distributing the samples, greater sample concentration should be used in gray-level transition boundaries, such as the boundary between the face and the background in this example.

The necessity of having to identify boundaries, even if only roughly, is a definite drawback of the nonuniform sampling approach. This method also is not practical for images containing relatively small uniform regions. For instance, nonuniform sampling would be difficult to justify for an image of a dense crowd of people.

When the number of gray levels must be kept small, the use of unequally spaced levels in the quantization process usually is desirable. A method similar to the nonuniform sampling technique discussed earlier may be used for the distribution of gray levels in an image. However, as the eye is relatively poor at estimating shades of gray near abrupt level changes, the approach in this case is to use few gray levels in the neighborhood of boundaries. The remaining levels can then be used in regions where gray-level variations are smooth, thus avoiding or reducing the false contours that often appear in these regions if they are too coarsely quantized.

This method is subject to the preceding observations about boundary detection and detail content. An alternative technique that is particularly attractive for distributing gray levels consists of computing the frequency of occurrence of all allowed levels. If gray levels in a certain range occur frequently, while others occur rarely, the quantization levels are finely spaced in this range and coarsely spaced outside of it. This method is sometimes called *tapered quantization*. We discuss these topics further in Chapter 6.

2.4 SOME BASIC RELATIONSHIPS BETWEEN PIXELS

In this section we consider several primitive, yet important relationships between pixels in a digital image. As mentioned before, an image is denoted by $f(x, y)$. When referring to a particular pixel, we use lowercase letters, such as p and q. A subset of pixels of $f(x, y)$ is denoted by S.

2.4.1 Neighbors of a Pixel

A pixel p at coordinates (x, y) has four *horizontal* and *vertical* neighbors whose coordinates are given by

$$(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1).$$

This set of pixels, called the 4-*neighbors* of p, is denoted by $N_4(p)$. Each pixel is a unit distance from (x, y), and some of the neighbors of p lie outside the digital image if (x, y) is on the border of the image.

The four *diagonal* neighbors of p have coordinates

$$(x + 1, y + 1), (x + 1, y - 1), (x - 1, y + 1), (x - 1, y - 1)$$

and are denoted by $N_D(p)$. These points, together with the 4-neighbors, are called the 8-*neighbors* of p, denoted by $N_8(p)$. As before, some of the points in $N_D(p)$ and $N_8(p)$ fall outside the image if (x, y) is on the border of the image.

2.4.2 Connectivity

Connectivity between pixels is an important concept used in establishing boundaries of objects and components of regions in an image. To establish whether two pixels are connected, it must be determined if they are adjacent in some sense (say, if they are 4-neighbors) and if their gray levels satisfy a specified criterion of similarity (say, if they are equal). For instance, in a binary image with values 0 and 1, two pixels may be 4-neighbors, but they are not said to be connected unless they have the same value.

Let V be the set of gray-level values used to define connectivity; for example, in a binary image, $V = \{1\}$ for the connectivity of pixels with value 1. In a gray-scale image, for the connectivity of pixels with a range of intensity values of, say, 32 to 64, it follows that $V = \{32, 33, \ldots, 63, 64\}$. We consider three types of connectivity:

(a) 4-*connectivity*. Two pixels p and q with values from V are 4-connected if q is in the set $N_4(p)$.

(b) 8-*connectivity*. Two pixels p and q with values from V are 8-connected if q is in the set $N_8(p)$.

(c) m-*connectivity* (mixed connectivity). Two pixels p and q with values from V are m-connected if

(i) q is in $N_4(p)$, *or*

(ii) q is in $N_D(p)$ *and* the set $N_4(p) \cap N_4(q)$ is empty. (This is the set of pixels that are 4-neighbors of both p and q and whose values are from V.)

