

Module 3

Image Restoration and Reconstruction: A model of Image degradation/restoration process, restoration in the presence of noise only, periodic noise reduction by frequency domain filtering.

Image Segmentation: Fundamentals, Point, Line and edge detection, thresholding (Foundation & Basic global thresholding only), Segmentation by region growing & region splitting & merging.

3.1 A model of Image degradation/restoration process :

The degradation of an image can occur due to various factors, including blurring, noise, and distortions introduced by the imaging system or environmental conditions.

A general **mathematical model** for degradation is:

$$g(x, y) = H[f(x, y)] + \eta(x, y)$$

where:

- $f(x, y)$ = Original (undegraded) image
- $g(x, y)$ = Observed (degraded) image
- H = Degradation function (system response)
- $\eta(x, y)$ = Noise introduced in the system

Here, H represents distortions such as **motion blur, defocus, or atmospheric turbulence**, while $\eta(x, y)$ represents additional **random noise**.

The goal of **image restoration** is to estimate the original image $f(x, y)$ from the degraded image $g(x, y)$ by reversing or minimizing the effects of H and $\eta(x, y)$

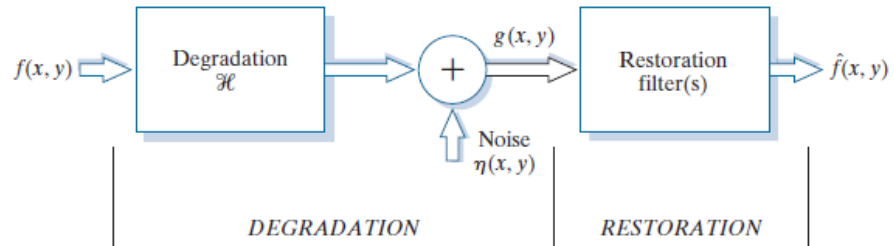
Mathematical Representation:

Restoration involves **finding an estimate** $\hat{f}(x, y)$ of the original image $f(x, y)$, using an inverse function H^{-1} :

$$\hat{f}(x, y) = H^{-1}[g(x, y)]$$

However, due to noise $\eta(x, y)$, directly computing H^{-1} is not always practical, and advanced filtering techniques (e.g., **Wiener filter**) are used.

FIGURE 5.1
A model of the
image
degradation/
restoration
process.



In this chapter, we model image degradation as an operator $\mathcal{H}$ that, together with an additive noise term, operates on an input image $f(x, y)$ to produce a degraded image $g(x, y)$ (see Fig. 5.1). Given $g(x, y)$, some knowledge about $\mathcal{H}$, and some knowledge about the additive noise term $\eta(x, y)$, the objective of restoration is to obtain an estimate $\hat{f}(x, y)$ of the original image. We want the estimate to be as close as possible to the original image and, in general, the more we know about $\mathcal{H}$ and η , the closer $\hat{f}(x, y)$ will be to $f(x, y)$.

We will show in Section 5.5 that, if $\mathcal{H}$ is a linear, position-invariant operator, then the degraded image is given in the spatial domain by

$$g(x, y) = (h \star f)(x, y) + \eta(x, y) \quad (5-1)$$

where $h(x, y)$ is the spatial representation of the degradation function. As in Chapters 3 and 4, the symbol “ $\star$ ” indicates convolution. It follows from the convolution theorem that the equivalent of Eq. (5-1) in the frequency domain is

$$G(u, v) = H(u, v)F(u, v) + N(u, v) \quad (5-2)$$

where the terms in capital letters are the Fourier transforms of the corresponding terms in Eq. (5-1). These two equations are the foundation for most of the restoration material in this chapter.

In the following three sections, we work only with degradations caused by noise. Beginning in Section 5.5 we look at several methods for image restoration in the presence of both $\mathcal{H}$ and η .

3.2 Noise Models

The principal sources of noise in digital images arise during image acquisition and/or transmission. The performance of imaging sensors is affected by a variety of environmental factors during image acquisition, and by the quality of the sensing elements themselves. For instance, in acquiring images with a CCD camera, light levels and sensor temperature are major factors affecting the amount of noise in the resulting image. Images are corrupted during transmission principally by interference in the transmission channel. For example, an image transmitted using a wireless network might be corrupted by lightning or other atmospheric disturbance.

SPATIAL AND FREQUENCY PROPERTIES OF NOISE

Relevant to our discussion are parameters that define the spatial characteristics of noise, and whether the noise is correlated with the image. Frequency properties refer to the frequency content of noise in the Fourier (frequency). For example, when the Fourier spectrum of noise is constant, the noise is called *white noise*. This terminology is a carryover from the physical properties of white light, which contains all frequencies in the visible spectrum in equal proportions.

With the exception of spatially periodic noise, we assume in this chapter that noise is independent of spatial coordinates, and that it is uncorrelated with respect to the image itself (that is, there is no correlation between pixel values and the values of noise components). Although these assumptions are at least partially invalid in some applications (quantum-limited imaging, such as in X-ray and nuclear-medicine imaging, is a good example), the complexities of dealing with spatially dependent and correlated noise are beyond the scope of our discussion.

SOME IMPORTANT NOISE PROBABILITY DENSITY FUNCTIONS

In the discussion that follows, we shall be concerned with the statistical behavior of the intensity values in the noise component of the model in Fig. 5.1. These may be considered random variables, characterized by a probability density function (PDF), as noted briefly as noted earlier. We create a noise image for simulation purposes by generating an array whose intensity values are random numbers with a specified probability density function. This approach is true for all the PDFs to be discussed shortly, with the exception of salt-and-pepper noise, which is applied differently. The following are among the most common noise PDFs found in image processing applications.

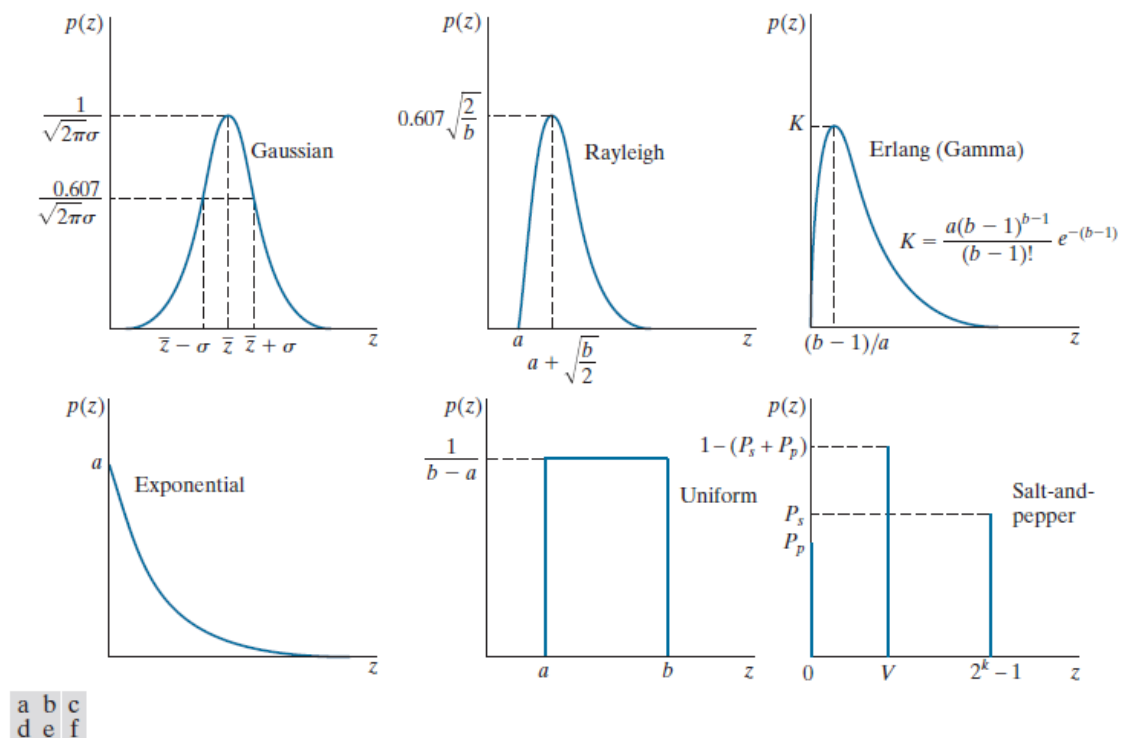


FIGURE 5.2 Some important probability density functions.

Gaussian Noise

Because of its mathematical tractability in both the spatial and frequency domains, Gaussian noise models are used frequently in practice. In fact, this tractability is so convenient that it often results in Gaussian models being used in situations in which they are marginally applicable at best.

The PDF of a *Gaussian* random variable, z , is defined by the following familiar

expression:

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z - \bar{z})^2}{2\sigma^2}} \quad -\infty < z < \infty \quad (5-3)$$

where z represents intensity, $\bar{z}$ is the mean (average) value of z , and σ is its standard deviation. Figure 5.2(a) shows a plot of this function. We know that for a Gaussian random variable, the probability that values of z are in the range $\bar{z} \pm \sigma$ is approximately 0.68; the probability is about 0.95 that the values of z are in the range $\bar{z} \pm 2\sigma$.

Rayleigh Noise

The PDF of *Rayleigh* noise is given by

$$p(z) = \begin{cases} \frac{2}{b}(z - a)e^{-(z - a)^2/b} & z \geq a \\ 0 & z < a \end{cases} \quad (5-4)$$

The mean and variance of z when this random variable is characterized by a Rayleigh PDF are

$$\bar{z} = a + \sqrt{\pi b/4} \quad (5-5)$$

and

$$\sigma^2 = \frac{b(4 - \pi)}{4} \quad (5-6)$$

Figure 5.2(b) shows a plot of the Rayleigh density. Note the displacement from the origin, and the fact that the basic shape of the density is skewed to the right. The Rayleigh density can be quite useful for modeling the shape of skewed histograms.

