{\sigma 2}} \qquad \text{(EQ 1.14)}$$

To perform the convolution on a digital image, the Gaussian must be sampled to create a small two-dimensional image. After the convolution, the Laplacian operator can be applied. This is:

Hildreth argue that edge maps of different scales contain important information about physically significant parameters. The visual world is made of elements such as contours, scratches, and shadows, which are highly localized at their own scale. This localization is also reflected in such physically important changes as reflectance change and illumination change. If the same edge is present in a set of edge maps of different scale, it represents the presence of an image intensity change due to a single physical phenomenon. If an edge is present in only one edge map, one reason may be that two independent physical phenomena are operating to produce intensity changes in the same region of the image.

To bandlimit an image at different cutoff frequencies, the impulse response $h(x, y)$ and frequency response $H(\Omega_x, \Omega_y)$ of the lowpass filter proposed [Marr and Hildreth; Canny] is Gaussian-shaped and is given by

$$h(x, y) = e^{-(x^2+y^2)/(2\pi\sigma^2)} \tag{8.17a}$$

$$H(\Omega_x, \Omega_y) = 2\pi^2\sigma^2 e^{-\pi\sigma^2(\Omega_x^2+\Omega_y^2)/2} \tag{8.17b}$$

where σ determines the cutoff frequency with larger σ corresponding to lower cutoff frequency. The choice of Gaussian shape is motivated by the fact that it is smooth and localized in both the spatial and frequency domains. A smooth $h(x, y)$ is less likely to introduce any changes that are not present in the original shape. A more localized $h(x, y)$ is less likely to shift the location of edges.

From the smoothed images, edges can be detected by using the edge detection algorithms discussed in the previous two sections. Depending on which method is used, the lowpass filtering operation in (8.17) and the partial derivative operation used for edge detection may be combined. For example, noting that $\nabla^2[\cdot]$ and convolution $*$ are linear, we obtain

$$\begin{aligned} \nabla^2(f(x, y) * h(x, y)) &= f(x, y) * [\nabla^2 h(x, y)] \\ &= f(x, y) * \left[\frac{\partial^2 h(x, y)}{\partial x^2} + \frac{\partial^2 h(x, y)}{\partial y^2}\right]. \end{aligned} \tag{8.18}$$

For the Gaussian function $h(x, y)$ in (8.17), $\nabla^2 h(x, y)$ and its Fourier transform are given by

$$\nabla^2 h(x, y) = \frac{e^{-(x^2+y^2)/(2\pi\sigma^2)}}{(\pi\sigma^2)^2}(x^2 + y^2 - 2\pi\sigma^2) \tag{8.19a}$$

$$F[\nabla^2 h(x, y)] = -2\pi^2\sigma^2 e^{-\pi\sigma^2(\Omega_x^2+\Omega_y^2)/2}(\Omega_x^2 + \Omega_y^2). \tag{8.19b}$$

Marr and Hildreth chose, for simplicity, to detect edges by looking for zero-crossing points of $\nabla^2 f(x, y)$. Bandlimiting $f(x, y)$ tends to reduce noise, thus reducing the noise sensitivity problem associated with detecting zero-crossing points. The functions $\nabla^2 h(x, y)$ and $-F[\nabla^2 h(x, y)]$ in (8.19) are sketched in Figure 8.36. Clearly, computing $f(x, y) * \nabla^2 h(x, y)$ is equivalent to bandpass filtering $f(x, y)$ where σ^2 in (8.19) is a parameter that controls the bandwidth of the bandpass filter. For a sequence $f(n_1, n_2)$, one approach is to simply replace x and y in (8.19) with n_1 and n_2.

Figure 8.37 shows an example of the approach under discussion. Figures 8.37(a), (b), and (c) show three images obtained by blurring the original image in Figure 8.33(a) with $h(n_1, n_2)$ obtained by replacing x and y of $h(x, y)$ in (8.17) with n_1 and n_2 with $\sigma^2 = 4$, 16, and 36, respectively. Figures 8.37(d), (e), and (f) show the images obtained by detecting zero crossings of $f(n_1, n_2) * \nabla^2 h(x, y)|_{x=n_1, y=n_2}$ with $\nabla^2 h(x, y)$ given by (8.19a) for $\sigma^2 = 4$, 16, and 36, respectively. Marr and Hildreth used the edge maps of different scales, such as those in Figures 8.37(d) (e), and (f) for object representation in their image understanding work.

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \qquad \text{(EQ 1.15)}$$

and could be computed using differences. However, because order does not matter in this case, we could compute the Laplacian of the Gaussian analytically and sample that function, creating a convolution mask that can be applied to the image to yield the same result. The Laplacian of a Gaussian (LoG) is:

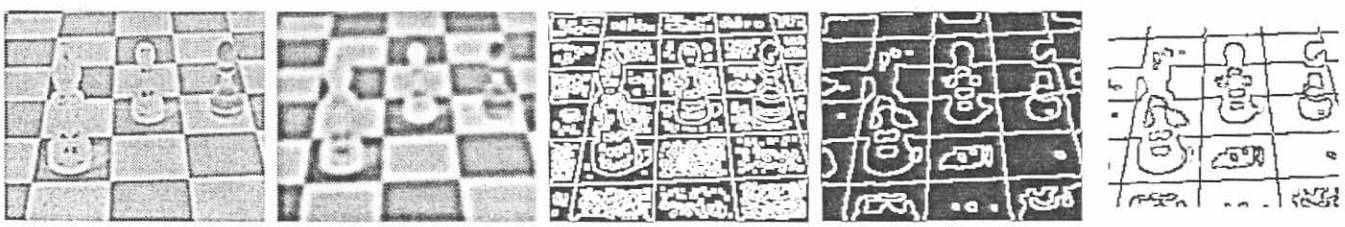
$$\nabla^2 G_\sigma = \left(\frac{r^2 - 2\sigma^2}{\sigma^4}\right) e^{\left(\frac{-r}{2\sigma^2}\right)} \qquad \text{(EQ 1.16)}$$

where $r = \sqrt{x^2 + y^2}$. This latter approach is the one taken in the C code implementing this operator, which appears at the end of this chapter.

This program first creates a two-dimensional, sampled version of the Laplacian of the Gaussian (called lgau in the function marr) and convolves this in the obvious way with the input image (function convolution). Then the zero crossings are identified, and pixels at those positions are marked.

A zero crossing at a pixel P implies that the values of the two opposing neighboring pixels in some direction have different signs. For example, if the edge through P is vertical, then the pixel to the left of P will have a different sign than the one to the right of P. There are four cases to test: up/down, left/right, and the two diagonals. This test is performed for each pixel in the Laplacian of the Gaussian by the function zero_cross.

In order to ensure that a variety of scales are used, the program uses two different Gaussians, and selects the pixels that have zero crossings in both scales as output edge pixels. More than two Gaussians could be used, of course. The program accepts a standard deviation value σ as a parameter, either from the command line or from the parameter file marr.par. It then uses both $\sigma + 0.8$ and $\sigma - 0.8$ as standard deviation values, does two convolutions, locates two sets of zero

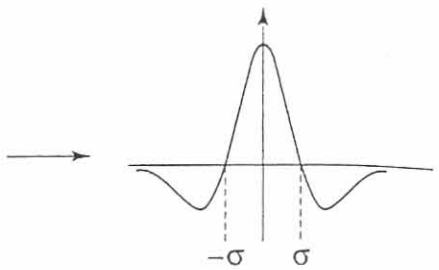


(a) (b) (c) (d) (e)

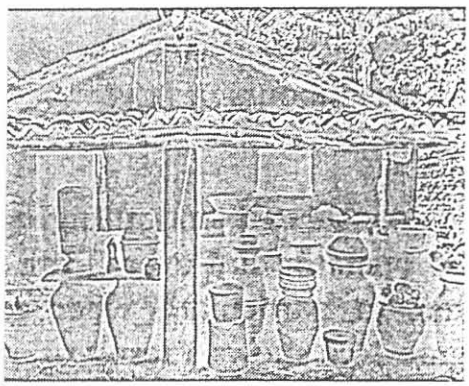
Figure 1.11 Steps in the computation of the Marr-Hildreth edge detector. (a) Convolution of the original image with the Laplacian of a Gaussian having $\sigma = 1.2$. (b) Convolution of the image with the Laplacian of a Gaussian having $\sigma = 2.8$. (c) Zero crossings found in (a). (d) Zero crossings found in (b). (e) Result, found by using zero crossings common to both.

.7
reth edge

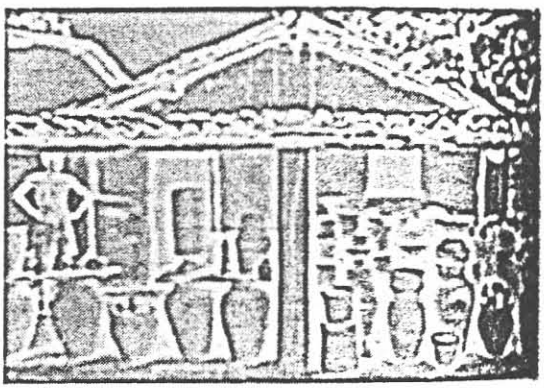
0	0	0	−1	−1	−2	−1	−1	0	0	0
0	0	−2	−4	−8	−9	−8	−4	−2	0	0
0	−2	−7	−15	−22	−23	−22	−15	−7	−2	0
−1	−4	−15	−24	−14	−1	−14	−24	−15	−4	−1
−1	−8	−22	−14	52	103	52	−14	−22	−8	−1
−2	−9	−23	−1	103	178	103	−1	−23	−9	−2
−1	−8	−22	−14	52	103	52	−14	−22	−8	−1
−1	−4	−15	−24	−14	−1	−14	−24	−15	−4	−1
0	−2	−7	−15	−22	−23	−22	−15	−7	−2	0
0	0	−2	−4	−8	−9	−8	−4	−2	0	0
0	0	0	−1	−1	−2	−1	−1	0	0	0



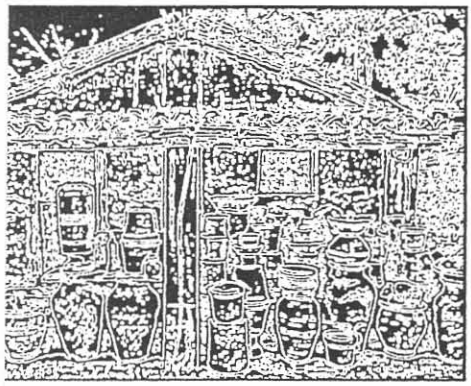
Digital implementation of the Man-Hildreth filter



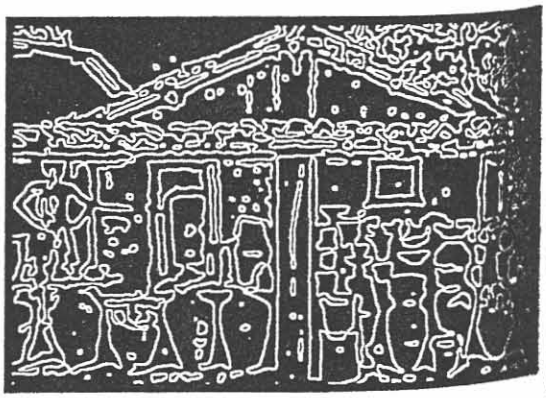
After convolulution of image with filter of 6 = 1.5



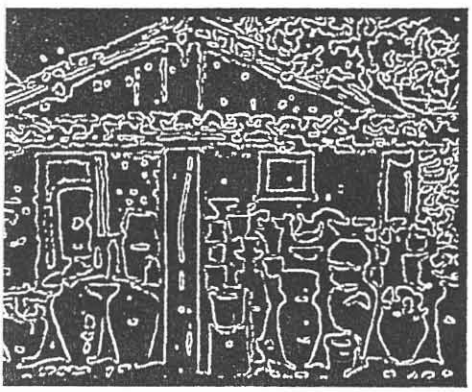
After convolulution of image with filter of 6 = 5



