CHAPTER 2

IMAGE PROCESSING BASICS

WHAT WILL WE LEARN?

- How is a digital image represented and stored in memory?
- What are the main types of digital image representation?
- What are the most popular image file formats?
- What are the most common types of image processing operations and how do they affect pixel values?

2.1 DIGITAL IMAGE REPRESENTATION

A digital image—whether it was obtained as a result of sampling and quantization of an analog image or created already in digital form—can be represented as a two-dimensional (2D) matrix of real numbers. In this book, we adopt the convention $f(x, y)$ to refer to monochrome images of size $M \times N$, where x denotes the row number (from 0 to $M-1$) and y represents the column number (between 0 and $N-1$) (Figure 2.1):

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{2.1}$$

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

FIGURE 2.1 A monochrome image and the convention used to represent rows (x) and columns (y) adopted in this book.

The value of the two-dimensional function $f(x, y)$ at any given pixel of coordinates (x_0, y_0), denoted by $f(x_0, y_0)$, is called the *intensity* or *gray level* of the image at that pixel. The maximum and minimum values that a pixel intensity can assume will vary depending on the data type and convention used. Common ranges are as follows: 0.0 (black) to 1.0 (white) for `double` data type and 0 (black) to 255 (white) for `uint8` (unsigned integer, 8 bits) representation.

The convention expressed by equation (2.1) and Figure 2.1 is consistent with programming languages that use 0-based array notation (e.g., Java, C, C++) and several other textbooks, but *not* with MATLAB and its Image Processing Toolbox (IPT), which use 1-based array notation. Whenever the situation calls for explicit disambiguation between the two conflicting conventions, we shall use the notation `f(p,q)` to refer to the MATLAB representation of $f(x, y)$ (where `p` denotes row and `q` denotes column):

$$\texttt{f(p,q)} = \begin{bmatrix} \texttt{f(1,1)} & \texttt{f(1,2)} & \cdots & \texttt{f(1,N)} \\ \texttt{f(2,1)} & \texttt{f(2,2)} & \cdots & \texttt{f(2,N)} \\ \vdots & \vdots & & \vdots \\ \texttt{f(M,1)} & \texttt{f(M,2)} & \cdots & \texttt{f(M,N)} \end{bmatrix} \tag{2.2}$$

Monochrome images are essentially 2D *matrices* (or *arrays*). Since MATLAB treats matrices as a built-in data type, it is easier to manipulate them without resorting

to common programming language constructs (such as double `for` loops to access each individual element within the array), as we shall see in Chapter 3.

Images are represented in digital format in a variety of ways. At the most basic level, there are two different ways of encoding the contents of a 2D image in digital format: *raster* (also known as *bitmap*) and *vector*. Bitmap representations use one or more two-dimensional arrays of pixels, whereas vector representations use a series of drawing commands to represent an image. Each encoding method has its pros and cons: the greatest advantages of bitmap graphics are their quality and display speed; their main disadvantages include larger memory storage requirements and size dependence (e.g., enlarging a bitmap image may lead to noticeable artifacts). Vector representations require less memory and allow resizing and geometric manipulations without introducing artifacts, but need to be rasterized for most presentation devices.

In either case, there is no such a thing as a perfect digital representation of an image. Artifacts due to finite resolution, color mapping, and many others will always be present. The key to selecting an adequate representation is to find a suitable compromise between size (in bytes), subjective quality, and interoperability of the adopted format or standard. For the remaining of this book, we shall focus exclusively on bitmap images.

2.1.1 Binary (1-Bit) Images

Binary images are encoded as a 2D array, typically using 1 bit per pixel, where a 0 usually means "black" and a 1 means "white" (although there is no universal agreement on that). The main advantage of this representation—usually suitable for images containing simple graphics, text, or line art—is its small size. Figure 2.2 shows a binary image (the result of an edge detection algorithm) and a 6×6 detailed region, where pixels with a value of 1 correspond to edges and pixels with a value of 0 correspond to the background.