Erlang (Gamma) Noise

The PDF of Erlang noise is

$$p(z) = \begin{cases} \frac{a^b z^{b-1}}{(b-1)!} e^{-az} & z \geq 0 \\ 0 & z < 0 \end{cases} \quad (5-7)$$

where the parameters are such that $a > b$, b is a positive integer, and “!” indicates factorial. The mean and variance of z are

$$\bar{z} = \frac{b}{a} \quad (5-8)$$

and

$$\sigma^2 = \frac{b}{a^2} \quad (5-9)$$

Figure 5.2(c) shows a plot of this density. Although Eq. (5-9) often is referred to as the *gamma* density, strictly speaking this is correct only when the denominator is the gamma function, $\Gamma(b)$. When the denominator is as shown, the density is more appropriately called the *Erlang* density.

Exponential Noise

The PDF of *exponential* noise is given by

$$p(z) = \begin{cases} ae^{-az} & z \geq 0 \\ 0 & z < 0 \end{cases} \quad (5-10)$$

where $a > 0$. The mean and variance of z are

$$\bar{z} = \frac{1}{a} \quad (5-11)$$

and

$$\sigma^2 = \frac{1}{a^2} \quad (5-12)$$

Note that this PDF is a special case of the Erlang PDF with $b = 1$. Figure 5.2(d) shows a plot of the exponential density function.

Uniform Noise

The PDF of *uniform* noise is

$$p(z) = \begin{cases} \frac{1}{b-a} & a \leq z \leq b \\ 0 & \text{otherwise} \end{cases} \quad (5-13)$$

The mean and variance of z are

$$\bar{z} = \frac{a+b}{2} \quad (5-14)$$

and

$$\sigma^2 = \frac{(b-a)^2}{12} \quad (5-15)$$

Figure 5.2(e) shows a plot of the uniform density.

Salt-and-Pepper Noise

If k represents the number of bits used to represent the intensity values in a digital image, then the range of possible intensity values for that image is $[0, 2^k - 1]$ (e.g., $[0, 255]$ for an 8-bit image). The PDF of *salt-and-pepper* noise is given by

$$p(z) = \begin{cases} P_s & \text{for } z = 2^k - 1 \\ P_p & \text{for } z = 0 \\ 1 - (P_s + P_p) & \text{for } z = V \end{cases} \quad (5-16)$$

where V is any integer value in the range $0 < V < 2^k - 1$.

Let $\eta(x, y)$ denote a salt-and-pepper noise image, whose intensity values satisfy Eq. (5-16). Given an image, $f(x, y)$, of the same size as $\eta(x, y)$, we corrupt it with salt-and-pepper noise by assigning a 0 to all locations in f where a 0 occurs in η . Similarly, we assign a value of $2^k - 1$ to all location in f where that value appears in η . Finally, we leave unchanged all location in f where V occurs in η .

If neither P_s nor P_p is zero, and especially if they are equal, noise values satisfying Eq. (5-16) will be white ($2^k - 1$) or black (0), and will resemble salt and pepper granules distributed randomly over the image; hence the name of this type of noise. Other names you will find used in the literature are *bipolar impulse noise* (*unipolar* if either P_s or P_p is 0), *data-drop-out noise*, and *spike noise*. We use the terms impulse and salt-and-pepper noise interchangeably.

The probability, P , that a pixel is corrupted by salt or pepper noise is $P = P_s + P_p$. It is common terminology to refer to P as the *noise density*. If, for example, $P_s = 0.02$ and $P_p = 0.01$, then $P = 0.03$ and we say that approximately 2% of the pixels in an image are corrupted by salt noise, 1% are corrupted by pepper noise, and the noise density is 3%, meaning that approximately 3% of the pixels in the image are corrupted by salt-and-pepper noise.

Although, as you have seen, salt-and-pepper noise is specified by the probability of each, and not by the mean and variance, we include the latter here for completeness. The mean of salt-and-pepper noise is given by

$$\bar{z} = (0)P_p + K(1 - P_s - P_p) + (2^k - 1)P_s \quad (5-17)$$

and the variance by

$$\sigma^2 = (0 - \bar{z})^2 P_p + (K - \bar{z})^2 (1 - P_s - P_p) + (2^k - 1)^2 P_s \quad (5-18)$$

where we have included 0 as a value explicit in both equations to indicate that the value of pepper noise is assumed to be zero.

As a group, the preceding PDFs provide useful tools for modeling a broad range of noise corruption situations found in practice. For example, Gaussian noise arises in an image due to factors such as electronic circuit noise and sensor noise caused by poor illumination and/or high temperature. The Rayleigh density is helpful in characterizing noise phenomena in range imaging. The exponential and gamma densities find application in laser imaging. Impulse noise is found in situations where quick transients, such as faulty switching, take place during imaging. The uniform density is perhaps the least descriptive of practical situations. However, the uniform density is quite useful as the basis for numerous random number generators that are used extensively in simulations (Gonzalez, Woods, and Eddins [2009]).

PERIODIC NOISE

Periodic noise in images typically arises from electrical or electromechanical interference during image acquisition. This is the only type of spatially dependent noise we will consider in this chapter. periodic noise can be reduced significantly via frequency domain filtering.

Estimating Noise Parameters

1. Periodic Noise Estimation

- **Fourier Spectrum Inspection:**
 - Periodic noise often creates **spikes** in the frequency domain.
 - These spikes can be detected visually using the **Fourier transform** of the image.
- **Direct Spatial Analysis:**
 - Periodicity can sometimes be inferred directly from the image.
 - This approach is usually only effective in **simple cases**.
- **Automated Detection:**
 - Effective when:
 - Frequency spikes are **prominent**, or
 - **Prior knowledge** about interference frequency locations is available.

2. Estimating Noise PDF Parameters

- **Sensor-Based Information:**
 - Parameters might be **partially known** from **sensor specifications**.
- **Capturing “Flat” Images:**
 - Used when imaging system is available.
 - Example: Imaging a **uniform gray board** under even lighting.
 - The result reveals **system noise characteristics**.

3. Estimation from Existing Images

- When only processed images are available:
 - Extract small **homogeneous patches** from regions of **constant intensity**.
 - Analyze histograms from these patches.

4. Histogram Analysis Example

- Vertical strips cropped from Gaussian, Rayleigh, and uniform images.
- Histograms computed from the patches closely match the original noise types.

5. Calculating Mean and Variance

- **Given:** Subimage (strip) S
 - Let $p_S(z_i)$ = normalized histogram (probability estimate) of intensity z_i , for $i = 0, 1, 2, \dots, L - 1$, where L = number of possible intensity levels (e.g., 256).

♦ **Mean:**

$$\bar{z} = \sum_{i=0}^{L-1} z_i \cdot p_S(z_i) \quad (5-19)$$

♦ **Variance:**

$$\sigma^2 = \sum_{i=0}^{L-1} (z_i - \bar{z})^2 \cdot p_S(z_i) \quad (5-20)$$

6. Using Histogram Shape to Identify Noise Type

- Compare histogram shape to known **PDFs**:
 - **Gaussian**: Fully defined by mean and variance.
 - **Rayleigh, Uniform, etc.**: Use mean and variance to solve for parameters **a** and **b**.
-

7. Impulse Noise Estimation

- Requires presence of **both black and white pixels**.
- Use mid-gray areas to ensure visibility of both extremes.
- Histogram **peak heights** at 0 and 255 indicate:
 - P_a = probability of black pixels
 - P_b = probability of white pixels

3.3 Restoration in the presence of noise only – Spatial Filtering

When an image is degraded only by additive noise, Eqs. (5-1) and (5-2) become

$$g(x, y) = f(x, y) + \eta(x, y) \quad (5-21)$$

and

$$G(u, v) = F(u, v) + N(u, v) \quad (5-22)$$

Spatial filtering is the method of choice for estimating $f(x, y)$ [i.e., denoising image $g(x, y)$] in situations when only additive random noise is present.

MEAN FILTERS :

Arithmetic Mean Filter

Geometric Mean Filter

Harmonic Mean Filter

Contraharmonic Mean Filter

The *arithmetic mean filter* is the simplest of the mean filters. Let S_{xy} represent the set of coordinates in a rectangular subimage window (neighborhood) of size $m \times n$, centered on point (x, y) . The arithmetic mean filter computes the average value of

the corrupted image, $g(x, y)$, in the area defined by S_{xy} . The value of the restored image $\hat{f}$ at point (x, y) is the arithmetic mean computed using the pixels in the region defined by S_{xy} . In other words,

$$\hat{f}(x, y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} g(r, c) \quad (5-23)$$

where, as in Eq. (2-43), r and c are the row and column coordinates of the pixels contained in the neighborhood S_{xy} . This operation can be implemented using a spatial kernel of size $m \times n$ in which all coefficients have value $1/mn$. A mean filter smooths local variations in an image, and noise is reduced as a result of blurring.

Geometric Mean Filter

An image restored using a *geometric mean filter* is given by the expression

$$\hat{f}(x, y) = \left[\prod_{(r,c) \in S_{xy}} g(r, c) \right]^{\frac{1}{mn}} \quad (5-24)$$

where Π indicates multiplication. Here, *each* restored pixel is given by the product of *all* the pixels in the subimage area, raised to the power $1/mn$. As Example 5.2 below illustrates, a geometric mean filter achieves smoothing comparable to an arithmetic mean filter, but it tends to lose less image detail in the process.

Harmonic Mean Filter

The *harmonic mean* filtering operation is given by the expression

$$\hat{f}(x, y) = \frac{mn}{\sum_{(r,c) \in S_{xy}} \frac{1}{g(r, c)}} \quad (5-25)$$

The harmonic mean filter works well for salt noise, but fails for pepper noise. It does well also with other types of noise like Gaussian noise.

Contraharmonic Mean Filter

The *contraharmonic mean filter* yields a restored image based on the expression

$$\hat{f}(x, y) = \frac{\sum_{(r,c) \in S_{xy}} g(r, c)^{Q+1}}{\sum_{(r,c) \in S_{xy}} g(r, c)^Q} \quad (5-26)$$

where Q is called the *order* of the filter. This filter is well suited for reducing or virtually eliminating the effects of salt-and-pepper noise. For positive values of Q , the filter eliminates pepper noise. For negative values of Q , it eliminates salt noise. It cannot do both simultaneously. Note that the contraharmonic filter reduces to the arithmetic mean filter if $Q = 0$, and to the harmonic mean filter if $Q = -1$.