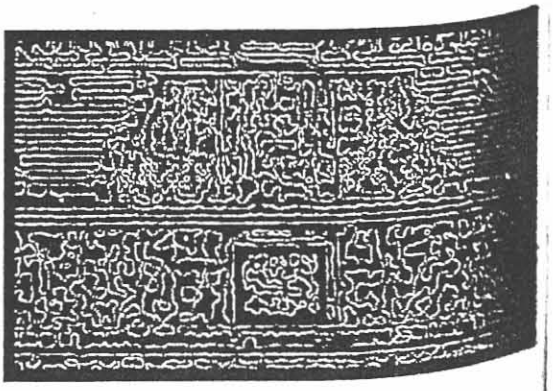
Zero crossings from above image



Zero crossings from above image



Zero crossing common to both



Result from the 'difficult' image

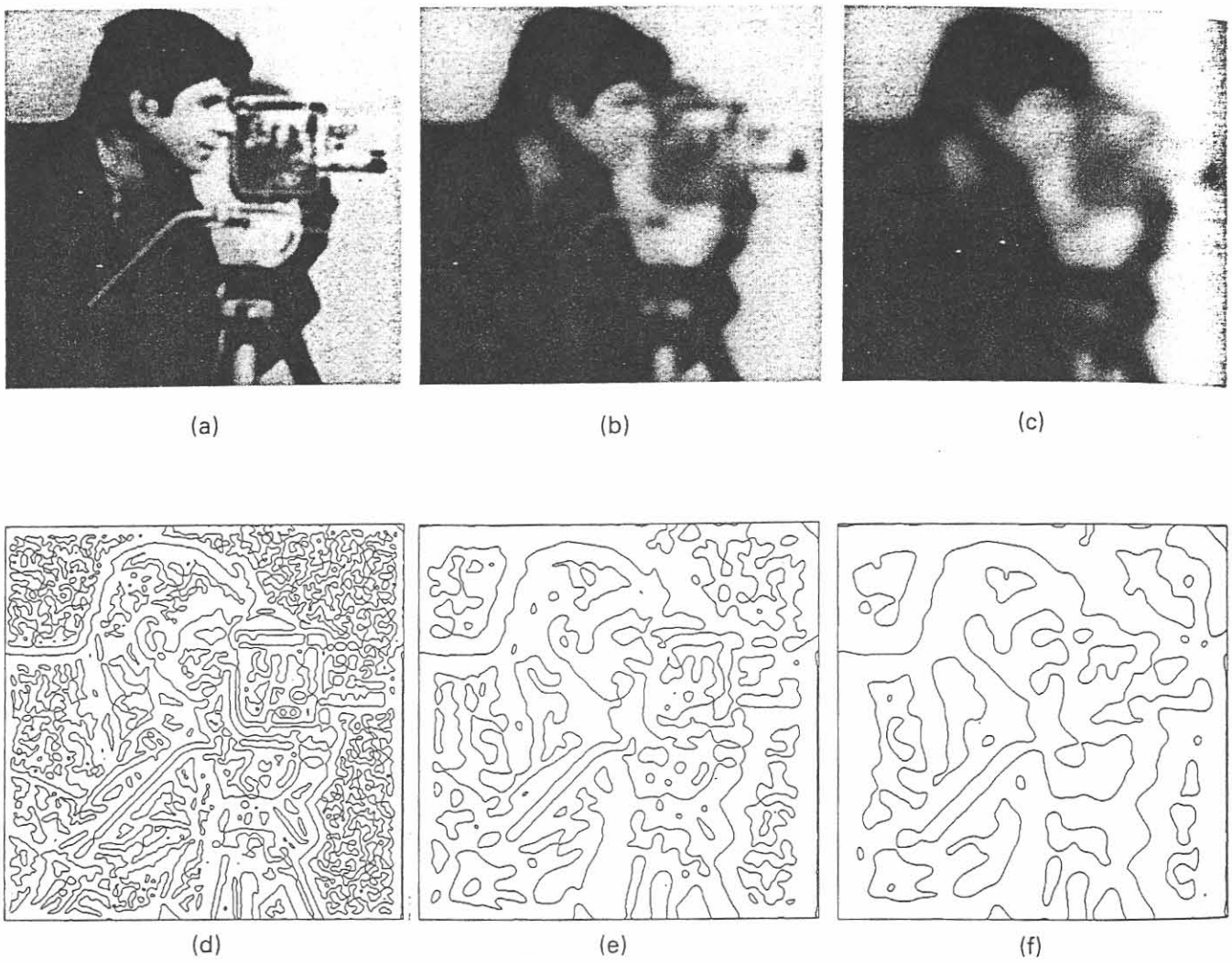


Figure 8.37 Edge maps obtained from lowpass filtered image. Blurred image with (a) $\sigma^2 = 4$; (b) $\sigma^2 = 16$; (c) $\sigma^2 = 36$. Result of applying Laplacian-based algorithm to the blurred image; (d) $\sigma^2 = 4$; (e) $\sigma^2 = 16$; (f) $\sigma^2 = 36$.

tinuous and has elliptical cross sections. The elliptical shape is chosen because of the small number of parameters involved in its characterization and because of some empirical evidence that it leads to a good estimate of percentage of stenosis. The 1-D cross section of $v(n_1, n_2)$, which consists of one blood vessel, is totally specified by three parameters, two representing the blood vessel boundaries and one related to the x-ray attenuation coefficient of iodine. The continuity of the vessel is guaranteed by fitting a cubic spline function to the vessel boundaries. The background $p(n_1, n_2)$ is modeled by a 2-D low-order polynomial. Low-order polynomials are very smooth functions, and their choice is motivated by the observation that objects in the background, such as tissue and bone, are much bigger than the blood vessels. The blurring function $g(n_1, n_2)$ is modeled by a known 2-D Gaussian function that takes into account the blurring introduced at various stages of the imaging process. The noise $w(n_1, n_2)$ is random background noise and is assumed to be white. The parameters in the model of $f(n_1, n_2)$ are the

4.3.5 Canny edge detection

Canny proposed a new approach to edge detection [Canny 83, Brady 84, Canny 86] that is optimal for step edges corrupted by white noise. The optimality of the detector is related to three criteria.

- The **detection** criterion expresses the fact that important edges should not be missed and that there should be no spurious responses.
- The **localization** criterion says that the distance between the actual and located position of the edge should be minimal.
- The **one response** criterion minimizes multiple responses to a single edge. This is partly covered by the first criterion, since when there are two responses to a single edge, one of them should be considered as false. This third criterion solves the problem of an edge corrupted by noise and works against non-smooth edge operators [Rosenfeld and Thurston 71].

Canny's derivation of a new edge detector is based on several ideas.

1. The edge detector was expressed for a 1D signal and the first two optimality criteria. A closed-form solution was found using the calculus of variations.
2. If the third criterion (multiple responses) is added, the best solution may be found by numerical optimization. The resulting filter can be approximated effectively with error less than 20% by the first derivative of a Gaussian smoothing filter with standard deviation σ [Canny 86]; the reason for doing this is the existence of an effective implementation. There is a strong similarity here to the Marr-Hildreth edge detector [Marr and Hildreth 80], which is based on the Laplacian of a Gaussian—see Section 4.3.3.
3. The detector is then generalized to two dimensions. A step edge is given by its position, orientation, and possibly magnitude (strength). It can be shown that convolving an image with a symmetric 2D Gaussian and then differentiating in the direction of the gradient (perpendicular to the edge direction) forms a simple and effective directional operator (recall that the Marr-Hildreth zero-crossing operator does not give information about edge direction, as it uses a Laplacian filter).

   Suppose G is a 2D Gaussian [equation (4.52)] and assume we wish to convolve the image with an operator G_n which is a first derivative of G in the direction $\mathbf{n}$.

$$G_n = \frac{\partial G}{\partial \mathbf{n}} = \mathbf{n} \cdot \nabla G \tag{4.62}$$

   The direction $\mathbf{n}$ should be oriented perpendicular to the edge. Although this direction is not known in advance, a robust estimate of it based on the smoothed gradient direction

Canny

1. Preliminary edge detection and non-maximum suppression

- Use derivative of Gaussian convolved with the image
- Check the gradients at neighboring points

  or else

- Take second derivative *normal* to the edge using a preliminary guess for the location of the edge

  Find the zeros of the second derivative $\frac{\partial^2 f}{\partial n^2}$

$$\frac{\partial^2 f}{\partial n^2} = \nabla\left(\nabla f \bullet n\right)\bullet n$$

$$= \frac{f^2{}_x f_{xx} + 2 f_x f_y f_{xy} + f^2{}_y f_{yy}}{\sqrt{f_x{}^2 + f_y{}^2}}$$

2. Use hysteresis with two thresholds T_L and T_H

If $I > T_H \Rightarrow$ edge

$I > T_L \Rightarrow$ not an edge

$T_L \le I \le T_H$ use connectivity to decide

3. Thinning the gradient

Thinning

Directional derivative

$$\frac{\partial f}{\partial n} = \nabla f \cdot n = \frac{\partial f}{\partial x}\cos(\theta) + \frac{\partial f}{\partial y}\sin(\theta)$$

where

$$\tan(\theta) = \frac{\partial f / \partial y}{\partial f / \partial x}$$

Then

$$|\nabla f| = \sqrt{\partial f / \partial x^{2} + \partial f / \partial y^{2}}$$

Calculating a square root is expensive. Instead replace by

1. $$|\nabla f| \sim \max\left(\left|\frac{\partial f}{\partial x}\right|, \left|\frac{\partial f}{\partial y}\right|\right)$$

2. $$|\nabla f| \sim \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$$

3. $$|\nabla f| \sim \max\left(\left|\frac{\partial f}{\partial x}\right|, \left|\frac{\partial f}{\partial y}\right|\right) + \frac{1}{4}\min\left(\left|\frac{\partial f}{\partial x}\right|, \left|\frac{\partial f}{\partial y}\right|\right)$$

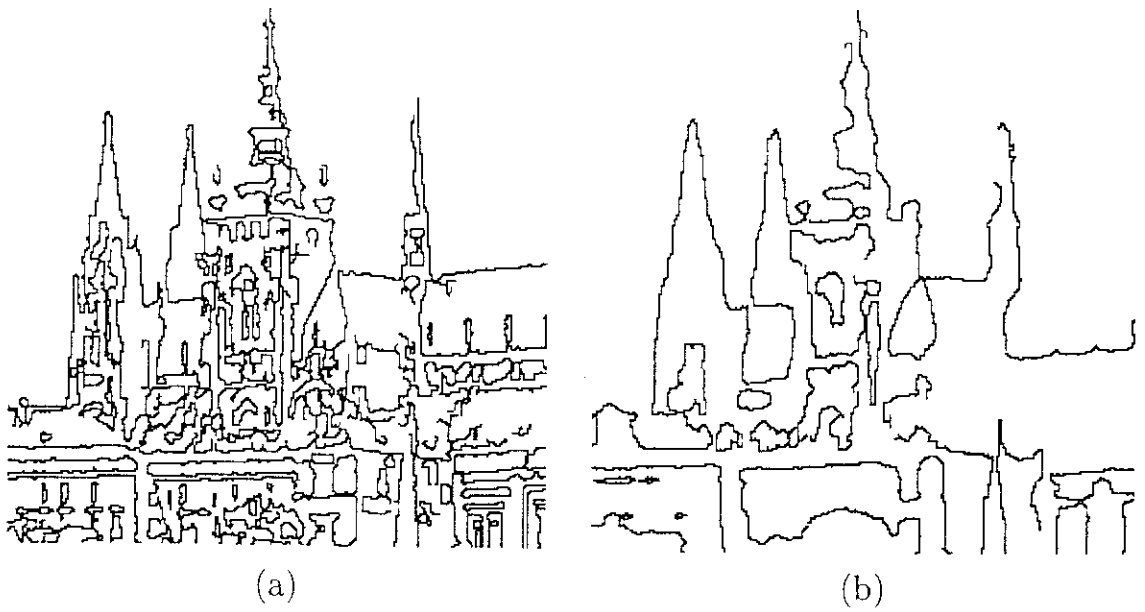
are not isotropic. The error is largest at 45°

Figure 6.15). Then the synthesized edge response is compared with the actual edge response for larger σ. Additional edges are marked only if they have a significantly stronger response than that predicted from synthetic output.

This procedure may be repeated for a sequence of scales. a cumulative edge map being built by adding those edges that were not identified at smaller scales.

Algorithm 4.4: Canny edge detector

1. Convolve an image f with a Gaussian of scale σ.
2. Estimate local edge normal directions $\mathbf{n}$ using equation (4.63) for each pixel in the image.
3. Find the location of the edges using equation (4.65) (non-maximal suppression).
4. Compute the magnitude of the edge using equation (4.66).
5. Threshold edges in the image with hysteresis to eliminate spurious responses.
6. Repeat steps (1) through (5) for ascending values of the standard deviation σ.
7. Aggregate the final information about edges at multiple scale using the 'feature synthesis' approach.