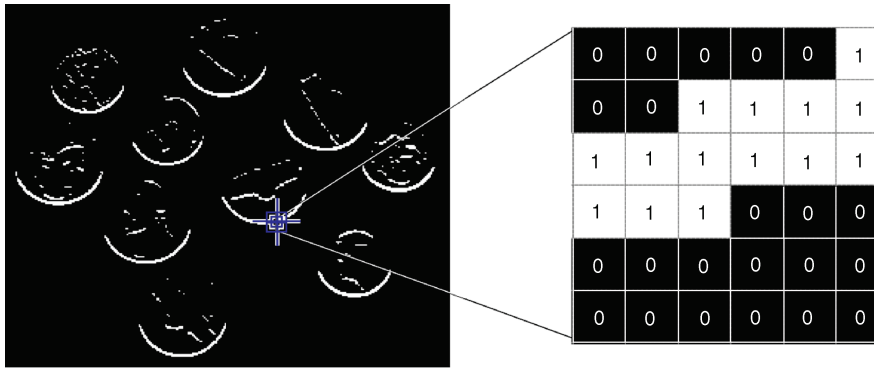

FIGURE 2.2 A binary image and the pixel values in a 6×6 neighborhood. Original image: courtesy of MathWorks.

In MATLAB

Binary images are represented in MATLAB using a *logical* array of 0's and 1's. Although we could also use an array of `uint8` and restrict the array values to be only 0 or 1, that would not, technically, be considered a binary image in MATLAB [GWE04]. Conversion from numerical to logical arrays can be accomplished using function `logical`.

Logical arrays can also be created and processed using relational and logical operators (see Chapter 6).

2.1.2 Gray-Level (8-Bit) Images

Gray-level (also referred to as *monochrome*) images are also encoded as a 2D array of pixels, usually with 8 bits per pixel, where a pixel value of 0 corresponds to "black," a pixel value of 255 means "white," and intermediate values indicate varying shades of gray. The total number of gray levels is larger than the human visual system requirements (which, in most cases, cannot appreciate any improvements beyond 64 gray levels), making this format a good compromise between subjective visual quality and relatively compact representation and storage.

Figure 2.3 shows a grayscale image and a 6×6 detailed region, where brighter pixels correspond to larger values.

In MATLAB

Intensity images can be represented in MATLAB using different data types (or *classes*). For monochrome images with elements of integer classes `uint8` and `uint16`, each pixel has a value in the [0, 255] and the [0, 65,535] range, respectively. Monochrome images of class `double` have pixel values in the [0.0, 1.0] range.

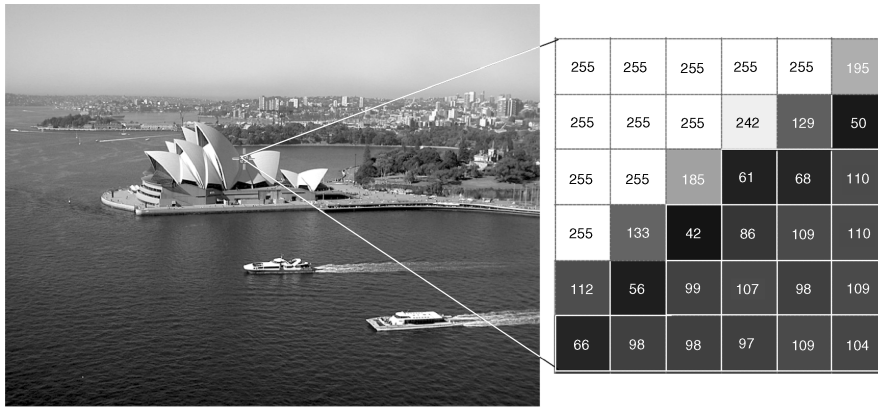

FIGURE 2.3 A grayscale image and the pixel values in a 6×6 neighborhood.

FIGURE 2.4 Color image (a) and its R (b), G (c), and B (d) components.

2.1.3 Color Images

Representation of color images is more complex and varied. The two most common ways of storing color image contents are *RGB* representation—in which each pixel is usually represented by a 24-bit number containing the amount of its red (R), green (G), and blue (B) components—and *indexed* representation—where a 2D array contains indices to a color palette (or *lookup table* - (*LUT*)).