ORDER-STATISTIC FILTERS

Median Filter

The best-known order-statistic filter in image processing is the *median filter*, which, as its name implies, replaces the value of a pixel by the median of the intensity levels in a predefined neighborhood of that pixel:

$$\hat{f}(x, y) = \text{median}_{(r,c) \in S_{xy}} \{g(r, c)\} \quad (5-27)$$

where, as before, S_{xy} is a subimage (neighborhood) centered on point (x, y) . The value of the pixel at (x, y) is included in the computation of the median. Median filters

are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of both bipolar and unipolar impulse noise.

Max and Min Filters

Although the median filter is by far the order-statistic filter most used in image processing, it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers, but you will recall from basic statistics that ranking lends itself to many other possibilities. For example, using the 100th percentile results in the so-called *max filter*, given by

$$\hat{f}(x, y) = \max_{(r,c) \in S_{xy}} \{g(r, c)\} \quad (5-28)$$

This filter is useful for finding the brightest points in an image or for eroding dark regions adjacent to bright areas. Also, because pepper noise has very low values, it is reduced by this filter as a result of the max selection process in the subimage area S_{xy} .

The 0th percentile filter is the *min filter*:

$$\hat{f}(x, y) = \min_{(r,c) \in S_{xy}} \{g(r, c)\} \quad (5-29)$$

This filter is useful for finding the darkest points in an image or for eroding light regions adjacent to dark areas. Also, it reduces salt noise as a result of the min operation.

Midpoint Filter

The *midpoint filter* computes the midpoint between the maximum and minimum values in the area encompassed by the filter:

$$\hat{f}(x, y) = \frac{1}{2} \left[\max_{(r,c) \in S_{xy}} \{g(r, c)\} + \min_{(r,c) \in S_{xy}} \{g(r, c)\} \right] \quad (5-30)$$

Note that this filter combines order statistics and averaging. It works best for randomly distributed noise, like Gaussian or uniform noise.

Alpha-Trimmed Mean Filter

Suppose that we delete the $d/2$ lowest and the $d/2$ highest intensity values of $g(r, c)$ in the neighborhood S_{xy} . Let $g_R(r, c)$ represent the remaining $mn - d$ pixels in S_{xy} . A filter formed by averaging these remaining pixels is called an *alpha-trimmed mean filter*. The form of this filter is

$$\hat{f}(x, y) = \frac{1}{mn - d} \sum_{(r, c) \in S_{xy}} g_R(r, c) \quad (5-31)$$

where the value of d can range from 0 to $mn - 1$. When $d = 0$ the alpha-trimmed filter reduces to the arithmetic mean filter discussed earlier. If we choose $d = mn - 1$, the filter becomes a median filter. For other values of d , the alpha-trimmed filter is useful in situations involving multiple types of noise, such as a combination of salt-and-pepper and Gaussian noise.

ADAPTIVE FILTERS

Once selected, the filters discussed thus far are applied to an image without regard for how image characteristics vary from one point to another. In this section, we take a look at two *adaptive* filters whose behavior changes based on statistical characteristics of the image inside the filter region defined by the $m \times n$ rectangular neighborhood S_{xy} . As the following discussion shows, adaptive filters are capable of performance superior to that of the filters discussed thus far.

Adaptive, Local Noise Reduction Filter

The simplest statistical measures of a random variable are its mean and variance. These are reasonable parameters on which to base an adaptive filter because they are quantities closely related to the appearance of an image. The mean gives a measure of average intensity in the region over which the mean is computed, and the variance gives a measure of image contrast in that region.

Our filter is to operate on a neighborhood, S_{xy} , centered on coordinates (x, y) . The response of the filter at (x, y) is to be based on the following quantities: $g(x, y)$, the value of the noisy image at (x, y) ; σ_η^2 , the variance of the noise; $\bar{z}_{S_{xy}}$, the local average intensity of the pixels in S_{xy} ; and $\sigma_{S_{xy}}^2$, the local variance of the intensities of pixels in S_{xy} . We want the behavior of the filter to be as follows:

1. If σ_η^2 is zero, the filter should return simply the value of g at (x, y) . This is the trivial, zero-noise case in which g is equal to f at (x, y) .
2. If the local variance $\sigma_{S_{xy}}^2$ is high relative to σ_η^2 , the filter should return a value close to g at (x, y) . A high local variance typically is associated with edges, and these should be preserved.
3. If the two variances are equal, we want the filter to return the arithmetic mean value of the pixels in S_{xy} . This condition occurs when the local area has the same properties as the overall image, and local noise is to be reduced by averaging.

An adaptive expression for obtaining $\hat{f}(x, y)$ based on these assumptions may be written as

$$\hat{f}(x, y) = g(x, y) - \frac{\sigma_\eta^2}{\sigma_{S_{xy}}^2} [g(x, y) - \bar{z}_{S_{xy}}] \quad (5-32)$$

The only quantity that needs to be known a priori is σ_η^2 , the variance of the noise corrupting image $f(x, y)$. This is a constant that can be estimated from sample noisy images using Eq. (3-26). The other parameters are computed from the pixels in neighborhood S_{xy} using Eqs. (3-27) and (3-28).

Adaptive Median Filter

The median filter in Eq. (5-27) performs well if the spatial density of the salt-and pepper noise is low (as a rule of thumb, P_s and P_p less than 0.2). We show in the following discussion that adaptive median filtering can handle noise with probabilities larger than these. An additional benefit of the adaptive median filter is that it seeks to preserve detail while simultaneously

smoothing non-impulse noise, something that the “traditional” median filter does not do. As in all the filters discussed in the preceding sections, the adaptive median filter also works in a rectangular neighborhood, S_{xy} . Unlike those filters, however, the adaptive median filter changes (increases) the size of S_{xy} during filtering, depending on certain conditions to be listed shortly. Keep in mind that the output of the filter is a single value used to replace the value of the pixel at (x, y) , the point on which region S_{xy} is centered at a given time. We use the following notation:

$z_{\min}$ = minimum intensity value in S_{xy}
 $z_{\max}$ = maximum intensity value in S_{xy}
 z_{med} = median of intensity values in S_{xy}
 z_{xy} = intensity at coordinates (x, y)
 $S_{\max}$ = maximum allowed size of S_{xy}

The adaptive median-filtering algorithm uses two processing levels, denoted level A and level B , at each point (x, y) :

Level A : If $z_{\min} < z_{\text{med}} < z_{\max}$, go to Level B
 Else, increase the size of S_{xy}
 If $S_{xy} \leq S_{\max}$, repeat level A
 Else, output z_{med} .
Level B : If $z_{\min} < z_{xy} < z_{\max}$, output z_{xy}
 Else output z_{med} .

where S_{xy} and $S_{\max}$ are odd, positive integers greater than 1. Another option in the last step of level A is to output z_{xy} instead of z_{med} . This produces a slightly less blurred result, but can fail to detect salt (pepper) noise embedded in a constant background having the same value as pepper (salt) noise.

This algorithm has three principal objectives: to remove salt-and-pepper (impulse) noise, to provide smoothing of other noise that may not be impulsive, and to reduce distortion, such as excessive thinning or thickening of object boundaries. The values $z_{\min}$ and $z_{\max}$ are considered statistically by the algorithm to be “impulse-like” noise components in region S_{xy} , even if these are not the lowest and highest possible pixel values in the image.

With these observations in mind, we see that the purpose of level *A* is to determine if the median filter output, z_{med} , is an impulse (salt or pepper) or not. If the condition $z_{\text{min}} < z_{\text{med}} < z_{\text{max}}$ holds, then z_{med} cannot be an impulse for the reason mentioned in the previous paragraph. In this case, we go to level *B* and test to see if the point in the center of the neighborhood is itself an impulse (recall that (x, y) is the location of the point being processed, and z_{xy} is its intensity). If the condition $z_{\text{min}} < z_{xy} < z_{\text{max}}$ is true, then the pixel at z_{xy} cannot be the intensity of an impulse for the same reason that z_{med} was not. In this case, the algorithm outputs the unchanged pixel value, z_{xy} . By not changing these “intermediate-level” points, distortion is reduced in the filtered image. If the condition $z_{\text{min}} < z_{xy} < z_{\text{max}}$ is false, then either $z_{xy} = z_{\text{min}}$ or $z_{xy} = z_{\text{max}}$. In either case, the value of the pixel is an extreme value and the algorithm outputs the median value, z_{med} , which we know from level *A* is not a noise impulse. The last step is what the standard median filter does. The problem is that the standard median filter replaces every point in the image by the median of the corresponding neighborhood. This causes unnecessary loss of detail.

Continuing with the explanation, suppose that level *A* *does* find an impulse (i.e., it fails the test that would cause it to branch to level *B*). The algorithm then increases the size of the neighborhood and repeats level *A*. This looping continues until the algorithm either finds a median value that is not an impulse (and branches to stage *B*), or the maximum neighborhood size is reached. If the maximum size is reached, the algorithm returns the value of z_{med} . Note that there is no guarantee that this value is not an impulse. The smaller the noise probabilities P_a and/or P_b are, or the larger S_{max} is allowed to be, the less likely it is that a premature exit will occur. This is plausible. As the density of the noise impulses increases, it stands to reason that we would need a larger window to “clean up” the noise spikes.

Every time the algorithm outputs a value, the center of neighborhood S_{xy} is moved to the next location in the image. The algorithm then is reinitialized and applied to the pixels in the new region encompassed by the neighborhood. As indicated in Problem 3.37, the median value can be updated iteratively from one location to the next, thus reducing computational load.

3.4 Periodic noise reduction using Frequency domain Filtering :

Periodic noise can be analyzed and filtered quite effectively using frequency domain techniques. The basic idea is that periodic noise appears as concentrated bursts of energy in the Fourier transform, at locations corresponding to the frequencies of the periodic interference. The approach is to use a selective filter to isolate the noise. The three types of selective filters (bandreject, bandpass, and notch). There is no difference between how these filters were used, and the way they are used for image restoration. In restoration of images corrupted by periodic interference, the tool of choice is a notch filter.