(a) (b)

Figure 4.23: *Canny edge detection at two different scales.*

Figure 4.23a shows the edges of Figure 4.10a detected by a Canny operator with $\sigma = 1.0$. Figure 4.23b shows the edge detector response for $\sigma = 2.8$ (feature synthesis has not been applied here).

to be solved analytically, an efficient approximation turns out to be the first derivative of a Gaussian function. Recall that a Gaussian has the form:

$$G(x) = e^{-\frac{x^2}{2\sigma 2}} \qquad \text{(EQ 1.21)}$$

The derivative with respect to x is therefore:

$$G'(x) = \left(-\frac{x}{\sigma^2}\right)e^{-\left(\frac{x^2}{2\sigma 2}\right)} \qquad \text{(EQ 1.22)}$$

In two dimensions, a Gaussian is given by:

$$G(x,y) = \sigma^2 e^{-\left(\frac{x_2+y_2}{2\sigma 2}\right)} \qquad \text{(EQ 1.23)}$$

and G has derivatives in both the x and y directions. The approximation to Canny's optimal filter for edge detection is G′, and so by convolving the input image with G′ we obtain an image E that has enhanced edges, even in the presence of noise, which has been incorporated into the model of the edge image.

A convolution is fairly simple to implement, but is expensive computationally, especially a two-dimensional convolution. This was seen in the Marr edge detector. However, a convolution with a two-dimensional Gaussian can be separated into two convolutions with one-dimensional Gaussians, and the differentiation can be done afterwards. Indeed, the differentiation can also be done by convolutions in one dimension, giving two images: one is the x component of the convolution with G′ and the other is the y component.

Thus, the Canny edge detection algorithm to this point is:

1. Read in the image to be processed, I.
2. Create a 1D Gaussian mask G to convolve with I. The standard deviation s of this Gaussian is a parameter to the edge detector.
3. Create a 1D mask for the first derivative of the Gaussian in the x and y directions; call these G_x and G_y. The same s value is used as in step 2 above.
4. Convolve the image I with G along the rows to give the x component image I_x, and down the columns to give the y component image I_y.
5. Convolve I_x with G_x to give I_x', the x component of I convolved with the derivative of the Gaussian, and convolve I_y with G_y to give I_y'.
6. If you want to view the result at this point the x and y components must be combined. The magnitude of the result is computed at each pixel (x,y) as:

$$M(x,y) = \sqrt{I'_x(x,y)^2 + I'_y(x,y)^2}$$

The magnitude is computed in the same manner as it was for the gradient, which is in fact what is being computed.

A complete C program for a Canny edge detector is given at the end of this chapter, but some explanation is relevant at this point. The main program opens the image file and reads it, and also reads in the parameters (such as σ). It then calls the function canny, which does most of the actual work. The first thing canny does is to compute the Gaussian filter mask (called gau in the program) and the derivative of a Gaussian filter mask (called dgau). The size of the mask to be used depends on σ; for small σ the Gaussian will quickly become zero, resulting in a small mask. The program determines the needed mask size automatically.

Next, the function computes the convolution as in step 4 above. The C function separable_convolution does this, being given the input image and the mask, and returning the x and y parts of the convolution (called smx and smy in the program; these are floating point 2D arrays). The convolution of step 5 above is then calculated by calling the C function dxy_separable_convolution twice, once for x and once for y. The resulting real images (called dx and dy in the program) are the x and y components of the image convolved with G'. The function norm will calculate the magnitude given any pair of x and y components.

The final step in the edge detector is a little curious at first, and needs some explanation. The value of the pixels in M is large if they are edge pixels and smaller if not, so thresholding could be used to show the edge pixel as white and the background as black. This does not give very good results; what must be done is to threshold the image based partly on the direction of the gradient at each pixel. The basic idea is that edge pixels have a direction associated with them; the magnitude of the gradient at an edge pixel should be greater than the magnitude of the gradient of the pixels on each side of the edge. The final step in the Canny edge detector is a *nonmaximum suppression* step, where pixels that are not local maxima are removed.

Figure 1.13 attempts to shed light on this process by using geometry. Part a of this figure shows a 3×3 region centered on an edge pixel, which in this case is vertical. The arrows indicate the direction of the gradient at each pixel, and the length of the arrows is proportional to the magnitude of the gradient. Here, nonmaximal suppression means that the center pixel, the one under consideration, must have a larger gradient magnitude than its neighbors *in the gradient direction*; these are the two pixels marked with an "x." That is: From the center pixel, travel in the direction of the gradient until another pixel is encountered; this is the first neighbor. Now, again starting at the center pixel, travel in the direction opposite to that of the gradient until another pixel is encountered; this is the second neighbor. Moving from one of these to the other passes though the edge pixel in a direction that crosses the edge, so the gradient magnitude should be largest at the edge pixel.

In this specific case, the situation is clear. The direction of the gradient is horizontal, and the neighboring pixels used in the comparison are exactly the left

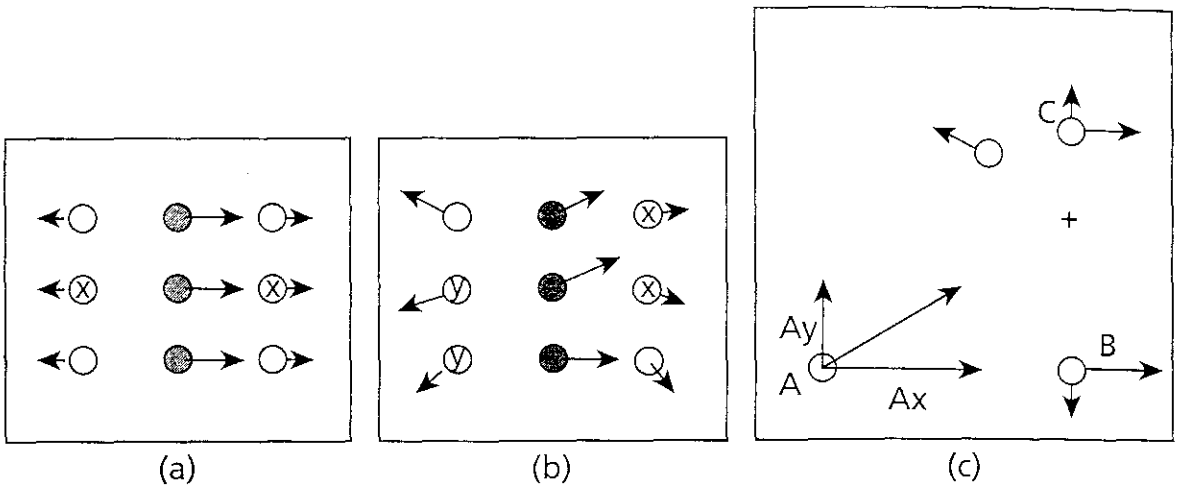


Figure 1.13 Nonmaximum suppression. (a) Simple case, where the gradient direction is horizontal. (b) Most cases have gradient directions that are not horizontal or vertical, so there is no exact gradient at the desired point. (c) Gradients at pixels neighboring A are used to estimate the gradient at the location marked with +.

and right neighbors. Unfortunately this does not happen very often. If the gradient direction is arbitrary, then following that direction will usually take you to a point in between two pixels. What is the gradient there? Its value cannot be known for certain, but it can be estimated from the gradients of the neighboring pixels. It is assumed that the gradient changes continuously as a function of position, and that the gradient at the pixel coordinates are simply sampled from the continuous case. If it is further assumed that the change in the gradient between any two pixels is a linear function, then the gradient at any point between the pixels can be approximated by a linear interpolation.

A more general case is shown in Figure 1.13b. Here the gradients all point in different directions, and following the gradient from the center pixel now takes us in between the pixels marked x. Following the direction opposite to the gradient takes us between the pixels marked y. Let's consider only the case involving the "x" pixels as shown in Figure 1.13c, since the other case is really the same. The pixel named A is the one under consideration, and pixels B and C are the neighbors in the direction of the positive gradient. The vector components of the gradient at A are A_x and A_y, and the same naming convention will be used for B and C.

Each pixel lies on a grid line having an integer x and y coordinate. This means that pixels A and B differ by one distance unit in the x direction. It must be determined which grid line will be crossed first when moving from A in the gradient direction. Then the gradient magnitude will be linearly interpolated using the two pixels on that grid line and on opposite sides of the crossing point, which

is at location (P_x, P_y). In Figure 1.13c the crossing point is marked with a "+," and is in between B and C. The gradient magnitude at this point is estimated as

$$G = (P_y - C_y)Norm(C) + (B_y - P_y)Norm(B) \qquad \text{(EQ 1.24)}$$

where the norm function computes the gradient magnitude.

Every pixel in the filtered image is processed in this way; the gradient magnitude is estimated for two locations, one on each side of the pixel, and the magnitude at the pixel must be greater than its neighbors'. In the general case there are eight major cases to check for, and some shortcuts that can be made for efficiency's sake, but the above method is essentially what is used in most implementations of the Canny edge detector. The function nonmax_suppress in the C source at the end of the chapter computes a value for the magnitude at each pixel based on this method, and sets the value to zero unless the pixel is a local maximum.

It would be possible to stop at this point, and use the method to enhance edges. Figure 1.14 shows the various stages in processing the chessboard test image of

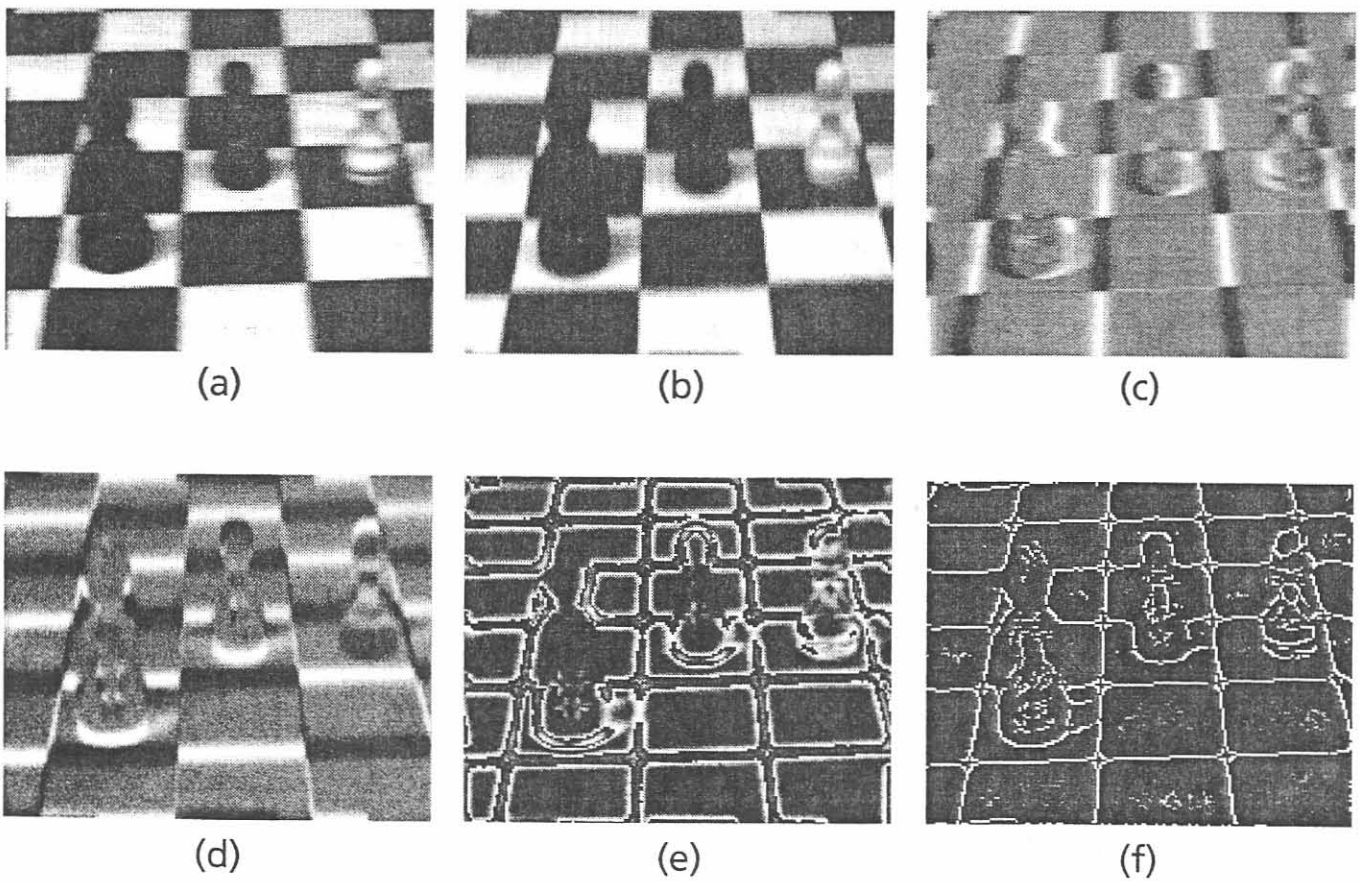


Figure 1.14 Intermediate results from the Canny edge detector. (a) *X* component of the convolution with a Gaussian. (b) *Y* component of the convolution with a Gaussian. (c) *X* component of the image convolved with the derivative of a Gaussian. (d) *Y* component of the image convolved with the derivative of a Gaussian. (e) Resulting magnitude image. (f) After nonmaximum suppression.

