UNIT: 1

Introduction to Digital Image Processing:

An image may be defined as a two-dimensional function, f(x, y), where x and y are *spatial* (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the *intensity* or *gray level* of the image at that point. When x, y, and the amplitude values of f are all finite, discrete quantities, we call the image a *digital image*. The field of *digital image processing* refers to processing digital images by means of a digital computer.

Digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are referred to as *picture elements, image elements, pels,* and *pixels*.

Low-level processes involve primitive operations such as image preprocessing to reduce noise, contrast enhancement, and image sharpening. A low-level process is characterized by the fact that both its inputs and outputs are images.

Mid-level processing on images involves tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. A mid-level process is characterized by the fact that its inputs generally are images, but its outputs are attributes extracted from those images (e.g., edges, contours, and the identity of individual objects).

Higher-level processing involves "making sense" of an ensemble of recognized objects, as in image analysis, and, at the far end of the continuum, performing the cognitive functions normally associated with vision.

1.2 The Origins of Digital Image Processing

One of the first applications of digital images was in the newspaper industry, when pictures were first sent by submarine cable between London and New York. Introduction of the Bartlane cable picture transmission system in the early 1920s reduced the time required to transport a picture across the Atlantic from more than a week to less than three hours. Specialized printing equipment coded pictures for cable transmission and then reconstructed them at the receiving end on a telegraph printer fitted with typefaces simulating a halftone pattern.

By the end of 1921 in favor of a technique based on photographic reproduction made from tapes perforated at the telegraph receiving terminal. The improvements are both in tonal quality and in resolution.

The early Bartlane systems were capable of coding images in five distinct levels of gray. This capability was increased to 15 levels in 1929. During this period, introduction of a system for developing a film plate via light beams that were modulated by the coded picture tape improved the reproduction process considerably.

The history of digital image processing is intimately tied to the development of the digital computer. In fact, digital images require so much storage and computational power that progress in the field of digital image processing has been dependent on the development of digital computers and of supporting technologies that include data storage, display, and transmission.

Some Applications:

Space applications, medical imaging, remote Earth resources observations, and astronomy.

Computerized axial tomography (CAT), also called computerized tomography (CT) for short, is one of the most important events in the application of image processing in medical diagnosis. Computerized axial tomography is a process in which a ring of detectors encircles an object (or patient) and an X-ray source, concentric with the detector ring, rotates about the object. The X-rays pass through the object and are collected at the opposite end by the corresponding detectors in the ring. As the source rotates, this procedure is repeated. Tomography consists of algorithms that use the sensed data to construct an image that represents a "slice" through the object.

Motion of the object in a direction perpendicular to the ring of detectors produces a set of such slices, which constitute a three-dimensional (3-D) rendition of the inside of the object. Tomography was invented independently by Sir Godfrey N. Hounsfield and Professor Allan M. Cormack, who shared the 1979 Nobel Prize in Medicine for their invention. It is interesting to note that X-rays were discovered in 1895 by Wilhelm Conrad Roentgen, for which he received the 1901 Nobel Prize for Physics.

In addition to applications in medicine and the space program, digital image processing techniques now are used in a broad range of applications. Computer procedures are used to enhance the contrast or code the intensity levels into color for easier interpretation of X-rays and other images used in industry, medicine, and the biological sciences. Geographers use the same or similar techniques to study pollution patterns from aerial and satellite imagery. Image enhancement and restoration procedures are used to process degraded images of unrecoverable objects or experimental results too expensive to duplicate. In archeology, image processing methods have successfully restored blurred pictures that were the only available records of rare artifacts lost or damaged after being photographed. In physics and related fields, computer techniques routinely enhance images of experiments in areas such as high-energy plasmas and electron microscopy. Similarly successful applications of image processing concepts can be found in astronomy, biology, nuclear medicine, law enforcement, defense, and industrial applications.

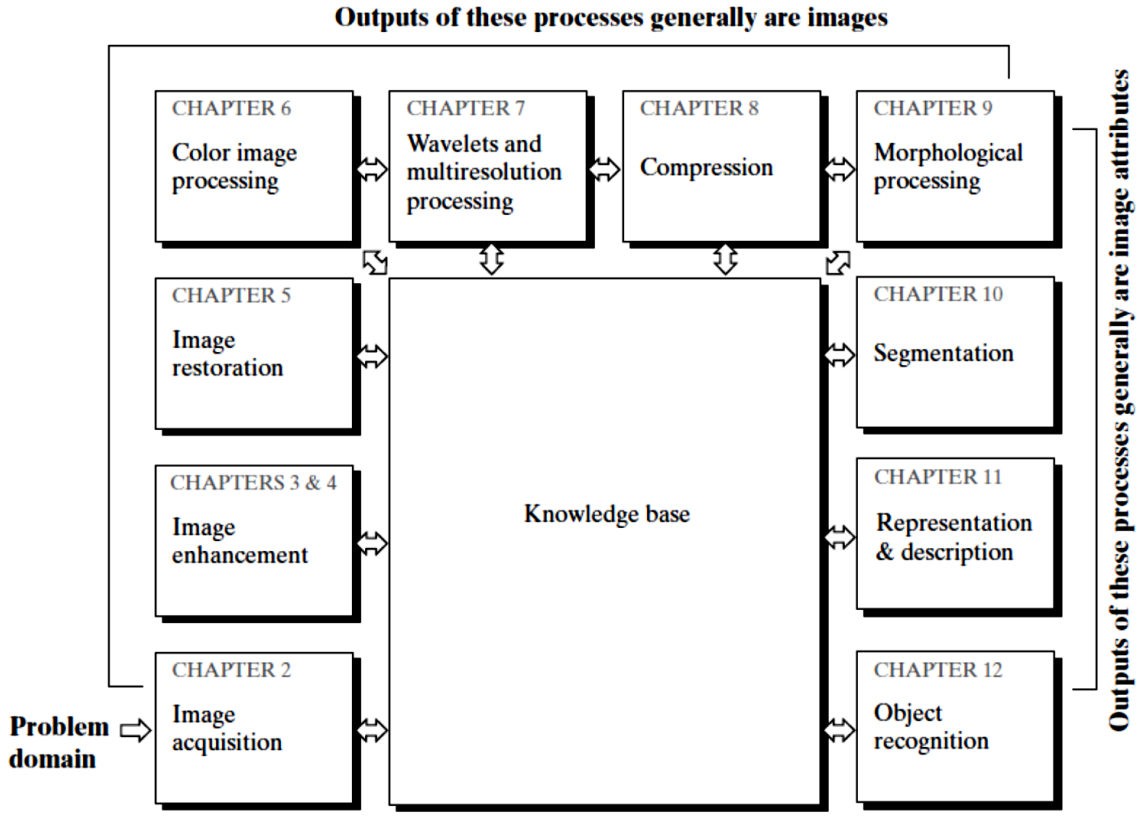
Fundamental Steps in Digital Image Processing

Image acquisition is the first process, being given an image that is already in digital form. Generally, the image acquisition stage involves preprocessing, such as scaling.

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. A familiar example of

enhancement is when we increase the contrast of an image because "it looks better." It is important to keep in mind that enhancement is a very subjective area of image processing

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation. Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a "good" enhancement result.



Color image processing is an area that has been gaining in importance because of the significant increase in the use of digital images over the Internet.

Wavelets are the foundation for representing images in various degrees of resolution. In particular, used for image data compression and for pyramidal representation, in which images are subdivided successively into smaller regions.

Compression, as the name implies, deals with techniques for reducing the storage required to save an image, or the bandwidth required to transmit it. Although storage technology has improved significantly over the past decade, the same cannot be said for transmission capacity. This is true particularly in uses of the Internet, which are characterized by significant pictorial content. Image compression is familiar (perhaps inadvertently) to most users of computers in the form of image file extensions, such as the jpg file extension used in the JPEG (Joint Photographic Experts Group) image compression standard.

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape. The material in this chapter begins a transition from processes that output images to processes that output image attributes

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually. On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely recognition is to succeed.

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region (i.e., the set of pixels separating one image region from another) or all the points in the region itself. In either case, converting the data to a form suitable for computer processing is necessary. The first decision that must be made is whether the data should be represented as a boundary or as a complete region. Boundary representation is appropriate when the focus is on external shape characteristics, such as corners and inflections. Regional representation is appropriate when the focus is on internal properties, such as texture or skeletal shape. In some applications, these representations complement each other. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. A method must also be specified for describing the data so that features of interest are highlighted. *Description,* also called *feature selection,* deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

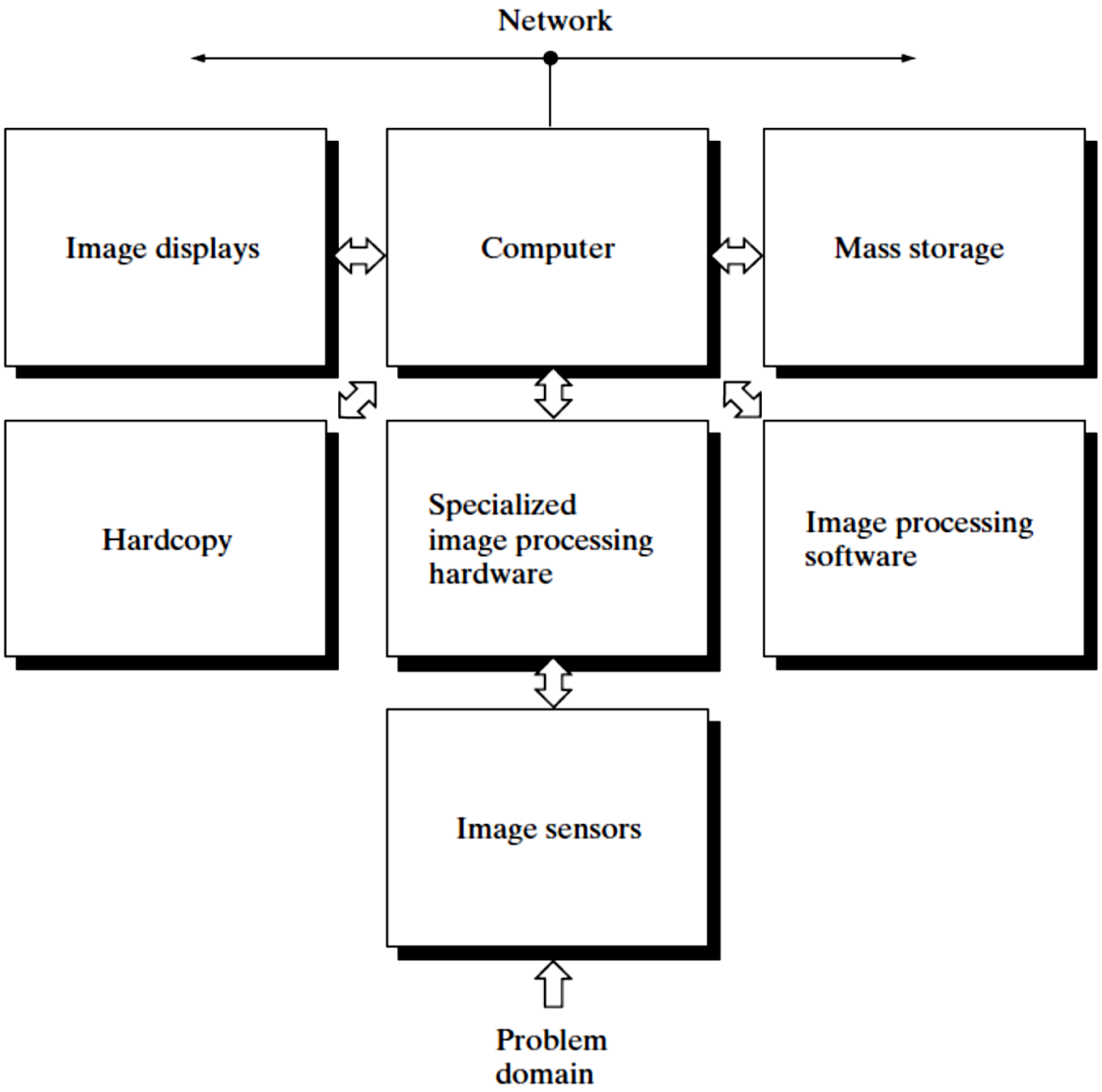
Recognition is the process that assigns a label (e.g., "vehicle") to an object based on its descriptors.

Components of an Image Processing System

Image sensing: With reference to *sensing,* two elements are required to acquire digital images. The first is a physical device that is sensitive to the energy radiated by the object we wish to image. The second, called a *digitizer,* is a device for converting the output of the physical sensing device into digital form. For instance, in a digital video camera, the sensors produce an electrical output proportional to light intensity. The digitizer converts these outputs to digital data.

Specialized image processing hardware usually consists of the digitizer, plus hardware that performs other primitive operations, such as an arithmetic logic unit (ALU), which performs arithmetic and logical operations in parallel on entire images. One example of how an ALU is used is in averaging images as quickly as they are digitized, for the purpose of noise reduction. This type of hardware sometimes is called a *front-end subsystem,* and its most distinguishing characteristic is speed. In other words, this unit performs functions that require fast data throughputs (e.g., digitizing and averaging video images at 30 frames_s) that the typical main computer cannot handle.

The *computer* in an image processing system is a general-purpose computer and can range from a PC to a supercomputer. In dedicated applications, sometimes specially designed computers are used to achieve a required level of performance, but our interest here is on general-purpose image processing systems. In these systems, almost any well-equipped PC-type machine is suitable for offline image processing tasks.



Software for image processing consists of specialized modules that perform specific tasks. A well-designed package also includes the capability for the user to write code that, as a minimum, utilizes the specialized modules. More sophisticated software packages allow the integration of those modules and general-purpose software commands from at least one computer language.

Mass storage capability is a must in image processing applications. An image of size 1024*1024 pixels, in which the intensity of each pixel is an 8-bit quantity, requires one

megabyte of storage space if the image is not compressed. When dealing with thousands, or even millions, of images, providing adequate storage in an image processing system can be a challenge. Digital storage forimage processing applications falls into three principal categories: (1) shortterm storage for use during processing, (2) on-line storage for relatively fast recall, and (3) archival storage, characterized by infrequent access. Storage is measured in bytes (eight bits), Kbytes (one thousand bytes), Mbytes (one million bytes), Gbytes (meaning giga, or one billion, bytes), and Tbytes (meaning tera, or one trillion, bytes).

One method of providing short-term storage is computer memory. Another is by specialized boards, called *frame buffers*, that store one or more images and can be accessed rapidly, usually at video rates (e.g., at 30 complete images per second).The latter method allows virtually instantaneous image *zoom*, as well as *scroll* (vertical shifts) and *pan* (horizontal shifts). Online storage generally takes the form of magnetic disks or optical-media storage. The key factor characterizing on-line storage is frequent access to the stored data. Finally, archival storage is characterized by massive storage requirements
but infrequent need for access. Magnetic tapes and optical disks housed in "jukeboxes" are the usual media for archival applications.

Image displays in use today are mainly color (preferably flat screen) TV monitors. Monitors are driven by the outputs of image and graphics display cards that are an integral part of the computer system. Seldom are there requirements for image display applications that cannot be met by display cards available commercially as part of the computer system. In some cases, it is necessary to have stereo displays, and these are implemented in the form of headgear containing two small displays embedded in goggles worn by the user.

Hardcopy devices for recording images include laser printers, film cameras, heat-sensitive devices, inkjet units, and digital units, such as optical and CD-ROM disks. Film provides the highest possible resolution, but paper is the obvious medium of choice for written material.For presentations, images are displayed on film transparencies or in a digital medium if image projection equipment is used.The latter approach is gaining acceptance as the standard for image presentations.

Networking is almost a default function in any computer system in use today. Because of the large amount of data inherent in image processing applications, the key consideration in image transmission is bandwidth. In dedicated networks, this typically is not a problem, but communications with remote sites via the Internet are not always as efficient. Fortunately, this situation is improving quickly as a result of optical fiber and other broadband technologies.

Image Sensing and Acquisition

The types of images in which we are interested are generated by the combination of an "illumination" source and the reflection or absorption of energy from that source by the elements of the "scene" being imaged. We enclose *illumination* and *scene* in quotes to emphasize the fact that they are considerably more general than the familiar situation in which a visible light source illuminates a common everyday 3-D (three-dimensional) scene. For example, the illumination may originate from a source of electromagnetic energy such as radar, infrared, or X-ray energy. But, as noted earlier, it could originate from less traditional sources, such as ultrasound or even a computer-generated illumination pattern.

Similarly, the scene elements could be familiar objects, but they can just as easily be molecules, buried rock formations, or a human brain. We could even image a source, such as acquiring images of the sun. Depending on the nature of the source, illumination energy is reflected from, or transmitted through, objects. An example in the first category is light reflected from a planar surface. An example in the second category is when X-rays pass through a patient's body for the purpose of generating a diagnostic X-ray film. In some applications, the reflected or transmitted energy is focused onto a photo converter (e.g., a phosphor screen), which converts the energy into visible light. Electron microscopy and some applications of gamma imaging use this approach.

Figure shows the three principal sensor arrangements used to transform illumination energy into digital images. The idea is simple: Incoming energy is transformed into a voltage by the combination of input electrical power and sensor material that is responsive to the particular type of energy being detected. The output voltage waveform is the response of the sensor(s), and a digital quantity is obtained from each sensor by digitizing its response.

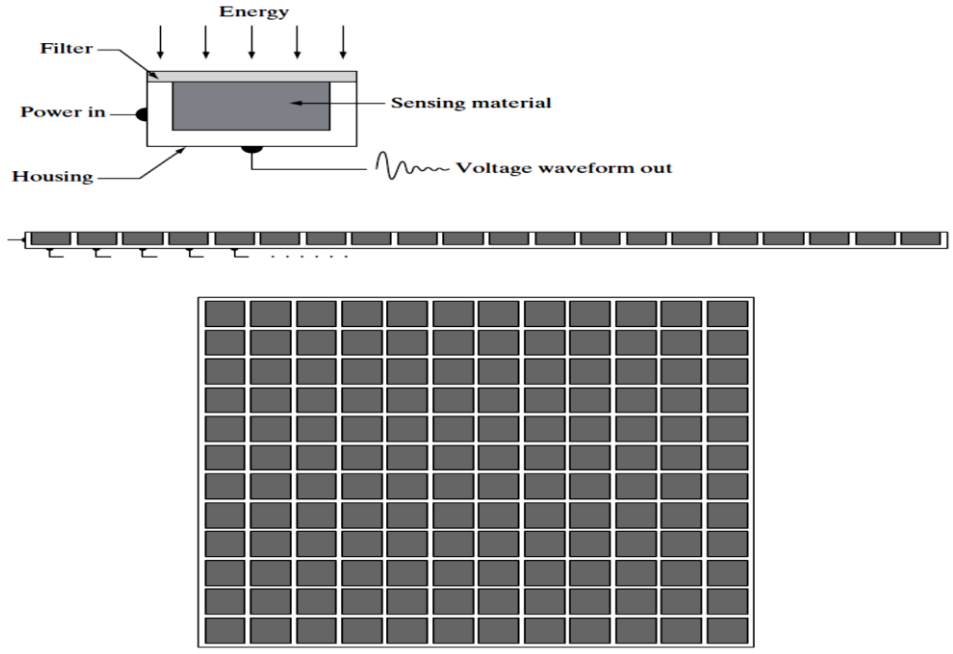


Image Acquisition Using a Single Sensor

The most familiar sensor of this type is the photodiode, which is constructed of silicon materials and whose output voltage waveform is proportional to light.The use of a filter in front

of a sensor improves selectivity. For example, a green (pass) filter in front of a light sensor favors light in the green band of the color spectrum. As a consequence, the sensor output will be stronger for green light than for other components in the visible spectrum.

In order to generate a 2-D image using a single sensor, there has to be relative displacements in both the x- and y-directions between the sensor and the area to be imaged. Figure shows an arrangement used in high-precision scanning, where a film negative is mounted onto a drum whose mechanical rotation provides displacement in one dimension. The single sensor is mounted on a lead screw that provides motion in the perpendicular direction. Since mechanical motion can be controlled with high precision, this method is an inexpensive (but slow) way to obtain high-resolution images. Other similar mechanical arrangements use a flat bed, with the sensor moving in two linear directions. These types of mechanical digitizers sometimes are referred to as *microdensitometers*.

Another example of imaging with a single sensor places a laser source coincident with the sensor. Moving mirrors are used to control the outgoing beam in a scanning pattern and to direct the reflected laser signal onto the sensor. This arrangement also can be used to acquire images using strip and array sensors.

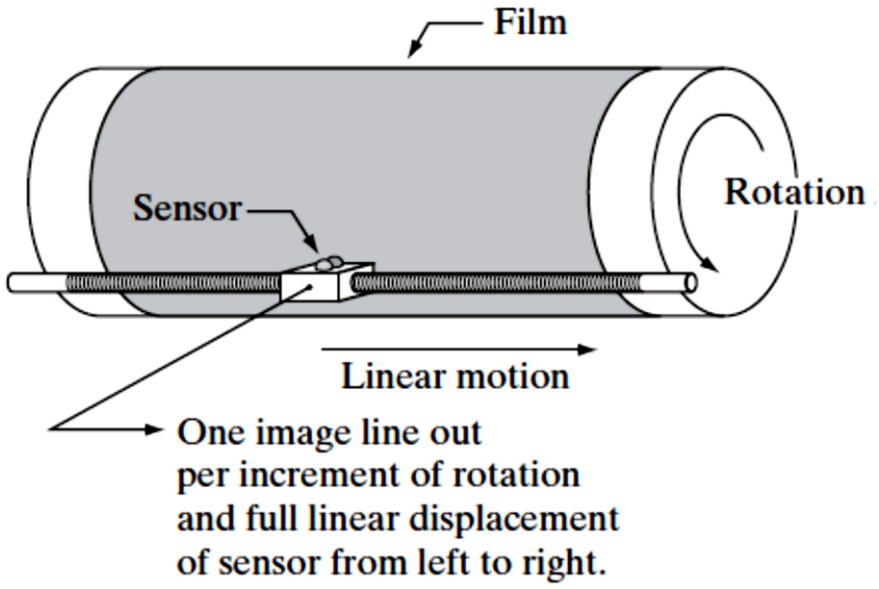
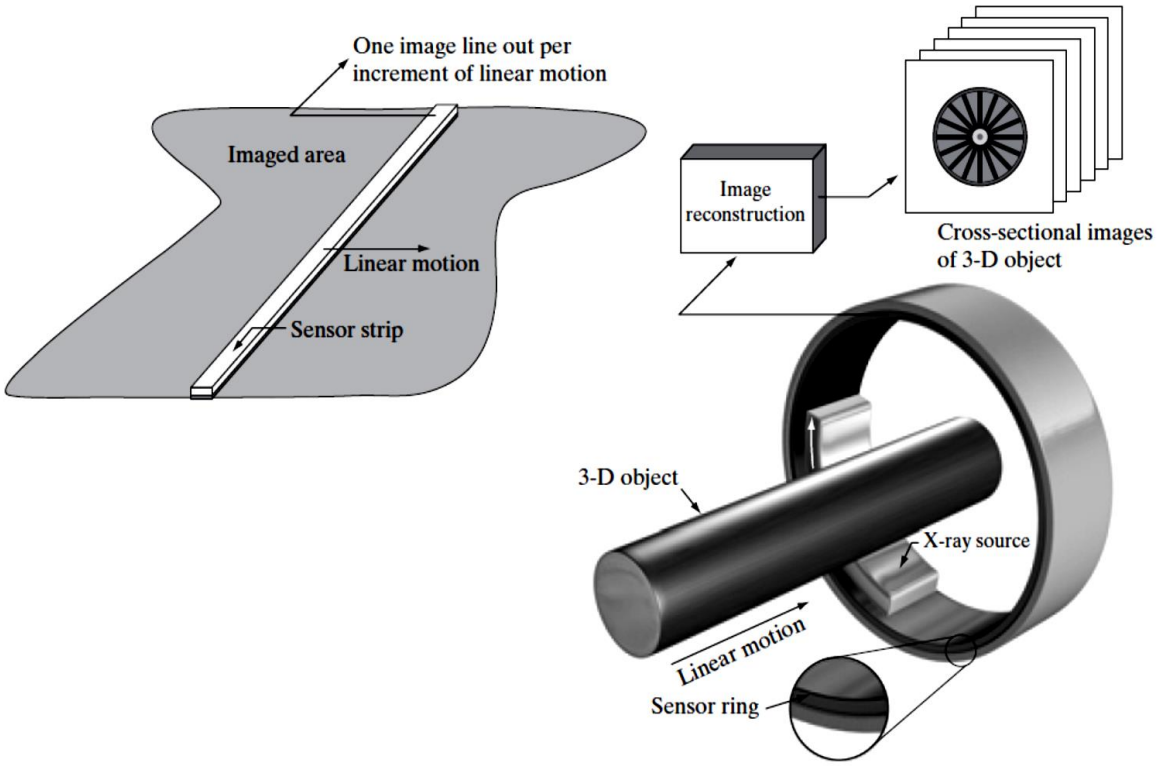


Image Acquisition Using Sensor Strips

A geometry that is used much more frequently than single sensors consists of an in-line arrangement of sensors in the form of a sensor strip. The strip provides imaging elements in one direction. Motion perpendicular to the strip provides imaging in the other direction, as shown in Fig.This is the type of arrangement used in most flat bed scanners. Sensing devices with 4000 or more in-line sensors are possible. In-line sensors are used routinely in airborne imaging applications, in which the imaging system is mounted on an aircraft that flies at a constant altitude and speed over the geographical area to be imaged. One-dimensional imaging sensor strips that respond to various bands of the electromagnetic spectrum are mounted perpendicular to the direction of flight. The imaging strip gives one line of an image at a time, and the motion of the strip completes the other dimension of a two-dimensional image. Lenses or other focusing schemes are used to project the area to be scanned onto the sensors.

Sensor strips mounted in a ring configuration are used in medical and industrial imaging to obtain cross-sectional ("slice") images of 3-D objects, as Fig shows A rotating X-ray source provides illumination and the portion of the sensors opposite the source collect the X-ray energy that pass through the object (the sensors obviously have to be sensitive to X-ray energy).This is the basis for medical and industrial computerized axial tomography (CAT) imaging. It is important to note that the output of the sensors must be processed by reconstruction algorithms whose objective is to transform the sensed data into meaningful cross-sectional images.



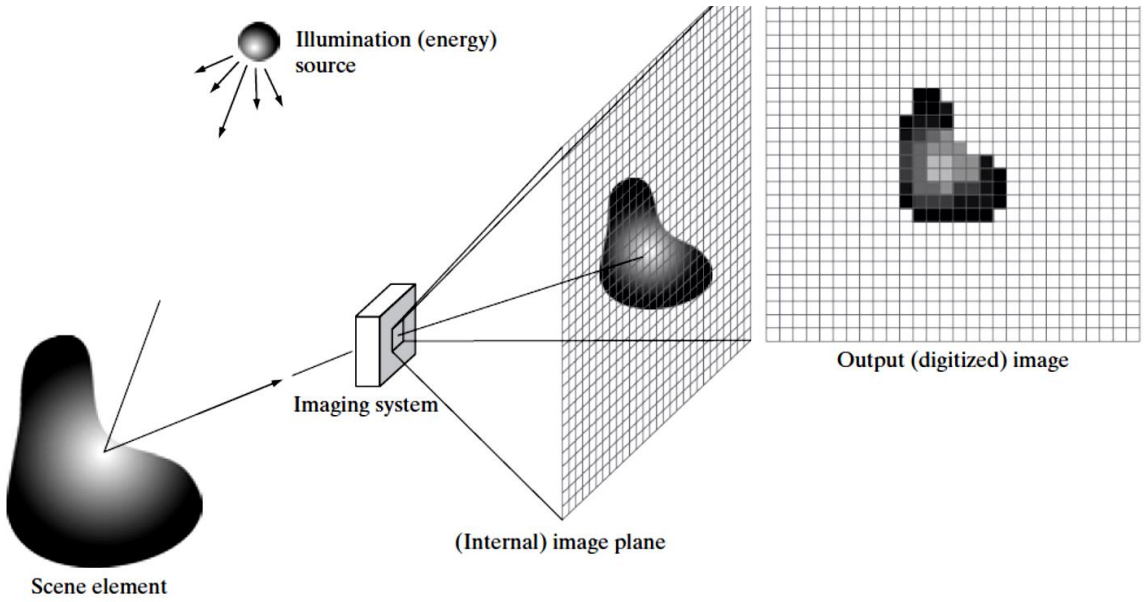
In other words, images are not obtained directly from the sensors by motion alone; they require extensive processing. A 3-D digital volume consisting of stacked images is generated as the object is moved in a direction perpendicular to the sensor ring. Other modalities of imaging based on the CAT principle include magnetic resonance imaging (MRI) and positron emission tomography (PET).

Image Acquisition Using Sensor Arrays

Individual sensors arranged in the form of a 2-D array. Numerous electromagnetic and some ultrasonic sensing devices frequently are arranged in an array format. This is also the predominant arrangement found in digital cameras. A typical sensor for these cameras is a CCD array, which can be manufactured with a broad range of sensing properties and can be packaged in rugged arrays of elements or more. CCD sensors are used widely in digital cameras and other light sensing instruments. The response of each sensor is proportional to the integral of the light energy projected onto the surface of the sensor, a property that is used in astronomical and other applications requiring low noise images. Noise reduction is achieved by letting the sensor integrate the input light signal over minutes or even hours. Since the sensor array shown in Fig

is two dimensional, its key advantage is that a complete image can be obtained by focusing the energy pattern onto the surface of the array.

The principal manner in which array sensors are used is shown in Fig. This figure shows the energy from an illumination source being reflected from a scene element, but, as mentioned at the beginning of this section, the energy also could be transmitted through the scene elements. The first function performed by the imaging system shown in Fig is to collect the incoming energy and focus it onto an image plane. If the illumination is light, the front end of the imaging system is a lens, which projects the viewed scene onto the lens focal plane, as Fig shows. The sensor array, which is coincident with the focal plane, produces outputs proportional to the integral of the light received at each sensor. Digital and analog circuitry sweep these outputs and convert them to a video signal, which is then digitized by another section of the imaging system. The output is a digital image, as shown diagrammatically in Fig.



A Simple Image Formation Model

Denote images by two-dimensional functions of the form f(x, y). The value or amplitude of f at spatial coordinates (x, y) is a positive scalar quantity whose physical meaning is determined by the source of the image. When an image is generated from a physical process, its values are proportional to energy radiated by a physical source (e.g., electromagnetic waves).As a consequence, f(x, y) must be nonzero and finite; that is,

$$0<f(x, y)<q.$$

The function f(x, y) may be characterized by two components: (1) the amount of source illumination incident on the scene being viewed, and (2) the amount of illumination reflected by the objects in the scene. Appropriately, these are called the *illumination* and *reflectance* components and are denoted by i(x, y) and r(x, y), respectively.The two functions combine as a product to form f(x, y):

$$f(x, y)=i(x, y)r(x, y)$$

$$\text{where } 0<i(x, y)<q \text{ (2.3-3) and } 0<r(x, y)<1.$$

reflectance is bounded by 0 (total absorption) and 1 (total reflectance).The nature of i(x, y) is determined by the illumination source, and r(x, y) is determined by the characteristics of the imaged objects.

It is noted that these expressions also are applicable to images formed via transmission of the illumination through a medium, such as a chest X-ray. In this case, we would deal with a *transmissivity* instead of a *reflectivity* function, but the limits would be the same as in Eq. (2.3-4), and the image function formed would be modeled as the product in Eq. (2.3-2).

The values given in Eqs. (2.3-3) and (2.3-4) are theoretical bounds. The following *average* numerical figures illustrate some typical ranges of i(x, y) for visible light. On a clear day, the sun may produce in excess of 90,000 lm_m2 of illumination on the surface of the Earth. This figure decreases to less than 10,000 lm_m2 on a cloudy day. On a clear evening, a full moon yields about 0.1 lm_m2 of illumination. The typical illumination level in a commercial office is about 1000 lm_m2. Similarly, the following are some typical values of r(x, y): 0.01 for black velvet, 0.65 for stainless steel, 0.80 for flat-white wall paint, 0.90 for silver-plated metal, and 0.93 for snow.

The intensity of a monochrome image at any coordinates (x0, y0) the *gray level* (/) of the image at that point.That is, / = f(x0 , y0)

From Eqs. (2.3-2) through (2.3-4), it is evident that / lies in the range Lmin <=/<= Lmax

In theory, the only requirement on *L*min is that it be positive, and on *L*max that it be finite. In practice,*L*min=iminrmin and *L*max=imaxrmax.Using the preceding average office illumination and range of reflectance values as guidelines, we may expect *L*min≠10 and *L*max≠1000 to be typical limits for indoor values in the absence of additional illumination.

The interval (Lmin , Lmax) is called the *gray scale*. Common practice is to shift this interval numerically to the interval [0, L-1], where /=0 is considered black and /=L-1 is considered white on the gray scale. All intermediate values are shades of gray varying from black to white.

Image Sampling and Quantization

The output of most sensors is a continuous voltage waveform whose amplitude and spatial behavior are related to the physical phenomenon being sensed. To create a digital image, we need to convert the continuous sensed data into digital form. This involves two processes: *sampling* and *quantization*.

Basic Concepts in Sampling and Quantization

The basic idea behind sampling and quantization is illustrated in Fig. Figure (a) shows a continuous image, f(x, y), that we want to convert to digital form. An image may be continuous with respect to the x- and y-coordinates, and also in amplitude. To convert it to digital form, we have to sample the function in both coordinates and in amplitude. Digitizing the coordinate values is called *sampling*. Digitizing the amplitude values is called *quantization*.

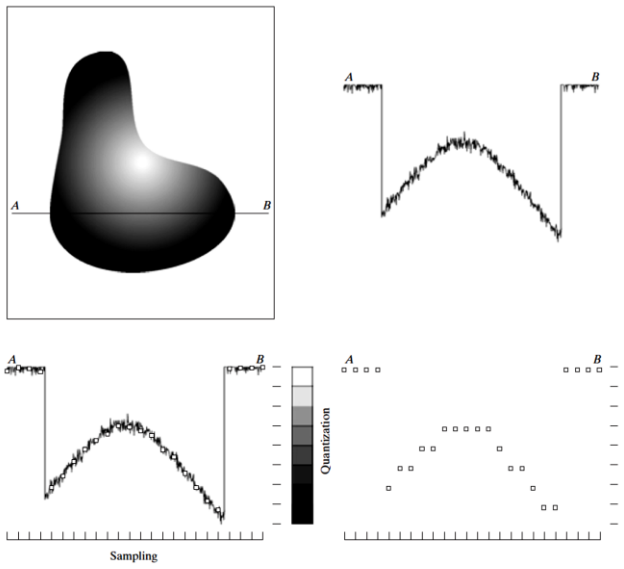
The one-dimensional function shown in Fig.(b) is a plot of amplitude (gray level) values of the continuous image along the line segment AB in Fig. (a).The random variations are due to image noise. To sample this function, we take equally spaced samples along line AB, as shown in Fig.(c).The location of each sample is given by a vertical tick mark in the bottom part of the figure. The samples are shown as small white squares superimposed on the function. The set of these discrete locations gives the sampled function. However, the values of the samples still span (vertically) a continuous range of gray-level values.

In order to form a digital function, the gray-level values also must be converted (*quantized*) into discrete quantities. The right side of Fig.(c) shows the gray-level scale divided into eight discrete levels, ranging from black to white. The vertical tick marks indicate the

specific value assigned to each of the eight gray levels. The continuous gray levels are quantized simply by assigning one of the eight discrete gray levels to each sample. The assignment is made depending on the vertical proximity of a sample to a vertical tick mark. The digital samples resulting from both sampling and quantization are shown in Fig.(d). Starting at the top of the image and carrying out this procedure line by line produces a two-dimensional digital image.

Sampling in the manner just described assumes that we have a continuous image in both coordinate directions as well as in amplitude. In practice, the method of sampling is determined by the sensor arrangement used to generate the image. When an image is generated by a single sensing element combined with mechanical motion, the output of the sensor is quantized in the manner described above. However, sampling is accomplished by selecting the number of individual mechanical increments at which we activate the sensor to collect data. Mechanical motion can be made very exact so, in principle, there is almost no limit as to how fine we can sample an image. However, practical limits are established by imperfections in the optics used to focus on the sensor an illumination spot that is inconsistent with the fine resolution achievable with mechanical displacements.

When a sensing strip is used for image acquisition, the number of sensors in the strip establishes the sampling limitations in one image direction. Mechanical motion in the other direction can be controlled more accurately, but it makes little sense to try to achieve sampling density in one direction that exceeds the sampling limits established by the number of sensors in the other. Quantization of the sensor outputs completes the process of generating a digital image.

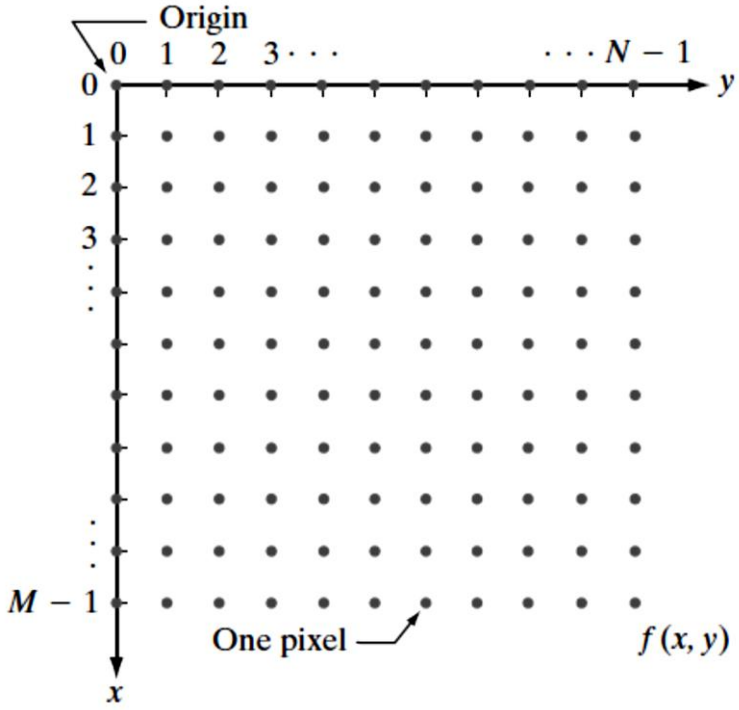


Representing Digital Images

The result of sampling and quantization is a matrix of real numbers. We will use two principal ways in this book to represent digital images. Assume that an image f(x, y) is sampled so that the resulting digital image hasMrows andNcolumns.

The values of the coordinates (x, y) now become *discrete* quantities. For notational clarity and convenience,we shall use integer values for these discrete coordinates. Thus, the values of the coordinates at the origin are (x, y)=(0, 0).

The next coordinate values along the first row of the image are represented as (x, y)=(0, 1). It is important to keep in mind that the notation (0, 1) is used to signify the second sample along the first row. It does *not* mean that these are the actual values of physical coordinates when the image was sampled.



The notation introduced in the preceding paragraph allows us to write the complete M*N digital image in the following compact matrix form:

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

The right side of this equation is by definition a digital image. Each element of this matrix array is called an *image element*, *picture element*, *pixel*, or *pel*. The terms *image* and *pixel* will be used throughout the rest of our discussions to denote a digital image and its elements.

In some discussions, it is advantageous to use a more traditional matrix notation to denote a digital image and its elements:

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix}.$$

Clearly, aij=f(x=i, y=j)=f(i, j)

Expressing sampling and quantization in more formal mathematical terms can be useful at times. Let *Z* and *R* denote the set of real integers and the set of real numbers, respectively. The sampling process may be viewed as partitioning the xy plane into a grid, with the coordinates of the center of each grid being a pair of elements from the Cartesian product Z2, which is the set of all ordered pairs of elements Azi, zj B, with zi and zj being integers from *Z*. Hence, f(x, y) is a digital image if (x, y) are integers from Z2 and f is a function that assigns a gray-level value (that is, a real number from the set of real numbers, *R*) to each distinct pair of coordinates (x, y). This functional assignment obviously is the quantization process described earlier. If the gray levels also are integers (as usually is the case in this and subsequent chapters), *Z* replaces *R*, and a digital image then becomes a 2-D function whose coordinates and amplitude values are integers.

This digitization process requires decisions about values for M, N, and for the number, L, of discrete gray levels allowed for each pixel. There are no requirements on *M* and *N*, other than that they have to be positive integers. However, due to processing, storage, and sampling hardware considerations, the number of gray levels typically is an integer power of 2: $L = 2^k$.

We assume that the discrete levels are equally spaced and that they are integers in the interval [0, L-1]. Sometimes the range of values spanned by the gray scale is called the *dynamic range* of an image, and we refer to images whose gray levels span a significant portion of the gray scale as having a high dynamic range. When an appreciable number of pixels exhibit this property, the image will have high contrast. Conversely, an image with low dynamic range tends to have a dull, washed out gray look.

The number, b, of bits required to store a digitized image is b=M*N*k.

When M=N, this equation becomes $b = N^2k$.

Some Basic Relationships Between Pixels

An image is denoted by f(x, y).When referring in this section to a particular pixel, we use lowercase letters, such as p and q.

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.

Connectivity:

Connectivity between pixels is a fundamental concept that simplifies the definition of numerous digital image concepts, such as regions and boundaries. To establish if two pixels are connected, it must be determined if they are neighbors and if their gray levels satisfy a specified criterion of similarity (say, if their gray levels are equal). For instance, in a binary image with values 0 and 1, two pixels may be 4-neighbors, but they are said to be connected only if they have the same value.

Let V be the set of gray-level values used to define adjacency. In a binary image, V={1} if we are referring to adjacency of pixels with value 1. In a grayscale image, the idea is the same, but set V typically contains more elements. For example, in the adjacency of pixels with a range of possible gray-level values 0 to 255, set V could be any subset of these 256 values. We consider three types of adjacency:

(a) 4-adjacency. Two pixels p and q with values from V are 4-adjacent if q is in the set N_4 (p).

(b) 8-adjacency. Two pixels p and q with values from V are 8-adjacent if q is in the set N_8 (p).

(c) m-adjacency (mixed adjacency).Two pixels p and q with values from V are m-adjacent if

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

(ii) q is in N_D (p) and the set has no pixels whose values are from V.

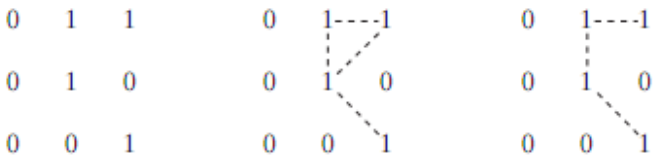
Mixed adjacency is a modification of 8-adjacency. It is introduced to eliminate the ambiguities that often arise when 8-adjacency is used. For example, consider the pixel arrangement shown in Fig. (a) for V= {1}.The three pixels at the top of Fig. (b) show multiple (ambiguous) 8- adjacency, as indicated by the dashed lines. This ambiguity is removed by using m-adjacency, as shown in Fig. (c).Two image subsets S1 and S2 are adjacent if some pixel in S1 is adjacent to some pixel in S2. It is understood here and in the following definitions that adjacent means 4-, 8-, or m-adjacent. A (digital) path (or curve) from pixel p with coordinates (x, y) to pixel q with coordinates (s, t) is a sequence of distinct pixels with coordinates

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

Where $(x_0, y_0) = (x, y), (x_n, y_n) = (s, t)$, and pixels (x_i, y_i) and (x_{i-1}, y_{i-1}) are adjacent for $1 \leq i \leq n$. In this case, n is the length of the path. If $(x_O, y_O) = (x_n, y_n)$, the path is a closed path. We can define 4-, 8-, or m-paths depending on the type of adjacency specified. For example, the paths shown in Fig. (b) between the northeast and southeast points are 8-paths, and the path in Fig. (c) is an m-path. Note the absence of ambiguity in the m-path. Let S represent a subset of pixels in an image. Two pixels p and q are said to be

connected in S if there exists a path between them consisting entirely of pixels in S. For any pixel p in S, the set of pixels that are connected to it in S is called a connected component of S. If it only has one connected component, then set S is called a connected set.

Let R be a subset of pixels in an image. We call R a region of the image if R is a connected set. The boundary (also called border or contour) of a region R is the set of pixels in the region that have one or more neighbors that are not in R. If R happens to be an entire image (which we recall is a rectangular set of pixels), then its boundary is defined as the set of pixels in the first and last rows and columns of the image. This extra definition is required because an image has no neighbors beyond its border. Normally, when we refer to a region, we are referring to a subset



a b c

Fig. (a) Arrangement of pixels; (b) pixels that are 8-adjacent (shown dashed) to the center pixel; (c) m-adjacency

Of an image, and any pixels in the boundary of the region that happen to coincide with the border of the image are included implicitly as part of the region boundary.

Distance Measures:

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

(a) $D(p, q) \geq 0 \quad (D(p, q) = 0 \quad \text{iff} \quad p = q)$,
(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

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

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 **D4 distance (also called city-block distance)** between p and q is defined as

$$D_4(p, q) = |x - s| + |y - t|.$$

In this case, the pixels having a D4 distance from (x, y) less than or equal to some value r form a diamond centered at (x, y). For example, the pixels with D4 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 D4 =1 are the 4-neighbors of (x, y).

The **D8 distance (also called chessboard distance)** between p and q is defined as

$$D_8(p, q) = \max(|x - s|, |y - t|).$$

In this case, the pixels with D8 distance from(x, y) less than or equal to some value r form a square centered at (x, y). For example, the pixels with D8 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 D8=1 are the 8-neighbors of (x, y). Note that the D4 and D8 distances between p and q are independent of any paths that might exist between the points because these distances involve only the coordinates of the points. If we elect to consider m-adjacency,

however, the D_m distance between two points is defined as the shortest m-path between the points. In this case, the distance between two pixels will depend on the values of the pixels along the path, as well as the values of their neighbors. For instance, consider the following arrangement of pixels and assume that p, p_2 , and p_4 have value 1 and that p_1 and p_3 can have a value of 0 or 1:

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

Suppose that we consider adjacency of pixels valued 1 (i.e. = {1}). If p_1 and p_3 are 0, the length of the shortest m-path (the Dm distance) between p and p_4 is 2. If p_1 is 1, then p_2 and p will no longer be m-adjacent (see the definition of m-adjacency) and the length of the shortest m-path becomes 3 (the path goes through the point's $pp_1p_2p_4$). Similar comments apply if p_3 is 1 (and p_1 is 0); in this case, the length of the shortest m-path also is 3. Finally, if both p_1 and p_3 are 1 the length of the shortest m-path between p and p_4 is 4. In this case, the path goes through the sequence of point's $pp_1p_2p_3p_4$.

Image Operations on a Pixel Basis

Images were represented in the form of matrices. As we know, matrix division is not defined. However, when we refer to an operation like "dividing one image by another," we mean specifically that the division is carried out between *corresponding* pixels in the two images. Thus, for example, if f and g are images, the first element of the image formed by "dividing" f by g is simply the first pixel in f divided by the first pixel in g; of course, the assumption is that none of the pixels in g have value 0. Other arithmetic and logic operations are similarly defined between corresponding pixels in the images involved.

Linear and Nonlinear Operations

Let *H* be an operator whose input and output are images. *H*is said to be a *linear* operator if, for any two images f and g and any two scalars a and b,

$$H(af + bg) = aH(f) + bH(g).$$

the result of applying a linear operator to the sum of two images (that have been multiplied by the constants shown) is identical to applying the operator to the images individually, multiplying the results by the appropriate constants, and then adding those results.

Linear operations are exceptionally important in image processing because they are based on a significant body of well-understood theoretical and practical results. Although nonlinear operations sometimes offer better performance, they are not always predictable, and for the most part are not well understood theoretically.

An Introduction to the Mathematical Tools Used in Digital Image Processing

This section has two principal objectives: (1) to introduce you to the various mathematical tools we use throughout the book; and (2) to help you begin de- veloping a "feel" for how these tools are used by applying them to a variety of basic image-processing tasks, some of which will be used numerous times in subsequent discussions. We expand the scope of the tools and their application as necessary in the following chapters.

Array versus Matrix Operations

An *array* operation involving one or more images is carried out on a *pixel-by-pixel* basis. We mentioned earlier in this chapter that images can be viewed equivalently as matrices. In fact, there are many situations in which opera- tions between images are carried out using matrix theory (see Section 2.6.6). It is for this reason that a clear distinction must be made between array and matrix operations. For example, consider the following 2 * 2 images:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

The *array product* of these two images is

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

On the other hand, the *matrix product* is given by

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

We assume array operations throughout the book, unless stated otherwise. For example, when we refer to raising an image to a power, we mean that each individual pixel is raised to that power; when we refer to dividing an image by another, we mean that the division is between corresponding pixel pairs, and so on.

Linear versus Nonlinear Operations

One of the most important classifications of an image-processing method is whether it is linear or nonlinear. Consider a general operator, *H* that produces an output image,

g (x, *y*), for a given input image, *f* (*x*, *y*):

$$H\left[f(x, y)\right] = g(x, y)$$

H is said to be a *linear operator* if

$$\begin{aligned} H\left[a_i f_i(x, y) + a_j f_j(x, y)\right] &= a_i H\left[f_i(x, y)\right] + a_j H\left[f_j(x, y)\right] \\ &= a_i g_i(x, y) + a_j g_j(x, y) \end{aligned} \qquad (2.6\text{-}2)$$

Where a_i, a_j, $f_i(x, y)$, and $f_j(x, y)$ are arbitrary constants and images (of the same size), respectively. Equation (2.6-2) indicates that the output of a linear operation due to the sum of two inputs is the same as performing the operation on the inputs individually and then summing the results. In addition, the output of a linear operation to a constant times an input is the same as the out- put of the operation due to the original input multiplied by that constant. The first property is called the property of *additivity* and the second is called the property of *homogeneity*.

As a simple example, suppose that H is the sum operator, ©; that is, the function of this operator is simply to sum its inputs. To test for linearity, we start with the left side of Eq. (2.6-2) and attempt to prove that it is equal to the right side:

$$\begin{aligned}\sum\left[a_i f_i(x, y) + a_j f_j(x, y)\right] &= \sum a_i f_i(x, y) + \sum a_j f_j(x, y) \\ &= a_i \sum f_i(x, y) + a_j \sum f_j(x, y) \\ &= a_i g_i(x, y) + a_j g_j(x, y)\end{aligned}$$

Where the first step follows from the fact that summation is distributive. So, an expansion of the left side is equal to the right side of Eq. (2.6-2), and we conclude that the sum operator is linear.

On the other hand, consider the max operation, whose function is to find the maximum value of the pixels in an image. For our purposes here, the simplest way to prove that this operator is nonlinear, is to find an example that fails the test in Eq. (2.6-2). Consider the following two images

$$f_1 = \begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix} \quad \text{and} \quad f_2 = \begin{bmatrix} 6 & 5 \\ 4 & 7 \end{bmatrix}$$

and suppose that we let $a_1 = 1$ and $a_2 = -1$. To test for linearity, we again start with the left side of Eq. (2.6-2):

$$\max\left\{(1)\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix} + (-1)\begin{bmatrix} 6 & 5 \\ 4 & 7 \end{bmatrix}\right\} = \max\left\{\begin{bmatrix} -6 & -3 \\ -2 & -4 \end{bmatrix}\right\}$$
$$= -2$$

Working next with the right side, we obtain

$$(1)\max\left\{\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}\right\} + (-1)\max\left\{\begin{bmatrix} 6 & 5 \\ 4 & 7 \end{bmatrix}\right\} = 3 + (-1)7$$
$$= -4$$

The left and right sides of Eq. (2.6-2) are not equal in this case, so we have proved that in general the max operator is nonlinear.

As you will see in the next three chapters, especially in Chapters 4 and 5, linear operations are exceptionally important because they are based on a large body of theoretical and practical results that are applicable to image process- ing. Nonlinear systems are not nearly as well understood, so their scope of application is more limited. However, you will encounter in the following chapters several nonlinear image processing operations whose performance far exceeds what is achievable by their linear counterparts.

Arithmetic Operations

Arithmetic operations between images are array operations which, as discussed in Section 2.6.1, means that arithmetic operations are carried out between cor- responding pixel pairs. The four arithmetic operations are denoted as

$$s(x, y) = f(x, y) + g(x, y)$$
$$d(x, y) = f(x, y) - g(x, y)$$

$$p(x, y) = f(x, y) * g(x, y)$$
$$v(x, y) = f(x, y) \div g(x, y)$$

It is understood that the operations are performed between corresponding pixel pairs in f and g for $x = 0, 1, 2, \ldots, M - 1$ and $y = 0, 1, 2, \ldots, N - 1$

A few comments about implementing image arithmetic operations are in order before we leave this section. In practice, most images are displayed using 8 bits (even 24-bit color images consist of three separate 8-bit channels). Thus, we expect image values to be in the range from 0 to 255. When images are saved in a standard format, such as TIFF or JPEG, conversion to this range is automatic. However, the approach used for the conversion depends on the system used. For example, the values in the difference of two 8-bit images can range from a minimum of -255 to a maximum of 255, and the values of a sum image can range from 0 to 510. Many software packages simply set all negative values to 0 and set to 255 all values that exceed this limit when converting images to 8 bits. Given an image f, an approach that guarantees that the full range of an arithmetic operation between images is "captured" into a fixed number of bits is as follows. First, we perform the operation

$$f_m = f - \min(f) \qquad (2.6\text{-}10)$$

which creates an image whose minimum value is 0. Then, we perform the operation

$$f_s = K[f_m / \max(f_m)] \qquad (2.6\text{-}11)$$

which creates a scaled image, f_s, whose values are in the range [0, K]. When working with 8-bit images, setting $K = 255$ gives us a scaled image whose in- tensities span the full 8-bit scale from 0 to 255. Similar comments apply to 16-bit images or higher. This approach can be used for all arithmetic operations. When performing division, we have the extra requirement that a small number should be added to the pixels of the divisor image to avoid division by 0.

Set and Logical Operations

In this section, we introduce briefly some important set and logical operations. We also introduce the concept of a fuzzy set.

Basic set operations

Let A be a set composed of *ordered pairs* of real numbers. If $a = (a_1, a_2)$ is an *element* of A, then we write

$$a \in A \tag{2.6-12}$$

Similarly, if a is not an element of A, we write

$$a \notin A \tag{2.6-13}$$

The set with no elements is called the *null* or *empty set* and is denoted by the symbol Ø.

A set is specified by the contents of two braces

If every element of a set A is also an element of a set B, then A is said to be a *subset* of B, denoted as

$$A \subseteq B \tag{2.6-14}$$

The *union* of two sets A and B, denoted by

$$C = A \cup B \tag{2.6-15}$$

is the set of elements belonging to either A, B, *or* both. Similarly, the *Intersection* of two sets A and B, denoted by

$$D = A \cap B \tag{2.6-16}$$

Is the set of elements belonging to both A *and* B. Two sets A and B are said to be *Disjoint* or *mutually exclusive* if they have no common elements, in which case,

$$A \cap B = \varnothing \tag{2.6-17}$$

The *set universe*, U, is the set of all elements in a given application. By definition, all set elements in a given application are members of the universe defined for that application. For example, if you are working with the set of real numbers, then the set universe is the real line, which contains all the real numbers. In image processing, we typically define the universe to be the rectangle containing all the pixels in an image.

The *complement* of a set A is the set of elements that are not in A:

$$A^c = \{w | w \notin A\}$$

The *difference* of two sets A and B, denoted $A - B$, is defined as

$$A - B = \{w | w \in A, w \notin B\} = A \cap B^c$$

(2.6-19) we see that this is the set of elements that belong to A, but not to B. We could,

for example, define A^C in terms of U and the set difference operation:

$A^C = U - A$.

Figure 2.31 illustrates the preceding concepts, where the universe is the set of coordinates contained within the rectangle shown, and sets A and B are the sets of coordinates contained within the boundaries shown. The result of the set operation indicated in each figure is shown in gray

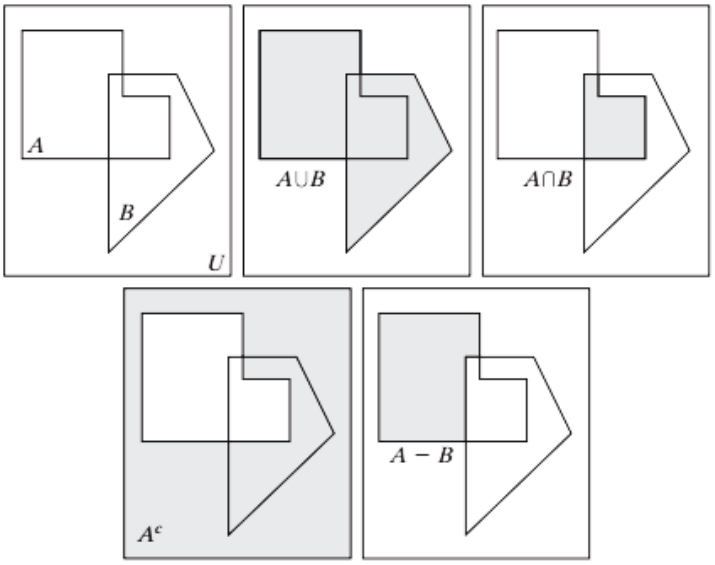
In the preceding discussion, set membership is based on position (coordinates). An implicit assumption when working with images is that the intensity of all pixels in the sets is the same, as we have not defined set operations involving intensity values (e.g., we have not specified what the intensities in the intersection of two sets is). The only way that the operations illustrated in Fig.

2.31 can make sense is if the images containing the sets are binary, in which case we can talk about set membership based on coordinates, the assumption being that all member of the sets have the same intensity. We discuss this in more de- tail in the following subsection.

When dealing with gray-scale images, the preceding concepts are not applicable, because we have to specify the intensities of all the pixels resulting from a set operation. In fact, as you will see in Sections 3.8 and 9.6, the union and intersection operations for gray-scale values usually are defined as the max and min of corresponding pixel pairs, respectively, while the complement is defined as the pairwise differences between a constant and the intensity of every pixel in an image. The fact that we deal with corresponding pixel pairs tells us that gray-scale set operations are array operations, as defined in Section 2.6.1. The following example is a brief illustration of set operations involving gray-scale images. We discuss these concepts further in the two sections mentioned above.

a b c
d e

FIGURE 2.31
(a) Two sets of coordinates, A and B, in 2-D space. (b) The union of A and B. (c) The intersection of A and B. (d) The complement of A. (e) The difference between A and B. In (b)–(e) the shaded areas represent the members of the set operation indicated.



Logical operations

When dealing with binary images, we can think of *foreground* (1-valued) and *background* (0-valued) sets of pixels. Then, if we define regions (objects) as being composed of foreground pixels, the set operations illustrated in Fig. 2.31 become operations between the coordinates of objects in a binary image. When dealing with binary images, it is common practice to refer to union, intersection, and complement as the OR, AND, and NOT *logical* operations, where "logical" arises from logic theory in which 1 and 0 denote true and false, respectively.

Consider two regions (sets) A and B composed of foreground pixels. The OR of these two sets is the set of elements (coordinates) belonging either to A or B or to both. The AND operation is the set of elements that are common to A and B. The NOT operation of a set A is the set of elements not in A. Be- cause we are dealing with images, if A is a given set of foreground pixels, NOT(A) is the set of all pixels in the image that are not in A, these pixels being background pixels and possibly other foreground pixels. We can think of this operation as turning all elements in A to 0 (black) and all the elements not in A to 1 (white). Figure 2.33 illustrates these operations. Note in the fourth row that the result of the operation shown is the set of foreground pix- els that belong to A but not to B, which is the definition of set difference in Eq. (2.6-19). The last row in the figure is the XOR (exclusive OR) operation, which is the set of foreground pixels belonging to A or B, but not both. Ob- serve that the preceding operations are between regions, which clearly can be irregular and of different sizes. This is as opposed to the gray-scale operations discussed earlier, which are array operations and thus require sets whose spa- tial dimensions are the same. That is, gray-scale set operations involve com- plete images, as opposed to regions of images.

We need be concerned in theory only with the cability to implement the AND, OR, and NOT logic operators because these three operators are *functionally*

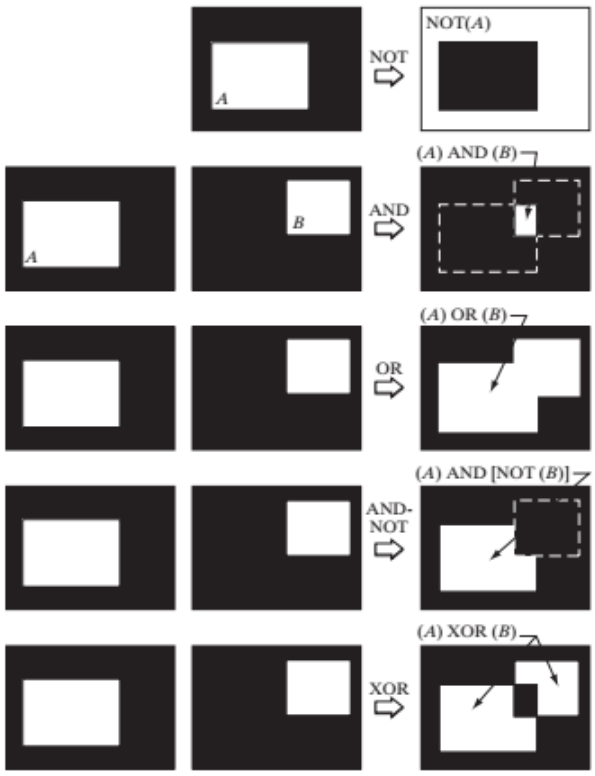


FIGURE 2.33 Illustration of logical operations involving foreground (white) pixels. Black represents binary 0s and white binary 1s. The dashed lines are shown for reference only. They are not part of the result.

Complete. In other words, any other logic operator can be implemented by using only these three basic functions, as in the fourth row of Fig. 2.33, where we implemented the set difference operation using AND and NOT. Logic operations are used extensively in image morphology

Fuzzy sets

The preceding set and logical results are *crisp* concepts, in the sense that ele- ments either are or are not members of a set. This presents a serious limitation in some applications. Consider a simple example. Suppose that we wish to cat- egorize all people in the world as being young or not young. Using crisp sets, let U denote the set of all people and let A be a subset of U, which we call the *set of young people*. In order to form set A, we need a *membership function* that assigns a value of 1 or 0 to every element (person) in U. If the value as- signed to an element of U is 1, then that element is a member of A; otherwise it is not. Because we are dealing with a bi-valued logic, the membership func- tion simply defines a threshold at or below which a person is considered young, and above which a person is considered not young. Suppose that we define as young any person of age 20 or younger. We see an immediate difficulty. A per- son whose age is 20 years and 1 sec would not be a member of the set of young people. This limitation arises regardless of the age threshold we use to classify a person as being young. What we need is more flexibility in what we mean by "young," that is, we need a *gradual* transition from young to not young. The the- ory of *fuzzy sets* implements this concept by utilizing membership functions

That are gradual between the limit values of 1 (definitely young) to 0 (definite- ly not young). Using fuzzy sets, we can make a statement such as a person being 50% young (in the middle of the transition between young and not young). In other words, age is

an imprecise concept, and fuzzy logic provides the tools to deal with such concepts. We explore fuzzy sets in detail in Section 3.8.

Spatial Operations

Spatial operations are performed directly on the pixels of a given image. We classify spatial operations into three broad categories: (1) single-pixel opera- tions, (2) neighborhood operations, and (3) geometric spatial transformations.

Single-pixel operations

The simplest operation we perform on a digital image is to alter the values of its individual pixels based on their intensity. This type of process may be ex- pressed as a transformation function, T, of the form:

$$s = T(z) \qquad (2.6\text{-}20)$$

Where z is the intensity of a pixel in the original image and s is the (mapped) intensity of the corresponding pixel in the processed image. For example, Fig. 2.34 shows the transformation used to obtain the negative of an 8-bit image, such as the image in Fig. 2.32(b), which we obtained using set operations. We discuss in Chapter 3 a number of techniques for specifying intensity trans- formation functions.

Neighborhood operations

Let S_{xy} denote the set of coordinates of a neighborhood centered on an arbi- trary point (x, y) in an image, f. Neighborhood processing generates a corres- ponding pixel at the same coordinates in an output (processed) image, g, such that the value of that pixel is determined by a specified operation involving the pixels in the input image with coordinates in S_{xy}. For example, suppose that the specified operation is to compute the average value of the pixels in a rec- tangular neighborhood of size $m * n$ centered on (x, y). The locations of pixels

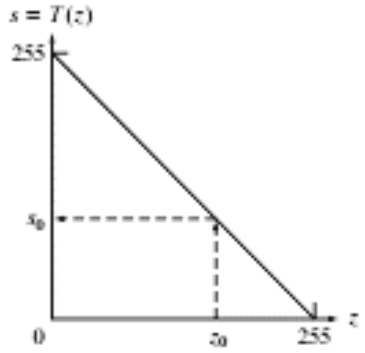
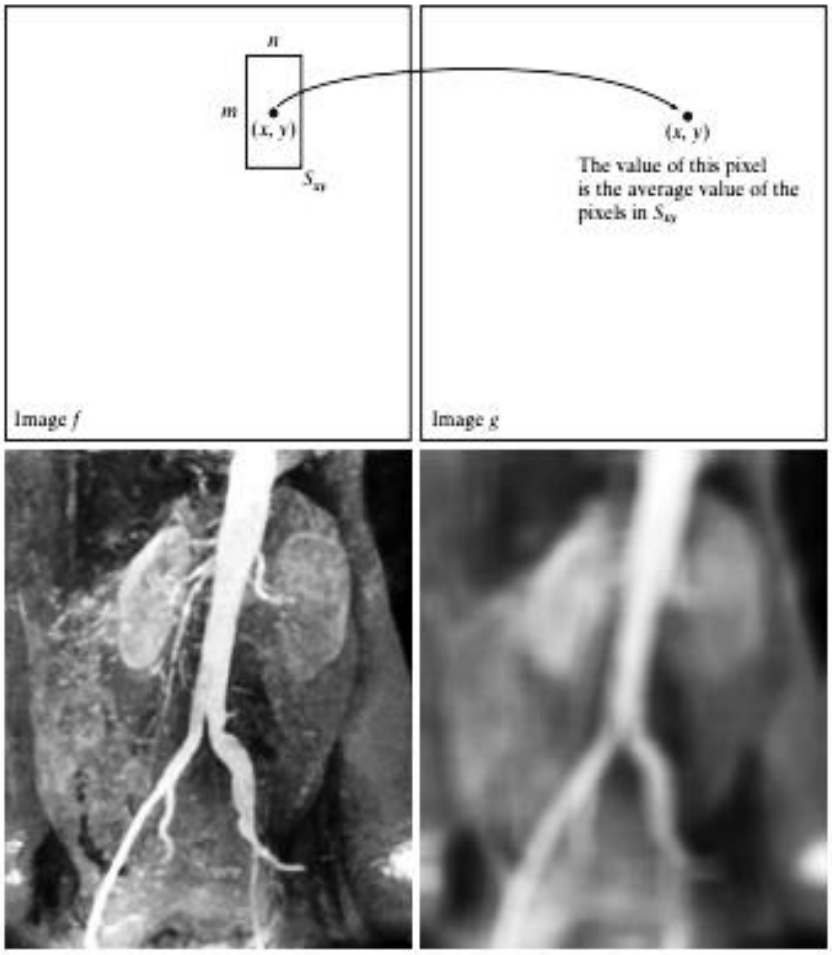


FIGURE 2.34 Intensity transformation function used to obtain the negative of an 8-bit image. The dashed arrows show transformation of an arbitrary input intensity value z_0 into its corresponding output value s_0.



a b
c d

FIGURE 2.35
Local averaging using neighborhood processing. The procedure is illustrated in (a) and (b) for a rectangular neighborhood. (c) The aortic angiogram discussed in Section 1.3.2. (d) The result of using Eq. (2.6-21) with $m = n = 41$. The images are of size 790 × 686 pixels.

in this region constitute the set S_{xy}. Figures 2.35(a) and (b) illustrate the process. We can express this operation in equation form as

$$g(x, y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} f(r, c) \qquad (2.6\text{-}21)$$

where r and c are the row and column coordinates of the pixels whose coordinates are members of the set S_{xy}. Image g is created by varying the coordi- nates (x, y) so that the center of the neighborhood moves from pixel to pixel in image f, and repeating the neighborhood operation at each new location. For instance, the image in Fig. 2.35(d) was created in this manner using a neigh- borhood of size 41 * 41. The net effect is to perform local blurring in the orig- inal image. This type of process is used, for example, to eliminate small details and thus render "blobs" corresponding to the largest regions of an image. We discuss neighborhood processing in Chapters 3 and 5, and in several other places in the book.

