Image Coding

10.0 INTRODUCTION

A major objective of image coding is to represent an image with as few bits as possible while preserving the level of quality and intelligibility required for the given application. Image coding has two major application areas. One is the reduction of channel bandwidth required for image transmission systems. Examples of this application include digital television, video conferencing, and facsimile. The other application is reduction of storage requirements. Examples of this application include reduction in the storage of image data from space programs and of video data in digital VCRs.

The levels of image quality and intelligibility required vary widely, depending on the application. In such applications as storage of image data from space programs and images of objects with historical value that no longer exist, regeneration of the original digital data may be very expensive or even impossible. In such applications, we may want to preserve all the information in the original digital data for possible future use. Image coding techniques which do not destroy any information and which allow exact reconstruction of the original digital data are said to be *information-preserving*. In applications such as digital television, it is not necessary for the coder to be information-preserving. In such applications, high quality is very important, but some information in the original data may be destroyed, so long as the decoded video on the TV monitor is acceptable to human viewers. In applications such as remotely piloted vehicles (RPVs), image intelligibility is essential, but we may be able to sacrifice a significant degree of quality. The more quality and intelligibility we can sacrifice, the lower will be the required bit rate.

Image coding is related to image enhancement and restoration. If we can enhance the visual appearance of the reconstructed image or if we can reduce the degradation that results from the image coding algorithm (quantization noise being an example), we may be able to reduce the number of bits required to represent an image at a given level of quality and intelligibility, or conversely to hold the number of bits steady while improving the image quality and intelligibility.

17.2 LOSSLESS COMPRESSION TECHNIQUES

Lossless data compression algorithms fall into two broad categories: dictionary-based techniques and statistical methods. Dictionary-based techniques generate a compressed file

containing fixed-length codes (usually 12 to 16 bits), each of which represents a particular sequence of values in the original file.

Statistical methods implement data compression by representing frequently occurring characters in the file with fewer bits than they do less commonly occurring ones [2]. This is the approach Samuel F. B. Morse used when he defined the international telegraph code. The often used letter *e*, for example, is represented by a single dot, whereas the much less common *z* is coded as dash, dash, dot, dot.

17.2.1 Dictionary-Based Techniques

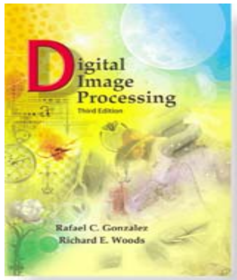
17.2.1.1 Run-Length Encoding

The simplest dictionary-based data compression technique is *run length encoding* (RLE). Images—particularly those having few gray levels—often contain regions of adjacent pixels, all with the same gray level or color. In an image being stored line by line, a series of pixels having the same gray-level value is called a *run*. One can store a code specifying that value, followed by the length of the run, rather than simply storing the same value many times over. This is run-length encoding. It achieves considerable compaction, for example, with graphics and with images of objects residing upon a constant background. Other types of images compress poorly. Under worst case conditions (for example, where every pixel differs from its neighbors) RLE can actually double the size of the file.

17.2.1.2 LZW Encoding

LZ coding is a lossless technique first described by Lemple and Ziv [3,4]. It was extended by Welch [5] to form the widely used, proprietary *LZW algorithm* [6]. Like RLE, it effects compression by encoding strings of characters. However, unlike RLE, it builds up a table of strings (particular sequences of bytes) and their corresponding codes as it encodes the file. A file of 8-bit bytes can be encoded, for example, into 12-bit codes. Of the 4,096 possible codes, 256 of them represent all possible single bytes. The remaining 3,840 are assigned to strings as they are encountered in the data during compression.

The first time a string not already in the table occurs, it is stored in full, along with the code that is assigned to it. Thereafter, when that string occurs again, only its code is stored. This squeezes redundancy out of the file. Not only is the string table built dynamically during compression, but it need not be stored with the compressed file: The decompression algorithm can reconstruct it from the information in the compressed file.



Name	Organization	Description
Continuous-Tone Still Images		
BMP	Microsoft	*Windows Bitmap*. A file format used mainly for simple uncompressed images.
GIF	CompuServe	*Graphic Interchange Format*. A file format that uses lossless LZW coding [8.2.4] for 1- through 8-bit images. It is frequently used to make small animations and short low resolution films for the World Wide Web.
PDF	Adobe Systems	*Portable Document Format*. A format for representing 2-D documents in a device and resolution independent way. It can function as a container for JPEG, JPEG 2000, CCITT, and other compressed images. Some PDF versions have become ISO standards.
PNG	*World Wide Web Consortium* (W3C)	*Portable Network Graphics*. A file format that losslessly compresses full color images with transparency (up to 48 bits/pixel) by coding the difference between each pixel's value and a predicted value based on past pixels [8.2.9].
TIFF	Aldus	*Tagged Image File Format*. A flexible file format supporting a variety of image compression standards, including JPEG, JPEG-LS, JPEG-2000, JBIG2, and others.
Video		
AVS	MII	*Audio-Video Standard*. Similar to H.264 but uses exponential Golomb coding [8.2.2]. Developed in China.
HDV	Company consortium	*High Definition Video*. An extension of DV for HD television that uses MPEG-2 like compression, including temporal redundancy removal by prediction differencing [8.2.9].
M-JPEG	Various companies	*Motion JPEG*. A compression format in which each frame is compressed independently using JPEG.
Quick-Time	Apple Computer	A media container supporting DV, H.261, H.262, H.264, MPEG-1, MPEG-2, MPEG-4, and other video compression formats.
VC-1 WMV9	SMPTE Microsoft	The most used video format on the Internet. Adopted for HD and *Blu-ray* high-definition DVDs. It is similar to H.264/AVC, using an integer DCT with varying block sizes [8.2.8 and 8.2.9] and context dependent variable-length code tables [8.2.1]—but no predictions within frames.

TABLE 8.4 Popular image compression standards, file formats, and containers, not included in Table 8.3.

17.2.2 Statistical Encoding Methods

17.2.2.1 The Information Content of a Message

Before discussing statistical coding techniques, we consider the classical theory of information content. Suppose we have a memoryless source of messages that uses an *alphabet* $\{a_k\}, k = 0,1,...,K-1$. Here the a_k are the symbols of that alphabet. Suppose further that the probability of occurrence of each symbol is known and denoted as $P(a_k)$. In a message from a memoryless source, the ordering of the symbols in the message is unimportant; only their presence in the message matters.

Shannon [7,8] defined a measure of the information imparted by the occurrence of the symbol a_k in a message as

$$I(a_k) = -\log[P(a_k)] \tag{1}$$

This measure is satisfying because (a) the more unlikely a symbol is, the more information its presence contributes to the message, and (b) the information in a message is the sum of the information contributed by the symbols that comprise it. Notice that a symbol that always appears in every message (i.e., $P(a_i) = 1$) conveys no information ($I(a_i) = 0$). An example of such a noninformative symbol is the word *Dear* in the salutation of a letter.

The *entropy* of the message source, defined by

$$H = E\{I(a_k)\} = -\sum_{k=0}^{K-1} P(a_k)\log[P(a_k)] \tag{2}$$

specifies the average information content (per symbol) of the messages generated by the source. The entropy of a message source is nonnegative and takes on its maximum value when all symbols are equally likely. If we choose 2 as the base for the logarithm, the units of entropy are *bits per symbol.*

The redundancy remaining in a message after encoding it by a particular coding scheme is the difference between the average word length of the code and the entropy of the source; that is,

$$R = E\{L_w(a_k)\} - H \tag{3}$$

where $L_w(a_k)$ is the length (in bits, for binary coding) of the code word used to represent the symbol a_k. A coding scheme removes all redundancy if it produces an average word length that is equal to the entropy of the message source. This can be achieved if one can design the code so that the word lengths are

$$L_w(a_k) = -\log[P(a_k)] \tag{4}$$

and this formula represents a lower bound on average word length. For binary coding, this is possible only if the probabilities of all the symbols are negative integer powers of two (e.g., 0.5, 0.25, etc.).

$$\text{Entropy} = -\sum_{i=0}^{L-1} p_i \log_2(p_i) \quad \text{(in bits per pixel)}$$

$$\text{where } p_i = \text{the probability of the } i\text{th gray level} = \frac{n_k}{N^2}$$

$$n_k = \text{the total number of pixels with gray value } k$$

$$L = \text{the total number of gray levels (e.g., 256 for 8 bits)}$$

This measure provides us with a theoretical minimum for the average number of bits per pixel that could be used to encode the image. This number is theoretically optimal and can be used as a metric for judging the success of a coding scheme.

EXAMPLE 5-5

Let $L = 8$, meaning that there are 3 bits/pixel in the original image. Now, lets say that the number of pixels at each gray-level value is equal (they have the same probability), that is:

$$p_0 = p_1 = \ldots = p_7 = \frac{1}{8}$$

Now, we can calculate the entropy as follows:

$$\text{Entropy} = -\sum_{i=0}^{7} p_i \log_2(p_i) = -\sum_{i=0}^{7} \frac{1}{8}\log_2\left(\frac{1}{8}\right) = 3$$

This tells us that the theoretical minimum for lossless coding for this image is 3 bits/pixel. In other words, there is no code that will provide better results than the one currently used (called the natural code, since $000_2 = 0$, $001_2 = 1$, $010_2 = 2$, ..., $111_2 = 7$). This example illustrates that the image with the most random distribution of gray levels, a uniform distribution, has the highest entropy.

EXAMPLE 5-6

Let $L = 8$, thus we have a natural code with 3 bits/pixel in the original image. Now let's say that the entire image has a gray level of 2, so

$$p_2 = 1, \text{ and } p_0 = p_1 = p_3 = p_4 = p_5 = p_6 = p_7 = 0$$

And the entropy is

$$\text{Entropy} = -\sum_{i=0}^{7} p_i \log_2(p_i) = -(1)\log_2(1) + 0 + \ldots + 0 = 0$$

This tells us that the theoretical minimum for coding this image is 0 bits/pixel. Why is this? Because the gray-level value is known to be 2. To code the entire image, we need only one value. This is called the certain event; it has a probability of 1.

The two preceding examples illustrate the range of the entropy:

$$0 \leq \text{Entropy} \leq \log_2(L)$$

The examples also illustrate the information theory perspective on information and randomness. The more randomness that exists in an image, the more evenly distributed the gray levels, and the more bits per pixel are required to represent the data. This also correlates to information—more randomness implies each individual value is less likely, which means more information is contained in each pixel value, so we need more bits to code each pixel value. This also provides us with one of the key concepts in coding theory: we want to assign a fewer number of bits to code more likely events. Intuitively, this makes sense. Given an image to code, a minimum overall file size will be achieved if a smaller number of bits is used to code the most frequent gray levels.

The entropy measure also provides us with a metric to evaluate coder performance. We can measure the average number of bits per pixel (*L*ength) in a coder by the following:

$$L_{\text{ave}} = \sum_{i=0}^{L-1} l_i p_i$$

$$\begin{aligned} \text{where } l_i &= \text{length in bits of the code for } i\text{th gray level} \\ p_i &= \text{histogram-probability of } i\text{th gray level} \end{aligned}$$

This can then be compared to the entropy, which provides the theoretical minimum. The closer L_{ave} is to the entropy, the better the coder.

5.2.1 Huffman Coding

The Huffman code, developed by D. Huffman in 1952, is a minimum length code. This means that given the statistical distribution of the gray levels (the histogram), the Huffman algorithm will generate a code that is as close as possible to the minimum bound, the entropy. This method results in a *variable length code*, where the code words are of unequal length. For complex images, Huffman coding alone will typically reduce the file by 10 to 50% (1.1:1 to 1.5:1), but this ratio can be improved to 2:1 or 3:1 by preprocessing for irrelevant information removal.

The Huffman algorithm can be described in five steps:

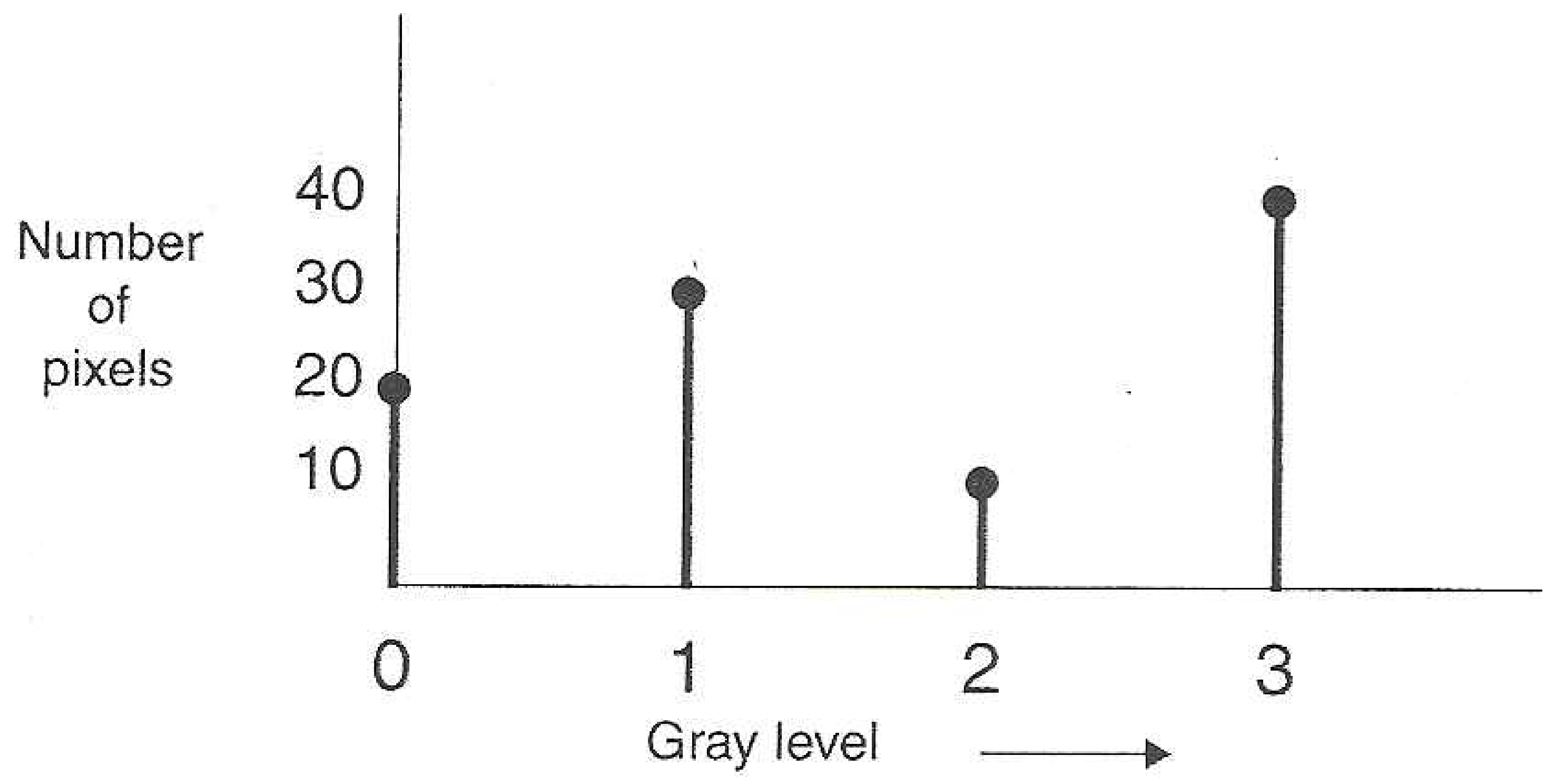
1. Find the gray-level probabilities for the image by finding the histogram.
2. Order the input probabilities (histogram magnitudes) from smallest to largest.
3. Combine the smallest two by addition.
4. GOTO step 2, until only two probabilities are left.
5. By working backward along the tree, generate code by alternating assignment of 0 and 1.

This procedure is best illustrated by example.

EXAMPLE 5-7

We have an image with 2 bits/pixel, giving four possible gray levels. The image is 10 rows by 10 columns. In Step 1 we find the histogram for the image. This is shown in Figure 5.2-1a, where we see that gray level 0 has 20 pixels, gray level 1 has 30 pixels, gray level 2 has 10 pixels, and

Figure 5.2-1 Huffman Coding Example



$$g_0 = \frac{20}{100} = 0.2$$

$$g_1 = \frac{30}{100} = 0.3$$

$$g_2 = \frac{10}{100} = 0.1$$

$$g_3 = \frac{40}{100} = 0.4$$

a. Step 1: Histogram.

$$\begin{tikzcd}
g_3 \arrow[r] & 0.4 \\
g_1 \arrow[r] & 0.3 \\
g_0 \arrow[r] & 0.2 \\
g_2 \arrow[r] & 0.1
\end{tikzcd}$$

b. Step 2: Order.

$$\begin{tikzcd}
0.4 \arrow[r] & 0.4 \\
0.3 \arrow[r] & 0.3 \\
0.2 \arrow[r] & 0.3 \\
0.1 \arrow[ru] &
\end{tikzcd}$$

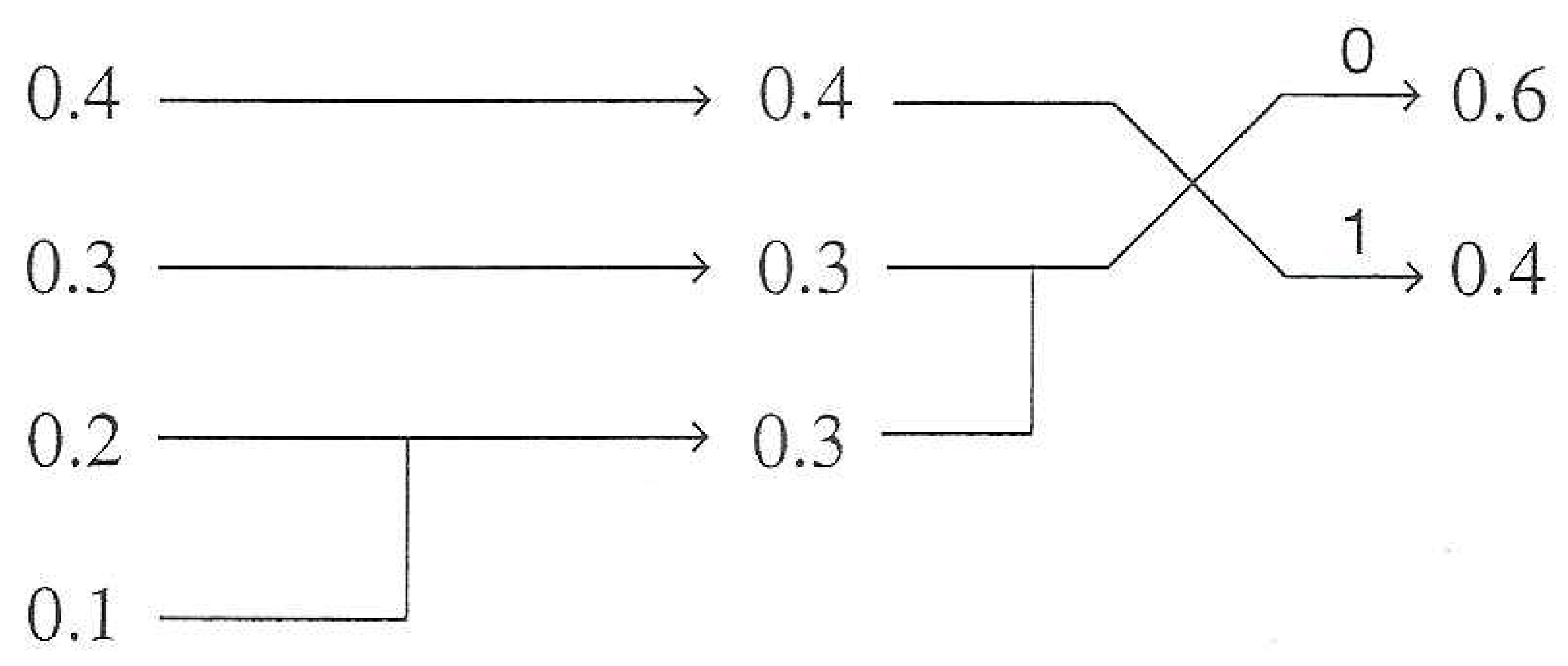
c. Step 3: Add.

$$\begin{tikzcd}
0.4 \arrow[r] & 0.4 \arrow[r] & 0.4 \\
0.3 \arrow[r] & 0.3 \arrow[r] & 0.6 \\
0.2 \arrow[r] & 0.3 \arrow[ru] & \\
0.1 \arrow[ru] & &
\end{tikzcd}$$

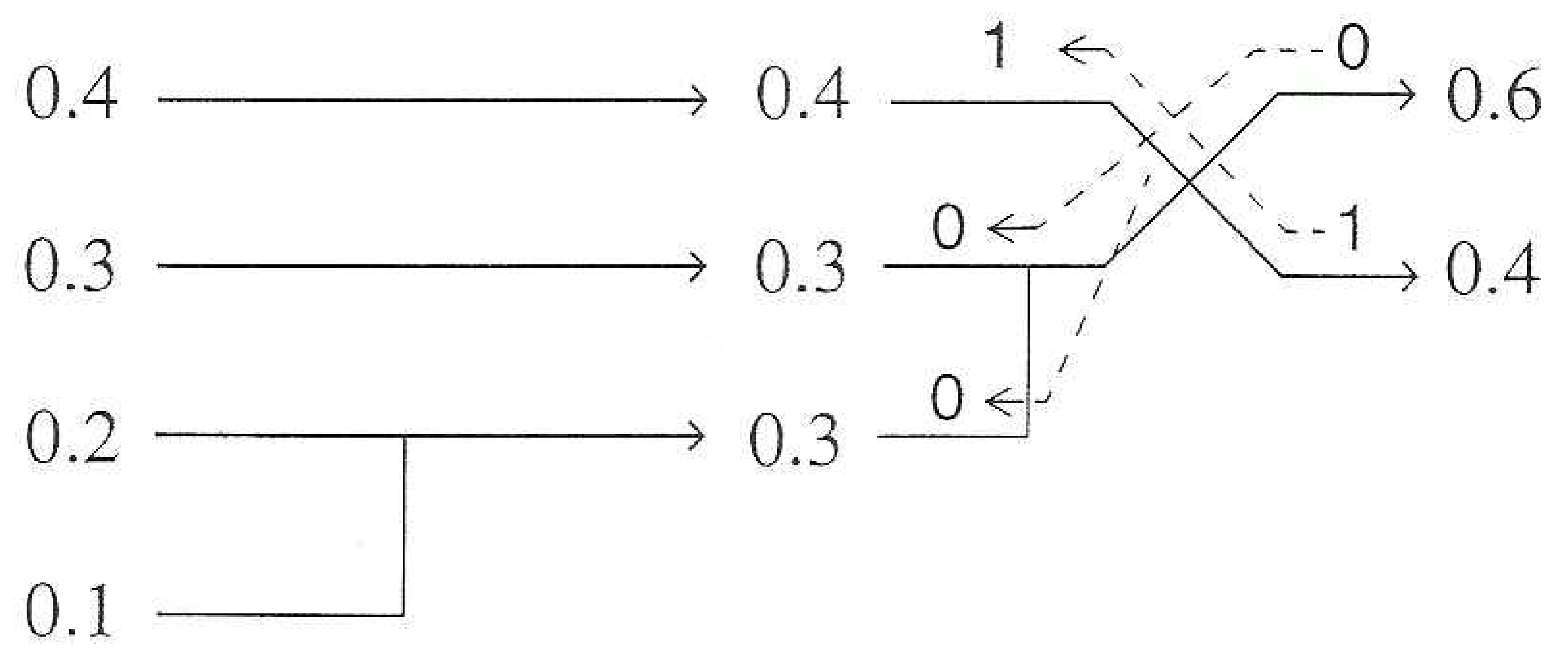
$$\begin{tikzcd}
0.4 \arrow[r] & 0.4 \arrow[rd] & 0.6 \\
0.3 \arrow[r] & 0.3 \arrow[ru] & 0.4 \\
0.2 \arrow[r] & 0.3 \arrow[ruu] & \\
0.1 \arrow[ru] & &
\end{tikzcd}$$

d. Step 4: Reorder and add until only two values remain.