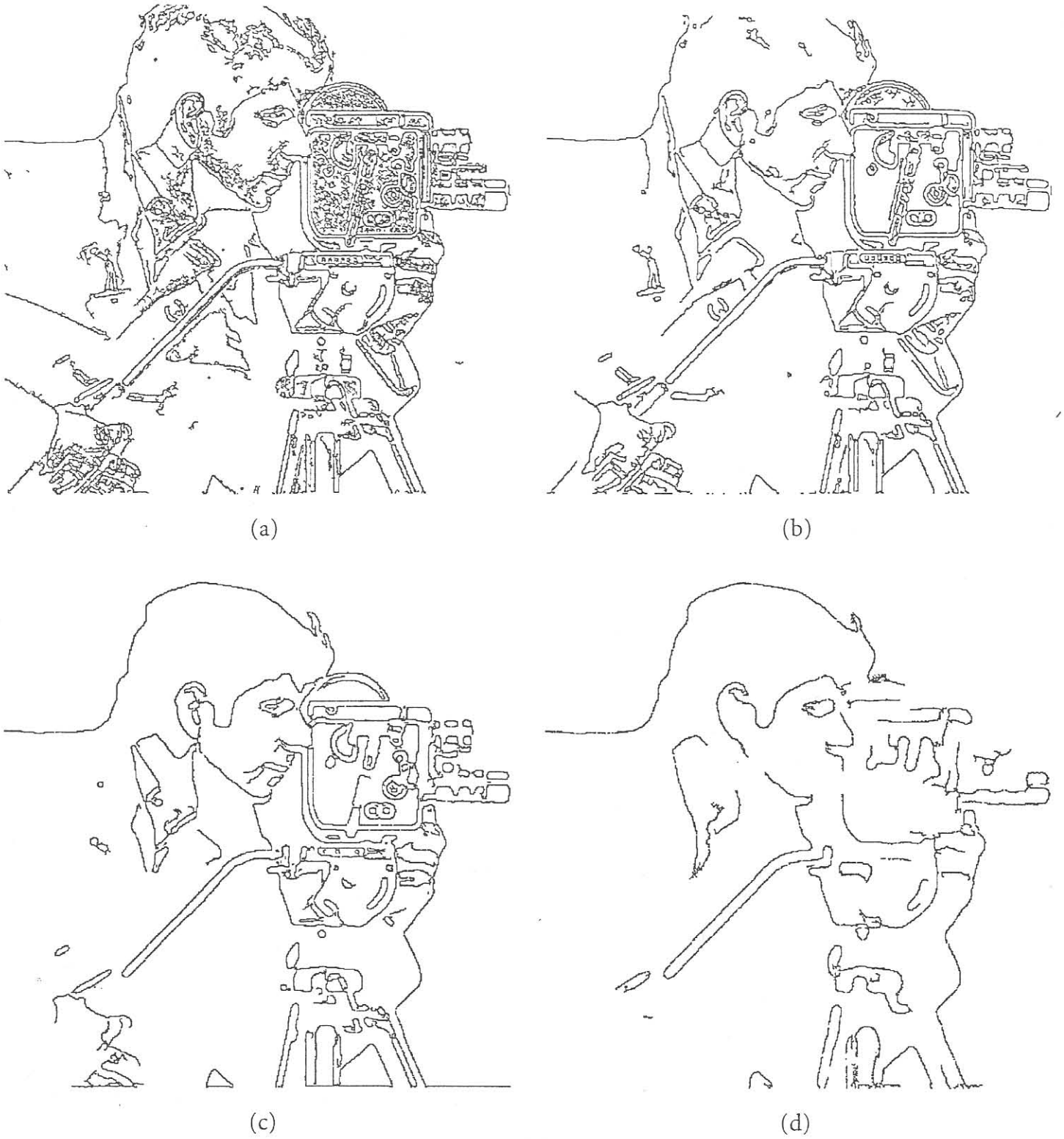


FIGURE 10 Canny edge detector of Eq. (20) applied after Gaussian smoothing over a range of σ: (a) $\sigma = 0.5$, (b) $\sigma = 1$, (c) $\sigma = 2$, (d) $\sigma = 4$. The thresholds are fixed in each case at $T_U = 10$ and $T_L = 4$.

$^7 f_c|$ along the gradient direction more difficult. If
nt image were to be computed by summing the
ient gradient vectors, not just their magnitudes,
ent orientations of the component gradients could
interfere and nullify some edges.

roaches to color edge detection, while generally less
ally efficient, tend to have better theoretical justi-
idean distance in color space between the color
ven pixel and its neighbors can be a good basis for

Trahanias and Venetsanopoulos [21] described th
tor order statistics as the basis for color edge detectio
per by Scharcanski and Venetsanopoulos [18] furthe
cept. Although not strictly founded on the gradient
their techniques are effective and worth mention l
of their vector bases. The basic idea is to look for cha
vector statistics, particularly vector dispersion, to
presence of edges.

Multispectral images can have many components

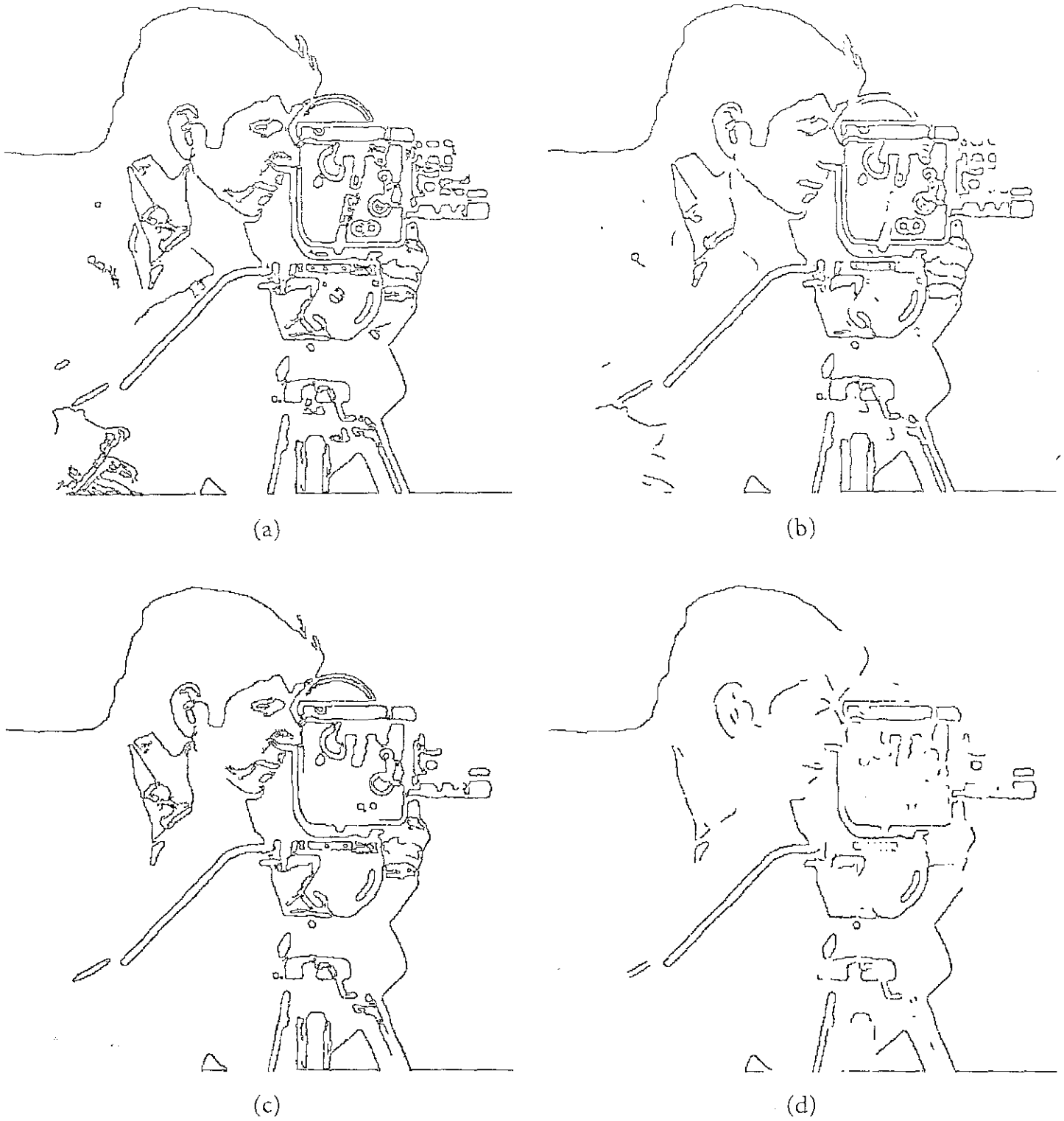


FIGURE 11 Canny edge detector of Eq. (20) applied after Gaussian smoothing with $\sigma = 2$: (a) $T_U = 10$, $T_L = 1$; (b) $T_U = T_L = 10$; (c) $T_U = 20$, $T_L = 1$; (d) $T_U = T_L = 20$. As T_L is changed, notice the effect on the results of hysteresis thresholding.

ging all components, which have first been Gaus-
and then finds the edges in that image. The
ly works well because multispectral images tend
orrelation between components. However, it is
e information to diminish or vanish if the com-
ctively interfere.

xplored operators for computing the vector gra-
d an edge detection approach based on combin-
ent gradients. A multispectral contrast function
he image is searched for pixels having maximal

image's magnitude and direction values at a given pixe
of the component having the greatest gradient ma
that pixel. Some edges can be missed by the maxim
technique because they may be swamped by differentl
stronger edges present in another band.

The method of combining component edge maps
efficient because an edge map must first be computec
band. On the positive side, this method is capable of det
edge that is detectable in at least one component ima
bination of component edge maps into a single resu

was assumed to be a convolution filter f that would smooth the noise and locate the edge. The problem is to identify the one filter that optimizes the three edge detection criteria.

In one dimension, the response of the filter f to an edge G is given by a convolution integral:

$$H = \int_{-W}^{W} G(-x)f(x)dx \qquad \text{(EQ 1.17)}$$

The filter is assumed to be zero outside of the region [−W,W]. Mathematically, the three criteria are expressed as:

$$SNR = \frac{A\left|\int_{-W}^{0} f(x)dx\right|}{n_0\sqrt{\int_{-W}^{W} f^2(x)dx}} \qquad \text{(EQ 1.18)}$$

$$Localization = \frac{A|f(0)|}{n_0\sqrt{\int_{-W}^{W} f^2 dx}} \qquad \text{(EQ 1.19)}$$

$$x_{zc} = \pi\left(\frac{\int_{-\infty}^{\infty} f^2(x)dx}{\int_{-\infty}^{\infty} f'^2(x)dx}\right)^{\frac{1}{2}} \qquad \text{(EQ 1.20)}$$

The value of *SNR* is the output signal to noise ratio (error rate), and should be as large as possible: we need lots of signal and little noise. The *localization* value represents the reciprocal of the distance of the located edge from the true edge, and should also be as large as possible, which means that the distance would be as small as possible. The value x_{zc} is a constraint; it represents the mean distance between zero crossings of f′ and is essentially a statement that the edge detector f will not have too many responses to the same edge in a small region.

Canny attempts to find the filter f that maximizes the product SNR × localization subject to the multiple-response constraint, and while the result is too complex

One possible way to evaluate an edge detector, based on the above discussion, was proposed by Pratt (1978), who suggested the following function:

$$E_1 = \frac{\sum_{i=1}^{I_A} \left(\frac{1}{1 + \alpha d(i)^2} \right)}{\max (I_A, I_I)} \qquad \text{(EQ 1.8)}$$

where I_A is the number of edge pixels found by the edge detector, I_I is the actual number of edge pixels in the test image, and the function d(i) is the distance between the actual *ith* pixel and the one found by the edge detector. The value α is used for scaling, and should be kept constant for any set of trials. A value of 1/9 will be used here, as it was in Pratt's work. This metric is, as discussed previously, a function of the distance between correct and measured edge positions, but is only indirectly related to the false positives and negatives.