Image Transforms

Need For Image Transforms & Spatial Frequencies in Image Processing

All the image processing approaches discussed thus far operate directly on the pixels of the input image; that is, they work directly in the *spatial domain*. In some cases, image processing tasks are best formulated by transforming the input images, carrying the specified task in a *transform domain*, and applying the inverse transform to return to the spatial domain. You will encounter a number of different transforms as you proceed through the book. A particu- larly important class of 2-D linear transforms, denoted $T(u, v)$, can be ex- pressed in the general form

$$T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) r(x, y, u, v) \qquad (2.6\text{-}30)$$

where $f(x, y)$ is the input image, $r(x, y, u, v)$ is called the *forward transformation kernel*, and Eq. (2.6-30) is evaluated for $u = 0, 1, 2, \acute{A}, M - 1$ and $v = 0, 1, 2, \acute{A}, N - 1$. As before, x and y are spatial variables, while M and N are the row and column dimensions of f.

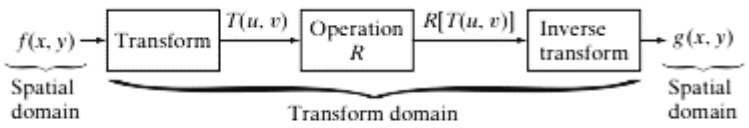


FIGURE 2.39 General approach for operating in the linear transform domain.

Variables u and v are called the *transform variables*. $T(u, v)$ is called the *forward transform* of $f(x, y)$. Given $T(u, v)$, we can recover $f(x, y)$ using the *inverse transform* of $T(u, v)$,

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u, v) s(x, y, u, v) \qquad (2.6\text{-}31)$$

Where $s(x, y, u, v)$ is called the *inverse transformation kernel*. Together, Eqs. (2.6-30) and (2.6-31) are called a *transform pair*.

Figure 2.39 shows the basic steps for performing image processing in the linear transform domain. First, the input image is transformed, the transform is then modified by a predefined operation, and, finally, the output image is obtained by computing the inverse of the modified transform. Thus, we see that the process goes from the spatial domain to the transform domain and then back to the spatial domain.

Sampling and the Fourier transform of sampled functions:

This section introduces the Fourier transform in one and two dimensions. The focus is mostly on a discrete formulation of the continuous transform and some of its properties.

The One-Dimensional Fourier Transform and its Inverse

The Fourier transform, ***F (u),*** of a single variable, continuous function, f (x), is defined by the equation

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux}\, dx$$

Where j^2 =-1 .Conversely, given ***F(u),***w e can obtain ***f(x)*** by means of the ***inverse*** Fourier transform

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux}\, du.$$

These two equations comprise the ***Fourier transform pair.*** They indicate the important fact mentioned in the previous section that a function can be recovered from its transform. These equations are easily extended to two variables, ***u*** and v:

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(ux+vy)}\, dx\, dy$$

and, similarly for the inverse transform,

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{j2\pi(ux+vy)}\, du\, dv.$$

The Fourier transform of a discrete function of one variable, ***f (x) , x*** = 0,1, 2,. . . , M - 1, is given by the equation

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) e^{-j2\pi ux/M} \qquad \text{for } u = 0, 1, 2, \ldots, M-1.$$

This ***discrete Fourier transform*** (DFT) is the foundation for most of the work in this chapter. Similarly, given ***F(u),*** we can obtain the original function back using the inverse DFT:

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{j2\pi ux/M} \qquad \text{for } x = 0, 1, 2, \ldots, M-1.$$

The ***1/M*** multiplier in front of the Fourier transform sometimes is placed in front of the inverse instead. Other times (not as often) both equations are multiplied by ***l/√M.*** The location of the multiplier does not matter. If two multipliers are used, the only requirement is that their product be equal to 1/M. considering their importance, these equations really are very simple.

In order to compute *F(u)* in Eq. *(4.2-5)* we start by substituting *u* = 0 in the exponential term and then summing for *all* values of *x.* We then substitute u = 1 in the exponential and repeat the summation over all values of *x.* We repeat this process for all *M* values sf

u in order to obtain the complete Fourier transform. It takes approximately *M~* summations and multiplications to compute the discrete Fourier transform (reduction of this number is an important topic of discussion in Section 4.6). Like f *(x),* the transform is a discrete quantity, and it has the same number of components as f *(x)* . Similar comments apply to the computation of the inverse Fourier transform.

An important property of the discrete transform pair is that, unlike the continuous case, we need not be concerned about the existence of the DFT or its inverse. The discrete Fourier transform and its inverse always exist. for digital image processing, existence of either the discrete transform OP its inverse is not an issue.

The concept of the frequency domain follows directly **from Euler's** formula:

$$e^{j\theta} = \cos\theta + j\sin\theta.$$

Substituting this expression into Eq. (4,2-5), and using the fact that cos (-θ) = cos θ, gives us

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)[\cos 2\pi ux/M - j\sin 2\pi ux/M]$$

for *u* = 0, I, 2,. . . , *M* - 1. Thus, we see that *each* term of the Fourier transform [that is, the value of *F(u)* for each value of *u]* is composed of the sum of all values of the function *f (x).The* values off (x), in turn, are multiplied by sines and cosines of various frequencies. The domain (values of *u)* over which the values of *F(u)* range is appropriately called the *frequency* domain, because *u* determines the frequency of the components of the transform. (The *x's* also affect the frequencies, but they are summed out and they all make the same contributions for each value of u.) Each of the *M* terms of *F (LIi)*s called a *frequency component* of the transform. Use of the terms frequency *domain* and *frequency components* is really no different from the terms time *domain* and *time components,* which we would use to express the domain and values off *(x)* if *x* were a time variable.

A useful analogy is to compare the Fourier transform to a glass prism. The prism is a physical device that separates light into various color components; each depending an its wavelength (or frequency) content. The Fourier transform may be viewed as a "mathematical prism that separates a function into various components, also based on frequency content. When we consider light, we talk about its spectral or frequency content. Similarly, the Fourier transform lets us characterize a function by its frequency content. This is a powerful concept that lies at the heart of linear filtering.

The components of the Fourier transform are complex quantities. As in the analysis of complex numbers, we find it convenient sometimes to express F(u) in polar coordinates:

$$F(u) = |F(u)|e^{-j\phi(u)}$$

$$|F(u)| = [R^2(u) + I^2(u)]^{1/2}$$

is called the magnitude or spectrum of the Fourier transform, and

$$\phi(u) = \tan^{-1}\left[\frac{I(u)}{R(u)}\right]$$

is called the phase angle or phase spectrum of the transform. R(u) and I(u) are the real and imaginary parts of F(u), respectively. In terms of image enhancement we are concerned primarily with properties of the spectrum.

Power spectrum, defined as the square of the Fourier spectrum:

$$P(u) = |F(u)|^2 = R^2(u) + I^2(u).$$

The term spectral density also is used to refer to the power spectrum.

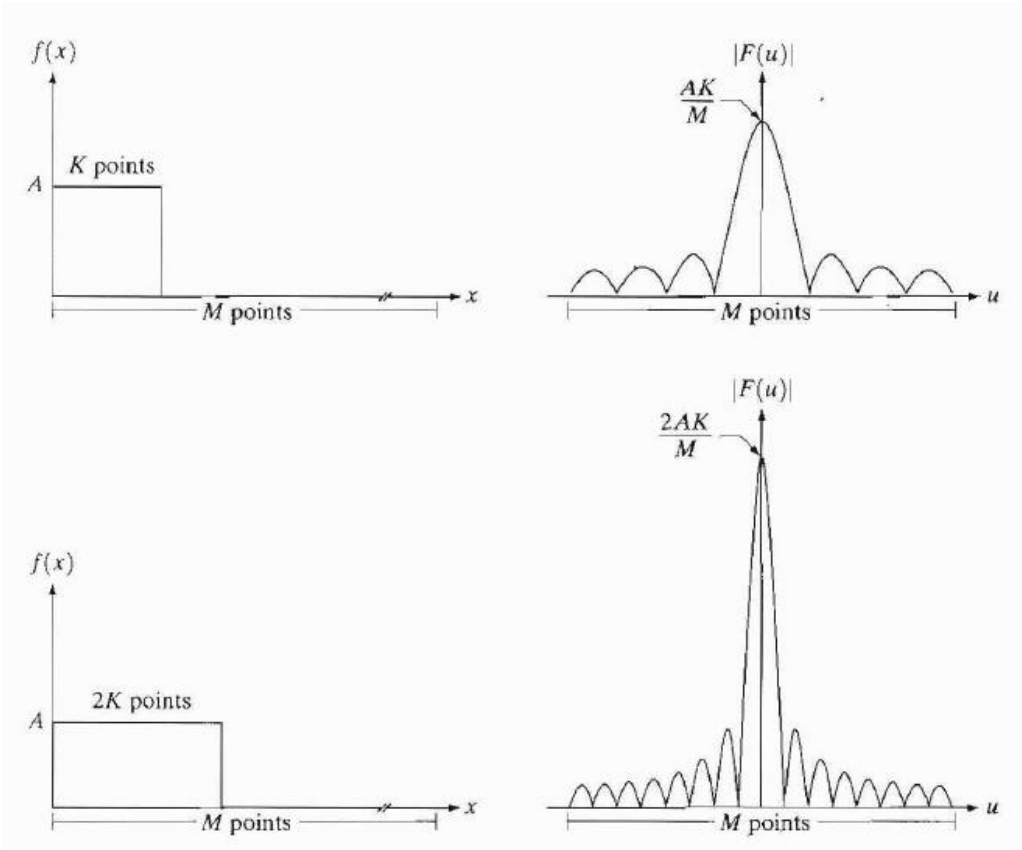
In the discrete transform, the function f (x) for ***x*** = 0,1,2,. . . , M - 1, represents M samples from its continuous counterpart. It is important to keep in mind that these samples are not necessarily always taken at integer values of x in the interval [O. M - l].They are taken at equally spaced, but otherwise arbitrary, points. This is usually represented by letting ***xo*** denote the first (arbitrarily located) point in the sequence. The first value of the sampled function is then f(xo). The next sample has taken a fixed interval Ax units away to give f (x,, + Ax). The kth sample gives us f (xu + kAx), and the final sample is f ***(x,,*** + [M - l]Ax).Thus, in the discrete case, when we write f (k), it is understood that we are utilizing shorthand notation that really means f (x, + k~x)In. terms of the notation we have used thus far, f (x) is then understood to mean

$$f(x) \triangleq f(x_0 + x\Delta x)$$

when dealing with discrete variables.7he variable *u* has a similar interpretation, but the sequence always starts at true zero frequency. Thus the sequence for the values of *u* is 0, *Au, ZAu, . . . , [M - l]Au.* Then, *F(u)* is understood to mean

$$F(u) \triangleq F(u\Delta u)$$

for u = 0,1,2,. . . , M - 1. This type of shorthand notation simplifies equations considerably and is much easier to follow.



Given the inverse relationship between a function and its transform illustrated in Fig, it is not surprising that ***Ax*** and *Au* are inversely related by the expression

$$\Delta u = \frac{1}{M\Delta x}.$$

This relationship is useful when measurements are an issue in the images being processed.

Extension to functions of two variables:

The discrete Fourier transform of a function (image) *f* (x, y***)*** of size M x N is given by the equation

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}.$$

This expression must be computed for values of ***u*** = 0, 1, 2,.. . ,M - 1, and also for *v* = *0,1,2,*. . . . N - 1. Similarly given *F(u, u)* , we obtain f ***(x , y)*** via the *inverse* Fourier transform, given by the expression

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

for x = *0,* 1, *2* ,... , *M* - 1 and y = *0,* 1,2, ... , N - 1. Both Equations comprise the *two-dimensional, discrete Fourier transform (DFT) pair. The variables u and v are the transform or frequency variables and* ***x*** *and y are the spatial or image variables. As in the one-dimensional case the location of the 1/MN constant is not important. Sometimes it is located in front of -the inverse transform. Other times it is found split into two equal terms of 1/√MN multiplying the transform and its inverse.*

We define the Fourier spectrum, phase angle. and power spectrum as

$$|F(u,v)| = [R^2(u,v) + I^2(u,v)]^{1/2}$$

$$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right]$$

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v)$$

where *R(u,v)* a nd *I(u,*v) are the real and imaginary parts of *F(u,v),*respectively.
It is common practice to multiply the input image function by *(-1)*$^{x+y}$ prior to computing the Fourier transform.

$$\Im[f(x,y)(-1)^{x+y}] = F(u - M/2, v - N/2)$$

where *ℑ[.*] denotes the Fourier transform of the argument. This equation states that the origin of the Fourier transform off *(x, y)(-l)Xty* [that is, *F(O,* O)] is located at u = M/2 and ***v*** = *N/2.* In other words, multiplying f ***(x,*** y) by (-I)'+? Shifts the origin of *F(u,v)* t o frequency coordinates *(M/2,N /2)*which is the center of the M X *N* area occupied *by* the *2-D DFT.* We refer to this area of the frequency domain as the *frequency rectangle.* It extends from *11* = 0 to ***u*** = M - 1, and from *v = 0* to *v* = N - 1 (keep in mind that *u* and *v* are integers). In order to guarantee that these shifted coordinates are integers. we require that M and N be even numbers. When implementing the Fourier transform in a computer, the limits of summations are from *u* = 1 to *M* and *v* = 1 to *N*= The actual center of the transform will then be at *u = (M/2)* + 1 and *u = (W/2)* + 1.

The value of the transform at *(u, v)* = (0,0) is,

$$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y),$$

which we see is the average off *(x , y).* In other words, if *(x, y)* is an image, the value of the Fourier transform at the origin is equal to the average gray level of the image. Because both frequencies are zero at the origin, F (0,0) sometimes is called the dc component of the spectrum. This terminology is from electrical engineering, where "dc" signifies direct current (i.e., current of zero frequency.
If *(x, y)* is real, its Fourier transform is conjugate symmetric; that is,

$$F(u,v) = F^*(-u,-v)$$

where "*" indicates the standard conjugate operation an a complex number. From this, it follows that

$$|F(u,v)| = |F(-u,-v)|,$$

which says that the spectrum of the Fourier transform is symmetric.

The relationships between samples in the spatial and frequency domains:

$$\Delta u = \frac{1}{M\Delta x}$$

$$\Delta v = \frac{1}{N\Delta y}.$$

Some Additional Properties of the 2-D Fourier Transform Translation

The Fourier transform pair has the following translation properties:

$$f(x, y)e^{j2\pi(u_0x/M + v_0y/N)} \Leftrightarrow F(u - u_0, v - v_0)$$

$$f(x - x_0, y - y_0) \Leftrightarrow F(u, v)e^{-j2\pi(ux_0/M + vy_0/N)},$$

The double arrow is used to designate a Fourier transform pair. When *uo = M/2* and *vo =* N/2, it follows that:

$$e^{j2\pi(u_0x/M + v_0y/N)} = e^{j\pi(x+y)} = (-1)^{x+y}.$$

$$f(x, y)(-1)^{x+y} \Leftrightarrow F(u - M/2, v - N/2)$$

$$f(x - M/2, y - N/2) \Leftrightarrow F(u, v)(-1)^{(u+v)}.$$

These results are based on the variables *u* and v having values in the range [0, M - 11 and [O, ***N*** - I], respectively. In a computer implementation these variables will run from *u* = 1 to *M* and v = 1 to N, in which case the actual center of the transform will be at *u* = (M/2) + 1 and *v* = *(N / 2)* + 1.

Distributivity and scaling

From the definition of the Fourier transform it follows that

$$\Im[f_1(x, y) + f_2(x, y)] = \Im[f_1(x, y)] + \Im[f_2(x, y)]$$

$$\Im[f_1(x, y) \cdot f_2(x, y)] \neq \Im[f_1(x, y)] \cdot \Im[f_2(x, y)].$$

The Fourier transform is distributive over addition, but not over multiplication. Identical comments apply to the inverse Fourier transform. Similarly, for two scalars a and b,

$$af(x, y) \Leftrightarrow aF(u, v)$$

$$f(ax, by) \Leftrightarrow \frac{1}{|ab|} F(u/a, v/b).$$

Rotation

If we introduce the polar coordinates

$$x = r\cos\theta \qquad y = r\sin\theta \qquad u = \omega\cos\varphi \qquad v = \omega\sin\varphi$$

then f *(x* ,y) and ***F(u,***v) become f (r, *θ)* and *F(w,φ***),** respectively. Direct substitution into the definition of the Fourier transform yields

$$f(r, \theta + \theta_0) \Leftrightarrow F(\omega, \varphi + \theta_0).$$

This expression indicates that rotating f (x, y) by an angle *θ0* rotates F *(u, v)* by the same angle. Similarly, rotating *F(u,v)* rotates f (x, y) by the same angle.

Periodicity and conjugate symmetry

The discrete Fourier transform has the following periodicity properties:

$$F(u, v) = F(u + M, v) = F(u, v + N) = F(u + M, v + N).$$

The inverse transform also is periodic:

$$f(x, y) = f(x + M, y) = f(x, y + N) = f(x + M, y + N).$$

The idea of conjugate symmetry is

$$F(u, v) = F^*(-u, -v)$$

from which it follows that the spectrum also is symmetric about the origin:

$$|F(u, v)| = |F(-u, -v)|.$$

Separability

The discrete Fourier transform can be expressed in the separable form

$$F(u, v) = \frac{1}{M}\sum_{x=0}^{M-1} e^{-j2\pi ux/M} \frac{1}{N}\sum_{y=0}^{N-1} f(x, y)e^{-j2\pi vy/N}$$

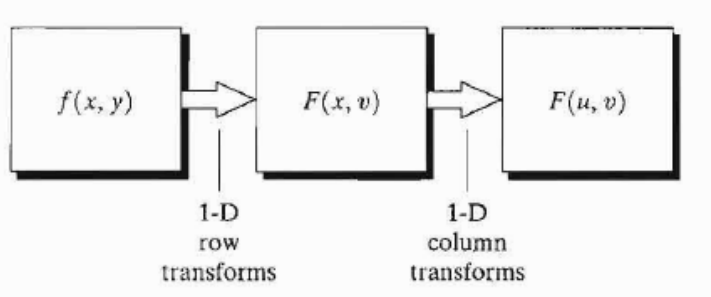
$$= \frac{1}{M}\sum_{x=0}^{M-1} F(x, v)e^{-j2\pi ux/M}$$

$$F(x, v) = \frac{1}{N}\sum_{y=0}^{N-1} f(x, y)e^{-j2\pi vy/N}.$$

For ***each*** value of x, and for values of v = 0,1,2,. . . , N - 1, this equation is a complete 1-D Fourier transform. In other words, ***F(x, v)*** is the Fourier transform along one row off (x, y). By varying ***x*** from **0** to M - I, we compute the Fourier transform along all rows off ***(x ,*** y). Thus far, the frequency variable u has remained constant. To complete the 2-D transform we have to vary u from 0 to M - 1 in Eq. After a little thought, it becomes evident that this involves computing the 1-D transform along each column of ***F(x,*** v).This is an important result. It tells us that we can compute the 2-D transform by first computing a 1-D transform along each row of the input image, and then computing a 1-D transform along each column of this intermediate result. The same comments hold if we reverse the

order of computation: columns first, followed by rows.

A similar development applies to computing the 2-D inverse Fourier transform. We first compute 1-D inverse transforms along each row of ***F(u.*** v) and then compute inverse 1-D transforms along each column of the intermediate result. As shown in the following section, it is possible to implement the inverse transform using a 1-D ,forward Fourier transform algorithm.



Computing the Inverse Fourier Transform Using a Forward Transform Algorithm

2-D Fourier transforms can be computed via the application of 1-D transforms. The 1-D Fourier transform pair was defined

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-j2\pi ux/M}$$

for $u = 0, 1, 2, \ldots, M-1$, and

$$f(x) = \sum_{u=0}^{M-1} F(u)e^{j2\pi ux/M}$$

for **x** = 0,1,2,. . . , M - 1.Taking the complex conjugate of Eq. and dividing both sides by M yields

$$\frac{1}{M} f^*(x) = \frac{1}{M}\sum_{u=0}^{M-1} F^*(u)e^{-j2\pi ux/M}.$$

Therefore, inputting ***F*(u) i***n to an algorithm designed to compute the forward transform gives the quantity f*(x)/M. Taking the complex conjugate and multiplying by M yields the desired inverse f ***(x) .*** A similar analysis for two variables yields:

$$\frac{1}{MN} f^*(x, y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F^*(u, v)e^{-j2\pi(ux/M+vy/N)}$$

which is in the form of a 2-D forward Fourier transform. If f ***(x) or*** *f* **(x,** *y)* are real functions (e.g., an image), then the complex conjugate on the left of Eq. is unnecessary; we simply take the real part of the result, ignoring the parasitic complex terms that are typical in most Fourier transform computations.

Computation of the 2-D transform by successive passes of the 1-D transform is a frequent source of confusion when the technique we have just developed is used to obtain the inverse. In other words, when a 1-D algorithm is used to compute the 2-D inverse, we do not compute the complex conjugate after processing each row or column. Instead, the

function $F^*(u, v)$ is treated as if it were f ***(x , y)*** in the forward, 2-D transform procedure summarized in Fig. 4.35. The complex conjugate (or real part, if applicable) of the result, multiplied by MN, yields the proper inverse f ***(x , y).*** We emphasize that the preceding comments regarding the constants M and N are based on the definition of the discrete Fourier transform that has all constants associated with the forward transform.

More on Periodicity: the Need for Padding

based on the convolution theorem, multiplication in the frequency domain is equivalent to convolution in the spatial domain, and vice versa. When working with discrete variables and the Fourier transform, we need to keep in mind the periodicity of the various functions involved. Although it may not be intuitive, this periodicity is a mathematical byproduct of the way in which the discrete Fourier transform pair is defined. Periodicity is part of the process, and it cannot be ignored.

$$f(x) * h(x) = \frac{1}{M} \sum_{m=0}^{M-1} f(m)h(x - m).$$

To simplify the notation, simple numbers instead of general symbols are used for the height and length of the functions. Figures (a) and (b) show the two functions we wish to convolve. Each function consists of 400 points. The first step in convolution is to mirror (flip) one of the functions about the origin. In this case this was done to the second function. Which is shown as *h(-m)*in Fig.(c).The next step is to "slide" *h(-m)* past f(m).This is done by adding a constant,^, to h(-m); that is, we form h(x - m), as shown in Fig.(d). Note that this is only *one* displacement value. This simple step is a frequent source of confusion when first encountered. It helps to remember that this is precisely what convolution is all about. In other words! to perform convolution we flip one oi the functions and slide it past the other. At each displacement (each value of x) the entire summation in Eq is carried out. This summation is nothing more than the sum of products off and ***h*** at a given displacement. The displacement x ranges over all values required to completely slide *h* past *f.* Figure (e) shows the result of completely sliding *h* past f and computing Eq. at each value of ***x.*** In this case ***x*** had to range for 0 to 799 for ***h(x*** - *m)* to slide completely past f. This figure is the convolution of the two functions. Keep clearly in mind that the variable in convolution is x.

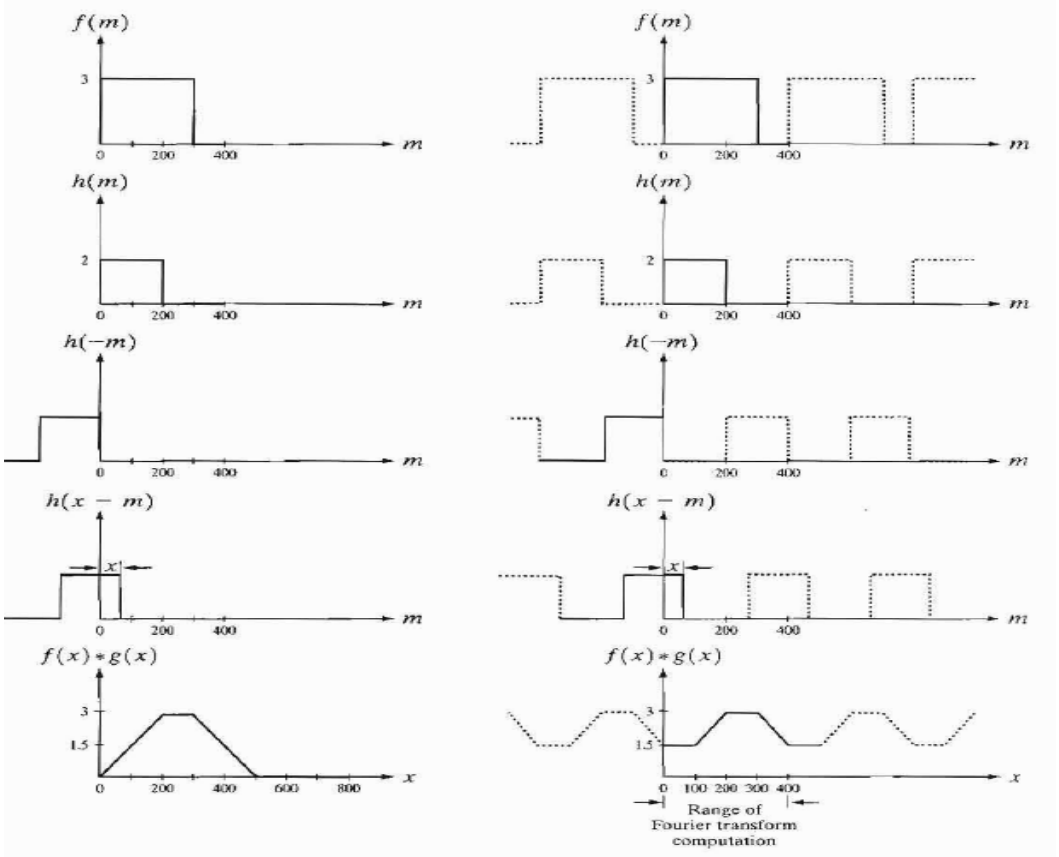
Using the DFT allows us to perform convolution in the frequency domain, but the functions are treated as periodic, with a period equal to the length of the functions.

In the frequency domain the procedure would be to compute the Fourier transforms of the functions in Figs.(a) and (b). According to the convolution theorem, the two transforms would then be multiplied and the inverse Fourier transform taken. The result would be the 400 points comprising the convolution shown in solid in Fig (j).This simple illustration shows that failure to handle the periodicity issue properly will give incorrect results if the convolution function is obtained using the Fourier transform. The result will have erroneous data at the beginning and have missing data at the end.

The solution to this problem is straightforward. Assume that f and h consist of ***A*** and *B* points, respectively. We append zeros to both functions so that they have identical periods, denoted by P. This procedure yields extended, or padded, functions given by

$$f_e(x) = \begin{cases} f(x) & 0 \le x \le A - 1 \\ 0 & A \le x \le P \end{cases}$$

$$g_e(x) = \begin{cases} g(x) & 0 \le x \le B - 1 \\ 0 & B \le x \le P. \end{cases}$$

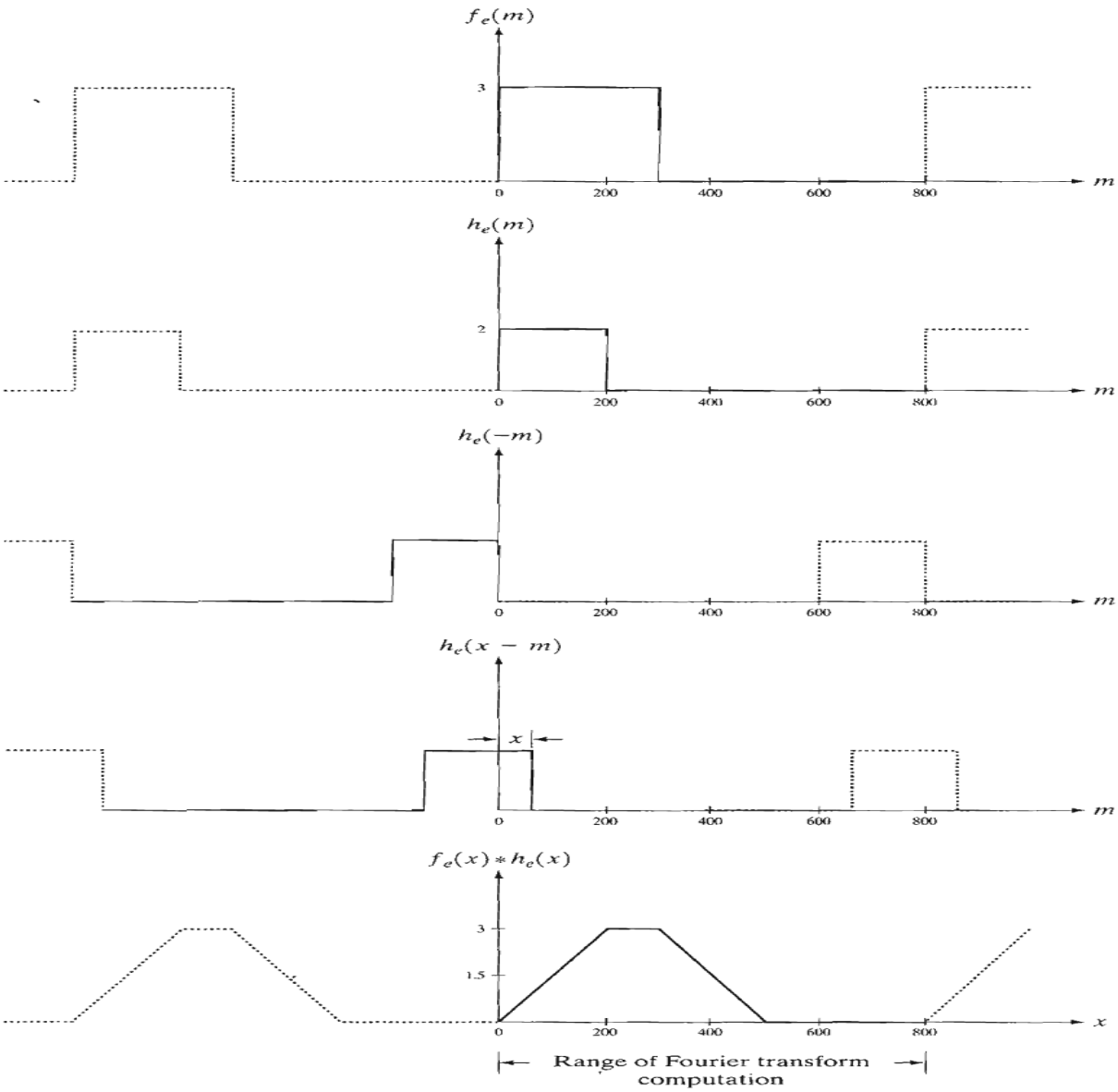


It can be show that, unless we choose P ***A*** + B - 1, the individual periods of the convolution will overlap. We already saw in Fig. the result of this phenomenon, which is commonly referred to as wrap around error. If ***P*** = A + *B* - 1. The periods will be adjacent. If P > *A* + *B* - 1, the periods will be separated, with the degree of separation being equal to the difference between P and ***A*** + B - 1.

Extension of these concepts to 2-D functions follows the same line of reasoning. Suppose that we have two images f ***(x,*** y) and ***h(x.*** y) of sizes A X Band C X D, 1.espectively. As in the 1 -D case, these arrays must be assumed periodic with some period P in the x-direction and Q in the y-direction. Wraparound error in 2-D convolution is avoided by choosing

$$P \geq A + C - 1$$

$$Q \geq B + D - 1.$$



The periodic **sequences** are f6rrned by extending f ***(x , y)*** and ***h(x,*** *y)* as follows:

$$f_e(x, y) = \begin{cases} f(x, y) & 0 \le x \le A - 1 \quad \text{and} \quad 0 \le y \le B - 1 \\ 0 & A \le x \le P \quad \text{or} \quad B \le y \le Q \end{cases}$$

$$h_e(x, y) = \begin{cases} h(x, y) & 0 \le x \le C - 1 \quad \text{and} \quad 0 \le y \le D - 1 \\ 0 & C \le x \le P \quad \text{or} \quad D \le y \le Q \end{cases}$$

The issue of padding is central to filtering. When **we** implement any of the frequency domain filters discussed in this chapter, we do it by multiplying the filter transfer function by the transform of the **image** we wish to process. By the convolution theorem, we know that this is the same as convolving the spatial representation of the filter with the image. Thus, unless **proper** padding is implemented, the results **will** be erroneous.

The Convolution and Correlation Theorems

The discrete convolution of two functions ***f(x,*** *y)* and ***h(x,*** *y)* of size *M* x N is denoted by *f (x , y)* * ***h(x.*** *y)* and is defined by the expression

$$f(x, y) * h(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x - m, y - n).$$

The convolution theorem consists of the following relationships between the two functions and their Fourier transforms:

$$f(x, y) * h(x, y) \Leftrightarrow F(u, v) H(u, v)$$

$$f(x, y) h(x, y) \Leftrightarrow F(u, v) * H(u, v).$$

The correlation of two functions f *(x,* y) and ***h(x,y)*** is defined as

$$f(x, y) \circ h(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f^*(m, n) h(x + m, y + n)$$

where f * denotes the complex conjugate off. We normally deal with real functions (images), in which case *f* * = f. The correlation function has exactly the same *form* as the convolution function with the exception of the complex conjugate and the fact that the second-term in the summation has positive instead of negative signs. This means that h is not mirrored about the origin. Everything else in the implementation of correlation is identical to convolution, including the need for padding.

Given the similarity of convolution and correlation, it is not surprising that there is a *correlation theorem,* analogous to the convolution theorem. Let *F(u, v)* and ***H(u,v)*** denote the Fourier transforms of f*(x* ,y) and ***h(x ,***y), respectively. One-half of the correlation

theorem states that spatial correlation, f *(x* ,y) ***0*** h(x, y) , and the frequency domain product, ***F * (u, v)*** *H(u,* ***v),*** constitute a Fourier transform pair. This result, formally stated as

$$f(x, y) \circ h(x, y) \Leftrightarrow F^*(u, v)H(u, v)$$

indicates that correlation in the spatial domain can be obtained by taking the inverse Fourier transform of the product ***F*(u,*** *v) H(u,* v), where *F** is the complex conjugate of F. **An** analogous result is that correlation in the frequency domain reduces to multiplication in the spatial domain: that is,

$$f^*(x, y)h(x, y) \Leftrightarrow F(u, v) \circ H(u, v)$$

These two results comprise the correlation theorem. It is assumed that all functions have been properly extended by padding.

As we know by now, convolution is the tie between filtering in the spatial and frequency domains. The principal use of correlation i s for matching. In ***matching,*** f ***(x ,*** y) is an image containing objects or regions. If we want to determine whether f contains a particular object or region in which we are interested, we let h(x, y) be that object or region (we normally call this image a ***template).*** Then, if there is a match, the correlation of the two functions will be maximum at the location where *h* finds a correspondence in *f.* preprocessing, like scaling and alignment is necessary in most practical applications, but the bulk of the process is performing the correlation.

Finally, **we** point out that the term *cross correlation* often is used in place of the term correlation to clarify that the images being correlated are different. This is as opposed to *auto correlation* in, which both images are identical. In the latter case, *we* have the *autocorrelation theorem*

$$f(x, y) \circ f(x, y) \Leftrightarrow |F(u, v)|^2$$

On the right side, we used the fact that the product of a complex quantity and its complex conjugate is the magnitude of the complex quantity squared. In words, this result states that the Fourier transform of the spatial autocorrelation is the power spectrum defined

$$|f(x, y)|^2 \Leftrightarrow F(u, v) \circ F(u, v)$$

Summary of Properties of the 2-D Fourier Transform

Property	Expression(s)
Fourier transform	$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)}$
Inverse Fourier transform	$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M+vy/N)}$
Polar representation	$F(u,v) = \lvert F(u,v)\rvert e^{-j\phi(u,v)}$
Spectrum	$\lvert F(u,v)\rvert = [R^2(u,v) + I^2(u,v)]^{1/2}$, $R = \text{Real}(F)$ and $I = \text{Imag}(F)$
Phase angle	$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right]$
Power spectrum	$P(u,v) = \lvert F(u,v)\rvert^2$
Average value	$\bar{f}(x,y) = F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$
Translation	$f(x,y)e^{j2\pi(u_0x/M+v_0y/N)} \Leftrightarrow F(u-u_0, v-v_0)$ $f(x-x_0, y-y_0) \Leftrightarrow F(u,v)e^{-j2\pi(ux_0/M+vy_0/N)}$ When $x_0 = u_0 = M/2$ and $y_0 = v_0 = N/2$, then $f(x,y)(-1)^{x+y} \Leftrightarrow F(u-M/2, v-N/2)$ $f(x-M/2, y-N/2) \Leftrightarrow F(u,v)(-1)^{u+v}$
Conjugate symmetry	$F(u,v) = F^*(-u,-v)$ $\lvert F(u,v)\rvert = \lvert F(-u,-v)\rvert$
Differentiation	$\frac{\partial^n f(x,y)}{\partial x^n} \Leftrightarrow (ju)^n F(u,v)$ $(-jx)^n f(x,y) \Leftrightarrow \frac{\partial^n F(u,v)}{\partial u^n}$
Laplacian	$\nabla^2 f(x,y) \Leftrightarrow -(u^2+v^2)F(u,v)$
Distributivity	$\Im[f_1(x,y) + f_2(x,y)] = \Im[f_1(x,y)] + \Im[f_2(x,y)]$ $\Im[f_1(x,y) \cdot f_2(x,y)] \neq \Im[f_1(x,y)] \cdot \Im[f_2(x,y)]$
Scaling	$af(x,y) \Leftrightarrow aF(u,v)$, $f(ax,by) \Leftrightarrow \frac{1}{\lvert ab\rvert}F(u/a, v/b)$
Rotation	$x = r\cos\theta$ $y = r\sin\theta$ $u = \omega\cos\varphi$ $v = \omega\sin\varphi$ $f(r, \theta+\theta_0) \Leftrightarrow F(\omega, \varphi+\theta_0)$
Periodicity	$F(u,v) = F(u+M,v) = F(u,v+N) = F(u+M,v+N)$ $f(x,y) = f(x+M,y) = f(x,y+N) = f(x+M,y+N)$
Separability	See Eqs. (4.6-14) and (4.6-15). Separability implies that we can compute the 2-D transform of an image by first computing 1-D transforms along each row of the image, and then computing a 1-D transform along each column of this intermediate result. The reverse, columns and then rows, yields the same result.

Property	Expression(s)
Computation of the inverse Fourier transform using a forward transform algorithm	$\frac{1}{MN} f^*(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F^*(u, v) e^{-j2\pi(ux/M + vy/N)}$ This equation indicates that inputting the function $F^*(u, v)$ into an algorithm designed to compute the forward transform (right side of the preceding equation) yields $f^*(x, y)/MN$. Taking the complex conjugate and multiplying this result by MN gives the desired inverse.
Convolution†	$f(x, y) * h(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x - m, y - n)$
Correlation†	$f(x, y) \circ h(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f^*(m, n) h(x + m, y + n)$
Convolution theorem†	$f(x, y) * h(x, y) \Leftrightarrow F(u, v) H(u, v)$; $f(x, y) h(x, y) \Leftrightarrow F(u, v) * H(u, v)$
Correlation theorem†	$f(x, y) \circ h(x, y) \Leftrightarrow F^*(u, v) H(u, v)$; $f^*(x, y) h(x, y) \Leftrightarrow F(u, v) \circ H(u, v)$
Some useful FT pairs:	
Impulse	$\delta(x, y) \Leftrightarrow 1$
Gaussian	$A2\pi\sigma^2 e^{-2\pi^2\sigma^2(x^2 + y^2)} \Leftrightarrow Ae^{-(u^2 + v^2)/2\sigma^2}$
Rectangle	$\text{rect}[a, b] \Leftrightarrow ab \frac{\sin(\pi ua)}{(\pi ua)} \frac{\sin(\pi vb)}{(\pi vb)} e^{-j\pi(ua + vb)}$
Cosine	$\cos(2\pi u_0 x + 2\pi v_0 y) \Leftrightarrow \frac{1}{2}[\delta(u + Mu_0, v + Nv_0) + \delta(u - Mu_0, v - Nv_0)]$
Sine	$\sin(2\pi u_0 x + 2\pi v_0 y) \Leftrightarrow j\frac{1}{2}[\delta(u + Mu_0, v + Nv_0) - \delta(u - Mu_0, v - Nv_0)]$

Filtering in the Frequency Domain

The frequency domain is nothing more than the space defined by values of the Fourier transform and its frequency variables *(u,v)*.

Some basic properties of the frequency domain

We start by observing in Eq that *each* term of *F(u,v)* contains all values of f (x, y), modified by the values of the exponential terms. Thus, with the exception of trivial cases, it usually is impossible to make direct associations between specific components of an image and its transform.

since frequency is directly related to rate of change, it is not difficult intuitively to associate frequencies in the Fourier transform with patterns of intensity variations in an image.

The slowest varying frequency component *(u* = v = 0) corresponds to the average gray level of an image. As we move away from the origin of the transform, the low frequencies correspond to the slowly varying components of an image.

As we move further away from the origin, the higher- frequencies begin to correspond to faster and faster gray level changes in the image. These are the edges of objects and other components of an image characterized by abrupt changes in gray level, such as noise.

Introduction to Fourier Transform

The forward transformation kernel is said to be *separable* if

$$r(x, y, u, v) = r_1(x, u)r_2(y, v) \qquad (2.6\text{-}32)$$

In addition, the kernel is said to be *symmetric* if $r_1(x, y)$ is functionally equal to $r_2(x, y)$, so that

$$r(x, y, u, v) = r_1(x, u)r_1(y, v) \qquad (2.6\text{-}33)$$

Identical comments apply to the inverse kernel by replacing *r* with *s* in the pre- ceding equations.

The 2-D Fourier transform discussed in Example 2.11 has the following for- ward and inverse kernels:

$$r(x, y, u, v) = e^{-j2\pi(ux/M + vy/N)}$$

And

$$s(x,y,u,v) = \frac{1}{MN} e^{j2\pi(ux/M+vy/N)}$$

Respectively, where $j = \sqrt{-1}$, so these kernels are complex. Substituting these kernels into the general transform formulations in Eqs. (2.6-30) and (2.6-31) gives us the *discrete Fourier transform pair*:

$$T(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)}$$

And

$$f(x,y) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} T(u,v) e^{j2\pi(ux/M+vy/N)}$$

These equations are of fundamental importance in digital image processing, and we devote most of Chapter 4 to deriving them starting from basic principles and then using them in a broad range of applications.

It is not difficult to show that the Fourier kernels are separable and symmetric (Problem 2.25), and that separable and symmetric kernels allow 2-D transforms to be computed using 1-D transforms (Problem 2.26). When the forward and inverse kernels of a transform pair satisfy these two conditions, and $f(x, y)$ is a square image of size $M * M$, Eqs. (2.6-30) and (2.6-31) can be expressed in matrix form:

$$\mathbf{T} = \mathbf{AFA} \qquad (2.6\text{-}38)$$

where **F** is an $M * M$ matrix containing the elements of $f(x, y)$ [see Eq. (2.4-2)], **A** is an $M * M$ matrix with elements $a_{ij} = r_1(i, j)$, and **T** is the resulting $M * M$ transform, with values $T(u, v)$ for $u, v = 0, 1, 2, \text{Á}, M - 1$.

To obtain the inverse transform, we pre- and post-multiply Eq. (2.6-38) by an inverse transformation matrix **B**:

$$\mathbf{BTB} = \mathbf{BAFAB} \qquad (2.6\text{-}39)$$

If $\mathbf{B} = \mathbf{A}^{-1}$,

$$\mathbf{F} = \mathbf{BTB} \qquad (2.6\text{-}40)$$

indicating that **F** [whose elements are equal to image $f(x, y)$] can be recov- ered completely from its forward transform. If **B** is not equal to $\mathbf{A}^{-1}$, then use of Eq. (2.6-40) yields an approximation.

$$\mathbf{F} = \mathbf{BAFAB} \quad (2.6\text{-}41)$$

***) Fourier Transform and its inverse.**

Let f(x) be a continuous function of a real variable x. The Fourier transform of f(x) is defined by the equation

$$\Im\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x)\exp[-j2\pi ux]\,dx$$

Where $j = \sqrt{-1}$

Given F(u), f(x) can be obtained by using the inverse Fourier transform

$$\Im^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{\infty} F(u)\exp[j2\pi ux]\,du.$$

The Fourier transform exists if f(x) is continuous and integrable and F(u) is integrable.
The Fourier transform of a real function, is generally complex,

$$F(u) = R(u) + jI(u)$$

Where R(u) and I(u) are the real and imiginary components of F(u). F(u) can be expressed in exponential form as

$$F(u) = |F(u)|\, e^{j\emptyset(u)}$$

where

$|F(u)| = [R^2(u) + I^2(u)]^{1/2}$ and $\emptyset(u, v) = \tan^{-1}[I(u, v)/R(u, v)]$

The magnitude function |F (u)| is called the Fourier Spectrum of f(x) and Φ(u) its phase angle. The variable u appearing in the Fourier transform is called the frequency variable.

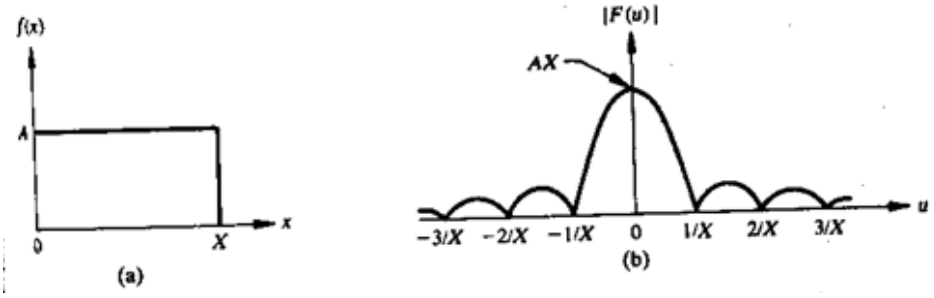


Fig 1 A simple function and its Fourier spectrum

The Fourier transform can be easily extended to a function f(x, y) of two variables**.** If f(x, y) is continuous and integrable and F(u,v) is integrable, following Fourier transform pair exists

$$\Im\{f(x, y)\} = F(u, v) = \int_{-\infty}^{\infty}\int f(x, y)\exp[-j2\pi(ux + vy)]\,dx\,dy$$

and

$$\Im^{-1}\{F(u, v)\} = f(x, y) = \int_{-\infty}^{\infty}\int F(u, v)\exp[j2\pi(ux + vy)]\,du\,dv$$

\

Where u, v are the frequency variables The Fourier spectrum, phase, are

$$|\,F(u, v)\,| = [R^2(u, v) + I^2(u, v)]^{1/2}$$

$$\text{Ø}(u, v) = \tan^{-1}[\,I(u, v)/R(u, v)\,]$$

Discrete Fourier transform and its inverse.

The discrete Fourier transform pair that applies to sampled function is given by,

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\exp[-j2\pi ux/N]$$

(1) For u = 0, 1, 2 . . . . , N-1, and

$$f(x) = \sum_{u=0}^{N-1} F(u)\exp[j2\pi ux/N]$$

(2)

For x = 0, 1, 2 . . . ., N-1.

In the two variable case the discrete Fourier transform pair is

$$F(u, v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x, y)\exp[-j2\pi(ux/M + vy/N)]$$

For u = 0, 1, 2 . . . , M-1, v = 0, 1, 2 . . . , N - 1, and

$$f(x, y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u, v)\exp[j2\pi(ux/M + vy/N)]$$

For x = 0, 1, 2 . . . , M-1, y = 0, 1, 2 . . . , N-1.

If M = N, then discrete Fourier transform pair is

$$F(u, v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x, y)\exp[-j2\pi(ux + vy)/N]$$

For u, v = 0, 1, 2 . . . , N – 1, and

$$f(x, y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u, v)\exp[j2\pi(ux + vy)/N]$$

For x, y = 0, 1, 2 . . . , N – 1

Fast Fourier Transform

Fourier Transform decomposes an image into its real and imaginary components which is a representation of the image in the frequency domain. If the input signal is an image then the number of frequencies in the frequency domain is equal to the number of pixels in the image or spatial domain. The inverse transform re-transforms the frequencies to the image in the spatial domain. The FFT and its inverse of a 2D image are given by the following equations:

$$F(x) = \sum_{n=0}^{N-1} f(n) e^{-j2\pi(x\frac{n}{N})}$$

$$f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x) e^{j2\pi(x\frac{n}{N})}$$

Where f(m,n) is the pixel at coordinates (m, n), F(x,y) is the value of the image in the frequency domain corresponding to the coordinates x and y, M and N are the dimensions of the image.

As can be seen from the equation, a naïve implementation of this algorithm is very expensive. But the beauty of FFT is that it is separable, namely, the 2D transform can be done as 2 1D transforms as shown below (shown only in the horizontal direction) - one in the horizontal direction followed by the other in the vertical direction on the result of the horizontal transform. The end result is equivalent to performing the 2D transform in the frequency space.

$$F(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

$$f(m,n) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(x,y) e^{j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

The FFT that's implemented in the application here requires that the dimensions of the image are a power of two. Another interesting property of the FFT is that the transform of N points can be rewritten as the sum of two N/2 transforms (divide and conquer). This is important because some of the computations can be reused thus eliminating expensive operations.

The output of the Fourier Transform is a complex number and has a much greater range than the image in the spatial domain. Therefore to accurately represent these values, they are stored as floats. Furthermore, the dynamic range of the Fourier coefficients are too large to be displayed on the screen, and hence, these values are scaled (usually by dividing by Height*Width of the image) to bring them within the range of values that can be displayed [3].

Properties of Fourier transform

separability property of 2D-DFT.

The separability property of 2D-DFT states that, the discrete Fourier transform pair can be expressed in the separable forms. i.e. ,

$$F(u, v) = \frac{1}{N}\sum_{x=0}^{N-1} \exp[-j2\pi ux/N] \sum_{y=0}^{N-1} f(x, y)\exp[-j2\pi vy/N] \quad (1)$$

For u, v = 0, 1, 2 . . . , N – 1, and

$$f(x, y) = \frac{1}{N}\sum_{u=0}^{N-1} \exp[j2\pi ux/N] \sum_{v=0}^{N-1} F(u, v)\exp[j2\pi vy/N] \quad (2)$$

For x, y = 0, 1, 2 . . . , N – 1

The principal advantage of the separability property is that F(u,v) or f(x,y) can be obtained in two steps by successive applications of the 1-D Fourier transform or its inverse. This advantage becomes evident if equation (1) is expressed in the form

$$F(u, v) = \frac{1}{N}\sum_{x=0}^{N-1} F(x, v)\exp[-j2\pi ux/N] \quad (3)$$

Where,

$$F(x, v) = N\left[\frac{1}{N}\sum_{y=0}^{N-1} f(x, y)\exp[-j2\pi vy/N]\right]. \quad (4)$$

For each value of x, the expression inside the brackets in eq(4) is a 1-D transform, with frequency values v = 0, 1, . . . , N-1. Therefore the 2-D function f(x, v) is obtained by taking a transform along each row of f(x, y) and multiplying the result by N. The desired result, F(u, v), is then obtained by taking a transform along each column of F(x, v), as indicated by eq(3)

(d)**the translation property.**

The translation properties of the Fourier transform pair are

$$f(x, y)\exp[j2\pi(u_0x + v_0y)/N] \Leftrightarrow F(u - u_0, v - v_0) \quad (1)$$

and

$$f(x - x_0, y - y_0) \Leftrightarrow F(u, v)\exp[-j2\pi(ux_0 + vy_0)/N] \quad (2)$$

Where the double arrow indicates the correspondence between a function and its Fourier Transform,

Equation (1) shows that multiplying f(x, y) by the indicated exponential term and taking the transform of the product results in a shift of the origin of the frequency plane to the point (u_o, v_o).

Consider the equation (1) with $u_o = v_o = N/2$ or

$\exp[j2\Pi(u_o x + v_o y)/N] = e^{j\Pi(x + y)}$

$= (-1)^{(x + y)}$

and

$$f(x, y)(-1)^{x+y} \# F(u - N/2, v - N/2)$$

Thus the origin of the Fourier transform of f(x, y) can be moved to the center of its corresponding N x N frequency square simply by multiplying f(x, y) by $(-1)^{x+y}$. In the one variable case this shift reduces to multiplication of f(x) by the term $(-1)^x$. Note from equation (2) that a shift in f(x, y) does not affect the magnitude of its Fourier transform as,

$$|F(u, v)\exp[-j2\pi(ux_0 + vy_0)/N]| = |F(u, v)|.$$

(iii) **distributivity and scaling property.**

Distributivity:

From the definition of the continuous or discrete transform pair,

$$\Im\{f_1(x, y) + f_2(x, y)\} = \Im\{f_1(x, y)\} + \Im\{f_2(x, y)\}$$

and, in general,

$$\Im\{f_1(x, y) \cdot f_2(x, y)\} \neq \Im\{f_1(x, y)\} \cdot \Im\{f_2(x, y)\}.$$

In other words, the Fourier transform and its inverse are distributive over addition but not over multiplication.

Scaling:

For two scalars a and b,

$$af(x, y) \# aF(u, v)$$

$$f(ax, by) \Leftrightarrow \frac{1}{|ab|} F(u/a, v/b).$$

Walsh transform.

The discrete Walsh transform of a function $f(x)$, denoted $W(u)$, is given by

$$W(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) \prod_{i=0}^{n-1} (-1)^{b_i(x)b_{n-1-i}(u)}$$

Walsh transform kernel is symmetric matrix having orthogonal rows and columns. These properties, which hold in general, lead to an inverse kernel given by

$$h(x, u) = \prod_{i=0}^{n-1} (-1)^{b_i(x)b_{n-1-i}(u)}.$$

Thus the inverse Walsh transform is given by

$$f(x) = \sum_{u=0}^{N-1} W(u) \prod_{i=0}^{n-1} (-1)^{b_i(x)b_{n-1-i}(u)}.$$

The 2-D forward and inverse Walsh kernels are given by

$$g(x, y, u, v) = \frac{1}{N} \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]}$$

and

$$h(x, y, u, v) = \frac{1}{N} \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]}.$$

Thus the forward and inverse Walsh transforms for 2-D are given by

$$W(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]}$$

and

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} W(u, v) \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]}$$

The Walsh Transform kernels are seperable and symmetric, because

$$\begin{aligned} g(x, y, u, v) &= g_1(x, u)g_1(y, v) \\ &= h_1(x, u)h_1(y, v) \\ &= \left[\frac{1}{\sqrt{N}} \prod_{i=0}^{n-1} (-1)^{b_i(x)b_{n-1-i}(u)}\right]\left[\frac{1}{\sqrt{N}} \prod_{i=0}^{n-1} (-1)^{b_i(y)b_{n-1-i}(v)}\right] \end{aligned}$$

Values of the 1-D walsh transform kernel for N = 8 is

u \ x	0	1	2	3	4	5	6	7
0	+	+	+	+	+	+	+	+
1	+	+	+	+	−	−	−	−
2	+	+	−	−	+	+	−	−
3	+	+	−	−	−	−	+	+
4	+	−	+	−	+	−	+	−
5	+	−	+	−	−	+	−	+
6	+	−	−	+	+	−	−	+
7	+	−	−	+	−	+	+	−

Hadamard transform.

1-D forward kernel for hadamard transform is

$$g(x, u) = \frac{1}{N} (-1)^{\sum_{i=0}^{n-1} b_i(x)b_i(u)}$$

Expression for the 1-D forward Hadamard transform is

$$H(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x)(-1)^{\sum_{i=0}^{n-1} b_i(x)b_i(u)}$$

Where $N = 2^n$ and u has values in the range 0, 1, . . . , N-1.

1-D inverse kernel for hadamard transform is

$$h(x, u) = (-1)^{\sum_{i=0}^{n-1} b_i(x)b_i(u)}$$

Expression for the 1-D inverse Hadamard transform is

$$f(x) = \sum_{u=0}^{N-1} H(u)(-1)^{\sum_{i=0}^{n-1} b_i(x)b_i(u)}$$

The 2-D kernels are given by the relations

$$g(x, y, u, v) = \frac{1}{N}(-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]}$$

and

$$h(x, y, u, v) = \frac{1}{N}(-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]}$$

2-D Hadamard transform pair is given by following equations

$$f(x, y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} H(u, v)(-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]}$$

$$H(u, v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x, y)(-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]}$$

Values of the 1-D hadamard transform kernel for N = 8 is

u \ x	0	1	2	3	4	5	6	7
0	+	+	+	+	+	+	+	+
1	+	−	+	−	+	−	+	−
2	+	+	−	−	+	+	−	−
3	+	−	−	+	+	−	−	+
4	+	+	+	+	−	−	−	−
5	+	−	+	−	−	+	−	+
6	+	+	−	−	−	−	+	+
7	+	−	−	+	−	+	+	−

The Hadamard matrix of lowest order N = 2 is

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

If $\mathbf{H_N}$ represents the matrix of order N, the recursive relationship is given by

$$\mathbf{H}_{2N} = \begin{bmatrix} \mathbf{H}_N & \mathbf{H}_N \\ \mathbf{H}_N & -\mathbf{H}_N \end{bmatrix}$$

Where $\mathbf{H}_{2N}$ is the Hadamard matrix of order 2N and $N = 2^n$

Haar transform.

The Haar transform is based on the Haar functions, $h_k(z)$, which are defined over the continuous, closed interval $z \varepsilon [0, 1]$, and for k = 0, 1, 2 . . . , N-1, where $N = 2^n$. The first step in generating the Haar transform is to note that the integer k can be decomposed uniquely as

$$k = 2^p + q - 1$$

where $0 \le p \le n-1$, q = 0 or 1 for p = 0, and $1 \le q \le 2^p$ for $p \ne 0$. For example, if N = 4, k, q, p have following values

k	p	q
0	0	0
1	0	1
2	1	1
3	1	2

The Haar functions are defined as

$$h_0(z) \triangleq h_{00}(z) = \frac{1}{\sqrt{N}}$$

for $z \varepsilon [0, 1]$ ……

and

$$h_k(z) \triangleq h_{pq}(z) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2} & \dfrac{q-1}{2^p} \le z < \dfrac{q-1/2}{2^p} \\ -2^{p/2} & \dfrac{q-1/2}{2^p} \le z < \dfrac{q}{2^p} \\ 0 & \text{otherwise for } z \in [0, 1]. \end{cases}$$

These results allow derivation of Haar transformation matrices of order N x N by formation of the *i*th row of a Haar matrix from elements oh $h_i(z)$ for z = 0/N, 1/N, . . . , (N-1)/N. For instance, when N = 2, the first row of the 2 x 2 Haar matrix is computed by using $h_O(z)$ with z = 0/2, 1/2. From equation (1) , $h_O(z)$ is equal to $\sqrt{2}$, independent of z, so the first row of the matrix has two identical $\sqrt{2}$ elements. Similarly row is computed. The 2 x 2 Haar matrix is

$$\mathbf{A}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Similarly matrix for N = 4 is

$$\mathbf{A}_4 = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

Properties of Haar transform:

(a) The Haar transform is real and orthogonal.

(b) The Haar transform is very fast. It can implement O(n) operations on an N x 1 vector.

(c) The mean vectors of the Haar matrix are sequentially ordered.

(d) It has a poor energy deal for images.

Slant transform.

The Slant transform matrix of order N x N is the recursive expression S_n is given by

$$= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ a_N & b_N & & -a_N & b_N & \\ 0 & \mathbf{I}_{(N/2)-2} & 0 & 0 & \mathbf{I}_{(N/2)-2} & \\ 0 & 1 & 0 & 0 & -1 & 0 \\ -b_N & a_N & & b_N & a_N & \\ 0 & \mathbf{I}_{(N/2)-2} & 0 & 0 & -\mathbf{I}_{(N/2)-2} & \end{bmatrix}\begin{bmatrix} \mathbf{S}_{N/2} & 0 \\ 0 & \mathbf{S}_{N/2} \end{bmatrix}$$

Where I_m is the identity matrix of order M x M, and

$$\mathbf{S}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The coefficients are

$$a_N = \left[\frac{3N^2}{4(N^2-1)}\right]^{1/2}$$

and

$$b_N = \left[\frac{N^2 - 4}{4(N^2 - 1)}\right]^{1/2}$$

The slant transform for N = 4 will be

$$S_4 = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix}$$

Properties of Slant transform

(e) The slant transform is real and orthogonal.

$\mathbf{S = S^*}$

$$\mathbf{S^{-1} = S^T}$$

(f) The slant transform is fast, it can be implemented in ($N \log_2 N$) operations on an N x 1 vector.

(g) The energy deal for images in this transform is rated in very good to excellent range.

(h) The mean vectors for slant transform matrix S are not sequentially ordered for $n \geq 3$.

Discrete cosine transform.

The 1-D discrete cosine transform is defined as

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x)\cos\left[\frac{(2x + 1)u\pi}{2N}\right]$$

For u = 0, 1, 2, . . , N-1. Similarly the inverse DCT is defined as

$$f(x) = \sum_{u=0}^{N-1} \alpha(u)C(u)\cos\left[\frac{(2x + 1)u\pi}{2N}\right]$$

For u = 0, 1, 2, . . , N-1

Where α is

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1, 2, \ldots, N-1. \end{cases}$$

The corresponding 2-D DCT pair is

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y)\cos\left[\frac{(2x + 1)u\pi}{2N}\right]\cos\left[\frac{(2y + 1)v\pi}{2N}\right]$$

For u, v = 0, 1, 2, . . , N-1, and

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u, v)\cos\left[\frac{(2x + 1)u\pi}{2N}\right]\cos\left[\frac{(2y + 1)v\pi}{2N}\right]$$

For x, y= 0, 1, 2, . . , N-1

SVD and KL Transfrom or The basic principle of Hotelling transform.

Hotelling transform:

The basic principle of hotelling transform is the statistical properties of vector

representation. Consider a population of random vectors of the form,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

And the mean vector of the population is defined as the expected value of x i.e.,

$$m_X = E\{x\}$$

The suffix m represents that the mean is associated with the population of x vectors. The expected value of a vector or matrix is obtained by taking the expected value of each elememt.
The covariance matrix C_X in terms of x and m_X is given as

$$C_X = E\{(x-m_X)\,(x-m_X)^T\}$$

T denotes the transpose operation. Since, x is n dimensional, $\{(x-m_X)\,(x-m_X)^T\}$ will be of n x n dimension. The covariance matrix is real and symmetric. If elements x_i and x_j are uncorrelated, their covariance is zero and, therefore, $c_{ij} = c_{ji} = 0$.

For M vector samples from a random population, the mean vector and covariance matrix can be approximated from the samples by

$$\mathbf{m}_x = \frac{1}{M}\sum_{k=1}^{M} \mathbf{x}_k$$

and

$$\mathbf{C}_x = \frac{1}{M}\sum_{k=1}^{M} \mathbf{x}_k\mathbf{x}_k^T - \mathbf{m}_x\mathbf{m}_x^T.$$

UNIT- II INTENSITY TRANSFORMATIONS AND SPATIAL FILTERING

Background:

Image enhancement approaches fall into two broad categories: spatial domain methods and frequency domain methods. The term *spatial domain* refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image. *Frequency domain* processing techniques are based on modifying the Fourier transform of an image.

Basic Gray Level Transformations:

The study of image enhancement techniques is done by discussing gray-level transformation functions. These are among the simplest of all image enhancement techniques. The values of pixels, before and after processing, will be denoted by r and s, respectively. As indicated in the previous section, these values are related by an expression of the form s=T(r), where T is a transformation that maps a pixel value r into a pixel value s. Since we are dealing with digital quantities, values of the transformation function typically are stored in a one-dimensional array and the mappings from r to s are implemented via table lookups. For an 8-bit environment, a lookup table containing the values of T will have 256 entries. As an introduction to gray-level transformations, consider Fig. 1.1, which shows three basic types of functions used frequently for image enhancement: linear (negative and identity transformations), logarithmic (log and inverse-log transformations), and power-law (nth power and nth root transformations).The identity function is the trivial case in which output intensities are identical to input intensities. It is included in the graph only for completeness.

Image Negatives:

The negative of an image with gray levels in the range [0, L-1] is obtained by using the negative transformation shown in Fig., which is given by the expression
$s = L - 1 - r$.

Reversing the intensity levels of an image in this manner produces the equivalent of a photographic negative. This type of processing is particularly suited for enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size.

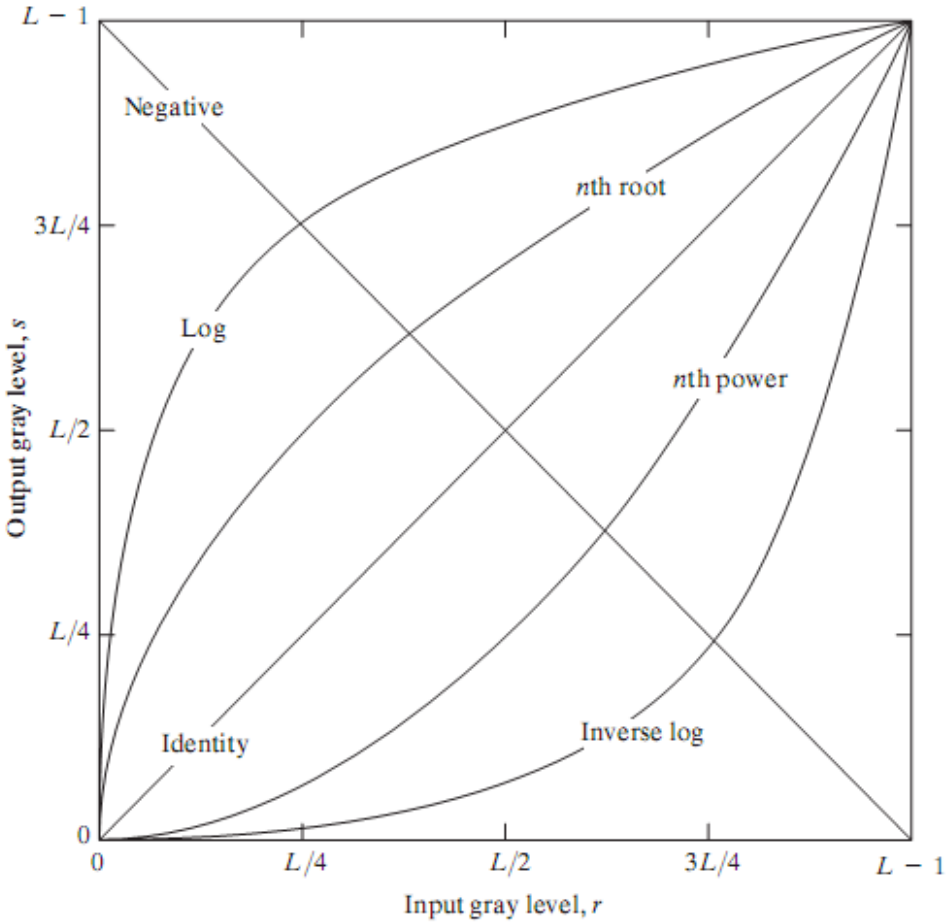


Fig. Some basic gray-level transformation functions used for image enhancement

Log Transformations:

The general form of the log transformation shown in Fig.1.1 is

$$s = c \log(1 + r)$$

where c is a constant, and it is assumed that $r \geq 0$.The shape of the log curve in Fig. 1.1 shows that this transformation maps a narrow range of low gray-level values in the input image into a wider range of output levels. The opposite is true of higher values of input levels. We would use a transformation of this type to expand the values of dark pixels in an image while compressing the higher-level values. The opposite is true of the inverse log transformation.

Any curve having the general shape of the log functions shown in Fig. 1.1 would accomplish this spreading/compressing of gray levels in an image. In fact, the power-law transformations discussed in the next section are much more versatile for this purpose than the

log transformation. However, the log function has the important characteristic that it compresses the dynamic range of images with large variations in pixel values. A classic illustration of an application in which pixel values have a large dynamic range is the Fourier spectrum. At the moment, we are concerned only with the image characteristics of spectra. It is not unusual to encounter spectrum values that range from 0 to or higher. While processing numbers such as these presents no problems for a computer, image display systems generally will not be able to reproduce faithfully such a wide range of intensity values. The net effect is that a significant degree of detail will be lost in the display of a typical Fourier spectrum.

Power-Law Transformations:

Power-law transformations have the basic form

$$s = cr^{\gamma}$$

where c and g are positive constants. Sometimes Eq. is written as $s = c(r + \varepsilon)^{\gamma}$ to account for an offset (that is, a measurable output when the input is zero).However, offsets typically are an issue of display calibration and as a result they are normally ignored in Eq. Plots of s versus r for various values of g are shown in Fig.

As in the case of the log transformation, power-law curves with fractional values of g map a narrow range of dark input values into a wider range of output values,with the opposite being true for high-er values of input levels. Unlike the log function, however, we notice here a family of possible transformation curves obtained simply by varying γ.

As expected, we see in Fig. that curves generated with values of g>1 have exactly the opposite effect as those generated with values of g<1. Finally, we note that Eq. reduces to the identity transformation when c = γ = 1. A variety of devices used for image capture, printing, and display respond according to a power law.
By convention, the exponent in the power-law equation is referred to as gamma. The proces used to correct this power-law response phenomena is called gamma correction.