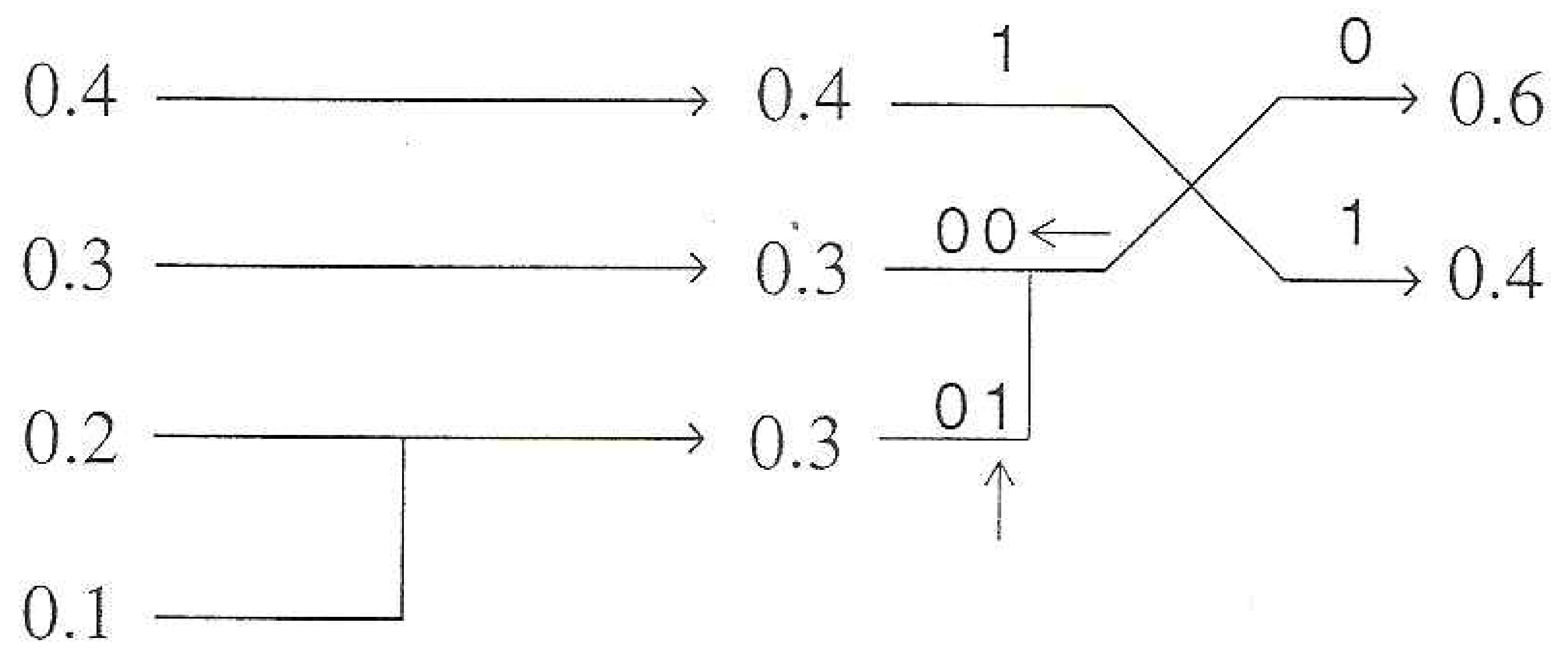
Figure 5.2-2 Huffman Coding Example, Step 5



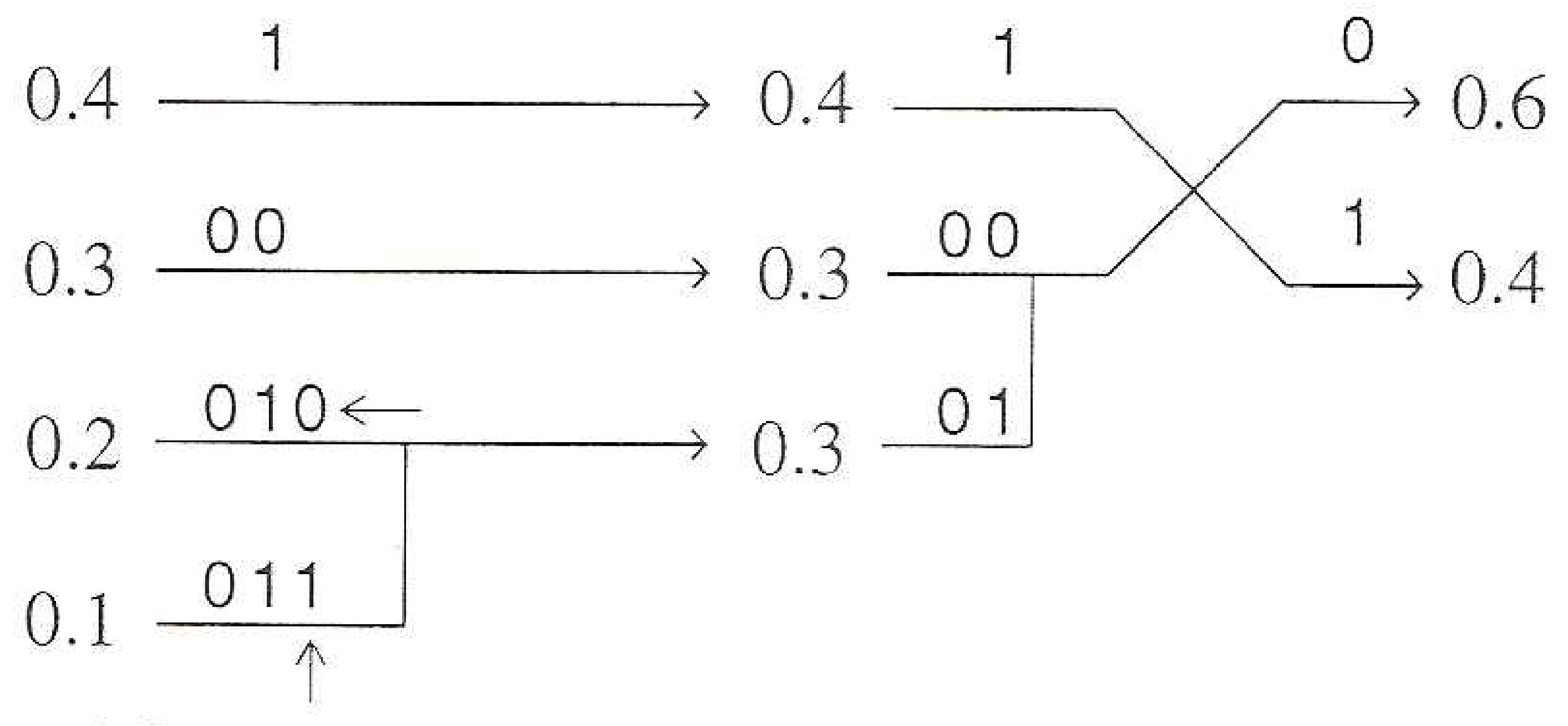
a. Assign 0 and 1 to the rightmost probabilities.



b. Bring 0 and 1 back along the tree.



c. Append 0 and 1 to previously added branches.



d. Repeat the process until the original branch is labeled.

gray level 3 has 40 pixels with the value. These are converted into probabilities by normalizing to the total number of pixels in the image. Next, in step 2, the probabilities are ordered as in Figure 5.2-1b. For step 3 we combine the smallest two by addition. Step 4 repeats steps 2 and 3, where we reorder (if necessary) and add the two smallest probabilities as in Figure 5.2-1d. This step is repeated until only two values remain. Because we have only two left in our example, we can continue to step 5 where the actual code assignment is made. The code assignment is shown in Figure 5.2-2. We start on the right-hand side of this tree and assign 0's and 1's, working our way back to the original probabilities. Figure 5.2-2a shows the first assignment of 0 and 1. A 0 is assigned to the 0.6 branch, and a 1, to the 0.4 branch. In Figure 5.2-2b the assigned 0 and 1 are brought back along the tree, and wherever a branch occurs the code is put on both branches. Now (Figure 5.2-2c) we assign the 0 and 1 to the branches labeled 0.3, appending to the existing code. Finally (Figure 5.2-2d), the codes are brought back one more level, and where the branch splits another assignment of 0 and 1 occurs (at the 0.1 and 0.2 branch). Now we have the Huffman code for this image as shown in Table 5.2-1.

Table 5.2-1 Huffman Code

Original Gray Level (Natural Code)	Probability	Huffman code
g_0: 00_2	0.2	010_2
g_1: 01_2	0.3	00_2
g_2: 10_2	0.2	011_2
g_3: 11_2	0.4	1_2

Note that two of the gray levels now have 3 bits assigned to represent them, but one gray level only has 1 bit assigned to represent it. The gray level represented by 1 bit, g_3, is the most likely to occur (40% of the time) and thus has the *least information in the information theoretic sense*. Remember that we learned from information theory that symbols with less information require fewer bits to represent them. The original image had an average of 2 bits/pixel, let us examine the entropy in bits per pixel and average bit length for the Huffman coded image file.

EXAMPLE 5 - 8

$$\text{Entropy} = -\sum_{i=0}^{3} p_i \log_2(p_i)$$

$$= -[(0.2)\log_2(0.2) + (0.3)\log_2(0.3) + (0.1)\log_2(0.1) + (0.4)\log_2(0.4)$$

$$\approx 1.846 \text{ bits/pixel}$$

(Note: $\log_2(x)$ can be found by taking $\log_{10}(x)$ and multiplying by 3.322)

$$L_{\text{ave}} = \sum_{i=0}^{L-1} l_i p_i$$

$$= 3(0.2) + 2(0.3) + 3(0.1) + 1(0.4)$$

$$= 1.9 \text{ bits/pixel} \quad \text{(Average length with Huffman code)}$$

In the example, we observe a 2.0:1.9 compression, which is about a 1.05 compression ratio, providing about 5% compression. From the example we can see that the Huffman code is highly dependent on the histogram, so any preprocessing to the histogram may help the compression ratio.

Huffman Coding

1. Optimal code for set of symbols **subject to the constraint that the symbols be coded one at a time**
2. coding / decoding is done by a lookup table instantaneously, uniquely and decodeable, block code
   where
   instantaneous = each code word can be decoded by itself
   block = each source symbol yields a fixed sequence of code
   unique = only one way
3. | | |
   |---|---|
   | J source symbols | 256 gray levels |
   | J-2 source reductions | 254 |
   | J-2 code assignments | 254 |

Alternatives are cheaper but not optimal.

LZW uses interpixel redundancies

5.2.2 Run-Length Coding

The image is an 8×8 binary image, which requires 3 bits for each run-length coded word. In the actual image file are stored 1's and 0's, although upon display the 1's become 255 (white) and the 0's are 0 (black). To apply RLC to this image, using horizontal RLC:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The RLC numbers are:

First row: 8

Second row: 0, 4, 4

Third row: 1, 2, 5

Fourth row: 1, 5, 2

Fifth row: 1, 3, 2, 1, 1

Sixth row: 2, 1, 2, 2, 1

Seventh row: 0, 4, 1, 1, 2

Eighth row: 8

Note that in the second and seventh rows, the first RLC number is 0, since we are using the c vention that the first number corresponds to the number of zeros in a run.

EXAMPLE 5-13

Given the following 8 × 8, 4-bit image:

$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 0 & 0 & 0 & 10 & 10 & 10 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 5 & 10 & 10 & 9 & 9 & 10 \\ 5 & 5 & 5 & 4 & 4 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The corresponding gray-level pairs are as follows:

First row: 10,8

Second row: 10,5 12,3

Third row: 10,5 12,3

Fourth row: 0,3 10,3 0,3

Fifth row: 5,3 0,5

Sixth row: 5,3 10,2 9,2 10,1

Seventh row: 5,3 4,3 0,2

Eighth row: 0,8

These numbers are then stored in the RLC compressed file as:

10, 8, 10, 5, 12, 3, 10, 5, 12, 3, 0, 3, 10, 3, 0, 3, 5, 3, 0, 5, 5, 3, 10, 2, 9, 2, 10, 1, 5, 3, 4, 3, 0, 2, 0, 8

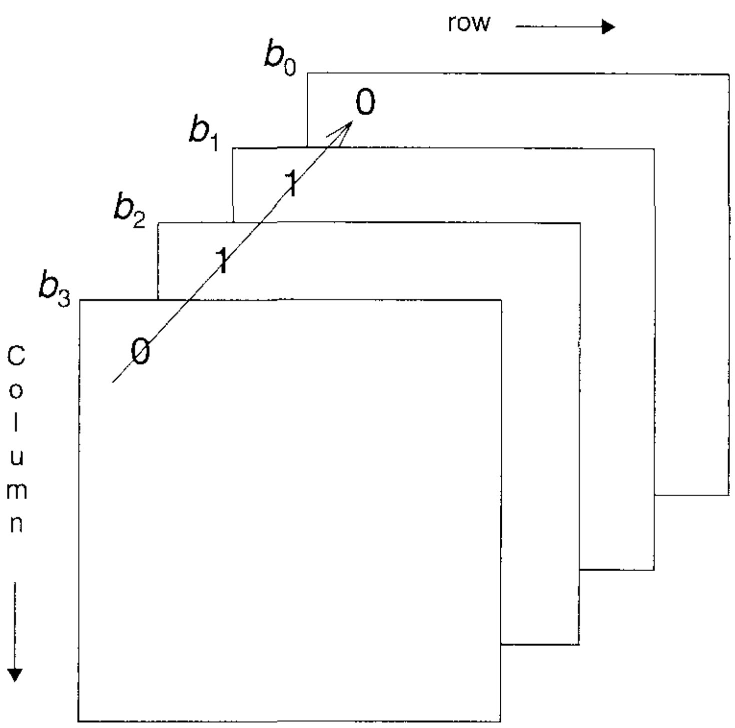
The decompression process requires the number of pixels in a row, and the type of coding used.

Standards for RLC have been defined by the International Telecommunications Union-Radio (ITU-R, previously CCIR). These standards, initially defined for use with FAX transmissions, have become popular for binary image compression. They use horizontal RLC but postprocess the resulting RLC with a Huffman encoding scheme. Newer versions of this standard also use a two-dimensional technique where the current line is coded based on a previous line. This additional processing helps to reduce the file size. These coding methods provide compression ratios of about 15 to 20 for typical documents.

2-3 Bit-Plane Run-Length Coding

b_3	b_2	b_1	b_0
0	0	0	0
0	0	0	1
0	0	1	0
. . .	. . .	. . .	. . .
0	1	1	0
1	1	1	1

a. 4 bits/pixel designation.



b. bit-planes $\rightarrow b_0^3\ b_0^2\ b_0^1\ b_0^0$.

Another way to extend basic RLC to gray-level images is to include the gray level of a particular run as part of the code. Here, instead of a single value for a run, two parameters are used to characterize the run. The pair (G, L) correspond to the gray-level value G and the run-length L. This technique is only effective with images containing a small number of gray levels.

This basic method can be extended to gray-level images by using a techniqu called bit-plane RLC. *Bit-plane RLC* works by applying basic RLC to each bit plar independently. In Figure 5.2-3 the idea of bit planes is illustrated. For each binaı digit in the gray-level value, an image plane is created, and this image plane (string of 0's and 1's) is then coded using RLC. Typical compression ratios of 0.5 to 1 are achieved with complex 8-bit monochrome images; so, without further processin this is not a good compression technique for complex images. Bit-plane RLC is mo useful for simple images, such as graphics files, where much higher compressic ratios are achieved. The compression results using this method can be improved b preprocessing to reduce the number of gray levels, but then the compression is *n* *lossless.* In order for this method to be effective, the reduced image data (in *natur* *code*) needs to be mapped to a *Gray code* (named after Frank Gray), where adjacer numbers differ in only one bit. Because adjacent pixel values are highly correlate adjacent pixel values tend to be relatively close in gray-level value, and this can b problematic for RLC.

EXAMPLE 5-12

In Figure 5.2-4 is shown the 4-bit Gray code and the natural binary code. The Gray code, by d inition, only has one bit changing in adjacent codes. However, in, for example, the 7 to 8 tran tion with the natural code, all four bits change:

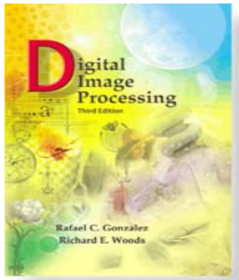
Natural Code	*Gray Code*
0 1 1 1	0 1 0 0
↓ ↓ ↓ ↓	↓ ↓ ↓ ↓
1 0 0 0	1 1 0 0

When a situation such as this example occurs, each bit plane experiences a trai sition, which adds a code for the run in each bit plane. However, with the Gray cod only one bit plane experiences the transition, so it only adds one extra code word.

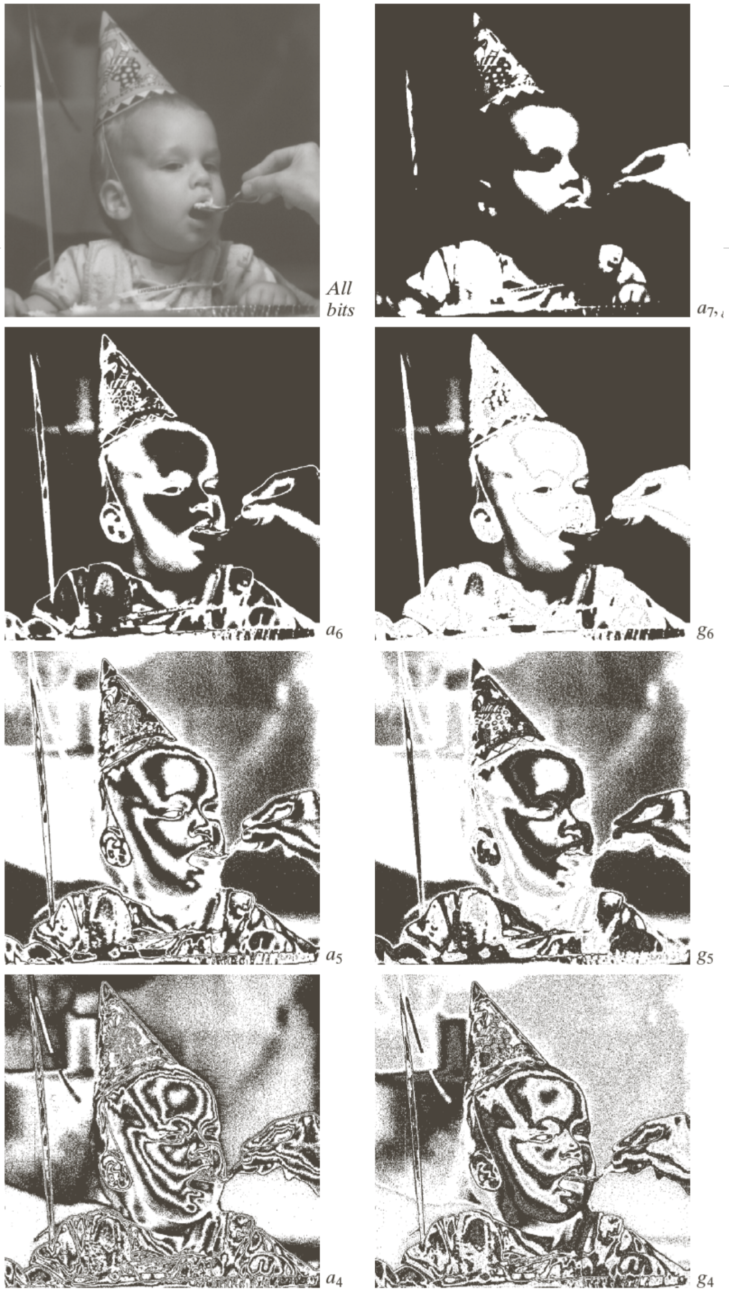
Figure 5.2-4 Gray Code

Decimal	4-bit Natural Code	4-bit Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

a. Gray code versus natural code.

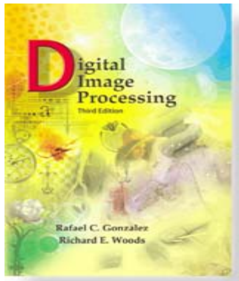


Digital Image Processing, 3rd ed.



a	b
c	d
e	f
g	h

FIGURE 8.19 (a) A 256-bit monochrome image. (b)–(h) The four most significant binary and Gray-coded bit planes of the image in (a).



Digital Image Processing, 3rd ed.

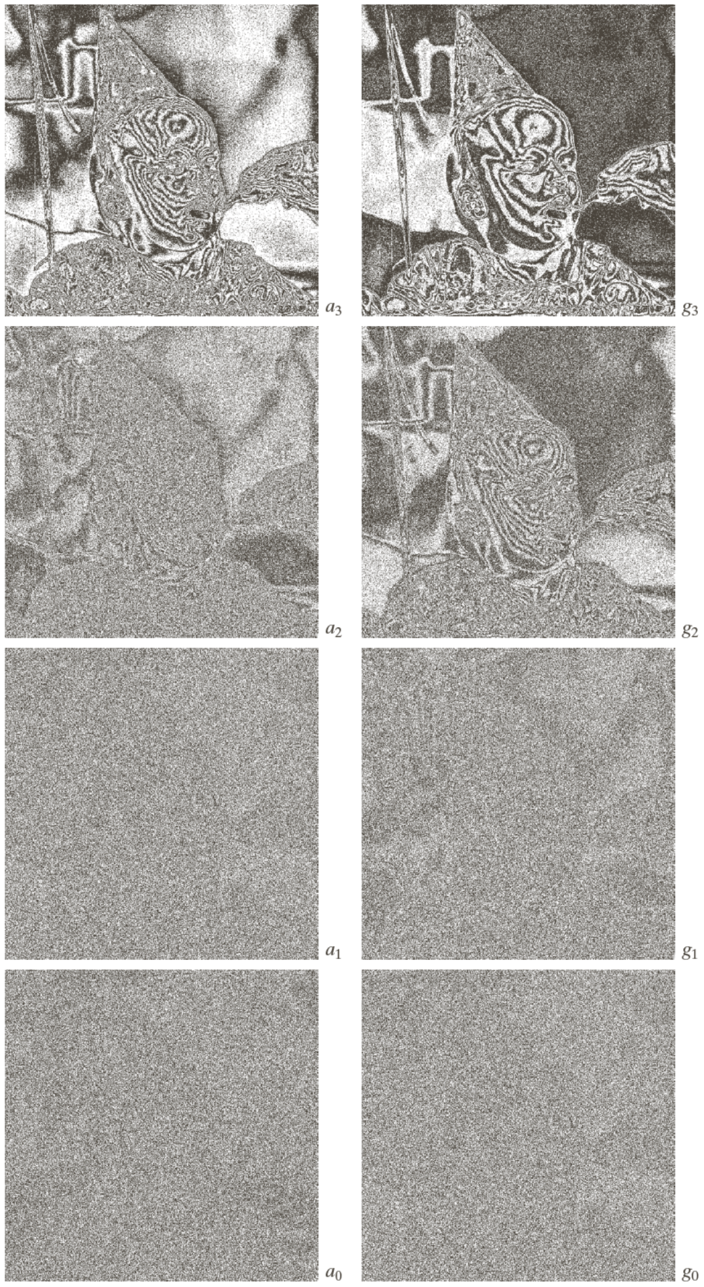
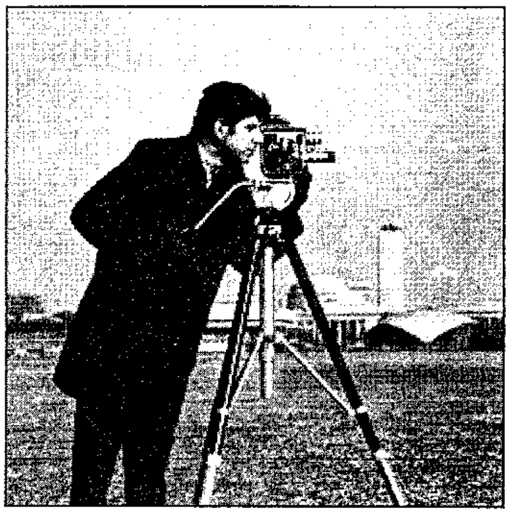
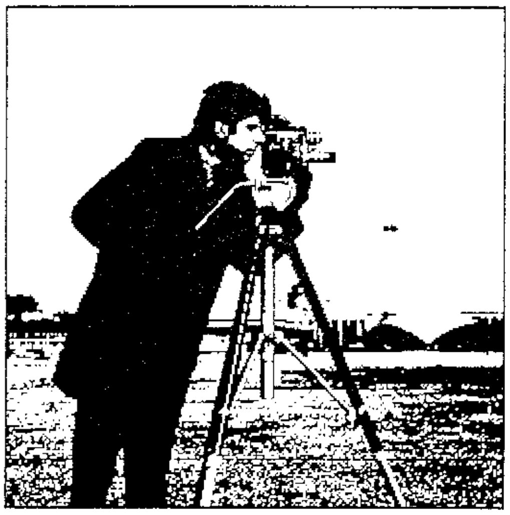
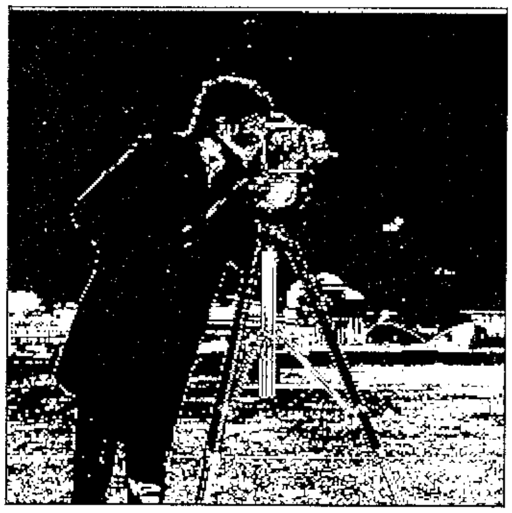


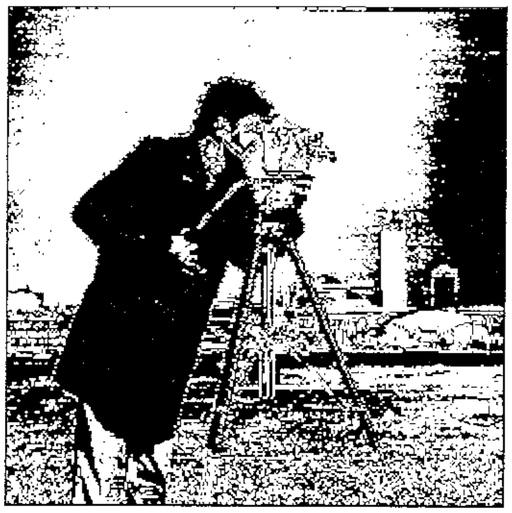
FIGURE 8.20 (a)–(h) The four least significant binary (left column) and Gray-coded (right column) bit planes of the image in Fig. 8.19(a).

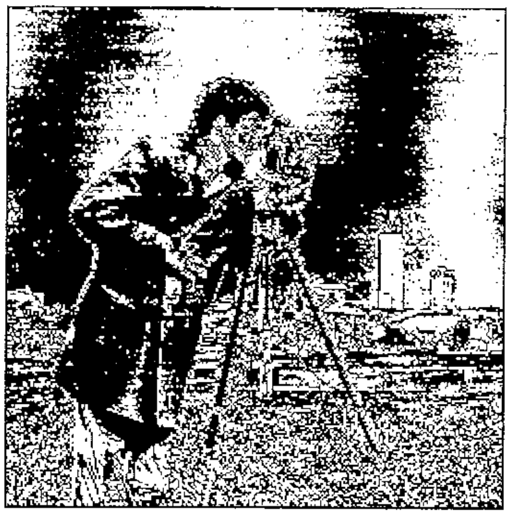
Figure 5.1-5 Bit-Plane Images


a. Original, 8-bit image.


b. Bit-plane 7, the most significant bit.


c. Bit-plane 6.


d. Bit-plane 5.


e. Bit-plane 4.