Kitchen and Rosenfeld (1981) also present an evaluation scheme, this one based on *local edge coherence*. It does not concern itself with the actual position of an edge, and so is a supplement to Pratt's metric. Instead, it measures how well the edge pixel fits into the local neighborhood of edge pixels. The first step is the definition of a function that measures how well an edge pixel is continued on the left; this function is:

$$L(k) = \begin{cases} a(d,d_k)a\left(\frac{k\pi}{4}, d + \frac{\pi}{2}\right) & \text{if neighbor k is an edge pixel} \\ 0 & \text{Otherwise} \end{cases} \qquad \text{(EQ 1.9)}$$

where d is the edge direction at the pixel being tested, d_0 is the edge direction at its neighbor to the right, d_1 is the direction of the upper-right neighbor, and so on counterclockwise about the pixel involved. The function a is a measure of the angular difference between any two angles:

$$a(\alpha, \beta) = \frac{\pi - |\alpha - \beta|}{\pi} \qquad \text{(EQ 1.10)}$$

A similar function measures directional continuity on the right of the pixel being evaluated:

$$R(k) = \begin{cases} a(d,d_k)a\left(\frac{k\pi}{4}, d - \frac{\pi}{2}\right) & \text{if neighbor k is an edge pixel} \\ 0 & \text{Otherwise} \end{cases} \qquad \text{(EQ 1.11)}$$

The overall continuity measure is taken to be the average of the best (largest) value of L(k) and the best value of R(k); this measure is called C.

Then a measure of thinness is applied. An edge should be a thin line, one pixel wide. Lines of a greater width imply that false positives exist, probably because the edge detector has responded more than once to the same edge. The thinness measure T is the fraction of the six pixels in the 3×3 region centered at the pixel being measured, not counting the center and the two pixels found by L(k) and R(k), that are edge pixels. The overall evaluation of the edge detector is:

$$E_2 = \gamma C + (1 - \gamma)T \qquad \text{(EQ 1.12)}$$

where γ is a constant: we will use the value 0.8 here.

We are now prepared to evaluate the two gradient operators. Each of the op-

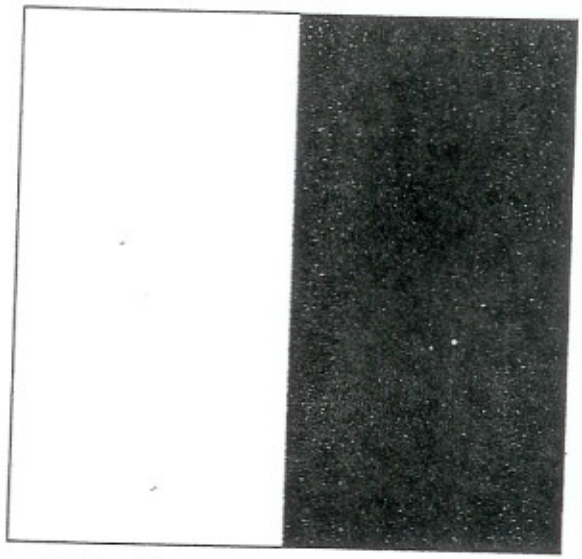
Then both the Pratt and the KR

$$R = \frac{1}{I_N} \sum_{i=1}^{I_F} \frac{1}{1 + \alpha d^2}$$

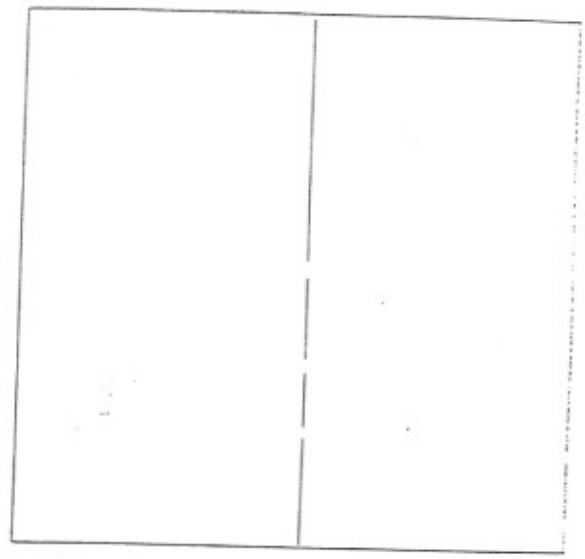
I_N = the maximum of I_I and I_A

I_I = the number of ideal edge points in the image

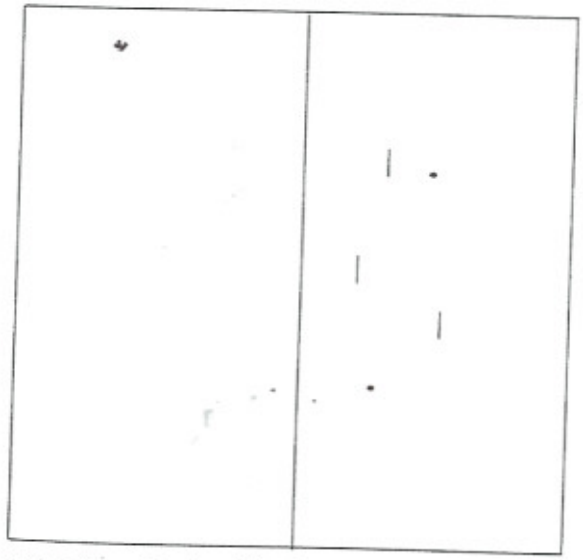
4 Errors in Edge Detection



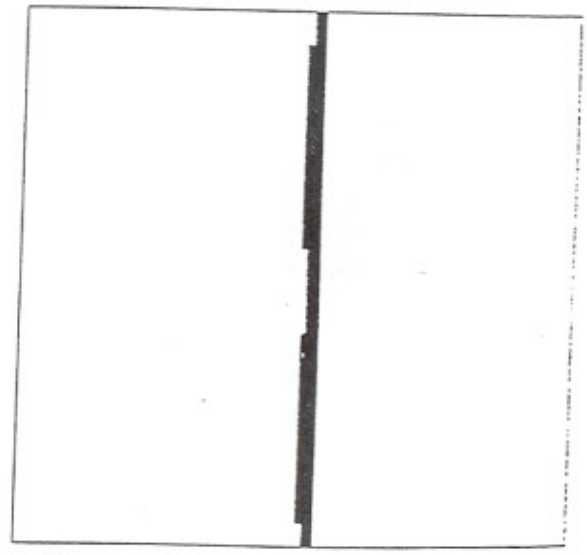
a. Original image.



b. Missed edge points.



c. Noise misclassified as edge points.



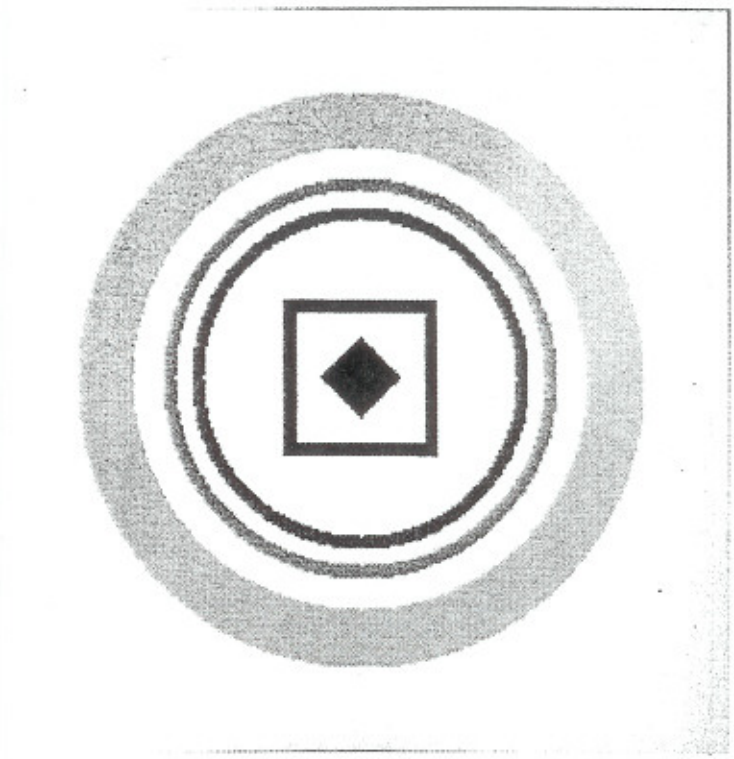
d. Smeared edge.

I_F = the number of edge points found by the edge detector

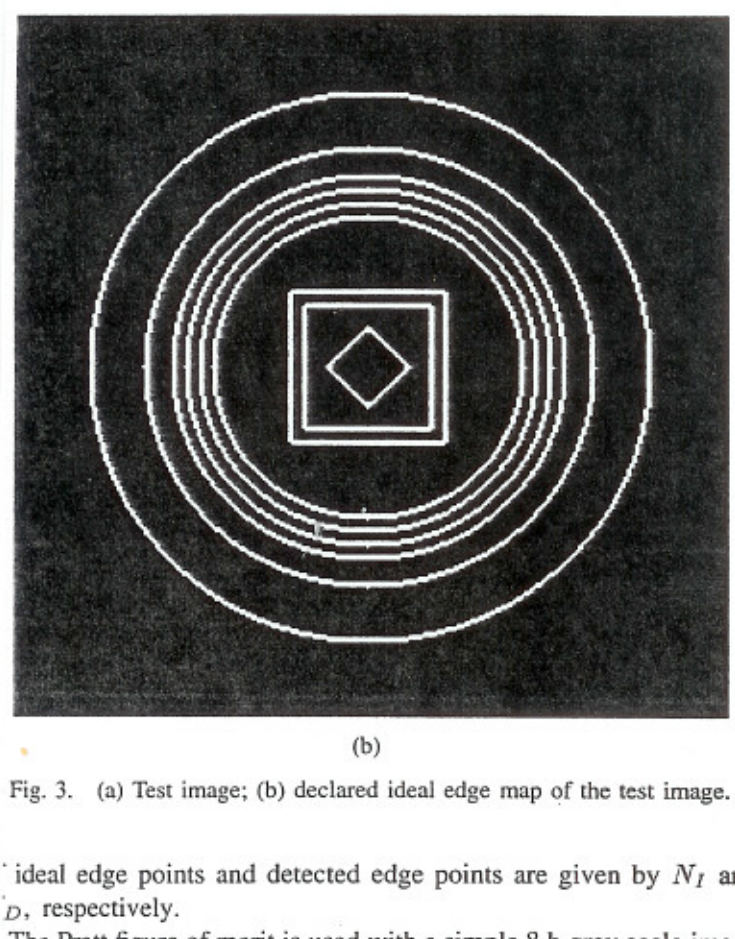
α = a scaling constant that can be adjusted to adjust the penalty for offset edge

d = the distance of a found edge point to an ideal edge point

For this metric, R will be 1 for a perfect edge. Normalizing to the maximum of th ideal and found edge points guarantees a penalty for smeared edges or missing edg points. In general, this metric assigns a better rating to smeared edges than to offse



(a)



(b)

Fig. 3. (a) Test image; (b) declared ideal edge map of the test image.

ideal edge points and detected edge points are given by N_I and $_D$, respectively.

The Pratt figure of merit is used with a simple 8 b grey scale image evaluate the performance of the prefilters. The test image is shown Fig. 3(a) and the declared ideal edge points are shown in Fig. 3(b). The test image has been blurred with a Gaussian point spread function and has been processed using each of the nonlinear sharpeners. he Pratt figure of merit for the Sobel edge detector with each of ıe prefilters is shown in Fig. 4 for all threshold values. A 7×7 indow is used for each of the filters and $k = l = 1$ for the LUM lter (maximum sharpening) and $j = 1$ for the CS filter (maximum arpening). Note that the figure of merit is significantly higher and is

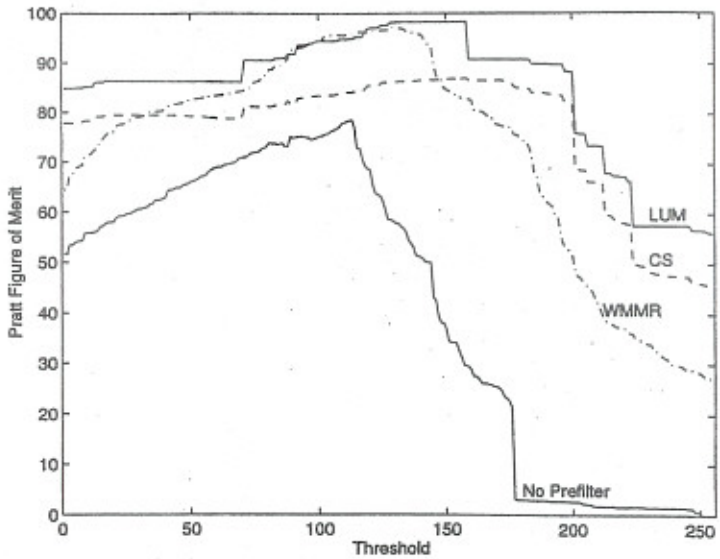


Fig. 4. Pratt figure of merit for the Sobel edge detector with prefilter operating on the blurred test image.