MORE ON NOTCH FILTERING

Notch reject filter transfer functions are constructed as products of highpass filter transfer functions whose centers have been translated to the centers of the notches. The general form of a notch filter transfer function is

$$H_{NR}(u, v) = \prod_{k=1}^Q H_k(u, v) H_{-k}(u, v) \quad (5-33)$$

where $H_k(u, v)$ and $H_{-k}(u, v)$ are highpass filter transfer functions whose centers are at (u_k, v_k) and $(-u_k, -v_k)$, respectively.[†] These centers are specified with respect to the center of the frequency rectangle, $[\text{floor}(M/2), \text{floor}(N/2)]$, where, as usual, M and N are the number of rows and columns in the input image. Thus, the distance computations for the filter transfer functions are given by

$$D_k(u, v) = [(u - M/2 - u_k)^2 + (v - N/2 - v_k)^2]^{1/2} \quad (5-34)$$

and

$$D_{-k}(u, v) = [(u - M/2 + u_k)^2 + (v - N/2 + v_k)^2]^{1/2} \quad (5-35)$$

For example, the following is a Butterworth notch reject filter transfer function of order n with three notch pairs:

$$H_{NR}(u, v) = \prod_{k=1}^3 \left[\frac{1}{1 + [D_{0k}/D_k(u, v)]^n} \right] \left[\frac{1}{1 + [D_{0k}/D_{-k}(u, v)]^n} \right] \quad (5-36)$$

a notch pass filter transfer function is obtained from a notch reject function using the expression

$$H_{NP}(u, v) = 1 - H_{NR}(u, v) \quad (5-37)$$

OPTIMUM NOTCH FILTERING

When periodic interference in an image is **complex or composed of multiple frequency components**, simple heuristic notch filters may remove important image data. In such cases, an **optimum filtering** approach that minimizes local **variance** in the estimated image is more effective.

- **Corrupted Image:**

$$g(x, y) = f(x, y) + h(x, y)$$

where

- $f(x, y)$: original (true) image
- $h(x, y)$: periodic interference pattern
- **Goal:** Estimate $f(x, y)$ as $\hat{f}(x, y)$, minimizing local variance.

- **Weighted Subtraction:**

Instead of directly subtracting estimated noise $h(x, y)$, subtract a **modulated portion** using a **weighting function** $w(x, y)$:

$$\hat{f}(x, y) = g(x, y) - w(x, y)h(x, y) \quad (5-40)$$

1. Identify Noise Components in the Frequency Domain

- Convert the noisy image to the **Fourier domain** using **DFT**: $G(u,v)$
- Noise appears as **spikes at specific frequencies**.
- We isolate these using a **notch pass filter** $H_{NP}(u,v)$ that **keeps only the interference** while rejecting the rest.

2. Extract the Noise Pattern

- Multiply $H_{NP}(u,v)$ with the transformed noisy image $G(u,v)$ to extract the interference pattern in the frequency domain:

$$N(u,v) = H_{NP}(u,v) G(u,v)$$

- Convert it **back to the spatial domain** using the **Inverse Fourier Transform**:

$$h(x,y) = F^{-1} \{H_{NP}(u,v) G(u,v)\}$$

- Now, $h(x,y)$ represents the **estimated noise pattern** in the image.

3. Remove the Noise from the Image

- Since the noisy image $g(x,y)$ consists of the **true image** $f(x,y)$ plus **noise** $h(x,y)$:

$$g(x,y) = f(x,y) + h(x,y)$$

- We subtract a **weighted portion of the noise** to get an **estimate** $\hat{f}(x,y)$:

$$\hat{f}(x,y) = g(x,y) - w(x,y) h(x,y)$$

- Here, $w(x,y)$ is the **weighting function** that decides **how much noise to subtract**.

4. Optimize the Filtering Process

To find the best $w(x,y)$, we **minimize the local variance** of $\hat{f}(x,y)$, ensuring that the remaining noise is as small as possible.

- The **local variance** in a neighborhood S_{xy} of size $m \times n$ is:

$$s^2(x,y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} [\hat{f}(r,c) - \bar{\hat{f}}]^2$$

- The **average value** of $\hat{f}(x,y)$ in S_{xy} is:

$$\bar{\hat{f}} = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} \hat{f}(r,c)$$

5. Derive the Optimal Weight Function $w(x,y)$

- Substituting $\hat{f}(x,y)$:

$$s^2(x,y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} [g(r,c) - wh(r,c) - \bar{g} + w\bar{h}]^2$$

- Assuming $w(x,y)$ is approximately **constant** in S_{xy} :

$$w(x, y) = \frac{\bar{g} - \bar{h}}{\bar{h}^2}$$

- This formula gives the **optimal value of $w(x,y)$** that minimizes noise while keeping the image quality.

6. Final Image Restoration

- Compute $w(x,y)$ using $w(x, y) = \frac{\bar{g} - \bar{h}}{\bar{h}^2}$ for every pixel.
- Substitute it into $\hat{f}(x, y) = g(x, y) - w(x, y)h(x, y)$ to get the **final restored image** $\hat{f}(x,y)$.

Optimum Notch Filtering is useful when multiple periodic noise patterns exist. Instead of blindly removing noise, it **selectively reduces interference** while keeping image details. The method minimizes **local variance** to ensure optimal filtering. This technique is widely used in **medical imaging, satellite images, and document restoration**, where preserving image details is critical.

Chapter : Image Segmentation

3.1 Fundamentals :

Let R represent the entire spatial region occupied by an image. We may view image segmentation as a process that partitions R into n subregions, $R_1, R_2, \dots, R_n$ such that

$$(a) \bigcup_{i=1}^n R_i = R.$$

(b) R_i is a connected set, for $i = 0, 1, 2, \dots, n$.

(c) $R_i \cap R_j = \emptyset$ for all i and j , $i \neq j$.

(d) $Q(R_i) = \text{TRUE}$ for $i = 0, 1, 2, \dots, n$.

(e) $Q(R_i \cup R_j) = \text{FALSE}$ for any adjacent regions R_i and R_j .

where $Q(R_k)$ is a logical predicate defined over the points in set R_k , and $\emptyset$ is the null set. The symbols $\cup$ and $\cap$ represent set union and intersection, respectively, as defined in Section 2.6. Two regions R_i and R_j are said to be *adjacent* if their union forms a connected set, as defined in Section 2.5. If the set formed by the union of two regions is not connected, the regions are said to be *disjoint*.

Condition (a) indicates that the segmentation must be *complete*, in the sense that every pixel must be in a region. Condition (b) requires that points in a region be connected in some predefined sense (e.g., the points must be 8-connected). Condition (c) says that the regions must be disjoint. Condition (d) deals with the properties that must be satisfied by the pixels in a segmented region—for example, $Q(R_i) = \text{TRUE}$ if all pixels in R_i have the same intensity. Finally, condition (e) indicates that two adjacent regions R_i and R_j must be different in the sense of predicate Q .[†]

Thus, we see that the fundamental problem in segmentation is to partition an image into regions that satisfy the preceding conditions. Segmentation algorithms for monochrome images generally are based on one of two basic categories dealing with properties of intensity values: *discontinuity* and *similarity*. In the first category, we assume that boundaries of regions are sufficiently different from each other, and from the background, to allow boundary detection based on local discontinuities in intensity. *Edge-based* segmentation is the principal approach used in this category. *Region-based* segmentation approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria.

3.2 Point, Line and Edge detection

The focus of this section is on segmentation methods that are based on detecting sharp, *local* changes in intensity. The three types of image characteristics in which we are interested are isolated points, lines, and edges. *Edge pixels* are pixels at which the intensity of an image changes abruptly, and *edges* (or *edge segments*) are sets of connected edge pixels. *Edge detectors* are local image processing tools designed to detect edge pixels. A *line* may be viewed as a (typically) thin edge segment in which the intensity of the background on either side of the line is either much higher or much lower than the intensity of the line pixels. In fact, as we will

discuss later, lines give rise to so-called “roof edges.” Finally, an *isolated point* may be viewed as a foreground (background) pixel surrounded by background (foreground) pixels.

- Given that averaging is analogous to integration, it is intuitive that abrupt, local changes in intensity can be detected using derivatives.
- Derivatives of a digital function are defined in terms of finite differences.
- We require that any approximation used for first derivatives
 - (1) must be zero in areas of constant intensity;
 - (2) must be nonzero at the onset of an intensity step or ramp;
 - (3) must be nonzero at points along an intensity ramp.
- Similarly, we require that an approximation used for second derivatives
 - (1) must be zero in areas of constant intensity;
 - (2) must be nonzero at the onset and end of an intensity step or ramp;
 - (3) must be zero along intensity ramps.

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

We obtain an approximation to the first-order derivative at an arbitrary point x of a one-dimensional function $f(x)$ by expanding the function $f(x + \Delta x)$ into a Taylor series about x

$$\begin{aligned} f(x + \Delta x) &= f(x) + \Delta x \frac{\partial f(x)}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f(x)}{\partial x^2} + \frac{(\Delta x)^3}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} \frac{\partial^n f(x)}{\partial x^n} \end{aligned} \quad (10-1)$$

where Δx is the separation between samples of f . For our purposes, this separation is measured in pixel units. Thus, following the convention in the book, $\Delta x = 1$ for the sample preceding x and $\Delta x = -1$ for the sample following x . When $\Delta x = 1$, Eq. (10-1) becomes

$$\begin{aligned} f(x + 1) &= f(x) + \frac{\partial f(x)}{\partial x} + \frac{1}{2!} \frac{\partial^2 f(x)}{\partial x^2} + \frac{1}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} \frac{\partial^n f(x)}{\partial x^n} \end{aligned} \quad (10-2)$$

Similarly, when $\Delta x = -1$,

$$\begin{aligned} f(x - 1) &= f(x) - \frac{\partial f(x)}{\partial x} + \frac{1}{2!} \frac{\partial^2 f(x)}{\partial x^2} - \frac{1}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \frac{\partial^n f(x)}{\partial x^n} \end{aligned} \quad (10-3)$$

In what follows, we compute *intensity differences* using just a few terms of the Taylor series. For first-order derivatives we use only the linear terms, and we can form differences in one of three ways.