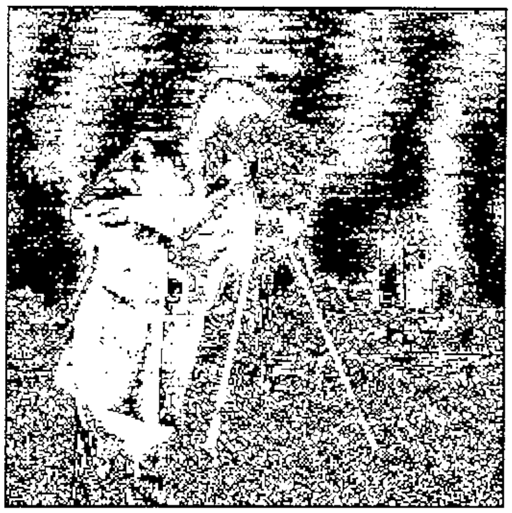
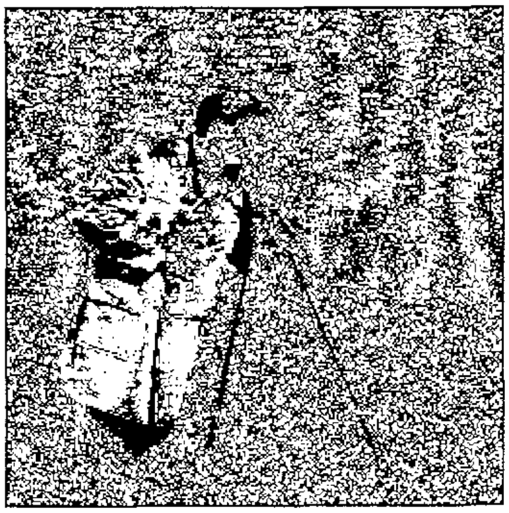
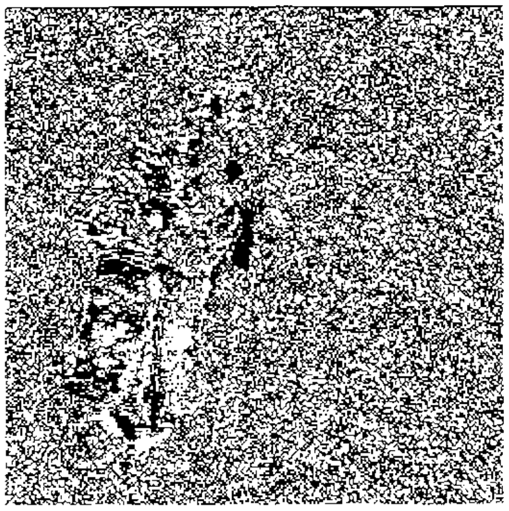
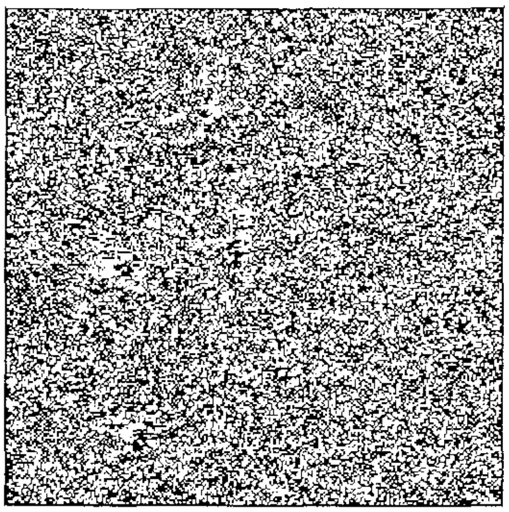

f. Bit-plane 3.

Figure 5.1-5 (Continued)


g. Bit-plane 2.


h. Bit-plane 1.


i. Bit-plane 0, the least significant bit.

coding. Secondly, the principal components transform (PCT, see Sections 1.7.3 and 2.4.3) can be used, which provides a theoretically optimal decorrelation. Also, the spectral domain is used for image compression, so this first stage may include mapping into the frequency or sequency domain. These methods are all *reversible*, that is information preserving, although all mapping methods are not reversible. The concept of reversibility is important to a compression method.

Depending on the mapping equation used, quantization may be necessary to convert the data into digital form. There are two ways to do this—uniform quantization or nonuniform quantization. In *uniform quantization* all the quanta, or subdivisions into which the range is divided, are of equal width. In *nonuniform quantization* these quantization bins are not all of equal width (see Figures 2.2-15 and 2.2-16). Often, nonuniform quantization bins are designed to take advantage of the response of the human visual system. For example, very high brightness levels appear the same—white—so wider quantization bins may be used over this range. In the spectral domain, the higher frequencies may also be quantized with wider bins because we are

2.7.1 Group 3 Fax

CCITT Draft Recommendation T.4 describes the standardization of Group 3 facsimile apparatus for document transmission (CCITT 1980). The standard specifies the scanning track, the dimensions of the apparatus, the transmission time, and the coding scheme. We will dispense quickly with the first three of these and then treat the coding scheme in complete detail.

Classic raster scanning is employed at both the transmitter and the receiver; that is, considering the document to be in a vertical plane facing the viewer, scanning proceeds from left to right along each line, starting at the top of the page and working to the bottom without interlacing. Each line has 1728 picture elements (pels). Each pel will be either black or white in any specific document. The standard scanning line width is 215 mm. In the vertical dimension the standard definition is 3.85 lines/mm; an optional mode with double vertical resolution (7.7 lines/mm) also is supported. Group 3 apparatus must accept documents of ISO A4 size or smaller. If one desires to send wider documents, horizontal scanning density should be reduced accordingly.

The Group 3 standard recommends a 20-ms *minimum* transmission time for the sum of the data bits, fill bits, and end-of-line (EOL) symbol that constitute the coded representation of any scan line. This requirement is imposed by mechanical limitations on the rate of advance of paper through the transmitting and receiving devices and the desire not to let the transmitter get too many lines ahead of the receiver. Because digital memory has become considerably cheaper than when the Group 3 standard originally was promulgated, today's fax machines often scan documents many times faster than the recommended 20 ms/line and store the results internally if transmission has to proceed at a lower rate. This saves the time of persons and machines at the transmitting station tasked with document handling. The Group 3 standard also provides recognized minimum transmission time options of 10 ms/line with a mandatory fallback to 20 ms if requested by the receiving machine, and of 5 ms/line with mandatory fallbacks to 10 ms and 20 ms. Finally, an option for a 40-ms/line minimum transmission time is supported, though modern equipment eschews it. Identification of the minimum transmission time of a scanning line is made during the premessage portion of the T.30 control procedure, sometimes referred to as Phase B of the call setup.

At the other extreme, no line transmission time may exceed 5 seconds. This requirement is imposed to retain synchronism and to prevent call dropping via inadvertent activation of disconnect-sensing circuitry.

G3 Coding Scheme

Group 3 fax machines use a one-dimensional runlength coding scheme. Here, "one-dimensional" refers to the fact that each successive horizontal line is coded independently of all the lines that precede and follow it, and "runlength" refers to the fact that each line is parsed into alternating runs of same-color pels—white run, black run, white run, black run, and so on.

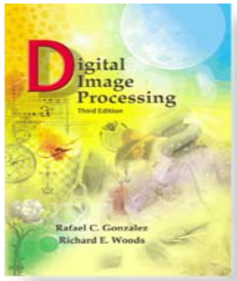
The first run on each line is assumed to be white; if it is not white, the codeword for a white run of length 0 is sent, namely, 00110101. The sum of the lengths of all the runs on a line must be 1728. The end of each line is signaled by the EOL codeword, 000000000001. We shall see shortly that this EOL pattern (11 zeros followed by a one) can never occur anywhere else within a valid encoding other than at the end of a line. This keeps the transmitter and receiver in line synch in the absence of channel errors and allows for rapid line resynch in the event of a channel error burst that either inserts an undesired EOL or corrupts an intended one. Also, the EOL signal is sent prior to the first line of each page; that is, page breaks are signified by two successive EOLs. Accordingly, any line insertion or deletion error that might occur will not propagate beyond its page.

White runs of lengths 0 through 63 are encoded via the Terminating White Code of Table 2.5. White runs of length 64 or greater are coded by concatenating a word from the Make Up White Code of Table 2.6 with a Terminating White Code word from Table 2.5. For example, a white run of length $1362 = 64 \cdot 21 + 18 = 1344 + 18$ is encoded by concatenating the codeword 011011010 for 1344 from Table 2.6 with the codeword 0100111 for 18 from Table 2.5.

Note that both the Terminating White Code and the Make Up White Code are prefix codes—no short codeword is the prefix of a longer codeword. More importantly, note that no word in the Make Up White Code is a prefix of a word in the Terminating White Code and, conversely, no word in the Terminating White Code is a prefix of a word in the Make Up White Code. In other words, the *union* of the Terminating White Code and the Make Up White Code is a prefix code. This permits Group 3 decoders to determine unambiguously whether a terminating code (TC) or a make up code (MUC) word has been sent.

Note that greater compression could be achieved if a prefix code different from that of Table 2.5 were used to encode the residue modulo 64 of runlengths that are 64 or longer. The reasons are that, since this code would be invoked if and only if a MUC word has just been sent, its words need not be restricted not to be prefixes of any of the TC or MUC words, thereby allowing its word lengths to match better the statistics of runlengths of 64 or more calculated modulo 64. CCITT Study Group XIV chose not to do this, probably because the improvement it afforded was insufficient to justify having to store three rather than two tables for runs of a given color.

Black runs are similarly encoded by the TC and MUC words shown in Tables 2.7 and 2.8. Words from the black codes are allowed to be prefixes of words from the white codes, and vice versa. No ambiguity results because run colors always alternate and every line starts with a white run. Although there are no black runs of length 0, a black TC word for length 0 is needed because



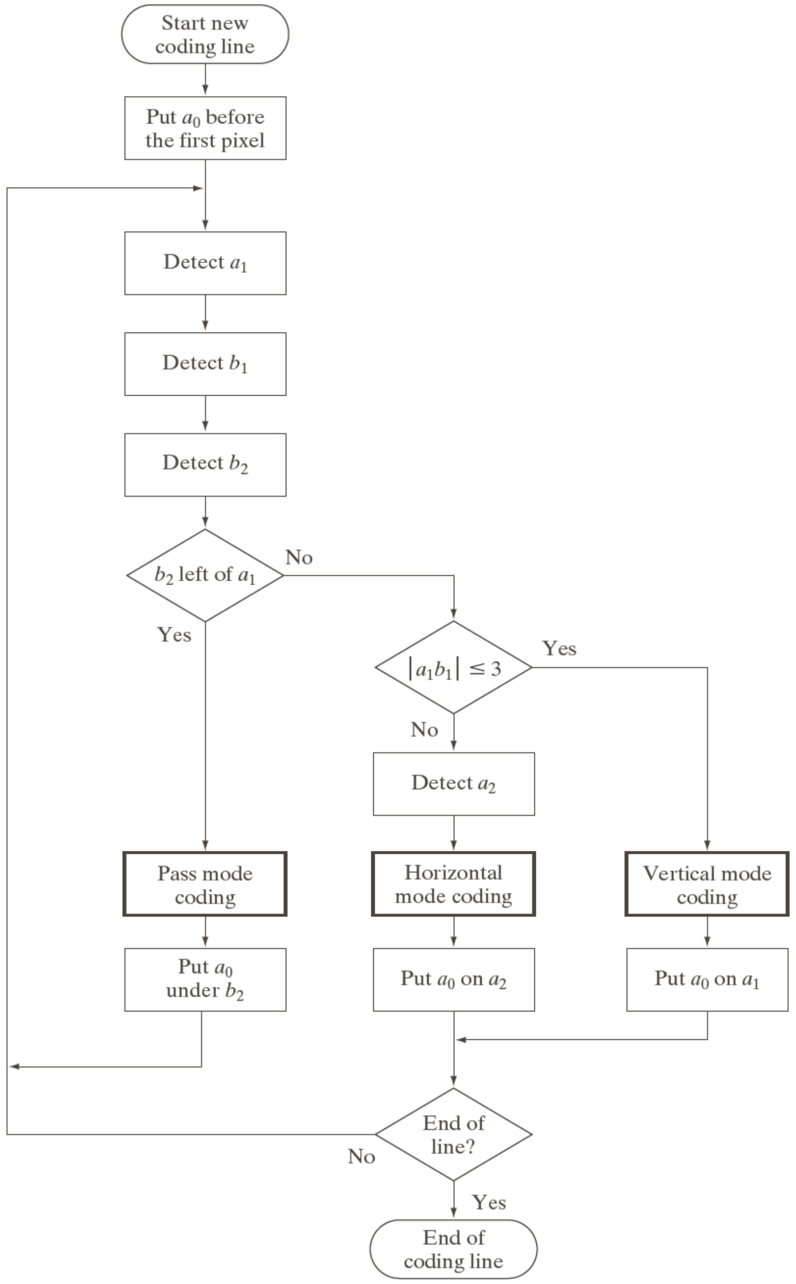


FIGURE 8.14 CCITT 2-D READ coding procedure. The notation $|a_1b_1|$ denotes the absolute value of the distance between changing elements a_1 and b_1.

5.2.3 Lempel-Ziv-Welch Coding

The *Lempel-Ziv-Welch* (LZW) coding algorithm works by coding strings of data. For images, these strings of data correspond to sequences of pixel values. It works by creating a string table that contains the strings and their corresponding codes. The string table is updated as the file is read, with new codes being inserted whenever a new string is encountered. If a string is encountered that is already in the table, the corresponding code for that string is put into the compressed file.

LZW coding uses code words with more bits than the original data. For example, with 8-bit image data, an LZW coding method could employ 10-bit words. The corresponding string table would then have $2^{10} = 1{,}024$ entries. This table consists of the original 256 entries, corresponding to the original 8-bit data, and allows 768 other entries for string codes. The string codes are assigned during the compression process, but the actual string table is not stored with the compressed data. During decompression the information in the string table is extracted from the compressed data itself.

For the GIF (and TIFF) image file format the LZW algorithm is specified, but there has been some controversy over this because the algorithm is patented (by Unisys Corporation under patent #4,558,302). Because these image formats are widely used, other methods similar in nature to the LZW algorithm have been developed to be used with these, or similar, image file formats. Similar versions of this algorithm include the *adaptive Lempel-Ziv*, used in the UNIX compress function, and the *Lempel-Ziv 77* algorithm used in the UNIX gzip function.

LZ

- $\left.\begin{array}{c}\text{Abraham Lempel}\\ \text{Jacob Ziv}\end{array}\right\}$ 1977, 1978

  gzip, zip, pkzip, winzip
- Welch LZW - 1984

  Unix Compress, GIF, TIFF
- Yakou LZY - 1992
- parse sequences into strings/phrases
- compress code to represent phrases
- If stationary asymptotically stable $\Longrightarrow$ entropy rate H
- Redundancy = compression - enthalpy

Example (simplified)

I am dumb and because I am dumb I can’t even tell you that I am dumb

$$\Downarrow$$

$1 and because $1 I can’t even tell you that $1
$1=(I am dumb)

LZ77

0001011000001010010010001 0011

0 00 1 011 0000 | 01010 0100 100

1. search for match from beginning
2. best match is 0101
3. add one more bit

need to store beginning + length

000 1 0 11 0 0 000101 001001 0001 0011

LZ78

0 00 1 01 10 000 010 ⁞ 100 1001 0001

1. phrase that achieves longest match with beginning of unparsed data

2. next source beyond end of maximal patch

i.e. remainder after ⁞ 100100100....

best match among beginning is 10

Hence add zero to previous match

0, 001, 011, 0000, 01010, 01001000, 10011,...

Tail-biting LZ77 uses only seven phrases for our data string as opposed to the nine phrases used by ordinary LZ77. Tail-biting is invoked when parsing the second and sixth phrases. The second phrase is formed by copying a phrase of length 2 starting at the lone 0 then in the window and then adding a terminal 1; we slide the first digit copied (a 0 in this case) into the window and then immediately copy it to get the second digit of the current phrase. Similarly, the sixth phrase is formed by copying a string of length 7 starting at the third position before the end of the window and then adding a terminal 0; a four-symbol tail is bitten in this case. (It can be argued that the tail actually grasps the head rather than the head biting the tail.) The spot to point to at which to begin the copying need not be unique; for example, there are two different places in the window to which we could point to start copying the 1001 portion of the seventh phrase.

The Lempel-Ziv-Welch (LZW) algorithm is a variant of LZ78. LZW is the core of the widely deployed Unix Compress utility. It parses our sample sequence as follows:

0, 00, 1, 0, 1, 10, 00, 001, 01, 0010, 010, 00100, 11···, ...

Note that the phrases of an LZW parsing are not unique and tend to be shorter than those of the other parsings we have considered so far. The advantages that account for LZW's popularity will be detailed in Section 3.3 when we analyze system performance.

Finally, the Lempel-Ziv-Yokoo (LZY) algorithm (Yokoo 1992; Kiyohara and Kawabata 1996) parses our sample sequence as follows:

0, 00, 1, 01, 10, 000, 01, 010, 0100, 100, 01001, 1...

LZY uses only 11 nonunique phrases to parse one more symbol of our sample sequence than LZW is able to parse using 13 phrases. We argue in Section 3.3 that LZY provides a desirable mix of properties from LZ77 (with tail-biting), LZ78, and LZW. Since LZY usually outperforms LZW in practical examples, it would have provided a better foundation for Unix Compress than does LZW. However, LZW already was too firmly entrenched to be displaced and also is computationally less intensive than LZY.

3.3 LZ Compression

The easiest way to understand how an LZ algorithm compresses is to apply it step by step to an example. We shall do this now for the 29-symbol sequence

TABLE 3.4 Compression Performance of LZW (Unix Compress)

Data Type	Compression Ratio
English text	1.8
Cobol files	2–6
Floating-point arrays	1.0
Formatted scientific data	2.1
System log data	2.6
Program source code	2.3
Object code	1.5

3.3.5 Comparative Performance of LZ77, LZW, and LZ

Table 3.4 (from Welch 1984) shows the compression perfor
by LZW for some common types of computer files. Since the
Unix Compress utility is, in essence, Welch's implementation c
interpret these results as relevant for Unix Compress. In each
alphabet consists of 8-bit bytes.

Here are some comments on these results:

1. The results for English text are fairly good. Experimentat
that, if people fluent in English are provided with trut
binary questions they are allowed to ask about a typic
then between 1.3 and 2.5 questions per letter of text su
deduce the sample English text losslessly, depending on w
distinguish between upper and lower case letters, use pu
and/or allow numbers. Hence, it should be possible to co
uses 8-bit byte representations of English characters (e.g.,
text file) by a factor of 8/2.5 = 3.2, but LZW achieved c
be appreciated, however, that LZW does not know that
English text (as opposed to French, Spanish, or Fortran).
the rules of English grammar, possesses no a priori know
vocabulary and spelling, and most of all possesses no sem
or life experience. Nonetheless, it is true that if an LZ
allowed to keep growing its data structure were given a l
(which it never will be given in practice), it would "learn"
Asymptotically, it would achieve compression comparable
humans and better than that of average humans!
2. Apparently, not all Cobol files are comparably redundant.

BLE .5 TeXfile and PSfile Performance of Unix Compress

File Name	Input Length	Output Length	Compression Ratio
file1.tex	75,053	33,797	2.22
file2.tex	64,957	30,786	2.16
fat.tex	107,697	49,533	2.17
file1.ps	209,746	80,413	2.61
file2.ps	315,921	119,683	2.64
file3.ps	244,686	94,975	2.58
file4.ps	222,201	185,827	2.59
cat.ps	992,554	378,233	2.62
fat.tex	107,697	49,533	2.17
fat.ps	693,713	230,491	3.01

to fat.tex. Note that fat.ps is a factor of 6.44 bigger than fat.tex. This is representative; the expansion factors for the conversion from TeX to PS of some 50 files were observed to run from just above 2 to as much as 12.

The comparative verbosity of PSfiles accounts for why it is more desirable to communicate a technical paper to someone as a TeXfile than as a PSfile. Nonetheless, PSfiles often get sent by email or by ftp because they are easier for recipients (especially non-LaTeX recipients) to print. Also, they are less vulnerable to incompatibilities between the sender's and the receiver's versions of LaTeX (what style files and fonts are mounted, storage and floating of encapsulated PostScript figures, and such).

When a PSfile is sent via a modem that uses data compression, it might seem that its verbosity shouldn't cause a problem. Indeed, the data compression routine in the modem (an LZ77 variant in most of today's commercial products) not only should eliminate the factor of 6.44 expansion that occurred when fat.ps was produced from fat.tex but should go on to achieve further compression by eliminating redundancy lurking in fat.tex. But Table 3.5 shows that this did not happen! (True, Unix Compress's version of LZW, not LZ77, was used to get the figures in Table 3.5. However, LZ77 achieves only moderately better results; see Table 3.6.) Rather, fat.ps was compressed only to 230,491 bytes, which is still 2.14 times as big as fat.tex and $2.14 \cdot 2.17 = 4.65$ times bigger than the compressed version of fat.tex. Although Compress saved us almost half a megabyte in the storing and/or transmitting of fat.ps, it might seem that

E Performance Comparisons of LZW, LZY, LZYEP, and LZ77

File	Bytes	compress LZW	LZY	LZYEP	gzip LZ77
A	1250	1.054	1.042	0.993	0.774
B	46,976	0.480	0.439	0.425	0.360
C	179,021	0.394	0.361	0.351	0.301

applied to `fat.tex`). Here are two reasons why our supposedly "universal" LZW algorithm did not accomplish this; no doubt, there are other reasons as well.

Reason 1 is that neither a TeX nor a PostScript file fully describes the desired hard copy unassisted. Rather, they refer to extensive code stored in the `/tex` directory in one case and in the PS-compatible printer memory in the other; the LZ compression algorithm in the above examples never sees this auxiliary information. In the limit of an extremely long file, this stored code would become a negligible consideration because there is only a finite amount of it. However, in practice we do not operate in this limit; indeed, the stored code is several times larger in the above examples than are the TeXfiles and PSfiles that LZ is being asked to compress. Although most of what's stored in the TeX utility and most of the bitmaps stored in the PostScript cartridge are not consulted when processing any particular target file, their presence makes it difficult to assess the extent to which PSfiles truly deserve to be accused of being verbose. To put it another way, TeX usually consumes many more megabytes of computer disk than PostScript does of printer memory, so a PSfile "deserves" to be longer than the corresponding TeXfile.

Reason 2 is that universality is an asymptotic property. To possess true universality, LZW must continue to grow the tree, or stack, of past phrases that can be copied, but practical LZW implementations do not do this for a variety of good reasons. In this regard, consider the entry for `cat.ps`, which was produced by concatenating `file1.ps` through `file4.ps` and thus has a length equal exactly to the sum of their lengths. Even though this file is almost a megabyte, the compression ratio achieved for it effectively equals the weighted mixtures of those achieved for the files that were concatenated to produce it. This is because each of the component files is already several times longer than the maximum number of entries Unix Compress is willing to hold in its LZW data structure. If we were to keep growing the data structure, slightly better performance would result. However, even with a growing data structure, a megabyte file is not nearly long enough to make the leading term in the redundancy analysis of (LZ) universal algorithms negligible (see Section 3.3.6).

10.1 QUANTIZATION

10.1.1 Scalar Quantization

Let f denote a continuous scalar quantity that may represent a pixel intensity, transform coefficient, or image model parameter. To represent f with a finite number of bits, only a finite number of reconstruction or quantization levels can be used. We will assume that a total of L levels are used to represent f. The process of assigning a specific f to one of L levels is called amplitude quantization, or quantization for short. If each scalar is quantized independently, the procedure is called *scalar quantization*. If two or more scalars are quantized jointly, the procedure is called *vector quantization* or *block quantization*. Vector quantization is discussed in Section 10.1.2.

Let $\hat{f}$ denote an f that has been quantized. We can express $\hat{f}$ as

$$\hat{f} = Q(f) = r_i, \qquad d_{i-1} < f \le d_i \tag{10.1}$$

where Q represents the quantization operation, r_i for $1 \le i \le L$ denotes L reconstruction levels, and d_i for $0 \le i \le L$ denotes $L + 1$ decision boundaries or decision levels. From (10.1), if f falls between d_{i-1} and d_i, it is mapped to the reconstruction level r_i. If we have determined reconstruction and decision levels, quantization of f is a deterministic process.

We can also express $\hat{f}$ in (10.1) as

$$\hat{f} = Q(f) = f + e_Q \tag{10.2}$$

where e_Q is the quantization error given by

$$e_Q = \hat{f} - f. \tag{10.3}$$

The quantization error e_Q is also called *quantization noise*. The quantity e_Q^2 can be viewed as a special case of a distortion measure $d(f, \hat{f})$, which is a measure of distance or dissimilarity between f and $\hat{f}$. Other examples of $d(f, \hat{f})$ include $|\hat{f} - f|$ and $||\hat{f}|^p - |f|^p|$. The reconstruction and decision levels are often determined by minimizing some error criterion based on $d(f, \hat{f})$, such as the average distortion D given by

$$D = E[d(f, \hat{f})] = \int_{f_0=-\infty}^{\infty} d(f_0, \hat{f}) p_f(f_0)\, df_0. \tag{10.4}$$

The most straightforward method of quantization is uniform quantization, in which the reconstruction and decision levels are uniformly spaced. Specifically, for a uniform quantizer,

$$d_i - d_{i-1} = \Delta, \qquad 1 \le i \le L \tag{10.5a}$$

$$r_i = \frac{d_i + d_{i-1}}{2}, \qquad 1 \le i \le L \tag{10.5b}$$

where Δ is the step size equal to the spacing between two consecutive reconstruction levels or two consecutive decision levels. An example of a uniform quantizer when $L = 4$ and f is assumed to be between 0 and 1 is shown in Figure 10.3.