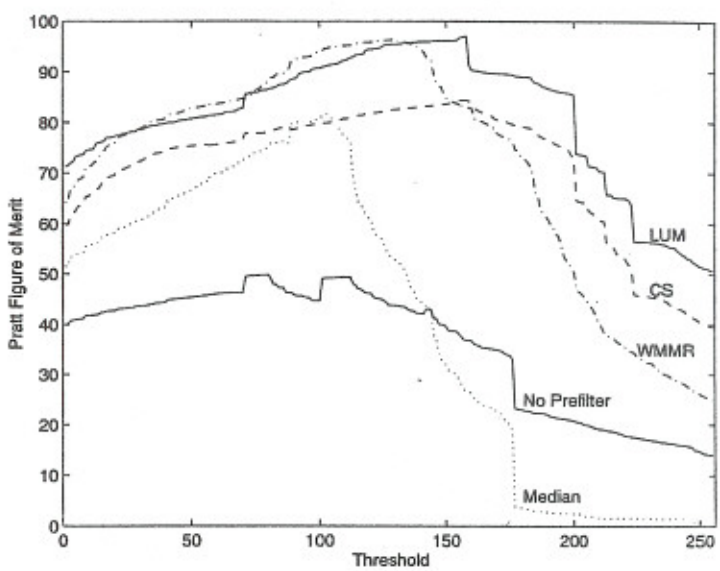


Fig. 5. Pratt figure of merit for the Sobel edge detector with prefilter operating on the blurred test image corrupted with impulsive noise ($p = 0.02$)

The Pratt figure of merit for the case where the test image is blurred and corrupted with $p = 0.02$ impulsive noise is shown in Fig. 5. Fo the LUM filter, $k = l = 5$ and for the CS filter, $j = 5$. Here the median is useful for low threshold values but actually degrade performance at higher threshold levels. The nonlinear sharpeners offe improved performance for all threshold values and the LUM filte generally gives the best results.

B. Natural Image Edge Detection

In this section, edge detection with a blurred natural image co rupted with impulsive noise is considered. Fig. 6(a) shows the imag "Boat" blurred with a 5×5 mean filter and corrupted with impulsiv noise having probability $p = 0.02$. The edge map of this image wit no prefilter is shown in 6(b). In this image, the impulses dominate th edge map, which clearly illustrates the need for some type of prefilte Fig. 6(c) shows the edge map generated after 3×3 median prefilterin The impulses have been removed, but the edges remain blurre Finally, the edge map generated after LUM prefiltering is show in Fig. 6(d). A 5×5 window is used with parameters $k = l = 5$. Th impulses have been suppressed and the edges have been enhance resulting in a detailed edge map. All of the thresholds have been s

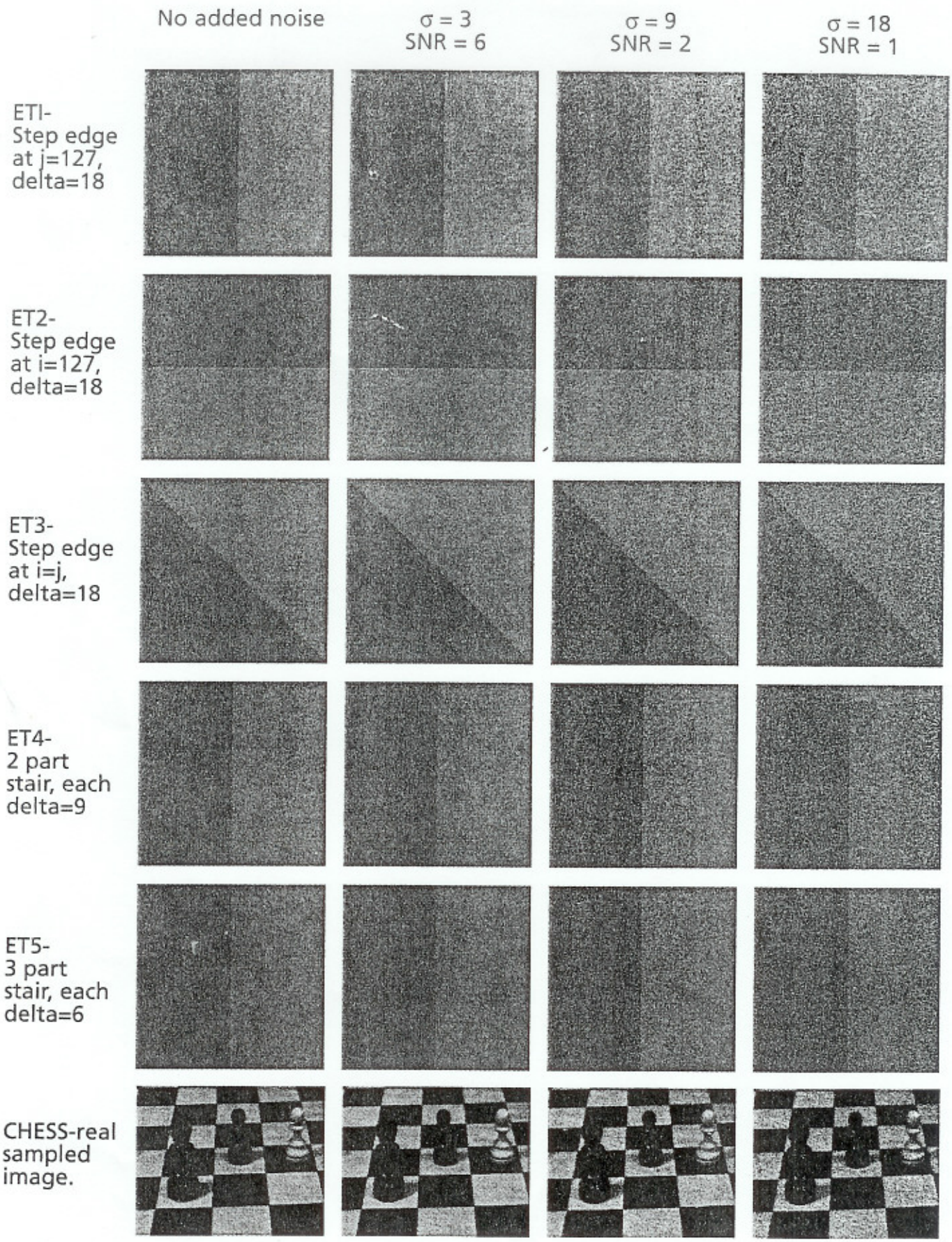


Figure 1.8 Standard test images for edge detector evaluation. There are three step edges and two stairs, plus a real sampled image; all have been subjected to normally distributed zero-mean noise with known standard deviations of 3, 9, and 18.

Table 1.3 Evaluation of the Sobel Edge Detector

Image	Evaluator	No Noise	SNR = 6	SNR = 2	SNR = 1
ET1	Eval 1	0.9727	0.9690	0.1173	0.0617
	Eval 2	0.8992	0.8934	0.4474	0.4263
ET2	Eval 1	0.9726	0.9706	0.1609	0.0526
	Eval 2	0.8992	0.8978	0.4215	0.4255
ET3	Eval 1	0.9726	0.9697	0.1632	0.0733
	Eval 2	0.9325	0.9186	0.4349	0.4240
ET4	Eval 1	0.4860	0.4786	0.0595	0.0373
	Eval 2	0.7328	0.6972	0.4426	0.4266
ET5	Eval 1	0.4627	0.3553	0.0480	0.0355
	Eval 2	0.7496	0.6293	0.4406	0.4250

edge pixel is quantized into eight possibilities here, and is $\pi/4*i$ where i is the number of the mask having the largest response.

Both of these edges detectors were evaluated using the test images of Figure 1.8. The results are outlined in Table 1.3.

The results for the Kirsch operator are found in Table 1.4.

Figure 1.10 shows the response of these templates applied to a selection of the test images. Based on the evaluations and the appearance of the test images,

Table 1.4 Evaluation of the Kirsch Edge Detector

Image	Evaluator	No Noise	SNR = 6	SNR = 2	SNR = 1
ET1	Eval 1	0.9727	0.9727	0.1197	0.0490
	Eval 2	0.8992	0.8992	0.4646	0.4922
ET2	Eval 1	0.9726	0.9726	0.1517	0.0471
	Eval 2	0.8992	0.8992	0.4528	0.4911
ET3	Eval 1	0.9726	0.9715	0.1458	0.0684
	Eval 2	0.9325	0.9200	0.4708	0.4907
ET4	Eval 1	0.4860	0.4732	0.0511	0.0344
	Eval 2	0.7328	0.7145	0.4819	0.4907
ET5	Eval 1	0.4627	0.3559	0.0412	0.0339
	Eval 2	0.7496	0.6315	0.5020	0.4894

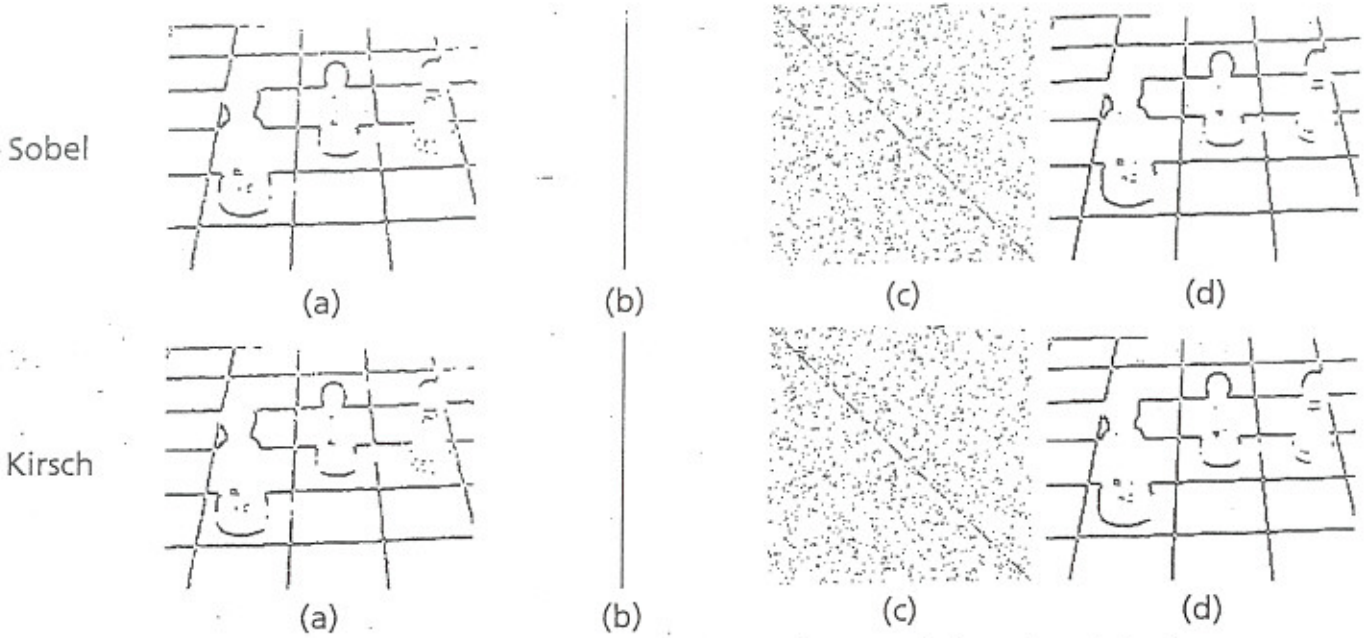


Figure 1.10 Sample results from the template edge detectors. (a) Chess image, noise $\sigma = 3$. (b) ET1, SNR=6. (c) ET3, SNR=2. (d) Chess image, noise $\sigma = 18$.

the Kirsch operator appears to be the best of the two template operators, although the two are very close. Both template operators are superior to the simple derivative operators, especially as the noise increases.