24-Bit (RGB) Color Images Color images can be represented using three 2D arrays of same size, one for each color channel: red (R), green (G), and blue (B) (Figure 2.4).[1] Each array element contains an 8-bit value, indicating the amount of red, green, or blue at that point in a [0, 255] scale. The combination of the three 8-bit values into a 24-bit number allows 2^{24} (16,777,216, usually referred to as 16 million or 16 M) color combinations. An alternative representation uses 32 bits per pixel and includes a fourth channel, called the *alpha channel*, that provides a measure of transparency for each pixel and is widely used in image editing effects.

[1]Color images can also be represented using alternative color models (or *color spaces*), as we shall see in Chapter 16.

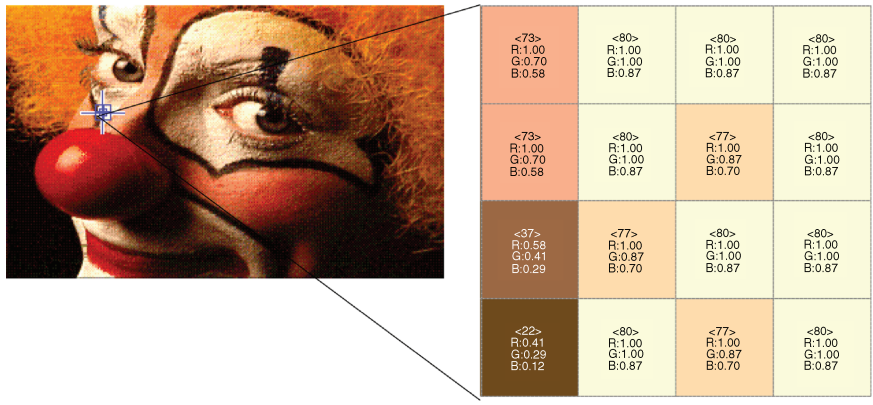

FIGURE 2.5 An indexed color image and the indices in a 4 × 4 neighborhood. Original image: courtesy of MathWorks.

Indexed Color Images A problem with 24-bit color representations is backward compatibility with older hardware that may not be able to display the 16 million colors simultaneously. A solution—devised before 24-bit color displays and video cards were widely available—consisted of an indexed representation, in which a 2D array of the same size as the image contains indices (pointers) to a *color palette* (or *color map*) of fixed maximum size (usually 256 colors). The color map is simply a list of colors used in that image.

Figure 2.5 shows an indexed color image and a 4 × 4 detailed region, where each pixel shows the index and the values of R, G, and B at the color palette entry that the index points to.

2.1.4 Compression

Since raw image representations usually require a large amount of storage space (and proportionally long transmission times in the case of file uploads/downloads), most image file formats employ some type of compression. Compression methods can be *lossy*—when a tolerable degree of deterioration in the visual quality of the resulting image is acceptable—or *lossless*—when the image is encoded in its full quality. The overall results of the compression process in terms of both storage savings—usually expressed in terms of compression ratio or bits per pixel (bpp)—and resulting quality loss (for the case of lossy techniques) may vary depending on the technique, format, options (such as the quality setting for JPEG), and actual image contents. As a general guideline, lossy compression should be used for general-purpose photographic images, whereas lossless compression should be preferred when dealing with line art, drawings, facsimiles, or images in which no loss of detail may be tolerable (most notably, space images and medical images). The topic of image compression will be discussed in detail in Chapter 17.

2.2 IMAGE FILE FORMATS

Most of the image file formats used to represent bitmap images consist of a *file header* followed by (often compressed) *pixel data*. The image file header stores information about the image, such as image height and width, number of bands, number of bits per pixel, and some signature bytes indicating the file type. In more complex file formats, the header may also contain information about the type of compression used and other parameters that are necessary to decode (i.e., decompress) the image.