The quantization noise e_Q is typically signal dependent. For example, the quantization noise e_Q for the uniform quantizer in Figure 10.3 is sketched in Figure 10.4. From Figure 10.4, e_Q is a function of f and therefore is signal dependent. It is possible to decorrelate the quantization noise e_Q for the uniform quantizer by a method known as *dithering* or *Roberts's pseudonoise technique*. As will be discussed in Section 10.3, decorrelation of quantization noise can be useful in improving the performance of an image coding system. It changes the characteristics of the degradation in the coded image. In addition, the decorrelated quantization noise may be reduced by the image restoration algorithms discussed in Chapter 9

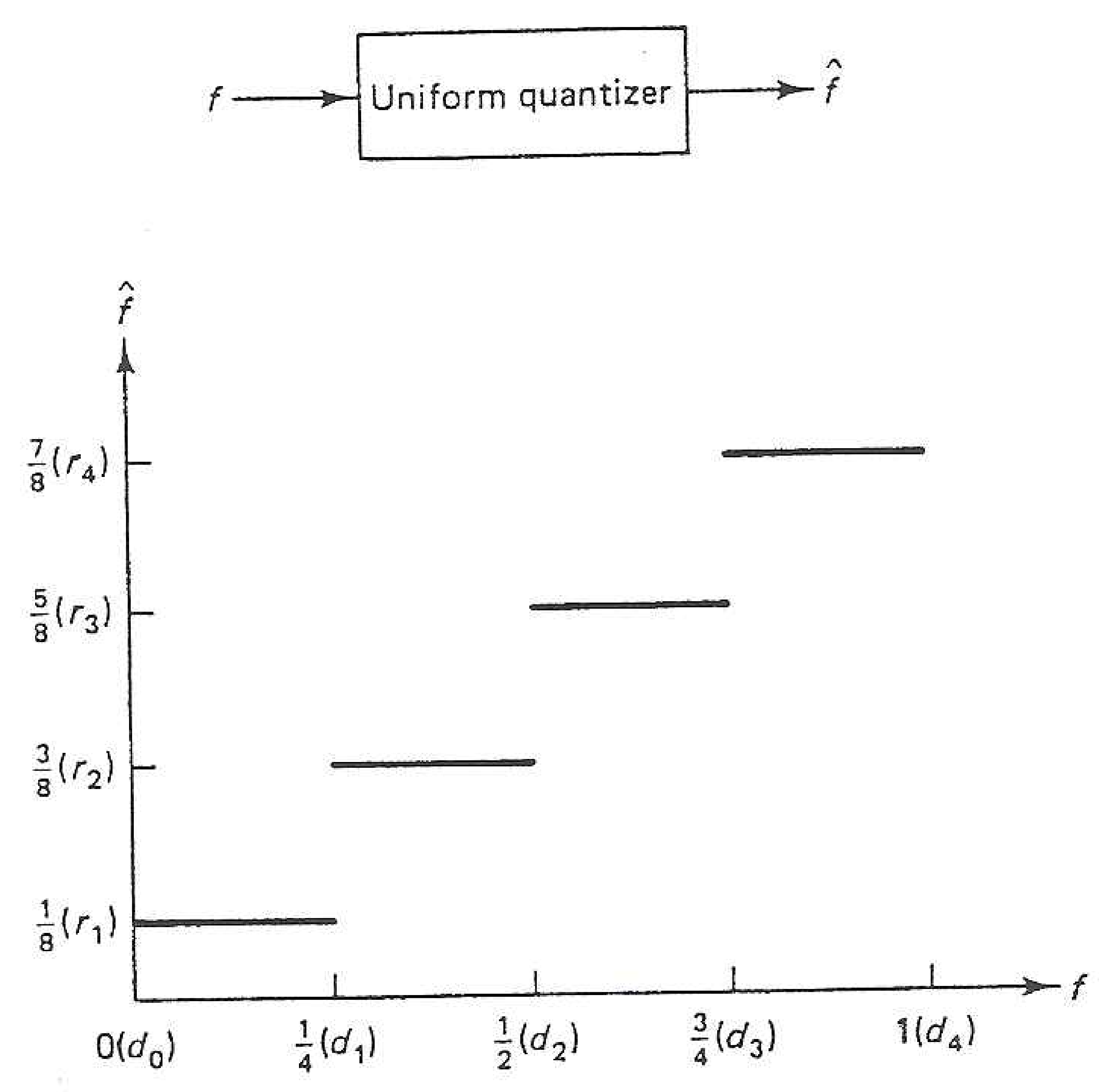


Figure 10.3 Example of uniform quantizer. The number of reconstruction levels is 4, f is assumed to be between 0 and 1, and $\hat{f}$ is the result of quantizing f. The reconstruction levels and decision boundaries are denoted by r_i and d_i, respectively.

Although uniform quantization is quite straightforward and appears to be a natural approach, it may not be optimal. Suppose f is much more likely to be in one particular region than in others. It is reasonable to assign more reconstruction levels to that region. Consider the example in Figure 10.3. If f rarely falls between d_0 and d_1, the reconstruction level r_1 is rarely used. Rearranging the reconstruction levels r_1, r_2, r_3, and r_4 so that they all lie between d_1 and d_4 makes more sense. Quantization in which reconstruction and decision levels do not have even spacing is called nonuniform quantization.

The optimum determination of r_i and d_i depends on the error criterion used. One frequently used criterion is the minimum mean square error (MMSE) criterion. Suppose we assume that f is a random variable with a probability density function $p_f(f_0)$. Using the MMSE criterion, we determine r_k and d_k by minimizing the average distortion D given by

$$D = E[d(\hat{f} - f)] = E[e_Q^2] = E[(\hat{f} - f)^2] \qquad (10.6)$$
$$= \int_{f_0=-\infty}^{\infty} p_f(f_0)(\hat{f} - f_0)^2 \, df_0.$$

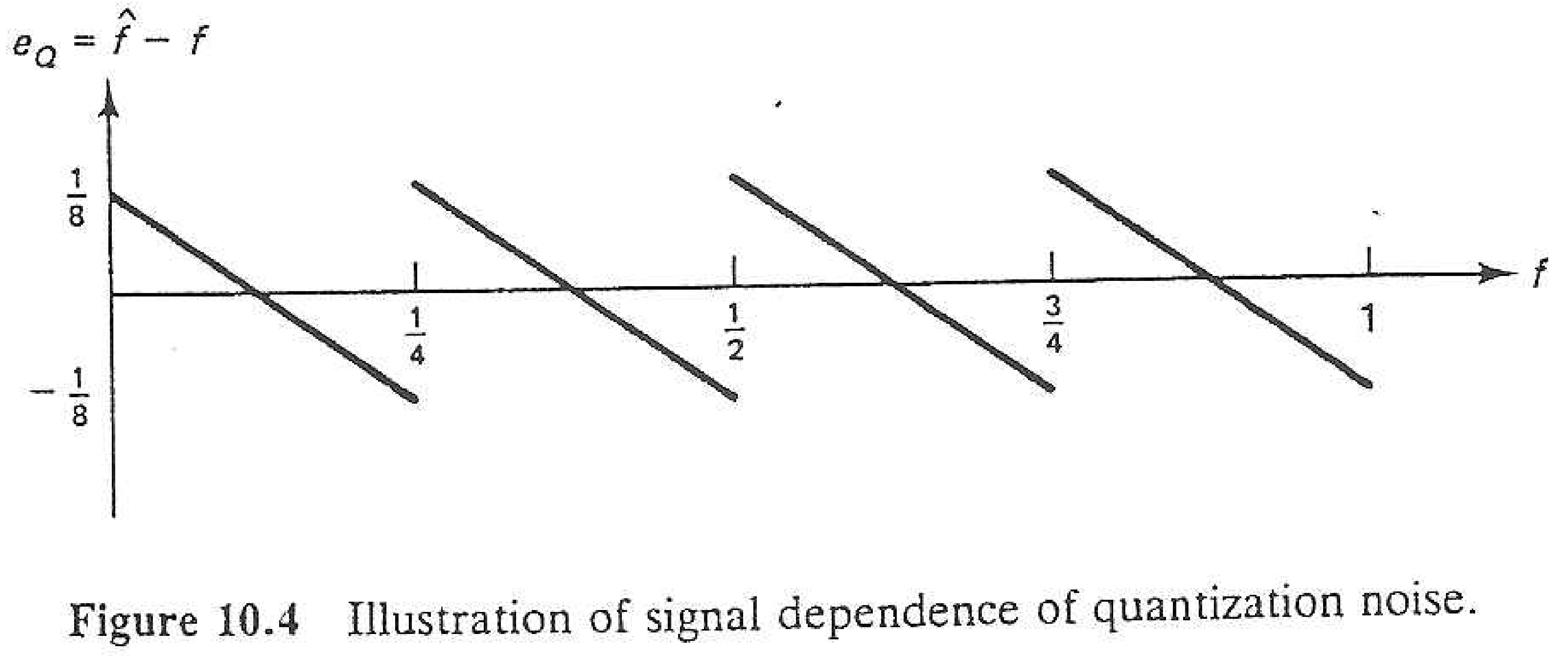


Figure 10.4 Illustration of signal dependence of quantization noise.

Quantization

- Quantization reduces ranges of values in a signal to a single value, thereby reducing entropy.
- Quantization is an integral part of lossy compression algorithms.
- Quantization is usually employed after transformation.

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

Basic Quantization Scheme: Thresholding

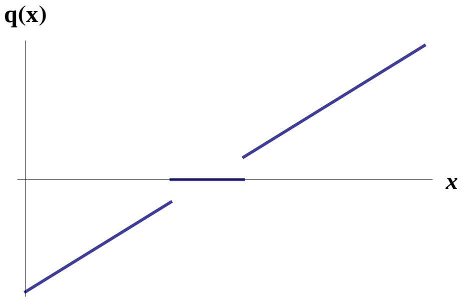
The most basic quantization technique is **Thresholding**:

Given a signal $\vec{x} = (x_i)$ and a single threshold σ, we replace values as follows:

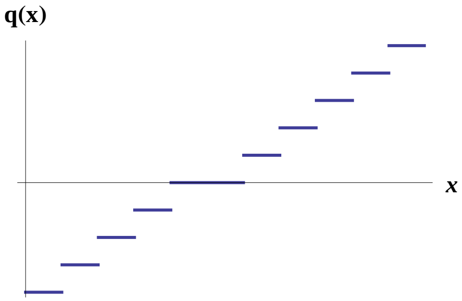
$$q(x_i) = \begin{cases} 0 & \text{if } |x_i| \leq \sigma \\ x_i & \text{if } |x_i| > \sigma \end{cases}$$

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

The thresholding quantization function:



The quantization function for a given region:



January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

$$e_q = x - Q(x), \qquad (2.\ 2)$$

x and *Q(x)* are input and quantized output, respectively.

- Quantization error is often referred to as *quantization noise*.

- Mean square quantization error, MSE_q:

$$MSE_q = \sum_{i=1}^{N} \int_{d_i}^{d_{i+1}} (x - Q(x))^2 f_X(x)dx \qquad (2.\ 3)$$

$f_x(x)$: probability density function (*pdf*)

the outer decision levels may be -∞ or ∞

when the *pdf, $f_x(x)$,* remains unchanged, fewer reconstruction levels (smaller *N*, coarse quantization) result in more distortion.

- Odd symmetry of the input-output characteristic respect to the *x=0* axis

implies that : *E(x)=0*

$$MSE_q = s_q^2$$

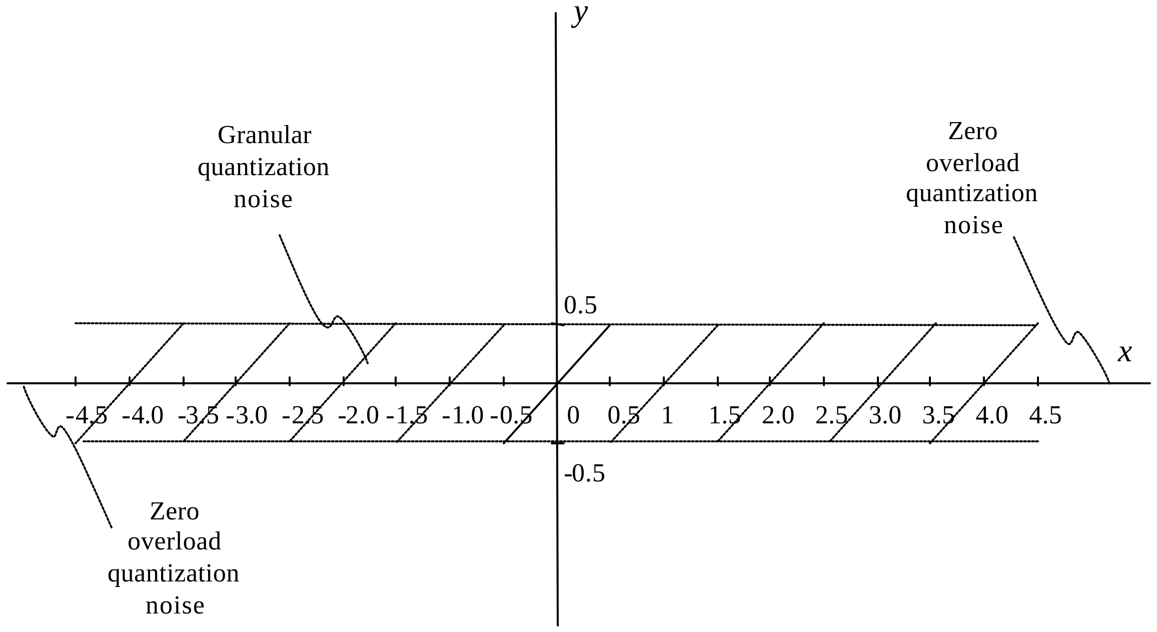


Figure 2. 8 Quantization noise of the quantizer shown in Figure 2.7

The mean square quantization error:

$$MSE_q = N \int_{d_1}^{d_2} (x - Q(x))^2 \frac{1}{N\Delta} dx$$

$$MSE_q = \frac{\Delta^2}{12}. \qquad (2.4)$$

$$SNR_{ms} = 10\log_{10}\frac{s_x^2}{s_q^2} = 10\log_{10} N^2. \quad (2.\ 5)$$

If we assume $N = 2^n$, we then have

$$SNR_{ms} = 20\log_{10} 2^n = 6.02n \quad dB.$$

- The interpretation of the above result:
  - If use natural binary code to code the reconstruction levels of a uniform quantizer with a uniformly distributed input source, then every increased bit in the coding brings out a 6.02 dB increase in the SNR_{ms}.
  - Equivalently from Equation 2.7, whenever the step size of the uniform quantizer decreases by a half, the MSE_q decreases four times.

2.2.2.2 <u>Conditions of Optimum Quantization</u>

- Derived by: [lloyd 1957, 1982; max 1960]

  for a given pdf $f_X(x)$.

The goal is to choose $\vec{L}$ and $\vec{p}$ to minimize the resulting quantization error

$$E(\vec{L},\vec{p}) = \sum_{i=1}^{m} |x_i - q(x_i)|^2$$

This is a classical Calculus problem. Rewriting we obtain:

$$E(\vec{L},\vec{p}) = \sum_{j=1}^{n} \sum_{x_i \in [L_j, L_{j+1})} (x_i - p_j)^2$$

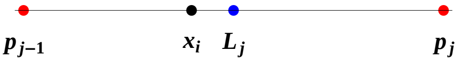
January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

$$E(\vec{L},\vec{p}) = \sum_{j=1}^{n} \sum_{x_i \in [L_j, L_{j+1})} (x_i - p_j)^2$$

Since minima will occur only if all partial derivatives are equal to 0, the following conditions need to be satisfied:

$$\frac{\partial E}{\partial p_j} = \sum_{x_i \in [L_j, L_{j+1})} 2(x_i - p_j) = 0 \Leftrightarrow p_j = \frac{\sum_{x_i \in [L_j, L_{j+1})} x_i}{\#\{i \mid x_i \in [L_j, L_{j+1})\}} \quad (1)$$

$$\frac{\partial E}{\partial L_j} = 0 \quad \Leftrightarrow \quad L_j = \frac{1}{2}(p_{j-1} + p_j) \quad (2)$$



January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

These equations can usually not be solved explicitly; instead a **two-step procedure** is used repeatedly until a fixed point has been (nearly?) reached:

- Equations (1) are used to update the values $\vec{p}$:

$$p_j^{\text{new}} = ave \ \{x_i \mid x_i \in [L_j, L_{j+1})\}$$

- Equations (2) then yield new values for $\vec{L}$:

$$L_j^{\text{new}} = \frac{1}{2}(p_{j-1}^{\text{new}} + p_j^{\text{new}}) \ , \quad j = 2, \ldots, n$$

The values for L_1 and L_{n+1} are left unchanged.

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

- Sufficient conditions:

  1. $x_1 = -\infty$ and $x_{N+1} = +\infty$ (2. 6)

  2. $\int_{d_i}^{d_{i+1}} (x - y_i) f_X(x) dx = 0 \quad i = 1, 2, \cdots, N$ (2. 7)

  3. $d_i = \frac{1}{2}(y_{i-1} + y_i) \quad i = 2, \cdots, N$ (2. 8)

  - First condition: for an input x whose range is $-\infty < x < \infty$.
  - Second: each reconstruction level is the centroid of the area under the pdf $f_X(x)$ and between the two adjacent decision levels.
  - Third: each decision level (except for the outer intervals) is the arithmetic average of the two neighboring reconstruction levels

- These conditions are general in the sense that there is no restriction imposed on the *pdf*.

2.2.2.3 <u>Optimum Uniform Quantizer with Different Input Distributions</u>

Table 2. 1 Optimal symmetric uniform quantizer for Gaussian, Laplacian and Gamma distributions (having zero mean and unit variance). Dutch[max 1960] [paez 1972]. The numbers enclosed in rectangles are the step sizes.

	Uniform			Gaussian			Laplacian			Gamma		
N	d_i	y_i	MSE	d_i	y_i	MSE	d_i	y_i	MSE	d_i	y_i	MSE
2	-1.000	-0.500	8.33×10^{-2}	-1.596	-0.798	0.363	-1.414	-0.707	0.500	-1.154	-0.577	0.668
	0.000	0.500		0.000	0.798		0.000	0.707		0.000	0.577	
	$\boxed{1.000}$			$\boxed{1.596}$			$\boxed{1.414}$			$\boxed{1.154}$		
4	-1.000	-0.750	2.08×10^{-2}	-1.991	-1.494	0.119	-2.174	-1.631	1.963×10^{-1}	-2.120	-1.590	0.320
	-0.500	-0.250		-0.996	-0.498		-1.087	-0.544		-1.060	-0.530	
	0.000	0.250		0.000	0.498		0.000	0.544		0.000	0.500	
	$\boxed{0.500}$	0.750		$\boxed{0.996}$	1.494		$\boxed{1.087}$	1.631		$\boxed{1.060}$	1.590	
	1.000			1.991			2.174			2.120		
8	-1.000	-0.875	5.21×10^{-3}	-2.344	-2.051	3.74×10^{-2}	-2.924	-2.559	7.17×10^{-2}	-3.184	-2.786	0.132
	-0.750	-0.625		-1.758	-1.465		-2.193	-1.828		-2.388	-1.990	
	-0.500	-0.375		-1.172	-0.879		-1.462	-1.097		-1.592	-1.194	
	-0.250	-0.125		-0.586	-0.293		-0.731	-0.366		-0.796	-0.398	
	0.000	0.125		0.000	0.293		0.000	0.366		0.000	0.398	
	$\boxed{0.250}$	0.375		$\boxed{0.586}$	0.879		$\boxed{0.731}$	1.097		$\boxed{0.796}$	1.194	
	0.500	0.625		1.172	1.465		1.462	1.828		1.592	1.990	
	0.750	0.875		1.758	2.051		2.193	2.559		2.388	2.786	
	1.000			2.344			2.924			3.184		
16	-1.000	-0.938	1.30×10^{-3}	-2.680	-2.513	1.15×10^{-2}	-3.648	-3.420	2.54×10^{-2}	-4.320	-4.050	5.01×10^{-2}
	-0.875	-0.813		-2.345	-2.178		-3.192	-2.964		-3.780	-3.510	
	-0.750	-0.688		-2.010	-1.843		-2.736	-2.508		-3.240	-2.970	
	-0.625	-0.563		-1.675	-1.508		-2.280	-2.052		-2.700	-2.430	
	-0.500	-0.438		-1.340	-1.173		-1.824	-1.596		-2.160	-1.890	
	-0.375	-0.313		-1.005	-0.838		-1.368	-1.140		-1.620	-1.350	
	-0.250	-0.188		-0.670	-0.503		-0.912	-0.684		-1.080	-0.810	
	-0.125	-0.063		-0.335	-0.168		-0.456	-0.228		-0.540	-0.270	
	0.000	0.063		0.000	0.168		0.000	0.228		0.000	0.270	
	$\boxed{0.125}$	0.188		$\boxed{0.335}$	0.503		$\boxed{0.456}$	0.684		$\boxed{0.540}$	0.810	
	0.250	0.313		0.670	0.838		0.912	1.140		1.080	1.350	
	0.375	0.438		1.005	1.173		1.368	1.596		1.620	1.890	
	0.500	0.563		1.340	1.508		1.824	2.052		2.160	2.430	
	0.625	0.688		1.675	1.843		2.280	2.508		2.700	2.970	
	0.750	0.813		2.010	2.178		2.736	2.964		3.240	3.510	
	0.875	0.938		2.345	2.513		3.192	3.420		3.780	4.050	
	1.000			2.680			3.648			4.320		

Table 2. 2 Optimal symmetric quantizer for uniform, Gaussian, Laplacian and Gamma distributions (The uniform distribution is between [-1, 1], the other three distributions have zero mean and unit variance.) [lloyed 1957, 1982] [max 1990] [paez 1972]

	Uniform			Gaussian			Laplacian			Gamma		
N	**d_i**	**y_i**	**MSE**	**d_i**	**y_i**	**MSE**	**d_i**	**y_i**	**MSE**	**d_i**	**y_i**	**MSE**
2	-1.000 0.000 1.000	-0.500 0.500	8.33 $\times10^{-2}$	-∞ 0.000 ∞	-0.799 0.799	0.363	-∞ 0.000 ∞	-0.707 0.707	0.500	-∞ 0.000 ∞	-0.577 0.577	0.668
4	-1.000 -0.500 0.000 0.500 1.000	-0.750 -0.250 0.250 0.750	2.08 $\times10^{-2}$	-∞ -0.982 0.000 -0.982 ∞	-1.510 -0.453 0.453 1.510	0.118	-∞ -1.127 0.000 1.127 ∞	-1.834 -0.420 0.420 1.834	1.765 $\times10^{-1}$	-∞ -1.205 0.000 1.205 ∞	-2.108 -0.302 0.302 2.108	0.233
8	-1.000 -0.750 -0.500 -0.250 0.000 0.250 0.500 0.750 1.000	-0.875 -0.625 -0.375 -0.125 0.125 0.375 0.625 0.875	5.21 $\times10^{-3}$	-∞ -1.748 -1.050 -0.501 0.000 0.501 1.050 1.748 ∞	-2.152 -1.344 -0.756 -0.245 0.245 0.756 1.344 2.152	3.45 $\times10^{-2}$	-∞ -2.377 -1.253 -0.533 0.000 0.533 1.253 2.377 ∞	-3.087 -1.673 -0.833 -0.233 0.233 0.833 1.673 3.087	5.48 $\times10^{-2}$	-∞ -2.872 -1.401 -0.504 0.000 0.504 1.401 2.872 ∞	-3.799 -1.944 -0.859 -0.149 0.149 0.859 1.944 3.799	7.12 $\times10^{-2}$
16	-1.000 -0.875 -0.750 -0.625 -0.500 -0.375 -0.250 -0.125 0.000 0.125 0.250 0.375 0.500 0.625 0.750 0.875 1.000	-0.938 -0.813 -0.688 -0.563 -0.438 -0.313 -0.188 -0.063 0.063 0.188 0.313 0.438 0.563 0.688 0.813 0.938	1.30 $\times10^{-3}$	-∞ -2.401 -1.844 -1.437 -1.099 -0.800 -0.522 -0.258 0.000 0.258 0.522 0.800 1.099 1.437 1.844 2.401 ∞	-2.733 -2.069 -1.618 -1.256 -0.942 -0.657 -0.388 -0.128 0.128 0.388 0.657 0.942 1.256 1.618 2.069 2.733	9.50 $\times10^{-3}$	-∞ -3.605 -2.499 -1.821 -1.317 -0.910 -0.566 -0.266 0.000 0.266 0.566 0.910 1.317 1.821 2.499 3.605 ∞	-4.316 -2.895 -2.103 -1.540 -1.095 -0.726 -0.407 -0.126 0.126 0.407 0.726 1.095 1.540 2.103 2.895 4.316	1.54 $\times10^{-2}$	-∞ -5.050 -3.407 -2.372 -1.623 -1.045 -0.588 -0.229 0.000 0.229 0.588 1.045 1.623 2.372 3.407 5.050 ∞	-6.085 -4.015 -2.798 -1.945 -1.300 -0.791 -0.386 -0.072 0.072 0.386 0.791 1.300 1.945 2.798 4.015 6.085	1.96$\times$ 10^{-2}

- The solution to optimum quantizer design for finitely many reconstruction levels N when input x obeys Gaussian distribution was obtained numerically [lloyd 1957, 1982, max 1960].
- Lloyd-Max quantizers.
- The design for Laplacian and Gamma distribution were tabulated in [paez 1972].
- Performance comparison

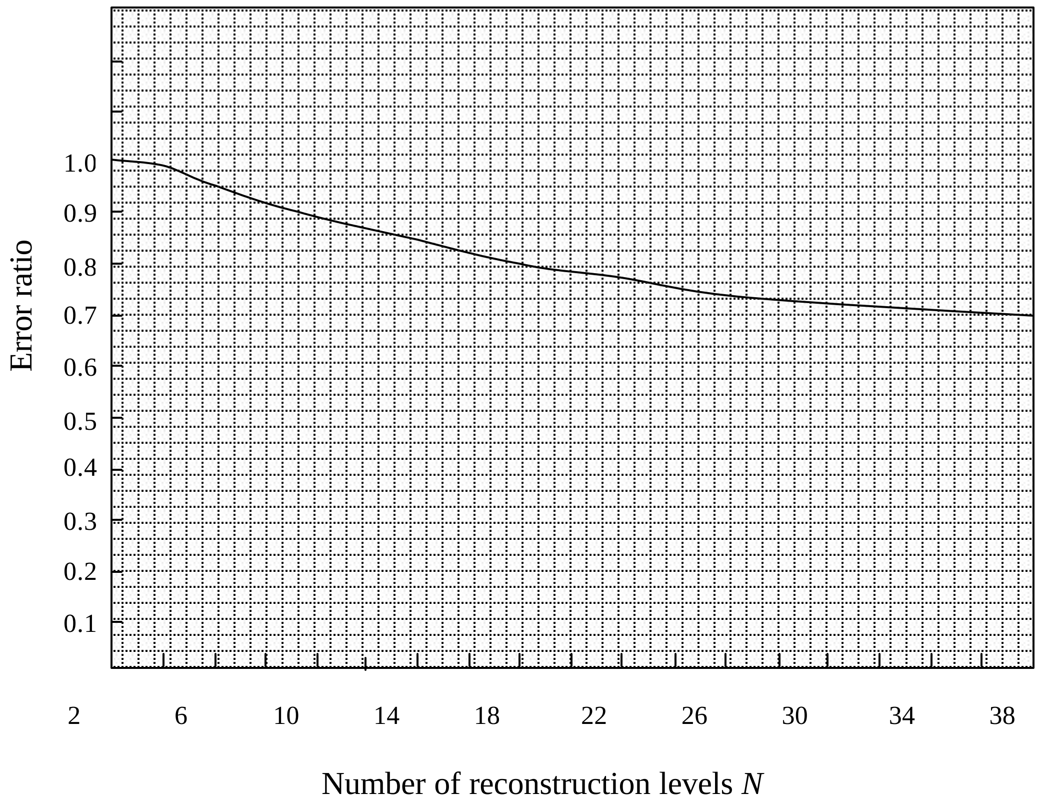


Figure 2. 9 Ratio of error for optimal quantizer to error for optimum uniform quantizer vs. number of reconstruction levels *N*. (Minimum mean square error for Gaussian distributed input with a zero mean and unit variance). Data from [max 1960].