It should be pointed out that in all cases studied so far there are unspecified aspects to the edge detection methods that will have an impact on their efficacy. Principal among these is the thresholding method used, but sometimes simple noise removal is done beforehand and edge thinning is done afterward. The model-based methods that follow generally include these features, sometimes as part of the edge model.

crossings, and merges the resulting edge pixels into a single image. The program is called marr, and can be invoked as

```
marr input.pgm 2.0
```

which would read in the image file named input.pgm and apply the Marr-Hildreth edge-detection algorithm using 1.2 and 2.8 as standard deviations.

Figure 1.11 illustrates the steps in this process, using the chess image (no noise) as an example. Figures 1.11a and b show the original image after being convolved with the Laplacian of the Gaussians, having σ values of 1.2 and 2.8 respectively. Figures 1.11c and 1.11d are the responses from these two different values of σ, and Figure 1.11e shows the result of merging the edge pixels in these two images.

Figure 1.12 shows the result of the Marr-Hildreth edge detector applied to the all of the test images of Figure 1.8. In addition, the evaluation of this operator is shown in Table 1.5.

The evaluations above tend to be low. Because of the width of the Gaussian filter, the pixels that are a distance less than about 4σ from the boundary of the

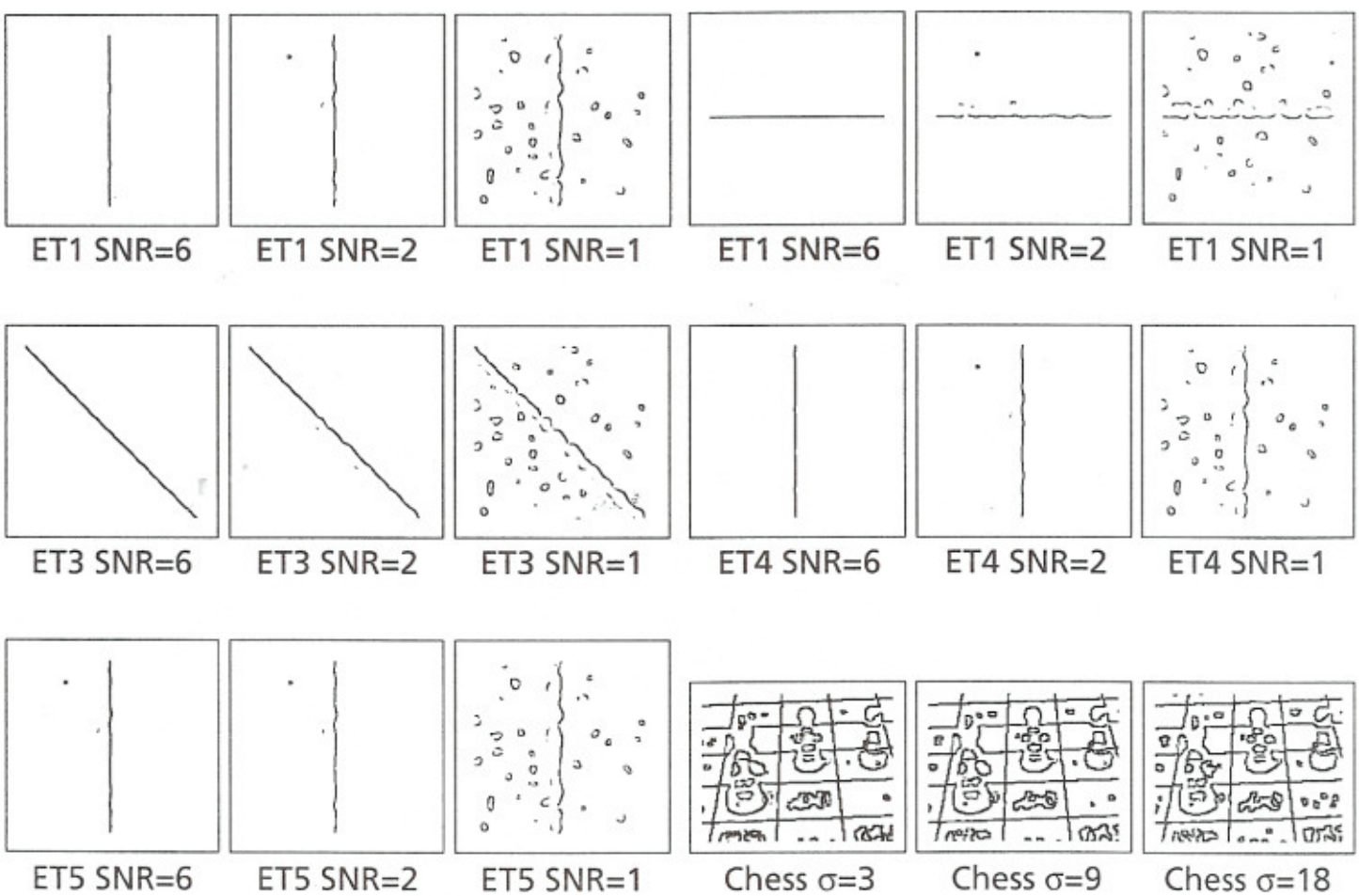


Figure 1.12 Edges from the test images as found by the Marr-Hildreth algorithm, using two resolution values.

Table 1.5 Evaluation of the Marr-Hildreth Edge Detector

Image	Evaluator	No Noise	SNR = 6	SNR = 2	SNR = 1
ET1	Eval 1	0.8968	0.7140	0.7154	0.2195
	Eval 2	0.9966	0.7832	0.6988	0.7140
ET2	Eval 1	0.6948	0.6948	0.6404	0.1956
	Eval 2	0.9966	0.7801	0.7013	0.7121
ET3	Eval 1	0.7362	0.7319	0.7315	0.2671
	Eval 2	0.9133	0.7766	0.7052	0.7128
ET4	Eval 1	0.4194	0.4117	0.3818	0.1301
	Eval 2	0.8961	0.7703	0.6981	0.7141
ET5	Eval 1	0.3694	0.3822	0.3890	0.1290
	Eval 2	0.9966	0.7626	0.6995	0.7141

image are not processed; hence E1 thinks of these as missing edge pixels. When this is taken into account, the evaluation using ET1 with no noise, as an example, becomes 0.9727. Some of the other low evaluations are, on the other hand, the fault of the method. Locality is not especially good, and the edges are not always thin. Still, this edge detector is much better than the previous ones in cases of low signal-to-noise ratio.

1.4 THE CANNY EDGE DETECTOR

In 1986, John Canny defined a set of goals for an edge detector and described an optimal method for achieving them.

Canny specified three issues that an edge detector must address. In plain English, these are:

1. **Error rate**—The edge detector should respond only to edges, and should find all of them; no edges should be missed.
2. **Localization**—The distance between the edge pixels as found by the edge detector and the actual edge should be as small as possible.
3. **Response**—The edge detector should not identify multiple edge pixels where only a single edge exists.

These seem reasonable enough, especially because the first two have already been discussed and used to evaluate edge detectors. The response criterion seems very similar to that of a false positive, at first glance.

Canny assumed a step edge subject to white Gaussian noise. The edge detector

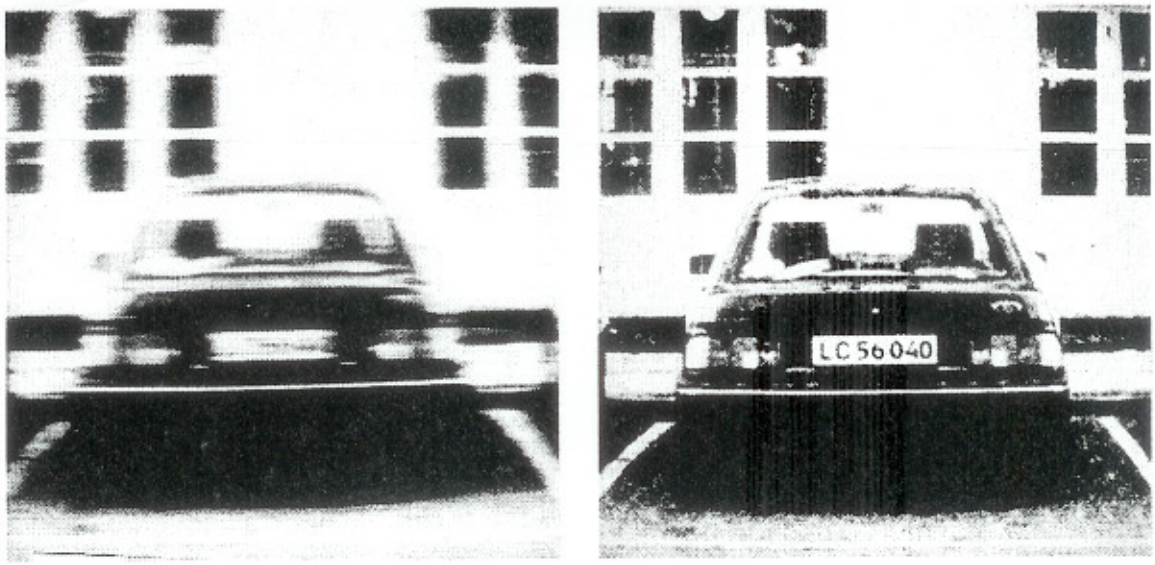
Table 1.6 Evaluation of Canny VS ISEF: E1

Image	Algorithm	No Noise	SNR = 6	SNR = 2	SNR = 1
ET1	Canny	0.9651	0.9498	0.5968	0.1708
	ISEF	0.9689	0.9285	0.7929	0.7036
ET2	Canny	0.9650	0.9155	0.6991	0.2530
	ISEF	0.9650	0.9338	0.8269	0.7170
ET3	Canny	0.9726	0.9641	0.8856	0.4730
	ISEF	0.8776	0.9015	0.7347	0.5238
ET4	Canny	0.5157	0.5092	0.3201	0.1103
	ISEF	0.4686	0.4787	0.4599	0.4227
ET5	Canny	0.5024	0.4738	0.3008	0.0955
	ISEF	0.4957	0.4831	0.4671	0.4074

Sobel, ∇_2, and ∇_1, in that order. The comparison between Canny and ISEF does depend on the parameters selected in each case, and it is likely that evaluations can be found that use a better choice of parameters. In some of these the Canny edge detector will come out ahead, and in some the ISEF method will win. The best set of parameters for a particular image is not known, and so ultimately the user is left to judge the methods.

Table 1.7 Evaluation of Canny VS ISEF: E2

Image	Algorithm	No Noise	SNR = 6	SNR = 2	SNR = 1
ET1	Canny	1.0000	0.5152	0.5402	0.5687
	ISEF	1.0000	0.9182	0.5756	0.5147
ET2	Canny	1.0000	0.6039	0.5518	0.5726
	ISEF	1.0000	0.9462	0.6018	0.5209
ET3	Canny	0.9291	0.7541	0.6032	0.5899
	ISEF	0.9965	0.9424	0.5204	0.4829
ET4	Canny	1.0000	0.7967	0.5396	0.5681
	ISEF	1.0000	0.5382	0.5193	0.5096
ET5	Canny	1.0000	0.5319	0.5269	0.5706
	ISEF	0.9900	0.6162	0.5243	0.5123

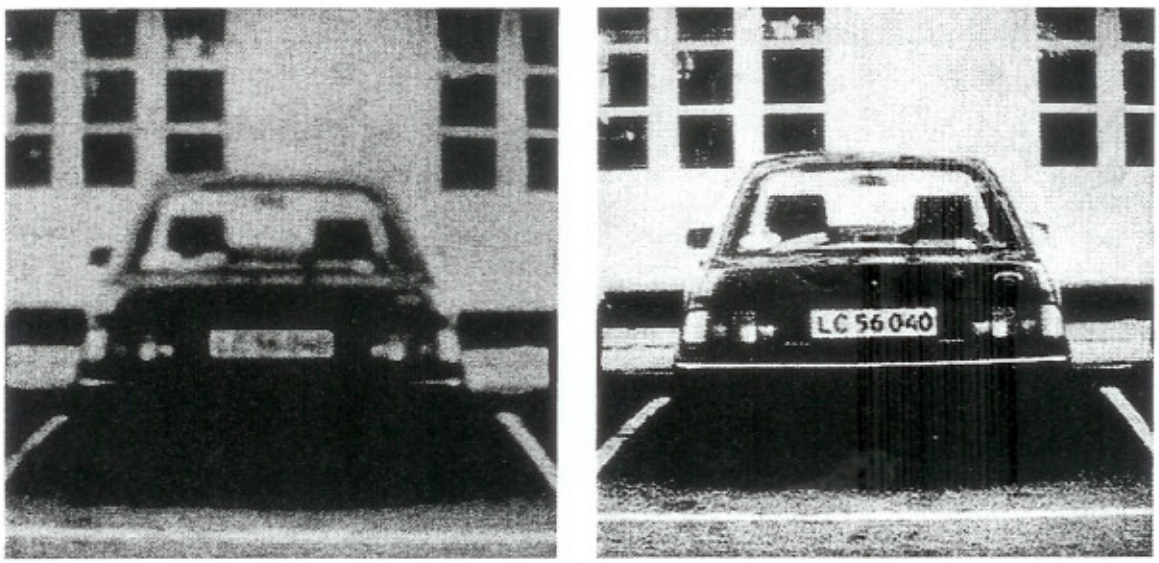


(a) (b)