For example, cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5.With reference to the curve for g=2.5 in Fig. we see that such display systems would tend to produce images that are darker than intended.

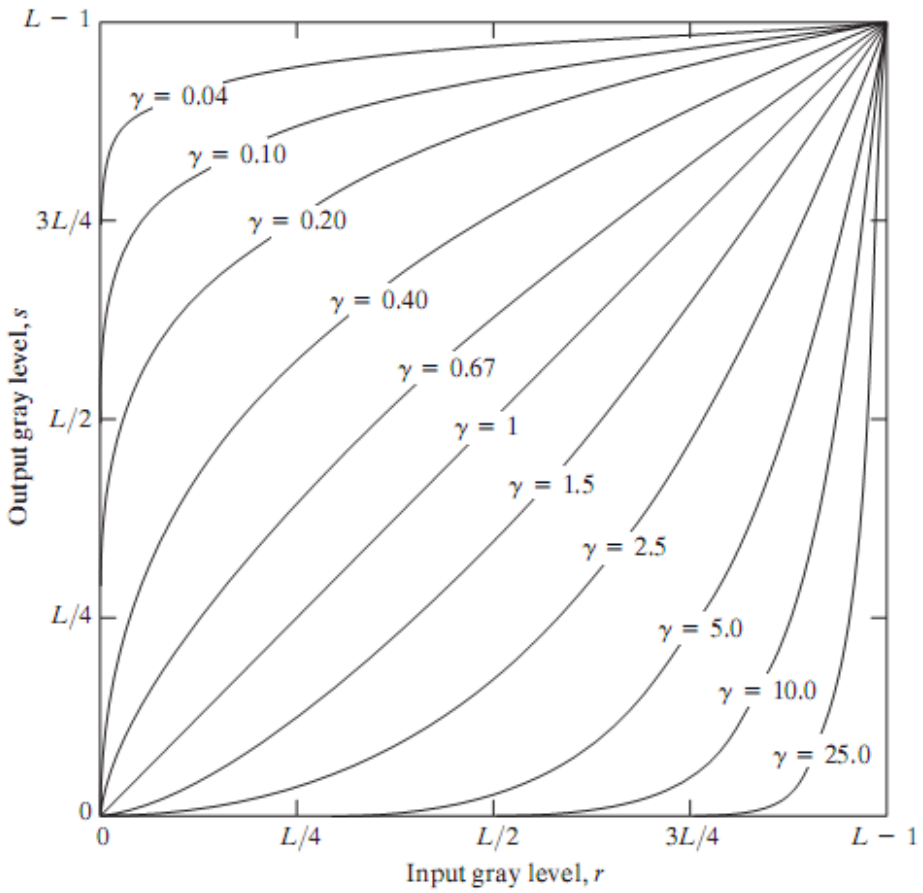


Fig. Plots of the equation $s = cr^{\gamma}$ for various values of γ (c=1 in all cases).

Piecewise-Linear Transformation Functions:

The principal advantage of piecewise linear functions over the types of functions we have discussed above is that the form of piecewise functions can be arbitrarily complex. In fact, as we will see shortly, a practical implementation of some important transformations can be formulated only as piecewise functions. The principal disadvantage of piecewise functions is that their specification requires considerably more user input.

Contrast stretching:

One of the simplest piecewise linear functions is a contrast-stretching transformation. Low- contrast images can result from poor illumination, lack of dynamic range in the imaging sensor, or even wrong setting of a lens aperture during image acquisition.The idea behind contrast stretching is to increase the dynamic range of the gray levels in the image being processed.

Figure 1.3 (a) shows a typical transformation used for contrast stretching.

The locations of points (r1 , s1) and (r2 , s2) control the shape of the transformation

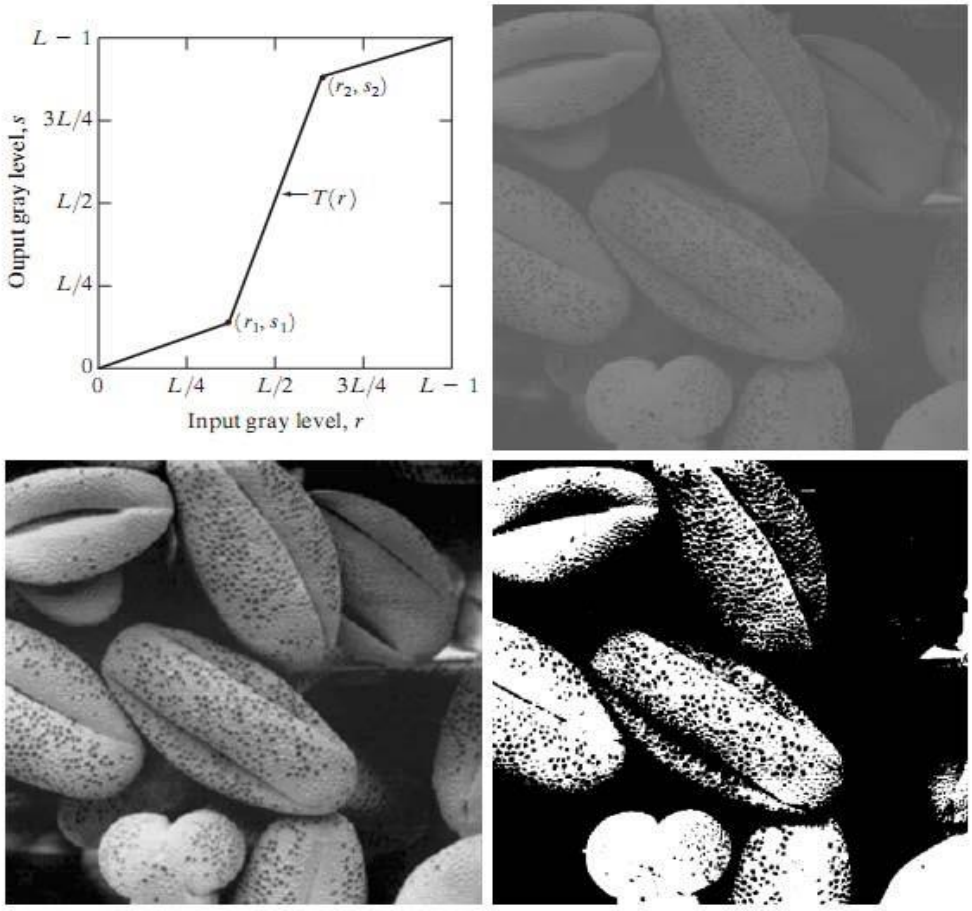


Fig. Contrast Stretching (a) Form of Transformation function (b) A low-contrast image **(c) Result of contrast stretching (d) Result of thresholding.**

function. If r1=s1 and r2=s2, the transformation is a linear function that produces no changes in gray levels. If r1=r2,s1=0 and s2=L-1, the transformation becomes a thresholding function that creates a binary image, as illustrated in Fig. (b). Intermediate values of (r_1 , s_1) and (r_2 , s_2) produce various degrees of spread in the gray levels of the output image, thus affecting its contrast. In general, r1 ≤ r2 and s1 ≤ s2 is assumed so that the function is single valued and monotonically increasing. This condition preserves the order of gray levels, thus preventing the creation of intensity artifacts in the processed image.

Figure (b) shows an 8-bit image with low contrast. Fig. (c) shows the result of contrast stretching, obtained by setting (r_1 , s_1) = (r_{min} , 0) and (r_2 , s_2) = (r_{max} , L-1) where r_{min} and r_{max} denote the minimum and maximum gray levels in the image, respectively. Thus, the transformation function stretched the levels linearly from their original range to the full range [0, L-1]. Finally, Fig. (d) shows the result of using the thresholding function defined previously, with r1 = r2 = m, the mean gray level in the image. The original image on which these results are based is a scanning electron microscope image of pollen, magnified approximately 700 times.

Gray-level slicing:

Highlighting a specific range of gray levels in an image often is desired. Applications include enhancing features such as masses of water in satellite imagery and enhancing flaws in X-ray images. There are several ways of doing level slicing, but most of them are variations of two basic themes. One approach is to display a high value for all gray levels in the range of interest and a low value for all other gray levels. This transformation, shown in Fig. (a), produces a binary image. The second approach, based on the transformation shown in Fig. (b), brightens the desired range of gray levels but preserves the background and gray-level tonalities in the image. Figure (c) shows a gray-scale image, and Fig. (d) shows the result of using the transformation in Fig. (a).Variations of the two transformations shown in Fig. are easy to formulate.

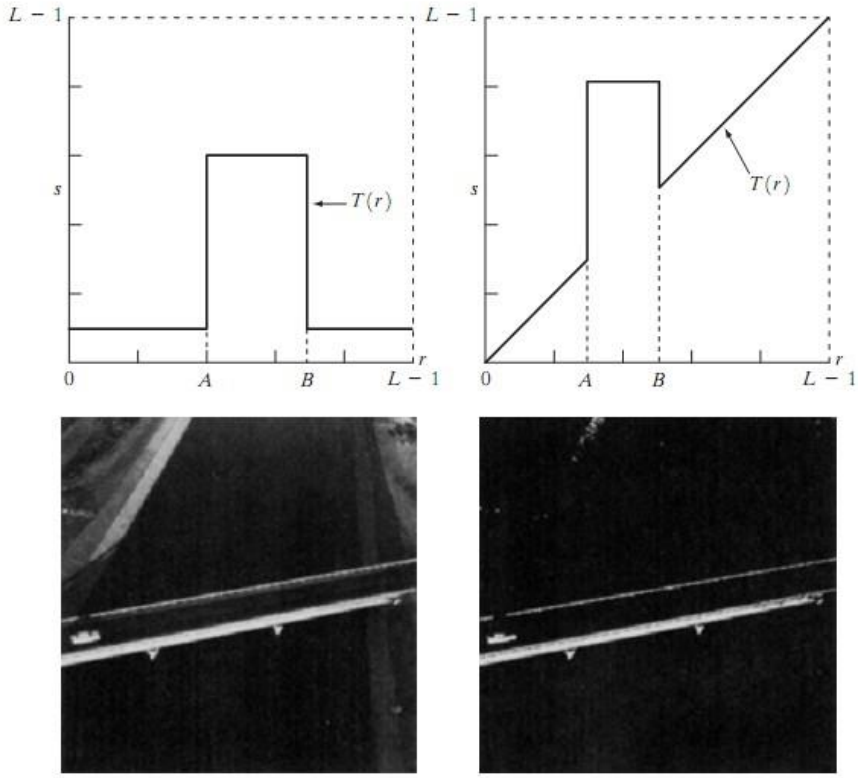


Fig. (a) This transformation highlights range [A, B] of gray levels and reduce all others to a constant level (b) This transformation highlights range [A, B] but preserves all other levels (c) An image (d) Result of using the transformation in (a).

Bit-plane slicing:

Instead of highlighting gray-level ranges, highlighting the contributionmade to total image appearance by specific bits might be desired. Suppose that each pixel in an image is represented by 8 bits. Imagine that the image is composed of eight 1-bit planes, ranging from bit-plane 0 for the least significant bit to bit plane 7 for the most significant bit. In terms of 8-bit bytes, plane 0

contains all the lowest order bits in the bytes comprising the pixels in the image and plane 7 contains all the high-order bits.Figure illustrates these ideas, and Fig. shows the various bit planes for the image shown in Fig. . Note that the higher-order bits (especially the top four) contain themajority of the visually significant data.The other bit planes contribute tomore subtle details in the image. Separating a digital image into its bit planes is useful for analyzing the relative importance played by each bit of the image, a process that aids in determining the adequacy of the number of bits used to quantize each pixel.

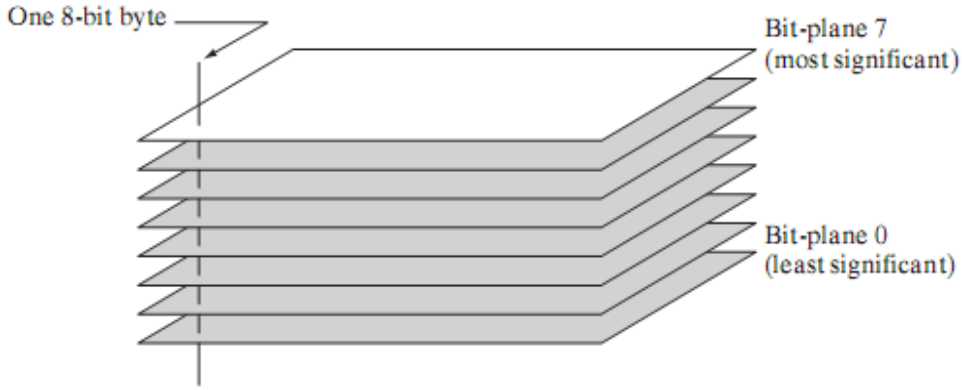


Fig.Bit-plane representation of an 8-bit image.

In terms of bit-plane extraction for an 8-bit image, it is not difficult to show that the (binary) image for bit-plane 7 can be obtained by processing the input image with a thresholding gray- level transformation function that (1) maps all levels in the image between 0 and 127 to one level (for example, 0); and (2) maps all levels between 129 and 255 to another (for example, 255).

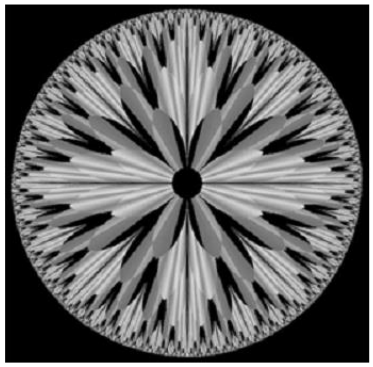


Fig.An 8-bit fractal image

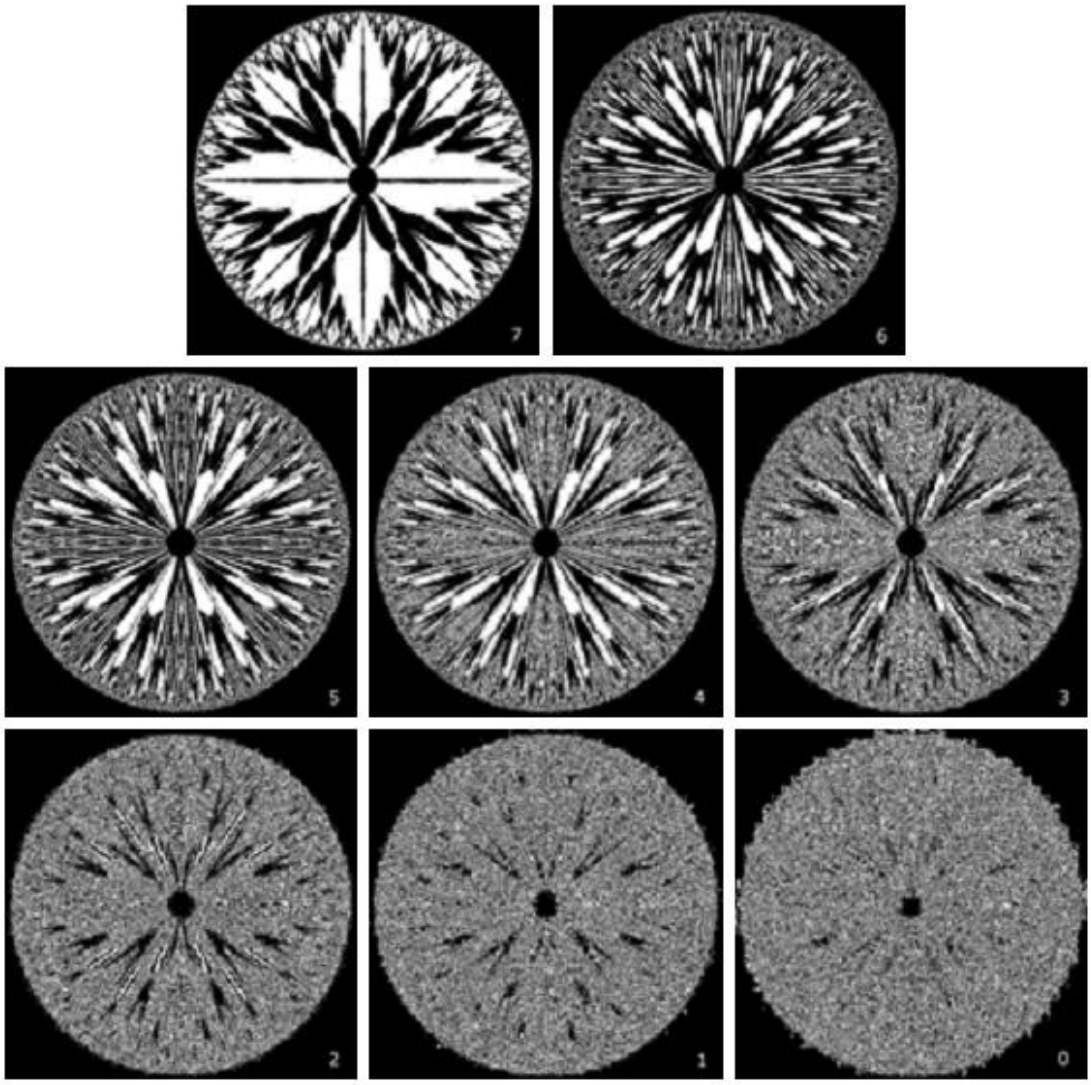


Fig. The eight bit planes of the image in Fig.. The number at the bottom, right of each image identifies the bit plane.

Objective of image enhancement, spatial domain, point processing.

The term spatial domain refers to the aggregate of pixels composing an image. Spatial domain methods are procedures that operate directly on these pixels. Spatial domain processes will be denoted by the expression

$$g(x, y) = T[f(x, y)]$$

where f(x, y) is the input image, g(x, y) is the processed image, and T is an operator on f, defined over some neighborhood of (x, y). In addition,T can operate on a set of input images, such as performing the pixel-by-pixel sum of K images for noise reduction.
The principal approach in defining a neighborhood about a point (x, y) is to use a square or

rectangular subimage area centered at (x, y), as Fig. shows. The center of the subimage is moved from pixel to pixel starting, say, at the top left corner. The operator T is applied at each location (x, y) to yield the output, g, at that location.The process utilizes only the pixels in the area of the image spanned by the neighborhood.

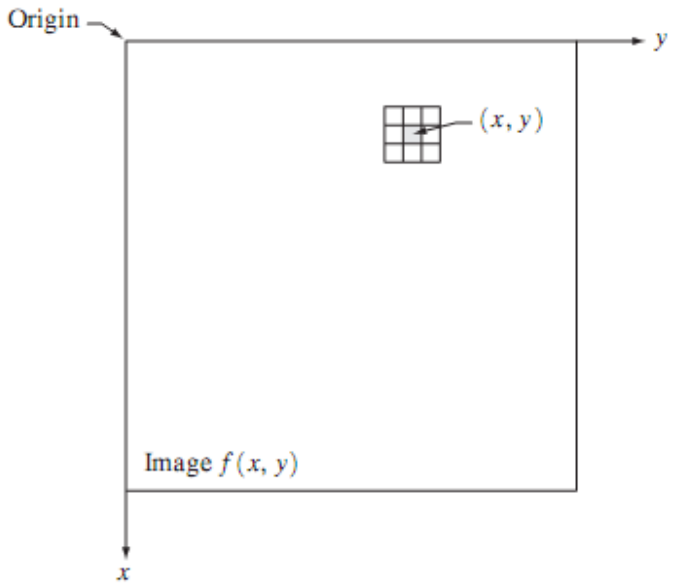


Fig. A 3*3 neighborhood about a point (x, y) in an image.

Although other neighborhood shapes, such as approximations to a circle, sometimes are used, square and rectangular arrays are by far the most predominant because of their ease of implementation. The simplest form of T is when the neighborhood is of size 1*1 (that is, a single pixel). In this case, g depends only on the value of f at (x, y), and T becomes a gray-level (also called an intensity or mapping) transformation function of the form

$$s = T(r)$$

where, for simplicity in notation, r and s are variables denoting, respectively, the gray level of f(x, y) and g(x, y) at any point (x, y). For example, if T(r) has the form shown in Fig. 2.2(a), the effect of this transformation would be to produce an image of higher contrast than the original by darkening the levels below m and brightening the levels above m in the original image. In this technique, known as contrast stretching, the values of r below m are compressed by the transformation function into a narrow range of s, toward black.The opposite effect takes place for values of r above m. In the limiting case shown in Fig. (b), T(r) produces a two-level (binary) image. A mapping of this form is called a thresholding function. Some fairly simple, yet powerful, processing approaches can be formulated with gray-level transformations. Because enhancement at any point in an image depends only on the gray level at that point,

techniques in this category often are referred to as point processing.

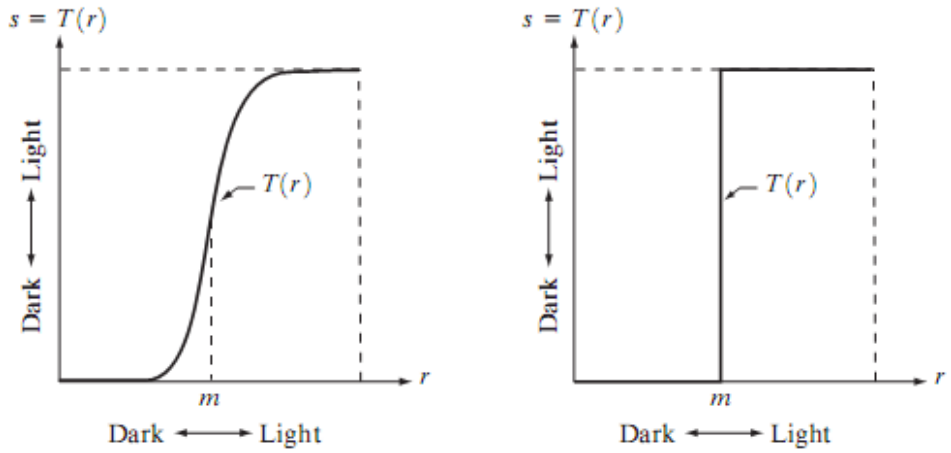


Fig. Gray level transformation functions for contrast enhancement.

Larger neighborhoods allow considerably more flexibility. The general approach is to use a function of the values of f in a predefined neighborhood of (x, y) to determine the value of g at (x, y).One of the principal approaches in this formulation is based on the use of so-called masks (also referred to as filters, kernels, templates, or windows). Basically, a mask is a small (say, 3*3) 2-D array, such as the one shown in Fig., in which the values of the mask coefficients determine the nature of the process, such as image sharpening.

Histogram Processing:

The histogram of a digital image with gray levels in the range [0, L-1] is a discrete function $h(r_k)= (n_k)$, where r_k is the kth gray level and n_k is the number of pixels in the image having gray level r_k. It is common practice to normalize a histogram by dividing each of its values by the total number of pixels in the image, denoted by n. Thus, a normalized histogram is given by

$$p(r_k) = n_k/n$$

For k=0,1,…… .,L-1. Loosely speaking, $p(r_k)$ gives an estimate of the probability of occurrence of gray level r_k. Note that the sum of all components of a normalized histogram is equal to 1 .

Histograms are the basis for numerous spatial domain processing techniques. Histogram manipulation can be used effectively for image enhancement. Histograms are simple to calculate in software and also lend themselves to economic hardware implementations, thus making them a popular tool for real-time image processing.

As an introduction to the role of histogram processing in image enhancement, consider Fig. , which is the pollen image shown in four basic gray-level characteristics: dark, light, low contrast, and high contrast. The right side of the figure shows the histograms corresponding to these images. The horizontal axis of each histogram plot corresponds to gray level values, r_k.

The vertical axis corresponds to values of $h(r_k) = n_k$ or $p(r_k) = n_k/n$ if the values are normalized. Thus, as indicated previously, these histogram plots are simply plots of $h(r_k) = n_k$ versus r_k or $p(r_k) = n_k/n$ versus r_k.

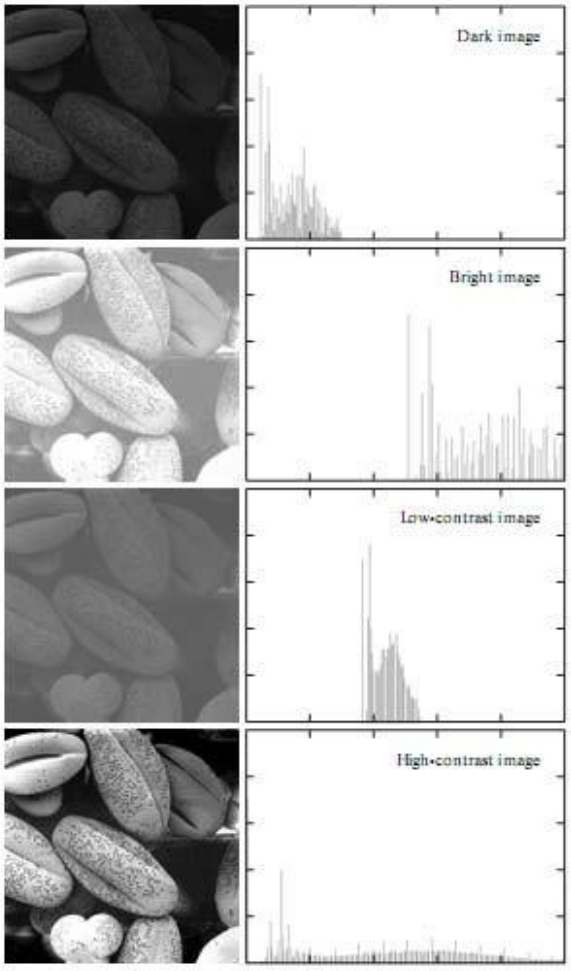


Fig. Four basic image types: dark, light, low contrast, high contrast, and their corresponding histograms.

We note in the dark image that the components of the histogram are concentrated on the low (dark) side of the gray scale. Similarly, the components of the histogram of the bright image are biased toward the high side of the gray scale.An image with low contrast has a histogram that will be narrow and will be centered toward the middle of the gray scale. For a monochrome image this implies a dull, washed-out gray look. Finally, we see that the components of the histogram in the high-contrast image cover a broad range of the gray scale and, further, that the distribution of pixels is not too far from uniform, with very few vertical lines being much higher than the others. Intuitively, it is reasonable to conclude that an image whose pixels tend to occupy the entire range of possible gray levels and, in addition, tend to be distributed uniformly,will have an appearance of high contrast and will exhibit a large variety of gray tones. The net effect will be an image that shows a great deal of gray-level detail and has high dynamic range. It will be shown shortly that it is possible to develop a

transformation function that can automatically achieve this effect, based only on information available in the histogram of the input image.

Histogram Equalization:

Consider for a moment continuous functions, and let the variable r represent the gray levels of the image to be enhanced. We assume that r has been normalized to the interval [0, 1], with r=0 representing black and r=1 representing white. Later, we consider a discrete formulation and allow pixel values to be in the interval [0, L-1]. For any r satisfying the aforementioned conditions, we focus attention on transformations of the form

$$s = T(r) \qquad 0 \le r \le 1$$

That produce a level s for every pixel value r in the original image. For reasons that will become obvious shortly, we assume that the transformation function T(r) satisfies the following conditions:

(a) T(r) is single-valued and monotonically increasing in the interval $0 \le r \le 1$; and (b) $0 \le T(r) \le 1$ for $0 \le r \le 1$.

The requirement in (a) that T(r) be single valued is needed to guarantee that the inverse transformation will exist, and the monotonicity condition preserves the increasing order from black to white in the output image. A transformation function that is not monotonically increasing could result in at least a section of the intensity range being inverted, thus producing some inverted gray levels in the output image. Finally, condition (b) guarantees that the output gray levels will be in the same range as the input levels. Figure gives an example of a transformation function that satisfies these two conditions. The inverse transformation from s back to r is denoted

$$r = T^{-1}(s) \qquad 0 \le s \le 1.$$

It can be shown by example that even if T(r) satisfies conditions (a) and (b), it is possible that the corresponding inverse T^{-1} (s) may fail to be single valued.

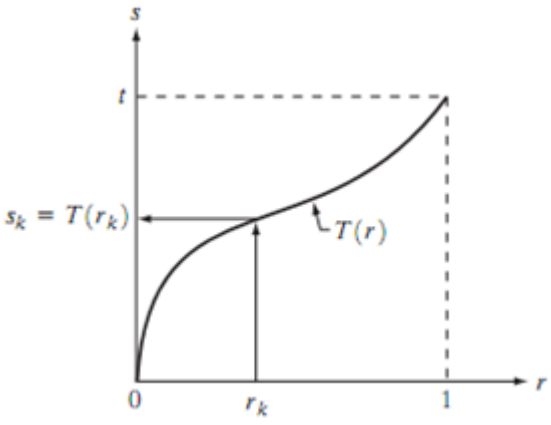


Fig. A gray-level transformation function that is both single valued and monotonically

increasing.

The gray levels in an image may be viewed as random variables in the interval [0, 1].One of the most fundamental descriptors of a random variable is its probability density function (PDF).Let $p_r(r)$ and $p_S(s)$ denote the probability density functions of random variables r and s, respectively, where the subscripts on p are used to denote that p_r and p_S are different functions. A basic result from an elementary probability theory is that, if $p_r(r)$ and T(r) are known and T^{-1} (s)
Satisfies condition (a), then the probability density function $p_S(s)$ of the transformed variable s can be obtained using a rather simple formula:

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right|.$$

Thus, the probability density function of the transformed variable, s, is determined by the gray- level PDF of the input image and by the chosen transformation function. A transformation function of particular importance in image processing has the form

$$s = T(r) = \int_0^r p_r(w)\, dw$$

Where w is a dummy variable of integration. The right side of Eq. above is recognized as the cumulative distribution function (CDF) of random variable r. Since probability density functions are always positive, and recalling that the integral of a function is the area under the function, it follows that this transformation function is single valued and monotonically increasing, and, therefore, satisfies condition (a). Similarly, the integral of a probability density function for variables in the range [0, 1] also is in the range [0, 1], so condition (b) is satisfied as well.

Given transformation function T(r), we find $p_S(s)$ by applying Eq. We know from basic calculus (Leibniz's rule) that the derivative of a definite integral with respect to its upper limit is simply the integrand evaluated at that limit. In other words,

$$\begin{aligned}\frac{ds}{dr} &= \frac{dT(r)}{dr} \\ &= \frac{d}{dr}\left[\int_0^r p_r(w)\, dw\right] \\ &= p_r(r).\end{aligned}$$

Substituting this result for dr/ds, and keeping in mind that all probability values are positive, yields

$$
\begin{aligned}
p_s(s) &= p_r(r)\left|\frac{dr}{ds}\right| \\
&= p_r(r)\left|\frac{1}{p_r(r)}\right| \\
&= 1 \qquad 0 \le s \le 1.
\end{aligned}
$$

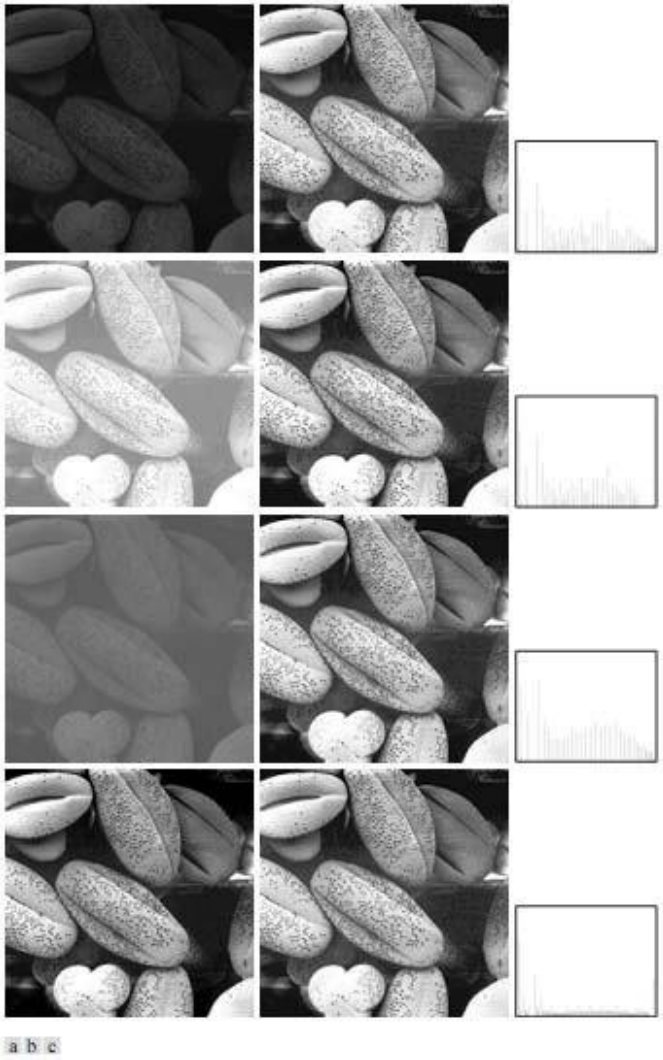
Because $p_S(s)$ is a probability density function, it follows that it must be zero outside the interval [0, 1] in this case because its integral over all values of s must equal 1.We recognize the form of $p_S(s)$ as a uniform probability density function. Simply stated, we have demonstrated that performing the transformation function yields a random variable s characterized by a uniform probability density function. It is important to note from Eq. discussed above that T(r) depends on $p_r(r)$, but, as indicated by Eq. after it, the resulting $p_S(s)$ always is uniform, independent of the form of $p_r(r)$. For discrete values we deal with probabilities and summations instead of probability density functions and integrals. The probability of occurrence of gray level r in an image is approximated by

$$
p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L - 1
$$

where, as noted at the beginning of this section, n is the total number of pixels in the image, n_k is the number of pixels that have gray level r_k, and L is the total number of possible gray levels in the image.The discrete version of the transformation function given in Eq. is

$$
\begin{aligned}
s_k &= T(r_k) = \sum_{j=0}^{k} p_r(r_j) \\
&= \sum_{j=0}^{k} \frac{n_j}{n} \qquad k = 0, 1, 2, \ldots, L - 1.
\end{aligned}
$$

Thus, a processed (output) image is obtained by mapping each pixel with level r_k in the input image into a corresponding pixel with level s_k in the output image. As indicated earlier, a plot of $p_r(r_k)$ versus r_k is called a histogram. The transformation (mapping) is called histogram equalization or histogram linearization. It is not difficult to show that the transformation in Eq. satisfies conditions (a) and (b) stated previously. Unlike its continuos counterpart, it cannot be proved in general that this discrete transformation will produce the discrete equivalent of a uniform probability density function, which would be a uniform histogram.



a b c

Fig. (a) Images from Fig. (b) Results of histogram equalization. (c) Corresponding histograms.
The inverse transformation from s back to r is denoted by

$$r_k = T^{-1}(s_k) \qquad k = 0, 1, 2, \ldots, L - 1$$

Histogram Matching (Specification):

Histogram equalization automatically determines a transformation function that seeks to produce an output image that has a uniform histogram.When automatic enhancement is desired, this is a good approach because the results from this technique are predictable and the method is simple to implement. In particular, it is useful sometimes to be able to specify the shape of the histogram that we wish the processed image to have.The method used to generate a processed image that has a specified histogram is called histogram matching or histogram specification.

Development of the method:

Let us return for a moment to continuous gray levels r and z (considered continuous random variables), and let $p_r(r)$ and $p_z(z)$ denote their corresponding continuos probability density functions. In this notation, r and z denote the gray levels of the input and output (processed) images, respectively.We can estimate $p_r(r)$ from the given input image, while $p_z(z)$ is the specified probability density function that we wish the output image to have.

Let s be a random variable with the property

$$s = T(r) = \int_0^r p_r(w)\, dw$$

where w is a dummy variable of integration.We recognize this expression as the continuos version of histogram equalization. Suppose next that we define a random variable z with the property

$$G(z) = \int_0^z p_z(t)\, dt = s$$

where t is a dummy variable of integration. It then follows from these two equations that G(z)=T(r) and, therefore, that z must satisfy the condition

$$z = G^{-1}(s) = G^{-1}[T(r)].$$

The transformation T(r) can be obtained once $p_r(r)$ has been estimated from the input image. Similarly, the transformation function G(z) can be obtained because $p_z(z)$ is given. Assuming that G^{-1} exists and that it satisfies conditions (a) and (b) in the histogram equalization process,

The above three equations show that an image with a specified probability density function can be obtained from an input image by using the following procedure:

Obtain the transformation function T(r).

To obtain the transformation function G(z).

Obtain the inverse transformation function G^{-1}

Obtain the output image by applying above Eq. to all the pixels in the input image.

The result of this procedure will be an image whose gray levels, z, have the specified probability density function $p_Z(z)$. Although the procedure just described is straightforward in principle, it's seldom possible in practice to obtain analytical expressions for T(r) and for G^{-1}.

Fortunately, this problem is simplified considerably in the case of discrete values.The price we pay is the same as in histogram equalization,where only an approximation to the desired histogram is achievable. In spite of this, however, some very useful results can be obtained even with crude approximations.

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j)$$
$$= \sum_{j=0}^{k} \frac{n_j}{n} \qquad k = 0, 1, 2, \ldots, L-1$$

where n is the total number of pixels in the image, n_j is the number of pixels with gray level r_j, and L is the number of discrete gray levels. Similarly, the discrete formulation is obtained from the given histogram $p_Z(z_i)$, i=0, 1, 2,……, L-1, and has the form

$$v_k = G(z_k) = \sum_{i=0}^{k} p_z(z_i) = s_k \qquad k = 0, 1, 2, \ldots, L-1.$$

As in the continuous case, we are seeking values of z that satisfy this equation.The variable v_k was added here for clarity in the discussion that follows. Finally, the discrete version of the above Eqn. is given by

$$z_k = G^{-1}[T(r_k)] \qquad k = 0, 1, 2, \ldots, L-1$$

Or

$$z_k = G^{-1}(s_k) \qquad k = 0, 1, 2, \ldots, L-1.$$

Implementation:

We start by noting the following: (1) Each set of gray levels $\{r_j\}$, $\{s_j\}$, and $\{z_j\}$, j=0, 1, 2, p , L- 1, is a one-dimensional array of dimension L X 1. (2) All mappings from r to s and from s to z are simple table lookups between a given pixel value and these arrays. (3) Each of the elements of these arrays, for example, s_k, contains two important pieces of information: The subscript k denotes the location of the element in the array, and s denotes the value at that location. (4) We need to be concerned only with integer pixel values. For example, in the case of an 8-bit image, L=256 and the elements of each of the arrays just mentioned are integers between 0 and 255.This implies that we now work with gray level values in the interval [0, L-1] instead of the normalized interval [0, 1] that we used before to simplify the development of

histogram processing techniques.

In order to see how histogram matching actually can be implemented, consider Fig. (a), ignoring for a moment the connection shown between this figure and Fig. (c). Figure (a) shows a hypothetical discrete transformation function s=T(r) obtained from a given image. The first gray level in the image, r_1 , maps to s_1 ; the second gray level, r_2 , maps to s_2 ; the kth level r_k maps to s_k; and so on (the important point here is the ordered correspondence between these values). Each value s_j in the array is precomputed, so the process of mapping simply uses the actual value of a pixel as an index in an array to determine the corresponding value of s.This process is particularly easy because we are dealing with integers. For example, the s mapping for an 8-bit pixel with value 127 would be found in the 128th position in array $\{s_j\}$ (recall that we start at 0) out of the possible 256 positions. If we stopped here and mapped the value of each pixel of an input image by the

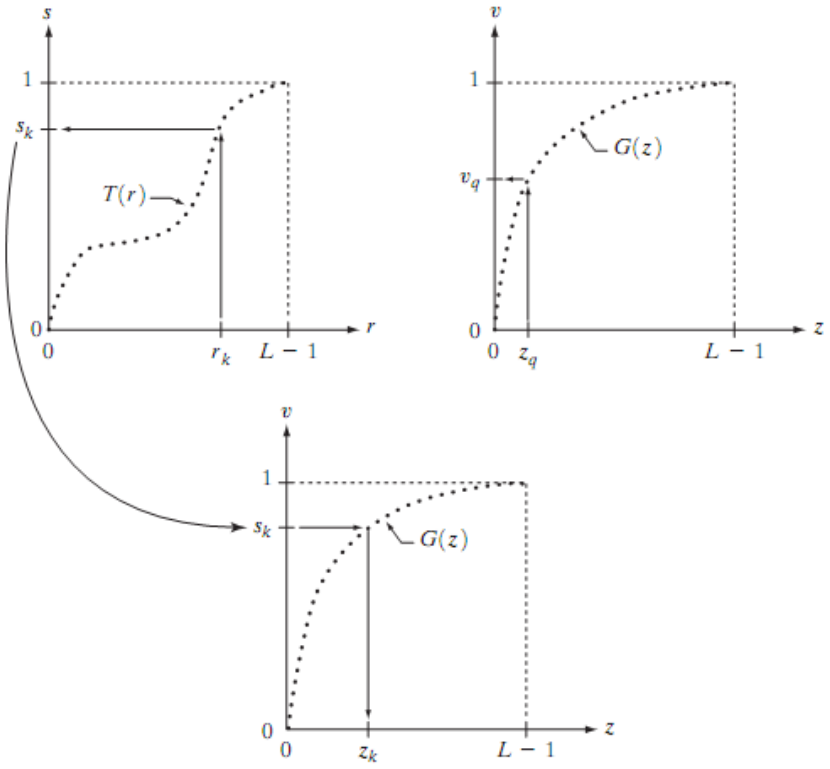


Fig (a) Graphical interpretation of mapping from r_k to s_k via T(r). (b) Mapping of z_q to its corresponding value v_q via G(z) (c) Inverse mapping from s_k to its corresponding value of z_k.

Method just described, the output would be a histogram-equalized image. In order to implement histogram matching we have to go one step further. Figure (b) is a hypothetical transformation function G obtained from a given histogram $p_z(z)$. For any z_q , this transformation function yields a corresponding value v_q. This mapping is shown by the arrows in Fig. 5(b). Conversely, given any value v_q, we would find the corresponding value z_q from

G^{-1}. In terms of the figure, all this means graphically is that we would reverse the direction of the arrows to map v_q into its corresponding z_q. However, we know from the definition that v=s for corresponding subscripts,
so we can use exactly this process to find the z_k corresponding to any value s_k that we computed previously from the equation $s_k = T(r_k)$ .This idea is shown in Fig.(c).

Since we really do not have the z's (recall that finding these values is precisely the objective of histogram matching),we must resort to some sort of iterative scheme to find z from s.The fact that we are dealing with integers makes this a particularly simple process. Basically, because $v_k = s_k$, we have that the z's for which we are looking must satisfy the equation $G(z_k)=s_k$, or $(G(z_k)- s_k)=0$. Thus, all we have to do to find the value of z_k corresponding to s_k is to iterate on values of z such that this equation is satisfied for k=0,1,2,…....., L-1. We do not have to find the inverse of G because we are going to iterate on z. Since we are dealing with integers, the closest we can get to satisfying the equation $(G(z_k)-s_k)=0$ is to let $z_k= \hat{z}$ for each value of k, where $\hat{z}$ is the smallest integer in the interval [0, L-1] such that

$$(G(\hat{z}) - s_k) \geq 0 \qquad k = 0, 1, 2, \ldots, L - 1.$$

Given a value s_k, all this means conceptually in terms of Fig. (c) is that we would start with and increase it in integer steps until Eq is satisfied, at which point we let repeating this process for all values of k would yield all the required mappings from s to z, which constitutes the implementation of Eq. In practice, we would not have to start with each time because the values of sk are known to increase monotonically. Thus, for k=k+1, we would start with $\hat{z} = z_k$ and increment in integer values from there.

Local Enhancement:

The histogram processing methods discussed in the previous two sections are global, in the sense that pixels are modified by a transformation function based on the gray-level content of an entire image. Although this global approach is suitable for overall enhancement, there are cases in which it is necessary to enhance details over small areas in an image. The number of pixels in these areas may have negligible influence on the computation of a global transformation whose shape does not necessarily guarantee the desired local enhancement. The solution is to devise transformation functions based on the gray-level distribution—or other properties—in the neighborhood of every pixel in the image.

The histogram processing techniques are easily adaptable to local enhancement.The procedure is to define a square or rectangular neighborhood and move the center of this area from pixel to pixel. At each location, the histogram of the points in the neighborhood is computed and either a histogram equalization or histogram specification transformation function is obtained. This function is finally used to map the gray level of the pixel centered in the neighborhood. The center of the neighborhood region is then moved to an adjacent pixel location and the procedure is repeated. Since only one new row or column of the neighborhood

changes during a pixel-to-pixel translation of the region, updating the histogram obtained in the previous location with the new data introduced at each motion step is possible.

This approach has obvious advantages over repeatedly computing the histogram over all pixels in the neighborhood region each time the region is moved one pixel location. Another approach used some times to reduce computation is to utilize no overlapping regions, but this method usually produces an undesirable checkerboard effect.

Fundamentals of Spatial Filtering:

Some neighborhood operations work with the values of the image pixels in the neighborhood and the corresponding values of a sub image that has the same dimensions as the neighborhood. The sub image is called a filter, mask, kernel, template, or window, with the first three terms being the most prevalent terminology. The values in a filter sub image are referred to as coefficients, rather than pixels. The concept of filtering has its roots in the use of the Fourier transform for signal processing in the so-called frequency domain. We use the term spatial filtering to differentiate this type of process from the more traditional frequency domain filtering.

The mechanics of spatial filtering are illustrated in Fig. The process consists simply of moving the filter mask from point to point in an image. At each point (x, y), the response of the filter at that point is calculated using a predefined relationship. The response is given by a sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the filter mask. For the 3 x 3 mask shown in Fig. , the result (or response), R, of linear filtering with the filter mask at a point (x, y) in the image is

$$R = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \cdots + w(0,0)f(x,y) + \cdots + w(1,0)f(x+1,y) + w(1,1)f(x+1,y+1),$$

which we see is the sum of products of the mask coefficients with the corresponding pixels directly under the mask. Note in particular that the coefficient w(0, 0) coincides with image value f(x, y), indicating that the mask is centered at (x, y) when the computation of the sum of products takes place. For a mask of size m x n,we assume that m=2a+1 and n=2b+1,where a and b are nonnegative integers.

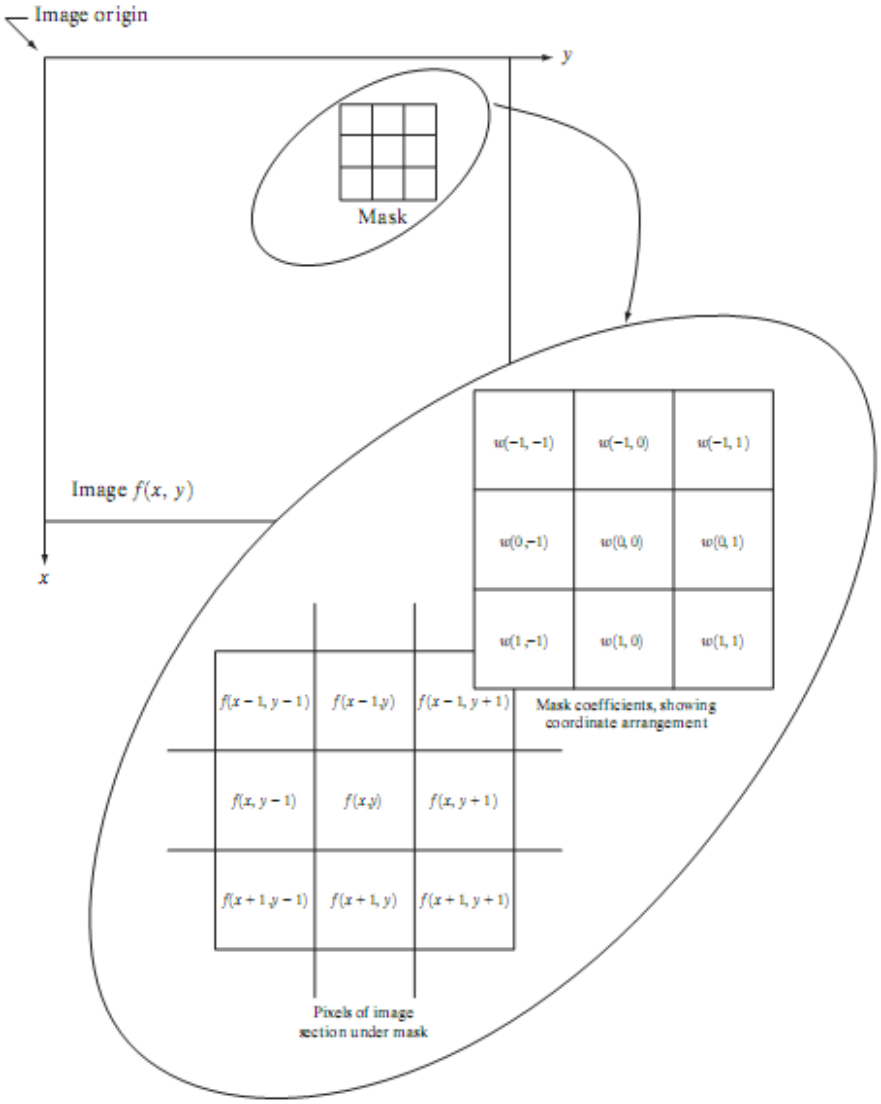


Fig. The mechanics of spatial filtering. The magnified drawing shows a 3X3 mask and the image section directly under it; the image section is shown displaced out from under the mask for ease of readability.

In general, linear filtering of an image f of size M x N with a filter mask of size m x n is given by the expression:

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)$$

where, from the previous paragraph, a=(m-1)/2 and b=(n-1)/2. To generate a complete filtered image this equation must be applied for x=0,1,2,……, M-1 and y=0,1,2,……, N-1. In this way, we are assured that the mask processes all pixels in the image. It is easily verified when m=n=3 that this expression reduces to the example given in the previous paragraph.

The process of linear filtering is similar to a frequency domain concept called convolution. For this reason, linear spatial filtering often is referred to as "convolving a mask with an image." Similarly, filter masks are sometimes called convolution masks. The term convolution kernel

also is in common use. When interest lies on the response, R, of an m x n mask at any point (x,y), and not on the mechanics of implementing mask convolution, it is common practice to simplify the notation by using the following expression:

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn}$$
$$= \sum_{i=1}^{mn} w_i z_i$$

where the w's are mask coefficients, the z's are the values of the image graylevels corresponding to those coefficients, and mn is the total number of coefficients in the mask. For the 3 x 3 general mask shown in Fig. the response at any point (x, y) in the image is given by

$$R = w_1 z_1 + w_2 z_2 + \ldots w_9 z_9$$
$$= \sum_{i=1}^{9} w_i z_i.$$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Fig. Another representation of a general 3 x 3 spatial filter mask.

An important consideration in implementing neighborhood operations for spatial filtering is the issue of what happens when the center of the filter approaches the border of the image. Consider for simplicity a square mask of size n x n.At least one edge of such a mask will coincide with the border of the image when the center of the mask is at a distance of (n-1)/2 pixels away from the border of the image. If the center of the mask moves any closer to the border, one or more rows or columns of the mask will be located outside the image plane. There are several ways to handle this situation. The simplest is to limit the excursions of the center of the mask to be at a distance no less than (n-1)/2 pixels from the border. The resulting filtered image will be smaller than the original, but all the pixels in the filtered imaged will have been processed with the full mask. If the result is required to be the same size as the original, then the approach typically employed is to filter all pixels only with the section of the mask that is fully contained in the image. With this approach, there will be bands of pixels near the border that will have been processed with a partial filter mask. Other approaches include "padding" the image by adding rows and columns of 0's (or other constant gray level),

or padding by replicating rows or columns. The padding is then stripped off at the end of the process.

This keeps the size of the filtered image the same as the original, but the values of the padding will have an effect near the edges that becomes more prevalent as the size of the mask increases. The only way to obtain a perfectly filtered result is to accept a somewhat smaller filtered image by limiting the excursions of the center of the filter mask to a distance no less than (n-1)/2 pixels from the border of the original image.

Smoothing Spatial Filters:

Smoothing filters are used for blurring and for noise reduction. Blurring is used in preprocessing steps, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by non-linear filtering.

Smoothing Linear Filters:

The output (response) of a smoothing, linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters sometimes are called averaging filters. The idea behind smoothing filters is straightforward. By replacing the value of every pixel in an image by the average of the gray levels in the neighborhood defined by the filter mask, this process results in an image with reduced "sharp" transitions in gray levels. Because random noise typically consists of sharp transitions in gray levels, the most obvious application of smoothing is noise reduction. However, edges (which almost always are desirable features of an image) also are characterized by sharp transitions in gray levels, so averaging filters have the undesirable side effect that they blur edges. Another application of this type of process includes the smoothing of false contours that result from using an insufficient number of gray l e v e l s .

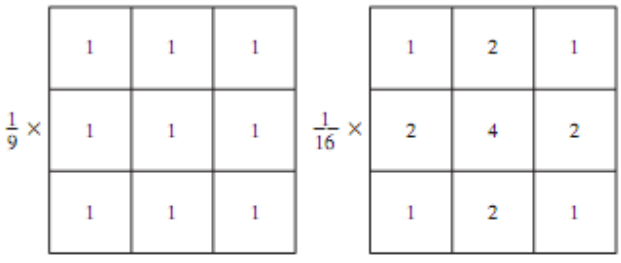


Fig.Two 3 x 3 smoothing (averaging) filter masks. The constant multiplier in front of each mask is equal to the sum of the values of its coefficients, as is required to compute an average.

A major use of averaging filters is in the reduction of "irrelevant" detail in an image. By "irrelevant "we mean pixel regions that are small with respect to the size of the filter mask. Figure shows two 3 x 3 smoothing filters. Use of the first filter yields the standard average of the pixels under the mask. This can best be seen by substituting the coefficients of the mask i n

$$R = \frac{1}{9}\sum_{i=1}^{9} z_i,$$

which is the average of the gray levels of the pixels in the 3 x 3 neighborhood defined by the mask.Note that, instead of being 1/9, the coefficients of the filter are all 1's.The idea here is that it is computationally more efficient to have coefficients valued 1. At the end of the filtering process the entire image is divided by 9. An m x n mask would have a normalizing constant equal to 1/mn.
A spatial averaging filter in which all coefficients are equal is sometimes called a box filter.

The second mask shown in Fig. is a little more interesting. This mask yields a so-called weighted average, terminology used to indicate that pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. In the mask shown in Fig. the pixel at the center of the mask is multiplied by a higher value than any other, thus giving this pixel more importance in the calculation of the average. The other pixels are inversely weighted as a function of their distance from the center of the mask. The diagonal terms are further away from the center than the orthogonal neighbors (by a factor of √2) and, thus, are weighed less than these immediate neighbors of the center pixel. The basic strategy behind weighing the center point the highest and then reducing the value of the coefficients as a function of increasing distance from the origin is simply an attempt to reduce blurring in the smoothing process. We could have picked other weights to accomplish the same general objective.

However, the sum of all the coefficients in the mask of Fig. is equal to 16, an attractive feature for computer implementation because it has an integer power of 2. In practice, it is difficult in general to see differences between images smoothed by using either of the masks in Fig. or similar arrangements, because the area these masks span at any one location in an image is so small.
The general implementation for filtering an M x N image with a weighted averaging filter of size m x n (m and n odd) is given by the expression

$$g(x, y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s, t) f(x + s, y + t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s, t)}$$

Order-Statistics Filters:

Order-statistics filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result. The best-known example in this category is the median filter, which, as its name implies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel (the original value of the pixel 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 impulse noise, also called salt-and-pepper noise because of its appearance as white and black dots superimposed on an image.

The median, ε, of a set of values is such that half the values in the set are less than or equal to ε, and half are greater than or equal to ε. In order to perform median filtering at a point in an image, we first sort the values of the pixel in question and its neighbors, determine their median, and assign this value to that pixel. For example, in a 3 x 3 neighborhood the median is the 5th largest value, in a 5 x 5 neighborhood the 13th largest value, and so on. When several values in a neighborhood are the same, all equal values are grouped. For example, suppose that a 3 x 3 neighborhood has values (10, 20, 20, 20, 15, 20, 20, 25, 100). These values are sorted as (10, 15, 20, 20, 20, 20, 20, 25, 100), which results in a median of 20. Thus, the principal function of median filters is to force points with distinct gray levels to be more like their neighbors. In fact, isolated clusters of pixels that are light or dark with respect to their neighbors, and whose area is less than $n^2/2$ (one-half the filter area), are eliminated by an n x n median filter. In this case "eliminated" means forced to the median intensity of the neighbors. Larger clusters are affected considerably less.

Sharpening filters Use of Second Derivatives for Enhancement–The Laplacian:

The approach basically consists of defining a discrete formulation of the second-order derivative and then constructing a filter mask based on that formulation. We are interested in isotropic filters, whose response is independent of the direction of the discontinuities in the image to which the filter is applied. In other words, isotropic filters are rotation invariant, in the sense that rotating the image and then applying the filter gives the same result as applying the filter to the image first and then rotating the result.

Development of the method:

It can be shown (Rosenfeld and Kak [1982]) that the simplest isotropic derivative operator is the Laplacian, which, for a function (image) f(x, y) of two variables, is defined as

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

Because derivatives of any order are linear operations, the Laplacian is a linear operator.

In order to be useful for digital image processing, this equation needs to be expressed in discrete form. There are several ways to define a digital Laplacian using neighborhoods. digital second.Taking into account that we now have two variables, we use the following notation for the partial second-order derivative in the x-direction:

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

and, similarly in the y-direction, as

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

The digital implementation of the two-dimensional Laplacian in Eq. is obtained by summing these two components

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

This equation can be implemented using the mask shown in Fig.(a), which gives an isotropic result for rotations in increments of 90°.

The diagonal directions can be incorporated in the definition of the digital Laplacian by adding two more terms to Eq., one for each of the two diagonal directions.The form of each new term is the same as either Eq.

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

a b
c d

Fig.. (a) Filter mask used to implement the digital Laplacian (b) Mask used to implement an extension of this equation that includes the diagonal neighbors. (c) and (d) Two other implementations of the Laplacian.

But the coordinates are along the diagonals. Since each diagonal term also contains a –2f(x, y) term, the total subtracted from the difference terms now would be –8f(x, y). The mask used to implement this new definition is shown in Fig.(b). This mask yields isotropic results

for increments of 45°. The other two masks shown in Fig. 11 also are used frequently in practice.

They are based on a definition of the Laplacian that is the negative of the one we used here. As such, they yield equivalent results, but the difference in sign must be kept in mind when combining (by addition or subtraction) a Laplacian-filtered image with another image. Because the Laplacian is a derivative operator, its use highlights gray-level discontinuities in an image and deemphasizes regions with slowly varying gray levels. This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be "recovered" while still preserving the sharpening effect of the Laplacian operation simply by adding the original and Laplacian images. As noted in the previous paragraph, it is important to keep in mind which definition of the Laplacian is used. If the definition used has a negative center coefficient, then we subtract, rather than add, the Laplacian image to obtain a sharpened result. Thus, the basic way in which we use the Laplacian for image enhancement is as follows:

$$g(x, y) = \begin{cases} f(x, y) - \nabla^2 f(x, y) & \text{if the center coefficient of the Laplacian mask is negative} \\ f(x, y) + \nabla^2 f(x, y) & \text{if the center coefficient of the Laplacian mask is positive.} \end{cases}$$

Use of First Derivatives for Enhancement—The Gradient:

First derivatives in image processing are implemented using the magnitude of the gradient. For a function f(x, y), the gradient of f at coordinates (x, y) is defined as the two-dimensional column vector

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}.$$

The magnitude of this vector is given by

$$\begin{aligned} \nabla f &= \text{mag}(\nabla \mathbf{f}) \\ &= \left[G_x^2 + G_y^2\right]^{1/2} \\ &= \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2}. \end{aligned}$$

The components of the gradient vector itself are linear operators, but the magnitude of this vector obviously is not because of the squaring and square root operations. On the other hand, the partial derivatives are not rotation invariant (isotropic), but the magnitude of the

gradient vector is. Although it is not strictly correct, the magnitude of the gradient vector often is referred to as the gradient.

The computational burden of implementing over an entire image is not trivial, and it is common practice to approximate the magnitude of the gradient by using absolute values instead of squares and square roots:

$$\nabla f \approx |G_x| + |G_y|.$$

This equation is simpler to compute and it still preserves relative changes in gray levels, but the isotropic feature property is lost in general. However, as in the case of the Laplacian, the isotropic properties of the digital gradient defined in the following paragraph are preserved only for a limited number of rotational increments that depend on the masks used to approximate the derivatives. As it turns out, the most popular masks used to approximate the gradient give the same result only for vertical and horizontal edges and thus the isotropic properties of the gradient are preserved only for multiples of 90°.

As in the case of the Laplacian, we now define digital approximations to the preceding equations, and from there formulate the appropriate filter masks. In order to simplify the discussion that follows, we will use the notation in Fig. 11.2 (a) to denote image points in a 3 x 3 region. For example, the center point, z5 , denotes f(x, y), z1 denotes f(x- 1, y-1), and so on. The simplest approximations to a first-order derivative that satisfy the conditions stated in that section are $G_X = (z_8 - z_5)$ and $G_Y = (z_6 - z_5)$. Two other definitions proposed by Roberts [1965] in the early development of digital image processing use cross differences:

$$G_x = (z_9 - z_5) \quad \text{and} \quad G_y = (z_8 - z_6).$$

we compute the gradient as

$$\nabla f = [(z_9 - z_5)^2 + (z_8 - z_6)^2]^{1/2}$$

If we use absolute values, then substituting the quantities in the equations gives us the following approximation to the gradient:

$$\nabla f \approx |z_9 - z_5| + |z_8 - z_6|.$$

This equation can be implemented with the two masks shown in Figs. (b) and(c). These masks are referred to as the Roberts cross-gradient operators. Masks of even size are awkward to implement. The smallest filter mask in which we are interested is of size 3 x 3.An approximation using absolute values, still at point z_5 , but using a 3*3 mask, is

$$\begin{aligned}\nabla f \approx\ & |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| \\ & + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|.\end{aligned}$$

The difference between the third and first rows of the 3 x 3 image region

approximates the derivative in the x-direction, and the difference between the third and first columns approximates the derivative in the y-direction. The masks shown in Figs. (d) and (e), called the Sobel operators. The idea behind using a weight value of 2 is to achieve some smoothing by giving more importance to the center point. Note that the coefficients in all the masks shown in Fig. 11.2 sum to 0, indicating that they would give a response of 0 in an area of constant gray level, as expected of a derivative operator.

Fig. A 3 x 3 region of an image (the z's are gray-level values) and masks used to compute the gradient at point labeled z_5 . All masks coefficients sum to zero, as expected of a derivative operator.

Fourier's contribution in this particular field states that any function that periodically repeats itself can be expressed as the sum of sines and/or cosines of different frequencies, each multiplied by a different coefficient (we now call this sum a Fourier series). It does not matter how complicated the function is; as long as it is periodic and meets some mild mathematical conditions, it can be represented by such a sum.

Even functions that are not periodic (but whose area under the curve is finite) can be expressed as the integral of sines and/or cosines multiplied by a weighing function. The formulation in this case is the Fourier transform, and its utility is even greater than the Fourier series in most practical problems. Both representations share the important characteristic that a function, expressed in either a Fourier series or transform, can be reconstructed (recovered) completely via an inverse process, with no loss of information. This is one of the most important characteristics of these representations because they allow us to work in the "Fourier domain" and then return to the original domain of the function without losing any information.

The application of Fourier initial ideas was in the field of heat diffusion, where they allowed the formulation of differential equations representing heat flow in such a way that solutions could be obtained for the first time. During the past century, and especially in the past 50 years, entire industries and academic disciplines have flourished as a result of Fourier's ideas. The advent of digital computation and the "discovery" of a fast Fourier transform (FFT) algorithm in the late 1950s (more about this later) revolutionized the field of signal processing. These two core technologies allowed for the first time practical processing and meaningful interpretation of a host of signals of exceptional human and industrial importance, from medical monitors and scanners to modern electronic communications.

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

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

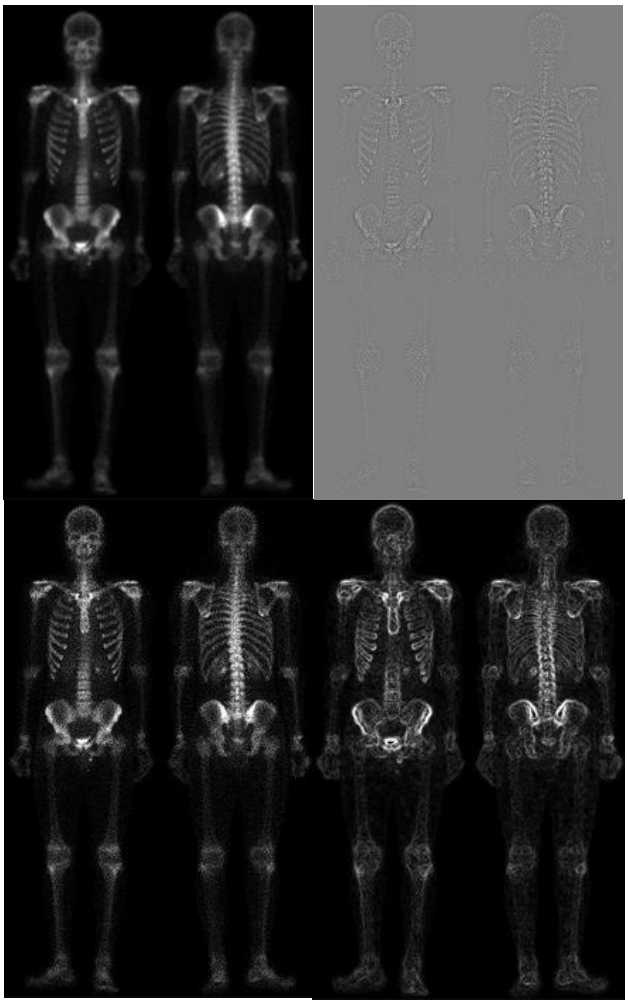
a
b c
d e

Combining Spatial Enhancement Methods

In this section we illustrate by means of an example how to combine several of the approaches developed in this chapter to address a difficult enhancement task.

The image shown in Fig. (a) is a nuclear whole body bone scan, used to detect diseases such as bone infection and tumors. Our objective is to enhance this image by sharpening it and by bringing out more of the skeletal detail. The narrow dynamic range of the gray levels and high noise content make this image difficult to enhance. The strategy we will follow is to utilize the Laplacian to highlight fine detail, and the gradient to enhance prominent edges. For reasons that will be explained shortly, a smoothed version of the gradient image will be used to mask the Laplacian image. Finally, we will attempt to increase the dynamic range of the gray levels by using a gray-level transformation.

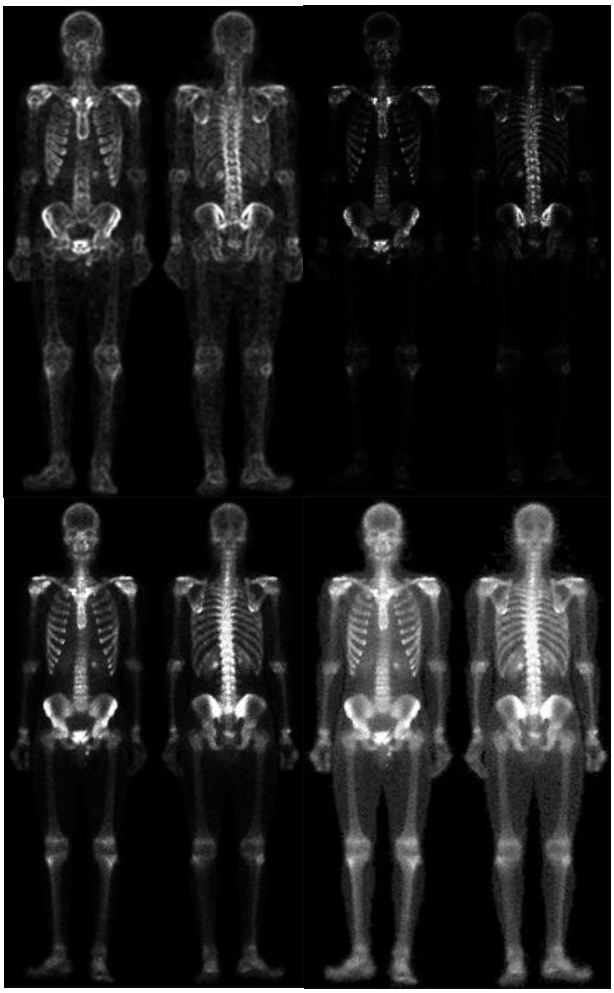
Figure (b) shows the Laplacian of the original image, obtained using the mask in Fig. (d). This image was scaled (for display only) using the same technique as in Fig. We can obtain a sharpened image at this point



a b
c d

FIGURE

Image of whole body bone scan.
Laplacian of (a).
(c) Sharpened image
obtained by adding (a) and (b).
(d) Sobel of (a).



e f
g h

FIGURE
(Continued)
(e) Sobel image smoothed With a 5 × 5 averaging filter.
(f) Mask image formed By the product of (c) and (e).
(g) Sharpened image obtained By the sum of (a) and (f).
(h) Final result obtained by applying A power-law transformation to (g). Compare (g) and (h) with (a).