The simplest file formats are the BIN and PPM formats. The BIN format simply consists of the raw pixel data, without any header. Consequently, the user of a BIN file must know the relevant image parameters (such as height and width) beforehand in order to use the image. The PPM format and its variants (PBM for binary images, PGM for grayscale images, PPM for color images, and PNM for any of them) are widely used in image processing research and many free tools for format conversion include them. The headers for these image formats include a 2-byte signature, or "magic number," that identifies the file type, the image width and height, the number of bands, and the maximum intensity value (which determines the number of bpp per band).

The Microsoft Windows bitmap (BMP) format is another widely used and fairly simple format, consisting of a header followed by raw pixel data.

The JPEG format is the most popular file format for photographic quality image representation. It is capable of high degrees of compression with minimal perceptual loss of quality. The technical details of the JPEG compression algorithm (and its presumed successor, the JPEG 2000 standard) will be discussed in Chapter 17.

Two other image file formats are very widely used in image processing tasks: GIF (Graphics Interchange Format) and TIFF (Tagged Image File Format). GIF uses an indexed representation for color images (with a palette of a maximum of 256 colors), the LZW (Lempel–Ziv–Welch) compression algorithm, and a 13-byte header. TIFF is a more sophisticated format with many options and capabilities, including the ability to represent truecolor (24 bpp) and support for five different compression schemes.

Portable Network Graphics (PNG) is an increasingly popular file format that supports both indexed and truecolor images. Moreover, it provides a patent-free replacement for the GIF format.

Some image processing packages adopt their own (sometimes proprietary) formats. Examples include XCF (the native image format of the GIMP image editing program) and RAW (which is a family of formats mostly adopted by camera manufacturers).

Since this book focuses on using MATLAB and its IPT, which provide built-in functionality for reading from and writing to most common image file formats,[2] we will not discuss in detail the specifics of the most popular image file formats any further. If you are interested in knowing more about the internal representation and details of these formats, refer to "Learn More About It" section at the end of the chapter.

[2] In the words of Professor Alasdair McAndrew [McA04], "you can use MATLAB for image processing very happily without ever really knowing the difference between GIF, TIFF, PNG, and all the other formats."

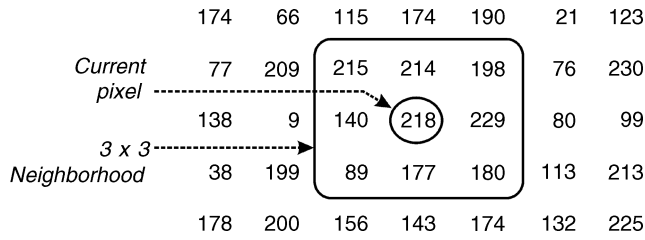

FIGURE 2.6 Pixels within a neighborhood.

2.3 BASIC TERMINOLOGY

This section introduces important concepts and terminology used to understand and express the properties of an image.

Image Topology It involves the investigation of fundamental image properties—usually done on binary images and with the help of morphological operators (see Chapter 13)—such as number of occurrences of a particular object, number of separate (not connected) regions, and number of holes in an object, to mention but a few.

Neighborhood The pixels surrounding a given pixel constitute its *neighborhood*, which can be interpreted as a smaller matrix containing (and usually centered around) the reference pixel. Most neighborhoods used in image processing algorithms are small square arrays with an odd number of pixels, for example, the 3×3 neighborhood shown in Figure 2.6.

In the context of image topology, neighborhood takes a slightly different meaning. It is common to refer to the *4-neighborhood* of a pixel as the set of pixels situated above, below, to the right, and to the left of the reference pixel (p), whereas the set of all of p's immediate neighbors is referred to as its *8-neighborhood*. The pixels that belong to the 8-neighborhood, but not to the 4-neighborhood, make up the *diagonal neighborhood* of p (Figure 2.7).