The *forward difference* is obtained from Eq. (10-2):

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x+1) - f(x) \quad (10-4)$$

where, as you can see, we kept only the linear terms. The *backward difference* is similarly obtained by keeping only the linear terms in Eq. (10-3):

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x) - f(x-1) \quad (10-5)$$

and the *central difference* is obtained by subtracting Eq. (10-3) from Eq. (10-2):

$$\frac{\partial f(x)}{\partial x} = f'(x) = \frac{f(x+1) - f(x-1)}{2} \quad (10-6)$$

-

The *second order derivative* based on a central difference, $\partial^2 f(x)/\partial x^2$, is obtained by adding Eqs. (10-2) and (10-3):

$$\frac{\partial^2 f(x)}{\partial x^2} = f''(x) = f(x+1) - 2f(x) + f(x-1) \quad (10-7)$$

- **Third- and Fourth-Order Derivatives**

These require values at $f(x \pm 2)$ and are derived by **eliminating lower-order terms** from the Taylor expansions.

- ◆ **Third-Order Derivative – Eq. (10-8):**

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

- ◆ **Fourth-Order Derivative – Eq. (10-9):**

$$\frac{\partial^4 f(x)}{\partial x^4} \approx f(x+2) - 4f(x+1) + 6f(x) - 4f(x-1) + f(x-2)$$

 These derivatives also show **symmetric coefficients**, contributing to more accurate approximations.

- **Extension to 2D Images (Functions of Two Variables)**

To apply derivatives in 2D images $f(x, y)$, we treat the x and y directions **independently**.

- ◆ **Second derivative in x-direction – Eq. (10-10):**

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

- ◆ **Second derivative in y-direction – Eq. (10-11):**

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

Using Derivative Sign to Classify Edges

The **sign of the second derivative** helps determine edge direction:

- **Negative second derivative** → **Light to dark** transition
- **Positive second derivative** → **Dark to light** transition

This is useful for **edge classification** and **object boundary analysis**.

In summary, we arrive at the following conclusions:

- (1) First-order derivatives generally produce thicker edges.
- (2) Second-order derivatives have a stronger response to fine detail, such as thin lines, isolated points, and noise.
- (3) Second-order derivatives produce a double-edge response at ramp and step transitions in intensity.
- (4) The sign of the second derivative can be used to determine whether a transition into an edge is from light to dark or dark to light.

Computing Derivatives via Convolution

In digital images, derivatives are computed via **spatial convolution** using small filter kernels (e.g., Sobel, Prewitt, Laplacian).

General Convolution Equation:

$$Z = \sum_{k=1}^9 w_k z_k$$

Where:

- z_k = pixel intensity in the 3×3 neighborhood
- w_k = kernel coefficient (from filter mask)

The **output Z** is the **response of the filter** at the center of the kernel.

$$\begin{aligned} Z &= w_1 z_1 + w_2 z_2 + \dots + w_9 z_9 \\ &= \sum_{k=1}^9 w_k z_k \end{aligned}$$

DETECTION OF ISOLATED POINTS

Based on the conclusions reached in the preceding section, we know that point detection should be based on the second derivative which, means using the Laplacian:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (10-13)$$

where the partial derivatives are computed using the second-order finite differences in Eqs.

For example, we can approximate the **second derivative in the x-direction** like this:

$$\frac{\partial^2 f}{\partial x^2} \approx f(x+1, y) + f(x-1, y) - 2f(x, y)$$

Same for y-direction:

$$\frac{\partial^2 f}{\partial y^2} \approx f(x, y+1) + f(x, y-1) - 2f(x, y)$$

Now, **add them together** to get the Laplacian:

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

Therefore Laplacian is then

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \quad (10-14)$$

• Binary Output with Thresholding

To convert the response into a **binary image**, we compare the absolute value of the Laplacian response to a **threshold T**:

$$g(x, y) = \begin{cases} 1, & \text{if } |Z(x, y)| > T \\ 0, & \text{otherwise} \end{cases} \quad (10-15)$$

This gives:

- 1 if an isolated point is detected
- 0 otherwise

In the example, if we set $T = 150$, then:

$$|Z| = 360 > 150 \Rightarrow g(x, y) = 1$$

All other pixels would yield much smaller Z values, hence $g(x, y) = 0$ elsewhere.

This gives us a **binary image** where only the isolated point is marked.

Line Detection

The next level of complexity is line detection. We know that for line detection we can expect second derivatives to result in a stronger filter response, and to produce thinner lines than first derivatives. Thus, we can use the Laplacian kernel for line detection also, keeping in mind that the double-line effect of the second derivative must be handled properly.

The following example illustrates the procedure.

-1	-1	-1	2	-1	-1	-1	2	-1	-1	-1	2
2	2	2	-1	2	-1	-1	2	-1	-1	2	-1
-1	-1	-1	-1	-1	2	-1	2	-1	2	-1	-1
Horizontal			+45°			Vertical			-45°		

To detect lines in specific directions, **directional masks** are used. These masks:

- Emphasize specific line orientations: **0° (horizontal), +45°, 90° (vertical), -45°**
- Have **larger central coefficients (e.g., 2)** in the target direction
- Coefficients **sum to zero**, ensuring **no response in uniform regions**

Filtering procedure

Let the filtered outputs be:

- $Z_1 \rightarrow$ horizontal kernel response
- $Z_2 \rightarrow +45^\circ$ kernel response
- $Z_3 \rightarrow$ vertical kernel response
- $Z_4 \rightarrow -45^\circ$ kernel response

For each image pixel, compute:

$$Z_k = \sum_{i,j} K_k(i,j) \cdot g(x+i, y+j)$$

where:

- K_k : kernel for direction k
- $g(x+i, y+j)$: pixel in the neighborhood

Line direction classification:

At a pixel (x, y) , compare responses from all 4 kernels:

If

$$Z_k > Z_j \quad \text{for all } j \neq k,$$

then the line at that pixel is **most likely in direction k** .

E.g., if $Z_1 > Z_2, Z_3, Z_4$, then the pixel is on a **horizontal line**.

Thresholding for final line map :

To finalize line detection in a given direction:

$$L_k(x, y) = \begin{cases} 1, & \text{if } |Z_k(x, y)| \geq T \\ 0, & \text{otherwise} \end{cases}$$

Where:

- $L_k(x, y)$: binary line map in direction k
- T : chosen threshold value

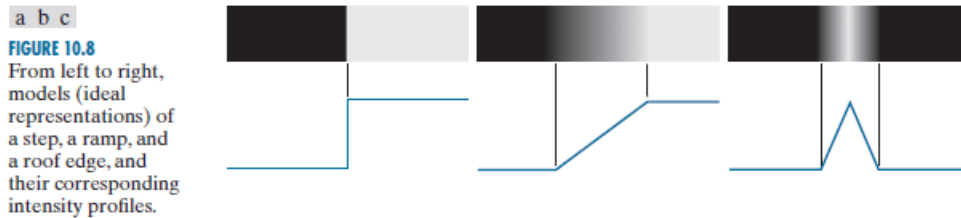
Thresholding retains only the strongest directional line responses.

Implementation steps:

1. **Choose Directional Kernels** ($0^\circ, \pm 45^\circ, 90^\circ$)
2. **Convolve Each Kernel** with the input image $\rightarrow$ get Z_1, Z_2, Z_3, Z_4
3. **Compare Responses** at each pixel to assign dominant direction
4. Optionally **Threshold** responses to generate binary line maps

EDGE MODELS

Edge detection is an approach used frequently for segmenting images based on abrupt (local) changes in intensity. We begin by introducing several ways to model edges and then discuss a number of approaches for edge detection.



Characterized by a transition between two intensity levels occurring ideally over the distance of one pixel.

Figure 10.8(a) shows a section of a vertical step edge and a horizontal intensity profile through the edge. Step edges occur, for example, in images generated by a computer for use in areas such as solid modeling and animation. These clean, *ideal* edges can occur over the distance of one pixel, provided that no additional processing (such as smoothing) is used to make them look “real.” Digital step edges are used frequently as edge models in algorithm development. For example, the Canny edge detection algorithm was derived originally using a step-edge model. In practice, digital images have edges that are blurred and noisy, with the degree of blurring determined principally by limitations in the focusing mechanism (e.g., lenses in the case of optical images), and the noise level determined principally by the electronic components of the imaging system. In such situations, edges are more The slope of the ramp is inversely proportional to the degree to which the edge is blurred. In this model, we no longer have a single “edge point” along the profile. Instead, an edge point now is any point contained in the ramp, and an edge segment would then be a set of such points that are connected.

A third type of edge is the so-called *roof edge*, having the characteristics illustrated in Fig. 10.8(c). Roof edges are models of lines through a region, with the base (width) of the edge being determined by the thickness and sharpness of the line. In the limit, when its base is one pixel wide, a roof edge is nothing more than a one-pixel-thick line running through a region in an image. Roof edges arise, for example, in range imaging, when thin objects (such as pipes) are closer to the sensor than the background (such as walls). The pipes appear brighter and thus create an image similar to the model in Fig. 10.8(c). Other areas in which roof edges appear routinely are in the digitization of line drawings and also in satellite images, where thin features, such as roads, can be modeled by this type of edge. It is not unusual to find images that contain all three types of edges.

a b

FIGURE 10.10
(a) Two regions of constant intensity separated by an ideal ramp edge.
(b) Detail near the edge, showing a horizontal intensity profile, and its first and second derivatives.