2.5 PCM

- Pulse code modulation (PCM) is closely related to quantization.
- PCM is <u>the earliest, best established, and most frequently applied</u> coding system

  despite the fact that

  - the most bit-consuming digitizing system (since it encodes each pixel independently)
  - a very demanding system in terms of bit error rate on the digital channel.
- PCM is now the most important form of pulse modulation.
- Pulse modulation links an analog signal to a pulse train in the following way.
  - The analog signal is first sampled
  - The sampled values are used to modulate a pulse train.

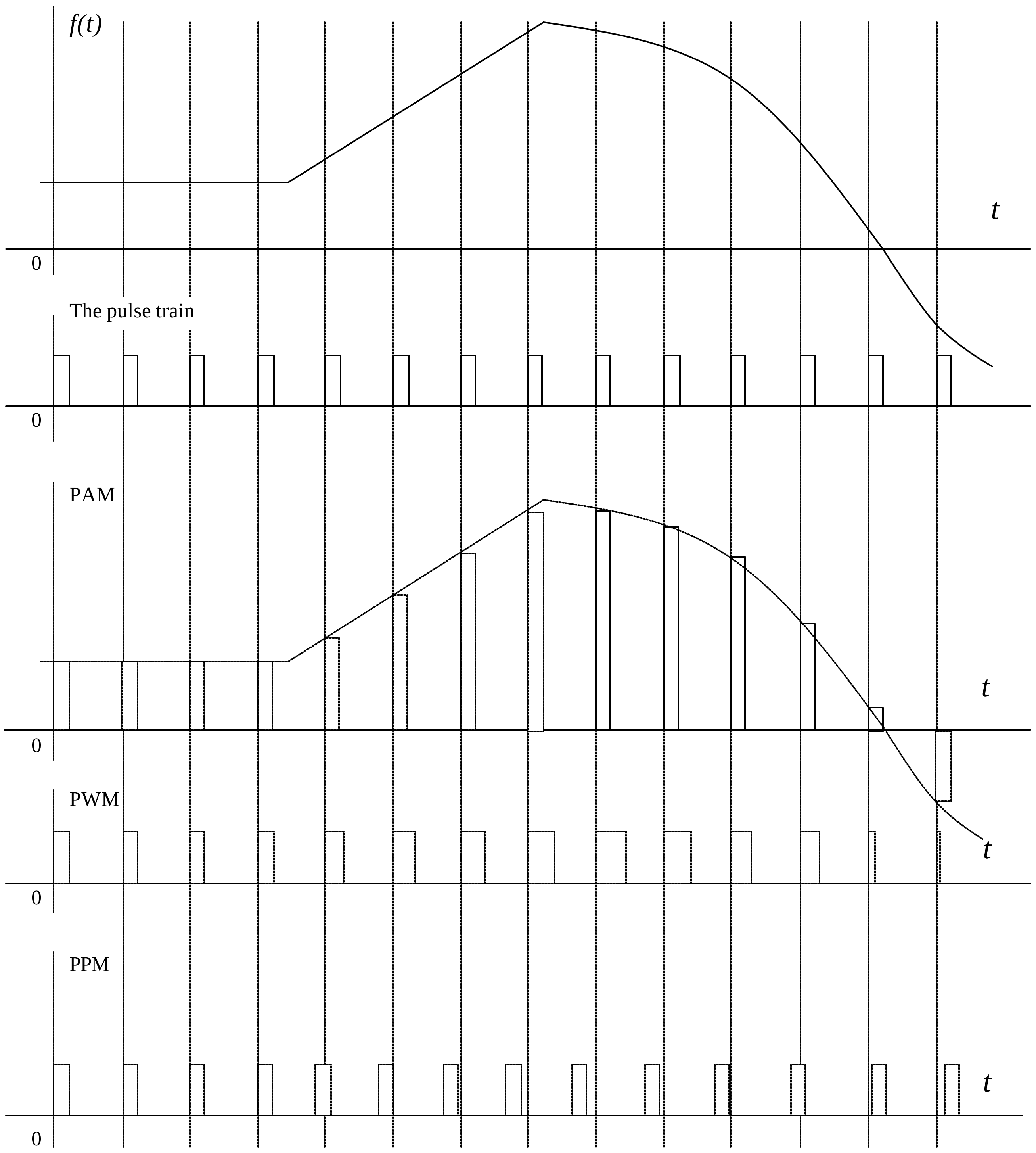
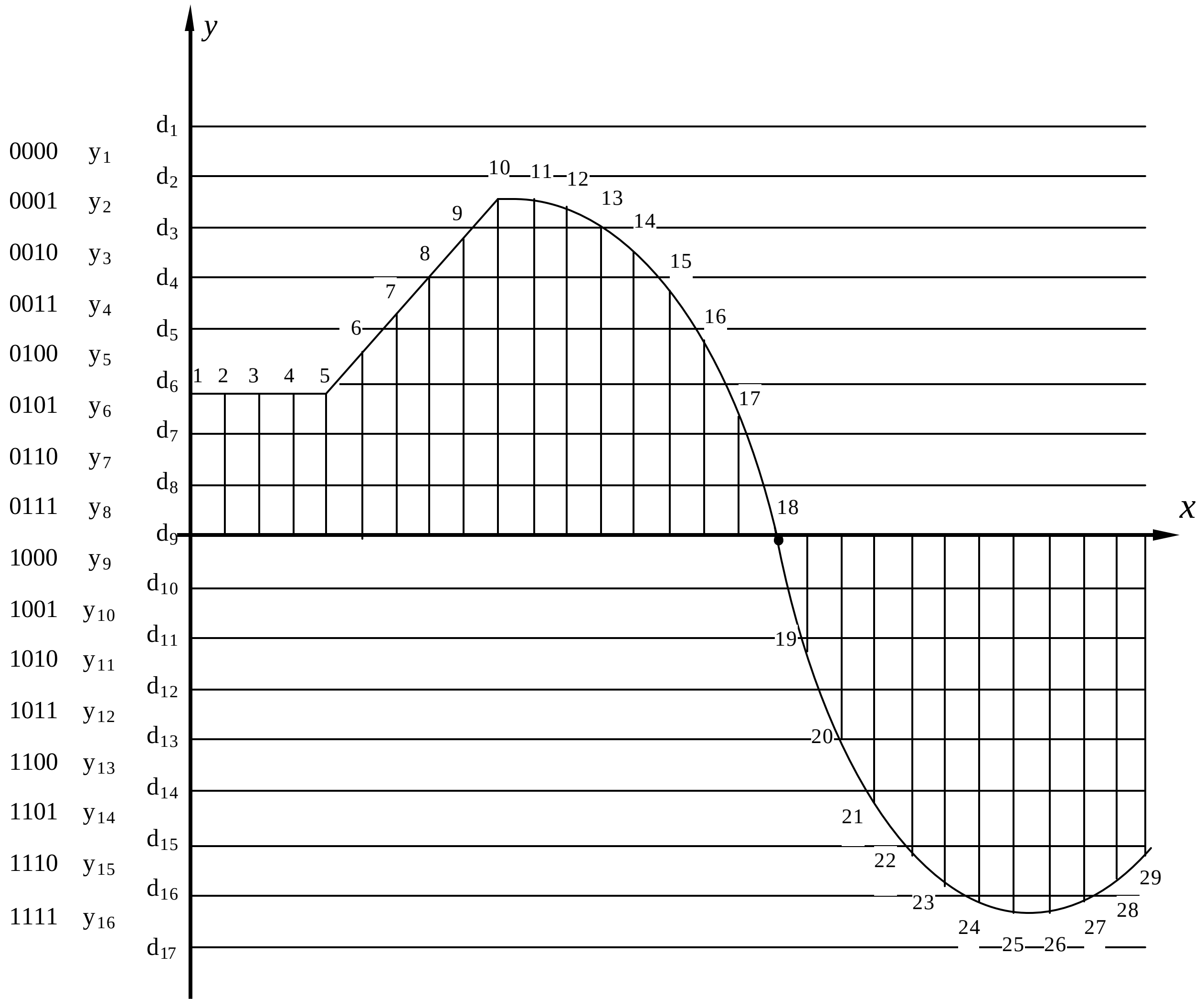


Figure 2.15 Pulse modulation



Output code (from left to right, from top to bottom):

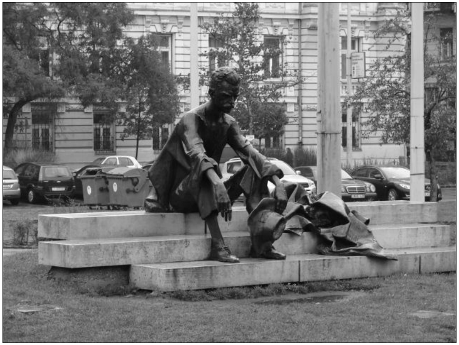
0101	0101	0101	0101	0101	0100	0011	0011	0010	0001	0001	0001	0010	0010	0011
0100	0101	1000	1010	1100	1101	1110	1110	1111	1111	1111	1111	1110	1110	

Figure 2. 16 Pulse code modulation (PCM)

- In PCM, a sampling, a uniform quantization, and a natural binary code converts the input **analog** signal into a **digital** signal.
- In this way, an analog signal modulates a pulse train with the natural binary code.
- By far, PCM is more popular than other types of pulse modulation

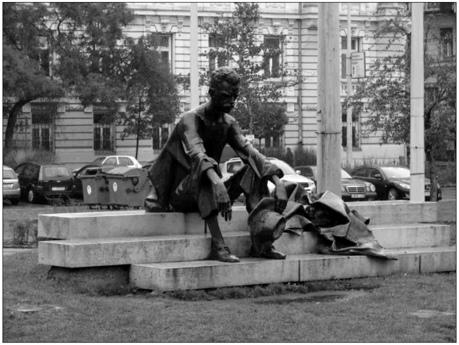
  since the *code* modulation is much more robust against various noises than amplitude modulation, width modulation and position modulation.
- In fact, almost all coding techniques include a PCM component.
- In digital image processing, given digital images usually appear in PCM format.
- It is known that an acceptable PCM representation of monochrome picture requires 6 to 8 bits per pixel [huang 1975].
- It is used so commonly in practice that its performance normally serves as a standard against which other coding techniques are compared.

As an example, we apply the algorithm to a grayscale image, with $n = 32$. The initial bins are chosen at random.



Original image, entropy=7.65

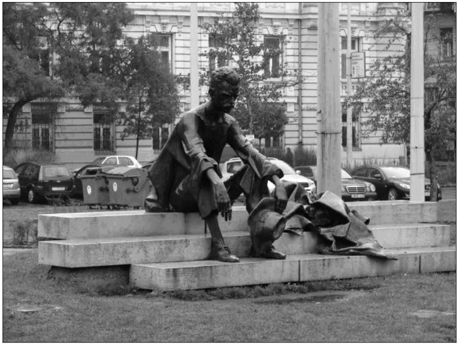
As an example, we apply the algorithm to a grayscale image, with $n = 32$. The initial bins are chosen at random.



21 iterations, entropy=4.75, PSNR 40.7

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

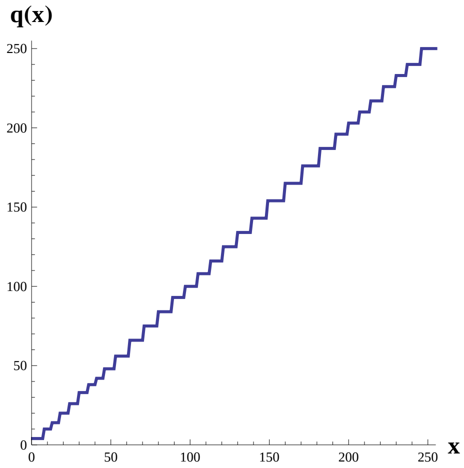
As an example, we apply the algorithm to a grayscale image, with $n = 32$. The initial bins are chosen at random.



28 iterations, entropy=4.56, PSNR 40.0

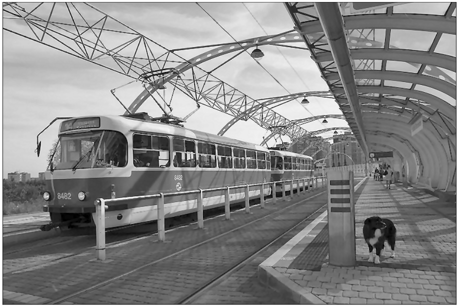
January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

Here is the quantization function for the second run:



January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

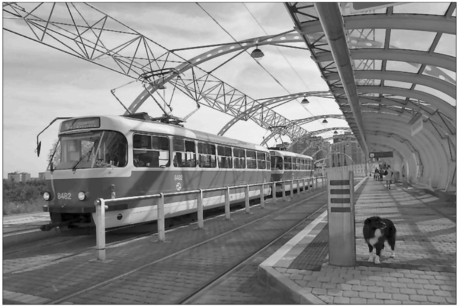
Quantization applied to "raw" images usually gives bad results in areas of gradual gray-value change (sky, water, etc.):



Original image:
entropy=7.63

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

Quantization applied to “raw” images usually gives bad results in areas of gradual gray-value change (sky, water, etc.):

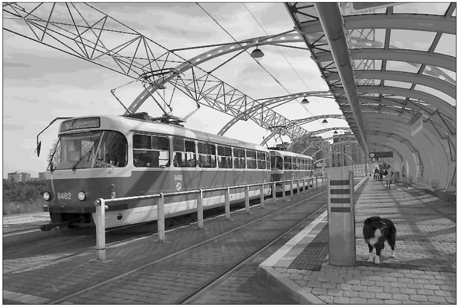


$n = 2^6$:

10 iterations, entropy=5.53, PSNR 45.1

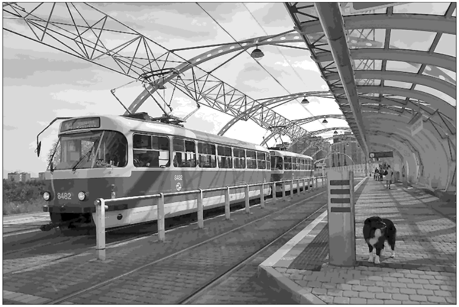
January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

Quantization applied to "raw" images usually gives bad results in areas of gradual gray-value change (sky, water, etc.):



$n = 2^5$:
18 iterations, entropy=4.35, PSNR 38.0

Quantization applied to “raw” images usually gives bad results in areas of gradual gray-value change (sky, water, etc.):



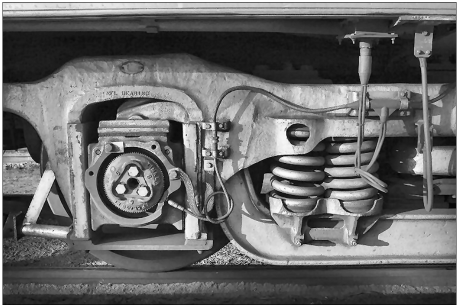
$n = 2^4$:

22 iterations, entropy=3.76, PSNR 34.7

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

We now compare step quantization to Lloyd-Max quantization for a transformed image:

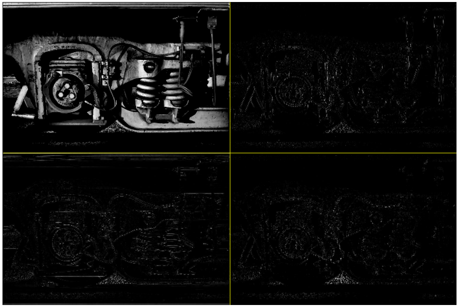
- We use the CDF97 wavelet transform once.
- For the step quantization we use $\tau = 16$:



Original image (-128): entropy=7.74

We now compare step quantization to Lloyd-Max quantization for a transformed image:

- We use the CDF97 wavelet transform once.
- For the step quantization we use $\tau = 16$:



CDF97-transformed image

We now compare step quantization to Lloyd-Max quantization for a transformed image:

- We use the CDF97 wavelet transform once.
- For the step quantization we use $\tau = 16$:

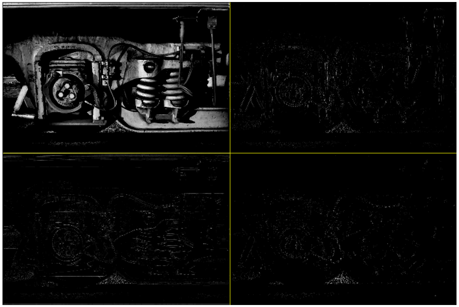
$\tau/2$=8	τ=16
τ=16	2τ=32

Step sizes for the quantization

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

We now compare step quantization to Lloyd-Max quantization for a transformed image:

- We use the CDF97 wavelet transform once.
- For the step quantization we use $\tau = 16$:

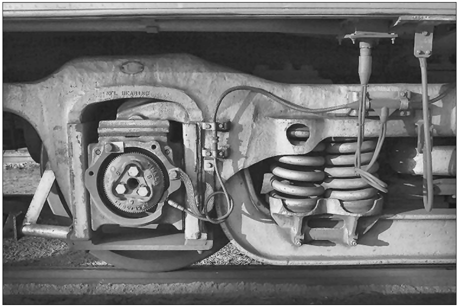


Quantized transform: entropy=2.75

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

We now compare step quantization to Lloyd-Max quantization for a transformed image:

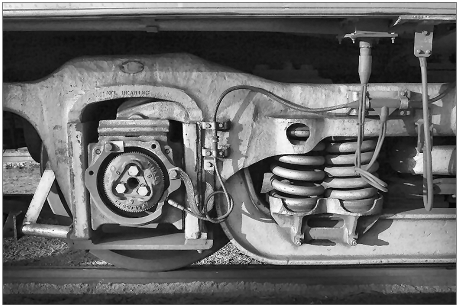
- We use the CDF97 wavelet transform once.
- For the step quantization we use $\tau = 16$:



Reconstructed image: PSNR 32.44

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

Here is the same example with Lloyd-Max quantization:



Original image (-128): entropy=7.74

January 6-9 | New Orleans, LA
2011 Joint Mathematics Meetings

Noting that $\hat{f}$ is one of the L reconstruction levels obtained by (10.1), we can write (10.6) as

$$D = \sum_{i=1}^{L} \int_{f_0=d_{i-1}}^{d_i} p_f(f_0)(r_i - f_0)^2 \, df_0. \tag{10.7}$$

To minimize D,

$$\begin{aligned} \frac{\partial D}{\partial r_k} &= 0, \qquad 1 \le k \le L \\ \frac{\partial D}{\partial d_k} &= 0, \qquad 1 \le k \le L-1 \\ d_0 &= -\infty \\ d_L &= \infty. \end{aligned} \tag{10.8}$$

From (10.7) and (10.8),

$$r_k = \frac{\displaystyle\int_{f_0=d_{k-1}}^{d_k} f_0 p_f(f_0)\, df_0}{\displaystyle\int_{f_0=d_{k-1}}^{d_k} p_f(f_0)\, df_0}, \qquad 1 \le k \le L \tag{10.9a}$$

$$d_k = \frac{r_k + r_{k+1}}{2}, \qquad 1 \le k \le L-1 \tag{10.9b}$$

$$d_0 = -\infty \tag{10.9c}$$

$$d_L = \infty. \tag{10.9d}$$

The first set of equations in (10.9) states that a reconstruction level r_k is the centroid of $p_f(f_0)$ over the interval $d_{k-1} \le f_0 \le d_k$. The remaining set of equations states that the decision level d_k except d_0 and d_L is the middle point between two reconstruction levels r_k and r_{k+1}. Equation (10.9) is a necessary set of equations for the optimal solution. For a certain class of probability density functions, including uniform, Gaussian, and Laplacian densities, (10.9) is also sufficient.

Solving (10.9) is a nonlinear problem. The nonlinear problem has been solved for some specific probability density functions. The solutions when $p_f(f_0)$ is uniform, Gaussian, and Laplacian are tabulated in Table 10.1. A quantizer based on the MMSE criterion is often referred to as a Lloyd-Max quantizer [Lloyd; Max]. From Table 10.1, the uniform quantizer is the optimal MMSE quantizer when $p_f(f_0)$ is a uniform probability density function. For other densities, the optimal solution is a nonuniform quantizer. For example, the optimal reconstruction and decision levels for the Gaussian $p_f(f_0)$ with variance of 1 when $L = 4$ are shown in Figure 10.5.

It is useful to evaluate the performance improvement that the optimal MMSE quantizer gives over the simpler uniform quantizer. As an example, consider a Gaussian $p_f(f_0)$ with mean of 0 and variance of 1. The average distortion D in

TABLE 10.1 PLACEMENT OF RECONSTRUCTION AND DECISION LEVELS FOR LLOYD-MAX QUANTIZER. FOR UNIFORM PDF, $p_f(f_0)$ IS ASSUMED UNIFORM BETWEEN -1 AND 1. THE GAUSSIAN PDF IS ASSUMED TO HAVE MEAN OF 0 AND VARIANCE OF 1. FOR THE LAPLACIAN PDF,

$$p_f(f_0) = \frac{\sqrt{2}}{2\sigma} e^{-\frac{\sqrt{2}|f_0|}{\sigma}} \text{ with } \sigma = 1.$$

	Uniform		Gaussian		Laplacian	
Bits	r_i	d_i	r_i	d_i	r_i	d_i
1	−0.5000	−1.0000	−0.7979	$-\infty$	−0.7071	$-\infty$
	0.5000	0.0000	0.7979	0.0000	0.7071	0.0000
		1.0000		∞		∞
2	−0.7500	−1.0000	−1.5104	$-\infty$	−1.8340	$-\infty$
	−0.2500	−0.5000	−0.4528	−0.9816	−0.4198	−1.1269
	0.2500	0.0000	0.4528	0.0000	0.4198	0.0000
	0.7500	0.5000	1.5104	0.9816	1.8340	1.1269
		1.0000		∞		∞
3	−0.8750	−1.0000	−2.1519	$-\infty$	−3.0867	$-\infty$
	−0.6250	−0.7500	−1.3439	−1.7479	−1.6725	−2.3796
	−0.3750	−0.5000	−0.7560	−1.0500	−0.8330	−1.2527
	−0.1250	−0.2500	−0.2451	−0.5005	−0.2334	−0.5332
	0.1250	0.0000	0.2451	0.0000	0.2334	0.0000
	0.3750	0.2500	0.7560	0.5005	0.8330	0.5332
	0.6250	0.5000	1.3439	1.0500	1.6725	1.2527
	0.8750	0.7500	2.1519	1.7479	3.0867	2.3769
		1.0000		∞		∞
4	−0.9375	−1.0000	−2.7326	$-\infty$	−4.4311	$-\infty$
	−0.8125	−0.8750	−2.0690	−2.4008	−3.0169	−3.7240
	−0.6875	−0.7500	−1.6180	−1.8435	−2.1773	−2.5971
	−0.5625	−0.6250	−1.2562	−1.4371	−1.5778	−1.8776
	−0.4375	−0.5000	−0.9423	−1.0993	−1.1110	−1.3444
	−0.3125	−0.3750	−0.6568	−0.7995	−0.7287	−0.9198
	−0.1875	−0.2500	−0.3880	−0.5224	−0.4048	−0.5667
	−0.0625	−0.1250	−0.1284	−0.2582	−0.1240	−0.2664
	0.0625	0.0000	0.1284	0.0000	0.1240	0.0000
	0.1875	0.1250	0.3880	0.2582	0.4048	0.2644
	0.3125	0.2500	0.6568	0.5224	0.7287	0.5667
	0.4375	0.3750	0.9423	0.7995	1.1110	0.9198
	0.5625	0.5000	1.2562	1.0993	1.5778	1.3444
	0.6875	0.6250	1.6180	1.4371	2.1773	1.8776
	0.8125	0.7500	2.0690	1.8435	3.0169	2.5971
	0.9375	0.8750	2.7326	2.4008	4.4311	3.7240
		1.0000		∞		∞

(10.6) as a function of the number of reconstruction levels L is shown in Figure 10.6 for the optimal MMSE quantizer (solid line). The average distortion D as a function of L is also shown in Figure 10.6 for the uniform quantizer* (dotted line), in which the reconstruction levels r_i are chosen to be symmetric with respect to

*The definition of uniform quantization by (10.5) has been extended in this case to account for a Gaussian random variable f whose value can range between $-\infty$ and ∞.