Simply by adding Figs. (a) and (b), which are an implementation of the second line in Eq. (3.7-5) (we used a mask with a positive center coefficient). Just by looking at the noise level in (b), we would expect a rather noisy sharpened image if we added Figs. (a) and (b), a fact that is confirmed by the result shown in Fig. (c). One way that comes immediately to mind to re- duce the noise is to use a median filter. However, median filtering is a non- linear process capable of removing image features. This is unacceptable in medical image processing.

An alternate approach is to use a mask formed from a smoothed version of the gradient of the original image. The motivation behind this is straightforward and is based on the properties of first- and second-order derivatives. The Laplacian, being a second-order derivative operator, has the definite advantage that it is superior in enhancing fine detail. However, this causes it to produce noisier results than the gradient. This noise is most objectionable in smooth areas, where it tends to be more visible. The gradient has a stronger response in areas of significant gray-level transitions (gray-level ramps and steps) than does the Laplacian. The response of the gradient to noise and fine detail is lower than the Laplacian's and can be lowered further by smoothing the gradient with an averaging filter. The idea, then, is to smooth the gradient and multiply it by the Laplacian image. In this context, we may view the smoothed gradient as a mask image. The product will preserve details in the

strong areas while reducing noise in the relatively flat areas. This process can be viewed roughly as combining the best features of the Laplacian and the gradient.

The result is added to the original to obtain a final sharpened image, and could even be used in boost filtering. Figure (d) shows the Sobel gradient of the original image, computed. Components G_x and G_y were obtained using the masks in Figs. respectively. Edges are much more dominant in this image than in the Laplacian image. The smoothed gradient image shown in Fig. (e) was obtained by using an averaging filter of size 5×5. The two gradient images were scaled for display in the same manner as the two Laplacian images. Because the smallest possible value of a gradient image is 0, the background is black in the scaled gradient images, rather than gray as in the scaled Laplacian. The fact that Figs. (d) and are much brighter than Fig. (b) is again evidence that the gradient of an image with significant edge content has values that are higher in general than in a Laplacian image.

The product of the Laplacian and smoothed-gradient image is shown in Fig. (f). Note the dominance of the strong edges and the relative lack of visible noise, which is the key objective behind masking the Laplacian with a smoothed gradient image. Adding the product image to the original resulted in the sharpened image shown in Fig. (g). The significant increase in sharp- ness of detail in this image over the original is evident in most parts of the image, including the ribs, spinal chord, pelvis, and skull. This type of improvement would not have been possible by using the Laplacian or gradient alone.

Filtering in Frequency domain:

Preliminary concepts:

Basics of filtering in the frequency domain

Filtering in the frequency domain is straightforward. It consists of the following steps:

1. Multiply the input image by $(-1)^{x+y}$ to center the transform, as indicated in Eq.

2. Compute *F(u, v),* the DFT of the image from (1).

3. Multiply F (u, v) by a filter function H (u, v).

4. Compute the inverse DFT of the result in (3).

5. Obtain the real part of the result in (4).

6. Multiply the result in (5) by $(-l)^{x+y}$.

The reason that H(u, *v)* is called a filter because it suppresses certain frequencies in the transform

while leaving others unchanged. The analogy from everyday life is a screen filter that passes certain objects and suppresses others, based strictly on their size.

In equation form, let f *(x,* y) represent the input image in Step 1 and *F(u,* v) its Fourier transfom. Then the Fourier transform of the output image is given by

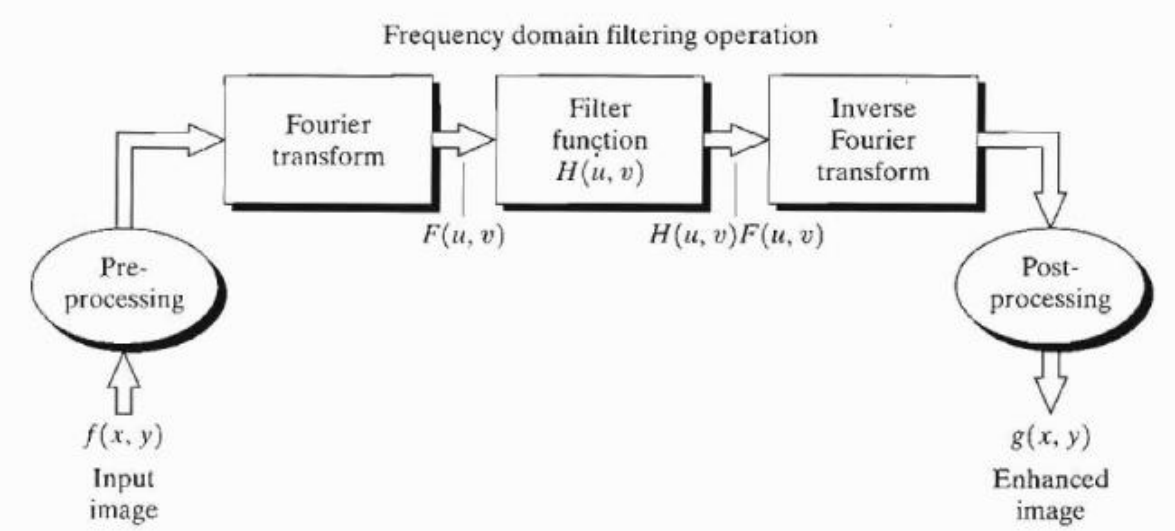
$$G(u, v) = H(u, v)F(u, v).$$

The multiplication of H and F involves two-dimensional functions and is defined on an element-by-element basis. That is, the first element of H multiplies the first element of *F,* the second element of H multiplies the second element of *F,* and so on. each component of H multiplies both the real and imaginary parts of the corresponding component in F. Such filters are called zero-phase-shift filters. As their name implies, these filters do not change the phase of the transform, a fact that can be seen in Eq by noting that the multiplier of the real and imaginary parts would cancel out because they have the same value.

The filtered image is obtained simply by taking the inverse Fourier transform of *G(u, v):*

$$\text{Filtered Image} = \Im^{-1}[G(u, v)].$$

The final image is obtained by taking the real part of this result and rnultiplying it by $(-1)^{x+y}$ to cancel the multiplication of the input image by this quantity. The inverse Fourier transform is, in general, complex. when the input image and the filter function are real, the imaginary components of the inverse transform should all be zero. the inverse DFT generally has parasitic imaginary components due to computational round-off errors. These components are ignored.



The filtering procedure just outlined is summarized in Fig. in a slightly more general form that includes pre- and post processing stages. In addition to the $(-1)^{x+y}$ examples of other processes might include cropping of the input image to its closest even dimensions (required for proper transform centering), gray-level scaling, conversion to floating point on input, and conversion to an 8-bit integer format on the output. Multiple filtering stages and other pre- and post processing functions are possible. The important point to keep in mind is that the filtering process ***is*** based on modifying the transform of an image in some way via a filter function and then taking the inverse of the result to obtain the processed output image.

Some basic filters and their properties

Suppose that we wish to force the average value of an image to zero. According to Eq), the average value of an image is given by *F(0,0).* If we set this term to zero in the frequency domain and take the inverse transform, then the average value of the resulting image will be zero. Assuming **that** the transform has been centered this operation by multiplying **all** values ***of F(u, v)*** **by** the filter function:

$$H(u, v) = \begin{cases} 0 & \text{if } (u, v) = (M/2, N/2) \\ 1 & \text{otherwise.} \end{cases}$$

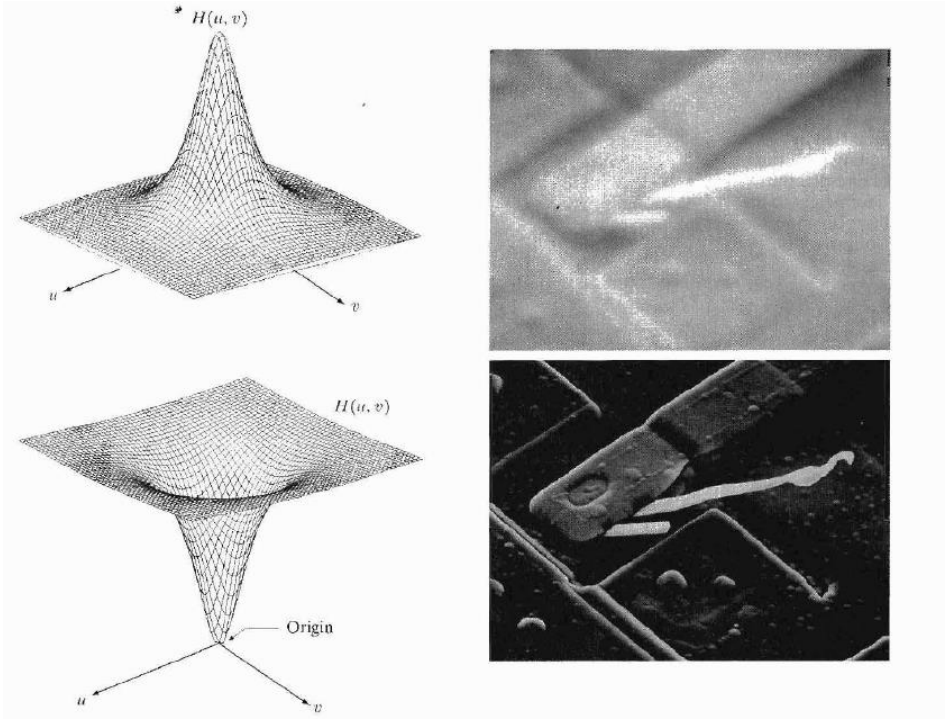
All this filter would do is set *F(0,0)* to zero and leave all other frequency components of the Fourier transform untouched, as desired. The processed image (with zero average value) can then be obtained by taking the inverse Fourier transform of *H(u, v)F(u, v),* as indicated in Eq. As stated earlier, both the real and imaginary parts of *F(u,* 2)) are multiplied by the filter function *H (u,* v). The filter is called a ***notch filter*** because it is a constant function **with** a hole (notch) at the origin. the drop in overall average gray level resulting from forcing **the** average **value** to **zero;** note **also** the "byproduct" result of making prominent edges stand out.

notch filters are exceptionally useful tools when it is possible to identify spatial image effects caused by specific, localized frequency domain components.

Low frequencies in the Fourier transform are responsible for the general gray-level appearance of an image over smooth areas. while high frequencies are responsible for detail, such as edges and noise. These ideas are discussed in more detail in the sections that follow, but it will be instructive to complement our illustration of the notch filter with an example of filters in these other two categories.

A filter that attenuates high frequencies while "passing" low frequencies is called a *lowpass* ***filter.*** A filter that has the opposite characteristic is appropriately called a *highpass* filter. We would expect a lowpass-filtered image to have less sharp detail than the original because the high frequencies have been attenuated. Similarly, a highpass-filtered image would have less gray level variations in smooth areas and emphasized transitional (e.g., edge) gray-level detail. Such an image will appear sharper.

The filters, *H(u, v),* shown are both circularly symmetric. After shifting their origin to the center of the frequency rectangle occupied by *F(u,* v), they were multiplied by the centered transform. Taking the real part of each result and multiplying it by $(-1)^{x+y}$ yielded the images on the right. As expected, the image in Fig. 4.7(b) is blurred, and the image in Fig.4.7(d) is sharp, with little smooth gray-level detail because the *F(0,O)* term has been set to zero. This is typical of high passed results, and a procedure often followed is to add a constant to the filter so that it will not completely eliminate *F(0,* 0).



Correspondence between Filtering in the Spatial and Frequency Domains

The most fundamental relationship between the spatial and frequency domains is established by a well-known result called the ***convolution*** *theorem.*

The process by which we move a mask from pixel to pixel in an image, and compute a predefined quantity at each pixel, is the foundation of the convolution process. Formally, the discrete convolution of two functions f ***(x ,*** y) and ***h(x,*** y) of size M ***X N*** is denoted by f **(x,** *y)* * ***h(x,*** y) and is defined by the expression

$$f(x, y) * h(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x - m, y - n).$$

With the exception of the leading constant, the minus signs, and the limits of the summation, this expression is similar in ***form*** to Eq. (3.5-1). The minus signs, in particular, simply mean that function h is mirrored about the origin. This is inherent in the definition of convolution. Equation (4.2-30) is really nothing more than an implementation for (1) flipping one function about the origin; (2) shifting that function with respect to the other by changing the values of ***(x ,*** *y);* and **(3)** computing a sum of products over all values of *m* and n, for *each* displacement *(x , y).* The displacements *(x,* y) are integer increments that stop when the functions no longer overlap.

Letting ***F(u,*** *u)* and *H(u,* ***v)*** denote the Fourier transforms off ***(x .*** y) and ***h(x,*** y), respectively, one-half of the convolution theorem simply states that f (x, *y)* * ***h(x,*** y) and *F(u, v)H(u,* ***v)*** constitute a Fourier transform pair. This result is formally stated as

$$f(x, y) * h(x, y) \Leftrightarrow F(u, v) H(u, v).$$

The double arrow is used to indicate that the expression on the left (spatial convolution) can be obtained by taking the inverse Fourier transform of the expression on the right [the product *F(u,* ***v)H(u,*** v) in the frequency domain]. Conversely, The expression on the right can be obtained by taking the ***forward*** Fourier transform of the expression on the **left-An** analogous result is that convolution in the frequency domain reduces to multiplication in the spatial domain, and vice versa; that is,

$$f(x, y) h(x, y) \Leftrightarrow F(u, v) * H(u, v).$$

An *impulse function* of *strength A,* located at coordinates (x0,y0), is denoted by *Aδ(X - xo,* y - *yo)* and is *defined* by the expression

$$\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} s(x, y) A\delta(x - x_0, y - y_0) = As(x_0, y_0).$$

This equation states that the summation of a function *s(x, y*) multiplied by an impulse is simply the value of the function at the location of the impulse, multiplied by the strength of the impulse. It is understood that the limits of the summation are the same as the limits spanned by the function. We point out that *Aδ (x - xo,* y - *yo* also is an image of *size M* x *N* . It is composed of all zeros, except at coordinates *(x0,yo),*w here the value of the image is *A.*

Convolution of a function with an impulse "copies" the value of that function at the location of the impulse. This characteristic is called the *shifting property* of the impulse function. Of particular importance at the moment is the case of a unit impulse located at the origin, which is denoted as *S(x,* y). In this case,

$$\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} s(x, y) \delta(x, y) = s(0, 0).$$

We can compute the Fourier transform of a unit impulse at the **origin**

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\delta(x,y)e^{-j2\pi(ux/M+vy/N)}$$
$$= \frac{1}{MN}$$

The Fourier transform of an impulse at the origin of the spatial domain *is* a *real* constant (this means that the phase angle is zero). If the impulse were located elsewhere, the transform would have complex components. The magnitude would be the same, with the translation of the impulse being reflected in a nonzero phase angle in the transform.

let *f (x, y) = δ (x, y}* and carry out the convolution, gives us

$$f(x,y) * h(x,y) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\delta(m,n)h(x-m,y-n)$$
$$= \frac{1}{MN}h(x,y)$$

By combining the results of last two Eqs, we obtain

$$f(x,y) * h(x,y) \Leftrightarrow F(u,v)H(u,v)$$
$$\delta(x,y) * h(x,y) \Leftrightarrow \Im[\delta(x,y)]H(u,v)$$
$$h(x,y) \Leftrightarrow H(u,v).$$

Using only the properties of the impulse function and the convolution theo*rem*, we have established that filters in the spatial and frequency domains constitute a Fourier transform pair. Thus, given a filter in the frequency domain, we can obtain the corresponding filter in the spatial domain by taking the inverse Fourier transform of the former. The reverse also is true.

Therefore, in practice, specifying a filter in the frequency domain and then taking the inverse transform to compute an equivalent spatial domain filter of the same size does not really help matters from a computational point of view. if both filters are of the same size, it generally is more efficient computationally to do the filtering in the frequency domain. But we use much smaller filters in the spatial domain.

Filtering often is more intuitive in the frequency domain. However, whenever possible, it makes more sense to filter in the spatial domain using small filter masks. Equation (4.2-37) tells us that we can specify filters in the frequency domain, take their inverse transform, and then use the resulting filter in the spatial domain as a guide for constructing smaller spatial filter masks. Keep in mind during the following discussion that the Fourier transform and its inverse are linear processes, *so* the discussion *is* by definition limited to linear filtering.

Filters based on Gaussian functions are of particular importance because their shapes are **easily** specified and because both the **forward** and inverse Fouri**er**

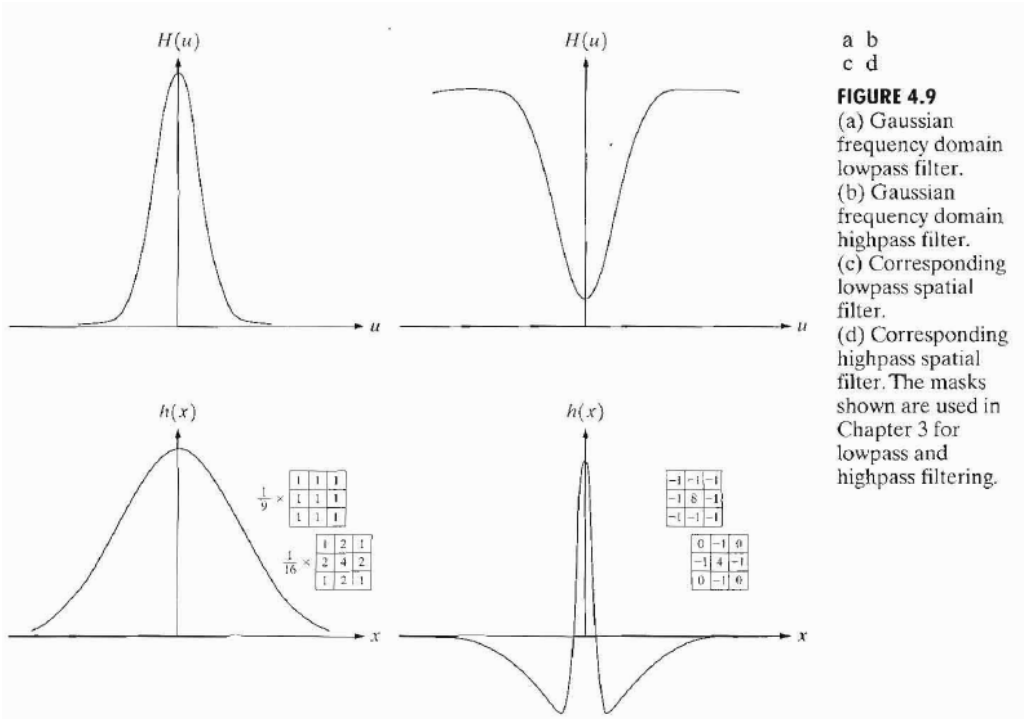
transforms of a Gaussian function are **real** Gaussian functions.
Let *H(u)* denote a frequency domain, Gaussian filter function given by the **equation**

$$H(u) = Ae^{-u^2/2\sigma^2}$$

where σ is **the** standard deviation of the **Gaussian curve.** It can be shown **that** the corresponding filter in the spatial domain is

$$h(x) = \sqrt{2\pi}\sigma Ae^{-2\pi^2\sigma^2x^2}.$$

These two equations represent an important result for two reasons: (1) They constitute a Fourier transform pair, both components of which are Gaussian *and* real. This facilitates analysis considerably because we do not have to be concerned with complex numbers. In addition. Gaussian curves are intuitive and easy to manipulate. (2)these functions behave reciprocally with respect to one another. In other words, when *H (u)* has a broad profile (large value of *g), h (x)* has a narrow profile, and vice versa. In fact, when *a* approaches infinity, *H(u)* tends toward a constant function and *h(.x)* tends toward an impulse.



a b
c d
FIGURE 4.9
(a) Gaussian frequency domain lowpass filter.
(b) Gaussian frequency domain highpass filter.
(c) Corresponding lowpass spatial filter.
(d) Corresponding highpass spatial filter. The masks shown are used in Chapter 3 for lowpass and highpass filtering.

A glaring similarity between the two filters is that all the values are positive in both domains. Thus, we arrive at the conclusion that we can implement lowpass filtering in the spatial domain by using a mask with all positive coefficients. Another important characteristic is the reciprocal relationship. The narrower the frequency domain filter, the more it will attenuate the low frequencies, resulting in increased blurring. In the spatial domain this means a wider filter, which in turn implies a larger mask.

More complex filters can be constructed from the basic Gaussian function. For instance, we can construct a highpass filter as a difference of Gaussians, as follows:

$$H(u) = Ae^{-u^2/2\sigma_1^2} - Be^{-u^2/2\sigma_2^2}$$

with A>= B and σ1 > σ2.The corresponding filter in the spatial domain is

$$h(x) = \sqrt{2\pi}\sigma_1 Ae^{-2\pi^2\sigma_1^2 x^2} - \sqrt{2\pi}\sigma_2 Be^{-2\pi^2\sigma_2^2 x^2}.$$

A question that often arises at this point in the development of frequency domain techniques is the issue of computational complexity. Why do in the frequency domain what could be done (at least partially) in the spatial domain using small spatial masks? The basic answer is twofold. First, as we have seen, the frequency domain carries with it a significant degree of intuitiveness regarding how to specify filters. The second part of the answer depends on the size of the spatial masks and is usually answered with respect to comparable implementations.

A benchmark used frequently for this purpose is implementation ***of*** convolution in the spatial and frequency domains. we know from the convolution theorem that we can obtain the same result via the frequency domain by taking the inverse transform of the product of the transforms of the two functions.

Smoothing Frequency-Domain Filters

Edges and other sharp transitions (such as noise) in the gray levels of an image contribute significantly to the high-frequency content of its Fourier transform. Hence smoothing (blurring) is achieved in the frequency domain by attenuating a specified range of high-frequency components in the transform of a given image.
Our basic "model" for filtering in the frequency domain is given by

$$G(u, v) = H(u, v)F(u, v)$$

where F(u, v) is the Fourier transform of the image to be smoothed. The objective is to select a filter transfer function H(u, v) that yields *G(u, v)* by attenuating the high-frequency components of F(u, *v).*
three types of lowpass filters: ideal, Butterworth, and Gaussian filters. These three filters cover the range from very sharp (ideal) to very smooth (Gaussian) filter functions. The Butterworth filter has a parameter, called the filter ***order.*** For high values of this parameter the Butterworth filter approaches the form of the ideal filter. For lower-order values, the Butteworth filter has a smooth form similar to the Gaussian filter. Thus, the Butterworth filter may be viewed as a transition between two "extremes."

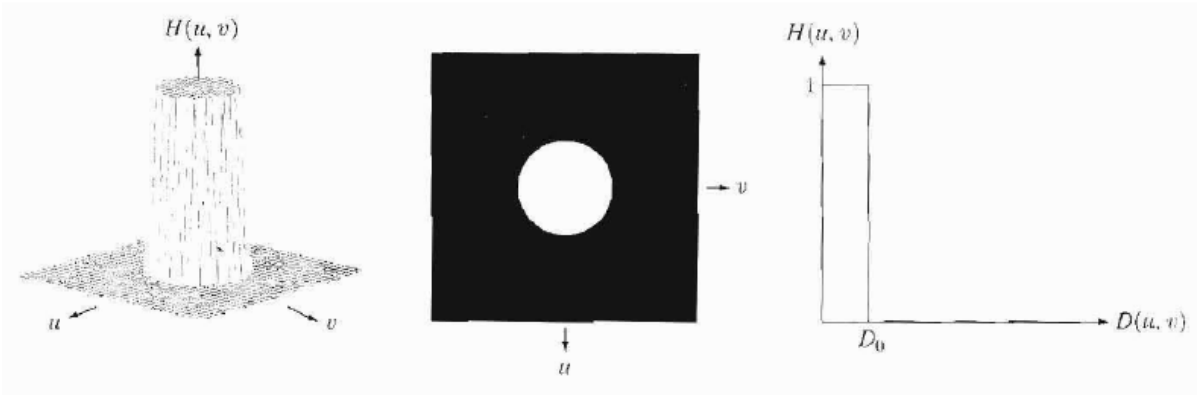
Ideal Lowpass Filters

'The simplest lowpass filter we can envision is a filter that "cuts off" all high frequency components of the Fourier transform that are at a distance greater than a specified distance *D,* from the origin of the (centered) transform. Such a filter is called a two-dimensional (2-D) ***ideal lowpass*** filter ***(ILPF)*** and has the transfer function

$$H(u, v) = \begin{cases} 1 & \text{if } D(u, v) \leq D_0 \\ 0 & \text{if } D(u, v) > D_0 \end{cases}$$

where Do is a specified nonnegative quantity, and D(u, *v)* is the distance from point (u, *v)* to the center of the frequency rectangle. If the image in question is of size M X N, we know that its transform also is of this size, so the center of the frequency rectangle is at (u, *v) = (M/2,* N/2) due to the fact that the transform has been centered. the distance from any point (u, *v)* to the center (origin) of the Fourier transform is given by

$$D(u, v) = \left[(u - M/2)^2 + (v - N/2)^2\right]^{1/2}.$$



The name *ideal filter* indicates that ***all*** frequencies inside a circle of radius D,, are passed with no attenuation, whereas all frequencies outside this circle are completely attenuated. The lowpass filters considered in this chapter are radially symmetric about the origin. This means that a cross section extending as a function of distance from the origin along a radial line is sufficient to specify the filter. The complete filter transfer function can be visualized by rotating the cross section 360" about the origin.

For an ideal lowpass filter cross section, the point of transition between ***H(u,*** v) = ***1*** and H***(u,v***) = 0 is called the ***cutoff frequencies.***
The lowpass filters introduced in this section are compared by studying their behavior as a function of the same cutoff frequencies. One way to establish a set of standard cutoff frequency loci is to compute circles that enclose specified amounts of total image power P1. This quantity is obtained by summing the components of the power spectrum at each point ***(u:*** *v),* for u = 0, 1,2.. . . . M – 1 and v=0,1,2 ,..., N-1.that is,

$$P_T = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} P(u, v)$$

If the transform has been centered, a circle of radius ***v*** with origin at the center of the frequency rectangle encloses *n* percent of the power, where

$$\alpha = 100\left[\sum_{u} \sum_{v} P(u, v)/P_T\right]$$

and the summation is taken over the values of ***(u.*** v) that lie inside the circle or on its boundary.

The blurring **and** ringing properties of the ILPF can be explained by reference

to the convolution theorem. The Fourier transforms of the original **image** f **(x, *y)*** and **the** blurred **image *g(x, y)* axe** related in the frequency domain by the equation.

$$G(u, v) = H(u, v)F(u, v)$$

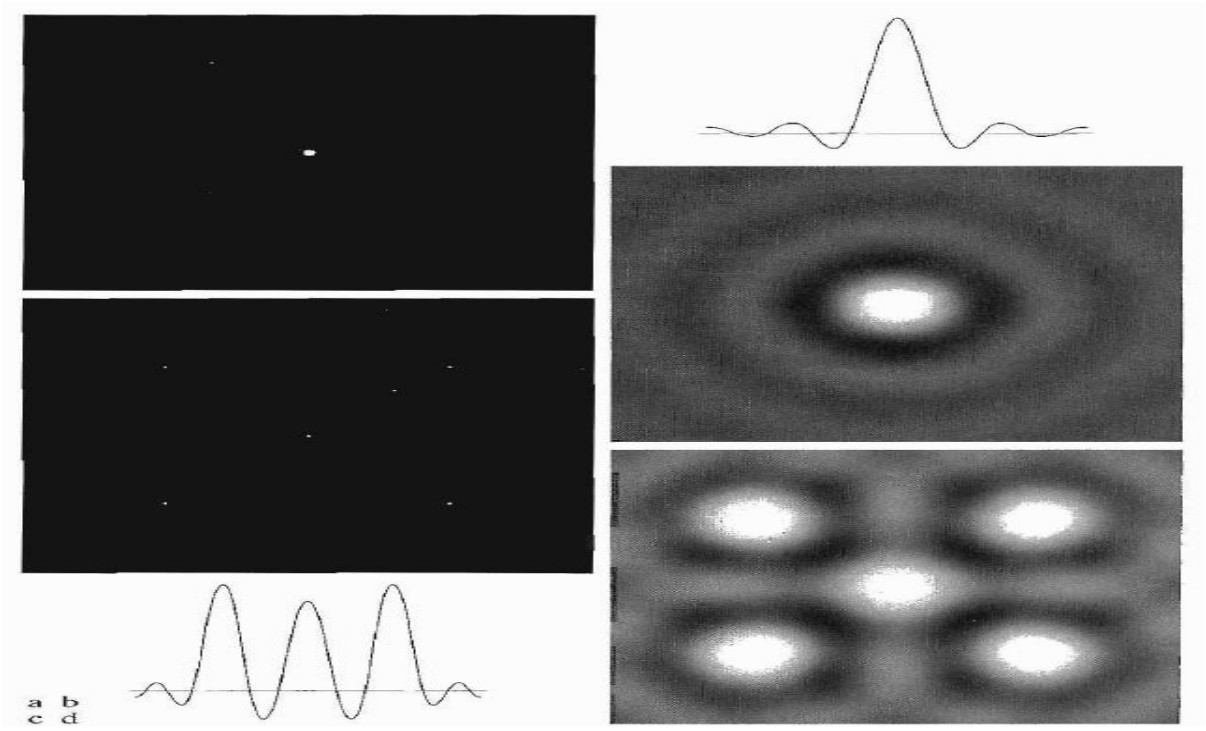
where, as before, ***H(u,*** *u)* is the filter function and F **and *G*** are **the** Fourier transforms of the two images just mentioned. The convolution theorem tells us that the corresponding process in the spatial domain is

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

where ***h(x,*** y) is the inverse Fourier transform of the filter transfer function ***H*** *(u,* v).

The **key** to understanding blurring as a convolution process in the spatial domain lies in the nature of ***h(x,*** y). This is the **function** *H(u,* ***v)*** in the ***frequency*** domain, The spatial filter function ***h(x, y)*** was obtained in the standard way: (1) *H(u,* v) **was** multiplied by $(-1)^{u+v}$ for centering; (2) this was followed by **the** inverse Dm, and (3) the real part of the inverse DFT **was** multiplied by $(-1)^{x+y}$. We see that the filter *h(x, y)* has two major distinctive characteristics: a dominant component at the origin, and concentric, circular components about the center component. The center component is primarily responsible for blurring. The concentric components are responsible primarily for the ringing characteristic of ideal filters, Both the radius of the center component and the number of circles per unit distance from the origin are inversely proportional to the value of the cutoff frequency of the ideal filter. The insert at the top is a gray level profile of a horizontal scan line through the center of the spatial filter. The axis shown indicates zero amplitude, so we see that the spatial filter has negative values. This normally is not a serious problem because the larger center component dominates the convolution result. However, the filtered image can have negative values, so scaling normally is required.

Suppose next that f *(x , y*) is a simple image composed of five bright pixels on a black background, as Fig. (c) shows. These bright points may be viewed as approximations to impulses, whose strength depends on the intensity of the points. Then the convolution of *h(x,* y) and *f (x , y)* is simply a process of "copying" *h(x,*y) at the location of each impulse, as noted in Section .T he result of this operation, shown in Fig. (d), explains how the original points are blurred as a consequence of convolving *f (x , y)* with the blurring filter function *h(x,* y). Note also that ringing was introduced during the same process. In fact, the ringing is *so* severe in this case that distortion is *caused* by their interference with one another. These concepts are extended conceptually to more complex images by considering each pixel as an impulse whose strength is proportional to the gray level of the pixel. The insert at the bottom of Fig. shows the gray-level profile of a diagonal scan line through the center of the filtered image.

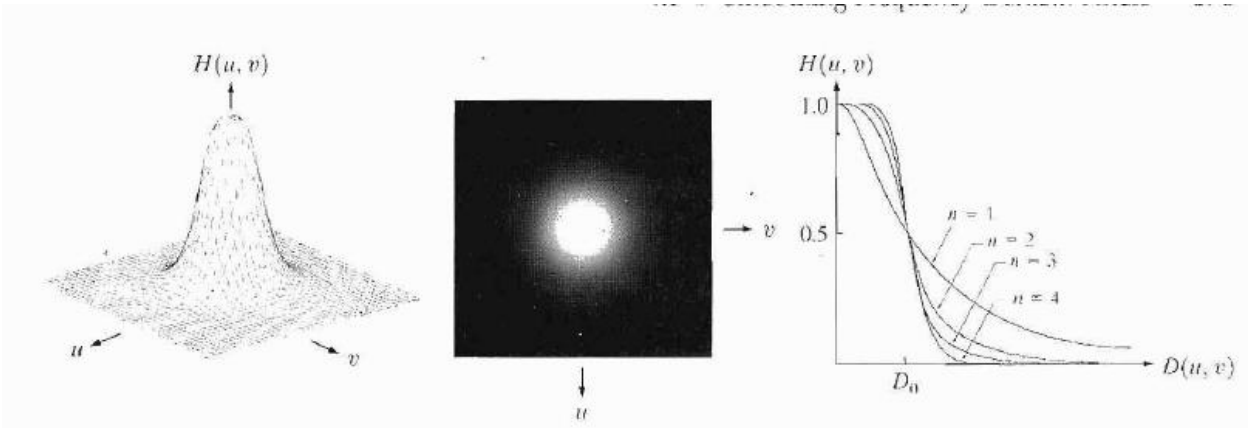


Butterworth Lowpass Filters

The transfer function of a Butterworth lowpass filter (BLPF) of order n, and with cutoff frequency at a distance *Do,* from the origin, is defined as

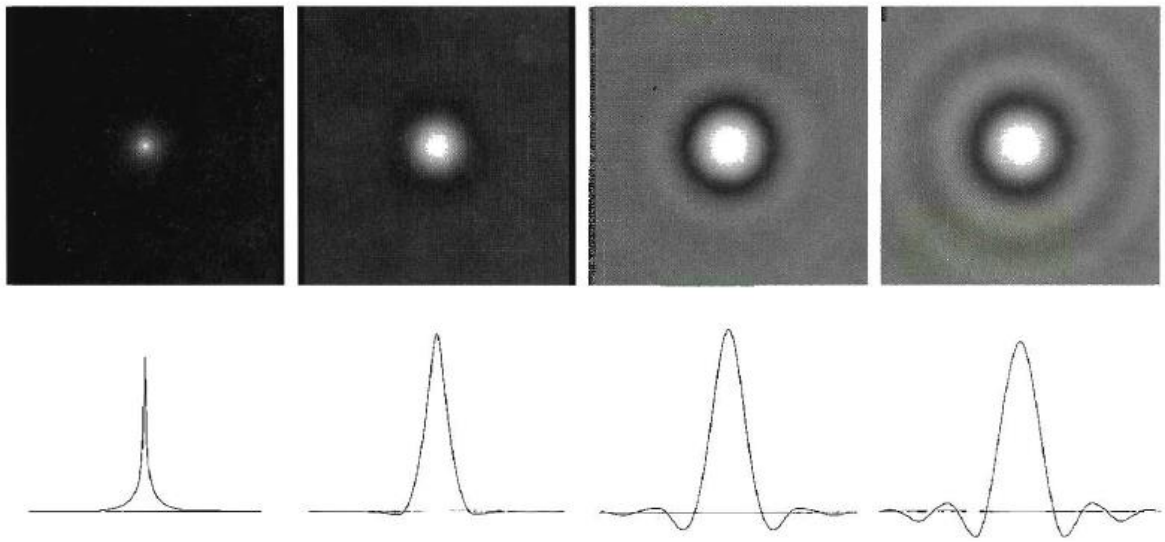
$$H(u,v) = \frac{1}{1 + \left[D(u,v)/D_0\right]^{2n}}$$

Unlike the ILPF, the BLPF transfer function does not have a sharp discontinuity that establishes a clear cutoff between passed and filtered frequencies. For filters with smooth transfer functions, defining a cutoff frequency locus at points for which *H(u,* v) is down to a certain fractional its maximum value is customary. *H(u, v)* = 0.5 (down *50%* from its maximum value of **2)** when *D(u,v) = Do.*



A Butterworth filter of order I has no ringing. Ringing generally is imperceptible in filters of order 2, but can become a significant factor in filters of higher order. Figure shows an interesting comparison between the *spatial* representations of BLPFs of various orders (with cutoff frequency of *5* pixels). Shown also is the gray-level profile along a horizontal scan line through the center of each fi1ter.

In order to facilitate comparisons additional enhancing with a gamma transformation was applied to the images of Fig. to accentuate even more the components further away from the origin. The BLPF of order 1 [Fig. 4 has neither ringing nor negative values. The filter of order 2 does show mild ringing and small negative values, **but** they certainly are less pronounced than in the ILPF, As the remaining images show, ringing in the BLPF becomes significant for higher-order filters, A Butterworth filter of order 20 already exhibits the characteristics of the 1LPE which can be seen by comparing Figs. (d) and (b). In the limit, both filters are identical. In general, BLPFs of order 2 are a good compromise between effective lowpass filtering and acceptable ringing characteristics,

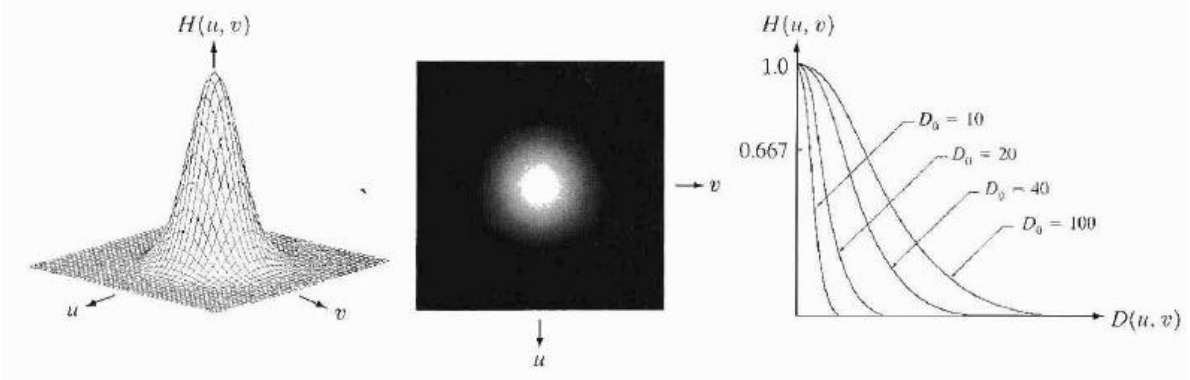


Gaussian Lowpass Filters

The form of these filters in two dimensions is given by

$$H(u, v) = e^{-D^2(u,v)/2\sigma^2}$$

D(u, v) is the distance from **the** origin of the Fourier transform, which we assume has been shifted to the center of the frequency rectangle using the procedure



σ is a measure of the spread of the Gaussian curve. By letting $\sigma = Do$, we can express the filter in a more familiar form in terms of the notation in this section:

$$H(u, v) = e^{-D^2(u,v)/2D_0^2}$$

where Do, is the cutoff frequency. When D(u. v) = Do, the filter is down to

0.607 of its maximum value. The inverse Fourier transform of the Gaussian lowpass filter also is Gaussian.

Sharpening Frequency Domain Filters

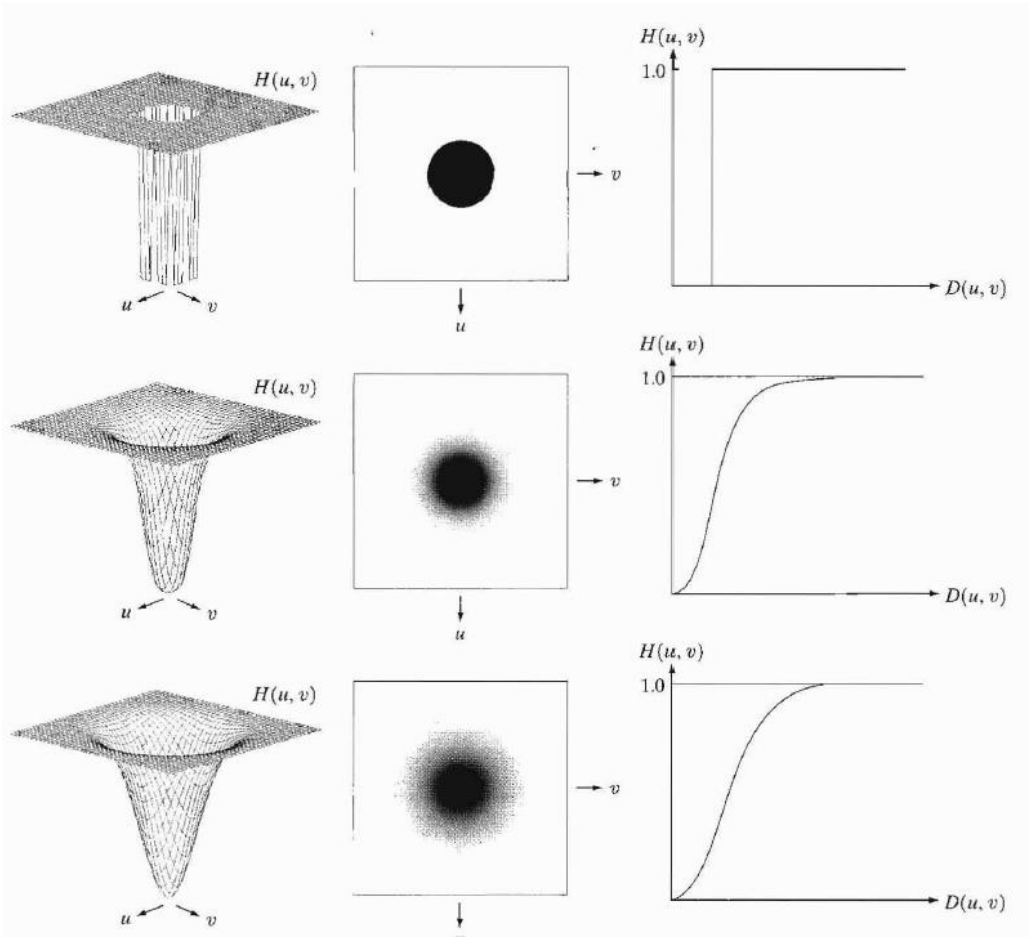
An image can be blurred by attenuating the high-frequency components of its Fourier transform. Because edges and other abrupt changes in gray levels are associated with high-frequency components, image sharpening can be achieved in the frequency domain by a *high**pass filtering*** process, which attenuates the low-frequency components without disturbing high-frequency information in the Fourier transform. We consider only zero-phase-shift filters that are radially symmetric.

Because the intended function of the filters in this section is to perform precisely the reverse operation of the ideal lowpass filters discussed in the previous section, the transfer function of the highpass filters discussed in this section can be obtained using the relation

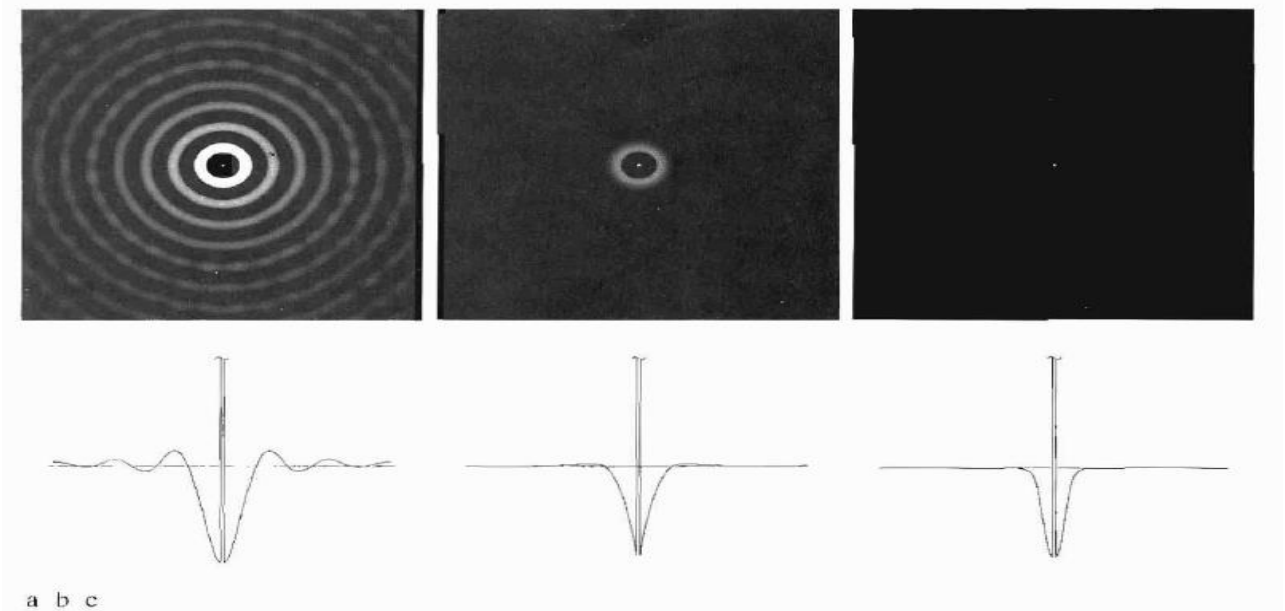
$$H_{hp}(u, v) = 1 - H_{lp}(u, v)$$

where ***H1p(u,*** v) is the transfer function of the corresponding lowpass filter. That is, when the lowpass filter attenuates frequencies, the highpass filter passes them, and vice versa.

we consider ideal, Butterworth, and Gaussian highpass filters. we illustrate the characteristics of these filters in both the frequency and spatial domains. Figure shows typical 3-D plots, image representations, and cross sections for these fi1ters.



a spatial representation of a frequency domain filter is obtained by (1) multiplying ***H(u,*** v) by $(-1)^{u+v}$ for centering; (2) computing the inverse DFT; and (3) multiplying the real part of the inverse DFT by $(-1)^{x+y}$.



a b c

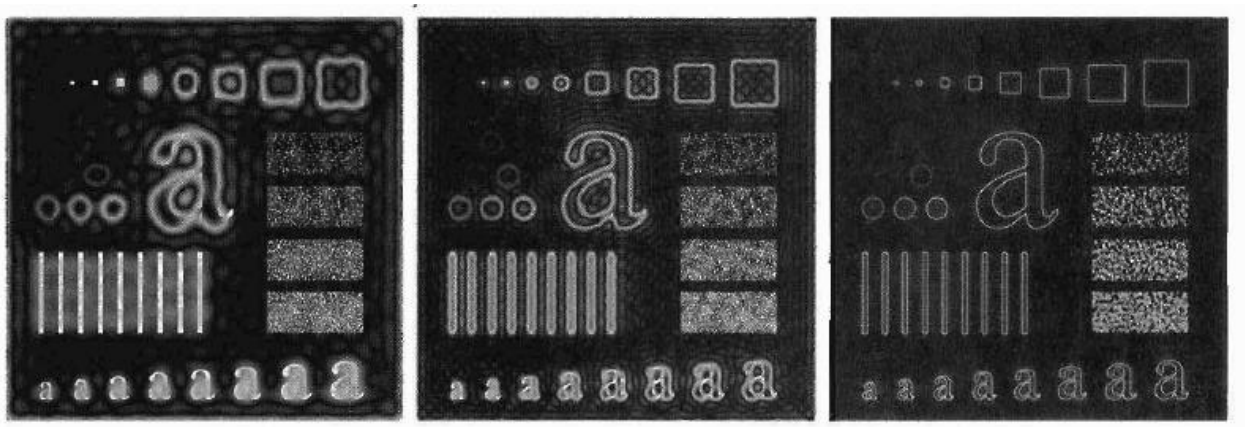
FIGURE 4.23 Spatial representations of typical (a) ideal, (b) Butterworth, and (c) Gaussian frequency domain highpass filters, and corresponding gray-level profiles.

Ideal Highpass Filters

A 2-D ideal highpass filter (IHPF) is defined as

$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$$

where *Do,* is the cutoff distance measured from the origin of the frequency rectangle. this filler is the opposite of the ideal lowpass filter in the sense that it sets to zero all frequencies inside a circle of radius *D,,* while passing, without attenuation, all frequencies outside the circle. **As** in the case of the ideal lowpass filter, the IHPF is not physically realizable with electronic components. However, since it can be implemented in a computer, we consider it for completeness.



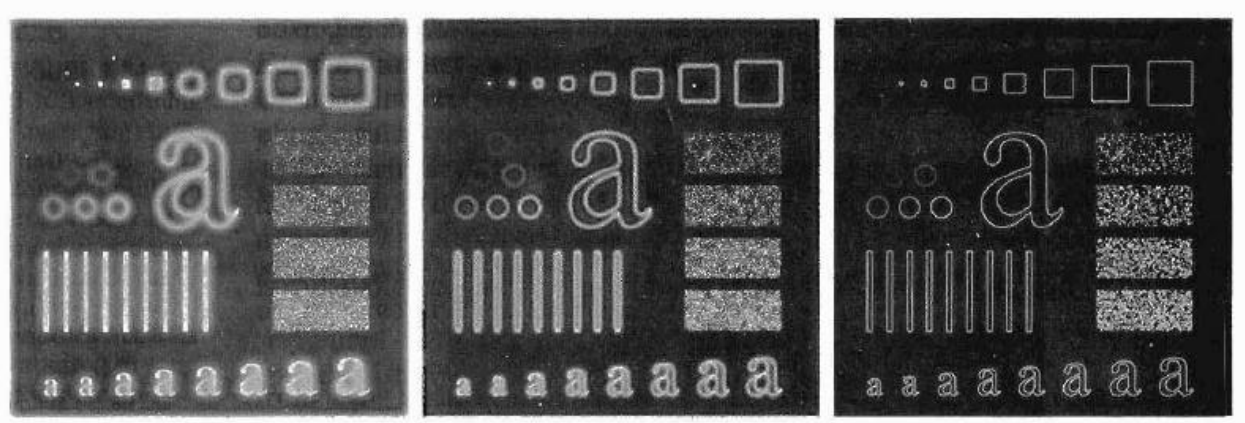
a b c

FIGURE 4.24 Results of ideal highpass filtering the image in Fig. 4.11(a) with $D_0 = 15$, 30, and 80, respectively. Problems with ringing are quite evident in (a) and (b).

Butterworth Highpass Filters

The transfer function of the Butterworth highpass filter (BHPF) of order n and with cutoff frequency locus at a distance *Do* from the origin is given by

$$H(u,v) = \frac{1}{1 + \left[D_0/D(u,v)\right]^{2n}}$$



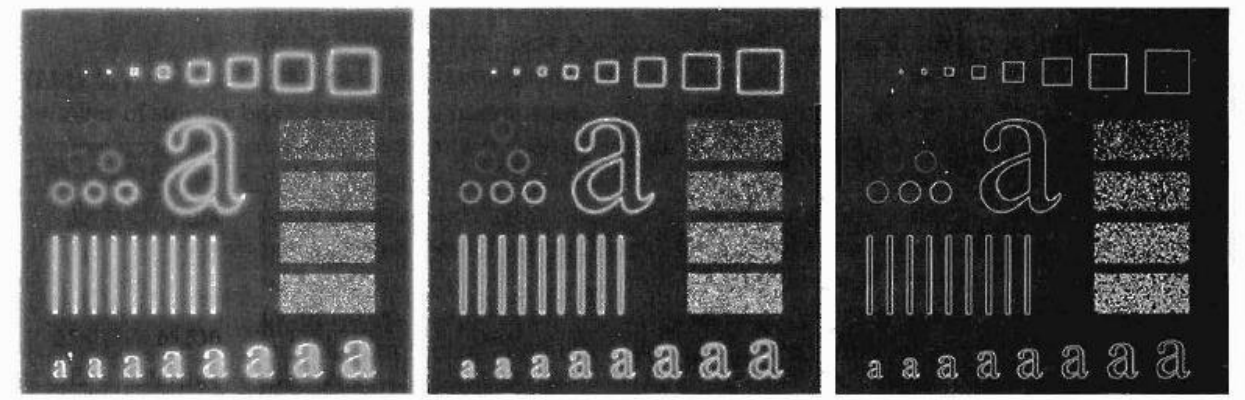
a b c

FIGURE 4.25 Results of highpass filtering the image in Fig. 4.11(a) using a BHPF of order 2 with $D_0 = 15$, 30, and 80, respectively. These results are much smoother than those obtained with an ILPF.

Gaussian Highpass Filters

The transfer function of the Gaussian highpass filter (GHPF) with cutoff frequency locus at a distance Do from the origin is given by

$$H(u, v) = 1 - e^{-D^2(u,v)/2D_0^2}$$



a b c

FIGURE 4.26 Results of highpass filtering the image of Fig. 4.11(a) using a GHPF of order 2 with $D_0 = 15$,

The Laplacian in the Frequency Domain

It can be shown that

$$\Im\left[\frac{d^n f(x)}{dx^n}\right] = (ju)^n F(u).$$

From this simple expression, it follows that

$$\Im\left[\frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}\right] = (ju)^2 F(u, v) + (jv)^2 F(u, v)$$
$$= -(u^2 + v^2) F(u, v).$$

The expression inside the brackets on the left side of Eq. is recognized as the Laplacian off *(x , y)*. Thus, we have the important result

$$\Im[\nabla^2 f(x, y)] = -(u^2 + v^2)F(u, v),$$

which simply says that the Laplacian can be implemented in the frequency domain by using the filter

$$H(u, v) = -(u^2 + v^2).$$

As in all filtering operations in this chapter, the assumption is that the origin of ***F(u,v)*** has been centered by performing the operation ***f(x,y)*** **$(-1)^{x+y}$ prior** to taking the transform of the image. As discussed earlier, if f (and *F)* are of size M X N, this operation shifts the center transform so that *(u,v) = (0, 0)* is at point *(M/2,N /2)* in the frequency rectang1e.As before, the center of the filter function also needs to be shifted:

$$H(u, v) = -[(u - M/2)^2 + (v - N/2)^2].$$

The Laplacian-filtered image in the spatial domain is obtained by computing the Inverse Fourier transform of *H(u,v) F(u,v)*

$$\nabla^2 f(x, y) = \Im^{-1}\{-[(u - M/2)^2 + (v - N/2)^2]F(u, v)\}.$$

Conversely, computing the Laplacian in the spatial domain and computing the Fourier transform of the result is equivalent to multiplying *F(u. v)* by ***H(u,*** *v).* We express this dual relationship in the familiar Fourier transform-pair notation

$$\nabla^2 f(x, y) \Leftrightarrow -[(u - M/2)^2 + (v - N/2)^2]F(u, v).$$

The spatial domain Laplacian filter function obtained by taking the inverse Fourier transform of Eq. *(4.4-9)* has some interesting properties, as Fig. *4.27* shows. Figure *4.27(a)* is a *3-D* perspective plot of Eq. *(4.4-9).* The function is centered at *(M/2.* ***N / 2) ,*** and its value at the top of the dome is zero. All other values are negative-Figure **4.27(b)** shows ***H(u, v)*** as **an image, also centered.** Figure 4.27(c) is the Laplacian in the spatial domain, obtained by multiplying by ***H(u, 2))*** by $(-I)^{x+y}$, taking the inverse Fourier transform, and multiplying the real part of the result by (-1)"+Y. Figure 4.27(d) is a zoomed section at about the origin of Fig. 4,27(c): Figure **4.27(e)** is a horizontal gray-level profile passing through the center of the zoomed section.

we form an enhanced image **g(x,** y) by subtracting the Laplacian from the original image:

$$g(x, y) = f(x, y) - \nabla^2 f(x, y).$$

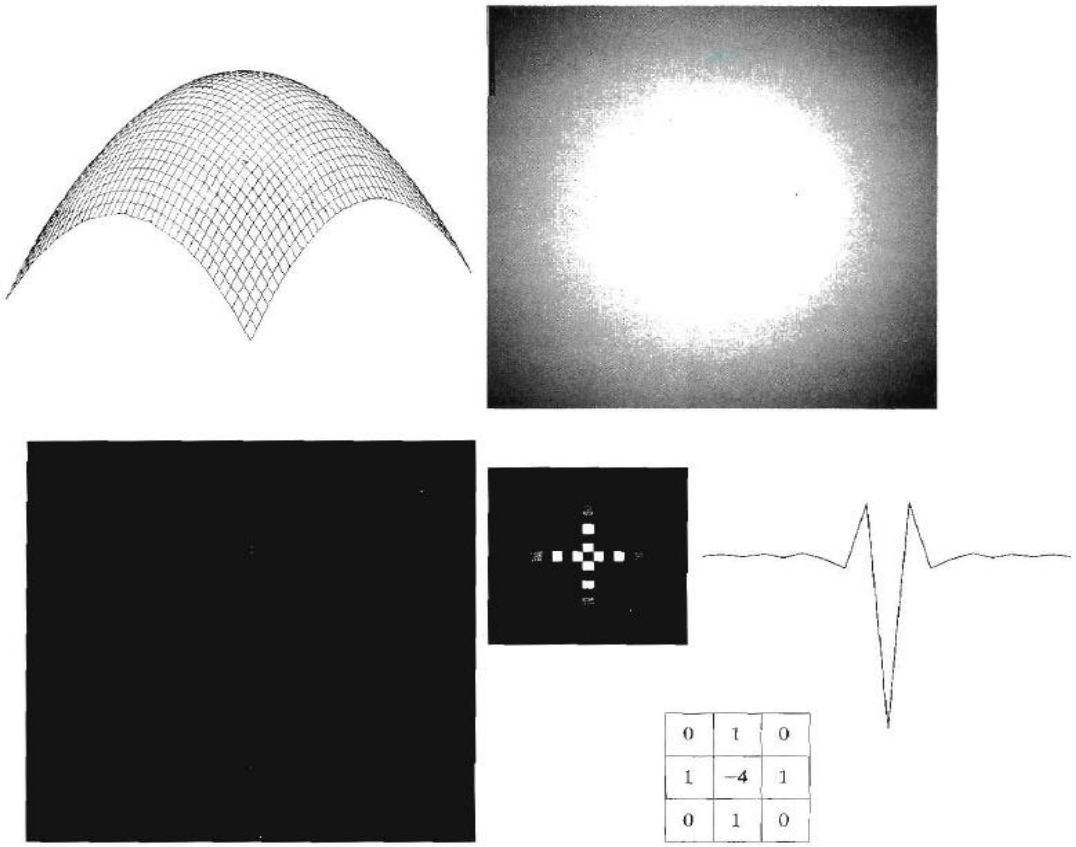
The Laplacian is subtracted from (rather than added to) the original because of the negative sign.

As in the spatial domain, where we obtained the enhanced image with a single mask, it is possible to perform the entire operation in the frequency domain with only one filter, given by

$$H(u, v) = [1 + [(u - M/2)^2 + (v - N/2)^2]].$$

The enhanced image is obtained with a single inverse transform operation:

$$g(x, y) = \Im^{-1}\{[1 + ((u - M/2)^2 + (v - N/2)^2)]F(u, v)\}.$$



a b
c d e
f

FIGURE 4.27 (a) 3-D plot of Laplacian in the frequency domain. (b) Image representation of (a). (c) Laplacian in the spatial domain obtained from the inverse DFT of (b). (d) Zoomed section of the origin of (c). (e) Gray-level profile through the center of (d). (f) Laplacian mask used in Section 3.7.

Selective filtering

Unsharp Masking, High-Boost Filtering, and High-Frequency Emphasis Filtering

All the filtered images have one thing in common: Their average background intensity has been reduced to near black.This is due to the fact that the highpass filters we applied to those images eliminate the zero-frequency component of their Fourier transforms. the solution to this problem consists of adding a portion of the image back to the filtered result. In fact, enhancement using the Laplacian does precisely this, by adding back the entire image to the filtered result. Sometimes it is advantageous to increase the contribution made by the original image to the overall filtered result. This approach, called *high-boost filtering,* is a generalization of unsharp masking.

Unsharp masking consists simply of generating a sharp image by subtracting from an image a blurred version of itself. Using frequency domain terminology, this means obtaining a highpass-filtered image by subtracting from the image a lowpass-filtered version of itself. That is,

$$f_{hp}(x, y) = f(x, y) - f_{lp}(x, y).$$

High-boost filtering generalizes this by multiplying f ***(x ,*** y) by a constant ***A>=1***

$$f_{hb} = Af(x, y) - f_{lp}(x, y).$$

Thus, high-boost filtering gives us the flexibility to increase the contribution made by the image to the overall enhanced result. This equation may be written as

$$f_{hb}(x, y) = (A - 1)f(x, y) + f(x, y) - f_{lp}(x, y).$$

$$f_{hb}(x, y) = (A - 1)f(x, y) + f_{hp}(x, y).$$

This result is based on a highpass rather than a lowpass image. When A = 1, high-boost filtering reduces to regular highpass filtering. As A increases past 1, the contribution made by the image itself becomes more dominant.

From Eq, *Fhp(u,v) = F(u,v) - fip(u,v)* . But, f1p*(u,v) = Hlp(u, v) F(u, v),* where ***Hlp*** is the transfer function of a lowpass filter. Therefore, unsharp masking can be implemented directly in the frequency domain by using the composite filter.

$$H_{hp}(u, v) = 1 - H_{lp}(u, v).$$

high-boost filtering can be implemented with the composite filter

$$H_{hb}(u, v) = (A - 1) + H_{hp}(u, v)$$

with ***A***>=1. The process consists of multiplying this filter by the (centered) transform of the input image and then taking the inverse transform of the product. Multiplication of the real part of this result by $(-l)^{x+Y}$ gives us the high boost filtered image Fhb(x, y) in the spatial domain.

it is advantageous to accentuate the contribution to enhancement made by the high-frequency components of an image. In this case, we simply multiply a highpass filter function by a constant and add an offset so that the zero frequency term is nor eliminated by the filter. This process, called *high frequency emphasis* has a filler transfer function given by

$$H_{hfe}(u, v) = a + bH_{hp}(u, v)$$

where ***a***>=0 and *b* > 0.Typical values of a are in the range 0.25 lo 0.5 and typical values of *b* are in the range 1.5 to 2.0. With reference to Eq. (4.4-17), we see that high-frequency emphasis reduces to high-boost filtering when *a* = ***(A*** - 1) and b = 1. When b > 1, the high frequencies are emphasized, thus giving this procedure its name.

Highpass filtering is not overly sensitive to this parameter, as long as the radius of the filter is not so small that frequencies near the origin of the transform are passed. The advantage of high-emphasis filtering the image is still dark, the gray-level tonality due to the low frequency components was not lost.

Homomorphic Filtering

The illumination-reflectance model can be used to develop a frequency domain procedure for improving the appearance of an image by simultaneous gray-level range compression and contrast enhancement. An image f (x, y) can be expressed as the

product of illumination and reflectance components:

$$f(x, y) = i(x, y)r(x, y).$$

cannot be used directly to operate separately on the frequency components of illumination and reflectance because the Fourier transform of the product of two functions is not separable; in other words,

$$\Im\{f(x, y)\} \neq \Im\{i(x, y)\}\Im\{r(x, y)\}.$$

suppose, however, that we define

$$\begin{aligned} z(x, y) &= \ln f(x, y) \\ &= \ln i(x, y) + \ln r(x, y). \end{aligned}$$

Then

$$\begin{aligned} \Im\{z(x, y)\} &= \Im\{\ln f(x, y)\} \\ &= \Im\{\ln i(x, y)\} + \Im\{\ln r(x, y)\} \end{aligned}$$

$$Z(u, v) = F_i(u, v) + F_r(u, v)$$

where *Fi(u,* v) and *Fr(u, v)* are the Fourier transforms of In i(x, y) and ln r(x, y), respectively.

If we process ***Z(u,*** v) by means of a filter function *H(u,* ***v)*** then

$$\begin{aligned} S(u, v) &= H(u, v)Z(u, v) \\ &= H(u, v)F_i(u, v) + H(u, v)F_r(u, v) \end{aligned}$$

where ***S(u,*** *v)* is the Fourier transform of the result. In the spatial domain,

$$\begin{aligned} s(x, y) &= \Im^{-1}\{S(u, v)\} \\ &= \Im^{-1}\{H(u, v)F_i(u, v)\} + \Im^{-1}\{H(u, v)F_r(u, v)\}. \end{aligned}$$

By letting

$$i'(x, y) = \Im^{-1}\{H(u, v)F_i(u, v)\}$$

$$r'(x, y) = \Im^{-1}\{H(u, v)F_r(u, v)\},$$

$$s(x, y) = i'(x, y) + r'(x, y).$$

FinalIy, as z(x, y) was formed by taking the logarithm of the original image f ***(x,*** y), the inverse (exponential) operation yields the desired enhanced image, denoted by g(x, y); that is,

$$\begin{aligned} g(x, y) &= e^{s(x,y)} \\ &= e^{i'(x,y)} \cdot e^{r'(x,y)} \\ &= i_0(x, y)r_0(x, y) \end{aligned}$$

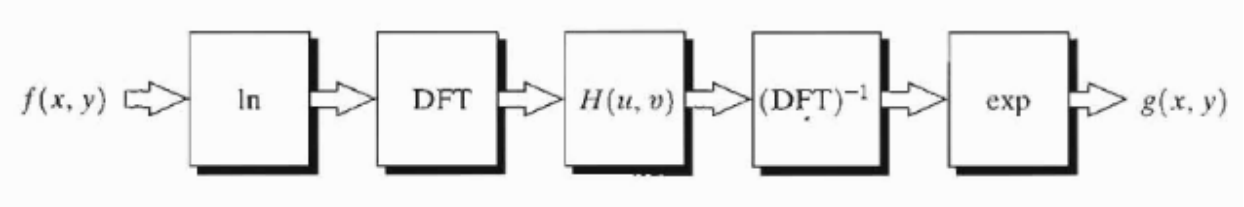
Where

$$i_0(x, y) = e^{i'(x,y)}$$

$$r_0(x, y) = e^{r'(x,y)}$$

are the illumination and reflectance components of the output image.

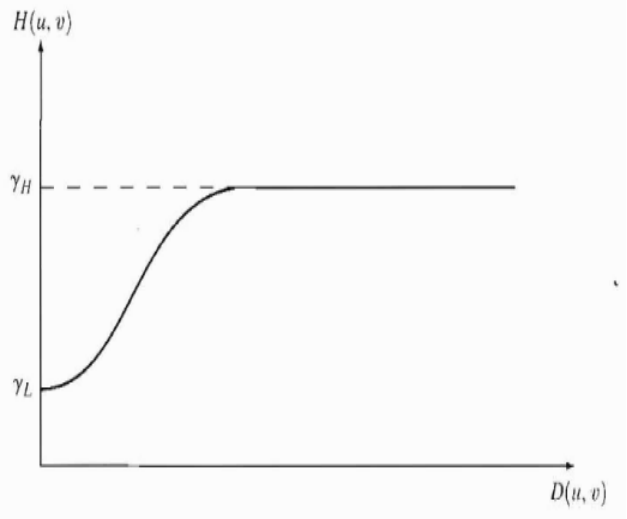
The enhancement approach using the foregoing concepts is summarized



This method is based on a special case of a class of systems known as *homomorphic systems. The key to the approach is the separation of the illumination and reflectance components achieved in the form.* The homomorphic filter function H(u,v) can then operate on these components separately

The illumination component of an image generally is characterized by slow spatial variations, while the reflectance component tends to vary abruptly, particularly at the junctions of dissimilar objects. These characteristics lead to associating the low frequencies of the Fourier transform of the logarithm of an image with illumination and the high frequencies with reflectance. A1 though these associations are rough approximations, they can be used to advantage in image enhancement.

A good deal of control can be gained over the illumination and reflectance components with a homomorphic filter-This control requires specification of a filter function *N(u, v)* that affects the low- and high-frequency components of the Fourier transform in different ways. Figure 4.32 shows a cross section of such a filter. If the parameters *yl,* and *yh* are chosen so that *yl* < 1 and *yh* > 1, the filter function shown in Fig tends to decrease the contribution made by the low frequencies (illumination) and amplify the contribution made by high frequencies (reflectance). The net result is simultaneous dynamic range compression and contrast enhancement.



The curve shape shown in Fig. can be approximated using the basic form of any of the ideal highpass filters discussed in the previous section. For example, using a slightly modified form of the Gaussian highpass filter gives us

$$H(u, v) = (\gamma_H - \gamma_L)\left[1 - e^{-c\left[D^2(u,v)/D_0^2\right]}\right] + \gamma_L$$

The constant c has been introduced to control the sharpness of the slope of the filter function as it transitions between yh, and yl.