Mixed connectivity is a modification of 8-connectivity and is introduced to eliminate the multiple path connections that often arise when 8-connectivity is used. For example, consider the pixel arrangement shown in Fig. 2.13(a). For $V = \{1\}$, the paths between 8-neighbors of the center pixel are shown by dashed lines in Fig. 2.13(b). Note the ambiguity in path connections that results from allowing 8-connectivity. This ambiguity is removed by using m-connectivity, as shown in Fig. 2.13(c).

A pixel p is *adjacent* to a pixel q if they are connected. We can define 4-, 8-, or m-adjacency depending on the type of connectivity specified. Two image subsets S_1 and S_2 are adjacent if some pixel in S_1 is adjacent to some pixel in S_2.

A *path* from pixel p with coordinates (x, y) to pixel q with coordinates (s, t) is a sequence of distinct pixels with coordinates

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$$

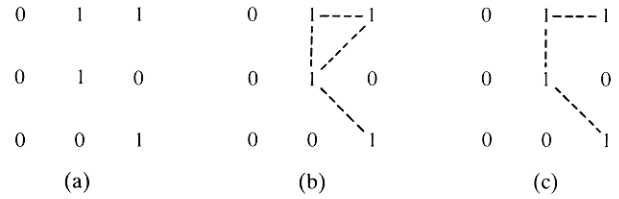

Figure 2.13 *(a) Arrangement of pixels; (b) 8-neighbors of the center pixel; (c) m-neighbors of the same pixel. The dashed lines are paths between that pixel and its neighbors.*

where $(x_0, y_0) = (x, y)$ and $(x_n, y_n) = (s, t)$, (x_i, y_i) is adjacent to (x_{i-1}, y_{i-1}), $1 \leq i \leq n$, and n is the *length* of the path. We can define 4-, 8-, or m-paths depending on the type of adjacency specified.

If p and q are pixels of an image subset S, then p is *connected* to q in S if there is a path from p to q consisting entirely of pixels in S. For any pixel p in S, the set of pixels in S that are connected to p is called a *connected component* of S. Hence any two pixels of a connected component are connected to each other, and distinct connected components are disjoint.

The ability to assign different labels to various disjoint, connected components of an image is of fundamental importance in automated image analysis. In the following section we develop a simple sequential connected component labeling procedure that operates on two rows of a binary image at a time. We develop a different approach based on morphology in Section 8.4.

2.4.3 Labeling of Connected Components

Imagine scanning an image pixel by pixel, from left to right and from top to bottom and assume for the moment that we are interested in 4-connected components. Let p denote the pixel at any step in the scanning process and let r and t denote the upper and left-hand neighbors of p, respectively. The nature of the scanning sequence ensures that when we get to p, points r and t have already been encountered (and labeled if they were 1's).

With the preceding concepts established, let us consider the following procedure. If the value of p is 0, simply move on to the next scanning position. If the value of p is 1, examine r and t. If they are both 0, assign a new label to p (as far as we know, based on the current information, this is the first time that this connected component has been encountered). If only one of the two neighbors is 1, assign its label to p. If they are both 1 and have the same label, assign that label to p. If they are both 1 and have different labels, assign one of the labels to p and make a note that the two labels are equivalent (that is, points r and t are connected through p). At the end of the scan, all points with value 1 have been labeled, but some of these labels may be equivalent. All we need to do now is sort all pairs of equivalent labels into equivalence classes

(see Section 2.4.4), assign a different label to each class, and then do a second pass through the image, replacing each label by the label assigned to its equivalence class.

To label 8-connected components we proceed in the same way, but the two upper diagonal neighbors of p, denoted by q and s, also have to be examined. The nature of the scanning sequence ensures that these neighbors have already been processed by the time the procedure gets to p. If p is 0, move on to the next scanning position. If p is 1 and all four neighbors are 0, assign a new label to p. If only one of the neighbors is 1, assign its label to p. If two or more neighbors are 1, assign one of the labels to p and make a note of the appropriate equivalences. After completing the scan of the image sort the equivalent label pairs into equivalence classes, assign a unique label to each class, and do a second scan through the image, replacing each label by the label assigned to its equivalence class.