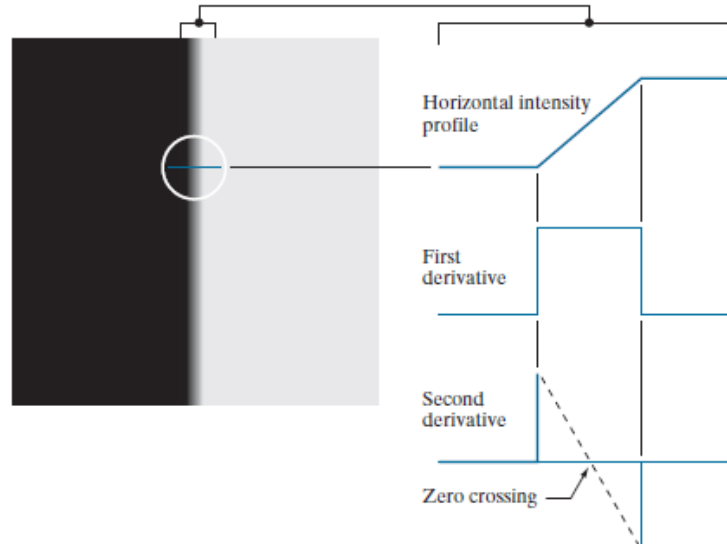


Figure 10.10(a) shows the image from which the segment in Fig. 10.8(b) was extracted.

Figure 10.10(b) shows a horizontal intensity profile. This figure shows also the first and second derivatives of the intensity profile. Moving from left to right along the intensity profile, we note that the first derivative is positive at the onset of the ramp and at points on the ramp, and it is zero in areas of constant intensity. The second derivative is positive at the beginning of the ramp, negative at the end of the ramp, zero at points on the ramp, and zero at points of constant intensity. The signs of the derivatives just discussed would be reversed for an edge that transitions from light to dark. The intersection between the zero intensity axis and a line extending between the extrema of the second derivative marks a point called the *zero crossing* of the second derivative.

We conclude from these observations that the *magnitude* of the first derivative can be used to detect the presence of an edge at a point in an image. Similarly, the *sign* of the second derivative can be used to determine whether an edge pixel lies on the dark or light side of an edge.

Two additional properties of the second derivative around an edge are:

- (1) it produces two values for every edge in an image; and
- (2) its zero crossings can be used for locating the centers of thick edges

In summary, the three steps performed typically for edge detection are:

1. *Image smoothing for noise reduction.* The need for this step is illustrated by the results in the second and third columns of Fig. 10.11.
2. *Detection of edge points.* As mentioned earlier, this is a local operation that extracts from an image all points that are potential edge-point candidates.
3. *Edge localization.* The objective of this step is to select from the candidate points only the points that are members of the set of points comprising an edge.

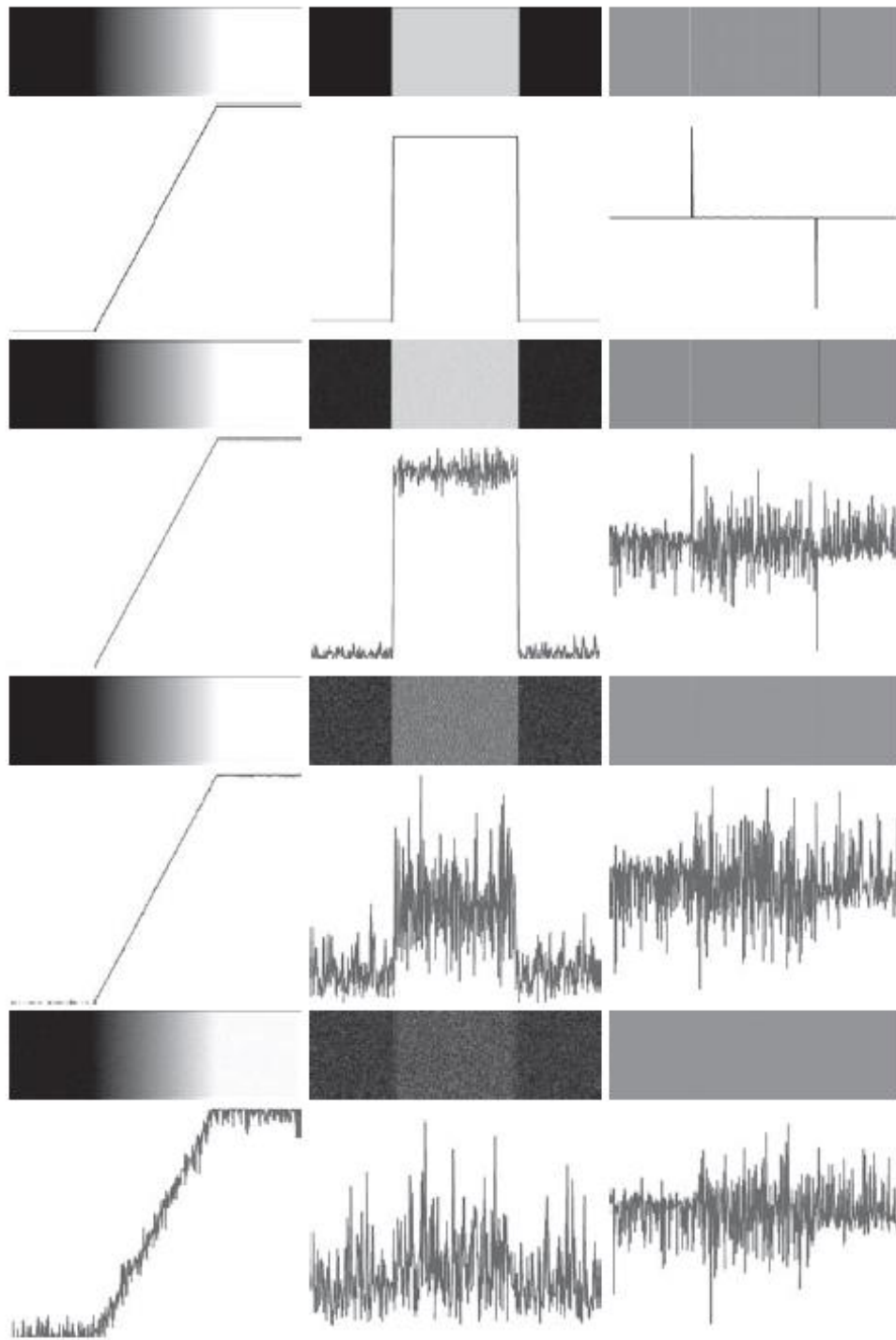


FIGURE 10.11 First column: 8-bit images with values in the range $[0,255]$, and intensity profiles of a ramp edge corrupted by Gaussian noise of zero mean and standard deviations of 0.0, 0.1, 1.0, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.

BASIC EDGE DETECTION

- Edges happen where the **intensity** (brightness) of pixels **changes quickly**—like going from dark to light.
- We use **derivatives** (like in math) to help us detect those changes.
- detecting changes in intensity for the purpose of finding edges can be accomplished using first- or second-order derivatives.
- We begin with first-order derivatives, and work with second-order derivatives in the following subsection.

The Image Gradient and Its Properties

- The gradient is written like this:

$$\nabla f(x, y) \equiv \text{grad}[f(x, y)] \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

Here:

- $g_x(x, y)$ = change of brightness in the **x-direction** (left–right)
 - $g_y(x, y)$ = change in the **y-direction** (up–down)
- The *magnitude*, $M(x, y)$, of this gradient vector at a point (x, y) is given by its Euclidean vector norm:

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

If g_x and g_y are big $\Rightarrow$ brightness is changing a lot $\Rightarrow$ likely an edge!

- The direction of the gradient vector at a point (x, y) is given by

$$\alpha(x, y) = \tan^{-1} \left[\frac{g_y(x, y)}{g_x(x, y)} \right]$$

This tells us **which way the edge is pointing**.

- Think of an arrow pointing in the direction where brightness changes fastest. That's the **gradient direction**.

GRADIENT OPERATORS

- Obtaining the gradient of an image requires computing the partial derivatives $\partial f / \partial x$ and $\partial f / \partial y$ at every pixel location in the image.
- Using forward differences we obtain

$$g_x(x, y) = \frac{\partial f(x, y)}{\partial x} = f(x+1, y) - f(x, y) \quad (10-19)$$

$$g_y(x, y) = \frac{\partial f(x, y)}{\partial y} = f(x, y+1) - f(x, y)$$

- This simply means: subtract the current pixel from its right neighbor for g_x , and from its bottom neighbor for g_y .

-1	-1	1
1		

These two equations can be implemented for all values of x and y by filtering $f(x, y)$ with the 1-D kernels.

a
b c
d e
f g

FIGURE 10.14
A 3×3 region of an image (the z 's are intensity values), and various kernels used to compute the gradient at the point labeled z_5 .

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

-1	0	0	-1
0	1	1	0

Roberts

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Prewitt

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

Sobel

When diagonal edge direction is of interest, we need 2-D kernels.

- The *Roberts cross-gradient operators* (Roberts [1965]) are one of the earliest attempts to use 2-D kernels with a diagonal preference

$$g_x = z_9 - z_5$$

$$g_y = z_8 - z_6$$

where z_i are pixel values in a 3×3 neighborhood numbered like this:

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

- These work well for sharp diagonal transitions but are not symmetrical, so they lack full directional context.

3×3 Kernels – More Accurate Derivatives :

Instead of 2×2 , we can use **3×3 kernels**, which give better context.

- **Prewitt Operator**

$$g_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

$$g_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

Sobel Operator

- Slight upgrade to Prewitt: gives more **weight (2x)** to the center pixels for **smoothing**:

$$g_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

- Center pixels are often more reliable (less noise), so Sobel filters reduce the impact of noise.

Gradient Magnitude

- We combine g_x and g_y to get **edge strength**:

$$M(x, y) = \sqrt{g_x^2 + g_y^2}$$

- Approximation

$$M(x, y) \approx |g_x| + |g_y|$$

The *Kirsch compass kernels*, are designed to detect edge magnitude *and* direction (angle) in all eight compass directions. Kirsch's approach was to determine the edge magnitude by convolving an image with all eight kernels and assign the edge magnitude at a point as the response of the kernel that gave strongest convolution value at that point. The edge angle at that point is then the direction associated with that kernel. For example, if the strongest value at a point in the image resulted from using the north (N) kernel, the edge magnitude at that point would be assigned the response of that kernel, and the direction would be 0° .

a b c d
e f g h

FIGURE 10.15
Kirsch compass
kernels. The edge
direction of
strongest response
of each kernel is
labeled below it.