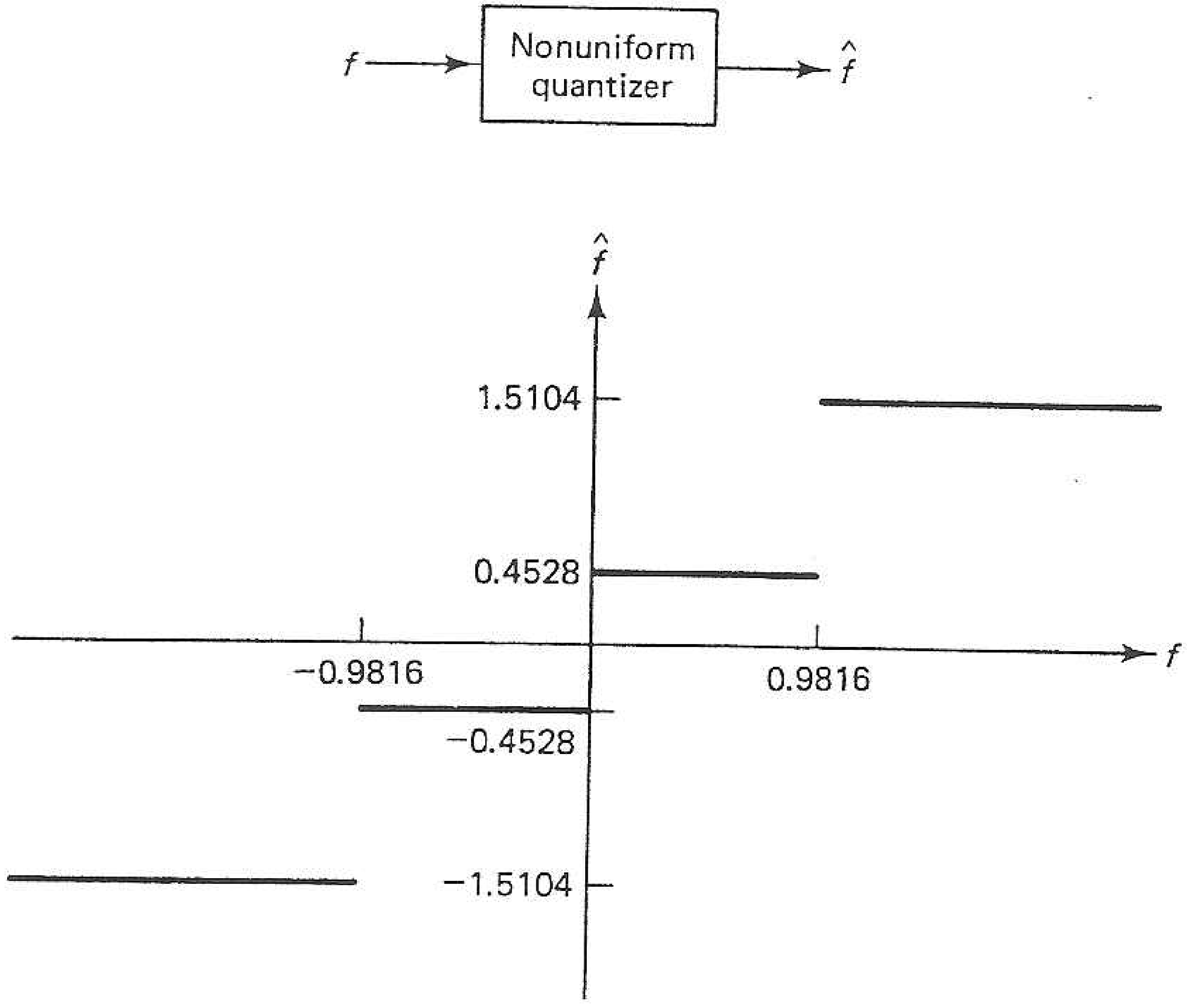


Figure 10.5 Example of a Lloyd-Max quantizer. The number of reconstruction levels is 4, and the probability density function for f is Gaussian with mean of 0 and variance of 1.

the origin, the minimum and maximum decision boundaries are assumed to be $-\infty$ and ∞, respectively, and the reconstruction step size Δ is chosen to minimize the average distortion D. From Figure 10.6, if codewords of uniform length are used to represent the reconstruction levels, the saving in the number of bits is in the range of $0 \sim \frac{1}{2}$ bit for L between 2 (1 bit) and 128 (7 bits). This example is based

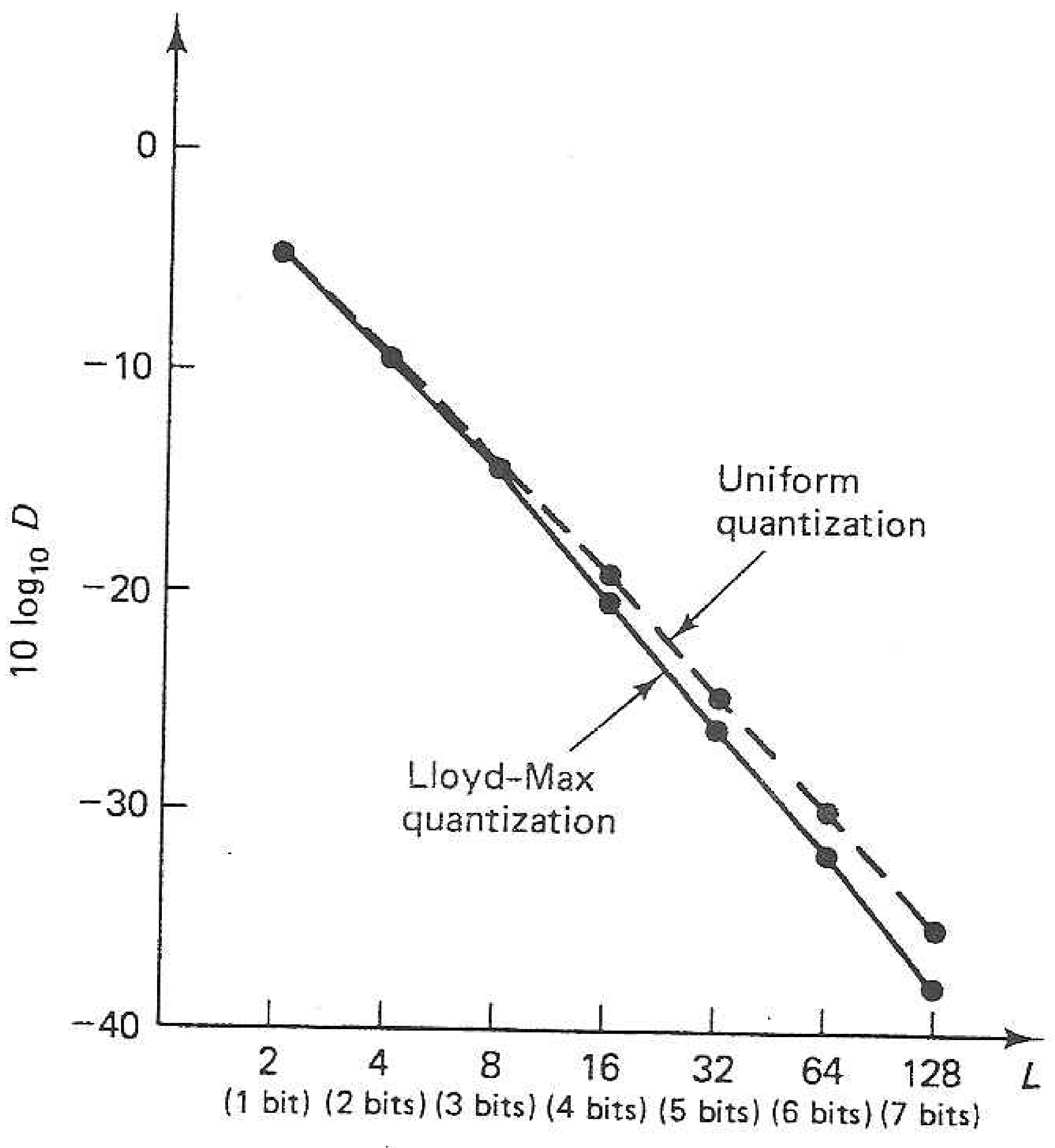


Figure 10.6 Comparison of average distortion $D = E[(\hat{f} - f)^2]$ as a function of L, the number of reconstruction levels, for a uniform quantizer (dotted line) and the Lloyd-Max quantizer (solid line). The vertical axis is $10 \log_{10} D$. The probability density function is assumed to be Gaussian with variance of 1.

Commonly used objective measures are the root-mean-square error e_{RMS}, the root-mean-square signal-to-noise ratio SNR_{RMS}, and the peak signal-to-noise ratio SNR_{PEAK}. We can define the error between an original, uncompressed pixel value and the reconstructed (decompressed) pixel value as

$$\begin{aligned} \text{error}(r,c) &= \hat{I}(r,c) - I(r,c) \\ \text{where } I(r,c) &= \text{the original image} \\ \hat{I}(r,c) &= \text{the decompressed image} \end{aligned}$$

Next, we can define the total error in an $N \times N$ decompressed image as

$$\text{Total error} = \sum_{r=0}^{N-1} \sum_{c=0}^{N-1} [\hat{I}(r,c) - I(r,c)]$$

The *root-mean-square error* is found by taking the square root ("root") of the error squared ("square") divided by the total number of pixels in the image ("mean"):

$$e_{RMS} = \sqrt{\frac{1}{N^2} \sum_{r=0}^{N-1} \sum_{c=0}^{N-1} [\hat{I}(r,c) - I(r,c)]^2}$$

The smaller the value of the error metrics, the better the compressed image represents the original image. Alternately, with the signal-to-noise (SNR) metrics, a larger number implies a better image. The SNR metrics consider the decompressed image $\hat{I}(r, c)$ to be the "signal" and the error to be "noise." We can define the *root-mean-square signal-to-noise ratio* as

$$SNR_{RMS} = \sqrt{\frac{\sum_{r=0}^{N-1} \sum_{c=0}^{N-1} [\hat{I}(r,c)]^2}{\sum_{r=0}^{N-1} \sum_{c=0}^{N-1} [\hat{I}(r,c) - I(r,c)]^2}}$$

Another related metric, the *peak signal-to-noise ratio*, is defined as

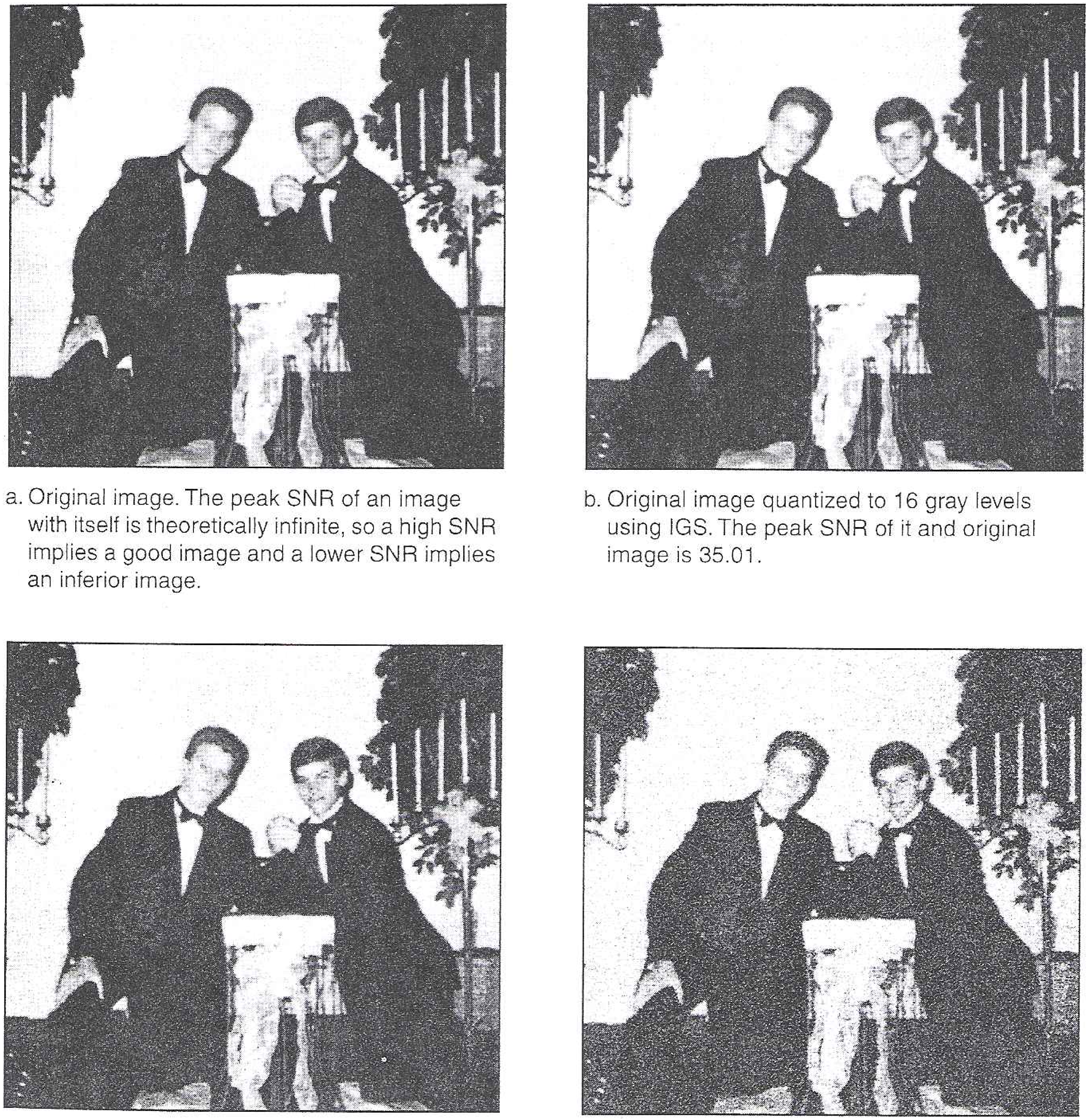
$$SNR_{PEAK} = 10 \log_{10} \frac{(L-1)^2}{\frac{1}{N^2} \sum_{r=0}^{N-1} \sum_{c=0}^{N-1} [\hat{I}(r,c) - I(r,c)]^2}$$

where L = the number of gray levels
(e.g., for 8 bits L = 256)

These objective measures are often used in research because they are easy to generate and seemingly unbiased, but remember that these metrics are not necessarily correlated to our perception of an image. The subjective measures are a better method for comparison of compression algorithms, if the goal is to achieve high-quality images as defined by our visual perception.

Subjective testing is performed by creating a database of images to be tested, quality. This scale can then be used by human test subjects to determine image fidelity. In order to provide unbiased results, evaluation with subjective measures requires careful selection of the test subjects and carefully designed evaluation experiments. The objective criteria, although widely used, are not necessarily correlated with our perception of image quality. However, they are useful as a relative measure in comparing different versions of the same image (see Figure 5.1-1).

Figure 5.1-1 Objective Fidelity Measures



a. Original image. The peak SNR of an image with itself is theoretically infinite, so a high SNR implies a good image and a lower SNR implies an inferior image.

b. Original image quantized to 16 gray levels using IGS. The peak SNR of it and original image is 35.01.

c. Original image with gaussian noise added with a variance of 200 and mean 0. Peak SNR of it and original image is 28.14.

d. Original image with gaussian noise added with a variance of 800 and mean 0. Peak SNR of it and original image is 22.73.

criterion. The results are then analyzed statistically, typically using the averages and standard deviations as metrics. Subjective fidelity measures can be classified into three categories. The first type are referred to as *impairment tests*, where the test subjects score the images in terms of how bad they are. The second type are *quality tests*, where the test subjects rate the images in terms of how good they are. The third type are called *comparison tests*, where the images are evaluated on a side-by-side basis. The comparison type tests are considered to provide the most useful results, as they provide a relative measure, which is the easiest metric for most people to determine. Impairment and quality tests require an absolute measure, which is more difficult to determine in an unbiased fashion. In Table 5.1-1 are examples of internationally accepted scoring scales for these three types of subjective fidelity measures.

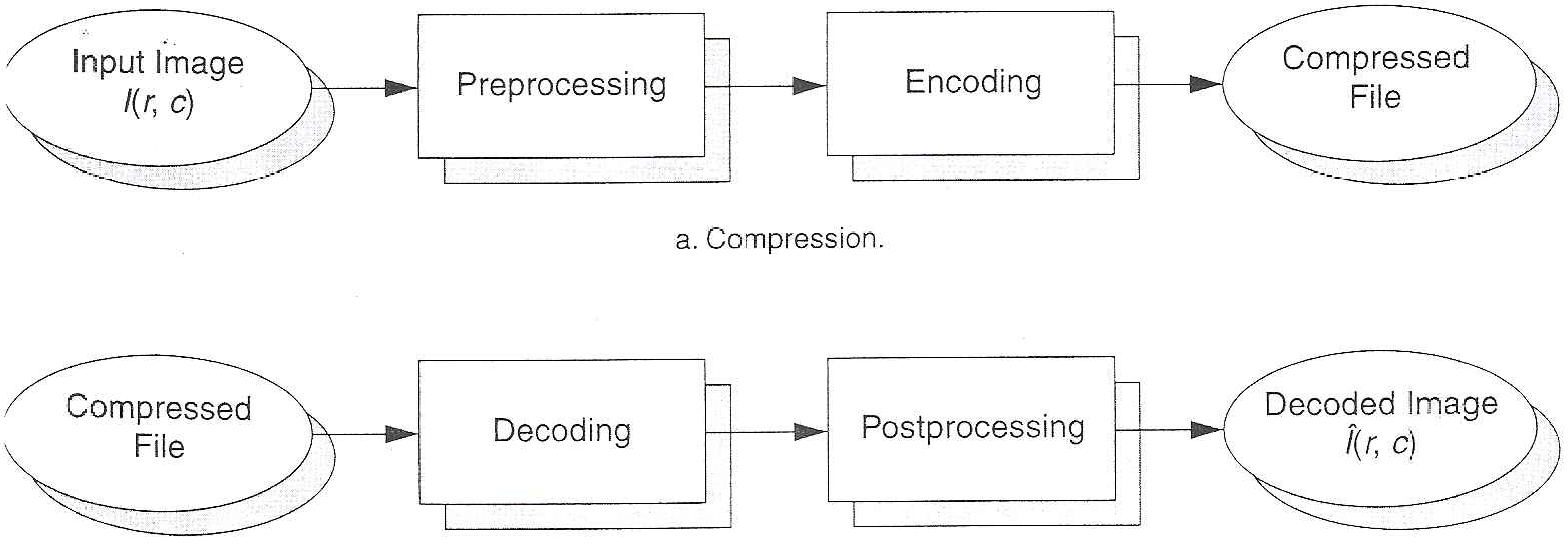
Table 5.1-1 Subjective Fidelity Scoring Scales

Impairment	Quality	Comparison
5—Imperceptible	A—Excellent	+2 much better
4—Perceptible, not annoying	B—Good	+1 better
3—Somewhat annoying	C—Fair	0 the same
2—Severely annoying	D—Poor	–1 worse
1—Unusable	E—Bad	–2 much worse

5.1.2 Compression System Model

The compression system model consists of two parts: the compressor and the decompressor. The *compressor* consists of a preprocessing stage and encoding stage, whereas the *decompressor* consists of a decoding stage followed by a postprocessing stage (Figure 5.1-2). Before encoding, preprocessing is performed to prepare the image

Figure 5.1-2 Compression System Model



information is available, we will provide the NMSE (normalized mean square error) and the SNR (signal to noise ratio) defined as

$$\text{NMSE in \%} = 100\left(\frac{\text{Var}\,[\hat{f}(n_1, n_2) - f(n_1, n_2)]}{\text{Var}\,[f(n_1, n_2)]}\right) \ \% \qquad (10.33\text{a})$$

$$\text{SNR in dB} = 10 \log \left(\frac{\text{NMSE in \%}}{100}\right)^{-1} \ \text{dB} \qquad (10.33\text{b})$$

where $f(n_1, n_2)$ is the original image, and $\hat{f}(n_1, n_2)$ is the coded image. The NMSE measure in (10.33a) is identical to the NMSE used in Chapter 9. The use of variance ensures that adding a bias to $\hat{f}(n_1, n_2)$ will not affect the NMSE.

10.3.1 Pulse Code Modulation (PCM)

The simplest waveform coding method is the basic pulse code modulation (PCM) system, in which the image intensity $f(n_1, n_2)$ is quantized by a uniform quantizer. The basic PCM system is shown in Figure 10.17. The image intensity $f(n_1, n_2)$ that has been quantized is denoted by $\hat{f}(n_1, n_2)$ in the figure. The PCM system can be used not only to code image intensities, but also to code transform coefficients and image model parameters. However, it was first used, and is still extensively used, in coding waveforms. Therefore, a PCM system without additional specification is regarded as a waveform coder. This also applies to other waveform coders, such as delta modulation (DM) and differential pulse code modulation (DPCM), which will be discussed later.

The basic PCM system in Figure 10.17 is typically used in obtaining an original digital image $f(n_1, n_2)$ from an analog image in most digital image processing applications. The spatial resolution of a digital image $f(n_1, n_2)$ is primarily determined by its size (number of pixels). The size of $f(n_1, n_2)$ is chosen on the basis of the resolution desired in a given application. A digital image of 1024 × 1024 pixels has resolution comparable to that of 35 mm film. A digital image of 512 × 512 pixels has resolution comparable to that of a TV frame (two fields). Digital images of 256 × 256 pixels and 128 × 128 pixels are also used in such applications as video telephone. As we decrease an image's size, its resolution is decreased, and details begin to disappear. This was illustrated in Figure 1.11. The bit rate used in the original monochrome digital image in most applications is 8 bits/pixel. Except where a very accurate representation of the original analog image is required, such as in medical image processing, the basic PCM system at 8 bits/pixel preserves sufficient quality and intelligibility for most applications. A digital image with a bit rate of 8 bits/pixel will be considered an original image in our discussions of monochrome image coding.

The bit rate used in our discussion is expressed in bits/pixel. It is important to note that the measure bits/pixel can be misleading. For example, if we obtain a digital image by sampling the analog image at a rate much higher than can be

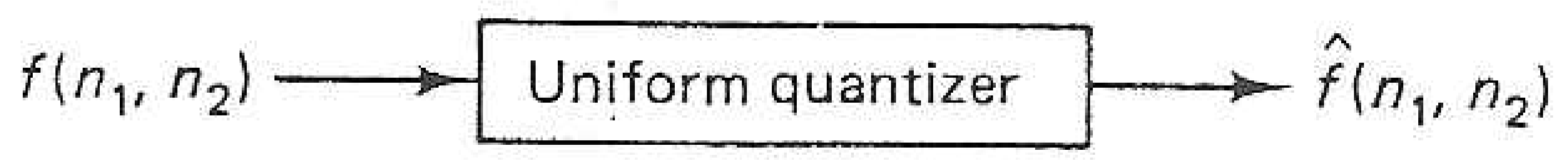


Figure 10.17 Basic pulse code modulation system.

5.3.4 Differential Predictive Coding

Differential predictive coding works by predicting the next pixel value based on the previous values and encoding the difference between the predicted value and the actual value (for analog signals, this is also called differential pulse code modulation or DPCM). This technique takes advantage of the fact that adjacent pixels are highly correlated, which means that the difference between adjacent pixels is typically small. Because this difference is small, it will take only a small number of bits to represent it. The use of a predictor allows us to further reduce the amount of information to be encoded. By using a simple prediction equation, we can estimate the next pixel value and then encode only the difference between the estimate and the actual value. This error is then quantized, to further reduce the data and to optimize visual results, and can then be coded.