2.4.4 Relations, Equivalence, and Transitive Closure

The labeling algorithm discussed in the previous section suggests the usefulness of formal tools for handling relationships and equivalences in pixel processing. Let us consider briefly some important concepts that are the bases of such relationships and equivalences.

A *binary relation*[†] R on a set A is a set of pairs of elements from A. If the pair (a, b) is in R, the notation often used is aRb which, in words, is interpreted to mean "a is related to b." Take for example, the set of points $A = \{p_1, p_2, p_3, p_4\}$ arranged as

$$\begin{array}{ll} p_1 & p_2 \\ p_3 & \\ & p_4 \end{array}$$

and define the relation "4-connected." In this case, R is the set of pairs of points from A that are 4-connected; that is, $R = \{(p_1, p_2), (p_2, p_1), (p_1, p_3), (p_3, p_1)\}$. Thus p_1 is related to p_2, and p_1 is related to p_3, and vice versa, but p_4 is not related to any other point under the relation "4-connected".

A binary relation R over set A is said to be

(a) *reflexive* if for each a in A, aRa;

(b) *symmetric* if for each a and b in A, aRb implies bRa; and

(c) *transitive* if for a, b, and c in A, aRb and bRc implies aRc.

A relation satisfying these three properties is called an *equivalence relation.*

[†] In this context, the word *binary* refers to "two," and has nothing to do with binary images.

An important property of equivalence relations is that, if R is an equivalence relation on a set A, then A can be divided into k disjoint subsets, called *equivalence classes*, for some k between 1 and ∞, inclusive, such that aRb if and only if a and b are in the same subset.

Expressing a relation in terms of a binary matrix is useful. For example, letting $R = \{(a, a), (a, b), (b, d), (d, b), (c, e)\}$ yields the matrix

$$\mathbf{B} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

if a 1 is inserted in the locations corresponding to elements that are related and 0's are inserted elsewhere. If the relation in question were reflexive, all the main diagonal terms would be 1; if R were symmetric, $\mathbf{B}$ would be a symmetric matrix.

As indicated above, transitivity implies if aRb and bRc then aRc. In the example just given, a is related to b and b is related to d because (a, b) and (b, d) are in R. However, we note that (a, d) is not in the set R. The set containing these "implied" relations is called the *transitive closure* of R and is denoted by R^+. Here, $R^+ = \{(a, a), (a, b), (a, d), (b, b), (b, d), (d, b), (d, d), (c, e)\}$. The fact that the set includes the pairs (b, b) and (d, d) follows from the definition of transitivity (that is, bRd and dRb, so bRb; and dRb and bRd, so dRd). Expressed in matrix form,

$$\mathbf{B}^+ = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

where the 1-valued elements determine the members of the transitive closure R^+. A straightforward procedure for computing matrix $\mathbf{B}^+$ from a given matrix $\mathbf{B}$ is as follows.

Let $\mathbf{B}$ be an $n \times n$ binary matrix representing a relation R over an alphabet

A of n symbols and compute the matrix

$$\mathbf{B}^+ = \mathbf{B} + \mathbf{BB} + \mathbf{BBB} + \cdots + (\mathbf{B})^n \tag{2.4-1}$$

where $(\mathbf{B})^n = \mathbf{BBB} \ldots \mathbf{B}$ (n times). The 1-valued elements of the matrix $\mathbf{B}^+$ represent the transitive closure, R^+, of relation R (Gries [1971]). The matrix operations are carried out in the usual manner, except that all multiplications are replaced by logical ANDs and all additions (including those shown in Eq. 2.4-1) are replaced by logical ORs. The order of operations in Eq. (2.4-1) is **B**, **B**(**B**), **B**(**BB**), **B**(**BBB**), . . . so that, at each step, we simply multiply the result up to that point by **B**. We leave it as an exercise to show that Eq. (2.4-1) gives the same result for $\mathbf{B}^+$ as in the example above.