UNIT- III IMAGE RESTORATION

As in image enhancement, the ultimate goal of restoration techniques is to improve an image in some predefined **sense.** Although there are areas of overlap, image enhancement is largely a subjective process, while image restoration is for the most part an objective process. Restoration attempts to reconstruct or recover an image that has been degraded **by** using a priori knowledge of the degradation phenomenon. Thus restoration techniques are oriented **toward** modeling the degradation and applying the inverse process in order to recover the original image.

A Model of the Image Degradation/Restoration Process

As Fig. 5.1 shows, the degradation process is modeled in this chapter as a degradation function that, together with an additive noise term, operates on an input image $f(x, y)$ to produce a degraded image $g(x, y)$. Given $g(x, y)$, some knowledge about the degradation function 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 input image and, in general, the more we know about H and η, the closer $\hat{f}(x, y)$ will be to $f(x, y)$. The approach used throughout most of this chapter is based on various types of image restoration filters.

It is shown in Section 5.5 that if H is a linear, position-invariant process, then the degraded image is given in the *spatial domain* by

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y) \qquad (5.1\text{-}1)$$

Where $h(x, y)$ is the spatial representation of the degradation function and, as in Chapter 4, the symbol indicates convolution. We know from the discussion in Sections 4.2.4 and 4.6.4 that convolution in the spatial domain is equal to multiplication in the frequency domain, so we may write the model in Eq. (5.1-1) in an equivalent frequency *domain representation*:

$$G(u, v) = H(u,v)F(u,v) + N(u,v) \qquad (5.1\text{-}2)$$

where the terms in capital letters are the Fourier transforms of the corresponding terms in E.q. (5.1-1).

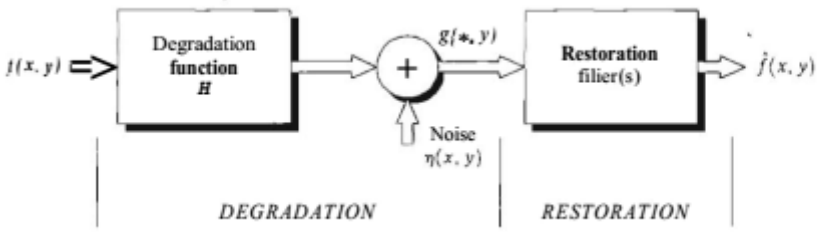


FIGURE 5.1 A model of the image degradation/ restoration process.

Noise Models

The principal sources of noise in digital images arise during image acquisition (Digitization) and/or transmission. The performance of imaging sensors is affected by a variety of factors, such as environmental conditions 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 due to interference in the channel used for transmission.

For example, an image transmitted using a wireless network might be corrupted as a result of 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 proper¬ ties refer to the frequency content of noise in the Fourier sense (i.e., as opposed to the electromagnetic spectrum). For example, when the Fourier spectrum of noise is constant, the noise usually is called *white* noise.

Some Important Noise Probability Density Functions

Based on the assumptions in the previous section, the *spatial* noise descriptor with which we shall be concerned is the statistical behavior of the gray-level 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). The following are among the most common PDFs found in image processing applications.

Gaussian noise

Because of its mathematical tractability in both the spatial and frequency do¬ mains. Gaussian (also called *norma!)* 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-given by

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\mu)^2/2\sigma^2} \qquad (5.2\text{-}1)$$

where z represents gray level, /x is the mean of average value of z, and cr is its standard deviation. The standard deviation squared, cr2, is called the *variance* of *z* -A plot of this function is shown in Fig. 5.2(a). When *z* is described by Eq. (5.2-1), approximately 70% of its values will be in the range $[(\mu-\sigma),(\mu+\sigma)]$, and about 95% will be in the range $[(\mu-2\sigma),(\mu+2\sigma)]$.

Rayleigh noise

The PDF of Rayleigh noise is given by

$$p(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b} & \text{for } z \geq a \\ 0 & \text{for } z < a. \end{cases} \qquad (5.2\text{-}2)$$

The mean and variance of this density are given by

$$\mu = a + \sqrt{\pi b/4} \qquad 5.2.3$$

And

$$\sigma^2 = \frac{b(4-\pi)}{4} \qquad 5.2.4$$

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 this density is skewed to the right. The Rayleigh density can be quite useful for approximating skewed histograms.

Erlang (Gamma) noise

The PDF of Erlang noise is given by

$$p(z) = \begin{cases} \dfrac{a^b z^{b-1}}{(b-1)!} e^{-az} & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases} \qquad (5.2\text{-}5)$$

where the parameters are such that $a > 0$, b is a positive integer, and ! indicates factorial. The mean and variance of this density are given by

$$u = \frac{b}{a}$$

and

$$\sigma^2 = \frac{b}{a^2}$$

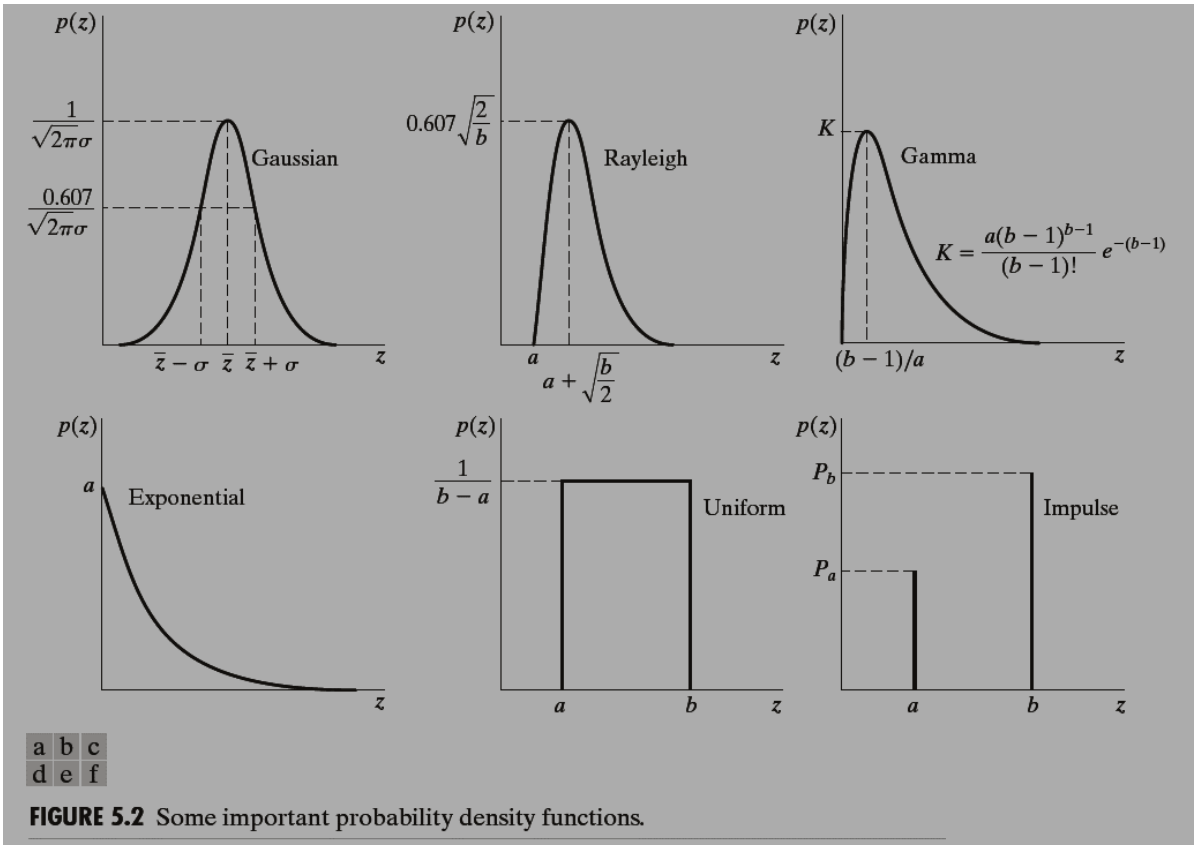


FIGURE 5.2 Some important probability density functions.

Figure shows a plot of this density. Although Eq. (5.2-5) often is referred to as Lhe *gamma density*, strictly speaking this is correct only when the denom ¬ inator is the gamma function, T(6). When the denominator is as shown, the den ¬ sity is more appropriately called the *Erlang density*.

Exponential noise
The PDF of ***exponential*** noise **is** given **by**

$$p(z) = \begin{cases} ae^{-az} & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases} \qquad (5.2\text{-}8)$$

where a >0. The mean and variance of this density function are

$$u = \frac{1}{a} \quad \text{and} \quad \sigma^2 = \frac{1}{a^2}$$

Note that this PDF is a special case of the Erlang PDF, with *6* = I . Figure5.2(d) **shows** *a* **plot** of this density function.

Uniform **noise**
The PDF of ***uniform*** noise is given by

$$p(z) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq z \leq b \\ 0 & \text{otherwise.} \end{cases} \qquad (5.2\text{-}11)$$

The mean of this density function is given by

$$u = \frac{a+b}{2}$$

and its variance by

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

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

Impulse (salt-and-pepper) noise
The PDF of *(bipolar) impulse noise* is given by

$$p(z) = \begin{cases} P_a & \text{for } z = a \\ P_b & \text{for } z = b \\ 0 & \text{otherwise} \end{cases} \qquad (5.2\text{-}14)$$

If $b > a$, gray-level b will appear as a light dot in the image. Conversely, level a will appear like a dark dot. If either *Pa,* or *Pb* is zero, the impulse noise is called uni polar. If either probability is zero, and especially if they are approximately equal, impulse noise values will resemble salt-and-pepper granules randomly distributed over the image. For this reason. bipolar impulse noise also is called *salt-and-pepper* noise.
Restoration in the Presence of Noise Only-Spatial Filtering
When the only degradation present in an image is noise, Eqs. (5.1-1) and (5.1-2) become

$$g(x, y) = f(x, y) + \eta(x, y) \qquad (5.3\text{-}1)$$

And

$$G(u, v) = F(u, v) + N(u, v). \qquad (5.3\text{-}2)$$

The noise terms are unknown, so subtracting them from ***g (x , y)*** or *G(u,v,)* is not a realistic option. In the case of periodic noise, it usually ***is*** possible to estimate ***N (u,v)from*** the spectrum of *G (u* ,v) , as noted in Section *5.2.3.* In this case ***N (u,v)*** Can be subtracted from G(u,v) to obtain an estimate of the orginal image. **in** general, however? this type of knowledge is the exception rather than the rule.

Restoration in the presence of noise only-Spatial filtering

Mean Filters

Arithmetic mean filter

This is the simplest of the mean filters. Let represent (he set of coordinates in a rectangular subimage window of size m X n, centered at point (.v, y). The arithmetic mean filtering process computes the average value of the corrupted image g(.r, y) in the area defined by Sxy. The value of the restored image / at any point (x. y) is simply the arithmetic mean computed using the pixels in the region defined by S_{xy}. In other words

$$\hat{f}(x, y) = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s, t). \qquad (5.3\text{-}3)$$

Geometric mean filter

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

$$\hat{f}(x, y) = \left[\prod_{(s,t) \in S_{xy}} g(s, t) \right]^{\frac{1}{mn}}. \qquad (5.3\text{-}4)$$

Harmonic mean filter

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

$$\hat{f}(x, y) = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s, t)}}. \qquad (5.3\text{-}5)$$

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 n o i s e .

Contra harmonic mean filter

The *contra harmonic* mean filtering operation yields a restored image based on the expression

$$\hat{f}(x, y) = \frac{\sum_{(s,t) \in S_{xy}} g(s, t)^{Q+1}}{\sum_{(s,t) \in S_{xy}} g(s, t)^{Q}} \qquad (5.3\text{-}6)$$

where Q is called the *order* of the filter. This filter is well suited for reducing or vir¬ tually 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 contraharmonicfilter reduces to the arithmetic mean filter if $Q = 0$, and to the harmonic mean filter if $Q = -1$.

Order¬Statistics Filters

Order-statistics filters were introduced in Section 3.6.2. We now expand the dis¬ cussion in that section and introduce some additional order-statistics filters. As noted in Section 3.6.2, order-statistics filters are spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encom¬ passed by the filter. The response of the filter at any point is determined by the ranking result.

Median filter

The best-known order-statistics filter is the *median filler*, which, as its name im¬ plies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel:

$$\hat{f}(x, y) = \underset{(s,t)\in S_{xy}}{\text{median}}\{g(s, t)\}. \qquad (5.3\text{-}7)$$

Max and min filters

Although the median filter is by far the order-statistics 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 the reader will recall from basic statis¬ tics that ranking lends itself to many other possibilities. For example, using the 100 th percentile results in the so— called *max filter*, given by

$$\hat{f}(x, y) = \max_{(s,t)\in S_{xy}}\{g(s, t)\}. \qquad (5.3\text{-}8)$$

This filter is useful for finding (he brightest points in an image. Also, because pep¬ per noise has very low values, it is reduced by this filter as a result of the max▾ selection process in the subinrage area *Sxy*.

The Oth percentile filter is the *min filter*

$$\hat{f}(x, y) = \min_{(s,t)\in S_{xy}}\{g(s, t)\}. \qquad (5.3\text{-}9)$$

Midpoint filter

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

$$\hat{f}(x, y) = \frac{1}{2}\left[\max_{(s,t)\in S_{xy}}\{g(s, t)\} + \min_{(s,t)\in S_{xy}}\{g(s, t)\}\right]. \qquad (5.3\text{-}10)$$

Alpha-trimmed mean filter

Suppose that we delete the $d/2$ lowest and the $d/2$ highest gray-level values of $g(s,t)$ in the neighborhood S_{xy}. Let $g_r(s,t)$ represent the remaining $mn - d$ pixels. A filter formed by averaging these remaining pixels is called an *alpha trimmed mean* filter:

$$\hat{f}(x, y) = \frac{1}{mn - d}\sum_{(s,t)\in S_{xy}} g_r(s, t) \qquad (5.3\text{-}11)$$

AdaptiveFilters

the Adaptive Filters.

Adaptive filters are filters whose behavior changes based on statistical characteristics of the image inside the filter region defined by the m X n rectangular window S_{xy}.

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 filler because they are quantities closely related to the appearance of an image.

The mean gives a measure of average gray level in the region over which the mean is computed, and the variance gives a measure of average contrast in that region.

This filter is to operate on a local region, S_{xy}. The response of the filter at any point (x, y) on which the region is centered is to be based on four quantities: (a) g(x, y), the value of the noisy image at (x, y); (b) a2, the variance of the noise corrupting /(x, y) to form g(x, y); (c) ray, the local mean of the pixels in S_{xy}; and (d) σ^2_L , the local variance of the pixels in S_{xy}.

The behavior of the filter to be as follows:

1. If σ^2_η is zero, the filler should return simply the value of g (x, y). This is the trivial, zero-noise case in which g (x, y) is equal to f (x, y).

2. If the local variance is high relative to σ^2_η the filter should return a value close to g (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 simply by averaging.

Adaptive local noise filter is given by,

$$\hat{f}(x, y) = g(x, y) - \frac{\sigma_\eta^2}{\sigma_L^2}[g(x, y) - m_L].$$

The only quantity that needs to be known or estimated is the variance of the overall noise, a2. The other parameters are computed from the pixels in S_{xy} at each location (x, y) on which the filter window is centered.

Adaptive median filter:

The median filter performs well as long as the spatial density of the impulse noise is not large (as a rule of thumb, P_a and P_b less than 0.2). The adaptive median filtering can handle impulse noise with probabilities even larger than these. An additional benefit of the adaptive median filter is that it seeks to preserve detail while smoothing nonimpulse noise, something that the "traditional" median filter does not do. The adaptive median filter also works in a rectangular window area S_{xy}. Unlike those filters, however, the adaptive median filter changes (increases) the size of S_{xy} during filter operation, depending on certain conditions. The output of the filter is a single value used to replace the value of the pixel at (x, y), the particular point on which the window S_{xy} is centered at a given time.

Consider the following notation:

z_{min} = minimum gray level value in

S_{xy} z_{max} = maximum gray level

value in S_{xy} z_{mcd} = median of gray

levels in S_{xy}

z_{xy} = gray level at coordinates (x, y)

S_{max} = maximum allowed size of

S_{xy}.

The adaptive median filtering algorithm works in two levels, denoted level A and level B, as follows:

Level A: $A1 = z_{med} - z_{min}$

$A2 = z_{med} -$

z_{max} If A1 > 0 AND A2 < 0, Go to

level B Else increase the window

size

If window size ≤ S_{max} repeat level

A Else output z_{xy}

Level B: $B1 = z_{xy} - z_{min}$

$B2 = z_{xy} - z_{max}$

If B1> 0 AND B2 < 0, output zxy

Else output z_{med}

Periodic Noise Reduction by Frequency Domain Filtering

In Chapter 4 we discussed lowpass and bighpass frequency domain filters as fundamental tools for image enhancement. In this section we discuss the more specialized bandreject, bandpass, and notch filters as tools for periodic noise reduction or removal

Bandreject Filters

Bandreject filters remove or attenuate a band of frequencies about the origin of the Fourier transform. An ideal bandreject filter is given by the expression

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) < D_0 - \frac{W}{2} \\ 0 & \text{if } D_0 - \frac{W}{2} \le D(u,v) \le D_0 + \frac{W}{2} \\ 1 & \text{if } D(u,v) > D_0 + \frac{W}{2} \end{cases} \qquad (5.4\text{-}1)$$

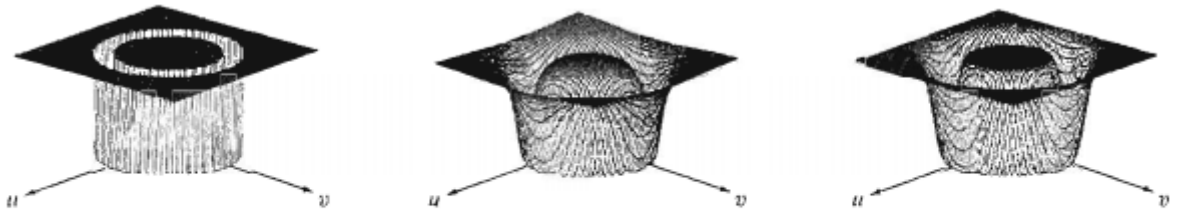
where $D(u, v)$ is the distance from the origin of the centered frequency rectan¬ gle, as given in Eq. (4.3-3), W is the width of the band, and Du is its radial center. Similarly,a Butterworth bandreject filter of order n is given by the expression

$$H(u,v) = \frac{1}{1 + \left[\dfrac{D(u,v)W}{D^2(u,v) - D_0^2}\right]^{2n}} \qquad (5.4\text{-}2)$$

and a Gaussian bandreject filter is given by

$$H(u,v) = 1 - e^{-\frac{1}{2}\left[\frac{D^2(u,v) - D_0^2}{D(u,v)W}\right]^2}. \qquad (5.4\text{-}3)$$

Figure 5.15 shows perspective plots of these three filters.



a. b. c

FIGURE 5.15 From left to right, perspective plots of ideal, Butterworth (of order 1), and Gaussian bandreject filters.

Bandpass Filters

A *bundpaw* filler performs the opposite operation of a bandreject filter. In Sec¬ tion 4.4 we showed how a highpass filter can be obtained from a correspond¬ ing lowpass filter by using Eq . (4.4-1). Similarly, the transfer function F/h[>(/ / , o) of a bandpass filter is obtained from a corresponding bandreject filler with transfer function $Hhr(u, v)$ by using the equation

7661083308

$$H_{bp}(u,v) = 1 - H_{br}(u,v). \qquad (5.4\text{-}4)$$

Inverse filtering

The objective is to minimize

$$J(\mathbf{f}) = \|\mathbf{n}(\mathbf{f})\|^2 = \|\mathbf{y} - \mathbf{H}\mathbf{f}\|^2$$

We set the first derivative of the cost function equal to zero

$$\frac{\partial J(\mathbf{f})}{\partial \mathbf{f}} = 0 \Rightarrow -2\mathbf{H}^{\mathbf{T}}(\mathbf{y} - \mathbf{H}\mathbf{f}) = \mathbf{0}$$

$$\mathbf{H}^{\mathbf{T}}\mathbf{H}\mathbf{f} = \mathbf{H}^{\mathbf{T}}\mathbf{y}$$

If $M = N$ and $\mathbf{H}^{-1}$ exists then

$$\mathbf{f} = \mathbf{H}^{-1}\mathbf{y}$$

According to the previous analysis if $\mathbf{H}$ (and therefore $\mathbf{H}^{-1}$) is block circulant the above problem can

be solved as a set of $M \times N$ scalar problems as follows

$$F(u,v) = \frac{H^*(u,v)Y(u,v)}{|H(u,v)|^2} \Rightarrow f(i,j) = \Im^{-1}\left[\frac{H^*(u,v)Y(u,v)}{|H(u,v)|^2}\right] = \frac{Y(u,v)}{H(u,v)}$$

Computational issues concerning inverse filtering

(I)

Suppose first that the additive noise $n(i, j)$ is negligible. A problem arises if $H(u,v)$ becomes very small or zero for some point filtering cannot be applied. (u,v) or for a whole region in the (u,v) plane. In that region inverse

Note that in most real applications $H(u,v)$ drops off rapidly as a function of distance from the origin !

Minimum Mean Square Error Wiener Filter Used For Image Restoration.

The inverse filtering approach makes no explicit provision for handling noise. This approach incorporates both the degradation function and statistical characteristics of noise into the restoration process. The method is founded on considering images and noise as random processes, and the objective is to find an estimate f of the uncorrupted image f such that the mean square error between them is minimized. This error measure is given by

$$e^2 = E\{(f - f)^2\}$$

where $E\{\bullet\}$ is the expected value of the argument. It is assumed that the noise and the image are uncorrelated; that one or the other has zero mean; and that the gray levels in the estimate are a linear function of the levels in the degraded image. Based on these conditions, the minimum of the error function is given in the frequency domain by the expression

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)S_f(u,v)}{S_f(u,v)|H(u,v)|^2 + S_\eta(u,v)}\right]G(u,v)$$
$$= \left[\frac{H^*(u,v)}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$
$$= \left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$

where we used the fact that the product of a complex quantity with its conjugate is equal to the magnitude of the complex quantity squared. This result is known as the Wiener filter, after N. Wiener [1942], who first proposed the concept in the year shown. The filter, which consists of the terms inside the brackets, also is commonly referred to as the minimum mean square error filter or the least square error filter. The Wiener filter does not have the same problem as the inverse filter with zeros in the degradation function, unless both H(u, v) and S_η(u, v) are zero for the same value(s) of u and v.

The terms in above equation are as follows: H

(u, v) = degradation function

H*(u, v) = complex conjugate of H (u, v)

$|H(u,v)|^2 = H^*(u,v) * H(u,v)$

$S_\eta(u,v) = |N(u,v)|^2$ = power spectrum of the noise

$S_f(u,v) = |F(u,v)|^2$ = power spectrum of the undegraded image.

As before, H (u, v) is the transform of the degradation function and G (u, v) is the transform of the degraded image. The restored image in the spatial domain is given by the inverse Fourier transform of the frequency-domain estimate F (u, v). Note that if the noise is zero, then the noise power spectrum vanishes and the Wiener filter reduces to the inverse filter.

When we are dealing with spectrally white noise, the spectrum $|N(u,v)|^2$ is a constant, which simplifies things considerably. However, the power spectrum of the undegraded image seldom is known. An approach used frequently when these quantities are not known or cannot be estimated is to approximate the equation as

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + K}\right]G(u,v)$$

where K is a specified constant.

Solution: if these points are known they can be neglected in the computation of $F(u, v)$.

(II)

In the presence of external noise we have that

$$\hat{F}(u,v) = \frac{H^*(u, v)\big(Y(u, v) - N(u, v)\big)}{|H(u, v)|^2} =$$

$$\frac{H^*(u, v)Y(u, v)}{|H(u, v)|^2} - \frac{H^*(u, v)N(u, v)}{|H(u, v)|^2} \Rightarrow$$

$$\hat{F}(u, v) = F(u, v) - \frac{N(u, v)}{H(u, v)}$$

If $H(u,v)$ becomes very small, the term $N(u, v)$ dominates the result.

Solution: again to carry out the restoration process in a limited neighborhood about the origin where $H(u,v)$ is not very small.

This procedure is called **pseudoinverse filtering**.

In that case we set

$$\hat{F}(u, v) = \begin{cases} \dfrac{H^*(u, v)Y(u, v)}{|H(u,v)|^2} & H(u, v) \neq 0 \\ 0 & H(u, v) = 0 \end{cases}$$

or

$$\hat{F}(u, v) = \begin{cases} \dfrac{H^*(u, v)Y(u, v)}{|H(u,v)|^2} & |H(u, v)| \geq \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

- In general, the noise may very well possess large components at high frequencies (u, v) , while $H(u,v)$ and $Y(u,v)$ normally will be dominated by low frequency components.
- ε is a small number chosen by the user.

Constrained least squares (CLS) filtering

It refers to a very large number of restoration algorithms.

The problem can be formulated as follows.

minimize

$$J(\mathbf{f}) = \|\mathbf{n}(\mathbf{f})\|^2 = \|\mathbf{y} - \mathbf{H}\mathbf{f}\|^2$$

subject to

$$\|\mathbf{C}\mathbf{f}\|^2 < \varepsilon$$

where

$\mathbf{Cf}$ is a high pass filtered version of the image.

The idea behind the above constraint is that the highpass version of the image contains a considerably large amount of noise!

Algorithms of the above type can be handled using optimization techniques.

Constrained least squares (CLS) restoration can be formulated by choosing an **f** to minimize the Lagrangian

$$\min\left(\|\mathbf{y} - \mathbf{H}\mathbf{f}\|^2 + \alpha\|\mathbf{C}\mathbf{f}\|^2\right)$$

Typical choice for $\mathbf{C}$ is the 2-D Laplacian operator given by

$$\mathbf{C} = \begin{bmatrix} 0.00 & -0.25 & 0.00 \\ -0.25 & 1.00 & -0.25 \\ 0.00 & -0.25 & 0.00 \end{bmatrix}$$

α represents either a Lagrange multiplier or a fixed parameter known as **regularisation parameter**.

α controls the relative contribution between the term $\|\mathbf{y} - \mathbf{H}\mathbf{f}\|^2$ and the term $\|\mathbf{C}\mathbf{f}\|^2$.

The minimization of the above leads to the following estimate for the original image

$$\mathbf{f} = \left(\mathbf{H}^{\mathbf{T}}\mathbf{H} + \alpha\mathbf{C}^{\mathbf{T}}\mathbf{C}\right)^{-1}\mathbf{H}^{\mathbf{T}}\mathbf{y}$$

Computational issues concerning the CLS method

(I) Choice of α

The problem of the choice of α has been attempted in a large number of studies and different techniques have been proposed.

One possible choice is based on a **set theoretic approach**: a restored image is approximated by an image which lies in the intersection of the two ellipsoids defined by

$$Q_{\mathbf{f}|\mathbf{y}} = \{\mathbf{f} \mid \|\mathbf{y} - \mathbf{H}\mathbf{f}\|^2 \le E^2\} \text{ and}$$

$$Q_{\mathbf{f}} = \{\mathbf{f} \mid \|\mathbf{C}\mathbf{f}\|^2 \le \varepsilon^2\}$$

The center of one of the ellipsoids which bounds the intersection of $Q_{\mathbf{f}|\mathbf{y}}$ and $Q_{\mathbf{f}}$, is given by the equation

$$\mathbf{f} = \left(\mathbf{H}^{\mathbf{T}}\mathbf{H} + \alpha\mathbf{C}^{\mathbf{T}}\mathbf{C}\right)^{-1}\mathbf{H}^{\mathbf{T}}\mathbf{y}$$

with $\alpha = (E/\varepsilon)^2$.

Problem: choice of E^2 and ε^2. One choice could be

$$\alpha = \frac{1}{\text{BSNR}}$$

Comments

With larger values of α, and thus more regularisation, the restored image tends to have more ***ringing***.

With smaller values of α, the restored image tends to have more ***amplified noise effects***.

The variance and bias of the error image in frequency domain are

$$Var(\alpha) = \sigma_n^2 \sum_{u=0}^{M}\sum_{v=0}^{N} \frac{|H(u,v)|^2}{\left(|H(u,v)|^2 + \alpha|C(u,v)|^2\right)^2}$$

$$Bias(\alpha) = \sigma_n^2 \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \frac{|F(u,v)|^2 \alpha^2 |C(u,v)|^4}{\left(|H(u,v)|^2 + \alpha|C(u,v)|^2\right)^2}$$

The minimum MSE is encountered close to the intersection of the above functions.

A good choice of α is one that gives the best compromise between the variance and bias of the error image.

ITERATIVE METHODS

They refer to a large class of methods that have been investigated extensively over the last decades.

Advantages

- There is no need to explicitly implement the inverse of an operator. The restoration process is monitored as it progresses. Termination of the algorithm may take place before convergence.
- The effects of noise can be controlled in each iteration.
- The algorithms used can be spatially adaptive.
- The problem specifications are very flexible with respect to the type of degradation. Iterative techniques can be applied in cases of spatially varying or nonlinear degradations or in cases where the type of degradation is completely unknown (blind restoration).

A general formulation

In general, iterative restoration refers to any technique that attempts to minimize a function of the form

$$\Phi(\mathbf{f})$$

using an updating rule for the partially restored image.

A widely used iterative restoration method is the method of **successive approximations** where the initial estimate and the updating rule for obtaining the restored image are given by

$$\mathbf{f_0} = \mathbf{0}$$

$$\begin{aligned}\mathbf{f_{k+1}} &= \mathbf{f_k} + \beta\Phi(\mathbf{f_k}) \\ &= \Psi(\mathbf{f_k})\end{aligned}$$

Next we present possible forms of the above iterative procedure.

Basic iteration

$$\Phi(\mathbf{f}) = \mathbf{y} - \mathbf{Hf}$$

$$\mathbf{f_0} = \mathbf{0}$$

$$\mathbf{f_{k+1}} = \mathbf{f_k} + \beta(\mathbf{y} - \mathbf{Hf_k}) = \beta\mathbf{y} + (\mathbf{I} - \beta\mathbf{H})\mathbf{f_k}$$

Least squares iteration

In that case we seek for a solution that minimizes the function

$$M(\mathbf{f}) = \|\mathbf{y} - \mathbf{Hf}\|^2$$

A necessary condition for $M(\mathbf{f})$ to have a minimum is that its gradient with respect to $\mathbf{f}$ is equal to zero, which results in the normal equations

$$\mathbf{H^T Hf} = \mathbf{H^T y}$$

and

$$\Phi(\mathbf{f}) = \mathbf{H^T}(\mathbf{y} - \mathbf{Hf})$$

$$\mathbf{f_0} = \beta\mathbf{H^T y}$$

$$\begin{aligned}\mathbf{f_{k+1}} &= \mathbf{f_k} + \beta\mathbf{H^T}(\mathbf{y} - \mathbf{Hf_k}) \\ &= \beta\mathbf{H^T y} + (\mathbf{I} - \beta\mathbf{H^T H})\mathbf{f_k}\end{aligned}$$

Constrained least squares iteration

In this method we attempt to solve the problem of constrained restoration iteratively.

As already mentioned the following functional is minimized

$$M(\mathbf{f},\alpha) = \|\mathbf{y} - \mathbf{Hf}\|^2 + \alpha \|\mathbf{Cf}\|^2$$

The necessary condition for a minimum is that the gradient of $M(\mathbf{f},\alpha)$ is equal to zero. That is

$$\Phi(\mathbf{f}) = \nabla_{\mathbf{f}} M(\mathbf{f},\alpha) = (\mathbf{H}^T\mathbf{H} + \alpha\mathbf{C}^T\mathbf{C})\mathbf{f} - \mathbf{H}^T\mathbf{y}$$

The initial estimate and the updating rule for obtaining the restored image are now given by

$$\mathbf{f_0} = \beta\mathbf{H}^T\mathbf{y}$$

$$\mathbf{f_{k+1}} = \mathbf{f_k} + \beta[\mathbf{H}^T\mathbf{y} - (\mathbf{H}^T\mathbf{H} + \alpha\mathbf{C}^T\mathbf{C})\mathbf{f_k}]$$

It can be proved that the above iteration (known as **Iterative CLS** or **Tikhonov-Miller Method**) converges if

$$0 < \beta < \frac{2}{|\lambda_{\max}|}$$

where $\lambda_{\max}$ is the maximum eigenvalue of the matrix $(\mathbf{H}^T\mathbf{H} + \alpha\mathbf{C}^T\mathbf{C})$

If the matrices **H** and **C** are block-circulant the iteration can be implemented in the frequency domain.

Projection onto convex sets (POCS)

The set-based approach described previously can be generalized so that any number of prior constraints can be imposed as long as the constraint sets are closed convex.

If the constraint sets have a non-empty intersection, then a solution that belongs to the intersection set can be found by the method of POCS.

Any solution in the intersection set is consistent with the a priori constraints and therefore it is a feasible solution.

Let $Q_1, Q_2, \ldots, Q_m$ be closed convex sets in a finite dimensional vector space, with $P_1, P_2, \ldots, P_m$ their respective projectors.

The iterative procedure

$$\mathbf{f_{k+1}} = P_1 P_2, \ldots P_m \mathbf{f_k}$$

converges to a vector that belongs to the intersection of the sets $Q_i, i = 1,2,\ldots,m$, for any starting vector $\mathbf{f_0}$.

An iteration of the form $\mathbf{f_{k+1}} = P_1 P_2 \mathbf{f_k}$ can be applied in the problem described previously, where we seek for an image which lies in the intersection of the two ellipsoids defined by

$$Q_{\mathbf{f}|\mathbf{y}} = \{\mathbf{f} \mid \|\mathbf{y}-\mathbf{H}\mathbf{f}\|^2 \le E_2\} \text{ and } Q_{\mathbf{f}} = \{\mathbf{f} \mid \|\mathbf{C}\mathbf{f}\|^2 \le \varepsilon_2\}$$

The respective projections $P_1\mathbf{f}$ and $P_2\mathbf{f}$ are defined by

$$P_1\mathbf{f} = \mathbf{f} + \lambda_1\left(\mathbf{I} + \lambda_1\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T(\mathbf{y}-\mathbf{H}\mathbf{f})$$

$$P_2\mathbf{f} = [\mathbf{I} - \lambda_2\left(\mathbf{I} + \lambda_2\mathbf{C}^T\mathbf{C}\right)^{-1}\mathbf{C}^T\mathbf{C}]\mathbf{f}$$

Brief description of other advanced methods

Spatially adaptive iteration

The functional to be minimized takes the form

$$M(\mathbf{f},\alpha) = \|\mathbf{y}-\mathbf{H}\mathbf{f}\|^2_{\mathbf{w}_1} + \alpha\|\mathbf{C}\mathbf{f}\|^2_{\mathbf{w}_2}$$

where

$$\|\mathbf{y}-\mathbf{H}\mathbf{f}\|^2_{\mathbf{w}_1} = (\mathbf{y}-\mathbf{H}\mathbf{f})^T\mathbf{W}_1(\mathbf{y}-\mathbf{H}\mathbf{f})$$

$$\|\mathbf{C}\mathbf{f}\|^2_{\mathbf{w}_2} = (\mathbf{C}\mathbf{f})^T\mathbf{W}_2(\mathbf{C}\mathbf{f})$$

$\mathbf{W}_1, \mathbf{W}_2$ are diagonal matrices, the choice of which can be justified in various ways. The entries in both matrices are non-negative values and less than or equal to unity.

In that case

$$\Phi(\mathbf{f}) = \nabla_{\mathbf{f}} M(\mathbf{f},\alpha) = (\mathbf{H}^T\mathbf{W}_1^T\mathbf{W}_1\mathbf{H} + \alpha\mathbf{C}^T\mathbf{W}_2^T\mathbf{W}_2\mathbf{C})\mathbf{f} - \mathbf{H}^T\mathbf{W}_1\mathbf{y}$$

A more specific case is

$$M(\mathbf{f},\alpha) = \|\mathbf{y}-\mathbf{H}\mathbf{f}\|^2 + \alpha\|\mathbf{C}\mathbf{f}\|^2_{\mathbf{w}}$$

where the weighting matrix is incorporated only in the regularization term. This method is known as **weighted regularised image restoration**. The entries in matrix **W** will be chosen so that the high-pass filter is only effective in the areas of low activity and a very little smoothing takes place in the edge areas.

Robust functionals

Robust functionals allow for the efficient supression of a wide variety of noise processes and permit the reconstruction of sharper edges than their quadratic counterparts. We are seeking to minimize

$$M(\mathbf{f},\alpha) = R_n(\mathbf{y}-\mathbf{H}\mathbf{f}) + \alpha R_x\mathbf{C}\mathbf{f}$$

$R_n(), R_x()$ are referred to as **residual** and **stabilizing** functionals respectively.

Computational issues concerning iterative techniques

(I) Convergence

The **contraction mapping theorem** usually serves as a basis for establishing convergence of iterative algorithms.

According to it iteration

$$\mathbf{f_0} = \mathbf{0}$$

$$\mathbf{f_{k+1}} = \mathbf{f_k} + \beta\Phi(\mathbf{f_k}) = \Psi(\mathbf{f_k})$$

converges to a unique fixed point $\mathbf{f}^*$, that is, a point such that $\Psi(\mathbf{f}^*) = \mathbf{f}^*$, for any initial vector, if the operator or transformation $\Psi(\mathbf{f})$ is a **contraction**.

This means that for any two vectors $\mathbf{f_1}$ and $\mathbf{f_2}$ in the domain of $\Psi(\mathbf{f})$ the following relation holds

$$\|\Psi(\mathbf{f_1}) - \Psi(\mathbf{f_2})\| \le \eta \|\mathbf{f_1} - \mathbf{f_2}\|$$

$\eta \le 1$

$\|\cdot\|$ any norm

The above condition is **norm dependent.**

(II) Rate of convergence

The termination criterion most frequently used compares the normalized change in energy at each iteration to a threshold such as

$$\frac{\|\mathbf{f_{k+1}} - \mathbf{f_k}\|^2}{\|\mathbf{f_k}\|^2} \le 10^{-6}$$

RECURSIVE METHODS

1. Kalman filtering

Kalman is a recursive filter based on an autoregressive (AR) parametrization of the prior statistical knowledge of the image.

A global **state vector** for an image model, at pixel position (i, j) is defined as

$$\underline{f}(i,j) = [\, f(i,j), f(i,j-1), \ldots,$$

$$f(i-1, j+N), f(i-1, j-M+1), \ldots, f(i-M+1, j-M+1)]^T$$

The image model is then defined as

$$\underline{f}(i,j) = A\,\underline{f}(i,j-1) + \underline{w}(i,j)$$

$$\underline{y}(i,j) = H\,\underline{f}(i,j) + \underline{n}(i,j)$$

- the noise $\underline{w}(i,j)$ and $\underline{n}(i,j)$, are assumed to be white, zero-mean, Gaussian processes, terms

  with covariance matrices $R_{\underline{ww}}$ and $R_{\underline{nn}}$
- A is the **state transition matrix**
- H is the so called **measurement matrix**

The Kalman filter algorithm

Prediction

$$\underline{\hat{f}}^{+}(m,n) = A\,\underline{\hat{f}}\,(m,n-1)$$

$$P^{+}(m,n) = AP(m,n-1)\,A^{T} + R_{\underline{ww}}$$

Update

$$K(m,n) = P^{+}(m,n)H^{T}[HP^{+}(m,n)H^{T} + R_{nn}]^{-1}$$

$$\underline{\hat{f}}(m,n) = A\,\underline{\hat{f}}(m,n-1) + K(m,n)[\underline{y}(m,n) - HA\,\underline{\hat{f}}(m,n-1)]$$

$$P(m,n) = [I - K(m,n)H\,]P^{+}(m,n)$$

where

$$P^{+}(m,n) = E\left\{\left(\underline{f}(m,n) - \underline{\hat{f}}^{+}(m,n)\right)\left(\underline{f}(m,n) - \underline{\hat{f}}^{+}(m,n)\right)^{T}\right\}$$

$$P(m,n) = E\left\{\left(\underline{f}(m,n) - \underline{\hat{f}}(m,n)\right)\left(\underline{f}(m,n) - \underline{\hat{f}}(m,n)\right)^{T}\right\}$$

2. Variations of the Kalman filtering

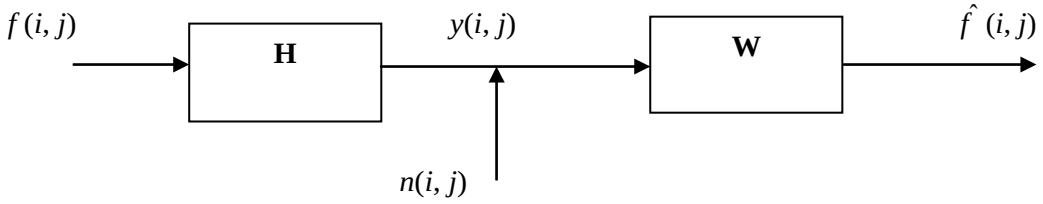
2.1 Reduced update Kalman filter (RUKF)

2.2 Reduced order model Kalman filter (ROMKF)

DIRECT METHODS

1. Wiener estimator (stochastic regularisation)

The image restoration problem can be viewed as a system identification problem as follows:



The objective is to minimize the following function

$$E\{(\mathbf{f}-\hat{\mathbf{f}})^{\mathbf{T}}(\mathbf{f}-\hat{\mathbf{f}})\}$$

To do so the following conditions should hold:

(i) $E\{\hat{\mathbf{f}}\}=E\{\mathbf{f}\}\Rightarrow E\{\mathbf{f}\}=\mathbf{W}E\{\mathbf{y}\}$

(ii) the error must be orthogonal to the observation about the mean

$$E\{(\hat{\mathbf{f}}-\mathbf{f})(\mathbf{y}-E\{\mathbf{y}\})^{\mathbf{T}}\}=0$$

From (i) and (ii) we have that

$$E\{(\mathbf{Wy}-\mathbf{f})(\mathbf{y}-E\{\mathbf{y}\})^{\mathbf{T}}\}=0\Rightarrow E\{(\mathbf{Wy}+E\{\mathbf{f}\}-\mathbf{W}E\{\mathbf{y}\}-\mathbf{f})(\mathbf{y}-E\{\mathbf{y}\})^{\mathbf{T}}\}=0\Rightarrow$$
$$E\{[\mathbf{W}(\mathbf{y}-E\{\mathbf{y}\})-(\mathbf{f}-E\{\mathbf{f}\})](\mathbf{y}-E\{\mathbf{y}\})^{\mathbf{T}}\}=0$$

If $\tilde{\mathbf{y}}=\mathbf{y}-E\{\mathbf{y}\}$ and $\tilde{\mathbf{f}}=\mathbf{f}-E\{\mathbf{f}\}$ then

$$E\{(\mathbf{W}\tilde{\mathbf{y}}-\tilde{\mathbf{f}})\tilde{\mathbf{y}}^{\mathbf{T}}\}=0\Rightarrow E\{\mathbf{W}\tilde{\mathbf{y}}\tilde{\mathbf{y}}^{\mathbf{T}}\}=E\{\tilde{\mathbf{f}}\tilde{\mathbf{y}}^{\mathbf{T}}\}\Rightarrow \mathbf{W}E\{\tilde{\mathbf{y}}\tilde{\mathbf{y}}^{\mathbf{T}}\}=E\{\tilde{\mathbf{f}}\tilde{\mathbf{y}}^{\mathbf{T}}\}\Rightarrow \mathbf{W}\mathbf{R}_{\tilde{\mathbf{y}}\tilde{\mathbf{y}}}=\mathbf{R}_{\tilde{\mathbf{f}}\tilde{\mathbf{y}}}$$

If the original and the degraded image are both zero mean then

$\mathbf{R}_{\tilde{\mathbf{y}}\tilde{\mathbf{y}}}=\mathbf{R}_{\mathbf{yy}}$ and $\mathbf{R}_{\tilde{\mathbf{f}}\tilde{\mathbf{y}}}=\mathbf{R}_{\mathbf{fy}}$.

In that case we have that $\mathbf{W}\mathbf{R}_{\mathbf{yy}}=\mathbf{R}_{\mathbf{fy}}$.

If we go back to the degradation model and find the autocorrelation matrix of the degraded image then we get that

$$\mathbf{y}=\mathbf{Hf}+\mathbf{n}\Rightarrow \mathbf{y}^{\mathbf{T}}=\mathbf{f}^{\mathbf{T}}\mathbf{H}^{\mathbf{T}}+\mathbf{n}^{\mathbf{T}}$$

$$E\{\mathbf{y}\mathbf{y}^{\mathbf{T}}\}=\mathbf{H}\mathbf{R}_{\mathbf{ff}}\mathbf{H}^{\mathbf{T}}+\mathbf{R}_{\mathbf{nn}}=\mathbf{R}_{\mathbf{yy}}$$

$$E\{\mathbf{f}\mathbf{y}^{\mathbf{T}}\}=\mathbf{R}_{\mathbf{ff}}\mathbf{H}^{\mathbf{T}}=\mathbf{R}_{\mathbf{fy}}$$

From the above we get the following result

$$\mathbf{W}=\mathbf{R}_{\mathbf{fy}}\mathbf{R}_{\mathbf{yy}}^{-1}=\mathbf{R}_{\mathbf{ff}}\mathbf{H}^{\mathbf{T}}(\mathbf{H}\mathbf{R}_{\mathbf{ff}}\mathbf{H}^{\mathbf{T}}+\mathbf{R}_{\mathbf{nn}})^{-1}$$

and the estimate for the original image is

$$\hat{\mathbf{f}}=\mathbf{R}_{\mathbf{ff}}\mathbf{H}^{\mathbf{T}}(\mathbf{H}\mathbf{R}_{\mathbf{ff}}\mathbf{H}^{\mathbf{T}}+\mathbf{R}_{\mathbf{nn}})^{-1}\mathbf{y}$$

Note that knowledge of $\mathbf{R}_{\mathbf{ff}}$ and $\mathbf{R}_{\mathbf{nn}}$ is assumed.

In frequency domain

$$W(u,v) = \frac{S_{ff}(u,v)H^*(u,v)}{S_{ff}(u,v)|H(u,v)|^2 + S_{nn}(u,v)}$$

$$\hat{F}(u,v) = \frac{S_{ff}(u,v)H^*(u,v)}{S_{ff}(u,v)|H(u,v)|^2 + S_{nn}(u,v)} Y(u,v)$$

Computational issues

The noise variance has to be known, otherwise it is estimated from a flat region of the observed image.

In practical cases where a single copy of the degraded image is available, it is quite common to use $S_{yy}(u,v)$ as an estimate of $S_{ff}(u,v)$. **This is very often a poor estimate !**

Wiener smoothing filter

In the absence of any blur, $H(u,v) = 1$ and

$$W(u,v) = \frac{S_{ff}(u,v)}{S_{ff}(u,v) + S_{nn}(u,v)} = \frac{(SNR)}{(SNR)+1}$$

(a) $(SNR) >> 1 \Rightarrow W(u,v) \cong 1$

(b) $(SNR) << 1 \Rightarrow W(u,v) \cong (SNR)$

(SNR) is high in low spatial frequencies and low in high spatial frequencies so $W(u,v)$ can be implemented with a lowpass (smoothing) filter.

Relation with inverse filtering

If $S_{nn}(u,v) = 0 \Rightarrow W(u,v) = \frac{1}{H(u,v)}$ which is the inverse filter

If $S_{nn}(u,v) \to 0$

$$\lim_{S_{nn}\to 0} W(u,v) = \begin{cases} \dfrac{1}{H(u,v)} & H(u,v) \neq 0 \\ 0 & H(u,v) = 0 \end{cases}$$

Geometric mean filter

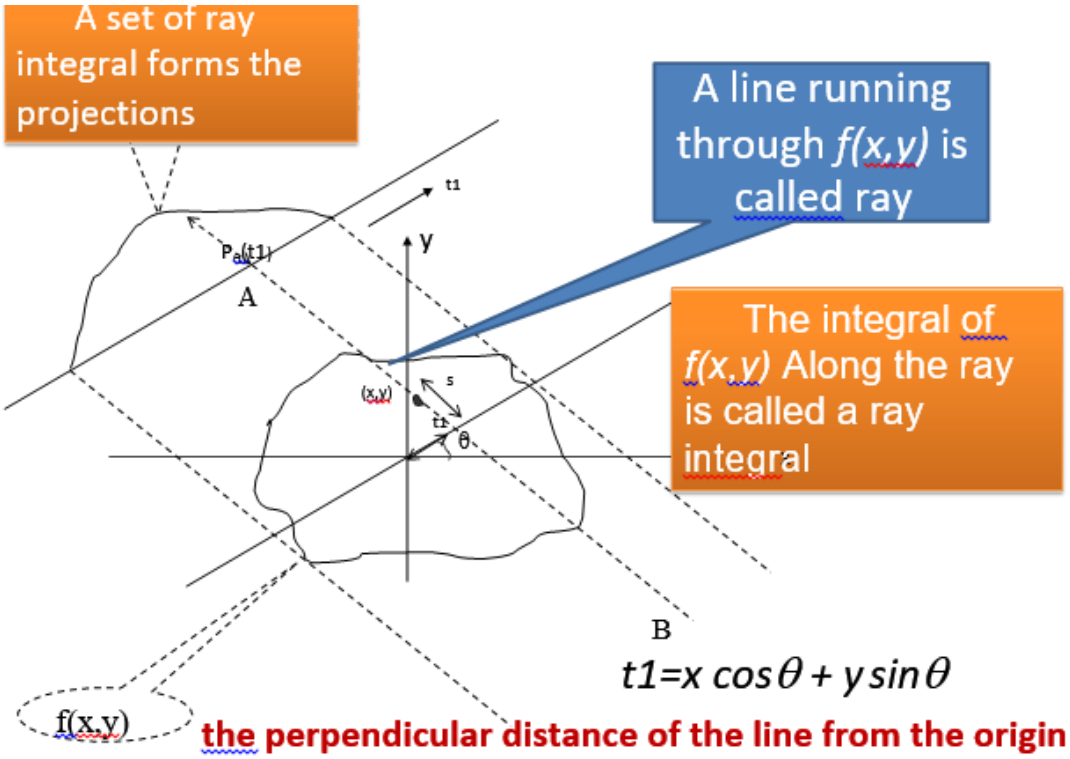
An image restored using a geometric mean filter is given by the expression

$$\hat{f}(x,y) = \left[\prod_{(s,t)\in S_{xy}} g(s,t) \right]^{\frac{1}{mn}}.$$

Here, each restored pixel is given by the product of the pixels in the sub image window, raised to the power 1/mn. As shown in Example 52, a geometric mean filter achieves smoothing comparable to the arithmetic mean filter, but it tends to lose less image detail in the process.

Image Reconstruction from projections

Image Reconstruction from projection is a special class of image restoration problem. Where a 2D object is reconstructed from several 1D projections. Each projection is obtained by projecting a parallel x-rays (or other penetrating radiation) beam through the object. Image Reconstruction from projection is a special class of image restoration problem. Where a 2D object is reconstructed from several 1D projections. Each projection is obtained by projecting a parallel x-rays (or other penetrating radiation) beam through the object.



Projections under all the angle θ will two dimensional representation of the image under which one coordinate is position in the projection profile t and, the other is the angle θ.

We can regard the parallel projection as a transformation, which transforms the image into another two-dimensional representation.

The integral of the function f(x,y) along the lines AB is called the ray integers and mathematically given by the function.

$$P_\theta(t_1) = \int f(x,y)ds$$

$$P_\theta(t1) = \int_{x=-\infty}^{+\infty} \int_{y=-\infty}^{+\infty} f(x,y)\delta(x\cos\theta + y\sin\theta - t_1)dx\,dy$$

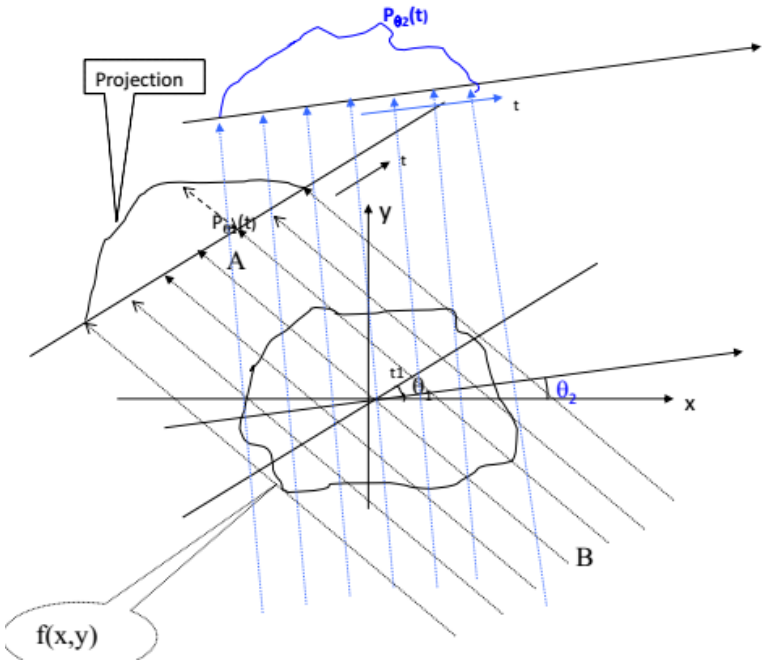
P_θ(t)as a function of t (for a given value of θ) defines the parallel projection of f(x,y) for an angleθ . The 2D function **P_θ(t) is called a Random Transform of f(x,y).**

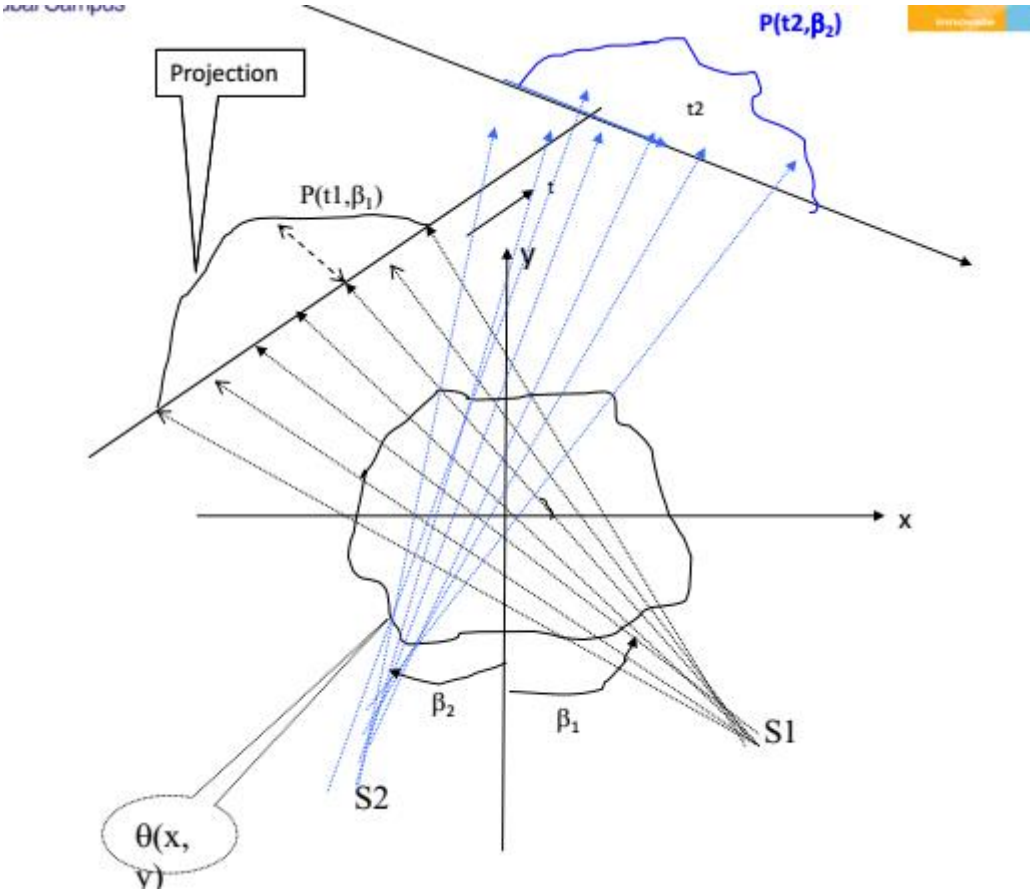
There are 2 methods to obtain projection data

*Parallel Projection

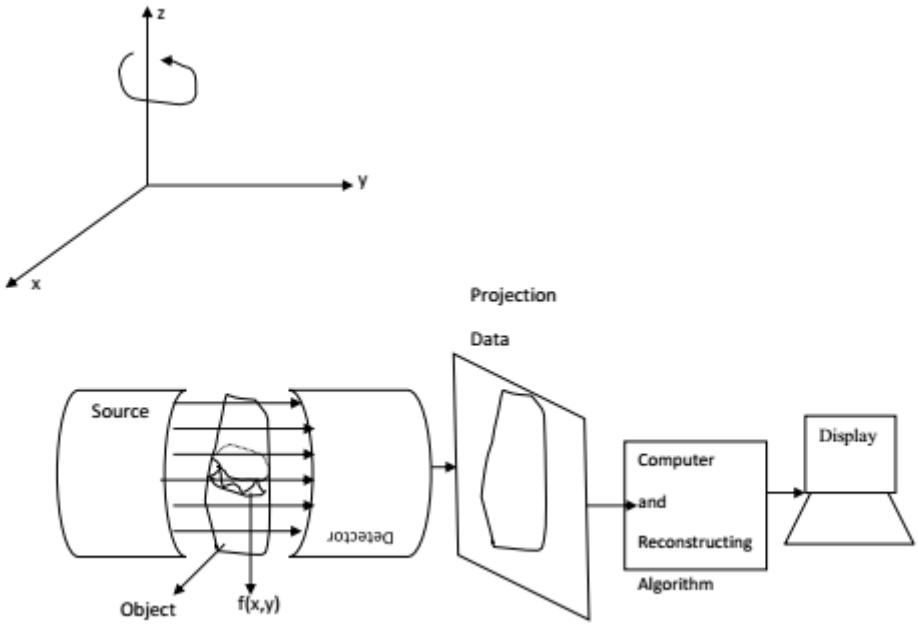
*Fan Beam Projection

The projection may also be generated by integrating a function along a set of lines originating from a point source, such projections are called as fan beam projection.





Methods for Generating Projection Data



UNIT- IV IMAGE COMPRESSION & WAVELETS AND MULTIRESOLUTION PROCESSING

Image Compression Fundamentals

Redundancies in a digital image.

The term data compression refers to the process of reducing the amount of data required to represent a given quantity of information. A clear distinction must be made between data and information. They are not synonymous. In fact, data are the means by which information is conveyed. Various amounts of data may be used to represent the same amount of information. Such might be the case, for example, if a long-winded individual and someone who is short and to the point were to relate the same story. Here, the information of interest is the story; words are the data used to relate the information. If the two individuals use a different number of words to tell the same basic story, two different versions of the story are created, and at least one includes nonessential data. That is, it contains data (or words) that either provide no relevant information or simply restate that which is already known. It is thus said to contain data redundancy.

Data redundancy is a central issue in digital image compression. It is not an abstract concept but a mathematically quantifiable entity. If n_1 and n_2 denote the number of information-carrying units in two data sets that represent the same information, the relative data redundancy R_D of the first data set (the one characterized by n_1) can be defined as

$$R_D = 1 - \frac{1}{C_R} \qquad C_R = \frac{n_1}{n_2}.$$

where C_R, commonly called the compression ratio, is

For the case $n_2 = n_1$, $C_R = 1$ and $R_D = 0$, indicating that (relative to the second data set) the first representation of the information contains no redundant data. When $n_2 << n_1$, $C_R \to \infty$ and $R_D \to 1$, implying significant compression and highly redundant data. Finally, when $n_2 >> n_1$, $C_R \to 0$ and $R_D \to \infty$, indicating that the second data set contains much more data than the original representation. This, of course, is the normally undesirable case of data expansion. In general, C_R and R_D lie in the open intervals $(0, \infty)$ and $(-\infty, 1)$, respectively. A practical compression ratio, such as 10 (or 10:1), means that the first data set has 10 information carrying units (say, bits) for every 1 unit in the second or compressed data set. The corresponding redundancy of 0.9 implies that 90% of the data in the first data set is redundant.

In this, we utilize formulation to show how the gray-level histogram of an image also can provide a great deal of insight into the construction of codes to reduce the amount of data used to represent it.

Let us assume, once again, that a discrete random variable r_k in the interval [0, 1] represents the gray levels of an image and that each r_k occurs with probability p_r (r_k).

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L - 1$$

where L is the number of gray levels, n_k is the number of times that the kth gray level appears in the image, and n is the total number of pixels in the image. If the number of bits used to represent each value of r_k is l (r_k), then the average number of bits required to represent each pixel is

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k).$$

That is, the average length of the code words assigned to the various gray-level values is found by summing the product of the number of bits used to represent each gray level and the probability that the gray level occurs. Thus the total number of bits required to code an M X N image is MNL_{avg}.

Interpixel Redundancy:

Consider the images shown in Figs. 1.1(a) and (b). As Figs. 1.1(c) and (d) show, these images have virtually identical histograms. Note also that both histograms are trimodal, indicating the presence of three dominant ranges of gray-level values. Because the gray levels in these images are not equally probable, variable-length coding can be used to reduce the coding redundancy that would result from a straight or natural binary encoding of their pixels. The coding process, however, would not alter the level of correlation between the pixels within the images. In other words, the codes used to represent the gray levels of each image have nothing to do with the correlation between pixels. These correlations result from the structural or geometric relationships between the objects in the image.

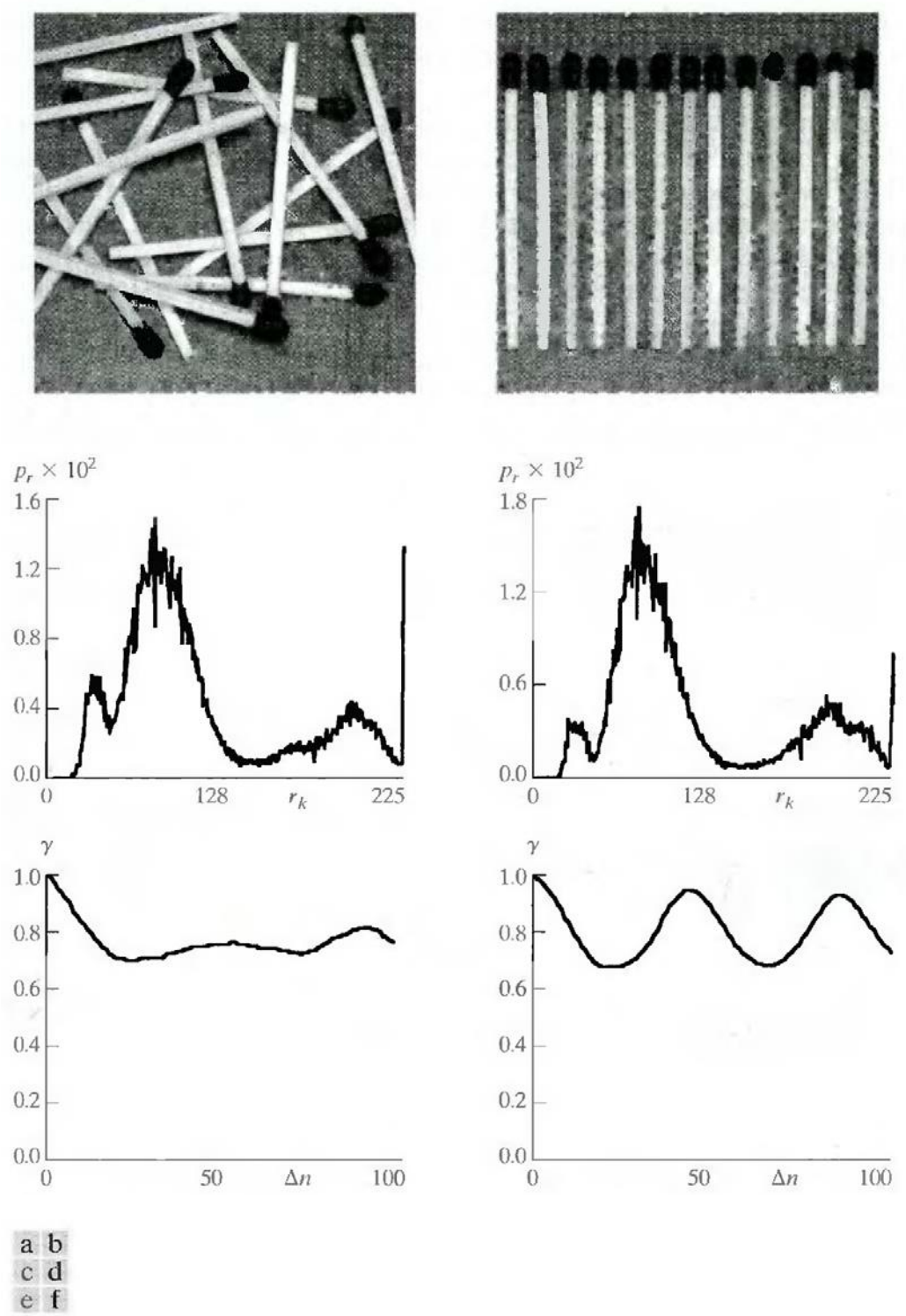


Fig.1.1 Two images and their gray-level histograms and normalized autocorrelation coefficients along one line.

Figures 1.1(e) and (f) show the respective autocorrelation coefficients computed along one line of each image.

$$\gamma(\Delta n) = \frac{A(\Delta n)}{A(0)}$$

where

$$A(\Delta n) = \frac{1}{N - \Delta n} \sum_{y=0}^{N-1-\Delta n} f(x, y) f(x, y + \Delta n).$$

The scaling factor in Eq. above accounts for the varying number of sum terms that arise for each integer value of Δn. Of course, Δn must be strictly less than N, the number of pixels on a line. The variable x is the coordinate of the line used in the computation. Note the dramatic difference between the shape of the functions shown in Figs. 1.1(e) and (f). Their shapes can be qualitatively related to the structure in the images in Figs. 1.1(a) and (b).This relationship is particularly noticeable in Fig. 1.1 (f), where the high correlation between pixels separated by 45 and 90 samples can be directly related to the spacing between the vertically oriented matches of Fig. 1.1(b). In addition, the adjacent pixels of both images are highly correlated. When Δn is 1, γ is 0.9922 and 0.9928 for the images of Figs. 1.1 (a) and (b), respectively. These values are typical of most properly sampled television images.

These illustrations reflect another important form of data redundancy—one directly related to the interpixel correlations within an image. Because the value of any given pixel can be reasonably predicted from the value of its neighbors, the information carried by individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of the values of its neighbors. A variety of names, including spatial redundancy, geometric redundancy, and interframe redundancy, have been coined to refer to these interpixel dependencies. We use the term interpixel redundancy to encompass them all.

In order to reduce the interpixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient (but usually "nonvisual") format. For example, the differences between adjacent pixels can be used to represent an image. Transformations of this type (that is, those that remove interpixel redundancy) are referred to as mappings. They are called reversible mappings if the original image elements can be reconstructed from the transformed data set.

Psychovisual Redundancy:

The brightness of a region, as perceived by the eye, depends on factors other than simply the light reflected by the region. For example, intensity variations (Mach bands) can be perceived in an area of constant intensity. Such phenomena result from the fact that the eye does not respond with equal sensitivity to all visual information. Certain information simply has less relative importance than other information in normal visual processing. This information is said to be psychovisually redundant. It can be eliminated without significantly impairing the quality of image perception.

That psychovisual redundancies exist should not come as a surprise, because human perception of the information in an image normally does not involve quantitative analysis of every pixel value in the image. In general, an observer searches for distinguishing features such as edges or textural regions and mentally combines them into recognizable groupings. The brain then correlates these groupings with prior knowledge in order to complete the image interpretation process. Psychovisual redundancy is fundamentally different from the redundancies discussed earlier. Unlike coding and interpixel redundancy, psychovisual redundancy is associated with real or quantifiable visual information. Its elimination is possible only because the information itself is not essential for normal visual processing. Since the elimination of psychovisually redundant data results in a loss of quantitative information, it is commonly referred to as quantization.

This terminology is consistent with normal usage of the word, which generally means the mapping of a broad range of input values to a limited number of output values. As it is an irreversible operation (visual information is lost), quantization results in lossy data compression.

Fidelity criterion.

The removal of psychovisually redundant data results in a loss of real or quantitative visual information. Because information of interest may be lost, a repeatable or reproducible means of quantifying the nature and extent of information loss is highly desirable. Two general classes of criteria are used as the basis for such an assessment:

A) Objective fidelity criteria and

B) Subjective fidelity criteria.