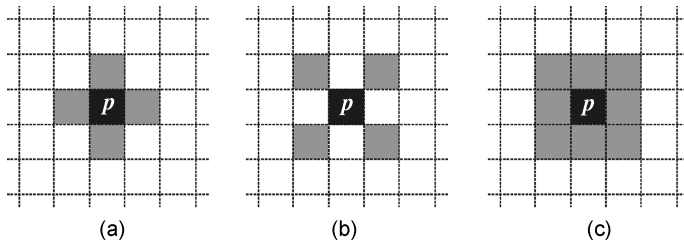

FIGURE 2.7 Concept of neighborhood of pixel p (from an image topology perspective): (a) 4-neighborhood; (b) diagonal neighborhood; (c) 8-neighborhood.

Adjacency In the context of image topology, two pixels p and q are *4-adjacent* if they are 4-neighbors of each other and *8-adjacent* if they are 8-neighbors of one another. A third type of adjacency—known as *mixed adjacency* (or simply *m-adjacency*)—is sometimes used to eliminate ambiguities (i.e., redundant paths) that may arise when 8-adjacency is used.

Paths In the context of image topology, a *4-path* between two pixels p and q is a sequence of pixels starting with p and ending with q such that each pixel in the sequence is 4-adjacent to its predecessor in the sequence. Similarly, an *8-path* indicates that each pixel in the sequence is 8-adjacent to its predecessor.

Connectivity If there is a 4-path between pixels p and q, they are said to be *4-connected*. Similarly, the existence of an 8-path between them means that they are *8-connected*.

Components A set of pixels that are connected to each other is called a *component*. If the pixels are 4-connected, the expression *4-component* is used; if the pixels are 8-connected, the set is called an *8-component*. Components are often labeled (and optionally pseudocolored) in a unique way, resulting in a *labeled image*, $L(x, y)$, whose pixel values are symbols of a chosen alphabet. The symbol value of a pixel typically denotes the outcome of a decision made for that pixel—in this case, the unique number of the component to which it belongs.

In MATLAB

MATLAB's IPT contains a function `bwlabel` for labeling connected components in binary images. An associated function, `label2rgb`, helps visualize the results by painting each region with a different color.

Figure 2.8 shows an example of using `bwlabel` and `label2rgb` and highlights the fact that the number of connected components will vary from 2 (when 8-connectivity is used, Figure 2.8b) to 3 (when 4-connectivity is used, Figure 2.8c).

Distances Between Pixels There are many image processing applications that require measuring distances between pixels. The most common distance measures

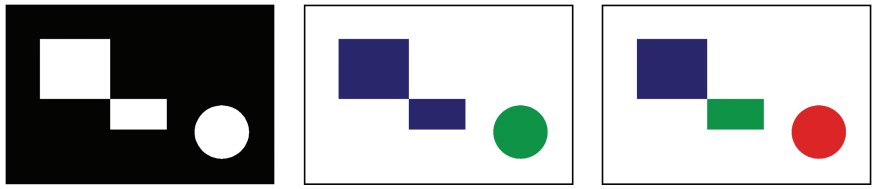

FIGURE 2.8 Connected components: (a) original (binary) image; (b) results for 8-connectivity; (c) results for 4-connectivity.

between two pixels p and q, of coordinates (x_0, y_0) and (x_1, y_1), respectively, are as follows:

- Euclidean distance:

$$D_e(p, q) = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \tag{2.3}$$

- D_4 (also known as *Manhattan* or *city block*) distance:

$$D_4(p, q) = |x_1 - x_0| + |y_1 - y_0| \tag{2.4}$$

- D_8 (also known as *chessboard*) distance:

$$D_8(p, q) = \max(|x_1 - x_0|, |y_1 - y_0|) \tag{2.5}$$

It is important to note that the distance between two pixels depends only on their coordinates, not their values. The only exception is the D_m distance, defined as "the shortest m-path between two m-connected pixels."