Implementation of Eq. (2.4-1) requires on the order of n^3 AND and OR operations. Warshall [1962] developed a more efficient algorithm that requires only OR operations involving the elements of **B** that have value 1:

Step 1. Set $j = 1$.

Step 2. For $i = 1, 2, \ldots, n$, if $b(i, j) = 1$, then, for $k = 1, 2, \ldots, n$, set $b(i, k) = b(i, k) + b(j, k)$.

Step 3. Increment j by 1.

Step 4. If $j \leq n$, go to *Step* 2; otherwise go to *Step* 5.

Step 5. Stop. The result is $\mathbf{B}^+$ in place of **B**.

It is instructive to verify that this procedure yields the same result as Eq. (2.4-1) for the example earlier in this section.

In practice (as in the algorithm presented at the end of Section 2.4.3) the assumption typically is that the relations are equivalence relations, in which case matrix **B** is symmetric and all the main diagonal terms are set to 1 prior to use of either Eq. (2.4-1) or Warshall's algorithm to compute the transitive closure. The equivalence classes of the various symbols in the alphabet leading to matrix $\mathbf{B}^+$ can then be determined by scanning this matrix from left to right and from top to bottom. When a 1 is encountered in, say, row i and column j, we set the symbol associated with the jth column equal to the symbol associated with the ith row (they are equivalent), zero out the jth column, and continue the scan of matrix $\mathbf{B}^+$.

2.4.5 Distance Measures

For pixels p, q, and z, with coordinates (x, y), (s, t), and (u, v) respectively, D is a *distance function* or *metric* if

(a) $D(p, q) \geq 0$ ($D(p, q) = 0$ iff $p = q$),

(b) $D(p, q) = D(q, p)$, and

(c) $D(p, z) \leq D(p, q) + D(q, z)$.

The *Euclidean distance* between p and q is defined as

$$D_e(p, q) = [(x - s)^2 + (y - t)^2]^{1/2} \tag{2.4-2}$$

For this distance measure, the pixels having a distance less than or equal to some value r from (x, y) are the points contained in a disk of radius r centered at (x, y).

The D_4 *distance* (also called *city-block distance*) between p and q is defined as

$$D_4(p, q) = |x - s| + |y - t| \tag{2.4-3}$$

In this case the pixels having a D_4 distance from (x, y) less than or equal to some value r form a diamond centered at (x, y). For example, the pixels with D_4 distance ≤ 2 from (x, y) (the center point) form the following contours of constant distance:

$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array}$$

The pixels with $D_4 = 1$ are the 4-neighbors of (x, y).

The D_8 *distance* (also called *chessboard distance*) between p and q is defined as

$$D_8(p, q) = \max(|x - s|, |y - t|). \tag{2.4-4}$$

In this case the pixels with D_8 distance from (x, y) less than or equal to some value r form a square centered at (x, y). For example, the pixels with D_8 distance ≤ 2 from (x, y) (the center point) form the following contours of constant distance:

$$\begin{array}{ccccc} 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{array}$$

The pixels with $D_8 = 1$ are the 8-neighbors of (x, y).

The D_4 distance between two points p and q is equal to the length of the shortest 4-path between these two points. The same applies to the D_8 distance. In fact, we can consider both the D_4 and D_8 distances between p and q regardless of whether a connected path exists between them because the definitions of these distances involve only the coordinates of these points. For m-connectivity,

however, the value of the distance (length of the path) between two pixels depends on the values of the pixels along the path and those of their neighbors. For instance, consider the following arrangement of pixels and assume that p, p_2, and p_4 have a value of 1 and that p_1 and p_3 can have a value of 0 or 1:

$$\begin{matrix} & p_3 & p_4 \\ p_1 & p_2 & \\ p & & \end{matrix}$$

If only connectivity of pixels valued 1 is allowed, and p_1 and p_3 are 0, the m distance between p and p_4 is 2. If either p_1 or p_3 is 1, the distance is 3. If both p_1 and p_3 are 1, the distance is 4.