When the level of information loss can be expressed as a function of the original or input image and the compressed and subsequently decompressed output image, it is said to be based on an objective fidelity criterion. A good example is the root-mean-square (rms) error between an input and output image. Let f(x, y) represent an input image and let f(x, y) denote an estimate or approximation of f(x, y) that results from compressing and subsequently decompressing the input. For any value of x and y, the error e(x, y) between f (x, y) and f^ (x, y) can be defined as

$$e(x, y) = \hat{f}(x, y) - f(x, y)$$

so that the total error between the two images is

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]$$

where the images are of size M X N. The root-mean-square error, e_{rms}, between f(x, y) and f^(x, y) then is the square root of the squared error averaged over the M X N array, or

$$e_{rms} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]^2\right]^{1/2}$$

A closely related objective fidelity criterion is the mean-square signal-to-noise ratio of the compressed-decompressed image. If f^ (x, y) is considered to be the sum of the original image f(x, y) and a noise signal e(x, y), the mean-square signal-to-noise ratio of the output image, denoted SNR_{rms}, is

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x, y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]^2}.$$

The rms value of the signal-to-noise ratio, denoted SNR_{rms}, is obtained by taking the square root of Eq. above.

Although objective fidelity criteria offer a simple and convenient mechanism for evaluating information loss, most decompressed images ultimately are viewed by humans. Consequently, measuring image quality by the subjective evaluations of a human observer often is more appropriate. This can be accomplished by showing a "typical" decompressed image to an appropriate cross section of viewers and averaging their evaluations. The evaluations may be made using an absolute rating scale or by means of side-by-side comparisons of f(x, y) and f^(x, y).

Image compression models.

Fig. shows, a compression system consists of two distinct structural blocks: an encoder and a decoder. An input image f(x, y) is fed into the encoder, which creates a set of symbols from the input data. After transmission over the channel, the encoded representation is fed to the decoder, where a reconstructed output image f^(x, y) is generated. In general, f^(x, y) may or may not be an exact replica of f(x, y). If it is, the system is error free or information preserving; if not, some level of distortion is present in the reconstructed image. Both the encoder and decoder shown in Fig. consist of two relatively independent functions or subblocks. The encoder is made up of a source encoder, which removes input redundancies, and a channel encoder, which increases the noise immunity of the source encoder's output. As would be expected, the decoder includes a channel decoder followed by a source decoder. If the channel between the encoder and decoder is noise free (not prone to error), the channel encoder and decoder are omitted, and the general encoder and decoder become the source encoder and decoder, respectively.

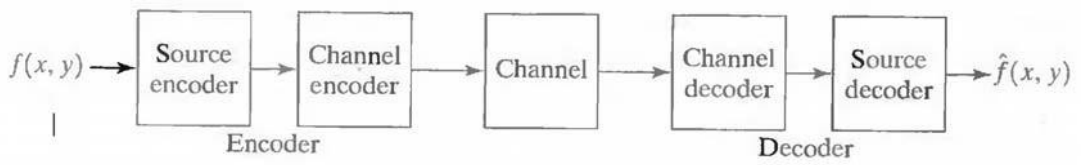
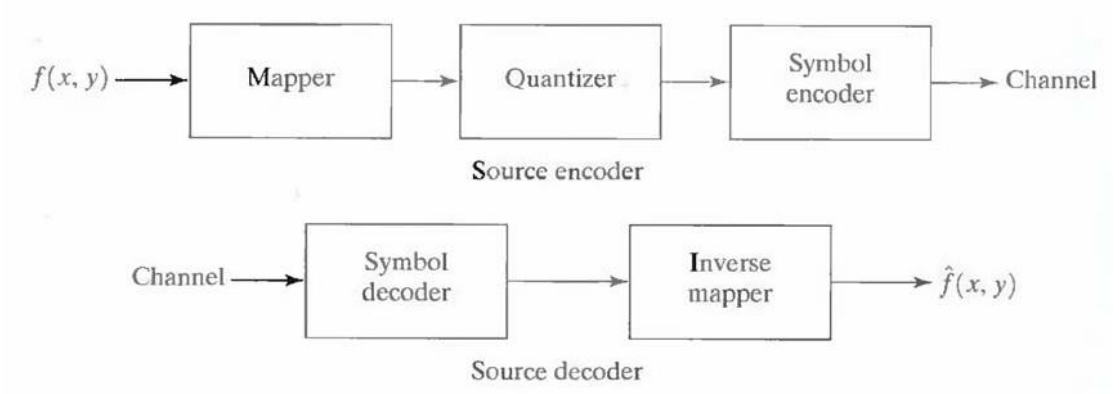


Fig. A general compression system model

The Source Encoder and Decoder:

The source encoder is responsible for reducing or eliminating any coding, interpixel, or psychovisual redundancies in the input image. The specific application and associated fidelity requirements dictate the best encoding approach to use in any given situation. Normally, the approach can be modeled by a series of three independent operations. As Fig. (a) shows, each operation is designed to reduce one of the three redundancies. Figure (b) depicts the corresponding source decoder. In the first stage of the source encoding process, the mapper transforms the input data into a (usually nonvisual) format designed to reduce interpixel redundancies in the input image. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image.



a
b

Fig. (a) Source encoder and (b) source decoder model

Run-length coding is an example of a mapping that directly results in data compression in this initial stage of the overall source encoding process. The representation of an image by a set of transform coefficients is an example of the opposite case. Here, the mapper transforms the image into an array of coefficients, making its interpixel redundancies more accessible for compression in later stages of the encoding process.

The second stage, or quantizer block in Fig. (a), reduces the accuracy of the mapper's output in accordance with some preestablished fidelity criterion. This stage reduces the psychovisual redundancies of the input image. This operation is irreversible. Thus it must be omitted when error-free compression is desired.

In the third and final stage of the source encoding process, the symbol coder creates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. The term symbol coder distinguishes this coding operation from the overall source encoding process. In most cases, a variable-length code is used to represent the mapped and quantized data set. It assigns the shortest code words to the most frequently occurring output values and thus reduces coding redundancy. The operation, of course, is reversible. Upon completion of the symbol coding step, the input image has been processed to remove each of the three redundancies.

Figure (a) shows the source encoding process as three successive operations, but all three operations are not necessarily included in every compression system. Recall, for example, that the quantizer must be omitted when error-free compression is desired. In addition, some compression techniques normally are modeled by merging blocks that are physically separate in

Fig. (a). In the predictive compression systems, for instance, the mapper and quantizer are often represented by a single block, which simultaneously performs both operations.

The source decoder shown in Fig. (b) contains only two components: a symbol decoder and an inverse mapper. These blocks perform, in reverse order, the inverse operations of the source encoder's symbol encoder and mapper blocks. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general source decoder model shown in Fig. (b).

The Channel Encoder and Decoder:

The channel encoder and decoder play an important role in the overall encoding-decoding process when the channel of Fig. is noisy or prone to error. They are designed to reduce the impact of channel noise by inserting a controlled form of redundancy into the source encoded data. As the output of the source encoder contains little redundancy, it would be highly sensitive to transmission noise without the addition of this "controlled redundancy." One of the most useful channel encoding techniques was devised by R. W. Hamming (Hamming [1950]). It is based on appending enough bits to the data being encoded to ensure that some minimum number of bits must change between valid code words. Hamming showed, for example, that if 3 bits of redundancy are added to a 4-bit word, so that the distance between any two valid code words is 3, all single-bit errors can be detected and corrected. (By appending additional bits of redundancy, multiple-bit errors can be detected and corrected.) The 7-bit Hamming (7, 4) code word $h_1, h_2, h_3 \ldots, h_6, h_7$ associated with a 4-bit binary number $b_3b_2b_1b_0$ is

$$
\begin{aligned}
h_1 &= b_3 \oplus b_2 \oplus b_0 \qquad & h_3 &= b_3 \\
h_2 &= b_3 \oplus b_1 \oplus b_0 \qquad & h_5 &= b_2 \\
h_4 &= b_2 \oplus b_1 \oplus b_0 \qquad & h_6 &= b_1 \\
& & h_7 &= b_0
\end{aligned}
$$

where $\oplus$ denotes the exclusive OR operation. Note that bits h_1, h_2, and h_4 are even- parity bits for the bit fields $b_3 b_2 b_0$, $b_3b_1b_0$, and $b_2b_1b_0$, respectively. (Recall that a string of binary bits has even parity if the number of bits with a value of 1 is even.) To decode a Hamming encoded result, the channel decoder must check the encoded value for odd parity over the bit fields in which even parity was previously established. A single-bit error is indicated by a nonzero parity word $c_4c_2c_1$, where

$$c_1 = h_1 \oplus h_3 \oplus h_5 \oplus h_7$$
$$c_2 = h_2 \oplus h_3 \oplus h_6 \oplus h_7$$
$$c_4 = h_4 \oplus h_5 \oplus h_6 \oplus h_7.$$

If a nonzero value is found, the decoder simply complements the code word bit position indicated by the parity word. The decoded binary value is then extracted from the corrected code word as $h_3h_5h_6h_7$.

Method of generating variable length codes with an example.

Variable-Length Coding:

The simplest approach to error-free image compression is to reduce only coding redundancy. Coding redundancy normally is present in any natural binary encoding of the gray levels in an image. It can be eliminated by coding the gray levels. To do so requires construction of a variable-length code that assigns the shortest possible code words to the most probable gray levels. Here, we examine several optimal and near optimal techniques for constructing such a code. These techniques are formulated in the language of information theory. In practice, the source symbols may be either the gray levels of an image or the output of a gray-level mapping operation (pixel differences, run lengths, and so on).

Huffman coding:

The most popular technique for removing coding redundancy is due to Huffman (Huffman [1952]). When coding the symbols of an information source individually, Huffman coding yields the smallest possible number of code symbols per source symbol. In terms of the noiseless coding theorem, the resulting code is optimal for a fixed value of n, subject to the constraint that the source symbols be coded one at a time.

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Figure 4.1 illustrates this process for binary coding (K-ary Huffman codes can also be constructed). At the far left, a hypothetical set of source symbols and their probabilities are ordered from top to bottom in terms of decreasing probability values. To form the first source reduction, the bottom two probabilities,
0.06 and 0.04, are combined to form a "compound symbol" with probability 0.1. This compound symbol and its associated probability are placed in the first source reduction column so that the

probabilities of the reduced source are also ordered from the most to the least probable. This process is then repeated until a reduced source with two symbols (at the far right) is reached.

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source, of course, is the symbols 0 and 1. As Fig. 4.2 shows, these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order of the 0 and 1 would work just as well). As the reduced source symbol with probability 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0 and 1 are arbitrarily

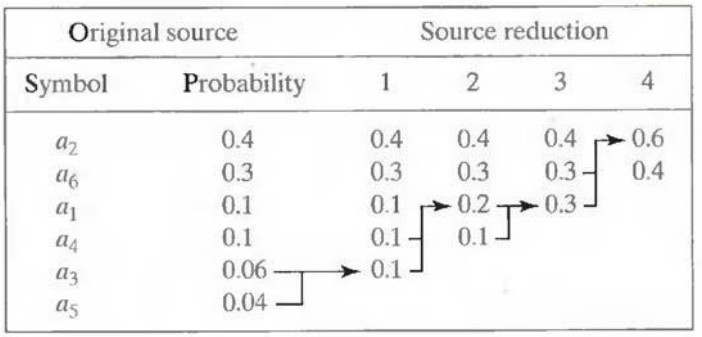


Fig. Huffman source reductions.

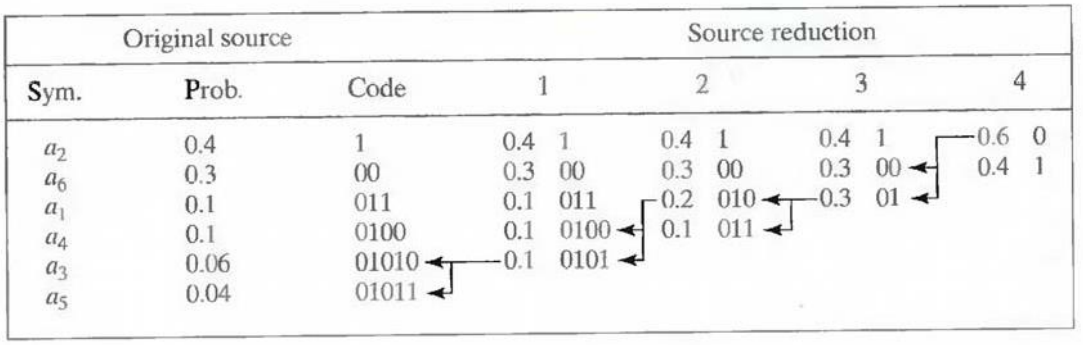


Fig. Huffman code assignment procedure.

appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original source is reached. The final code appears at the far left in Fig. . The average length of this code is

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$$
$$= 2.2 \text{ bits/symbol}$$

and the entropy of the source is 2.14 bits/symbol. The resulting Huffman code efficiency is 0.973.

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time. After the code has been created, coding and/or decoding is accomplished in a simple lookup table manner. The code itself is an instantaneous uniquely decodable block code. It is called a block code because each source symbol is mapped into a fixed sequence of code symbols. It is instantaneous, because each code word in a string of code symbols can be decoded without referencing succeeding symbols. It is uniquely decodable, because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a left to right manner. For the binary code of Fig. 4.2, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code word is 01010, which is the code for symbol a_3 .The next valid code is 011, which corresponds to symbol a_1. Continuing in this manner reveals the completely decoded message to be $a_3a_1a_2a_2a_6$.

Arithmetic encoding process with an example.

Arithmetic coding:

Unlike the variable-length codes described previously, arithmetic coding generates nonblock codes. In arithmetic coding, which can be traced to the work of Elias, a one-to-one correspondence between source symbols and code words does not exist. Instead, an entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval of real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller and the number of information units (say, bits) required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence. Because the technique does not require, as does Huffman's approach, that each source symbol translate into an integral number of code symbols (that is, that the symbols be coded one at a time), it achieves (but only in theory) the bound established by the noiseless coding theorem.

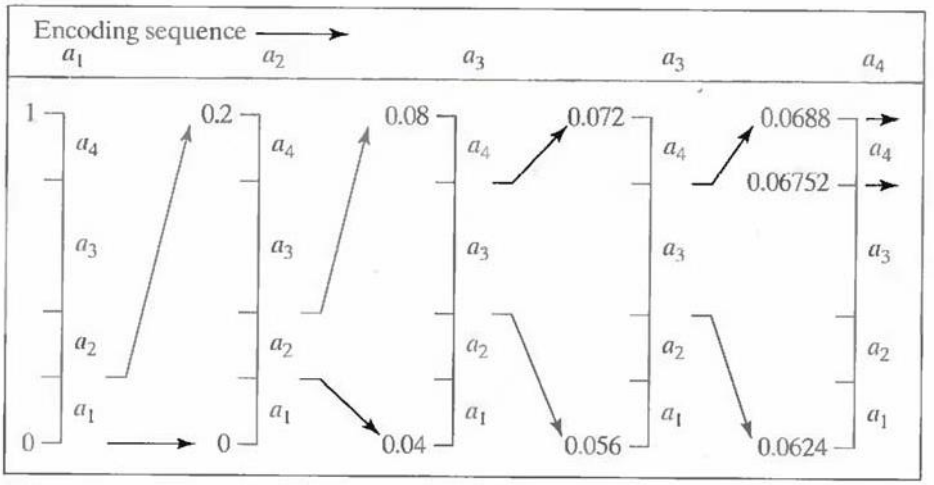


Fig. Arithmetic coding procedure

Figure illustrates the basic arithmetic coding process. Here, a five-symbol sequence or message, $a_1a_2a_3a_3a_4$, from a four-symbol source is coded. At the start of the coding process, the message is assumed to occupy the entire half-open interval [0, 1). As Table shows, this interval is initially subdivided into four regions based on the probabilities of each source symbol. Symbol ax, for example, is associated with subinterval [0, 0.2). Because it is the first symbol of the message being coded, the message interval is initially narrowed to [0, 0.2). Thus in Fig. [0, 0.2) is expanded to the full height of the figure and its end points labeled by the values of the narrowed range. The narrowed range is then subdivided in accordance with the original source symbol probabilities and the process continues with the next message symbol.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

Table Arithmetic coding example

In this manner, symbol a_2 narrows the subinterval to [0.04, 0.08), a_3 further narrows it to [0.056, 0.072), and so on. The final message symbol, which must be reserved as a special end-of-

message indicator, narrows the range to [0.06752, 0.0688). Of course, any number within this subinterval—for example, 0.068—can be used to represent the message.

In the arithmetically coded message of Fig. 5.1, three decimal digits are used to represent the five-symbol message. This translates into 3/5 or 0.6 decimal digits per source symbol and compares favorably with the entropy of the source, which is 0.58 decimal digits or 10-ary units/symbol. As the length of the sequence being coded increases, the resulting arithmetic code approaches the bound established by the noiseless coding theorem.

In practice, two factors cause coding performance to fall short of the bound: (1) the addition of the end-of-message indicator that is needed to separate one message from an- other; and (2) the use of finite precision arithmetic. Practical implementations of arithmetic coding address the latter problem by introducing a scaling strategy and a rounding strategy (Langdon and Rissanen [1981]). The scaling strategy renormalizes each subinterval to the [0, 1) range before subdividing it in accordance with the symbol probabilities. The rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent the coding subintervals from being represented accurately.

LZW coding with an example.

LZW Coding:

The technique, called Lempel-Ziv-Welch (LZW) coding, assigns fixed-length code words to variable length sequences of source symbols but requires no a priori knowledge of the probability of occurrence of the symbols to be encoded. LZW compression has been integrated into a variety of mainstream imaging file formats, including the graphic interchange format (GIF), tagged image file format (TIFF), and the portable document format (PDF).

LZW coding is conceptually very simple (Welch [1984]). At the onset of the coding process, a codebook or "dictionary" containing the source symbols to be coded is constructed. For 8-bit monochrome images, the first 256 words of the dictionary are assigned to the gray values 0, 1, 2..., and 255. As the encoder sequentially examines the image's pixels, gray- level sequences that are not in the dictionary are placed in algorithmically determined (e.g., the next unused) locations. If the first two pixels of the image are white, for instance, sequence “255- 255” might be assigned to location 256, the address following the locations reserved for gray levels 0 through 255. The next time that two consecutive white pixels are encountered, code word 256, the address of the location containing sequence 255-255, is used to represent them. If a 9-bit, 512-word dictionary is employed in the coding process, the original (8 + 8) bits that were used to represent the two pixels are replaced by a single 9-bit code word. Cleary, the size of the

dictionary is an important system parameter. If it is too small, the detection of matching gray- level sequences will be less likely; if it is too large, the size of the code words will adversely affect compression performance.

Consider the following 4 x 4, 8-bit image of a vertical edge:

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Table details the steps involved in coding its 16 pixels. A 512-word dictionary with the following starting content is assumed:

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

Locations 256 through 511 are initially unused. The image is encoded by processing its pixels in a left-to-right, top-to-bottom manner. Each successive gray-level value is concatenated with a variable—column 1 of Table 6.1 —called the "currently recognized sequence." As can be seen, this variable is initially null or empty. The dictionary is searched for each concatenated sequence and if found, as was the case in the first row of the table, is replaced by the newly concatenated and recognized (i.e., located in the dictionary) sequence. This was done in column 1 of row 2.

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

Table LZW coding example

No output codes are generated, nor is the dictionary altered. If the concatenated sequence is not found, however, the address of the currently recognized sequence is output as the next encoded value, the concatenated but unrecognized sequence is added to the dictionary, and the currently recognized sequence is initialized to the current pixel value. This occurred in row 2 of the table. The last two columns detail the gray-level sequences that are added to the dictionary when scanning the entire 4 x 4 image. Nine additional code words are defined. At the conclusion of coding, the dictionary contains 265 code words and the LZW algorithm has successfully identified several repeating gray-level sequences—leveraging them to reduce the original 128-bit image lo 90 bits (i.e., 10 9-bit codes). The encoded output is obtained by reading the third column from top to bottom. The resulting compression ratio is 1.42:1.

A unique feature of the LZW coding just demonstrated is that the coding dictionary or code book is created while the data are being encoded. Remarkably, an LZW decoder builds an identical decompression dictionary as it decodes simultaneously the encoded data stream. . Although not needed in this example, most practical applications require a strategy for handling dictionary overflow. A simple solution is to flush or reinitialize the dictionary when it becomes full and continue coding with a new initialized dictionary. A more complex option is

to monitor compression performance and flush the dictionary when it becomes poor or unacceptable. Alternately, the least used dictionary entries can be tracked and replaced when necessary.

Concept of bit plane coding method.

Bit-Plane Coding:

An effective technique for reducing an image's interpixel redundancies is to process the image's bit planes individually. The technique, called bit-plane coding, is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images and compressing each binary image via one of several well-known binary compression methods.

Bit-plane decomposition:

The gray levels of an m-bit gray-scale image can be represented in the form of the base 2 polynomial

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_1 2^1 + a_0 2^0.$$

Based on this property, a simple method of decomposing the image into a collection of binary images is to separate the m coefficients of the polynomial into m 1-bit bit planes. The zeroth- order bit plane is generated by collecting the a_0 bits of each pixel, while the (m - 1) st-order bit plane contains the a_{m-1}, bits or coefficients. In general, each bit plane is numbered from 0 to m-1 and is constructed by setting its pixels equal to the values of the appropriate bits or polynomial coefficients from each pixel in the original image. The inherent disadvantage of this approach is that small changes in gray level can have a significant impact on the complexity of the bit planes. If a pixel of intensity 127 (01111111) is adjacent to a pixel of intensity 128 (10000000), for instance, every bit plane will contain a corresponding 0 to 1 (or 1 to 0) transition. For example, as the most significant bits of the two binary codes for 127 and 128 are different, bit plane 7 will contain a zero-valued pixel next to a pixel of value 1, creating a 0 to 1 (or 1 to 0) transition at that point.

An alternative decomposition approach (which reduces the effect of small gray-level variations) is to first represent the image by an m-bit Gray code. The m-bit Gray code $g_{m-1}... g_2 g_1 g_0$ that corresponds to the polynomial in Eq. above can be computed from

$$g_i = a_i \oplus a_{i+1} \qquad 0 \le i \le m-2$$
$$g_{m-1} = a_{m-1}.$$

Here, ⊗denotes the exclusive OR operation. This code has the unique property that successive code words differ in only one bit position. Thus, small changes in gray level are less likely to affect all m bit planes. For instance, when gray levels 127 and 128 are adjacent, only the 7th bit plane will contain a 0 to 1 transition, because the Gray codes that correspond to 127 and 128 are 11000000 and 01000000, respectively.

Lossless predictive coding.

Lossless Predictive Coding:

The error-free compression approach does not require decomposition of an image into a collection of bit planes. The approach, commonly referred to as lossless predictive coding, is based on eliminating the interpixel redundancies of closely spaced pixels by extracting and coding only the new information in each pixel. The new information of a pixel is defined as the difference between the actual and predicted value of that pixel.

Figure 8.1 shows the basic components of a lossless predictive coding system. The system consists of an encoder and a decoder, each containing an identical predictor. As each successive pixel of the input image, denoted f_n, is introduced to the encoder, the predictor generates the anticipated value of that pixel based on some number of past inputs. The output of the predictor is then rounded to the nearest integer, denoted $f\hat{}_n$ and used to form the difference or prediction error which is coded using a variable-length code (by the symbol encoder) to generate the next element of the compressed data stream.

$$e_n = f_n - \hat{f}_n,$$

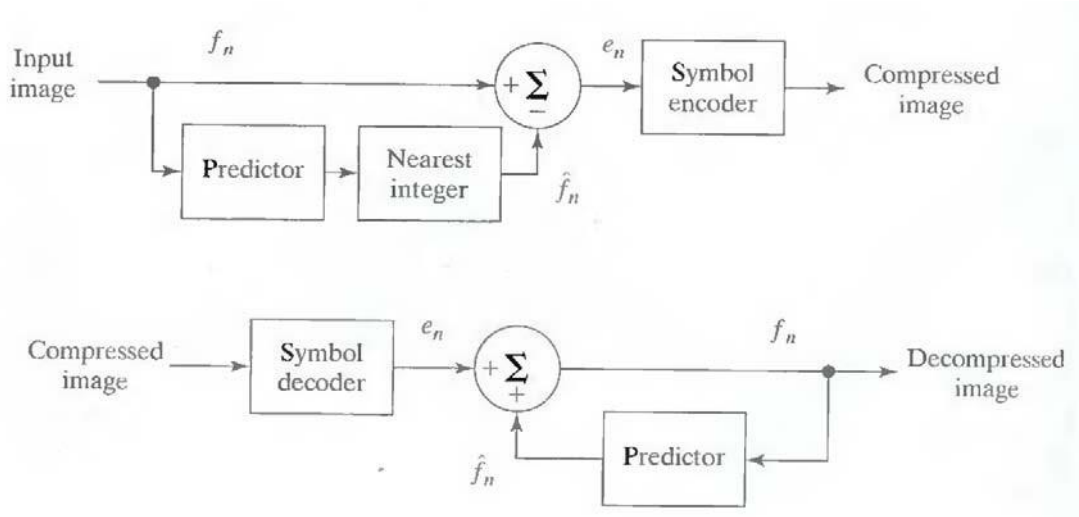


Fig. A lossless predictive coding model: (a) encoder; (b) decoder

The decoder of Fig. 8.1 (b) reconstructs en from the received variable-length code words and performs the inverse operation

$$f_n = e_n + \hat{f}_n.$$

Various local, global, and adaptive methods can be used to generate $f\hat{}_n$. In most cases, however, the prediction is formed by a linear combination of m previous pixels. That is,

$$\hat{f}_n = \text{round}\left[\sum_{i=1}^{m} \alpha_i f_{n-i}\right]$$

where m is the order of the linear predictor, round is a function used to denote the rounding or nearest integer operation, and the α_i, for i = 1,2,..., m are prediction coefficients. In raster scan applications, the subscript n indexes the predictor outputs in accordance with their time of occurrence. That is, f_n, $f\hat{}_n$ and e_n in Eqns. above could be replaced with the more explicit notation f (t), f^(t), and e (t), where t represents time. In other cases, n is used as an index on the spatial coordinates and/or frame number (in a time sequence of images) of an image. In 1-D linear predictive coding, for example, Eq. above can be written as

$$\hat{f}_n(x, y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x, y - i)\right]$$

where each subscripted variable is now expressed explicitly as a function of spatial coordinates x and y. The Eq. indicates that the 1-D linear prediction f(x, y) is a function of the previous pixels on the current line alone. In 2-D predictive coding, the prediction is a function of the previous pixels in a left-to-right, top-to-bottom scan of an image. In the 3-D case, it is based on these pixels and the previous pixels of preceding frames. Equation above cannot be evaluated for the first m pixels of each line, so these pixels must be coded by using other means (such as a Huffman code) and considered as an overhead of the predictive coding process. A similar comment applies to the higher-dimensional cases.

Lossy predictive coding.

Lossy Predictive Coding:

In this type of coding, we add a quantizer to the lossless predictive model and examine the resulting trade-off between reconstruction accuracy and compression performance. As Fig.9 shows, the quantizer, which absorbs the nearest integer function of the error-free encoder, is inserted between the symbol encoder and the point at which the prediction error is formed. It maps the prediction error into a limited range of outputs, denoted $e^{\wedge}_n$ which establish the amount of compression and distortion associated with lossy predictive coding.

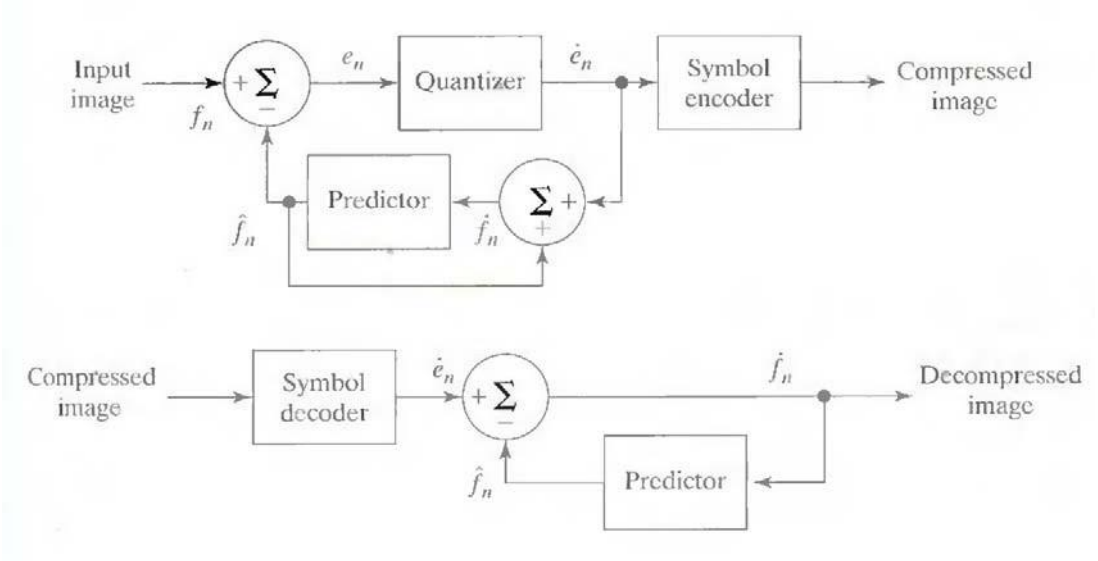


Fig. A lossy predictive coding model: (a) encoder and (b) decoder.

In order to accommodate the insertion of the quantization step, the error-free encoder of figure must be altered so that the predictions generated by the encoder and decoder are equivalent. As Fig.9 (a) shows, this is accomplished by placing the lossy encoder's predictor within a feedback loop, where its input, denoted $\dot{f}_n$, is generated as a function of past predictions and the corresponding quantized errors. That is,

$$\dot{f}_n = \dot{e}_n + \hat{f}_n$$

This closed loop configuration prevents error buildup at the decoder's output. Note from Fig. 9 (b) that the output of the decoder also is given by the above Eqn.

Optimal predictors:

The optimal predictor used in most predictive coding applications minimizes the encoder's mean-square prediction error

$$E\{e_n^2\} = E\{[f_n - \hat{f}_n]^2\}$$

subject to the constraint that

$$\dot{f}_n = \dot{e}_n + \hat{f}_n \approx e_n + \hat{f}_n = f_n$$

and

$$\hat{f}_n = \sum_{i=1}^{m} \alpha_i f_{n-i}.$$

That is, the optimization criterion is chosen to minimize the mean-square prediction error, the quantization error is assumed to be negligible ($\dot{e}_n \approx e_n$), and the prediction is constrained to a linear combination of m previous pixels.1 These restrictions are not essential, but they simplify the analysis considerably and, at the same time, decrease the computational complexity of the predictor. The resulting predictive coding approach is referred to as differential pulse code modulation (DPCM).

Block diagram about transform coding system.

Transform Coding:

All the predictive coding techniques operate directly on the pixels of an image and thus are spatial domain methods. In this coding, we consider compression techniques that are based on modifying the transform of an image. In transform coding, a reversible, linear transform (such as the Fourier transform) is used to map the image into a set of transform coefficients, which are then quantized and coded. For most natural images, a significant number of the coefficients have small magnitudes and can be coarsely quantized (or discarded entirely) with little image distortion. A variety of transformations, including the discrete Fourier transform (DFT), can be used to transform the image data.

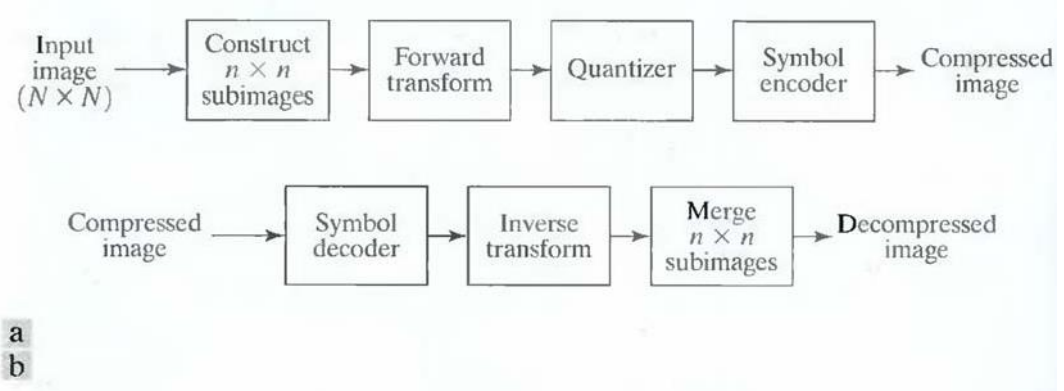


Fig. A transform coding system: (a) encoder; (b) decoder.

Figure shows a typical transform coding system. The decoder implements the inverse sequence of steps (with the exception of the quantization function) of the encoder, which performs four relatively straightforward operations: subimage decomposition, transformation, quantization, and coding. An N X N input image first is subdivided into subimages of size n X n,
which are then transformed to generate $(N/n)^2$ subimage transform arrays, each of size n X n.
The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients. The quantization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least information. These coefficients have the smallest impact on reconstructed subimage quality. The encoding process terminates by coding (normally using a variable-length code) the quantized coefficients. Any or all of the transform encoding steps can be adapted to

local image content, called adaptive transform coding, or fixed for all subimages, called nonadaptive transform coding.

Image Pyramid

What is the Image Pyramid? Of course it has nothing to do with the ancient Egypt. Image Pyramid-formally called "pyramid representation of image"- is a image and signal processing technique, to represent a single image using a set of cascading images. Image pyramid provides many useful properties for many application, such as noise reduction, image analysis, image enhancement, etc.

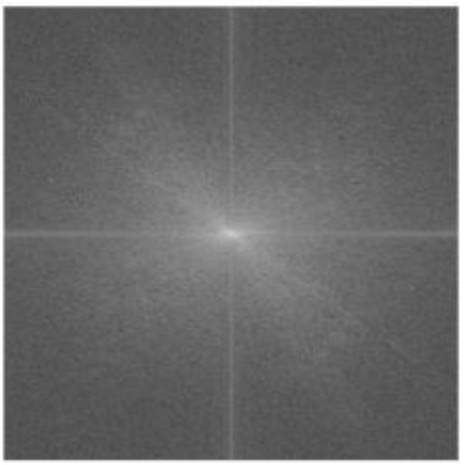
Why we need these Pyramids?

Images that we can see in everyday life are represented using so-called ***Spatial domain,*** which is actually the things that most people will think of when talk about images. This spatial domain represents images by the luminance values of each image's location. There are many processing operations that can be done with the spatial domain, such as basic low-pass filter and high-pass filter, adaptive filter, median filter, and many other things end with filter. Normally most of these filters work by using local information from each location. However, since images mostly contain complex informations, sometimes they are hard works to process in spatial domain.

Lena in Spatial domain

There is another popular type of image representation known as the ***frequency domain***. If we perform the *Fast Fourier Transform* to the image, we will get the totally different image called frequency spectrum.

For those who didn't know about the frequency domain, let's simply explain like this: From the signal theory, every signal (including image, of course) can be express as a linear combination of a set of various-frequency sine and cosine signals. The process to transform from normal signal to frequency domain is called Fourier Transformation, and the transformed result – in this case, frequency spectrum- is actually a set of coefficients of each sine and cosine signals. In our case, low frequency represents smooth details, shapes, and colour in the image, and high frequency represents fine details and noise in the image.
The introducing of frequency domain make more possibility for image processing. For example, we can just discard the high frequency responses to remove most noises from the image! Sounds great except that the beautiful object edge also gone with the noises. A major problem of frequency domain is that it will mix all frequency from every object in the image together, formally called *"loss of locality"*, and most of the time we don't want to mix our faces' beautiful details with the detail of the grass in background.



Lena in frequency domain

In order to get the advantages of both spatial domain and frequency domain, the new representation is invented and called ***spatial-frequency*** domain. This new domain give us the ability to deal with separate frequency easily as well as preserve the locality of the information. Our image pyramid is also in this domain.

What is exactly this Pyramid?

The image pyramid is actually a representation of the image by a set of the different frequency-band images . For example, if we put our original Lena image to construct her pyramid, we may get, for simplicity, 3 layers of pyramid. The first image will represent low frequency band (smooth detail), the second image will represent middle frequency band (some detail), and the last image will represent high frequency band (finest detail). Image Pyramid is also called "multi-scale" and "multi-resolution" because of this characteristic.

Example of Gaussian Pyramid Example of Laplacian Pyramid

How to construct this Pyramid?

To construct the pyramid, we have 2 options here. Since we know that the each layer of pyramid represents a different band of image's frequency, we may use multiple filters, a filter for each band, to convolute with the image. Some of us may already known that just only a single time of convolution will need some effort, do not try to imagine multiple time of it. (Convolution literally means "things that extremely complicated and difficult to follow")

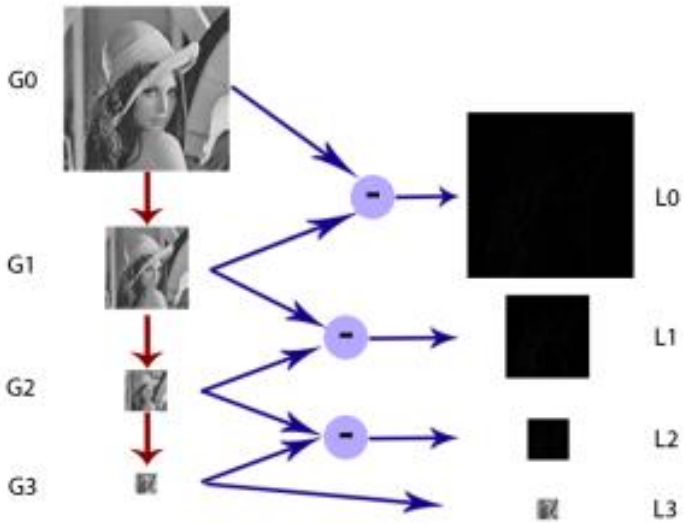
Another option is , according to some great-mind persons, instead of using multiple filters to the image, we can use multiple image to convolute with a single filter! The best part is these multiple images are obtains by reducing size of the original image. This way is much better than the first option. So we will use this technique to construct Lena's pyramid.

How to construct Guassian Pyramid

Comparison of each layers in the same size

We can notice that each pyramid layer is constructed by applying the low-pass filter to the upper part and then reduce its size by the factor of 0.5. This process is called "*Reduce operation*". Because each layer of this pyramid is a low-pass filtered image with a different filter range, this pyramid is called ***"Gaussian Pyramid"*** or the pyramid of low-pass filtered image.

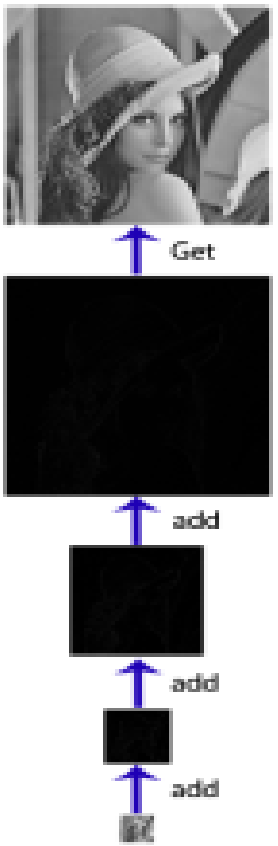
The Gaussian Pyramid itself still doesn't have much usefulness. However we can use it to construct another pyramid called ***"Laplacian Pyramid"*** or the pyramid of band-pass filtered image. Each layer of this pyramid is made from subtract two consecutive layer from Gaussian Pyramid together. In order to subtract, the lower later need to be upsampled to be the same size to the upper layer, and this process called "*Expand operation*".



How to construct Laplacian Pyramid

This Laplacian Pyramid is the true useful member of the image pyramid. Each layer of this pyramid is the band-pass image, which means we can now do some things to the specific frequency just like in the frequency domain. We also see that even after its frequencies are shown, the local features of the image are still there. By using this pyramid we can get the so-called spatial-frequency domain as explained in the beginning.

After we finish our work with this pyramid and need our Lena back, we can just sum every layer of the Laplacian Pyramid together so that we can get back the original image (or result image after we have done something). This process is called "*Reconstruction*"
Reconstruction



In conclusion, the image pyramid is another technique to represent the image. Unlike another domain such as spatial domain and frequency domain, because the image pyramid, which is the spatial-frequency domain, have the advantages of both domain. Moreover, the computation of image pyramid is much easier than that of the frequency domain

Subband coding & HAAR TRANSFORMS MULTI RESOLUTION EXPRESSIONS:

HAAR TRANSFORM:

The Haar transform is both seperable and symmetric and can be expressed in matrix form

T = HFH

Where **F** is an N X N image matrix, **H** is an N XN transformation matrix, and **T** is the resulting N X N transform. For the Haar transform, transformation matrix **H** contains the Haar basis function, $h_k(Z)$.

They are defined over the continuous, closed interval $z \in [0,1]$ for k=0,1,2,...., N – 1, where $N = 2^n$. To generate **H**, consider the integer k such that $k = 2^p + q - 1$, where $0 \le p \le n - 1$, q = 0 or 1 for p = 0, and $1 \le q \le 2^p$ for $p \ne 0$. Then the Haar basis function are

$$h_0(z) = h_{00}(z) = \frac{1}{\sqrt{N}}, \qquad z \in [0, 1]$$

and

$$h_k(z) = h_{pq}(z) = \frac{1}{\sqrt{N}} \begin{cases} 2^{\frac{p}{2}} & \frac{q-1}{2^p} \le z < \frac{q-0.5}{2^p} \\ -2^{\frac{p}{2}} & \frac{q-0.5}{2^p} \le z < \frac{q}{2^p} \\ 0 & otherwise, z \in [0, 1] \end{cases}$$

The i*th* row of an N X N Haar transformation matrix contains the elements of $h_i(z)$ for z = 0/N, 1/N, 2/N,....., (N – 1)/N. If N = 4, for example, k,q, and p assume the values

k	p	q
0	0	0
1	0	1
2	1	1
3	1	2

and the 4 X 4 transformation matrix, $\mathbf{H_4}$, is

$$\mathbf{H_4} = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

In a similar manner, the 2 X 2 transformation matrix, $\mathbf{H_2}$, is

$$\mathbf{H_2} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

MULTI RESOLUTION EXPANSIONS:

In Multi Resolution Analysis (MRA), a scaling function is used to create a series of approximations of a function or image, each differing by a factor of 2 from its neighbouring approximations. Additional functions, called wavelets, are then used to encode the difference in information between adjacent approximations.

1. **Series expansion:**

A signal or function f (x) can often be better analyzed as a linear combination of expansion functions.

$f(x) = \sum_k \propto_k \varphi_k(x)$

where k is an integer index of the finite or infinite sum, the $\propto_k$ are real valued expansion coefficients, and the $\varphi_k(x)$ are real- valued expansion functions.

If the expansion is unique, there is one set of $\propto_k$ for any given f(x), the $\varphi_k(x)$ are called basis functions, and the expansion set, $\{\varphi_k(x)\}$, is called a basis for the class of functions. The expressible functions form a function space that is referred to as the closed span of the expansion set V.

$$V = \overline{\text{Span}\{\varphi_k(x)\}}$$

2. Scaling Expansion:

Consider the set of expansion functions composed of integer translations and binary scalings of the real, square-integrable function $\varphi(x)$; that is, the set $\{\varphi_{j,k}(x)\}$ where

$\varphi_{j,k}(x) = 2^{j/2}\ \varphi(2^j x - k)$

For all j, k ϵ **Z** and $\varphi(x) \epsilon L^2(R)$. Here, k determines the position of $\varphi_{j,k}(x)$ along the x-axis , j determines $\varphi_{j,k}(x)$'s width and $2^{j/2}$ controls its height or amplitude. Because the shape of $\varphi_{j,k}(x)$ changes with j, $\varphi(x)$ is called a *scaling function.*

3. Wavelet Functions:

A wavelet function $\Psi(x)$ together with its integer translates and binary scaling, spans the difference between any two adjacent scaling subspaces, V_j and V_{j+1} . This situation is illustrated graphically in Fig. 5. The set $\{\Psi_{j,k}(x)\}$ of wavelets is defined as

$\Psi_{j,k}(x) = 2^{j/2}\ \Psi(2^j x - k)$

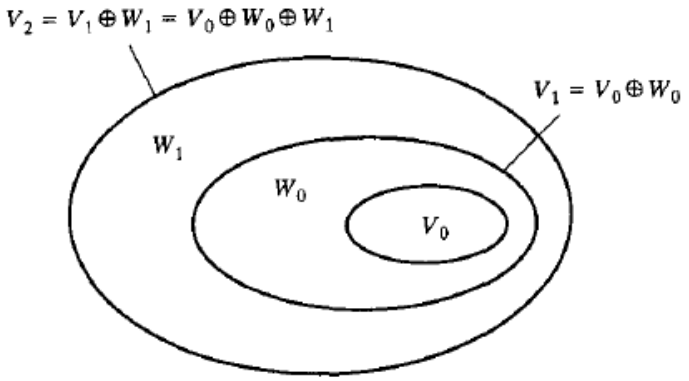


Fig. 5: The relationship between scaling and wavelet function spaces.

For all k ϵ Z that spans the W_j spaces in the figure. The scaling functions can be written as

$$W_j = \overline{\text{Span}\{\Psi_{j,k}(x)\}}$$

Any wavelet function can be expressed as a weighted sum of shifted, double-resolution scaling functions.

$$\Psi(x) = \sum_n h_\Psi(n)\sqrt{2}\ \varphi(2x - n)$$

where the $h_\Psi(n)$ are called the *wavelet function coefficients* and $h_\Psi$is called the *wavelet vector*.

WAVELET TRANSFORMS IN ONE DIMENSION:

1. Wavelet Series Expansions:

The wavelet series expansion of function f(x) ϵ L^2(R) relative to wavelet Ψ(x) and scaling function φ(x).

f(x) = $\sum_k c_{j_0}(k)\ \varphi_{j_0,k}(x) + \sum_{j=j_0}^{\infty} \sum_k d_j(k)\ \Psi_{j,k}(x)$ .....eq.(1)

where j_0 is an arbitrary starting scale

The $c_{j_0}(k)$'s are normally called the *approximation* or *scaling coefficients*;

The $d_j(k)$'s are referred to as the *detail* or *wavelet coefficients.* This is because the first sum in eq.(1) uses scaling function to provide an approximation of f(x) at scale j_0. For each higher scale j $\geq j_0$ in the second sum is added to the approximation to provide increasing detail. If the expansion functions forms an orthogonal basis or tight frame, which is often the case, the expansion coefficients are calculated.

$c_{j_0}(k) = \langle f(x), \varphi_{j_0,k}(x) \rangle = \int f(x)\ \varphi_{j_0,k}(x)\ dx$ .....eq.(2)

and $d_j(k) = \langle f(x), \Psi_{j,k}(x) \rangle = \int f(x)\ \Psi_{j,k}(x)\ dx$ .....eq.(3)

2. The Discrete Wavelet Transform

The wavelet series expansion maps a function of a continuous variable into a sequence of coefficients. If the function being expanded is a sequence of numbers, like samples of a continuous function f(x), the resulting coefficient are called the *discrete wavelet transform* (DWT).

For this case, the series expansion defined through Eqs (1) through (3) becomes the DWT transform pair.

$$W_{\varphi}(j_0, k) = \frac{1}{\sqrt{M}} \sum_x f(x)\ \varphi_{j_0,k}(x)$$

$$W_{\Psi}(j, k) = \frac{1}{\sqrt{M}} \sum_x f(x)\ \Psi_{j,k}(x)$$

For j $\geq j_0$ and

$$f(x) = \frac{1}{\sqrt{M}} \sum_k W_{\varphi}(j_0, k)\ \varphi_{j_0,k}(x) + \frac{1}{\sqrt{M}} \sum_{j=j_0}^{\infty} \sum_k W_{\Psi}(j, k)\ \Psi_{j,k}(x)$$

Here, f(x), $\varphi_{j_0,k}(x)$, and $\Psi_{j,k}(x)$ are functions of the discrete variable x = 0,1,2,.....,M-1.

3. The Continuous Wavelet Transform:

The natural extension of the discrete wavelet transform is the *continuous wavelet transform* (CWT), which transforms a continuous function into a highly redundant function of two continuous variables – translation and scale. The resulting transform is easy to interpret and valuable for time-frequency analysis.

The continuous wavelet transform of a continuous, square-integrable function, f(x), relative to a real-valued wavelet, Ψ (x), is

$$W_{\Psi}(s, \tau) = \int_{-\infty}^{\infty} f(x)\, \Psi_{s,k}(x)\, dx$$

$$\text{Where } \Psi_{s,\tau}(x) = \frac{1}{\sqrt{s}} \Psi\left(\frac{x-\tau}{s}\right)$$

And s and τ are called *scale* and *translation* parameters.

The inverse continuous wavelet transform is given as

$$f(x) = \frac{1}{C_{\Psi}} \int_{0}^{\infty} \int_{-\infty}^{\infty} W_{\Psi}(s, \tau) \frac{\Psi_{s,\tau}(x)}{s^2}\, d\tau\, ds$$

Where

$$C_{\Psi} = \int_{-\infty}^{\infty} \frac{|\Psi(u)|^2}{|u|}\, du$$

And Ψ(u) is the fourier transform of Ψ(x)

THE FAST WAVELET TRANSFORM:

The fast wavelet transform (FWT) is computationally efficient implementation of the discrete wavelet transform (DWT) that exploits a relationship between the coefficients of the DWT at adjacent scales. The FWT resembles the two band subband coding scheme.

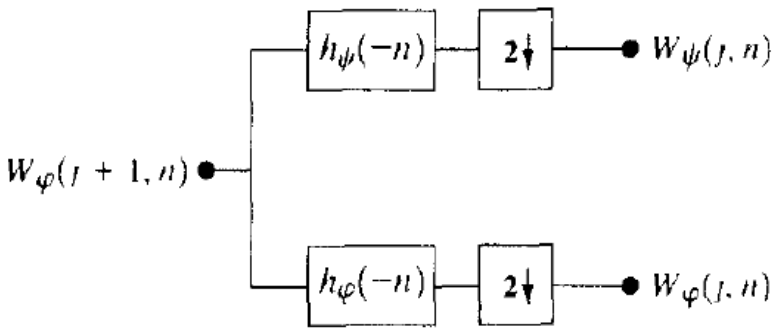


Fig.6: An FWT analysis bank.

The Fig.6 shows the FWT analysis filter bank and this filter bank can be "iterated" to create multistage structures for computing DWT coefficients at two or more successive scales.

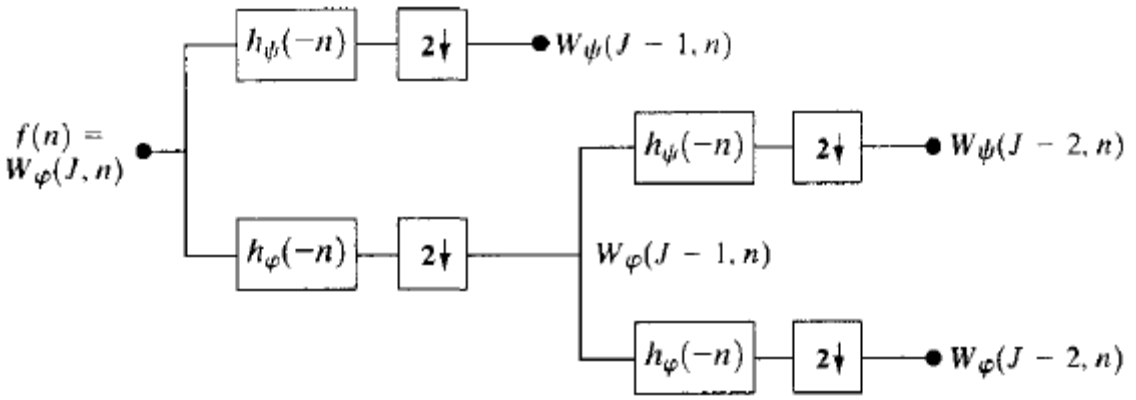


Fig. 7: A two stage or two scale FWT analysis bank

For example, Fig.7 shows a two stage filter bank for generating the coefficients at the two highest scales of the transform. Note that the highest scale coefficient are assumed to be samples of the function itself. That is W_{φ}(J, n) = f(n), where J is the highest scale.

The first filter bank in Fig.7 splits the original function into a low pass, approximation component, which corresponds to scaling coefficients W_{φ}(J - 1, n), and a high pass, detail component, corresponds to coefficients W_{Ψ}(J – 1, n). This is graphically illustrates in Fig.8, where the scaling space V_J is split into wavelet subspace W_{J-1} and scaling sub space V_{J-1}.

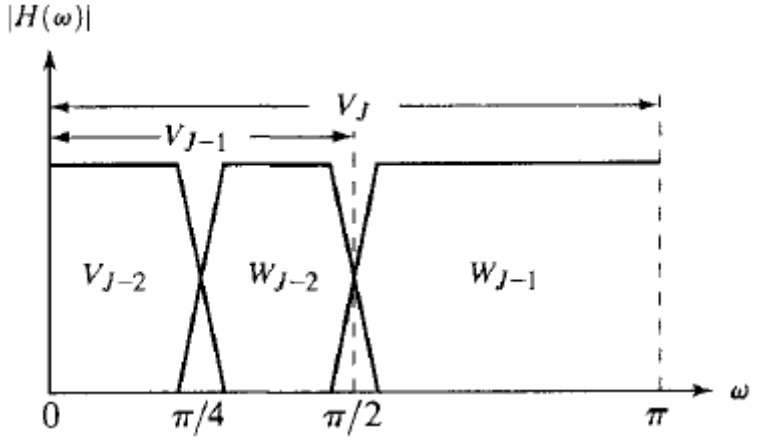


Fig. 8: Frequency splitting characteristics of Fig.7

The spectrum of the original function is split into two half-band components. The second filter bank of Fig. 8 splits the spectrum and subspace V_{J-1}, the lower half-band, into quater-band subspaces W_{J-2} and V_{J-2} with corresponding DFT coefficients W_{Ψ}(J – 1, n) and W_{φ}(J - 1, n), respectively. The two-stage filter bank in Fig. 7 is easily extended to any number of scales.

WAVELET TRANSFORM IN TWO DIMENSIONS:

The one dimensional transforms are easily extended to two-dimensional functions like images. In two dimensions, a two dimensional scaling function, $\varphi(x,y)$, and three two-dimensional wavelets, $\Psi^H(x,y)$, $\Psi^V(x,y)$, and $\Psi^D(x,y)$, are required. Each is the product of a one-dimensional scaling function φ and corresponding wavelet Ψ. Excluding products that produce one-dimensional results, like $\varphi(x)\,\Psi(x)$, the four remaining products produce the *seperable* scaling functions.

$$\varphi(x, y) = \varphi(x)\,\varphi(y)$$

And seperable, "directional sensitive" wavelets

$$\Psi^H(x, y) = \Psi(x)\,\varphi(y)$$
$$\Psi^V(x, y) = \varphi(x)\,\Psi(y)$$
$$\Psi^D(x, y) = \Psi(x)\,\Psi(y)$$

These wavelets measure functional variations (intensity or gray-level variations for images) along different directions: Ψ^H measures variations along columns (for example: horizontal edges), Ψ^V responds to variations along rows (like vertical edges), and Ψ^D corresponds to variations along diagonals.

The directional sensitivity is a natural consequence of the separability imposed by above equations; it does not increase the computational complexity of the two-dimensional transform.

The extension of one dimensional DWT to two dimensions is straightforward method. Consider the scaled and translated basis functions:

$\varphi_{j,m,n}(x,y) = 2^{j/2}\,\varphi(2^j x - m, 2^j y - n)$,

$\Psi_{j,m,n}(x,y) = 2^{j/2}\,\Psi^1(2^j x - m, 2^j y - n)$, $\quad i = \{H, V, D\}$

Where index i identifies the directional wavelets. The discrete wavelet transform of function f(x, y) of size M X N is

$$W_\varphi(j_0, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{v=0}^{N-1} f(x,y)\,\varphi_{j_0,m,n}(x,y)$$
$$W_\Psi{}^i(j, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{v=0}^{N-1} f(x,y)\,\Psi^i_{j,m,n}(x,y)$$

f(x, y) is obtained via the inverse discrete wavelet transform.

$$f(x, y) = \frac{1}{\sqrt{MN}} \sum_m \sum_n W_\varphi(j_0, m, n)\,\varphi_{j_0,m,n}(x,y) +$$
$$\frac{1}{\sqrt{MN}} \sum_{i=H,V} \sum_{j=j_0}^{\infty} \sum_m \sum_n W_\Psi{}^i(j, m, n)\,\Psi^i_{j,m,n}(x,y)$$

like the one-dimensional discrete wavelet transform, the two dimensional DWT can be implemented using digital filter and down samplers. With separable two dimensional scaling and wavelet functions, the one-dimensional FWT of the rows of f(x, y), followed by the one-dimensional FWT of the resulting columns.

Wavelet Coding:

The wavelet coding is based on the idea that the coefficients of a transform that decorrelates the pixels of an image can be coded more efficiently than the original pixels themselves. If the transform's basis functions—in this case wavelets—pack most of the important visual information into a small number of coefficients, the remaining coefficients can be quantized coarsely or truncated to zero with little image distortion.

Figure 11 shows a typical wavelet coding system. To encode a 2^J X 2^J image, an analyzing wavelet, Ψ, and minimum decomposition level, J - P, are selected and used to compute the image's discrete wavelet transform. If the wavelet has a complimentary scaling function φ, the fast wavelet transform can be used. In either case, the computed transform converts a large portion of the original image to horizontal, vertical, and diagonal decomposition coefficients with zero mean and Laplacian-like distributions.

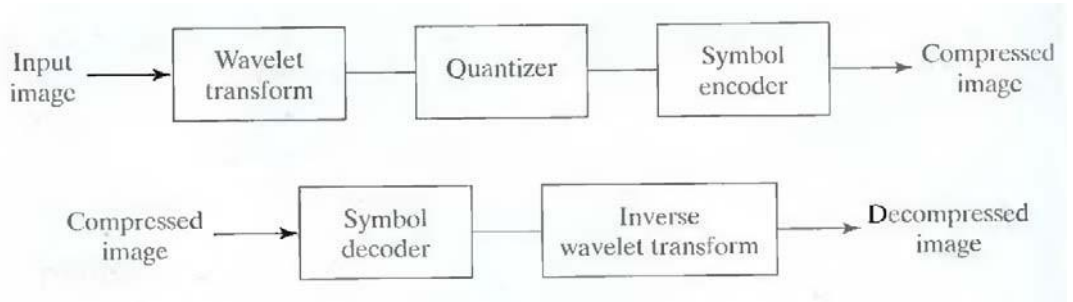


Fig. A wavelet coding system: (a) encoder; (b) decoder.

Since many of the computed coefficients carry little visual information, they can be quantized and coded to minimize Interco efficient and coding redundancy. Moreover, the quantization can be adapted to exploit any positional correlation across the P decomposition levels. One or more of the lossless coding methods, including run-length, Huffman, arithmetic, and bit-plane coding, can be incorporated into the final symbol coding step. Decoding is accomplished by inverting the encoding operations—with the exception of quantization, which cannot be reversed exactly.

The principal difference between the wavelet-based system and the transform coding system is the omission of the transform coder's subimage processing stages.

Because wavelet transforms are both computationally efficient and inherently local (i.e., their basis functions are limited in duration), subdivision of the original image is unnecessary.

UNIT- IV IMAGE SEGMENTAION & MORPHOLOGICAL IMAGE PROCESSING

FUNDAMENTALS

POINT AND LINE DETECTION

The derivative operator's role in segmentation.

Gradient operators:

First-order derivatives of a digital image are based on various approximations of the 2-D gradient. The gradient of an image f (x, y) at location (x, y) is defined as the vector

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}.$$

It is well known from vector analysis that the gradient vector points in the direction of maximum rate of change of f at coordinates (x, y). An important quantity in edge detection is the magnitude of this vector, denoted by Af, where

$$\nabla f = \text{mag}(\nabla \mathbf{f}) = \left[G_x^2 + G_y^2\right]^{1/2}.$$

This quantity gives the maximum rate of increase of f (x, y) per unit distance in the direction of Af. It is a common (although not strictly correct) practice to refer to Af also as the gradient. The direction of the gradient vector also is an important quantity. Let α (x, y) represent the direction angle of the vector Af at (x, y). Then, from vector analysis,

$$\alpha(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

where the angle is measured with respect to the x-axis. The direction of an edge at (x, y) is perpendicular to the direction of the gradient vector at that point. Computation of the gradient of an image is based on obtaining the partial derivatives &f/&x and &f/&y at every pixel location. Let the 3x3 area shown in Fig. 1.1 (a) represent the gray levels in a neighborhood of an image. One of the simplest ways to implement a first-order partial derivative at point z_5 is to use the following Roberts cross-gradient operators:

$$G_x = (z_9 - z_5)$$

and

$$G_y = (z_8 - z_6).$$

These derivatives can be implemented for an entire image by using the masks shown in Fig. 1.1(b). Masks of size 2 X 2 are awkward to implement because they do not have a clear center. An approach using masks of size 3 X 3 is given by

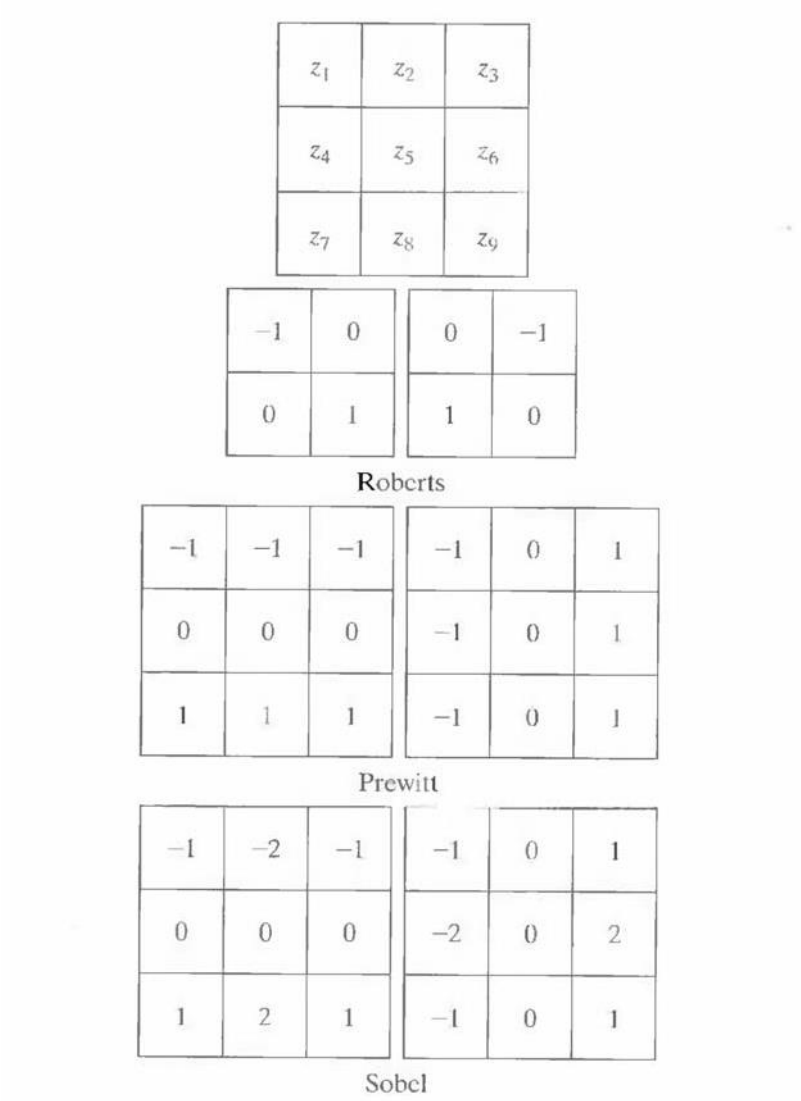


Fig.1.1 A 3 X 3 region of an image (the z's are gray-level values) and various masks used to compute the gradient at point labeled z_5.

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

and

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

A weight value of 2 is used to achieve some smoothing by giving more importance to the center point. Figures 1.1(f) and (g), called the Sobel operators, and are used to implement these two equations. The Prewitt and Sobel operators are among the most used in practice for computing digital gradients. The Prewitt masks are simpler to implement than the Sobel masks, but the latter have slightly superior noise-suppression characteristics, an important issue when dealing with derivatives. Note that the coefficients in all the masks shown in Fig. 1.1 sum to 0, indicating that they give a response of 0 in areas of constant gray level, as expected of a derivative operator.

The masks just discussed are used to obtain the gradient components G_x and G_y. Computation of the gradient requires that these two components be combined. However, this implementation is not always desirable because of the computational burden required by squares and square roots. An approach used frequently is to approximate the gradient by absolute values:

$$\nabla f \approx |G_x| + |G_y|.$$

This equation is much more attractive computationally, and it still preserves relative changes in gray levels. However, this is not an issue when masks such as the Prewitt and Sobel masks are used to compute Gx and Gy.

It is possible to modify the 3 X 3 masks in Fig. 1.1 so that they have their strongest responses along the diagonal directions. The two additional Prewitt and Sobel masks for detecting discontinuities in the diagonal directions are shown in Fig. 1.2.

0	1	1	−1	−1	0
−1	0	1	−1	0	1
−1	−1	0	0	1	1

Prewitt

0	1	2	−2	−1	0
−1	0	1	−1	0	1
2	1	0	0	1	2

Sobel

Fig.1.2 Prewitt and Sobel masks for detecting diagonal edges

The Laplacian:

The Laplacian of a 2-D function f(x, y) is a second-order derivative defined as

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

For a 3 X 3 region, one of the two forms encountered most frequently in practice is

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_6 + z_8)$$

0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

Fig.1.3 Laplacian masks used to implement Eqns. above.

where the z's are defined in Fig. 1.1(a). A digital approximation including the diagonal neighbors is given by

$$\nabla^2 f = 8z_5 - (z_1 + z_2 + z_3 + z_4 + z_6 + z_7 + z_8 + z_9).$$

Masks for implementing these two equations are shown in Fig. 1.3. We note from these masks that the implementations of Eqns. are isotropic for rotation increments of 90° and 45°, respectively.

Edge detection.

Intuitively, an edge is a set of connected pixels that lie on the boundary between two regions. Fundamentally, an edge is a "local" concept whereas a region boundary, owing to the way it is defined, is a more global idea. A reasonable definition of "edge" requires the ability to measure gray-level transitions in a meaningful way. We start by modeling an edge intuitively. This will lead us to formalism in which "meaningful" transitions in gray levels can be measured. Intuitively, an ideal edge has the properties of the model shown in Fig. 2.1(a). An ideal edge according to this model is a set of connected pixels (in the vertical direction here), each of which is located at an orthogonal step transition in gray level (as shown by the horizontal profile in the figure).

In practice, optics, sampling, and other image acquisition imperfections yield edges that are blurred, with the degree of blurring being determined by factors such as the quality of the image acquisition system, the sampling rate, and illumination conditions under which the image is acquired. As a result, edges are more closely modeled as having a "ramp like" profile, such as the one shown in Fig.2.1 (b).

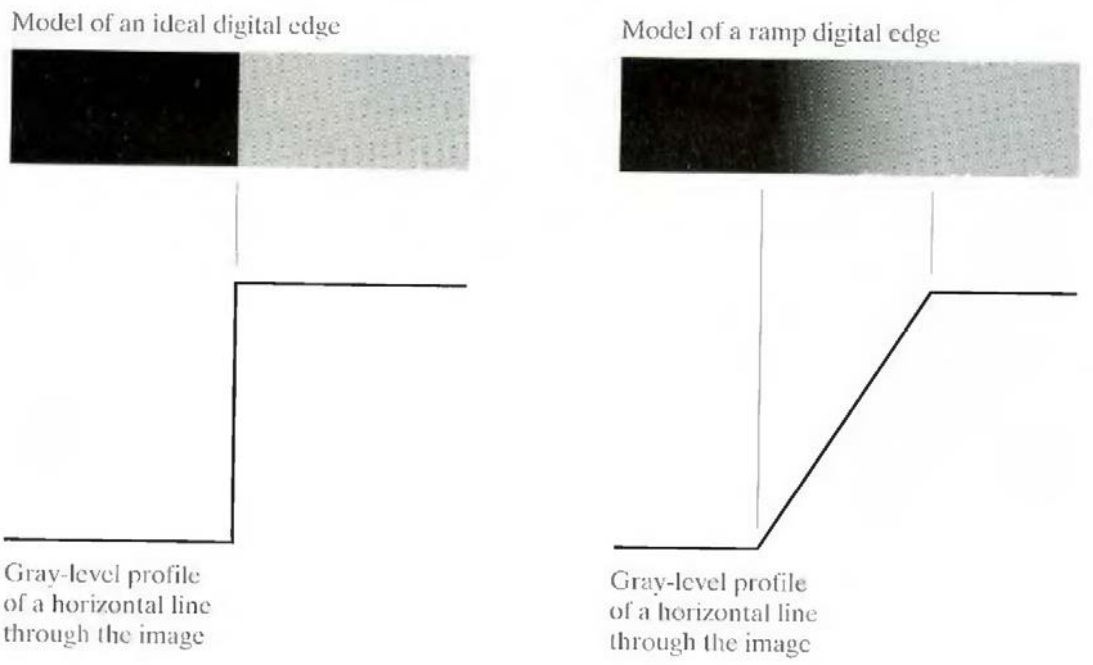


Fig.2.1 (a) Model of an ideal digital edge (b) Model of a ramp edge. The slope of the ramp is proportional to the degree of blurring in the edge.