Figure 4.29: *Restoration of motion blur using Wiener filtration. Courtesy P. Kohout, Crir inalistic Institute, Prague.*

4.5 Summary

- **Image pre-processing**
  - Operations with images at the lowest level of abstraction—both input and outp are intensity images—are called *pre-processing.*
  - The aim of pre-processing is an improvement of the image data that suppress unwilling distortions or enhances some image features important for further pr cessing.
  - Four basic types of pre-processing methods exist:
    * Brightness transformations
    * Geometric transformations
    * Local neighborhood pre-processing
    * Image restoration
- **Pixel brightness transformations**
  - There are two classes of pixel brightness transformations:
    * *Brightness corrections*
    * *Gray-scale transformations*
  - Brightness corrections modify pixel brightness taking into account its origin brightness and its position in the image.
  - Gray-scale transformations change brightness without regard to position in t image.
  - Frequently used brightness transformations include:



(a) (b)

Figure 4.30: *Restoration of wrong focus blur using Wiener filtration. Courtesy P. Kohout, Criminalistic Institute, Prague.*

    * Brightness thresholding
    * Histogram equalization
    * Logarithmic gray-scale transforms
    * Look-up table transforms
    * Pseudo-color transforms
  - The goal of histogram equalization is to create an image with equally distributed brightness levels over the whole brightness scale.

- **Geometric transformations**
  - Geometric transforms permit the elimination of the geometric distortions that occur when an image is captured.
  - A geometric transform typically consists of two basic steps:
    * *Pixel co-ordinate transformation*
    * *Brightness interpolation*
  - Pixel co-ordinate transformations map the co-ordinates of the input image pixel to a point in the output image; *affine* and *bilinear* transforms are frequently used.
  - The output point co-ordinates do not usually match the digital grid after the transform and interpolation is employed to determine brightnesses of output pixels; *nearest-neighbor*, *linear*, and *bi-cubic* interpolations are frequently used.

- **Local pre-processing**
  - Local pre-processing methods use a small neighborhood of a pixel in an input image to produce a new brightness value in the output image.
  - For the pre-processing goal, two groups are common: *smoothing* and *edge detection.*

- Smoothing aims to suppress noise or other small fluctuations in the image; it is equivalent to suppressing high frequencies in the Fourier transform domain.
- Smoothing approaches based on direct averaging blur image edges. More sophisticated approaches reduce blurring by averaging in homogeneous local neighborhoods.
- *Median* smoothing is a non-linear operation; it reduces the blurring of edges by replacing the current point in the image by the median of the brightnesses in its neighborhood.
- *Gradient operators* determine edges—locations in which the image function undergoes rapid changes; they have a similar effect to suppressing low frequencies in the Fourier transform domain.
- *Edge* is a property attached to an individual pixel and has two components, *magnitude* and *direction*.
- Most gradient operators can be expressed using *convolution masks*; examples include Roberts, Laplace, Prewitt, Sobel, Robinson, and Kirsch operators.
- The main disadvantage of convolution edge detectors is their scale dependence and noise sensitivity. There is seldom a sound reason for choosing a particular size of a local neighborhood operator.
- *Zero-crossings* of the second derivative are more robust than small-size gradient detectors and can be calculated as a Laplacian of Gaussians (LoG) or as a difference of Gaussians (DoG).
- The *Canny* edge detector is optimal for step edges corrupted by white noise. The optimality criterion is based on requirements of *detecting* important edges, small *localization* error, and *single-edge response*. Canny edge detection starts with convolving an image with a symmetric 2D Gaussian and then differentiating in the direction of the gradient; further steps include *non-maximal edge suppression*, *hysteresis thresholding*, and *feature synthesis*.
- Edges can also be detected in multi-spectral images.
- Other local pre-processing operations include *line finding*, *line thinning*, *line filling*, and *interest point detection*.
- In *adaptive neighborhood pre-processing*, the neighborhood sizes and shapes are dependent on characteristics of image data and on parameters defining measures of homogeneity of a neighborhood.

- **Image restoration**
  - Image restoration methods aim to suppress degradation using knowledge about its nature. Most image restoration methods are based on *deconvolution* applied globally to the entire image.
  - Relative-constant-speed movement of the object with respect to the camera, wrong lens focus, and atmospheric turbulence are three typical image degradations with simple degradation functions.
  - *Inverse filtration* assumes that degradation was caused by a linear function.

ıage Analysis

Image analysis techniques return information about the structure of an image. This section describes toolbox functions that you can use for these image analysis techniques:

- Edge detection
- Quadtree decomposition

The functions described in this section work only with grayscale intensity images.

Edge Detection

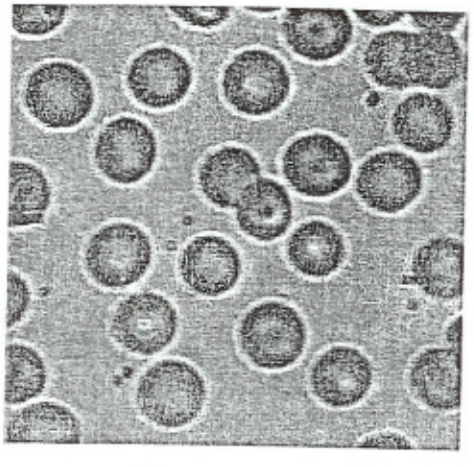
You can use the edge function to detect edges, which are those places in an image that correspond to object boundaries. To find edges, this function looks for places in the image where the intensity changes rapidly, using one of these two criteria:

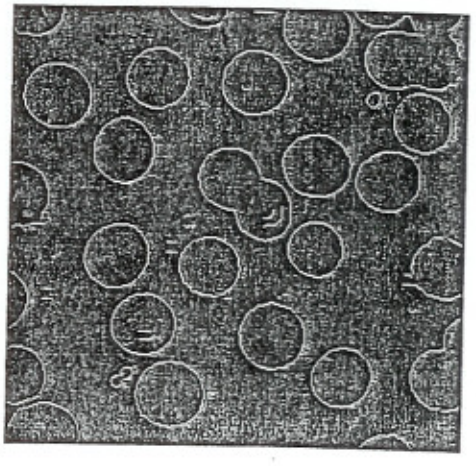
- Places where the first derivative of the intensity is larger in magnitude than some threshold
- Places where the second derivative of the intensity has a zero crossing

edge provides a number of derivative estimators, each of which implements one of the definitions above. For some of these estimators, you can specify whether the operation should be sensitive to horizontal or vertical edges, or both. edge returns a binary image containing 1's where edges are found and 0's elsewhere.

The example below uses the Sobel method to detect the edges in an image of blood cells. By default, the operation is sensitive to both horizontal and vertical edges.

```
I = imread('blood1.tif');
BW = edge(I,'sobel');
imshow(I)
figure, imshow(BW)
```

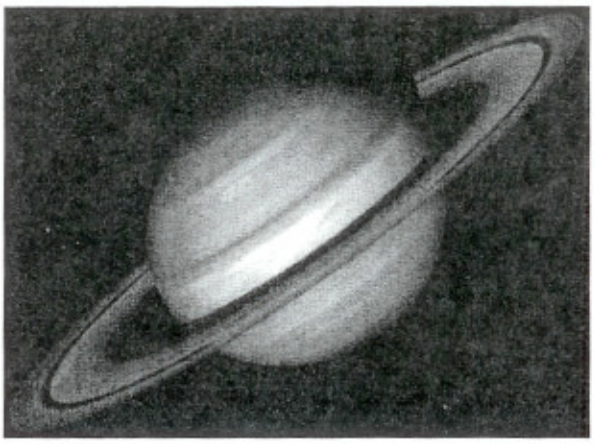


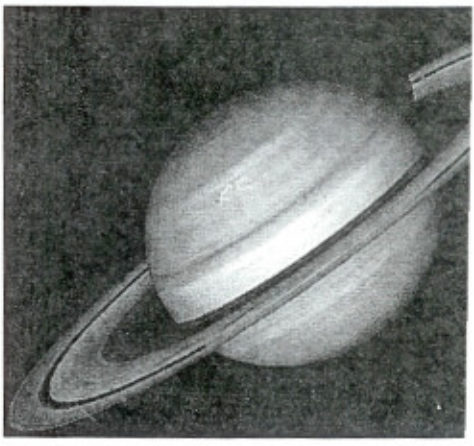


For an interactive demonstration of edge detection, try running edgedemo.

Example

```
I = imread('saturn.tif');
h = fspecial('unsharp',0.5);
I2 = filter2(h,I)/255;
imshow(I)
figure, imshow(I2)
```





Algorithms

`fspecial` creates Gaussian filters using:

$$h_g(n_1, n_2) = e^{-(n_1^2 + n_2^2)/(2\pi\sigma^2)}$$

$$h(n_1, n_2) = \frac{h_g(n_1, n_2)}{\sum_{n_1}\sum_{n_2} h_g}$$

`fspecial` creates Laplacian filters using:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

$$\nabla^2 \approx \frac{4}{(\alpha+1)} \begin{bmatrix} \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4} \\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \end{bmatrix}$$

fspecial

Purpose

Create predefined filters

Syntax

```
h = fspecial(type)
h = fspecial(type,parameters)
```

Description

`h = fspecial(type)` creates a two-dimensional filter `h` of the specified typ (`fspecial` returns `h` as a computational molecule, which is the appropriate form to use with `filter2`.) `type` is a string having one of these values:

- `'gaussian'` for a Gaussian lowpass filter
- `'sobel'` for a Sobel horizontal edge-emphasizing filter
- `'prewitt'` for a Prewitt horizontal edge-emphasizing filter
- `'laplacian'` for a filter approximating the two-dimensional Laplacian op ator
- `'log'` for a Laplacian of Gaussian filter
- `'average'` for an averaging filter
- `'unsharp'` for an unsharp contrast enhancement filter

Depending on `type`, `fspecial` may take additional parameters which you c supply. These parameters all have default values.

`h = fspecial('gaussian',n,sigma)` returns a rotationally symmetric Gau sian lowpass filter with standard deviation `sigma` (in pixels). `n` is a 1-by-2 vect specifying the number of rows and columns in `h`. (`n` can also be a scalar, in which case `h` is `n`-by-`n`.) If you do not specify the parameters, `fspecial` uses t default values of [3 3] for `n` and 0.5 for `sigma`.

`h = fspecial('sobel')` returns this 3-by-3 horizontal edge-finding and y-derivative approximation filter:

```
[ 1   2   1
  0   0   0
 -1  -2  -1 ]
```

To find vertical edges, or for x-derivatives, use `-h'`.

`fspecial` creates Laplacian of Gaussian (LoG) filters using:

$$h_g(n_1, n_2) = e^{-(n_1^2 + n_2^2)/(2\pi\sigma^2)}$$

$$h(n_1, n_2) = \frac{(n_1{}^2 + n_2{}^2 - 2\pi\sigma^2)h_g(n_1, n_2)}{\pi^2\sigma^4 \sum_{n_1}\sum_{n_2} h_g}$$

`fspecial` creates averaging filters using:

```
ones(n(1),n(2))/(n(1)*n(2))
```

`fspecial` creates unsharp filters using:

$$\frac{1}{(\alpha+1)}\begin{bmatrix} -\alpha & \alpha-1 & -\alpha \\ \alpha-1 & \alpha+5 & \alpha-1 \\ -\alpha & \alpha-1 & -\alpha \end{bmatrix}$$

`conv2`, `edge`, `filter2`, `fsamp2`, `fwind1`, `fwind2`

`del2` in the online MATLAB Function Reference