2.4 OVERVIEW OF IMAGE PROCESSING OPERATIONS

In this section, we take a preliminary look at the main categories of image processing operations. Although there is no universal agreement on a taxonomy for the field, we will organize them as follows:

- *Operations in the Spatial Domain*: Here, arithmetic calculations and/or logical operations are performed on the original pixel values. They can be further divided into three types:
  - *Global Operations*: Also known as *point operations*, in which the entire image is treated in a uniform manner and the resulting value for a processed pixel is a function of its original value, regardless of its location within the image. *Example:* contrast adjustment (Chapters 8 and 9).
  - *Neighborhood-Oriented Operations*: Also known as *local* or *area operations*, in which the input image is treated on a pixel-by-pixel basis and the resulting value for a processed pixel is a function of its original value and the values of its neighbors. *Example:* spatial-domain filters (Chapter 10).
  - *Operations Combining Multiple Images*: Here, two or more images are used as an input and the result is obtained by applying a (series of) arithmetic or logical operator(s) to them. *Example:* subtracting one image from another for detecting differences between them (Chapter 6).
- *Operations in a Transform Domain*: Here, the image undergoes a mathematical transformation—such as Fourier transform (FT) or discrete cosine transform (DCT)—and the image processing algorithm works in the transform domain. *Example:* frequency-domain filtering techniques (Chapter 12).

2.4.1 Global (Point) Operations

Point operations apply the same mathematical function, often called *transformation function*, to all pixels, regardless of their location in the image or the values of their neighbors. Transformation functions in the spatial domain can be expressed as

$$g(x, y) = T\left[f(x, y)\right] \tag{2.6}$$

where $g(x, y)$ is the processed image, $f(x, y)$ is the original image, and T is an operator on $f(x, y)$.

Since the actual coordinates do not play any role in the way the transformation function processes the original image, a shorthand notation can be used:

$$s = T[r] \tag{2.7}$$

where r is the original gray level and s is the resulting gray level after processing.

Figure 2.9 shows an example of a transformation function used to reduce the overall intensity of an image by half: $s = r/2$. Chapter 8 will discuss point operations and transformation functions in more detail.

2.4.2 Neighborhood-Oriented Operations

Neighborhood-oriented (also known as *local* or *area*) operations consist of determining the resulting pixel value at coordinates (x, y) as a function of its original value and the value of (some of) its neighbors, typically using a *convolution* operation. The

(a) (b)

FIGURE 2.9 Example of intensity reduction using a transformation function: (a) original image; (b) output image.

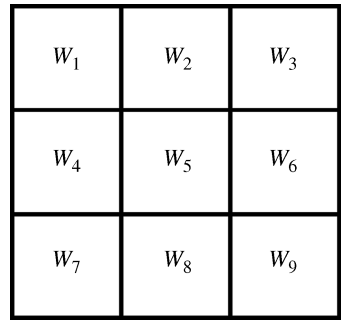

FIGURE 2.10 A 3 × 3 convolution mask, whose generic weights are W_1, ..., W_9.

convolution of a source image with a small 2D array (known as *window*, *template*, *mask*, or *kernel*) produces a destination image in which each pixel value depends on its original value and the value of (some of) its neighbors. The convolution mask determines which neighbors are used as well as the relative weight of their original values. Masks are normally 3 × 3, such as the one shown in Figure 2.10. Each mask coefficient ($W_1, \ldots, W_9$) can be interpreted as a *weight*. The mask can be thought of as a small window that is overlaid on the image to perform the calculation on one pixel at a time. As each pixel is processed, the window moves to the next pixel in the source image and the process is repeated until the last pixel has been processed.

Convolution operations are widely used in image processing. Depending on the choice of kernel, the same basic operation can be used to blur an image, enhance it, find its edges, or remove noise from it. Chapter 10 will explain convolution in detail and discuss image enhancement using neighborhood operations.

2.4.3 Operations Combining Multiple Images

There are many image processing applications that combine two images, pixel by pixel, using an arithmetic or logical operator, resulting in a third image, Z:

$$X \quad opn \quad Y = \quad Z \tag{2.8}$$

where X and Y may be images (arrays) or scalars, Z is necessarily an array, and *opn* is a binary mathematical (+, −, ×, /) or logical (AND, OR, XOR) operator. Figure 2.11 shows schematically how pixel-by-pixel operations work. Chapter 6 will discuss arithmetic and logic pixel-by-pixel operations in detail.