The slope of the ramp is inversely proportional to the degree of blurring in the edge. In this model, we no longer have a thin (one pixel thick) path. Instead, an edge point now is any point contained in the ramp, and an edge would then be a set of such points that are connected. The "thickness" of the edge is determined by the length of the ramp, as it transitions from an initial to a final gray level. This length is determined by the slope, which, in turn, is determined by the degree of blurring. This makes sense: Blurred edges lend to be thick and sharp edges tend to be thin. Figure 2.2(a) shows the image from which the close-up in Fig. 2.1(b) was extracted. Figure 2.2(b) shows a horizontal gray-level profile of the edge between the two regions. This figure also shows the first and second derivatives of the gray-level profile. The first derivative is positive at the points of transition into and out of the ramp as we move from left to right along the profile; it is constant for points in the ramp; and is zero in areas of constant gray level. The second derivative is positive at the transition associated with the dark side of the edge, negative at the transition associated with the light side of the edge, and zero along the ramp and in areas of constant gray level. The signs of the derivatives in Fig. 2.2(b) would be reversed for an edge that transitions from light to dark.

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 (i.e. to determine if a point is on a ramp). 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. We note two additional properties of the second derivative around an edge: A) It produces two values for every edge in an image (an undesirable feature); and B) an imaginary straight line joining the extreme positive and negative values of the second derivative would cross zero near the midpoint of the edge. This zero-crossing property of the second derivative is quite useful for locating the centers of thick edges.

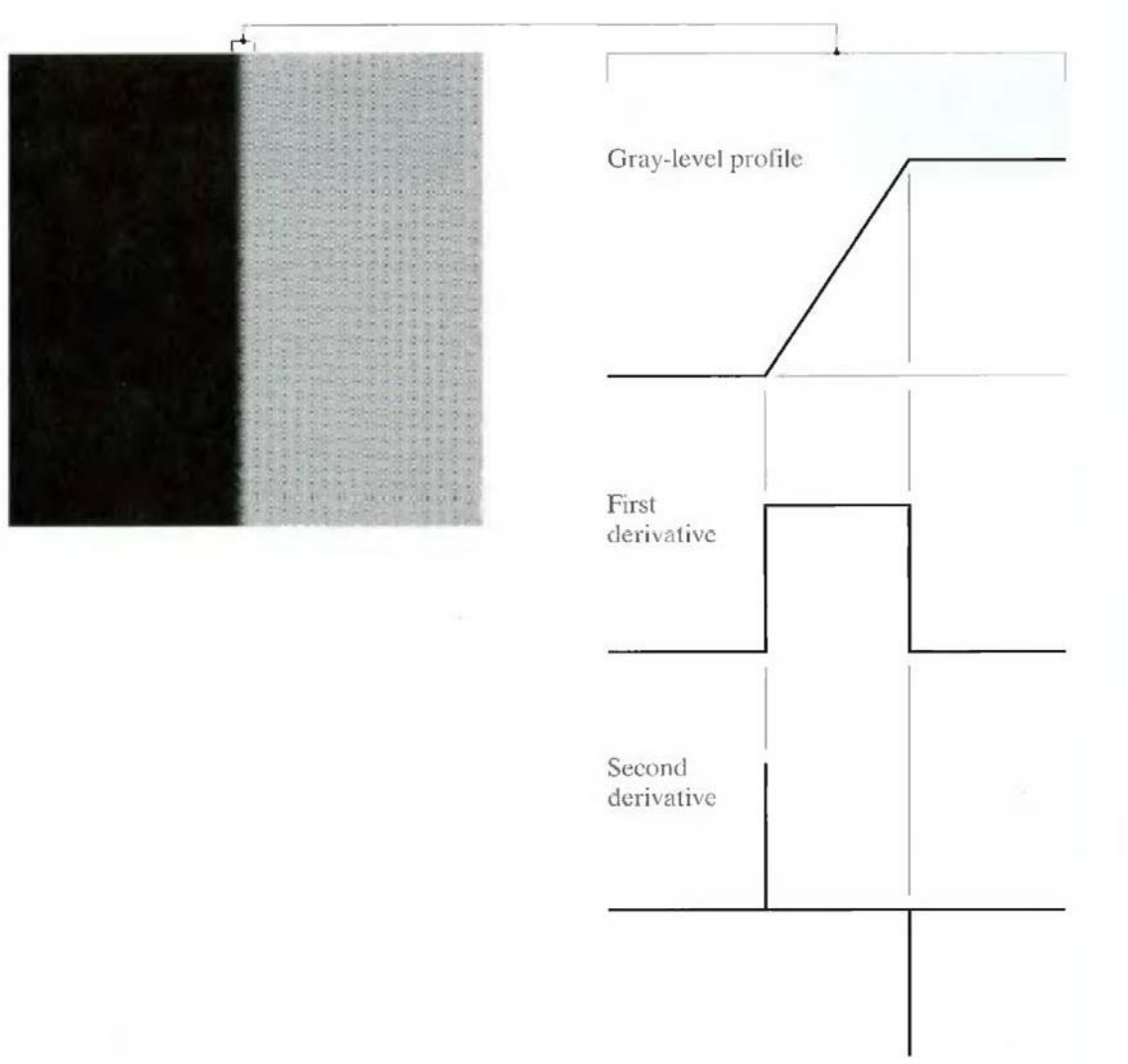


Fig.2.2 (a) Two regions separated by a vertical edge (b) Detail near the edge, showing a gray-level profile, and the first and second derivatives of the profile.

Edge linking procedures.

The different methods for edge linking are as follows

(i) Local processing

(ii) Global processing via the Hough Transform

(iii) Global processing via graph-theoretic techniques.

(i) Local Processing:

One of the simplest approaches for linking edge points is to analyze the characteristics of pixels in a small neighborhood (say, 3 X 3 or 5 X 5) about every point (x, y) in an image that has been labeled an edge point. All points that are similar according to a set of predefined criteria are linked, forming an edge of pixels that share those criteria.

The two principal properties used for establishing similarity of edge pixels in this kind of analysis are (1) the strength of the response of the gradient operator used to produce the edge pixel; and (2) the direction of the gradient vector. The first property is given by the value of Af.

Thus an edge pixel with coordinates (x_o, y_o) in a predefined neighborhood of (x, y), is similar in magnitude to the pixel at (x, y) if

$$|\nabla f(x, y) - \nabla f(x_0, y_0)| \leq E$$

The direction (angle) of the gradient vector is given by Eq. An edge pixel at (x_o, y_o) in the predefined neighborhood of (x, y) has an angle similar to the pixel at (x, y) if

$$|\alpha(x, y) - \alpha(x_0, y_0)| < A$$

where A is a nonnegative angle threshold. The direction of the edge at (x, y) is perpendicular to the direction of the gradient vector at that point.

A point in the predefined neighborhood of (x, y) is linked to the pixel at (x, y) if both magnitude and direction criteria are satisfied. This process is repeated at every location in the image. A record must be kept of linked points as the center of the neighborhood is moved from pixel to pixel. A simple bookkeeping procedure is to assign a different gray level to each set of linked edge pixels.

(ii) Global processing via the Hough Transform:

In this process, points are linked by determining first if they lie on a curve of specified shape. We now consider global relationships between pixels. Given n points in an image, suppose that we want to find subsets of these points that lie on straight lines. One possible solution is to first find all lines determined by every pair of points and then find all subsets of points that are close to particular lines. The problem with this procedure is that it involves finding $n(n - 1)/2 \sim n^2$ lines and then performing $(n)(n(n - l))/2 \sim n^3$ comparisons of every point to all lines. This approach is computationally prohibitive in all but the most trivial applications.

Hough [1962] proposed an alternative approach, commonly referred to as the Hough transform. Consider a point (x_i, y_i) and the general equation of a straight line in slope-intercept form, $y_i = a.x_i + b$. Infinitely many lines pass through (x_i, y_i) but they all satisfy the equation $y_i = a.x_i + b$ for varying values of a and b. However, writing this equation as $b = -a.x_i + y_i$, and considering the ab-plane (also called parameter space) yields the equation of a single line for a fixed pair (x_i, y_i). Furthermore, a second point (x_j, y_j) also has a line in parameter space associated with it, and this line intersects the line associated with (x_i, y_i) at (a', b'), where a' is the slope and b' the intercept of the line containing both (x_i, y_i) and (x_j, y_j) in the xy-plane. In fact, all points contained on this line have lines in parameter space that intersect at (a', b'). Figure 3.1 illustrates these concepts.

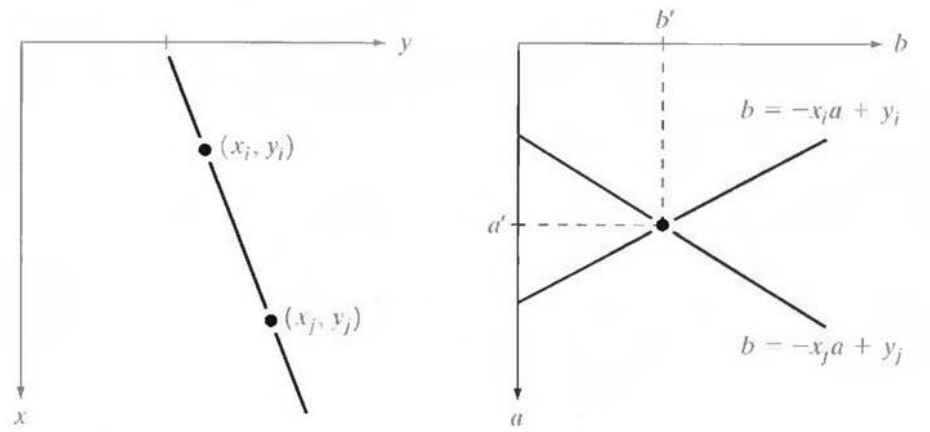


Fig.3.1 (a) xy-plane (b) Parameter space

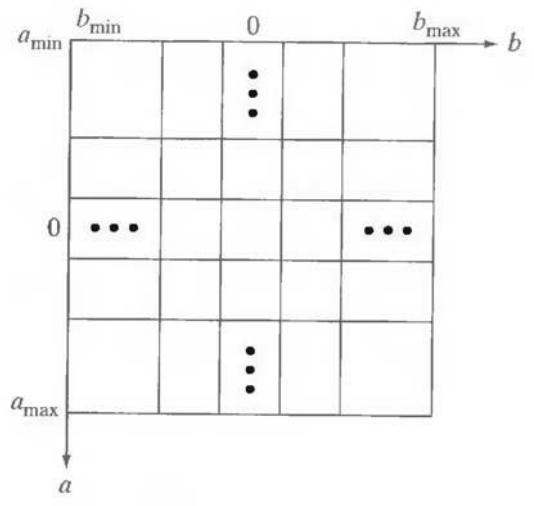


Fig.3.2 Subdivision of the parameter plane for use in the Hough transform

The computational attractiveness of the Hough transform arises from subdividing the parameter space into so-called accumulator cells, as illustrated in Fig. 3.2, where (a_{max} , a_{min}) and (b_{max} , b_{min}), are the expected ranges of slope and intercept values. The cell at coordinates (i, j), with accumulator value A(i, j), corresponds to the square associated with parameter space coordinates (a_i , b_i).

Initially, these cells are set to zero. Then, for every point (x_k, y_k) in the image plane, we let the parameter a equal each of the allowed subdivision values on the fl-axis and solve for the corresponding b using the equation $b = - x_k a + y_k$ .The resulting b's are then rounded off to the nearest allowed value in the b-axis. If a choice of a_p results in solution b_q, we let A (p, q) = A (p, q) + 1. At the end of this procedure, a value of Q in A (i, j) corresponds to Q points in the xy- plane lying on the line $y = a_i x + b_j$. The number of subdivisions in the ab-plane determines the accuracy of the co linearity of these points. Note that subdividing the a axis into K increments gives, for every point (x_k, y_k), K values of b corresponding to the K possible values of a. With n image points, this method involves nK computations. Thus the procedure just discussed is linear in n, and the product nK does not approach the number of computations discussed at the beginning unless K approaches or exceeds n.

A problem with using the equation y = ax + b to represent a line is that the slope approaches infinity as the line approaches the vertical. One way around this difficulty is to use the normal representation of a line:

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

Figure 3.3(a) illustrates the geometrical interpretation of the parameters used. The use of this representation in constructing a table of accumulators is identical to the method discussed for the slope-intercept representation. Instead of straight lines, however, the loci are sinusoidal curves in the $\rho\theta$ -plane. As before, Q collinear points lying on a line $x \cos\theta_j + y \sin\theta_j = \rho$, yield Q sinusoidal curves that intersect at (p_i, θ_j) in the parameter space. Incrementing θ and solving for the corresponding p gives Q entries in accumulator A (i, j) associated with the cell determined by (p_i, θ_j). Figure 3.3 (b) illustrates the subdivision of the parameter space.

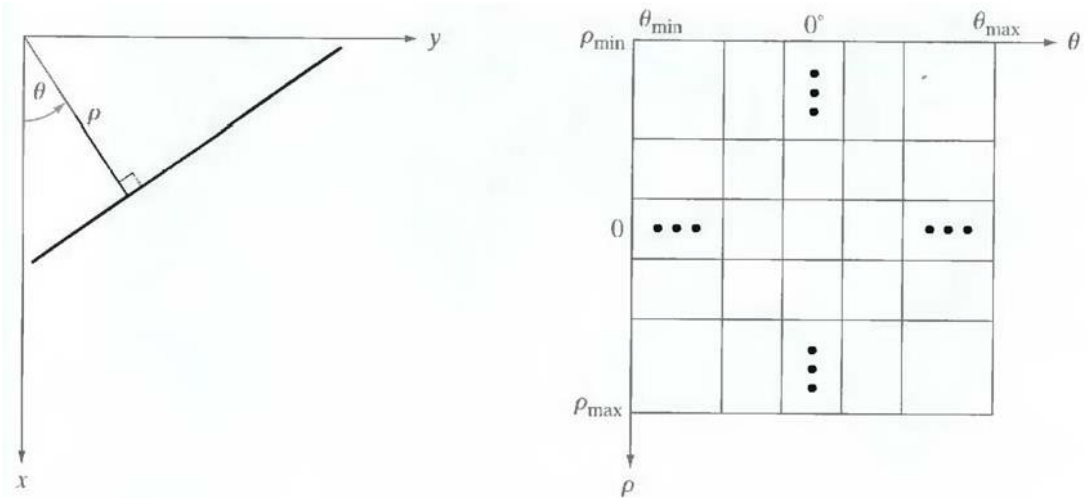


Fig.3.3 (a) Normal representation of a line (b) Subdivision of the ρθ-plane into cells

The range of angle θ is $\pm 90°$, measured with respect to the x-axis. Thus with reference to Fig. 3.3 (a), a horizontal line has $\theta = 0°$, with ρ being equal to the positive x-intercept. Similarly, a vertical line has $\theta = 90°$, with p being equal to the positive y-intercept, or $\theta = -90°$, with ρ being equal to the negative y-intercept.

(iii) Global processing via graph-theoretic techniques

In this process we have a global approach for edge detection and linking based on representing edge segments in the form of a graph and searching the graph for low-cost paths that correspond to significant edges. This representation provides a rugged approach that performs well in the presence of noise.

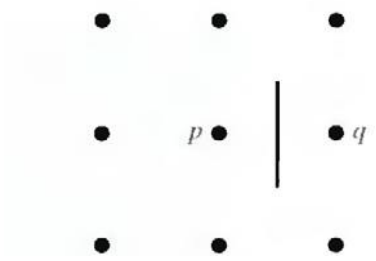


Fig.3.4 Edge clement between pixels p and q

We begin the development with some basic definitions. A graph G = (N,U) is a finite, nonempty set of nodes N, together with a set U of unordered pairs of distinct elements of N. Each pair (n_i, n_j) of U is called an arc. A graph in which the arcs are directed is called a directed graph. If an arc is directed from node n_i to node n_j, then n_j is said to be a successor of the parent node n_i. The process of identifying the successors of a node is called expansion of the node. In each graph we define levels, such that level 0 consists of a single node, called the start or root node, and the nodes in the last level are called goal nodes. A cost c (n_i, n_j) can be associated with every arc (n_i, n_j). A sequence of nodes n_1, n_2... n_k, with each node n_i being a successor of node n_{i-1} is called a path from n_1 to n_k. The cost of the entire path is

$$c = \sum_{i=2}^{k} c(n_{i-1}, n_i).$$

The following discussion is simplified if we define an edge element as the boundary between two pixels p and q, such that p and q are 4-neighbors, as Fig.3.4 illustrates. Edge elements are identified by the xy-coordinates of points p and q. In other words, the edge element in Fig. 3.4 is defined by the pairs (x_p, y_p) (x_q, y_q). Consistent with the definition an edge is a sequence of connected edge elements.

We can illustrate how the concepts just discussed apply to edge detection using the 3 X 3 image shown in Fig. 3.5 (a). The outer numbers are pixel

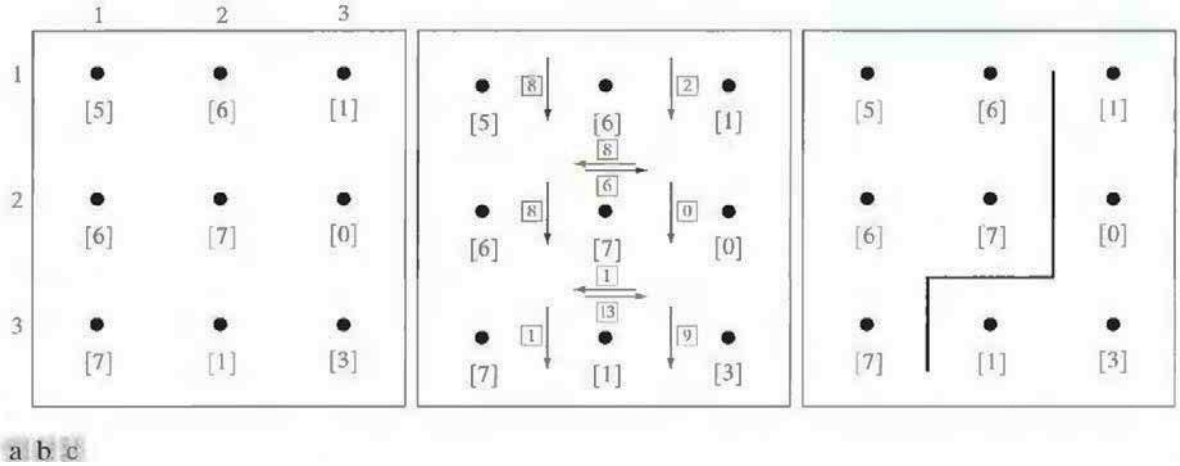


Fig.3.5 (a) A 3 X 3 image region, (b) Edge segments and their costs, (c) Edge corresponding to the lowest-cost path in the graph shown in Fig. 3.6

coordinates and the numbers in brackets represent gray-level values. Each edge element, defined by pixels p and q, has an associated cost, defined as

$$c(p, q) = H - [f(p) - f(q)]$$

where H is the highest gray-level value in the image (7 in this case), and f(p) and f(q) are the gray-level values of p and q, respectively. By convention, the point p is on the right-hand side of the direction of travel along edge elements. For example, the edge segment (1, 2) (2, 2) is between points (1, 2) and (2, 2) in Fig. 3.5 (b). If the direction of travel is to the right, then p is the point with coordinates (2, 2) and q is point with coordinates (1, 2); therefore, c (p, q) = 7 - [7 - 6] = 6. This cost is shown in the box below the edge segment. If, on the other hand, we are traveling to the left between the same two points, then p is point (1, 2) and q is (2, 2). In this case the cost is 8, as shown above the edge segment in Fig. 3.5(b). To simplify the discussion, we assume that edges start in the top row and terminate in the last row, so that the first element of an edge can be only between points (1, 1), (1, 2) or (1, 2), (1, 3). Similarly, the last edge element has to be between points (3, 1), (3, 2) or (3, 2), (3, 3). Keep in mind that p and q are 4-neighbors, as noted earlier. Figure 3.6 shows the graph for this problem. Each node (rectangle) in the graph corresponds to an edge element from Fig. 3.5. An arc exists between two nodes if the two corresponding edge elements taken in succession can be part of an edge.

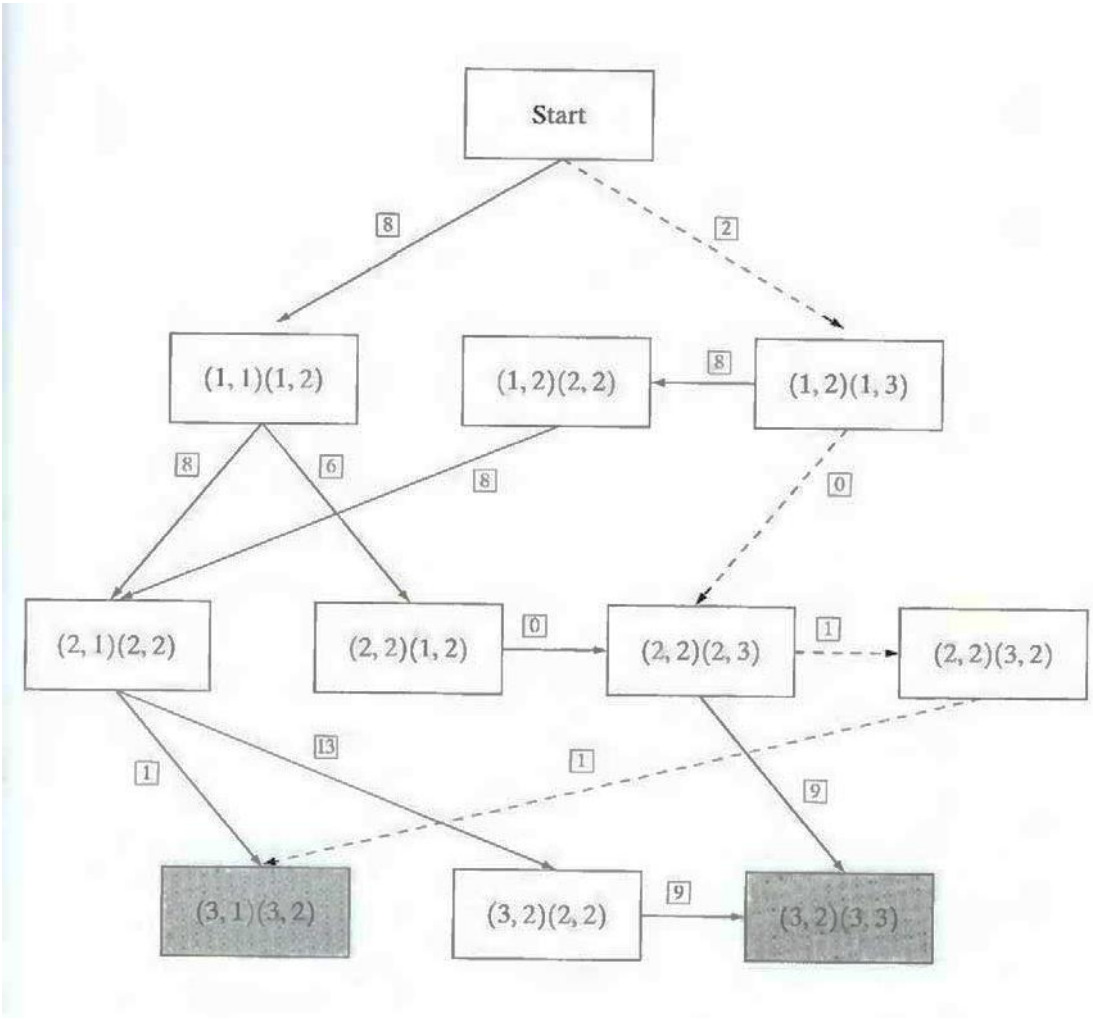


Fig. 3.6 Graph for the image in Fig.3.5 (a). The lowest-cost path is shown dashed.

As in Fig. 3.5 (b), the cost of each edge segment, is shown in a box on the side of the arc leading into the corresponding node. Goal nodes are shown shaded. The minimum cost path is shown dashed, and the edge corresponding to this path is shown in Fig. 3.5 (c).

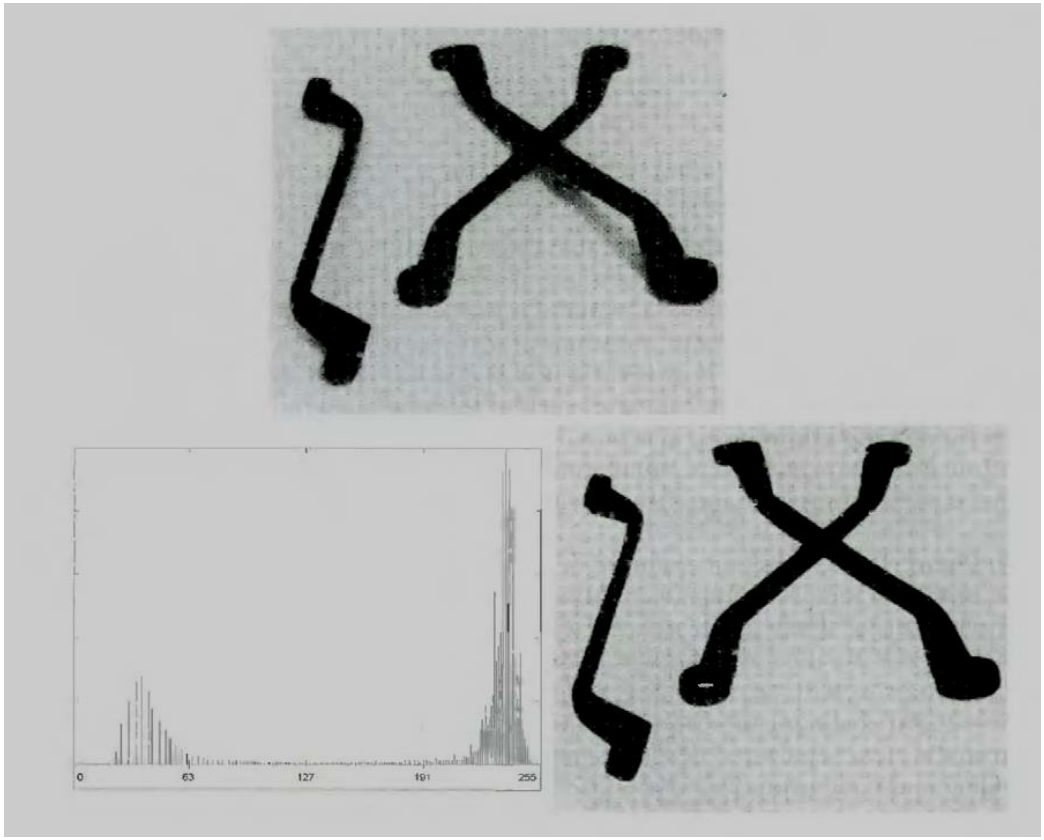
Thresholding & Global Thresholding

Thresholding:

Because of its intuitive properties and simplicity of implementation, image thresholding enjoys a central position in applications of image segmentation.

Global Thresholding:

The simplest of all thresholding techniques is to partition the image histogram by using a single global threshold, T. Segmentation is then accomplished by scanning the image pixel by pixel and labeling each pixel as object or back-ground, depending on whether the gray level of that pixel is greater or less than the value of T. As indicated earlier, the success of this method depends entirely on how well the histogram can be partitioned.



a
b c **Fig.4.1 FIGURE 10.28 (a) Original image, (b) Image histogram, (c) Result of global thresholding with T midway between the maximum and minimum gray levels.**

Figure 4.1(a) shows a simple image, and Fig. 4.1(b) shows its histogram. Figure 4.1(c) shows the result of segmenting Fig. 4.1(a) by using a threshold T midway between the maximum and minimum gray levels. This threshold achieved a "clean" segmentation by eliminating the shadows and leaving only the objects themselves. The objects of interest in this case are darker than the background, so any pixel with a gray level $\leq$ T was labeled black (0), and any pixel with a gray level $\geq$ T was labeled white (255).The key objective is merely to generate a binary image, so the black-white relationship could be reversed. The type of global thresholding just described can be expected to be successful in highly controlled environments. One of the areas in which this often is possible is in industrial inspection applications, where control of the illumination usually is feasible.

The threshold in the preceding example was specified by using a heuristic approach, based on visual inspection of the histogram. The following algorithm can be used to obtain T automatically:

1. Select an initial estimate for T.

2. Segment the image using T. This will produce two groups of pixels: G_1 consisting of all pixels with gray level values >T and G_2 consisting of pixels with values < T.

3. Compute the average gray level values μ_1 and μ_2 for the pixels in regions G_1 and G_2.

4. Compute a new threshold value:

$$T = \frac{1}{2}(\mu_1 + \mu_2).$$

5. Repeat steps 2 through 4 until the difference in T in successive iterations is smaller than a predefined parameter T_o.

When there is reason to believe that the background and object occupy comparable areas in the image, a good initial value for T is the average gray level of the image. When objects are small compared to the area occupied by the background (or vice versa), then one group of pixels will dominate the histogram and the average gray level is not as good an initial choice. A more appropriate initial value for T in cases such as this is a value midway between the maximum and minimum gray levels. The parameter T_o is used to stop the algorithm after changes become small in terms of this parameter. This is used when speed of iteration is an important issue.

Basic adaptive thresholding process used in image segmentation.

Basic Adaptive Thresholding:

Imaging factors such as uneven illumination can transform a perfectly segmentable histogram into a histogram that cannot be partitioned effectively by a single global threshold. An approach for handling such a situation is to divide the original image into subimages and then utilize a different threshold to segment each subimage. The key issues in this approach are how to subdivide the image and how to estimate the threshold for each resulting subimage. Since the threshold used for each pixel depends on the location of the pixel in terms of the subimages, this type of thresholding is adaptive.

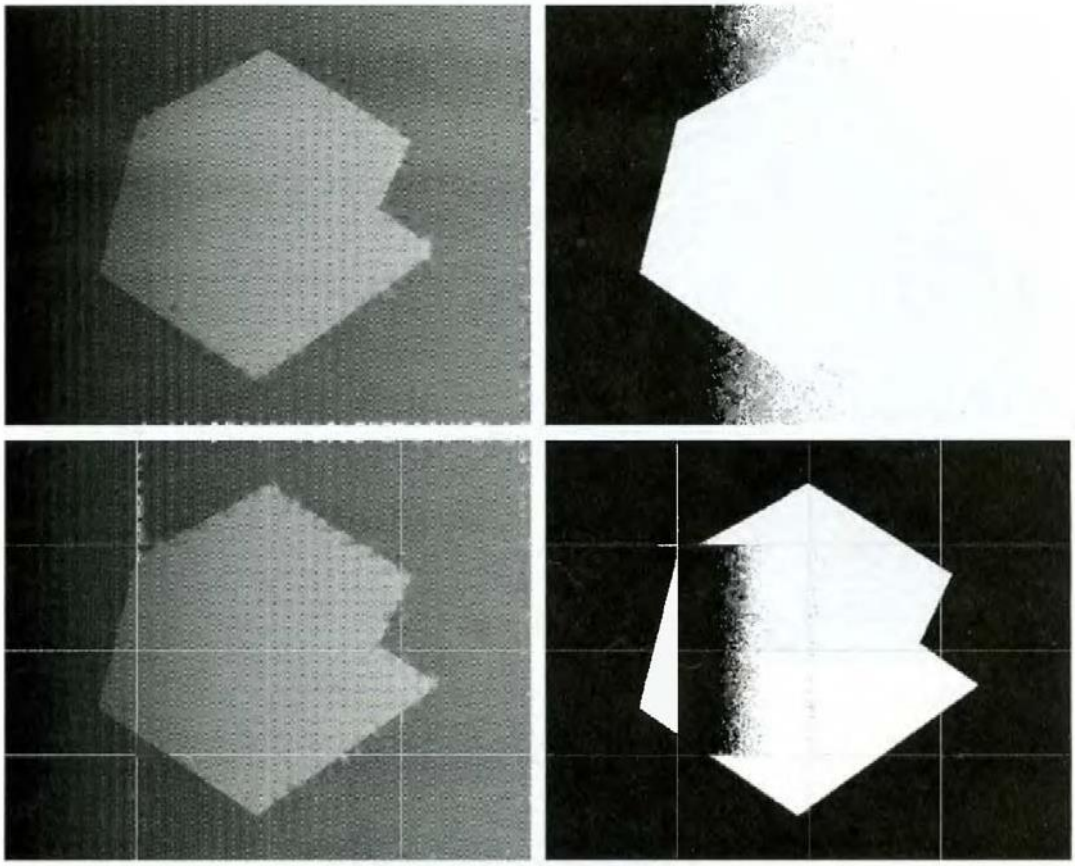


Fig.5 (a) Original image, (b) Result of global thresholding. (c) Image subdivided into individual subimages (d) Result of adaptive thresholding.

We illustrate adaptive thresholding with a example. Figure 5(a) shows the image, which we concluded could not be thresholded effectively with a single global threshold. In fact, Fig. 5(b) shows the result of thresholding the image with a global threshold manually placed in the valley of its histogram. One approach to reduce the effect of nonuniform illumination is to subdivide the image into smaller subimages, such that the illumination of each subimage is approximately uniform. Figure 5(c) shows such a partition, obtained by subdividing the image into four equal parts, and then subdividing each part by four again. All the subimages that did not contain a boundary between object and back-ground had variances of less than 75. All subimages containing boundaries had variances in excess of 100. Each subimage with variance greater than 100 was segmented with a threshold computed for that subimage using the algorithm. The initial

value for T in each case was selected as the point midway between the minimum and maximum gray levels in the subimage. All subimages with variance less than 100 were treated as one composite image, which was segmented using a single threshold estimated using the same algorithm. The result of segmentation using this procedure is shown in Fig. 5(d).

With the exception of two subimages, the improvement over Fig. 5(b) is evident. The boundary between object and background in each of the improperly segmented subimages was small and dark, and the resulting histogram was almost unimodal.

Threshold selection based on boundary characteristics.

Use of Boundary Characteristics for Histogram Improvement and Local Thresholding:

It is intuitively evident that the chances of selecting a "good" threshold are enhanced considerably if the histogram peaks are tall, narrow, symmetric, and separated by deep valleys. One approach for improving the shape of histograms is to consider only those pixels that lie on or near the edges between objects and the background. An immediate and obvious improvement is that histograms would be less dependent on the relative sizes of objects and the background. For instance, the histogram of an image composed of a small object on a large background area (or vice versa) would be dominated by a large peak because of the high concentration of one type of pixels.

If only the pixels on or near the edge between object and the background were used, the resulting histogram would have peaks of approximately the same height. In addition, the probability that any of those given pixels lies on an object would be approximately equal to the probability that it lies on the back-ground, thus improving the symmetry of the histogram peaks.

Finally, as indicated in the following paragraph, using pixels that satisfy some simple measures based on gradient and Laplacian operators has a tendency to deepen the valley between histogram peaks.

The principal problem with the approach just discussed is the implicit assumption that the edges between objects and background arc known. This information clearly is not available during segmentation, as finding a division between objects and background is precisely what segmentation is all about. However, an indication of whether a pixel is on an edge may be obtained by computing its gradient. In addition, use of the Laplacian can yield information regarding whether a given pixel lies on the dark or light side of an edge. The average value of the Laplacian is 0 at the transition of an edge, so in practice the valleys of histograms formed from the pixels selected by a gradient/Laplacian criterion can be expected to be sparsely populated. This property produces the highly desirable deep valleys.

The gradient ∇f at any point (x, y) in an image can be found. Similarly, the Laplacian $\nabla^2 f$ can also be found. These two quantities may be used to form a three-level image, as follows:

$$s(x, y) = \begin{cases} 0 & \text{if } \nabla f < T \\ + & \text{if } \nabla f \geq T \quad \text{and} \quad \nabla^2 f \geq 0 \\ - & \text{if } \nabla f \geq T \quad \text{and} \quad \nabla^2 f < 0 \end{cases}$$

where the symbols 0, +, and - represent any three distinct gray levels, T is a threshold, and the gradient and Laplacian are computed at every point (x, y). For a dark object on a light background, the use of the Eqn. produces an image s(x, y) in which (1) all pixels that are not on an edge (as determined by ∇f being less than T) are labeled 0; (2) all pixels on the dark side of an edge are labeled +; and (3) all pixels on the light side of an edge are labeled -. The symbols + and - in Eq. above are reversed for a light object on a dark background. Figure 6.1 shows the labeling produced by Eq. for an image of a dark, underlined stroke written on a light background.

The information obtained with this procedure can be used to generate a segmented, binary image in which l's correspond to objects of interest and 0's correspond to the background. The transition (along a horizontal or vertical scan line) from a light background to a dark object must be characterized by the occurrence of a - followed by a + in s (x, y). The interior of the object is composed of pixels that are labeled either 0 or +. Finally, the transition from the object back to the background is characterized by the occurrence of a + followed by a -. Thus a horizontal or vertical scan line containing a section of an object has the following structure:

$$(\cdots)(-,+)(0 \text{ or } +)(+,\ -)(\cdots)$$

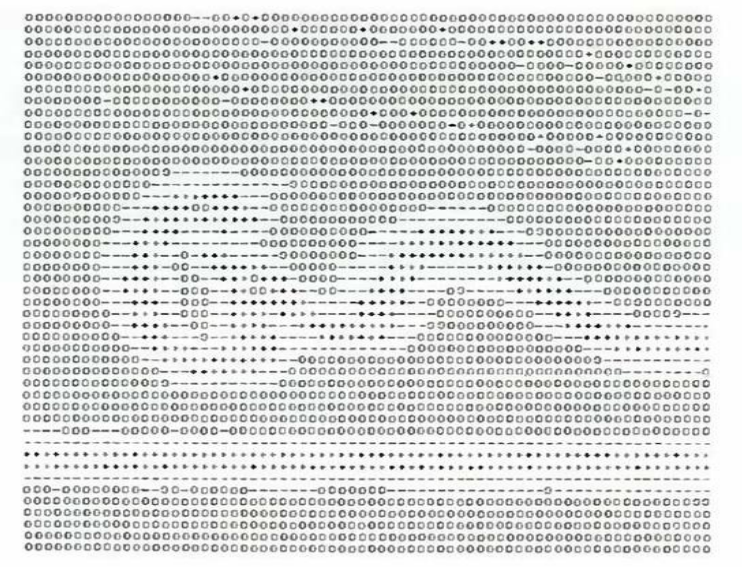


Fig.6.1 Image of a handwritten stroke coded by using Eq. discussed above

where (…) represents any combination of +, -, and 0. The innermost parentheses contain object points and are labeled 1. All other pixels along the same scan line are labeled 0, with the exception of any other sequence of (- or +) bounded by (-, +) and (+, -).

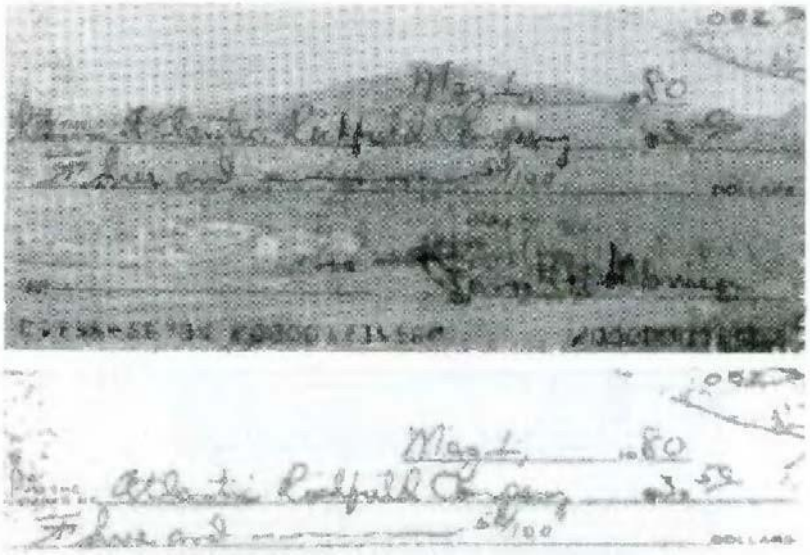


Fig.6.2 (a) Original image, (b) Image segmented by local thresholding.

Figure 6.2 (a) shows an image of an ordinary scenic bank check. Figure 6.3 shows the histogram as a function of gradient values for pixels with gradients greater than 5. Note that this histogram has two dominant modes that are symmetric, nearly of the same height, and arc separated by a distinct valley. Finally, Fig. 6.2(b) shows the segmented image obtained by with T at or near the midpoint of the valley. Note that this example is an illustration of local thresholding, because the value of T was determined from a histogram of the gradient and Laplacian, which are local properties.

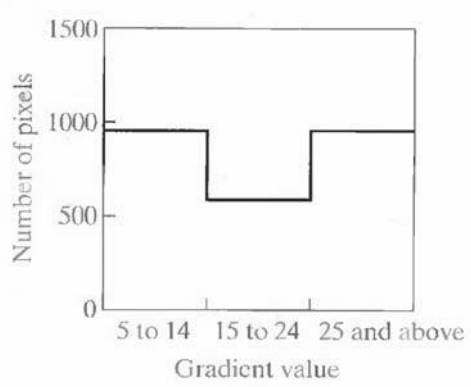


Fig.6.3 Histogram of pixels with gradients greater than 5

Region based segmentation.

Region-Based Segmentation:

The objective of segmentation is to partition an image into regions. We approached this problem by finding boundaries between regions based on discontinuities in gray levels, whereas segmentation was accomplished via thresholds based on the distribution of pixel properties, such as gray-level values or color.

Basic Formulation:

Let R represent the entire image region. We may view segmentation as a process that partitions R into n subregions, R_1, R_2..., R_n, such that

(a) $\bigcup_{i=1}^{n} R_i = R.$
(b) R_i is a connected region, $i = 1, 2, \ldots, n.$
(c) $R_i \cap R_j = \varnothing$ for all i and $j, i \neq j.$
(d) $P(R_i) =$ TRUE for $i = 1, 2, \ldots, n.$
(e) $P(R_i \cup R_j) =$ FALSE for $i \neq j.$

Here, P (R_i) is a logical predicate defined over the points in set R_i and Ø` is the null set. Condition (a) indicates that the segmentation must be complete; that is, every pixel must be in a region. Condition (b) requires that points in a region must be connected in some predefined sense. Condition (c) indicates 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 P (R_i) = TRUE if all pixels in R_i, have the same gray level. Finally, condition (c) indicates that regions R_i and R_j are different in the sense of predicate P.

Region Growing:

As its name implies, region growing is a procedure that groups pixels or subregions into larger regions based on predefined criteria. 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 properties similar to the seed (such as specific ranges of gray level or color). When a priori information is not available, the procedure is to compute at every pixel the same set of properties that ultimately will be used to assign pixels to regions during the growing process. If the result of these computations shows clusters of values, the pixels whose properties place them near the centroid of these clusters can be used as seeds.

The selection of similarity criteria depends not only on the problem under consideration, but also on the type of image data available. For example, the analysis of land-use satellite imagery depends heavily on the use of color. This problem would be significantly more difficult, or even impossible, to handle without the inherent information available in color images. When the images are monochrome, region analysis must be carried out with a set of descriptors based on gray levels and spatial properties (such as moments or texture).

Basically, growing a region should stop when no more pixels satisfy the criteria for inclusion in that region. Criteria such as gray level, texture, and color, are local in nature and do not take into account the "history" of region growth. Additional criteria that increase the power of a region- growing algorithm utilize the concept of size, likeness between a candidate pixel and the pixels grown so far (such as a comparison of the gray level of a candidate and the average gray level of the grown region), and the shape of the region being grown. The use of these types of descriptors is based on the assumption that a model of expected results is at least partially available.

Figure 7.1 (a) shows an X-ray image of a weld (the horizontal dark region) containing several cracks and porosities (the bright, white streaks running horizontally through the middle of the image). We wish to use region growing to segment the regions of the weld failures. These segmented features could be used for inspection, for inclusion in a database of historical studies, for controlling an automated welding system, and for other numerous applications.

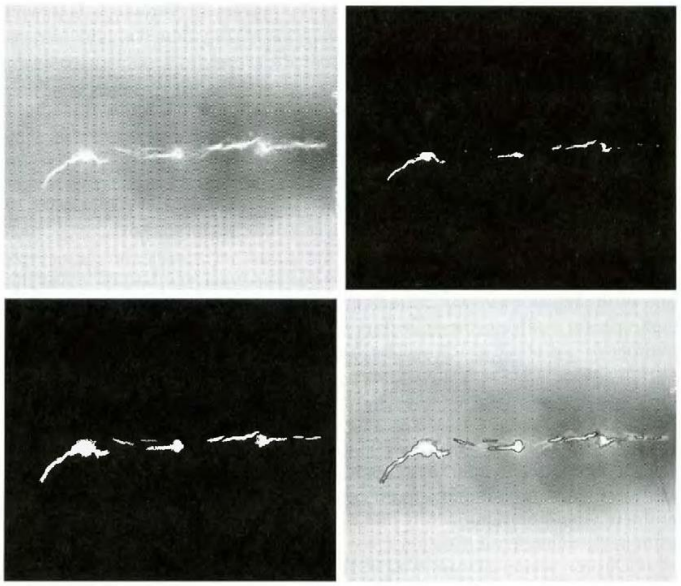


Fig.7.1 (a) Image showing defective welds, (b) Seed points, (c) Result of region growing, (d) Boundaries of segmented ; defective welds (in black).

The first order of business is to determine the initial seed points. In this application, it is known that pixels of defective welds tend to have the maximum allowable digital value B55 in this case). Based on this information, we selected as starting points all pixels having values of 255. The points thus extracted from the original image are shown in Fig. 10.40(b). Note that many of the points are clustered into seed regions.

The next step is to choose criteria for region growing. In this particular example we chose two criteria for a pixel to be annexed to a region: (1) The absolute gray-level difference between any pixel and the seed had to be less than 65. This number is based on the histogram shown in Fig. 7.2 and represents the difference between 255 and the location of the first major valley to the left, which is representative of the highest gray level value in the dark weld region. (2) To be included in one of the regions, the pixel had to be 8-connected to at least one pixel in that region.

If a pixel was found to be connected to more than one region, the regions were merged. Figure 7.1 (c) shows the regions that resulted by starting with the seeds in Fig. 7.2 (b) and utilizing the criteria defined in the previous paragraph. Superimposing the boundaries of these regions on the original image [Fig. 7.1(d)] reveals that the region-growing procedure did indeed segment the defective welds with an acceptable degree of accuracy. It is of interest to note that it was not necessary to specify any stopping rules in this case because the criteria for region growing were sufficient to isolate the features of interest.

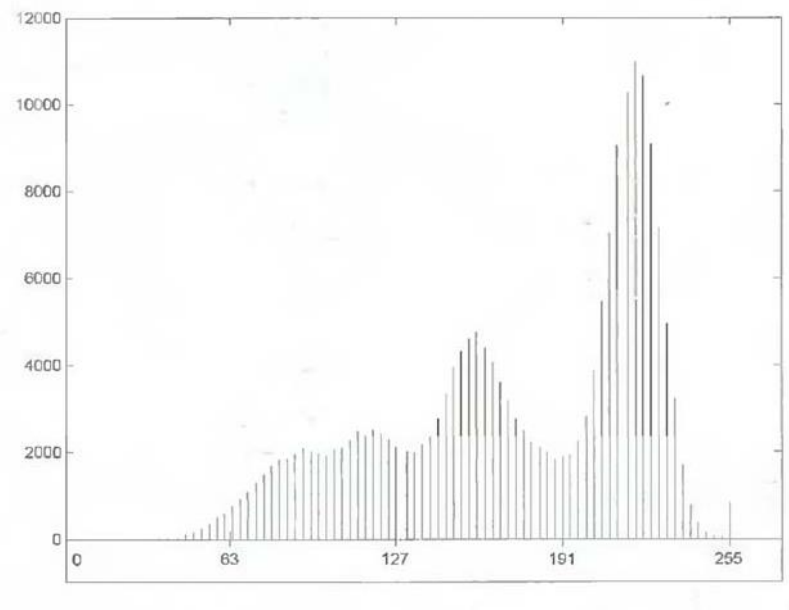


Fig.7.2 Histogram of Fig. 7.1 (a)

Region Splitting and Merging:

The procedure just discussed grows regions from a set of seed points. An alternative is to subdivide an image initially into a set of arbitrary, disjointed regions and then merge and/or split the regions in an attempt to satisfy the conditions. A split and merge algorithm that iteratively works toward satisfying these constraints is developed.

Let R represent the entire image region and select a predicate P. One approach for segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i, $P(R_i)$ = TRUE. We start with the entire region. If P(R) = FALSE, we divide the image into quadrants. If P is FALSE for any quadrant, we subdivide that quadrant into subquadrants, and so on. This particular splitting technique has a convenient representation in the form of a so-called quadtree (that is, a tree in which nodes have exactly four descendants), as illustrated in Fig. 7.3. Note that the root of the tree corresponds to the entire image and that each node corresponds to a subdivision. In this case, only R_4 was subdivided further.

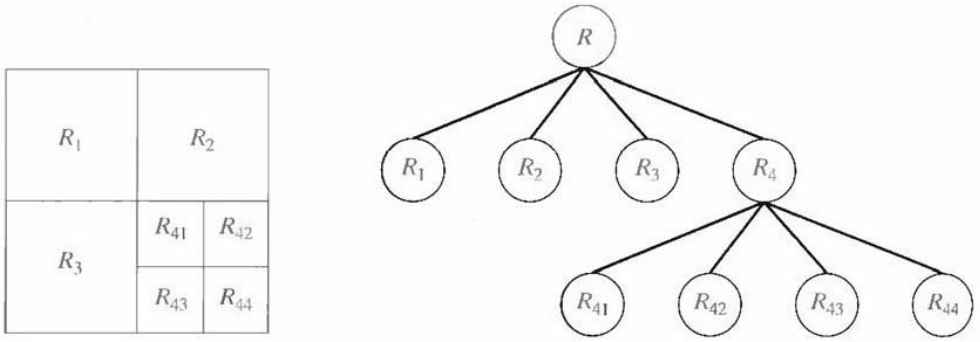


Fig. 7.3 (a) Partitioned image (b) Corresponding quadtree.

If only splitting were used, the final partition likely would contain adjacent regions with identical properties. This drawback may be remedied by allowing merging, as well as splitting. Satisfying the constraints, requires merging only adjacent regions whose combined pixels satisfy the predicate P. That is, two adjacent regions R_j and R_k are merged only if $P(R_j \cup R_k)$ = TRUE.

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

1. Split into four disjoint quadrants any region R_i, for which $P(R_i)$ = FALSE.

2. Merge any adjacent regions R_j and R_k for which $P(R_j \cup R_k)$ = TRUE.

3. Stop when no further merging or splitting is possible.

Several variations of the preceding basic theme are possible. For example, one possibility is to split the image initially into a set of blocks. Further splitting is carried out as described previously, but merging is initially limited to groups of four blocks that are descendants in the quadtree representation and that satisfy the predicate P. When no further mergings of this type are possible, the procedure is terminated by one final merging of regions satisfying step 2. At this point, the merged regions may be of different sizes. The principal advantage of this approach is that it uses the same quadtree for splitting and merging, until the final merging step.

MORPHOLOGICAL IMAGE PROCESSING:

PRELIMINARIES:

Some basic concepts from set theory that are needed as foundation for this unit.

Some Basic Concepts from Set Theory:

Let A be a set in Z^2. If a= (a_1 , a_2) is an element of A, then we write

$$a \in A$$

Similarly, if "a" is not an element of A, we write

$$a \notin A$$

The set with no elements is called the ***null or empty set*** and is denoted by the symbol Ø**.** A set is specified by the contents of two braces {.}. The elements of the sets with which we are concerned in this unit as the coordinates of pixels representing objects or other features of interest in an image. Foe example, when we write an expression of the form C= { w/w = -d, for d ϵ D} we meant that set C is the set of elements, w, such that w is formed by multiplying each of the two coordinates of all the elements of set D by -1.

If every element of a set A is also an element of another set B, the A is said to be a subset of B, denoted as

$$A \subseteq B$$

The union of two sets A and B, denoted by

$$C = A \cup B$$

Is the set of all elements belonging to either A, B, or both. Similarly, the intersection of two sets A and B are denoted by

$$D = A \cap B$$

Is the set of all elements belonging to both A and B.

Two sets A and B are said to be disjoint or mutually exclusive if they have no common elements. In this case

$$A \cap B = \emptyset$$

The Principal logic operation used in image processing are AND, OR and NOT (COMPLEMENT). Their properties are summarized in Table.1. These operations are functionally complete in the sense that they can be combined to form any other logic operation.

Table.1 : The three basic operations

P	Q	P AND Q	P OR Q	NOT (P)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

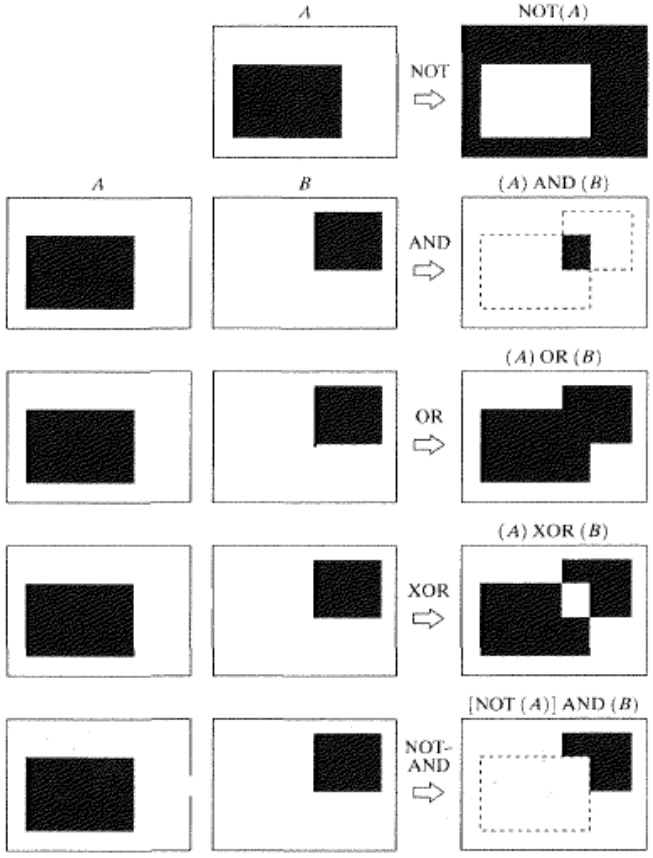


Fig.1: Some logic operations between binary images

Logic operations are performed on a pixel by pixel basis between corresponding pixels of two or more images (except NOT, which operates on the pixels of a single image). 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. Fig.1 shows various examples of logic operations involving images, where black indicates a binary 1 and white indicates 0.

DILATION:

With A and B are sets in Z^2, the dilation of A by B, denoted $A \oplus B$, is defined as

$$A \oplus B = \{z / (\hat{B})_z \cap A \neq \emptyset\}$$

This equation is based on obtaining the reflection of B about its origin and shifting this reflection by z. The dilation of A by B then is the set of all displacements, z, such that $\hat{B}$ and A overlap by at least one element. Set B is commonly referred to as the structuring element in dilation, as well as in other morphological operations.

The dilation is based on set operations, whereas convolution is based on arithmetic operations, the basic process of flipping B about its origin and then successively displacing it so that it slides over set (image) A is analogous to the convolution process.

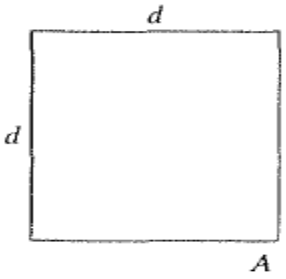


Fig.2: Set A

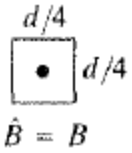


Fig.3: Square structuring element (a dot in the center)

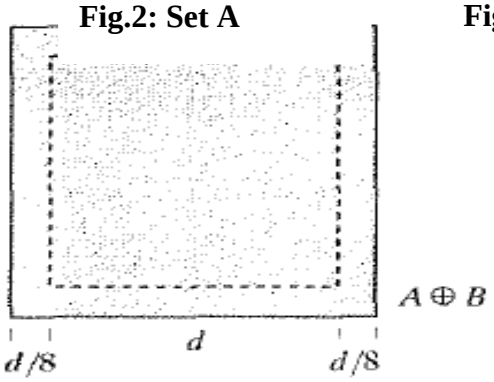


Fig. 4: Dilation A by B shown shaded

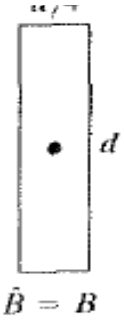


Fig. 5: Elongaed structuring element

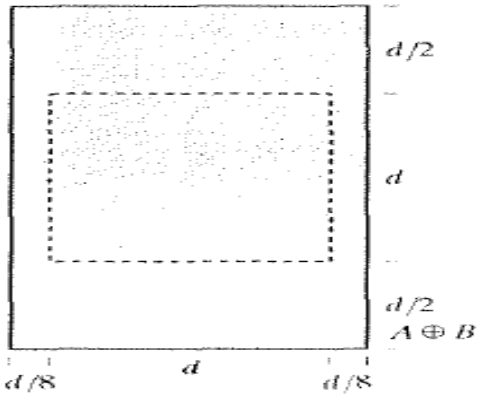


Fig. 6: Dilation of A using this element.

The Fig.2 shows a sir eflection (the dark dot denotes the origin of the element). In this case the structuring element and its reflection are equal because B is symmetric with respect to its origin. The dashed line in Fig.4 shows the original set for reference and the solid line shows the limit beyond which any further displacements of the origin of $\hat{B}$ by z would cause the intersection of $\hat{B}$ and A to be empty. Therefore, all points inside this boundary constitute the dilation of A by B. The Fig. 5 shows a structuring element designed to achieve more dilation vertically than horizontal. Fig.6 shows the dilation achieved with this element.

EROSION:

For sets A and B in Z^2 the erosion of A by B, denoted A Θ B, is defined as

$$A \ominus B = \{z \,/\, (B)_z \subseteq A\}$$

In words, this equation indicates that the erosion of A by B is the set of all points z such that B, translated by z, is contained in A. As in the case of dilation, the above equation is not the only definition of erosion. However, this equation usually is favored in practical implementations of morphology.

Fig.7: Set A

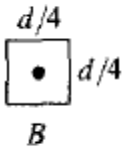


Fig.8:Square structuring element

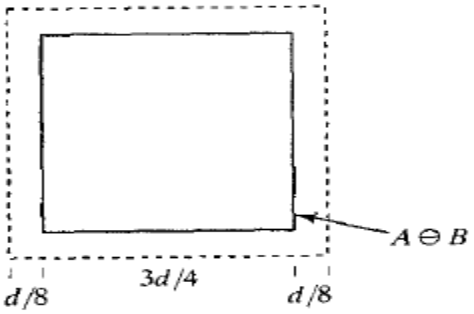


Fig. 9: Erosion of A by B,shown shaded **Fig. 10: elongated structuring element**

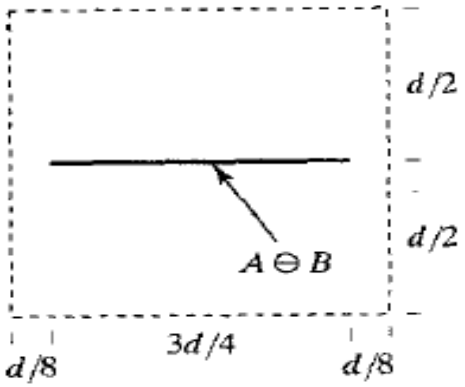


Fig. 11: Erosion of A using this element

The Fig.7-11 shows the process similar to that shown in dilation. As before, set A is shown as a dashed line for reference in Fig.9. The boundary of the shaded region shows the limit beyond which further displacement of the origin of B would cause this set to cease being completely contained in A. Thus, the locus of points within this boundary (i.e., the shaded region) constitutes the erosion of A by B, Fig. 10 shows an elongated structuring element and Fig. 11shows the erosion of A by this element. Here, the original set was eroded down the line. One of the simplest uses of erosion is for eliminating irrelevant detail (in terms of size) from a binary image).

OPENING AND CLOSING:

Opening generally smoothes the contour of an object, breaks narrow isthmuses, and eliminates thin protrusions. Closing also tends to smooth sections of contours but, as opposed to opening, it generally fuses narrow breaks and long thin gulfs, eliminates small holes, and fills gaps in the contour.

The opening of set A by structuring element B, denoted $A \circ B$, is defined as

$$A \circ B = (A \ominus B) \oplus B$$

Thus, the opening A by B is the erosion of A by B, followed by a dilation of the result of B.

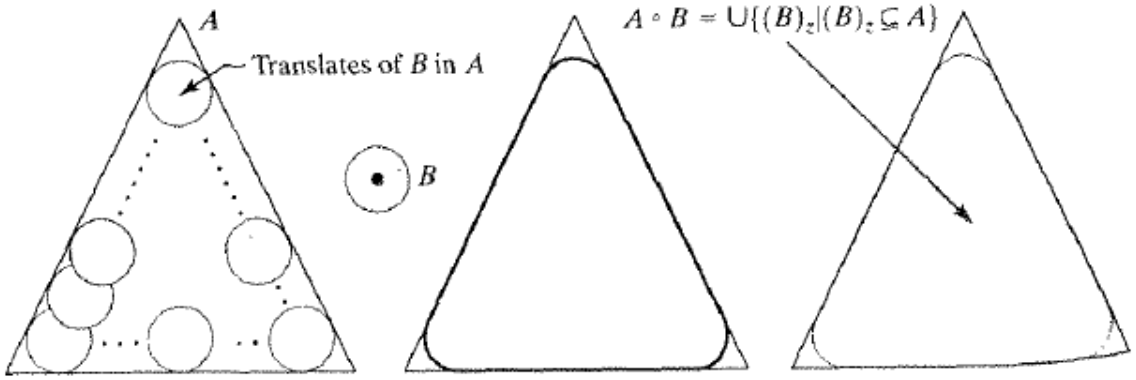
Similarly, the closing of set A by structuring element B, denoted A• B, is defined as

$$A\bullet B = (A \oplus B) \ominus B$$

Which, in words, says that the closing of A by B is simply the dilation of A by B, followed by the erosion of the result by B.

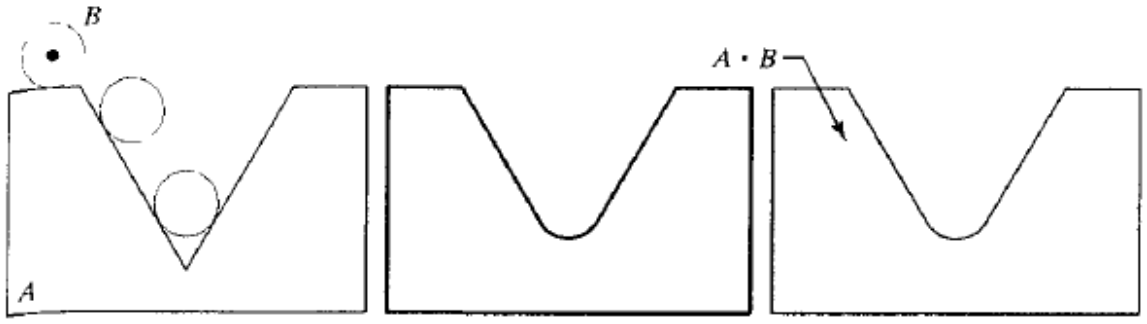
The opening operation has a simple geometric interpretation (Fig. 12). Suppose that we view the structuring element B as a (flat) "rolling ball." The boundary of $A \circ B$ is then established by the points in B that reach the farthest into the boundary of A as B is rolled around the inside of this boundary. This geometric fitting property of the opening of A by B is obtained by taking the union of all translates of B that fit into A. That is, opening can be expressed as fitting process such that

$$A \circ B = U \{(B)_z / (B)_z \subseteq A\}$$



a b c d

Fig.12: (a) Structuring element B "rolling" along the inner boundary of A (the dot indicates the origin of B). (c) The heavy line is the outer boundary of the opening (d) Complete opening (Shaded)



a b c

Fig. 13: (a) Structuring element B "rolling" on the outer boundary of set A. (b) Heavy

line is the outer boundary of the closing. (c) Complete closing.

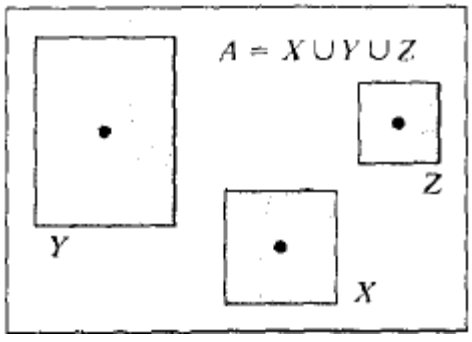
Where U {.} denotes the union of all the sets inside the braces.

Closing has a similar geometric interpretation, except that now we roll B on the outside of the boundary (Fig. 13). It will be shown shortly that opening and closing are duals of each other, so having to roll the ball on the outside is not unexpected. Geometrically, a point w is an element of A• B if and only if $(B)_z \cap A \neq \emptyset$ for any translate of $(B)_z$ that contains w. Fig.13 illustrates the basic geometrical properties of closing.

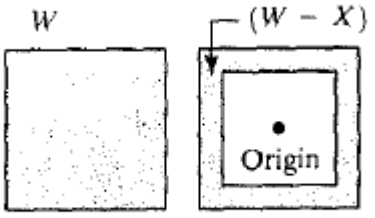
THE HIT-OR-MISS TRANSFORMATION:

The morphological hit-or-miss transform is a basic tool for shape detection. The Fig.14 shows a set A consisting of three shapes (subsets), denoted X, Y, Z. The shading in Figs.14 (a) through (c) indicates the original sets, where as the shading in Figs.14(d) and (e) indicates the result of morphological operations. The objective is to find the location of one of the shapes, say, X.

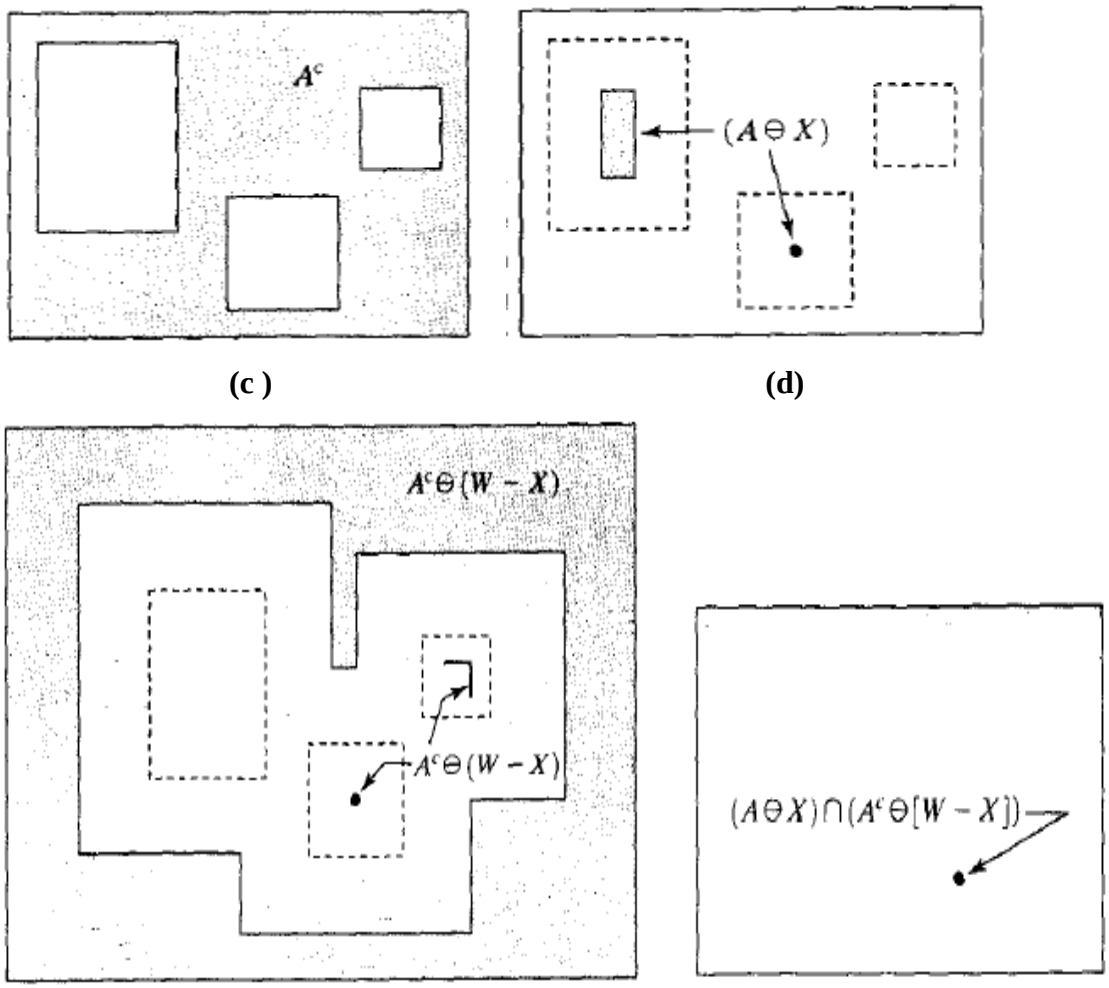
Let the origin of each shape be located at its center of gravity. Let X be enclosed by a small window,W. The Local background of X with respect to W is defined as the set difference (W– X), as shown in Fig.14(b). Fig. 14(c) shows the complement of A, which is needed later. Fig.14(d) shows the erosion of A by X . Fig.14 (e) shows the erosion of the complement of A by the local background set (W- X). the outer shaded region in the Fig.14(e) is part of the erosion. The Figs.14(d) and (e) that the set of locations for which X exactly fits inside A is the intersection of the erosion of A by X and the erosion of A^c by (W- X) as shown in Fig. 14(f). This intersection is precisely the location sought. In other words, if B denotes the set composed of X and its background, the match (o set of matches) of B in A.