<table><tr><td>-3</td><td>-3</td><td>5</td></tr><tr><td>-3</td><td>0</td><td>5</td></tr><tr><td>-3</td><td>-3</td><td>5</td></tr></table>	-3	-3	5	-3	0	5	-3	-3	5	<table><tr><td>-3</td><td>5</td><td>5</td></tr><tr><td>-3</td><td>0</td><td>5</td></tr><tr><td>-3</td><td>-3</td><td>-3</td></tr></table>	-3	5	5	-3	0	5	-3	-3	-3	<table><tr><td>5</td><td>5</td><td>5</td></tr><tr><td>-3</td><td>0</td><td>-3</td></tr><tr><td>-3</td><td>-3</td><td>-3</td></tr></table>	5	5	5	-3	0	-3	-3	-3	-3	<table><tr><td>5</td><td>5</td><td>-3</td></tr><tr><td>5</td><td>0</td><td>-3</td></tr><tr><td>-3</td><td>-3</td><td>-3</td></tr></table>	5	5	-3	5	0	-3	-3	-3	-3
-3	-3	5																																					
-3	0	5																																					
-3	-3	5																																					
-3	5	5																																					
-3	0	5																																					
-3	-3	-3																																					
5	5	5																																					
-3	0	-3																																					
-3	-3	-3																																					
5	5	-3																																					
5	0	-3																																					
-3	-3	-3																																					
N	NW	W	SW																																				
<table><tr><td>5</td><td>-3</td><td>-3</td></tr><tr><td>5</td><td>0</td><td>-3</td></tr><tr><td>5</td><td>-3</td><td>-3</td></tr></table>	5	-3	-3	5	0	-3	5	-3	-3	<table><tr><td>-3</td><td>-3</td><td>-3</td></tr><tr><td>5</td><td>0</td><td>-3</td></tr><tr><td>5</td><td>5</td><td>-3</td></tr></table>	-3	-3	-3	5	0	-3	5	5	-3	<table><tr><td>-3</td><td>-3</td><td>-3</td></tr><tr><td>-3</td><td>0</td><td>-3</td></tr><tr><td>5</td><td>5</td><td>5</td></tr></table>	-3	-3	-3	-3	0	-3	5	5	5	<table><tr><td>-3</td><td>-3</td><td>-3</td></tr><tr><td>-3</td><td>0</td><td>5</td></tr><tr><td>-3</td><td>5</td><td>5</td></tr></table>	-3	-3	-3	-3	0	5	-3	5	5
5	-3	-3																																					
5	0	-3																																					
5	-3	-3																																					
-3	-3	-3																																					
5	0	-3																																					
5	5	-3																																					
-3	-3	-3																																					
-3	0	-3																																					
5	5	5																																					
-3	-3	-3																																					
-3	0	5																																					
-3	5	5																																					
S	SE	E	NE																																				

Combining the Gradient with Thresholding

1. Gradient Image

- The **gradient** of an image represents the **rate of change in intensity**.
- Strong gradients = likely edge locations.

Gradient magnitude image (from methods like Sobel, Prewitt, etc.) often contains both weak and strong edges.

2. Thresholding the Gradient Image

Basic Thresholding Procedure:

- Choose a threshold value TTT (often a **percentage** of the **maximum gradient magnitude**).
- Set:

$$E(x, y) = \begin{cases} 1, & \text{if } \nabla f(x, y) \geq T \\ 0, & \text{otherwise} \end{cases}$$

Where:

- $\nabla f(x, y)$: gradient magnitude at point (x, y)
- $E(x, y)$: binary edge map (1 = edge, 0 = no edge)

3. Effect of Thresholding Alone

Thresholding helps remove **low-magnitude gradients** (often noise).

4. Smoothing Before Gradient Computation

- **Pre-smoothing** reduces noise **before** computing the gradient.
- Usually done using a **Gaussian filter**:

$$f_s(x, y) = f(x, y) * G(x, y)$$

- Where $G(x, y)$ is a Gaussian kernel.
- Gradient is then computed on the smoothed image:

$$\nabla f_s(x, y) = \sqrt{\left(\frac{\partial f_s}{\partial x}\right)^2 + \left(\frac{\partial f_s}{\partial y}\right)^2}$$

5. Smoothing + Thresholding (Combined Approach)

- Enhances edge sharpness and connectivity.

Example :

- **Smoothing first** (as in Fig. 10.18(d))
- **Then thresholding**
- **Result:** fewer broken edges, improved continuity in weak-gradient areas (e.g., sloped edge of the roof)

Implementation Workflow

1. Start with original image $f(x, y)$
2. Apply smoothing (e.g., Gaussian blur):
$$f_s(x, y) = f(x, y) * G(x, y)$$
3. Compute gradient magnitude:
$$\nabla f_s(x, y)$$
4. Choose a threshold T (e.g., 33% of max gradient value)
5. Threshold the gradient image to obtain binary edge map

MORE ADVANCED TECHNIQUES FOR EDGE DETECTION

1. Marr-Hildreth Edge Detector (Laplacian of Gaussian - LoG)

Goal: Detect edges by identifying points in an image where intensity changes sharply, while reducing the effect of noise.

Key Ideas:

1. **Multi-scale Analysis:** Edges appear differently at different image resolutions.
2. **Zero-Crossings:** Edges are found where the second derivative (Laplacian) crosses zero.

Steps:

1. **Smooth the Image** using a **Gaussian filter**:

- Reduces noise.
- Uses:

$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

2. **Apply the Laplacian** (second derivative) to the smoothed image.

- This gives us areas where the intensity changes most rapidly.

3. **Detect Zero-Crossings** in the result.

- A zero-crossing is where pixel values change sign (e.g., from positive to negative), indicating an edge.

□ The combination of Gaussian and Laplacian is called **LoG** (Laplacian of Gaussian).

□ Instead of computing LoG directly, we can approximate it using **Difference of Gaussians (DoG)**, which is faster and close enough.

Pros:

- Isotropic (rotation-invariant).
- Simpler than some modern methods.

Cons :

- Sensitive to noise.
- Can detect false edges due to the second derivative's nature.

2. Canny Edge Detector

Goal: Find **precise**, **thin**, and **accurate** edges with minimal noise interference.

Key Design Criteria :

1. **Low error rate** (detect all real edges).
2. **Good localization** (edges are in the right place).
3. **Single edge response** (no double edges or thick lines).

Steps:

1. **Smooth the Image** using a **Gaussian filter** (like Marr-Hildreth).
2. **Compute the Gradient:**
 - Find **magnitude and direction** of edges using first derivatives:

$$M(x, y) = \sqrt{g_x^2 + g_y^2}$$

3. **Non-Maximum Suppression:**
 - Thins the edges by keeping only the pixels with the highest gradient magnitude in the direction of the edge.
4. **Hysteresis Thresholding:**
 - Uses **two thresholds**:
 - High threshold T_H : Strong edge.
 - Low threshold T_L : Possible edge.
 - Keeps connected weak edges only if they are linked to strong edges (reduces noise).

Pros :

- Excellent edge localization.
- Clean, thin edges with fewer false positives.
- Robust against noise (thanks to smoothing and double-thresholding).

Cons:

- More complex.
- Slower than simpler detectors (but still efficient).

LINKING EDGE POINTS

Edge detection often yields **broken or incomplete edges** due to:

- Noise
- Non-uniform illumination
- Discontinuities in intensity

Edge linking connects fragmented edge pixels into **meaningful, continuous edges or boundaries**.

Two Main Edge Linking Approaches :

1. Local Processing
2. Global Processing

a. Local Processing

- Analyzes edge pixels in a **small neighborhood**, usually 3×3 .
- Links pixels with **similar properties** to form continuous edges.
- **Gradient Magnitude Similarity:**

$$|M(s, t) - M(x, y)| \leq E$$

Where M is the gradient magnitude, and E is the threshold.

- **Gradient Direction Similarity:**

$$|a(s, t) - a(x, y)| \leq A$$

Where a is the gradient direction, and A is the angle threshold.

- A pixel is **linked** if both magnitude and direction thresholds are satisfied.

- ☐ Repeat the process for **every edge pixel**.
- ☐ Keep track of connected pixels, often by assigning a **unique intensity** to each group.
- ☐ **Computationally expensive** due to neighborhood checks around each edge point.

The preceding formulation is computationally expensive because all neighbors of every point have to be examined. A simplification particularly well suited for real time applications consists of the following steps:

1. Compute the gradient magnitude and angle arrays, $M(x,y)$ and $\alpha(x,y)$, of the input image, $f(x,y)$.
2. Form a binary image, $g(x,y)$, whose value at any point (x,y) is given by:

$$g(x,y) = \begin{cases} 1 & \text{if } M(x,y) > T_M \text{ AND } \alpha(x,y) = A \pm T_A \\ 0 & \text{otherwise} \end{cases}$$

where T_M is a threshold, A is a specified angle direction, and $\pm T_A$ defines a “band” of acceptable directions about A .

3. Scan the rows of g and fill (set to 1) all gaps (sets of 0's) in each row that do not exceed a specified length, L . Note that, by definition, a gap is bounded at both ends by one or more 1's. The rows are processed individually, with no “memory” kept between them.
4. To detect gaps in any other direction, θ , rotate g by this angle and apply the horizontal scanning procedure in Step 3. Rotate the result back by $-\theta$.

b. Global Processing with Hough Transform

- Suitable for unstructured environments with no prior knowledge of object location.
- Finds groups of edge pixels that form specified shapes (e.g., straight lines, circles).
- Links edge points based on global geometric relationships, not local neighborhoods.

Original Idea:

- Proposed by Hough (1962).
- Convert the line equation into a form where each image point corresponds to a curve in parameter space.
- Intersections in parameter space identify collinear points in image space.

Line Representations

1. Slope-Intercept Form:

- $y = ax + b$
- Problem: Vertical lines $\rightarrow$ infinite slope a .