2.4.4 Operations in a Transform Domain

A *transform* is a mathematical tool that allows the conversion of a set of values to another set of values, creating, therefore, a new way of representing the same information. In the field of image processing, the original domain is referred to as *spatial domain*, whereas the results are said to lie in the *transform domain*. The

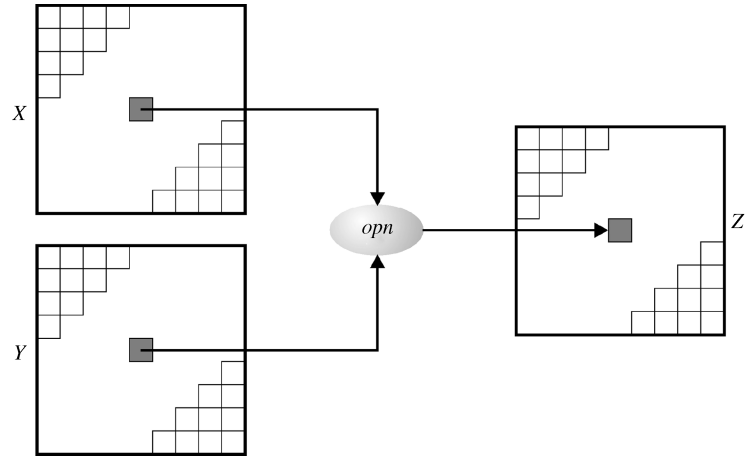

FIGURE 2.11 Pixel-by-pixel arithmetic and logic operations.

motivation for using mathematical transforms in image processing stems from the fact that some tasks are best performed by transforming the input images, applying selected algorithms in the transform domain, and eventually applying the inverse transformation to the result (Figure 2.12). This is what happens when we filter an image in the 2D frequency domain using the FT and its inverse, as we shall see in Chapter 11.

WHAT HAVE WE LEARNED?

- Images are represented in digital format in a variety of ways. *Bitmap* (also known as *raster*) representations use one or more two-dimensional arrays of pixels (picture elements), whereas *vector* representations use a series of drawing commands to represent an image.
- Binary images are encoded as a 2D array, using 1 bit per pixel, where usually—but not always—a 0 means "black" and a 1 means "white."

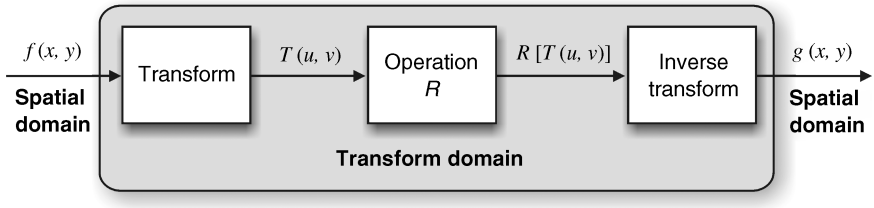

FIGURE 2.12 Operations in a transform domain.

- Gray-level (monochrome) images are encoded as a 2D array of pixels, using 8 bits per pixel, where a pixel value of 0 usually means "black" and a pixel value of 255 means "white," with intermediate values corresponding to varying shades of gray.
- The two most common ways of storing color image contents are *RGB* representation—in which each pixel is usually represented by a 24-bit number containing the amount of its red (R), green (G), and blue (B) components—and *indexed* representation—where a 2D array contains indices to a color palette (or *look up table*).
- Some of the most popular image file formats in use today are BMP, GIF, JPEG, TIFF, and PNG.
- MATLAB's built-in functions for reading images from files and writing images to files (`imread` and `imwrite`, respectively) support most file formats and their variants and options.
- Image topology is the field of image processing concerned with investigation of fundamental image properties (e.g., number of connected components and number of holes in an object) using concepts such as adjacency and connectivity.
- Image processing operations can be divided into two big groups: *spatial domain* and *transform domain*. Spatial-domain techniques can be further divided into pixel-by-pixel (*point*) or neighborhood-oriented (*area*) operations.

LEARN MORE ABOUT IT

- The book by Miano [Mia99] is a useful guide to graphic file formats.