A block diagram of this process is shown in Figure 5.3-7, where we can see that the predictor must be in the feedback loop so that it matches the decompression system. The system must be initialized by retaining the first value(s) without any compression in order to calculate the first prediction. From the block diagram, we have the following:

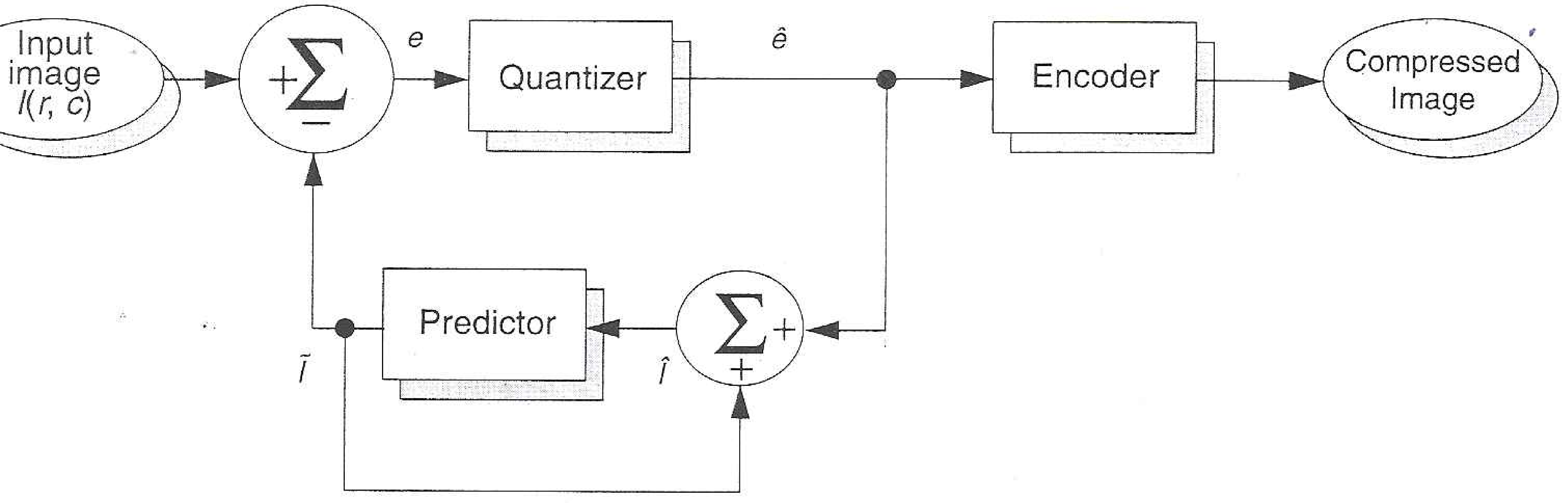
$$\begin{aligned} \tilde{I} &= \text{the predicted next pixel value} \\ \hat{I} &= \text{the reconstructed pixel value} \\ e &= I - \tilde{I} = \text{error} \\ \hat{e} &= \hat{I} - \tilde{I} = \text{quantized error} \end{aligned}$$

The prediction equation is typically a function of the previous pixel(s) and can also include global or application-specific information.

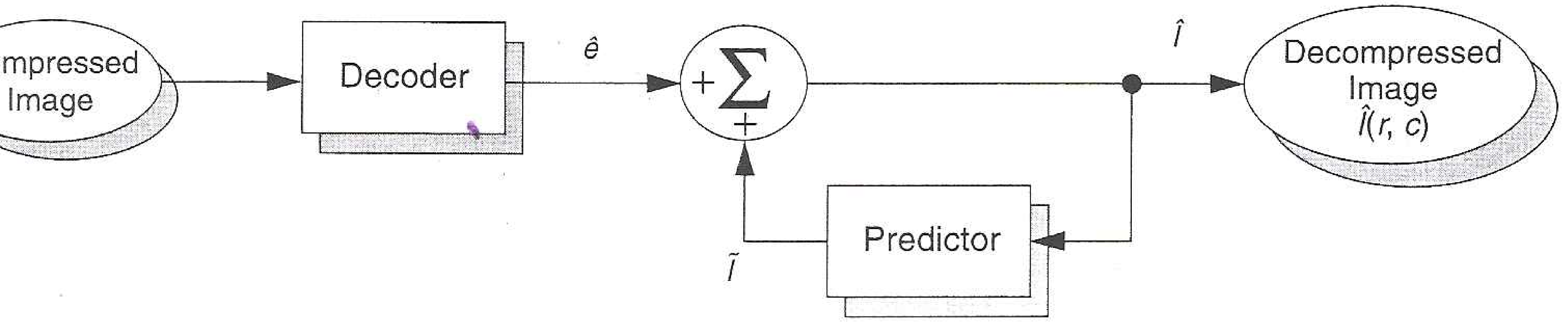
The theoretically optimum predictor, using only the previous value and based on minimizing mean-squared error between the original and the decompressed image, is given by

$$\begin{aligned} \tilde{I}(r, c+1) &= \rho\hat{I}(r,c) + (1-\rho)\,\overline{I}(r,c) \\ \text{where } \overline{I}(r,c) &= \text{the average value for the image} \\ \rho &= \text{the normalized correlation between pixel values} \end{aligned}$$

ure 5.3-7 Differential Predictive Coding



a. Compression.



b. Decompression

For most images ρ is between 0.85 and 0.95. When the next pixel value has been predicted, the error is calculated:

$$e(r, c+1) = I(r, c+1) - \tilde{I}(r, c+1)$$

This error signal is then quantized such that

$$\hat{e}(r, c+1) = \hat{I}(r, c+1) - \tilde{I}(r, c+1)$$

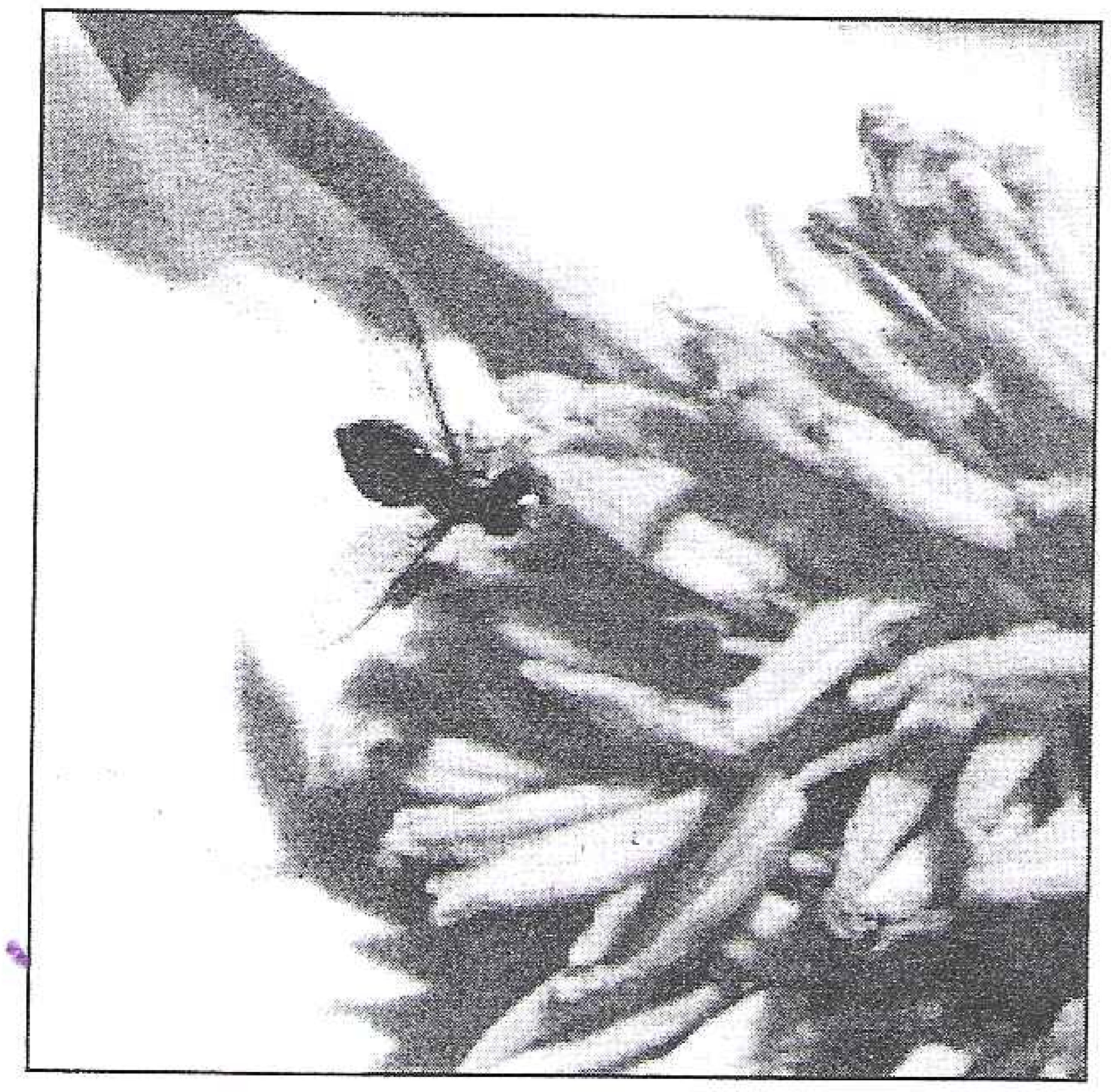
This quantized error can then be encoded using a lossless encoder, such as a Huffman coder. It should be noted that it is important that the predictor uses the same values during both compression and decompression, specifically the reconstructed values and not the original values (see Figure 5.3-7). In Figure 5.3-8 we see the results from using the original image values in the prediction, compared to using the reconstructed (decompressed) pixel values in the predictor. With these examples the quantization used was simply truncation ("clipping").

The prediction equation can be one-dimensional or two-dimensional, that is, it can be based on previous values in the current row only or on previous rows also (see Figure 5.3-9). The following prediction equations are typical examples of those used in practice, with the first being one-dimensional and the next two being two-dimensional:

$$\begin{aligned}
\tilde{I}(r, c+1) &= 0.97\hat{I}(r, c) \\
\tilde{I}(r, c+1) &= 0.49\hat{I}(r, c) + 0.49\hat{I}(r-1, c+1) \\
\tilde{I}(r, c+1) &= 0.74\hat{I}(r, c) + 0.74\hat{I}(r-1, c+1) - 0.49\hat{I}(r-1, c)
\end{aligned}$$

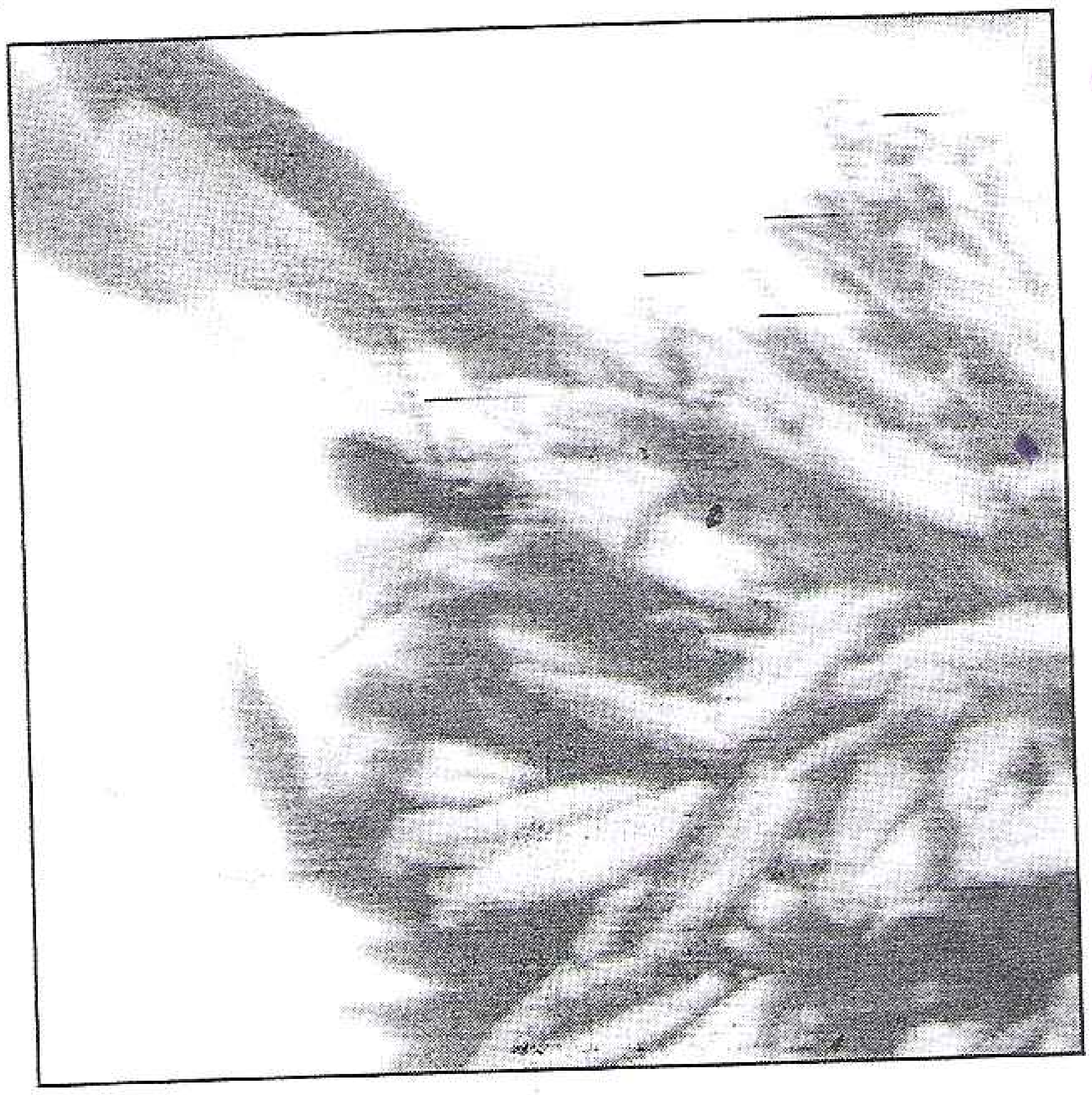
Using more of the previous values in the predictor increases the complexity of the computations for both compression and decompression, and it has been determined

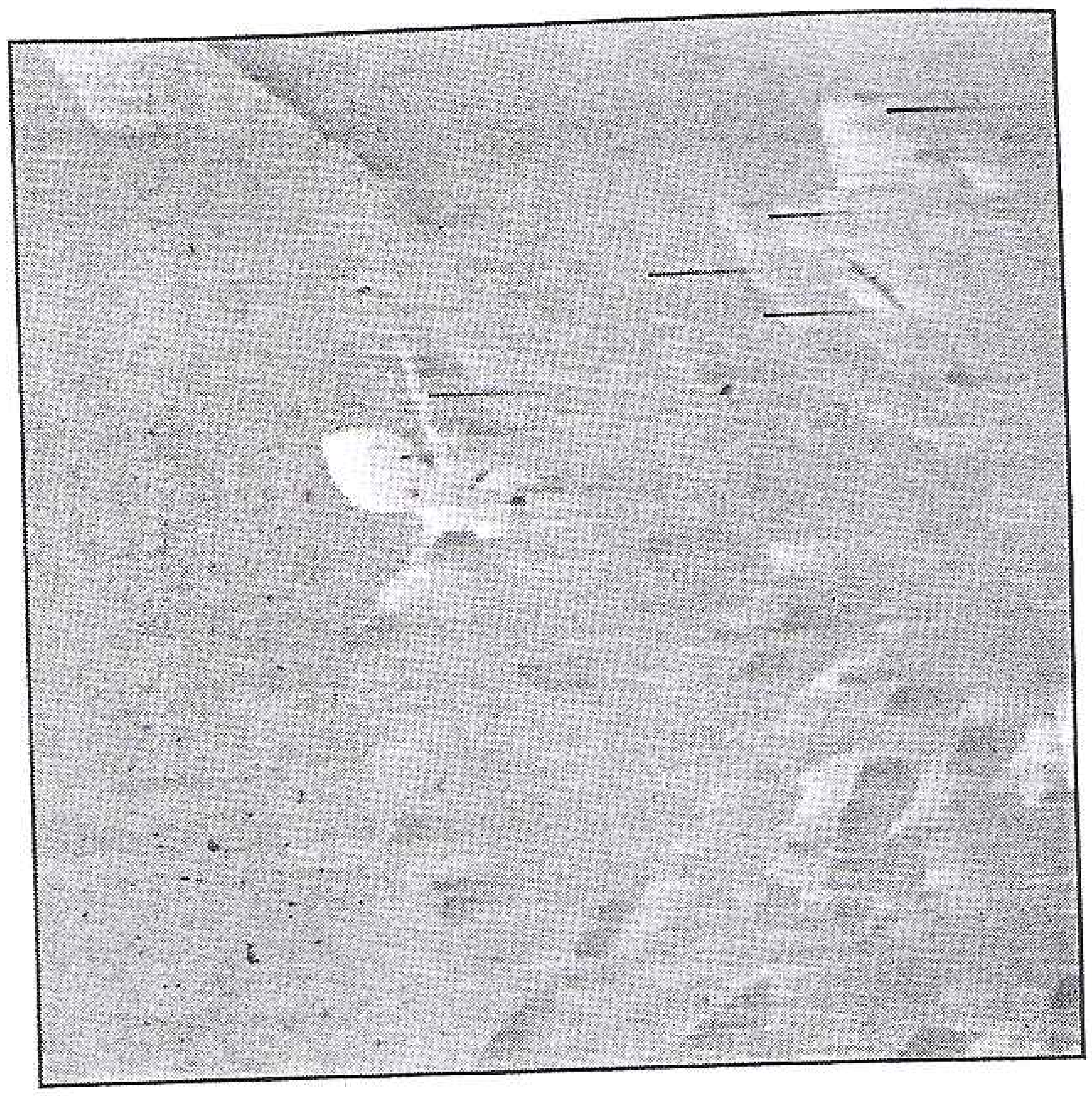
igure 5.3-8 DPC Example

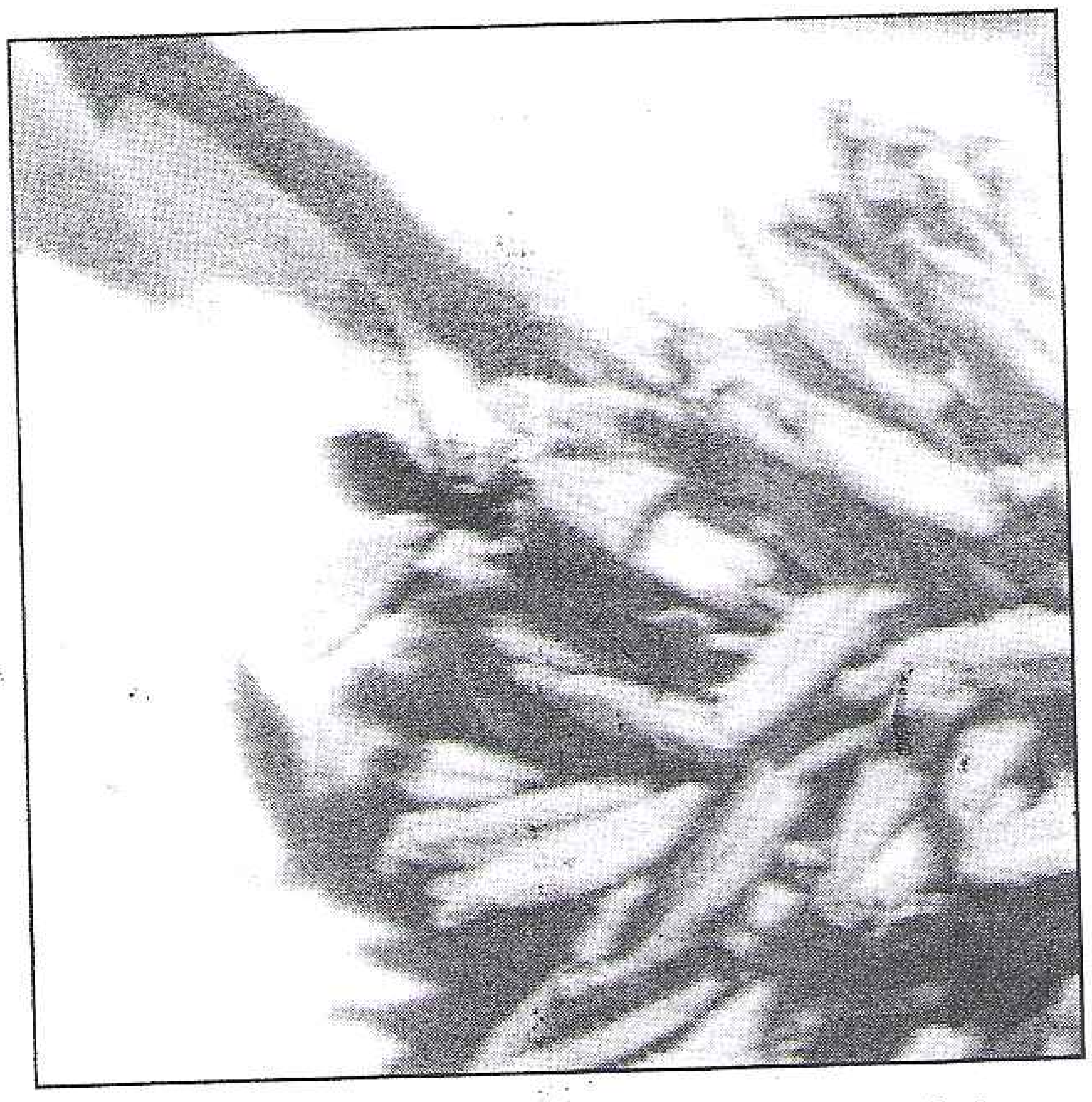


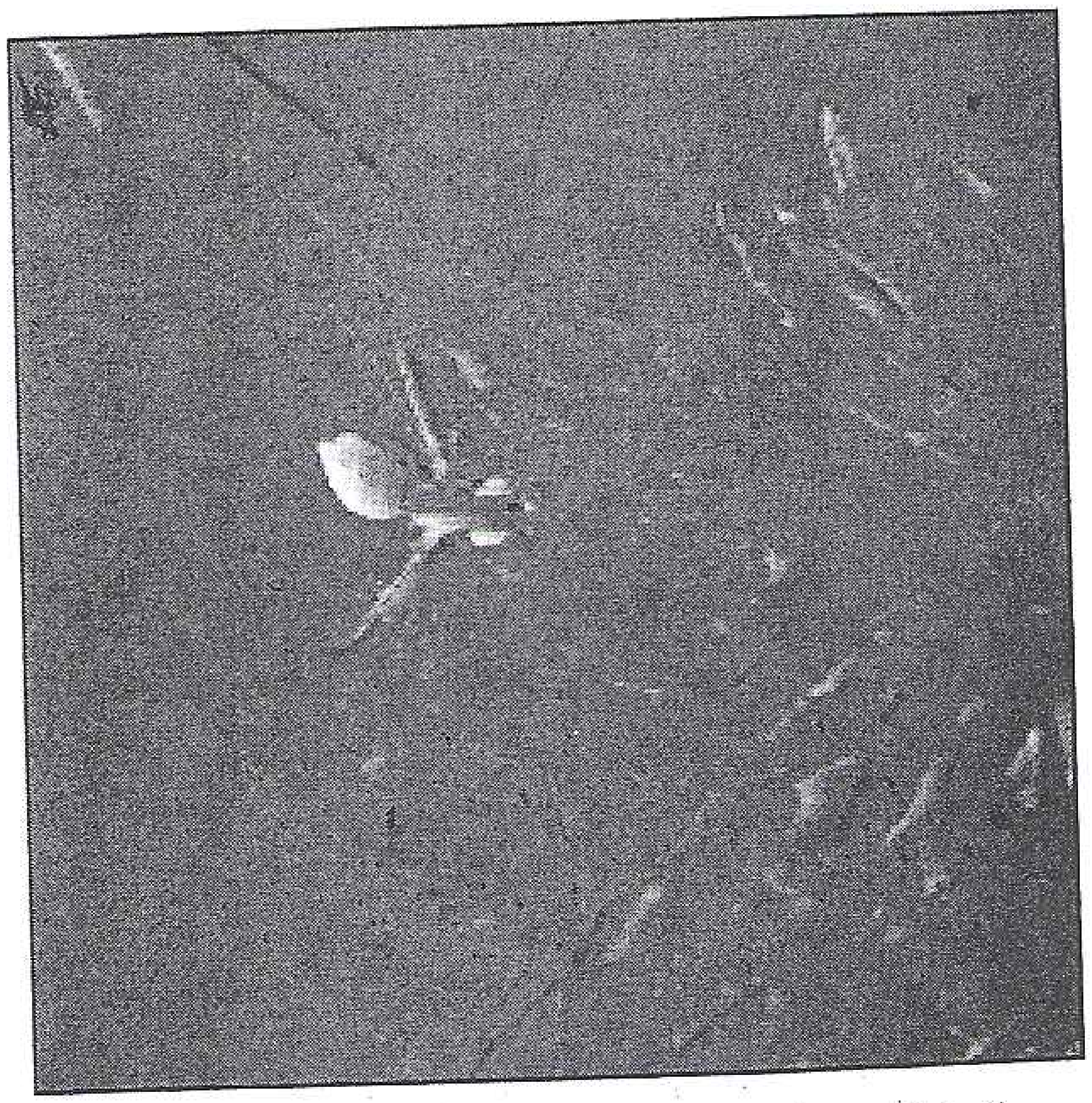
a. Original image.

Figure 5.3-8 (Continued)


b. DPC using original values in predictor, clipping to the maximum, 5 bits/pixel, normalized correlation .90.


c. Error image of (b), multiplied to show detail.


d. DPC using reconstructed values in predictor, clipping to the maximum, 5 bits/pixel, normalized correlation .90.


e. Error image of (d), multiplied to show detail.

that using more than three of the previous values provides no significant impro
ment in the resulting image.