2.4.6 Arithmetic/Logic Operations

Arithmetic and logic operations between pixels are used extensively in most branches of image processing. The arithmetic operations between two pixels p and q are denoted as follows:

$$\begin{aligned} \text{Addition:} \quad & p + q \\ \text{Subtraction:} \quad & p - q \\ \text{Multiplication:} \quad & p*q \text{ (also, } pq \text{ and } p \times q) \\ \text{Division:} \quad & p \div q \end{aligned}$$

Arithmetic operations on entire images are carried out pixel by pixel. The principal use of image addition is for image averaging to reduce noise. Image subtraction is a basic tool in medical imaging, where it is used to remove static background information. One of the principal uses of image multiplication (or division) is to correct gray-level shading resulting from nonuniformities in illumination or in the sensor used to acquire the image. Arithmetic operations involve only one spatial pixel location at a time, so they can be done "in place," in the sense that the result of performing an arithmetic operation at location (x, y) can be stored in that location in one of the existing images, as that location will not be visited again.

The principal logic operations used in image processing are AND, OR, and COMPLEMENT, denoted as follows:

$$\begin{aligned} \text{AND:} \quad & p\text{AND}q \text{ (also, } p \cdot q) \\ \text{OR:} \quad & p\text{OR}q \text{ (also, } p + q) \\ \text{COMPLEMENT:} \quad & \text{NOT}q \text{ (also, } \overline{q}) \end{aligned}$$

These operations are *functionally complete* in the sense that they can be combined to form any other logic operation. Logic operations apply only to binary

images, whereas arithmetic operations apply to multivalued pixels. Logic operations are basic tools in binary image processing, where they are used for tasks such as masking, feature detection, and shape analysis. Logic operations on entire images are performed pixel by pixel. Because the AND operation of two binary variables is 1 only when both variables are 1, the result at any location in a resulting AND image is 1 only if the corresponding pixels in the two input images are 1. As logic operations involve only one pixel location at a time, they can be done in place, as in the case of arithmetic operations. Figure 2.14 shows various examples of logic operations, where black indicates 1 and white indicates 0. The XOR (exclusive OR) operation yields a 1 when one or the other pixel (but *not* both) is 1, and it yields a 0 otherwise. This operation is unlike the OR operation, which is 1 when one or the other pixel is 1, or both pixels are 1.

In addition to pixel-by-pixel processing on entire images, arithmetic and logic operations are used in neighborhood-oriented operations. Neighborhood processing typically is formulated in the context of so-called *mask* operations (the terms *template*, *window*, and *filter* also are often used to denote a mask). The idea behind mask operations is to let the value assigned to a pixel be a function of its gray level and the gray level of its neighbors. For instance, consider the subimage area shown in Fig. 2.15(a), and suppose that we want to replace the value of z_5 with the average value of the pixels in a 3×3 region centered at the pixel with value z_5. To do so entails performing an arithmetic operation of the form

$$z = \frac{1}{9}(z_1 + z_2 + \cdots + z_9) = \frac{1}{9}\sum_{i=1}^{9} z_i$$

and assigning to z_5 the value of z.

With reference to the mask shown in Fig. 2.15(b), the same operation can be obtained in more general terms by centering the mask at z_5 multiplying each pixel under the mask by the corresponding coefficient, and adding the results; that is,

$$z = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i \tag{2.4-5}$$

If we let $w_i = 1/9$, $i = 1, 2, \ldots, 9$, this operation yields the same result as the averaging procedure just discussed.