(a)



(b)



(c) **(d)**

(e) **(f)**

Fig. 14: (a) Set A (b) A window ,W, and local background of X, with respect to W, (W - X). (c)Complement of A. (d) Erosion of A by X. (e) Erosion of A^c by (W –X). (f) Intersection of (d) and (e), showing the location of the origin of X, as desired.

SOME BASIC MORPHOLOGICAL ALGORITHMS:

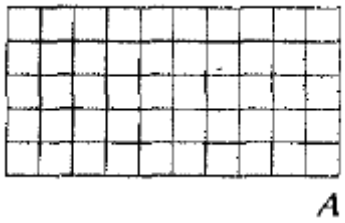
1. **Boundary Extraction:**

The boundary of a set A, denoted by $\beta(A)$, can be obtained by first eroding A by B and then performing the set difference between A and its erosion.

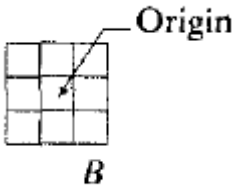
$\beta(A) = A - (A \ominus B)$

Where B is a suitable element.

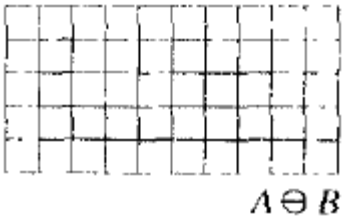
Fig. 15.illustrates the mechanics of boundary extraction. It shows a simple binary object, a structuring element B, and the result of using above equation. Although the structuring element shown in Fig. 15(b) is among the most frequently used, it is by no means unique.



(a)



(b)



(c)

(d)

Fig. 15: (a) Set A. (b) Structuring element B. (c) A eroded by B. (d) Boundary, given by the set difference between A and its erosion.

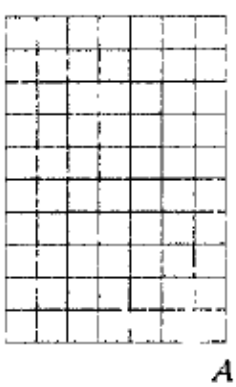
2.Region Filtering:

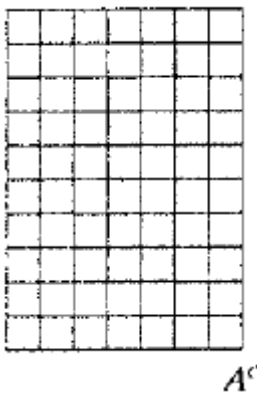
The region filling is a simple alogorithm whichis based on set dilations,complementation, and intersections. In Fig. 16 A denotes a set containing a subset whose elements are 8-connected boundary points of a region. Beginning with a point p inside the boundary, the objective is to fill the entire region with 1's.

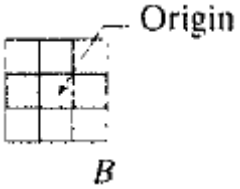
If all nonboundary(background) points are labeled as 0, then the value of 1 to p are used to begin. The following procedure then fills the region with 1's

$X_k = (X_{k-1} \oplus B) \cap A^c$ k=1,2,3,…….

Where X_0 = p, and B is the symmetric structuring element shown in Fig. 16 (c). the algorithm terminates at iteration step k if $X_k = X_{k-1}$. The set union of X_k and A contains the filled set and its boundary.







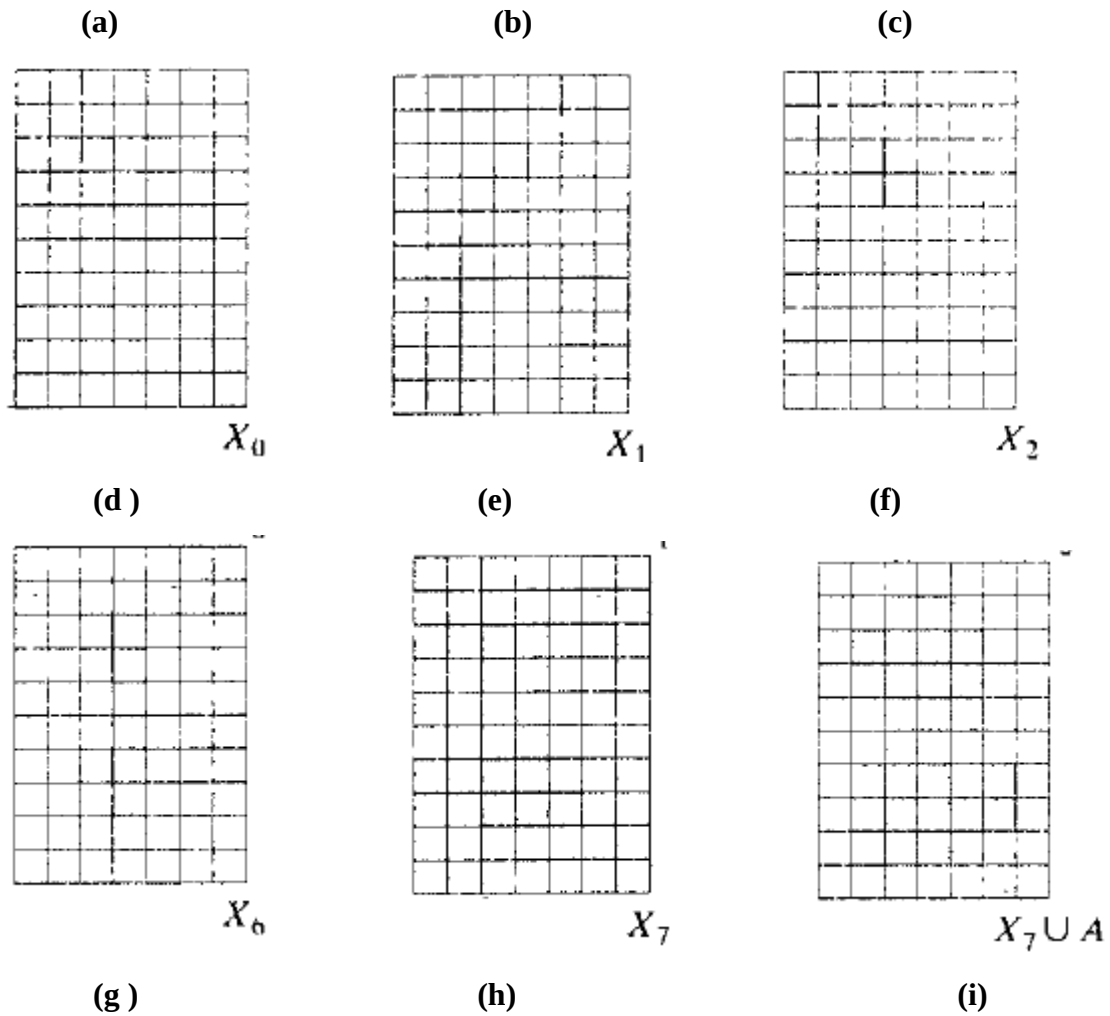


Fig. 16: Region Filling. (a) Set A (b) Complement of A (c) Structuring element B (d) Initial point inside the boundary. (e)-(h) various steps.

Gray Scale Morphology:

In gray scale images on the contrary to binary images we deal with digital image functions of the form f(x,y) as an input image and b(x,y) as a structuring element. (x,y) are integers from Z*Z that represent a coordinates in the image. f(x,y) and b(x,y) are functions that assign gray level value to each distinct pair of coordinates. For example the domain of gray values can be 0-255, whereas 0 – is black, 255- is white.

Dilation – Gray-Scale

Equation for gray-scale dilation is:

$$(f \oplus b)(s,t) =$$
$$\max\{f(s-x,t-y)+b(x,y) | (s-x),(t-y) \in D_f, (x,y) \in D_b\}$$

Df and Db are domains of f and b.

- The condition that (s-x),(t-y) need to be in the domain of f and x,y in the domain of b, is analogous to the condition in the binary definition of dilation, where the two sets need to overlap by at least one element.

We will illustrate the previous equation in terms of

1-D. and we will receive an equation for 1 variable:

$$(f \oplus b)(s) = \max\{f(s-x)+b(x) | (s-x) \in D_f \text{ and } x \in D_b\}$$

The requirements the (s-x) is in the domain of f and x is in the domain of b imply that f and b overlap by at least one element. Unlike the binary case, f, rather than the structuring element b is shifted. Conceptually f sliding by b is really not different than b sliding by f.

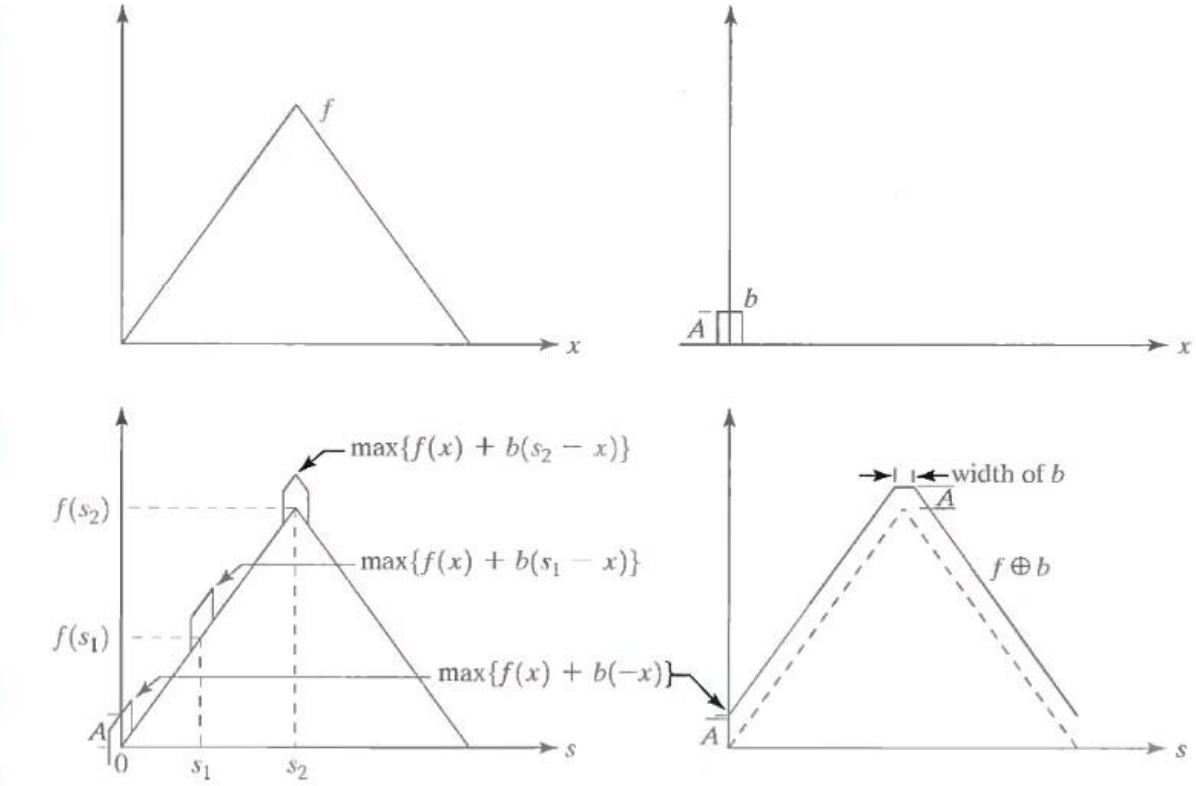
The general effects of performing dilation on a gray scale image is

Twofold:

1. If all the values of the structuring elements are positive than the output image tends to be brighter than the input.

2. Dark details either are reduced or eliminated, depending on how their values and shape relate to the structuring element used for dilation

Dilation – Gray-Scale example



a b
c d

FIGURE 9.27 (a) A simple function. (b) Structuring element of height A. (c) Result of dilation for various positions of sliding b past f. (d) Complete result of dilation (shown solid).

Erosion – Gray-Scale

- Gray-scale erosion is defined as:

$$(f \ominus b)(s,t) = \min\{f(s+x, t+y) - b(x,y) | (s+x), (t+y) \in D_f, (x,y) \in D_b\}$$

- The condition that (s+x),(t+y) have to be in the domain of f, and x,y have to be in the domain of b, is completely analogous to the condition in the binary definition of erosion, where the structuring element has to be completely combined by the set beingeroded.
- The same as in erosion we illustrate with 1-D function

$$(f \ominus b)(s) = \min\{f(s+x) - b(x) | (s+x) \in D_f \text{ and } x \in D_b\}$$

Erosion– Gray-Scale example 1

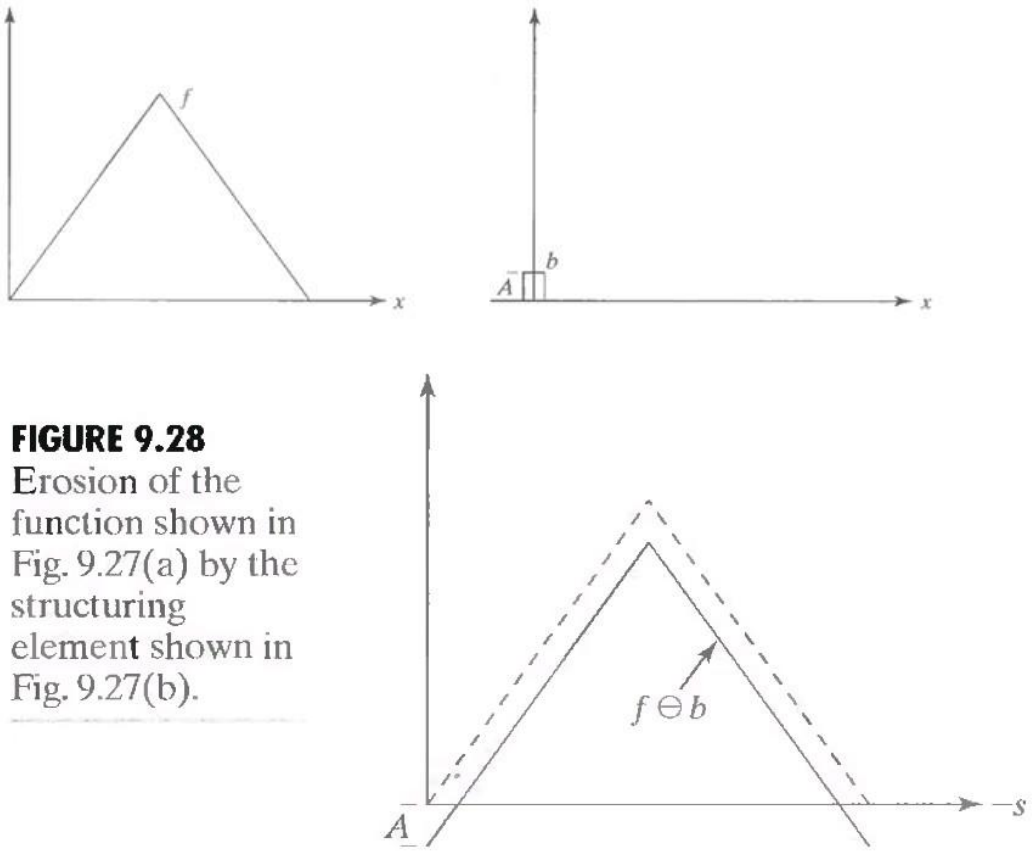
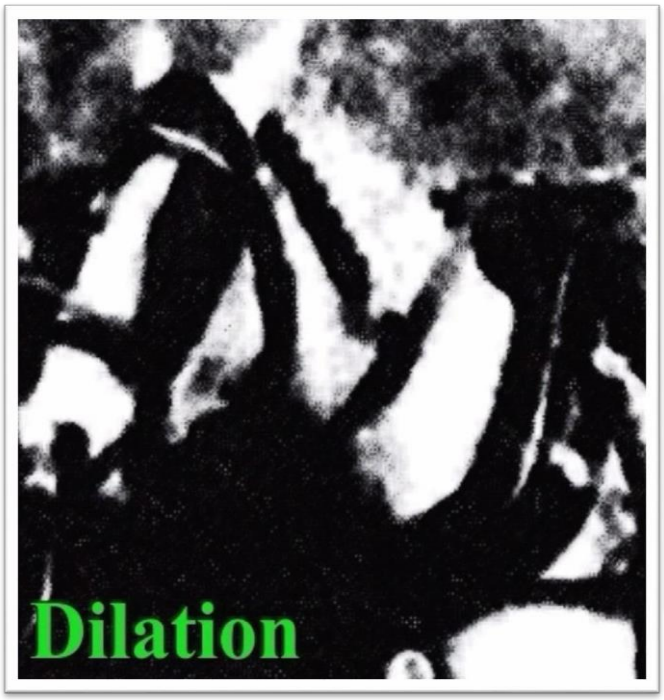


FIGURE 9.28
Erosion of the function shown in Fig. 9.27(a) by the structuring element shown in Fig. 9.27(b).

- General effect of performing an erosion in grayscale images:
    1. If all elements of the structuring element are positive, the output image tends to be darker than the input image.
    2. The effect of bright details in the input image that are smaller in area than the structuring element is reduced, with the degree of reduction being determined by the grayscale values surrounding by the bright detail and by shape and amplitude values of the structuring element itself.
- Similar to binary image grayscale erosion and dilation are duals with respect to function complementation andreflection.

Dilation & Erosion– Gray-Scale

Over Applying the Filter

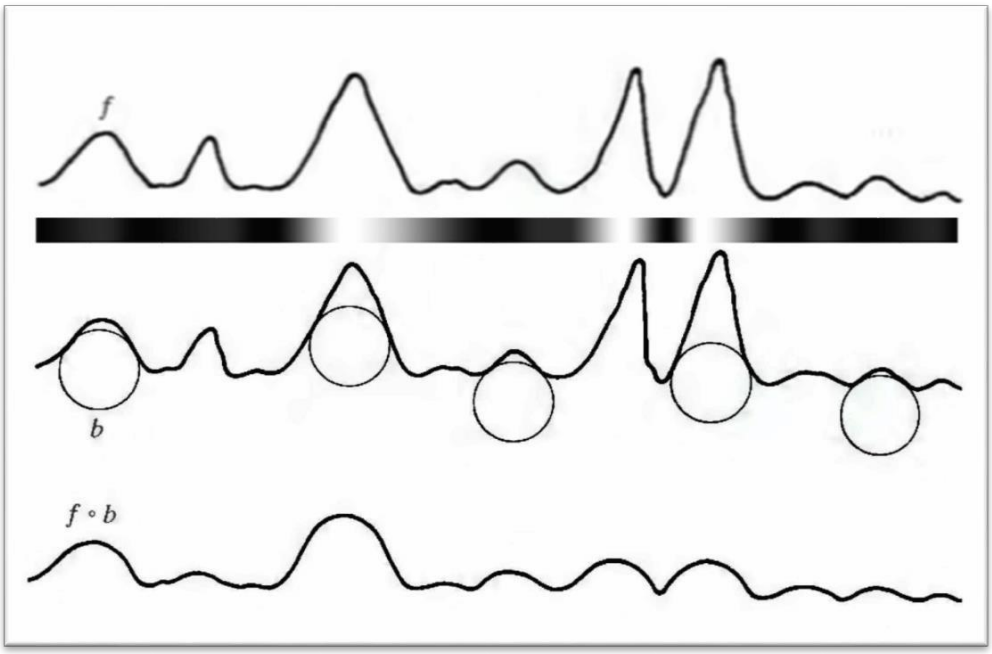


Filter Demonstration

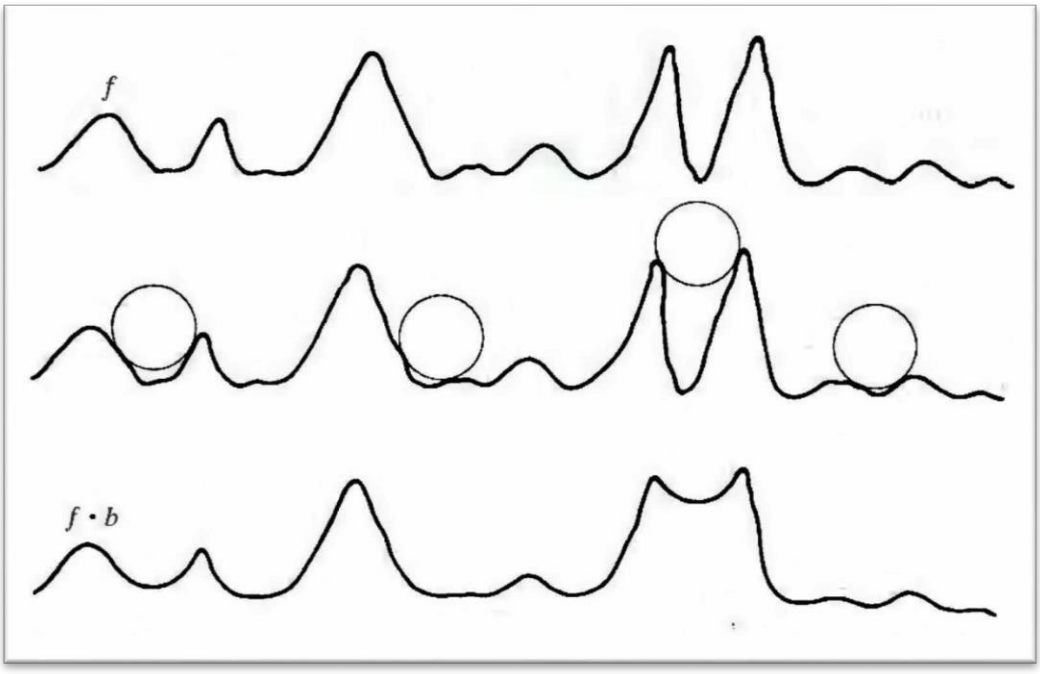
Opening And Closing

- Similar to the binary algorithms
- Opening – $f \circ b = (f \ominus b) \oplus b.$
- Closing – $f \bullet b = (f \oplus b) \ominus b.$
- In the opening of a gray-scale image, we remove small light details, while relatively undisturbed overall gray levels and larger bright features
- In the closing of a gray-scale image, we remove small dark details, while relatively undisturbed overall gray levels and larger dark features

Opening a G-S picture is describable as pushing object B under the scan- line graph, while traversing the graph according the curvature of B



Closing a G-S picture is describable as pushing object B on top of the scan-line graph, while traversing the graph according the curvature of B



The peaks are usually remains in their original form

UNIT- VI
COLOR IMAGE PROCESSING

Color fundamentals.

Color of an object is determined by the nature of the light reflected from it. When a beam of sunlight passes through a glass prism, the emerging beam of light is not white but consists instead of a continuous spectrum of colors ranging from violet at one end to red at the other. As Fig. shows, the color spectrum may be divided into six broad regions: violet, blue, green, yellow, orange, and red. When viewed in full color (Fig.), no color in the spectrum ends abruptly, but rather each color blends smoothly into the next.

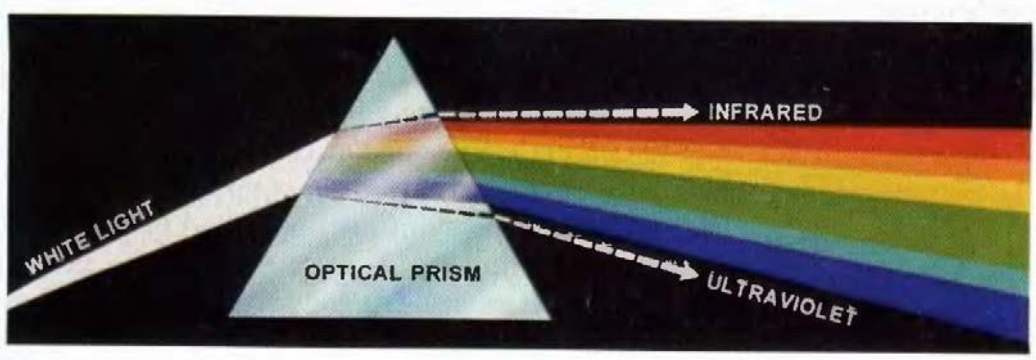


Fig. Color spectrum seen by passing white light through a prism.

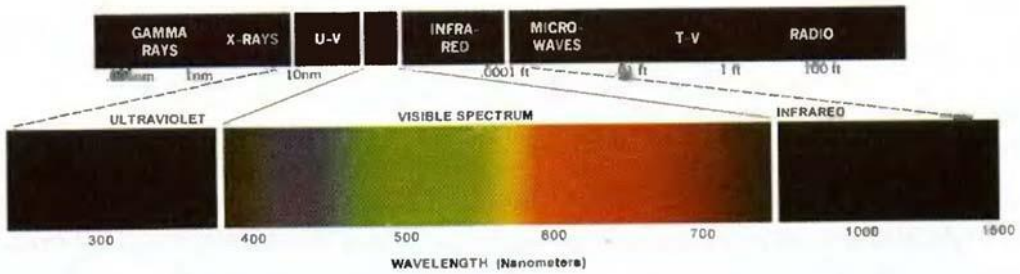


Fig. Wavelengths comprising the visible range of the electromagnetic spectrum.

As illustrated in Fig. , visible light is composed of a relatively narrow band of frequencies in the electromagnetic spectrum. A body that reflects light that is balanced in all visible wavelengths appears white to the observer. However, a body that favors reflectance in a limited range of the visible spectrum exhibits some shades of color. For example, green objects reflect light with wavelengths primarily in the 500 to 570 nm range while absorbing most of the energy at other wavelengths.

Characterization of light is central to the science of color. If the light is achromatic (void of color), its only attribute is its intensity, or amount. Achromatic light is what viewers see on a black and white television set.

Three basic quantities are used to describe the quality of a chromatic light source: radiance, luminance, and brightness.

Radiance:

Radiance is the total amount of energy that flows from the light source, and it is usually measured in watts (W).

Luminance:

Luminance, measured in lumens (lm), gives a measure of the amount of energy an observer perceives from a light source. For example, light emitted from a source operating in the far infrared region of the spectrum could have significant energy (radiance), but an observer would hardly perceive it; its luminance would be almost zero.

Brightness:

Brightness is a subjective descriptor that is practically impossible to measure. It embodies the achromatic notion of intensity and is one of the key factors in describing color sensation.

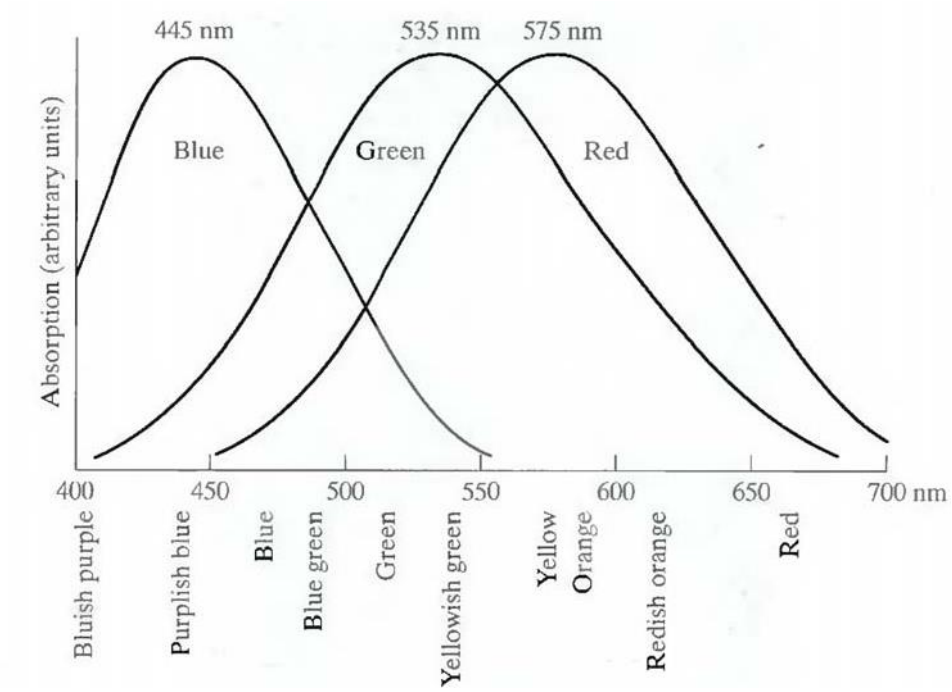


Fig. Absorption of light by the red, green, and blue cones in the human eye as a function of wavelength.

Cones are the sensors in the eye responsible for color vision. Detailed experimental evidence has established that the 6 to 7 million cones in the human eye can be divided into three principal sensing categories, corresponding roughly to red, green, and blue. Approximately 65%of all cones are sensitive to red light, 33% are sensitive to green light, and only about 2% are sensitive to blue (but the blue cones are the most sensitive). Figure 5.1.3 shows average experimental curves detailing the absorption of light by the red, green, and blue cones in the eye. Due to these absorption characteristics of the human eye, colors arc seen as variable combinations of the so- called primary colors red (R), green (G), and blue (B).

The primary colors can be added to produce the secondary colors of light --magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). Mixing the three primaries, or a

secondary with its opposite primary color, in the right intensities produces white light.

The characteristics generally used to distinguish one color from another are brightness, hue, and saturation. Brightness embodies the chromatic notion of intensity. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Hue represents dominant color as perceived by an observer. Saturation refers to the relative purity or the amount of white light mixed with a hue. The pure spectrum colors are fully saturated. Colors such as pink (red and white) and lavender (violet and white) are less saturated, with the degree of saturation being inversely proportional to the amount of white light-added.

Hue and saturation taken together are called chromaticity, and. therefore, a color may be characterized by its brightness and chromaticity.

Color models.

The purpose of a color model (also called color space or color system) is to facilitate the specification of colors in some standard, generally accepted way. In essence, a color model is a specification of a coordinate system and a subspace within that system where each color is represented by a single point.

The RGB Color Model:

In the RGB model, each color appears in its primary spectral components of red, green, and blue. This model is based on a Cartesian coordinate system. The color subspace of interest is the cube shown in Fig. 5.2, in which RGB values are at three corners; cyan, magenta, and yellow are at three other corners; black is at the origin; and white is at the corner farthest from the origin. In this model, the gray scale (points of equal RGB values) extends from black to white along the line joining these two points. The different colors in this model arc points on or inside the cube, and are defined by vectors extending from the origin.

For convenience, the assumption is that all color values have been normalized so that the cube shown in Fig. 5.2 is the unit cube. That is, all values of R, G. and B are assumed to be in the range [0, 1].

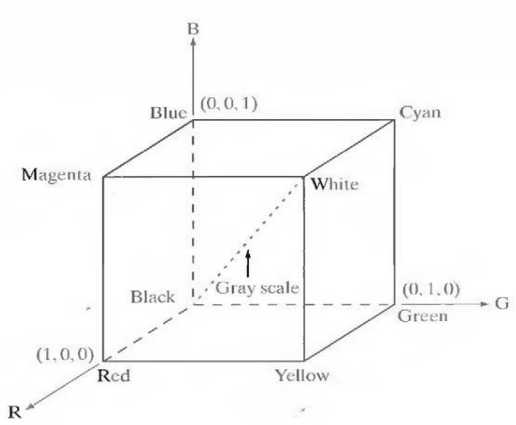


Fig. 5.2 Schematic of the RGB color cube.

Images represented in the RGB color model consist of three component images, one for each primary color. When fed into an RGB monitor, these three images combine on the phosphor screen to produce a composite color image. The number of bits used to represent each pixel in RGB space is called the pixel depth.

Consider an RGB image in which each of the red, green, and blue images is an 8-bit image. Under these conditions each RGB color pixel [that is, a triplet of values (R, G, B)] is said to have a depth of 24 bits C image planes times the number of bits per plane). The term full-color image is used often to denote a 24-bit RGB color image. The total number of colors in a 24-bit RGB image is $(2^8)^3 = 16{,}777{,}216$.

RGB is ideal for image color generation (as in image capture by a color camera or image display in a monitor screen), but its use for color description is much more limited.

CMY color model.

Cyan, magenta, and yellow are the secondary colors of light or, alternatively, the primary colors of pigments. For example, when a surface coated with cyan pigment is illuminated with white light, no red light is reflected from the surface. That is, cyan subtracts red light from reflected white light, which itself is composed of equal amounts of red, green, and blue light.

Most devices that deposit colored pigments on paper, such as color printers and copiers, require CMY data input or perform an RGB to CMY conversion internally. This conversion is performed using the simple operation (1) where, again, the assumption is that all color values have been normalized to the range [0, 1]. Equation (1) demonstrates that light reflected from a surface coated with pure cyan does not contain red (that is, C = 1 — R in the equation).

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (1)$$

Similarly, pure magenta does not reflect green, and pure yellow does not reflect blue. Equation (1) also reveals that RGB values can be obtained easily from a set of CMY values by subtracting the individual CMY values from 1. As indicated earlier, in image processing this color model is used in connection with generating hardcopy output, so the inverse operation from CMY to RGB generally is of little practical interest.

Equal amounts of the pigment primaries, cyan, magenta, and yellow should produce black. In practice, combining these colors for printing produces a muddy-looking black.

HSI color model.

When humans view a color object, we describe it by its hue, saturation, and brightness. Hue is a color attribute that describes a pure color (pure yellow, orange, or red), whereas saturation gives a measure of the degree to which a pure color is diluted by white light. Brightness is a subjective descriptor that is practically impossible to measure. It embodies the achromatic notion of intensity and is one of the key factors in describing color sensation.

Intensity (gray level) is a most useful descriptor of monochromatic images. This quantity definitely is measurable and easily interpretable. The HSI (hue, saturation, intensity) color model, decouples the intensity component from the color-carrying information (hue and saturation) in a color image. As a result, the HSI model is an ideal tool for developing image processing algorithms based on color descriptions that are natural and intuitive to humans.

In Fig 5.4 the primary colors are separated by 120°. The secondary colors are 60° from the primaries, which means that the angle between secondaries is also 120°. Figure 5.4(b) shows the same hexagonal shape and an arbitrary color point (shown as a dot).The hue of the point is determined by an angle from some reference point. Usually (but not always) an angle of 0° from the red axis designates 0 hue, and the hue increases counterclockwise from there. The saturation (distance from the vertical axis) is the length of the vector from the origin to the point. Note that the origin is defined by the intersection of the color plane with the vertical intensity axis. The important components of the HSI color space are the vertical intensity axis, the length of the vector to a color point, and the angle this vector makes with the red axis.

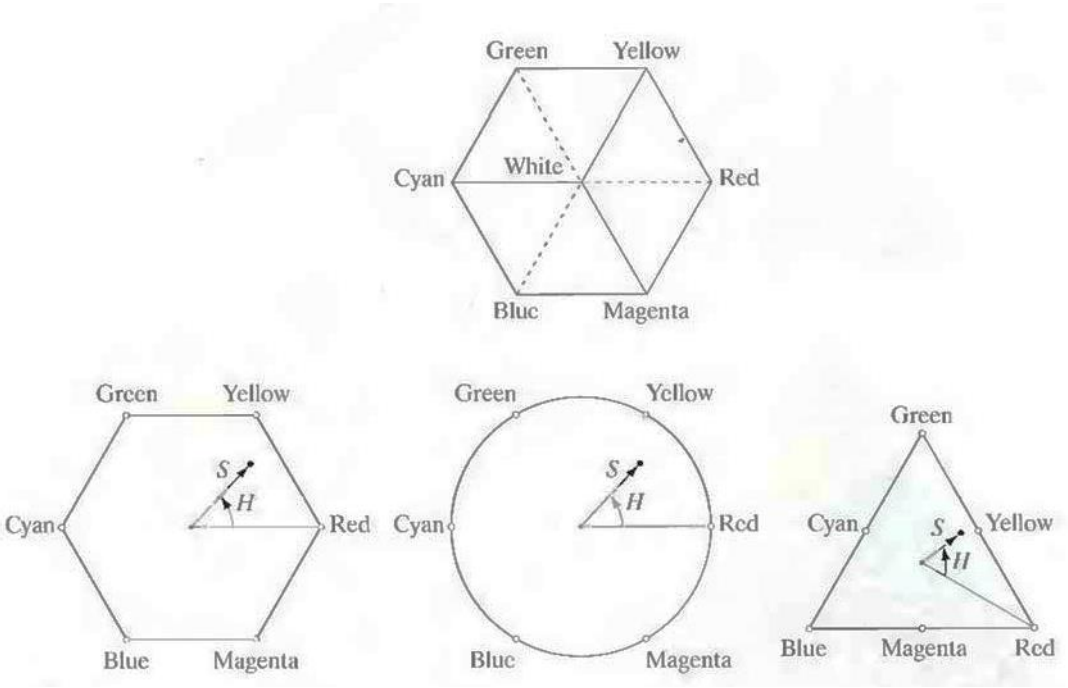


Fig 5.4 Hue and saturation in the HSI color model.

procedure for conversion from RGB color model to HSI color model.

Given an image in RGB color format, the H component of each RGB pixel is obtained using the equation

$$H = \begin{cases} \theta & \text{if } B \le G \\ 360 - \theta & \text{if } B > G \end{cases} \quad (1)$$

With

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}. \quad (2)$$

The saturation component is given by

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)]. \quad (3)$$

Finally, the intensity component is given by

$$I = \frac{1}{3}(R+G+B). \quad (4)$$

It is assumed that the RGB values have been normalized to the range [0, 1] and that angle θ is measured with respect to the red axis of the HST space. Hue can be normalized to the range [0, 1] by dividing by 360° all values resulting from Eq. (1). The other two HSI components already are in this range if the given RGB values are in the interval [0, 1].

procedure for conversion from HSI color model to RGB color model.

Given values of HSI in the interval [0,1], one can find the corresponding RGB values in the same range. The applicable equations depend on the values of H. There are three sectors of interest, corresponding to the 120° intervals in the separation of primaries.\

RG sector ($0^o \leq$ H <120°):

When H is in this sector, the RGB components are given by the equations

B = I (1 – S)

$$G = 3I - (R + B)$$

$$R = I\,[1 + (S * \cos H / \cos(60^o - H)]$$

GB sector ($120^o \leq H < 240^o$):

If the given value of H is in this sector, first subtract 120° from it.

$$H = H - 120^0$$

Then the RGB components are

$$R = I\,(1 - S)$$
$$B = 3I - (R + G)$$

$$G = I\,[1 + (S * \cos H / \cos(60^o - H)]$$

BR sector ($240^o \leq H \leq 360^o$):

If H is in this range, subtract 240^o from it

$$H = H - 240^0$$

Then the RGB components are

$$G = I\,(1 - S)$$

$$R = 3I - (B + G)$$

$$B = I\,[1 + (S * \cos H / \cos(60^o - H)]$$

Pseudocolor Image Processing.

Pseudocolor (also called false color) image processing consists of assigning colors to gray values based on a specified criterion. The term pseudo or false color is used to differentiate the process of assigning colors to monochrome images from the processes associated with true color images. The process of gray level to color transformations is known as pseudocolor image processing.

The two techniques used for pseudocolor image processing are,

(i) Intensity Slicing

(ii) Gray Level to Color Transformation

(i) Intensity Slicing:

The technique of intensity (sometimes called density) slicing and color coding is one of the simplest examples of pseudocolor image processing. If an image is interpreted as a 3-D function (intensity versus spatial coordinates), the method can be viewed as one of placing planes parallel to the coordinate plane of the image; each plane then "slices" the function in the area of intersection. Figure 5.8 shows an example of using a plane at $f(x, y) = l_i$ to slice the image function into two levels.

If a different color is assigned to each side of the plane shown in Fig. 5.8, any pixel whose gray level is above the plane will be coded with one color, and any pixel below the plane will be coded with the other. Levels that lie on the plane itself may be arbitrarily assigned one of the two colors. The result is a two-color image whose relative appearance can be controlled by moving the slicing plane up and down the gray-level axis.

In general, the technique may be summarized as follows. Let [0, L - 1] represent the gray scale, let level l_o represent black [f(x, y) = 0], and level l_{L-1} represent white [f(x, y) = L - 1]. Suppose that P planes perpendicular to the intensity axis are defined at levels $l_1, l_2, \ldots, l_p$. Then, assuming that $0 < P < L - 1$, the P planes partition the gray scale into P + 1 intervals, $V_1, V_2, \ldots, V_p + 1$. Gray-level to color assignments are made according to the relation

$$f(x, y) = c_k \quad \text{if} \quad f(x, y) \in V_k$$

where c_k is the color associated with the kth intensity interval V_k defined by the partitioning planes at $l = k - 1$ and $l = k$.

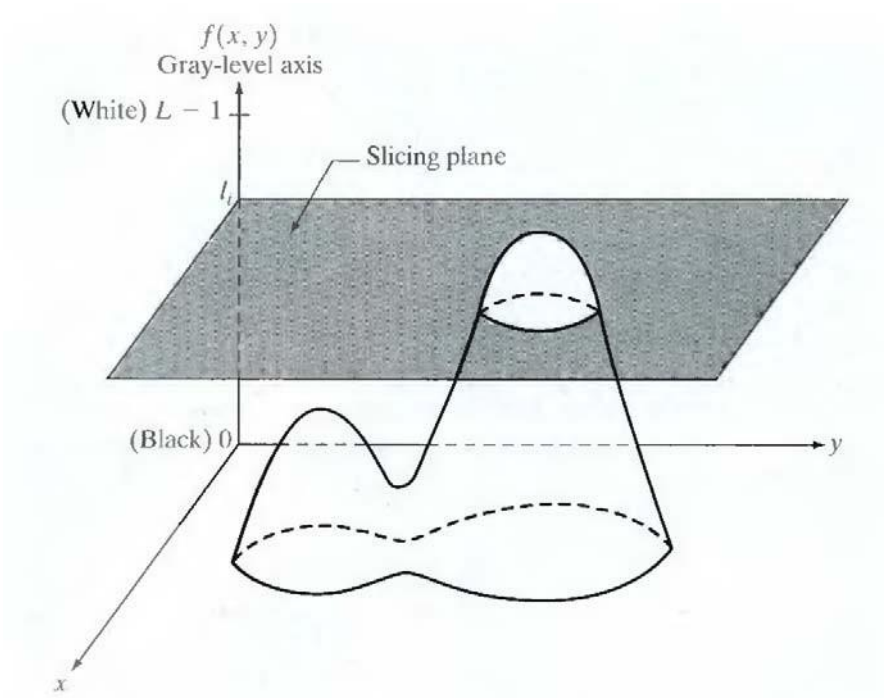


Fig 5.8.1 Geometric interpretation of the intensity-slicing technique.

The idea of planes is useful primarily for a geometric interpretation of the intensity-slicing technique. Figure 5.8.2 shows an alternative representation that defines the same mapping as in Fig. 5.8.1. According to the mapping function shown in Fig. 5.8.2, any input gray level is assigned one of two colors, depending on whether it is above or below the value of l_i When more levels are used, the mapping function takes on a staircase form.

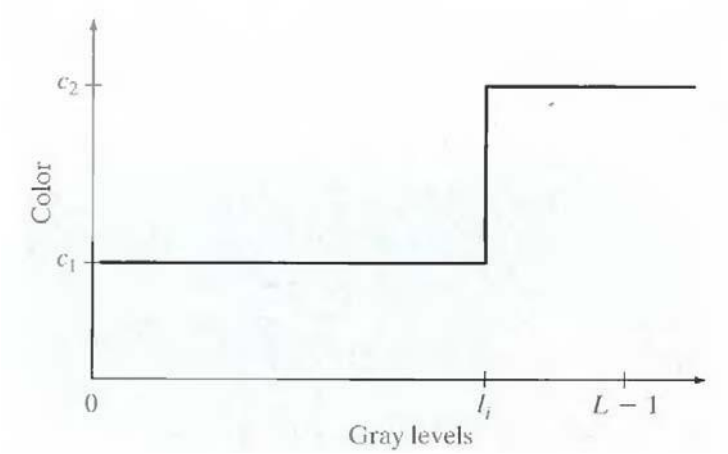


Fig 5.8.2 An alternative representation of the intensity-slicing technique.

(ii) Gray Level to Color Transformation:

The idea underlying this approach is to perform three independent transformations on the gray level of any input pixel. The three results are then fed separately into the red, green, and blue channels of a color television monitor. This method produces a composite image whose color content is modulated by the nature of the transformation functions. Note that these are transformations on the gray-level values of an image and are not functions of position.

In intensity slicing, piecewise linear functions of the gray levels are used to generate colors. On the other hand, this method can be based on smooth, nonlinear functions, which, as might be expected, gives the technique considerable flexibility.

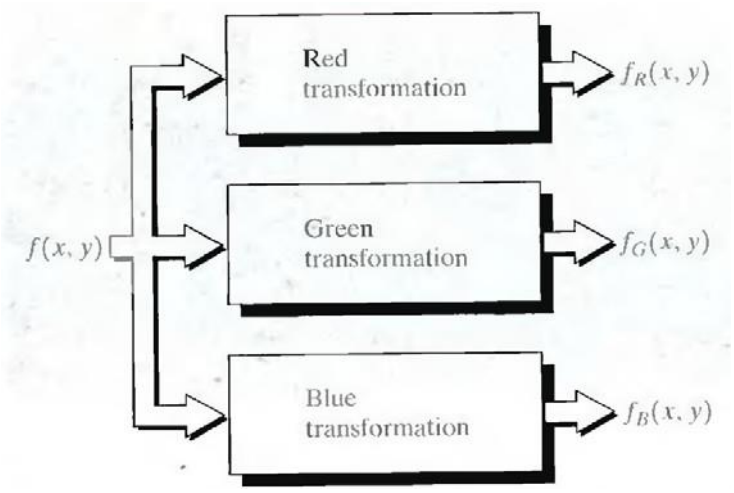


Fig. 5.8.3 Functional block diagram for pseudocolor image processing.

The output of each transformation is a composite image.

Basics of full color image processing.

Full-color image processing approaches fall into two major categories. In the first category, each component image is processed individually and then form a composite processed color image from the individually processed components. In the second category, one works with color pixels directly. Because full-color images have at least three components, color pixels really are vectors. For example, in the RGB system, each color point can be interpreted as a vector extending from the origin to that point in the RGB coordinate system.

Let c represent an arbitrary vector in RGB color space:

$$\mathbf{c} = \begin{bmatrix} c_R \\ c_G \\ c_B \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \quad (1)$$

This equation indicates that the components of c are simply the RGB components of a color image at a point. If the color components are a function of coordinates (x, y) by using the notation

$$\mathbf{c}(x, y) = \begin{bmatrix} c_R(x, y) \\ c_G(x, y) \\ c_B(x, y) \end{bmatrix} = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix}. \quad (2)$$

For an image of size M X N, there are MN such vectors, c(x, y), for x = 0,1, 2,...,M- l; y = 0,1,2,...,N- 1.

It is important to keep clearly in mind that Eq. (2) depicts a vector whose components are spatial variables in x and y.

In order for per-color-component and vector-based processing to be equivalent, two conditions have to be satisfied: First, the process has to be applicable to both vectors and scalars. Second, the operation on each component of a vector must be independent of the other components.

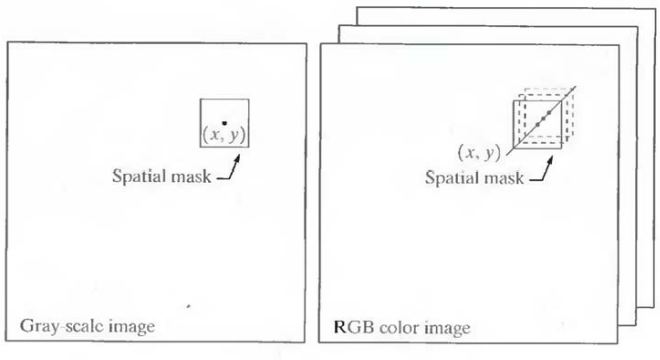


Fig 9 Spatial masks for gray-scale and RGB color images.

Fig 9 shows neighborhood spatial processing of gray-scale and full-color images. Suppose that the process is neighborhood averaging. In Fig. 9(a), averaging would be Accomplished by summing the gray levels of all the pixels in the neighborhood and dividing by the total number of pixels in the neighborhood. In Fig. 9(b), averaging would be done by summing all the vectors in the neighborhood and dividing each component by the total number of vectors in the neighborhood. But each component of the average vector is the sum of the pixels in the image corresponding to that component, which is the same as the result that would be obtained if the averaging were done on a per-color-component basis and then the vector was formed.

Color Transformations

- It is useful to think of a color image as a vector valued image, where each pixel has associated with it, as vector of three values.

- Each components of this vector corresponds to a different aspect of color, depending on the color model being used. For example, in an RGB model, the three values in the vector respectively denote the red, green, and blue components of the color of that pixel. In an HSI model, the three values in the vector denote the hue, saturation, and intensity of the color of that pixel.

- We can think of color transformations as a transformation of vectors.

$$s_i = T_i(r_1, r_2, r_3), \qquad i = 1,2,3.$$

- Here (r_1, r_2, r_3) represent the color components of the input image $f(m,n)$, whereas (s_1, s_2, s_3) represent the color components of the output image $g(m,n)$.

- In theory, any color transformation can be performed in any color space model. However, in practice, some transformations are better suited to specific models.

- Moreover, the cost of conversion between the models must be a factor in implementation of a particular transformation.

Example: Modifying intensity

- Consider a simple transformation involving intensity scaling:

$$g(m, n) = kf(m, n)$$

where $0 < k < 1$ is a scaling factor.

- In HSI space, this can be implemented as

$$s_1 = r_1, \quad s_2 = r_2, \quad \text{and} \quad s_3 = kr_3$$

- In RGB space, this can be implemented as

$$s_1 = kr_1, \quad s_2 = kr_2, \quad \text{and} \quad s_3 = kr_3$$

- In CMY space, this can be implemented as

$$s_1 = kr_1 + 1 - k, \quad s_2 = kr_2 + 1 - k, \quad \text{and} \quad s_3 = kr + 1 - k_3$$

- Although the fewest operations are involved in the HSI space, the computations involved in conversion back and forth from RGB space more than offsets any savings in computation in HSI space.

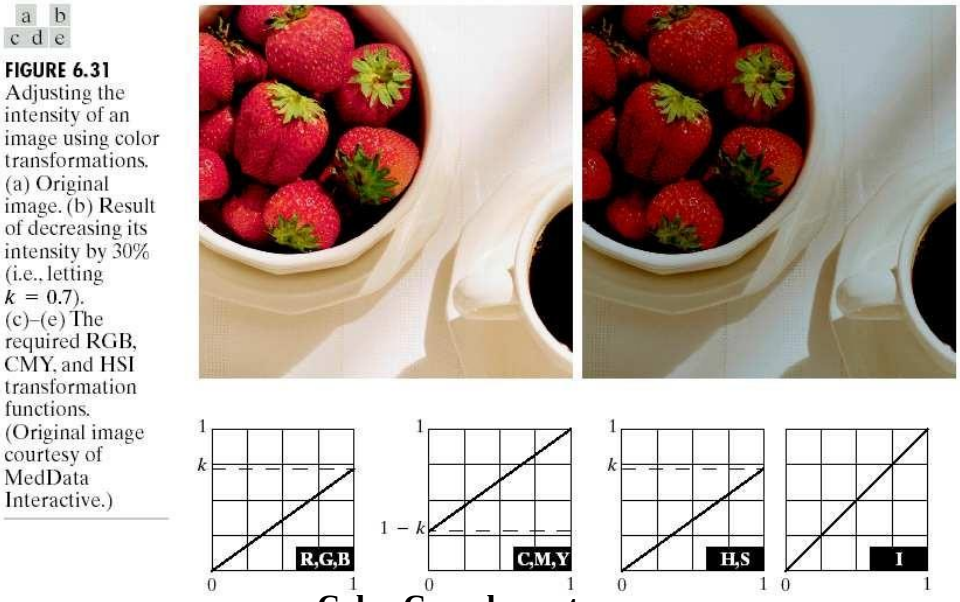


FIGURE 6.31 Adjusting the intensity of an image using color transformations. (a) Original image. (b) Result of decreasing its intensity by 30% (i.e., letting $k = 0.7$). (c)–(e) The required RGB, CMY, and HSI transformation functions. (Original image courtesy of MedData Interactive.)

Color Complement

- Hues opposite one another in a color circle are called complements.

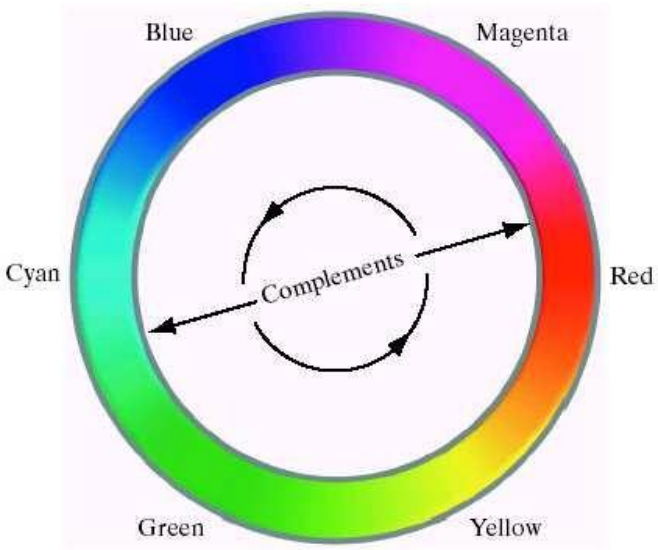


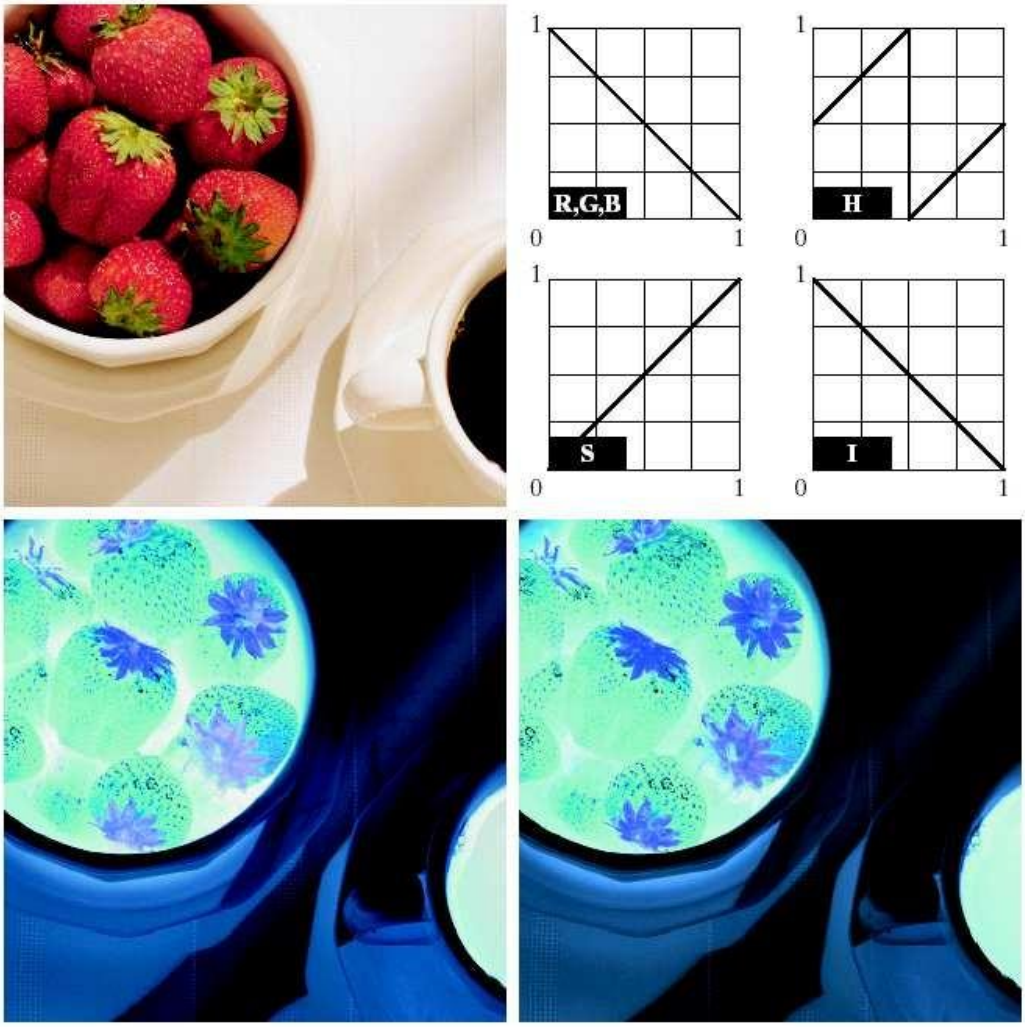
FIGURE 6.32 Complements on the color circle.

- This is analogous to gray-scale negatives.

- As in the grayscale case, this transformation is useful in enhancing details embedded in dark portions of a color image.

- Complementation can be easily implemented in the RGB space. However, there is no simple equivalent of this in the HIS space. An approximation is possible.

Example



a b
c d

FIGURE 6.33 Color complement transformations. (a) Original image. (b) Complement transformation functions. (c) Complement of (a) based on the RGB mapping functions. (d) An approximation of the RGB complement using HSI transformations.

- Color slicing is similar to intensity slicing ---

Smoothing of Color Images

- Just like the case of grayscale images, smoothing of color images can be performed to remove abrupt transitions of gray values.
- This can be done either in the RGB domain or the HSI domain.
- In the RGB domain, all the three color components are individually transformed by an appropriate smoothing mask, say a 3×3 mask:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

- In the HSI domain, only the I component is transformed by means of a spatial smoothing mask, leaving the H and S components unchanged.
- In general, the final result in the two cases would be different. Because the average of two colors is a color intermediate between the two, the former approach has the potential of introducing colors not present in the original image. The latter approach does not have this problem, since the Hue and Saturation components are preserved.

Example

a b
c d

FIGURE 6.38
(a) RGB image.
(b) Red component image.
(c) Green component.
(d) Blue component.

a b c

FIGURE 6.39 HSI components of the RGB color image in Fig. 6.38(a). (a) Hue. (b) Saturation. (c) Intensity.

a b c

FIGURE 6.40 Image smoothing with a 5 × 5 averaging mask. (a) Result of processing each RGB component image. (b) Result of processing the intensity component of the HSI image and converting to RGB. (c) Difference between the two results.

Color Image Sharpening

- Sharpening of color images can be performed in a manner analogous to smoothing, using appropriate masks, say the Laplacian mask

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Example

a b c

FIGURE 6.41 Image sharpening with the Laplacian. (a) Result of processing each RGB channel. (b) Result of processing the intensity component and converting to RGB. (c) Difference between the two results.

color segmentation process.

Segmentation is a process that partitions an image into regions and partitioning an image into regions based on color is known as color segmentation.

Segmentation in HSI Color Space:

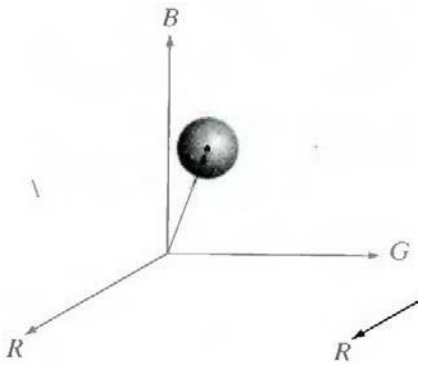
If anybody want to segment an image based on color, and in addition, to carry out the process on individual planes, it is natural to think first of the HSI space because color is conveniently represented in the hue image. Typically, saturation is used as a masking image in order to isolate further regions of interest in the hue image. The intensity image is used less frequently for segmentation of color images because it carries no color information.

Segmentation in RGB Vector Space:

Although, working in HSI space is more intuitive, segmentation is one area in which better results generally are obtained by using RGB color vectors. The approach is straightforward. Suppose that the objective is to segment objects of a specified color range in an RGB image. Given a set of sample color points representative of the colors of interest, we obtain an estimate of the "average" color that we wish to segment. Let this average color be denoted by the RGB vector **a**. The objective of segmentation is to classify each RGB pixel in a given image as having a color in the specified range or not. In order to perform this comparison, it is necessary to have a measure of similarity. One of the simplest measures is the Euclidean distance. Let **z** denote an arbitrary point in RGB space. **z** is similar to **a** if the distance between them is less than a specified threshold, D_o. The Euclidean distance between **z** and **a** is given by

$$\begin{aligned} D(\mathbf{z},\mathbf{a}) &= \|\mathbf{z}-\mathbf{a}\| \\ &= \left[(\mathbf{z}-\mathbf{a})^T(\mathbf{z}-\mathbf{a})\right]^{\frac{1}{2}} \\ &= \left[(z_R-a_R)^2+(z_G-a_G)^2+(z_B-a_B)^2\right]^{\frac{1}{2}} \end{aligned}$$

where the subscripts R, G, and B, denote the RGB components of vectors **a** and **z** .The locus of points such that D(z, a) ≤ Do is a solid sphere of radius D_o.



Points contained within or on the surface of the sphere satisfy the specified color criterion; points outside the sphere do not. Coding these two sets of points in the image with, say, black and white, produces a binary segmented image.

A useful generalization of previous equation is a distance measure of the form

$$D(z, a) = [(z-a)^T C^{-1} (z-a)]^{1/2}$$

Where C is the covariance matrix1 of the samples representative of the color to be segmented. The above equation represents an ellipse with color points such that D(z, a) ≤ D_o.

Noise in color images

- The previously discussed noise models are applicable to color images as well.
- Typically, noise affects all the three color components.
- Usually, across the three color channels, the noise is independent and its statistical characteristic are identical.
- However, due to different illumination conditions or selective malfunction of camera hardware in a particular channel, this may not be the case.
- Noise filtering by means of a simple averaging can be accomplished by performing the operation independently on the R, G, and B channels and combining the results.
- However, more complicated filters like the median filter are not as straight-forward to formulate in the color domain and will not be pursued here.

Color Image Compression:

Image compression is an application of data compression that encodes the original image with few bits. The objective of image compression is to reduce the redundancy of the image and to store or transmit data in an efficient form. Fig 1.1 shows the block diagram of the general image storage system. The main goal of such system is to reduce the storage quantity as much as possible, and the decoded image displayed in the monitor can be similar to the original image as much as can be. The essence of each block will be introduced in the following sections

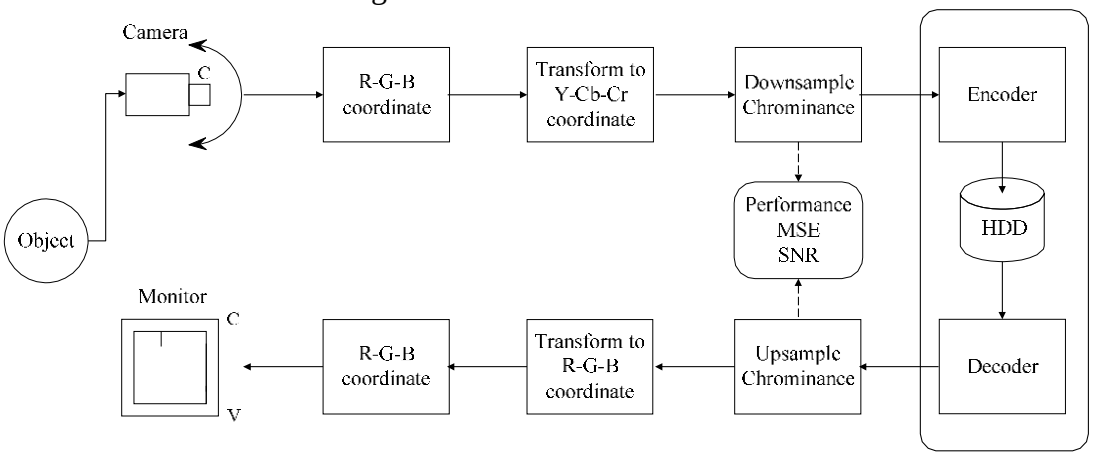


Fig. General Image Storage System

Color Specification

The Y, Cb, and Cr components of one color image are defined in YUV color coordinate, where Y is commonly called the luminance and Cb, Cr are commonly called the chrominance. The meaning of luminance and chrominance is described as follows

- **Luminance:** received brightness of the light, which is proportional to the total energy in the visible band.
- **Chrominance:** describe the perceived color tone of a light, which depends on the wavelength composition of light chrominance is in turn characterized by two attributes – hue and saturation.
  1. **hue:** Specify the color tone, which depends on the peak wavelength of the light
  2. **saturation:** Describe how pure the color is, which depends on the spread or bandwidth of the light spectrum

The RGB primary commonly used for color display mixes the luminance and chrominance attributes of a light. In many applications, it is desirable to describe a color in terms of its luminance and chrominance content separately, to enable more efficient processing and transmission of color signals. Towards this goal, various three-component color coordinates have been developed, in which one component reflects the luminance and the other two collectively characterize hue and saturation.

One such coordinate is the YUV color space. The $[Y\ Cb\ Cr]^T$ values in the YUV Coordinate are related to the $[R\ G\ B]^T$ values in the RGB coordinate by

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.334 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

Similarly, if we would like to transform the YUV coordinate back to RGB coordinate, the inverse matrix can be calculated from (1.1), and the inverse transform is taken to obtain the corresponding RGB components.

Spatial Sampling of Color Component

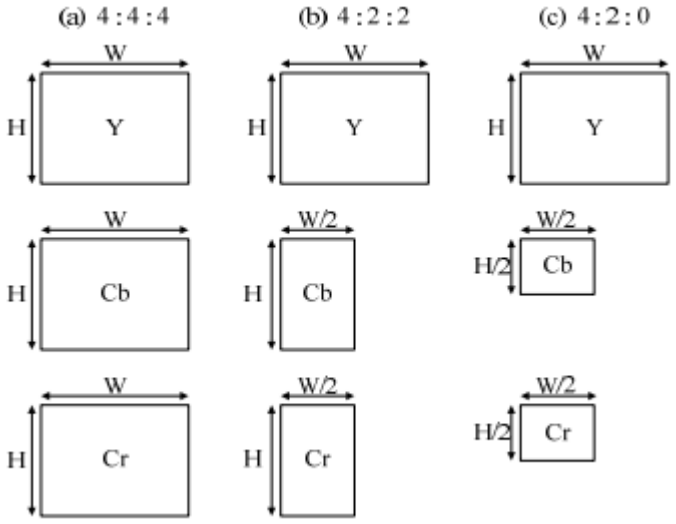


Fig. The three different chrominance down sampling format

Because the eyes of human are more sensitive to the luminance than the chrominance, the sampling rate of chrominance components is half that of the luminance component. This will result in good performance in image compression with almost no loss of characteristics in visual perception of the new up sampled image. There are three color formats in the baseline system:

- 4:4:4 format: The sampling rate of the luminance component is the same

as those of the chrominance.

- 4:2:2 format: There are 2 Cb samples and 2 Cr samples for every 4 Y samples. This leads to half number of pixels in each line, but the same number of lines per frame.

- 4:2:0 format: Sample the Cb and Cr components by half in both the horizontal and vertical directions. In this format, there are also 1 Cb sample and 1 Cr sample for every 4 Y samples.

At the decoder, the down sampled chrominance components of 4:2:2 and 4:2:0 formats should be up sampled back to 4:4:4 format.