2. Normal Form (used in HT):

$$x \cos \theta + y \sin \theta = r$$

- r : perpendicular distance from origin
- θ : angle of normal to the line
- Range:
 - $-90^\circ \leq \theta \leq 90^\circ$
 - $-D \leq r \leq D$, where D is max image diagonal

Hough Transform Algorithm

1. Edge Detection:

- Get binary edge map (e.g., using Canny, Sobel, etc.)

2. Initialize Accumulator:

- Create 2D array $A(i, j)$ for (r, θ) space
- Each cell corresponds to a potential line

3. Voting:

For each edge point (x_k, y_k) :

- For each allowed angle θ_q :
 - Compute $r_p = x_k \cos \theta_q + y_k \sin \theta_q$
 - Round r_p to the nearest valid bin
 - Increment cell $A(p, q)$

4. Line Detection:

- Peaks in $A(i, j)$ correspond to lines with many collinear points

5. Gap Bridging (optional):

- Check continuity of pixels for each line (e.g., fill gaps $<$ threshold)

Extension: Detecting Other Shapes

Generalized Form:

- HT can detect any shape defined by:

$$g(\mathbf{v}, \mathbf{c}) = 0$$

- $\mathbf{v}$: coordinates
- $\mathbf{c}$: coefficients (parameters)

Example: Circle Detection

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2$$

- Parameter space: 3D (c_1, c_2, c_3)
- Accumulator: 3D cube, $A(i, j, k)$

Advantages of Hough Transform

- Robust to noise and missing data
- Effective in global shape detection
- Extendable to arbitrary shapes

Limitations

- High computational cost for complex shapes (many parameters)
- Accuracy depends on resolution of parameter space

3.3 Thresholding

- Thresholding is a fundamental image segmentation technique.
- It's valued for its simplicity, intuitive nature, and computational speed.
- Partitions an image based on intensity values.

Global Thresholding

- A single threshold value (T) is used across the entire image.
- Binary segmentation rule:

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases}$$

- Effective when histogram has two distinct modes.

Variable Thresholding

- Threshold varies across the image.
- Types:
 - Local/Regional: T depends on local neighborhood stats.
 - Adaptive/Dynamic: T depends on both spatial position and neighborhood.

Multiple Thresholding

- Used when more than two histogram modes are present.
- Segments image into three or more classes using two or more thresholds.
- Difficult to use with noisy or ambiguous histograms.
- Example:

$$g(x, y) = \begin{cases} a & \text{if } f(x, y) > T_2 \\ b & \text{if } T_1 < f(x, y) \leq T_2 \\ c & \text{if } f(x, y) \leq T_1 \end{cases}$$

Factors Affecting Thresholding

- Success depends on valley depth and separation in histogram.
- Key influencing factors:
 1. Separation between peaks
 2. Noise level
 3. Relative size of objects and background

4. Illumination uniformity
5. Reflectance uniformity

Role of Noise

- Noise broadens histogram modes, making thresholding harder.
- Small noise (e.g., $\sigma = 10$): modes are still separable.
- Large noise (e.g., $\sigma = 50$): histogram modes may merge $\rightarrow$ thresholding fails.

Illumination & Reflectance Effects

- Nonuniform illumination alters histogram, can destroy separation between modes.
- Solutions:
 1. Shading correction via inverse intensity function.
 2. Top-hat transformation to remove background trends.
 3. Variable thresholding to “work around” nonuniformity.

BASIC GLOBAL THRESHOLDING ALGORITHM

When the intensity distributions of objects and background pixels are sufficiently distinct, it is possible to use a single (*global*) threshold applicable over the entire image. In most applications, there is usually enough variability between images that, even if global thresholding is a suitable approach, an algorithm capable of estimating the threshold value for each image is required. The following iterative algorithm can be used for this purpose:

1. Select an initial estimate for the global threshold, T .
2. Segment the image using T in Eq. (10-46). This will produce two groups of pixels: G_1 , consisting of pixels with intensity values $> T$; and G_2 , consisting of pixels with values $\leq T$.
3. Compute the average (mean) intensity values m_1 and m_2 for the pixels in G_1 and G_2 , respectively.
4. Compute a new threshold value midway between m_1 and m_2 :
$$T = \frac{1}{2}(m_1 + m_2)$$
5. Repeat Steps 2 through 4 until the difference between values of T in successive iterations is smaller than a predefined value, ΔT .

The algorithm is stated here in terms of successively thresholding the input image and calculating the means at each step, because it is more intuitive to introduce it in this manner. However, it is possible to develop an equivalent (and more efficient) procedure by expressing all computations in the terms of the image histogram, which has to be computed only once.

- Histogram-based implementations are more efficient.
- Guaranteed to converge if T is initially between min and max intensity values.

3.4 SEGMENTATION BY REGION GROWING AND BY REGION SPLITTING AND MERGING

- The goal of segmentation is to partition an image into meaningful regions.
- Previous methods (Sections 10.2 & 10.3) used discontinuities or thresholding based on pixel property distributions.
- This section focuses on methods that directly identify regions rather than edges or intensity jumps.

REGION GROWING

As its name implies, *region growing* is a procedure that groups pixels or subregions into larger regions based on predefined criteria for growth. The basic approach is to start with a set of “seed” points, and from these grow regions by appending to each seed those neighboring pixels that have predefined properties similar to the seed.

Definition

- A procedure that groups pixels/subregions into larger regions based on **predefined similarity criteria**.

Basic Concept

- Start from **seed points**.
- Grow regions by **appending neighboring pixels** that satisfy the similarity criteria.

Seed Selection

- **A priori information** can guide seed selection.
- Without prior info:
 - Compute descriptors (e.g., intensity, texture) at all pixels.
 - Choose seed points near **cluster centroids** in feature space.

Similarity Criteria

- Depend on the **problem type** and **image data** (e.g., intensity, color, texture).
- **Color** is critical in satellite imagery; monochrome images rely on intensity and spatial descriptors.

Importance of Connectivity

- Without enforcing **connectivity**, results may be misleading.
- Connectivity ensures pixels grouped together are **spatially coherent**.

Stopping Rule

- Region growth stops when **no more neighboring pixels** satisfy inclusion criteria.
- Criteria can include:
 - Pixel-to-seed similarity (e.g., intensity/color difference).

- Pixel-to-region similarity (e.g., compare with average intensity).
- Region size and shape (if model-based expectations are available).

A basic region-growing algorithm based on 8-connectivity may be stated as follows.

1. Find all connected components in $S(x, y)$ and reduce each connected component to one pixel; label all such pixels found as 1. All other pixels in S are labeled 0.
2. Form an image f_Q such that, at each point (x, y) , $f_Q(x, y) = 1$ if the input image satisfies a given predicate, Q , at those coordinates, and $f_Q(x, y) = 0$ otherwise.
3. Let g be an image formed by appending to each seed point in S all the 1-valued points in f_Q that are 8-connected to that seed point.
4. Label each connected component in g with a different region label (e.g., integers or letters). This is the segmented image obtained by region growing.

- **Region growing is data-driven and can be guided with domain knowledge.**
- **Success depends on proper seed selection, predicate design, and connectivity enforcement.**
- **Flexible and intuitive but sensitive to noise and seed placement.**

REGION SPLITTING AND MERGING

- An alternative to region growing.
- Begins with the entire image region and subdivides (splits) it.
- Later merges adjacent regions to refine segmentation.
- Goal: Ensure all regions satisfy a predicate Q , related to segmentation criteria (e.g., intensity uniformity, texture).

Let R represent the entire image region and select a predicate Q . One approach for segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i , $Q(R_i) = \text{TRUE}$. We start with the entire region, R . If $Q(R) = \text{FALSE}$, we divide the image into quadrants. If Q is FALSE for any quadrant, we subdivide that quadrant into sub-quadrants, and so on. This splitting technique has a convenient representation in the form of so-called *quadtrees*; that is, trees in which each node has exactly four descendants, as Fig. 10.47 shows (the images corresponding to the nodes of a quadtree sometimes are called *quadregions* or *quadimages*). Note that the root of the tree corresponds to the entire image, and that each node corresponds to the subdivision of a node into four descendant nodes. In this case, only R_4 was subdivided further.

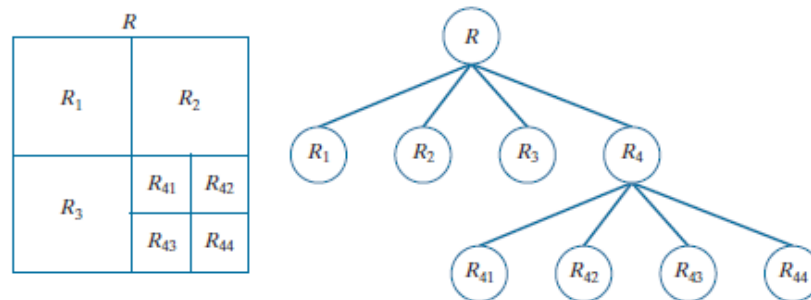
If only splitting is used, the final partition normally contains adjacent regions with identical properties. This drawback can be remedied by allowing *merging* as well as splitting. Satisfying the constraints of segmentation outlined in Section 10.1 requires merging only adjacent regions whose combined pixels satisfy the predicate Q . That is, two adjacent regions R_j and R_k are merged only if $Q(R_j \cup R_k) = \text{TRUE}$.

The preceding discussion can be summarized by the following procedure in which, at any step, we

1. Split into four disjoint quadrants any region R_i for which $Q(R_i) = \text{FALSE}$.
2. When no further splitting is possible, merge any adjacent regions R_j and R_k for which $Q(R_j \cup R_k) = \text{TRUE}$.

See Section 2.5
regarding region
adjacency.

a b
FIGURE 10.47
(a) Partitioned
image.
(b) Corresponding
quadtree.
 R represents
the entire image
region.



3. Stop when no further merging is possible.

Numerous variations of this basic theme are possible. For example, a significant simplification results if in Step 2 we allow merging of any two adjacent regions R_j and R_k if each one satisfies the predicate individually. This results in a much simpler (and faster) algorithm, because testing of the predicate is limited to individual quadregions. As the following example shows, this simplification is still capable of yielding good segmentation results.