Equation (2.4-5) is used widely in image processing. Proper selection of the coefficients and application of the mask at each pixel position in an image makes possible a variety of useful image operations, such as noise reduction, region thinning, and edge detection. However, applying a mask at each pixel location in an image is a computationally expensive task. For example, applying a 3×3 mask to a 512×512 image requires nine multiplications and eight

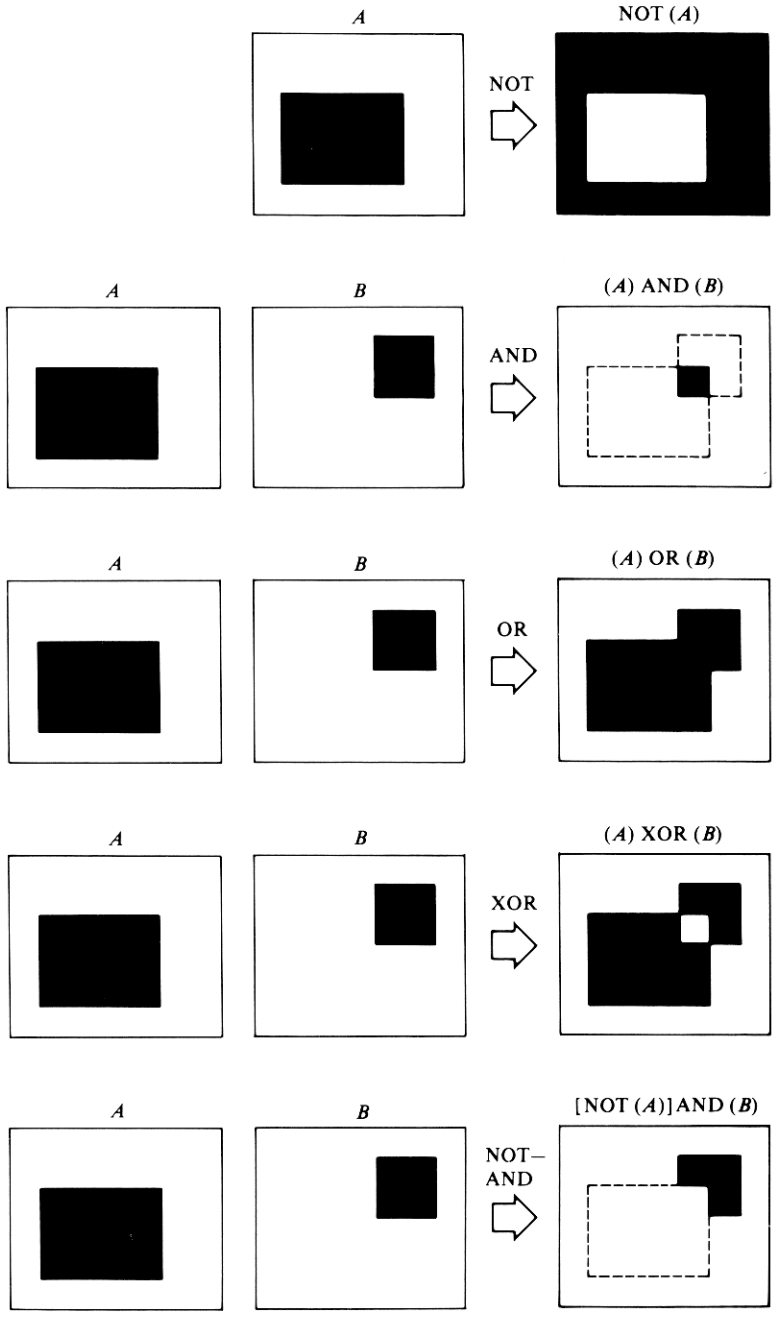

Figure 2.14 *Some examples of logic operations on binary images.*

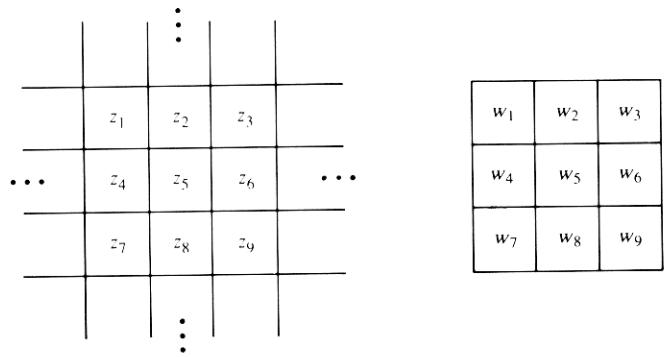

Figure 2.15 *(a) Subarea of an image showing pixel values; (b) a 3 × 3 mask with general coefficients.*

additions at each pixel location, for a total of 2,359,296 multiplications and 2,097,152 additions.

As indicated in Section 1.4.3, most modern image processors are equipped with an Arithmetic–Logic Unit (ALU), whose function is to perform arithmetic and logic operations in parallel, typically at video-frame rates. For U.S. standard video, an ALU can perform an arithmetic or logic operation between two 512 × 512 images in 1/30 sec. (This time interval is often called one *frame* or one *frame time*.) Given the importance of mask operations in image processing it is of interest to consider in some detail how to use an ALU to accelerate mask processing. For the purpose of illustration, we consider the 3 × 3 mask shown in Fig. 2.15(b) and the implementation expressed by Eq. (2.4-5). However, the method is easily extendible to an $n \times m$ mask and other arithmetic or logic operations.

The algorithm given here requires two image frame buffers with the capability to scroll and pan by one pixel location (see Section 1.4.2). Let frame buffer A contain the image to which the mask is to be applied. At the end of the process, frame buffer B will contain the result of the operation. Recall that ALU operations are performed on all pixels in one frame time, whereas all buffer shifts are performed virtually instantaneously. We assume that all shifts are by one pixel. Letting $B = A$ initially, and using a dash to indicate no operation, we follow the procedure shown in Table 2.3. The last two shifts are required because, at the end of the last operation on B, the image is in a position equivalent to having the mask with its w_7 coefficient over the z_5 position. The two shifts correct this misalignment.

The key to understanding the procedure in Table 2.3 is to examine what

Table 2.3 ALU Operations

Operations on A	*Operations on B*
—	Multiply by w_5
Shift right	—
—	Add w_4*A
Shift down	—
—	Add w_1*A
Shift left	—
—	Add w_2*A
Shift left	—
—	Add w_3*A
Shift up	—
—	Add w_6*A
Shift up	—
—	Add w_9*A
Shift right	—
—	Add w_8*A
Shift right	—
—	Add w_7*A
Shift left	—
Shift down	—

happens in a single pixel of B by considering how a mask would have to be shifted in order to produce the result of Eq. (2.4-5) in that location. The first operation on B produces w_5 multiplied by the pixel value at that location. Since that value is z_5, we have $w_5 z_5$ after this operation. The first shift to the right brings the neighbor with value z_4 (see Fig. 2.15a) over that location. The next operation multiplies z_4 by w_4 and adds the result to the location of the first step. So at this point the result is $w_4 z_4 + w_5 z_5$ at the location in question. The next shift on A and ALU operation on B produce $w_1 z_1 + w_4 z_4 + w_5 z_5$ at that location, and so on. The operations are done in parallel for all locations in B, so this procedure takes place simultaneously at the other locations in that frame buffer. In most ALUs, the operation of multiplying an image by a constant (say, w_i*A) followed by an ADD is done in one frame time. Thus the ALU implementation of Eq. (2.4-5) for an entire image takes on the order of nine frame times (9/30 sec). For an $n \times m$ mask it would take on the order of nm frame times.

2.5 IMAGING GEOMETRY

In the following discussion we present several important transformations used in imaging, derive a camera model, and treat the stereo imaging problem in some detail.