The results of DPC can be further improved by using an optimal quantizer, s
as the Lloyd-Max quantizer, instead of simply truncating the resulting error.
Lloyd-Max quantizer assumes a specific distribution for the prediction error. Ass

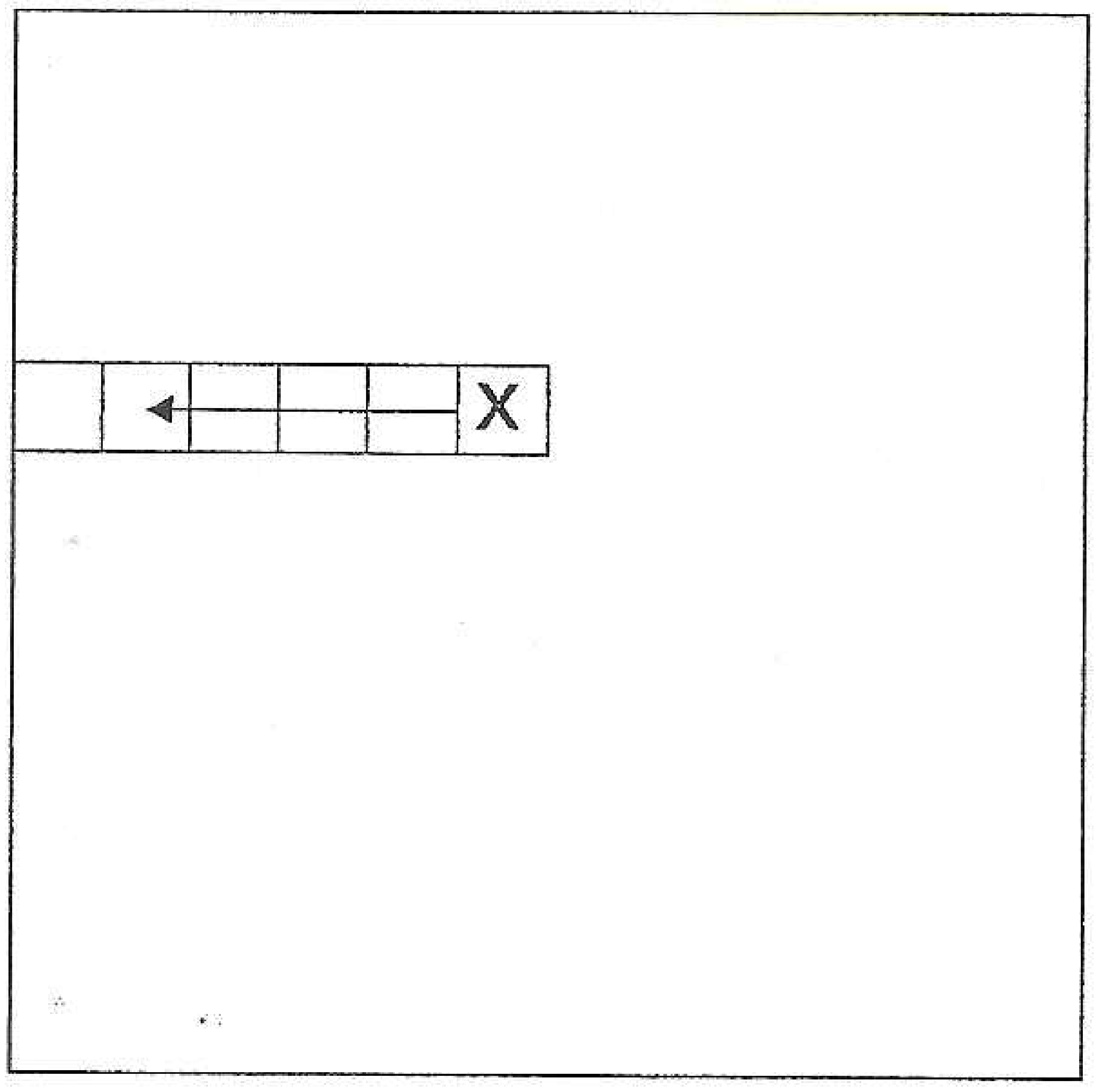
The results of DPC can be further improved by using an optimal quantizer, such as the Lloyd-Max quantizer, instead of simply truncating the resulting error. The Lloyd-Max quantizer assumes a specific distribution for the prediction error. Assuming a 2-bit code for the error and a laplacian distribution for the error, the Lloyd-Max quantizer is defined as follows (see Figure 5.3-10):

ERROR RANGE		QUANTIZED VALUE
$0 \leq e < 1.102\sigma$	$\rightarrow$	$+0.395\sigma$
$1.102\sigma \leq e < \infty$	$\rightarrow$	$+1.81\sigma$
$-1.102\sigma \leq e < 0$	$\rightarrow$	-0.395σ
$-1.102\sigma \leq e < -\infty$	$\rightarrow$	-1.81σ

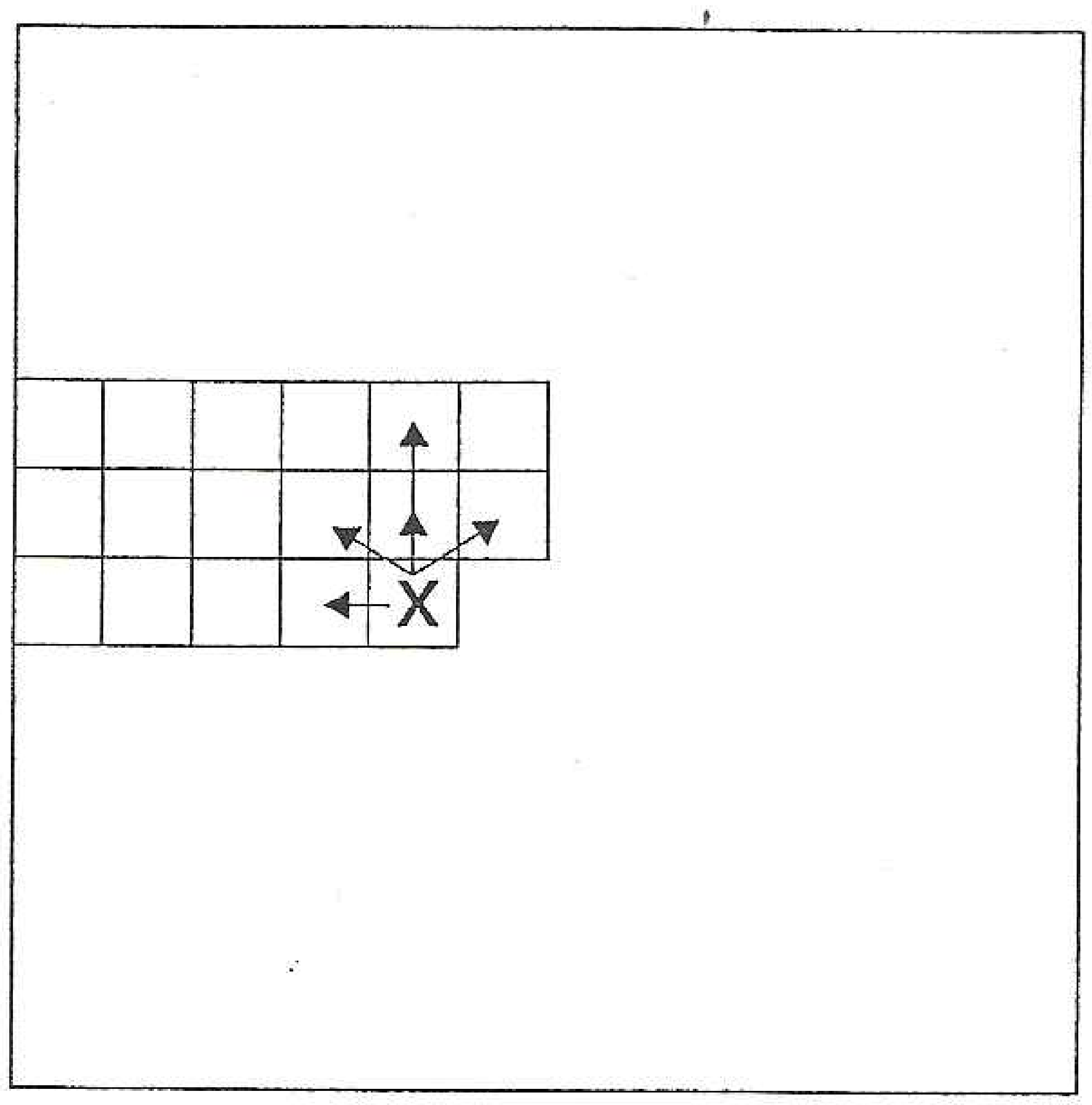
where σ = the standard deviation of the error distribution

Tables for the coefficients for n-bit codes can be found in the references. For most images, the standard deviation σ for the error signal is between 3 and 15. In Figure 5.3-11 is a comparison of using the Lloyd-Max quantizer and truncation as a quantization method. Figure 5.3-12 shows the error images and decompressed images using different bit rates for DPC compression (with Lloyd-Max quantization and a one-dimensional predictor).

Figure 5.3-9 DPC Predictor Dimensions

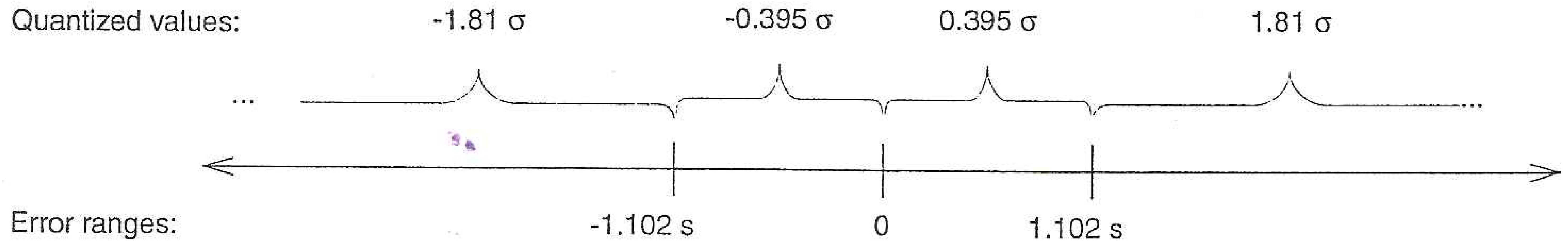


a. One-dimensional predictor, based on current row only; x = current pixel.



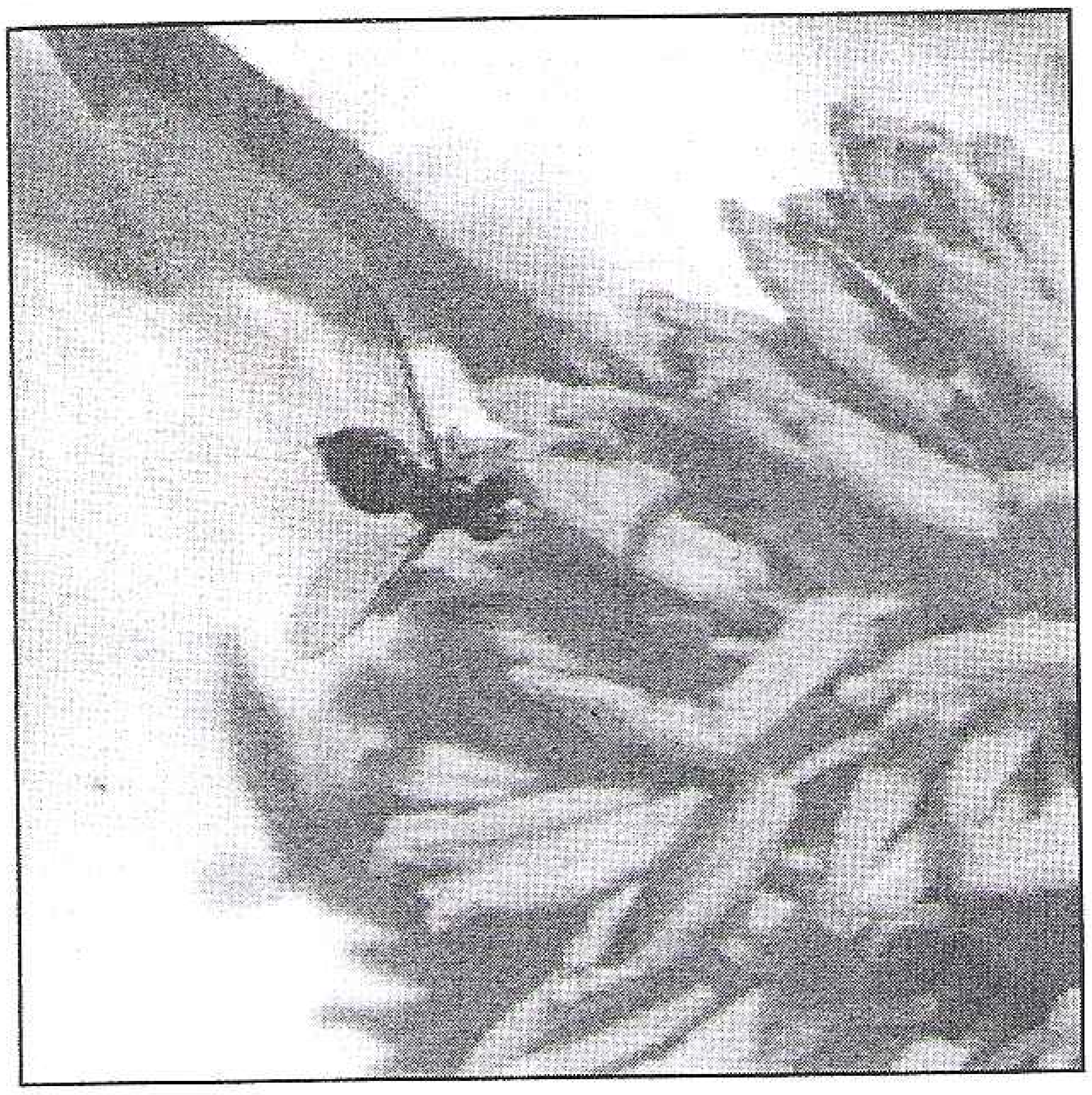
b. Two-dimensional predictor based on current and previous row or rows.

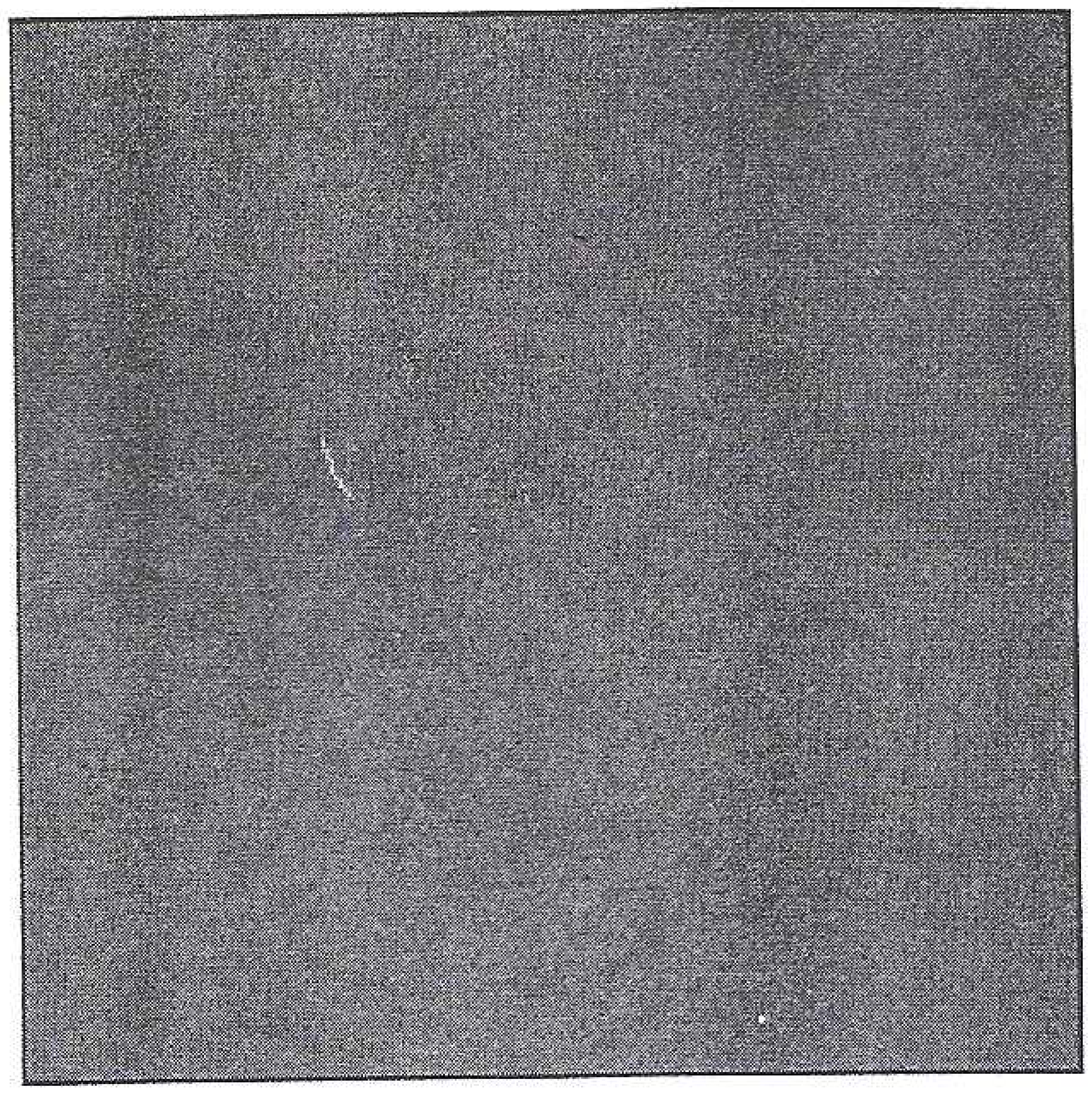
Figure 5.3-10 Lloyd-Max Quantizer

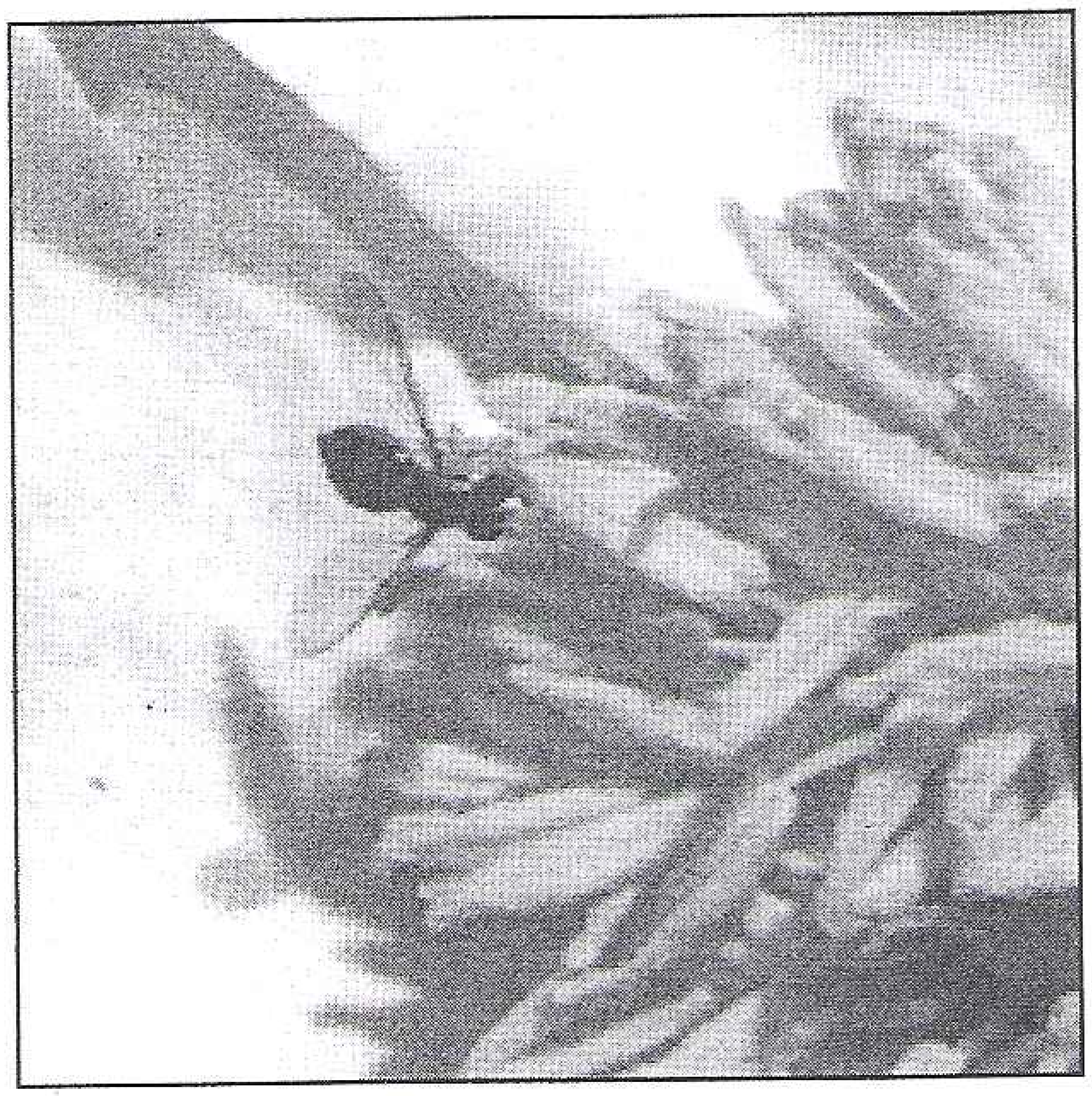


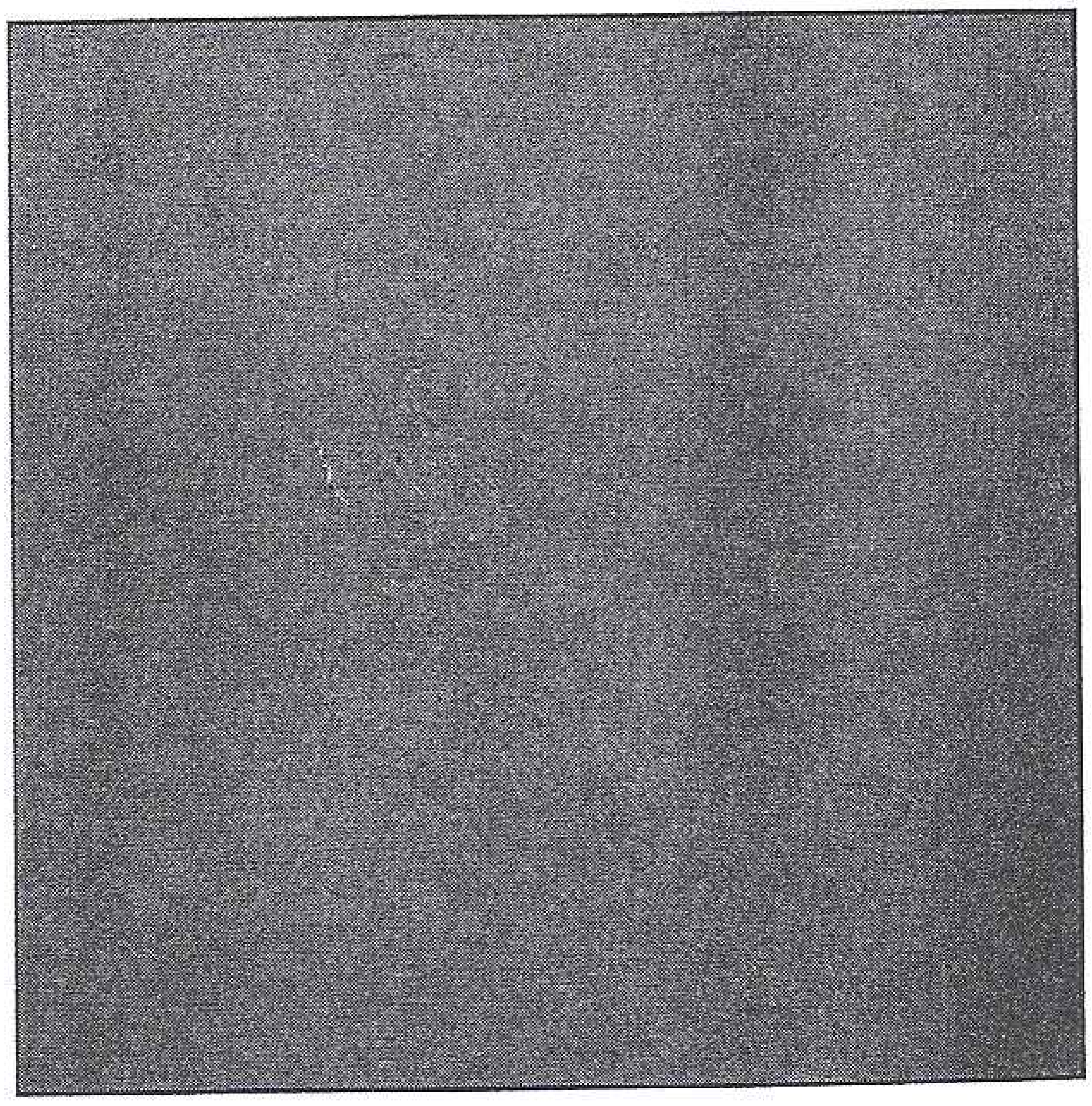
a. 2-bit Lloyd-Max quantizer with laplacian error distribution.

Figure 5.3-12 (Continued)


h. Lloyd-Max quantizer, using 4 bits/pixel, normalized correlation = 0.90, with standard deviation = 10.


i. Error image for (h).


j. Lloyd-Max quantizer, using 5 bits/pixel, normalized correlation = 0.90, with standard deviation = 10.


k. Error image for (j).

for the encoding process, and consists of any number of operations that are application specific. After the compressed file has been decoded, postprocessing can be performed to eliminate some of the potentially undesirable artifacts brought about by the compression process. Often, many practical compression algorithms are a combination of a number of different individual compression techniques.

The compressor can be further broken down into stages as illustrated in Figure 5.1-3. The first stage in preprocessing is data reduction. Here, the image data can be reduced by gray-level and/or spatial quantization, or they can undergo any desired image enhancement (for example, noise removal) process. The second step in preprocessing is the mapping process, which maps the original image data into another mathematical space where it is easier to compress the data. Next, as part of the encoding process, is the quantization stage, which takes the potentially continuous data from the mapping stage and puts it in discrete form. The final stage of encoding involves coding the resulting data, which maps the discrete data from the quantizer onto a code in an optimal manner. A compression algorithm may consist of all the stages, or it may consist of only one or two of the stages.

The decompressor can be further broken down into the stages shown in Figure 5.1-4. Here the decoding process is divided into two stages. The first, the decoding stage, takes the compressed file and reverses the original coding by mapping the codes

Figure 5.1-3 The Compressor

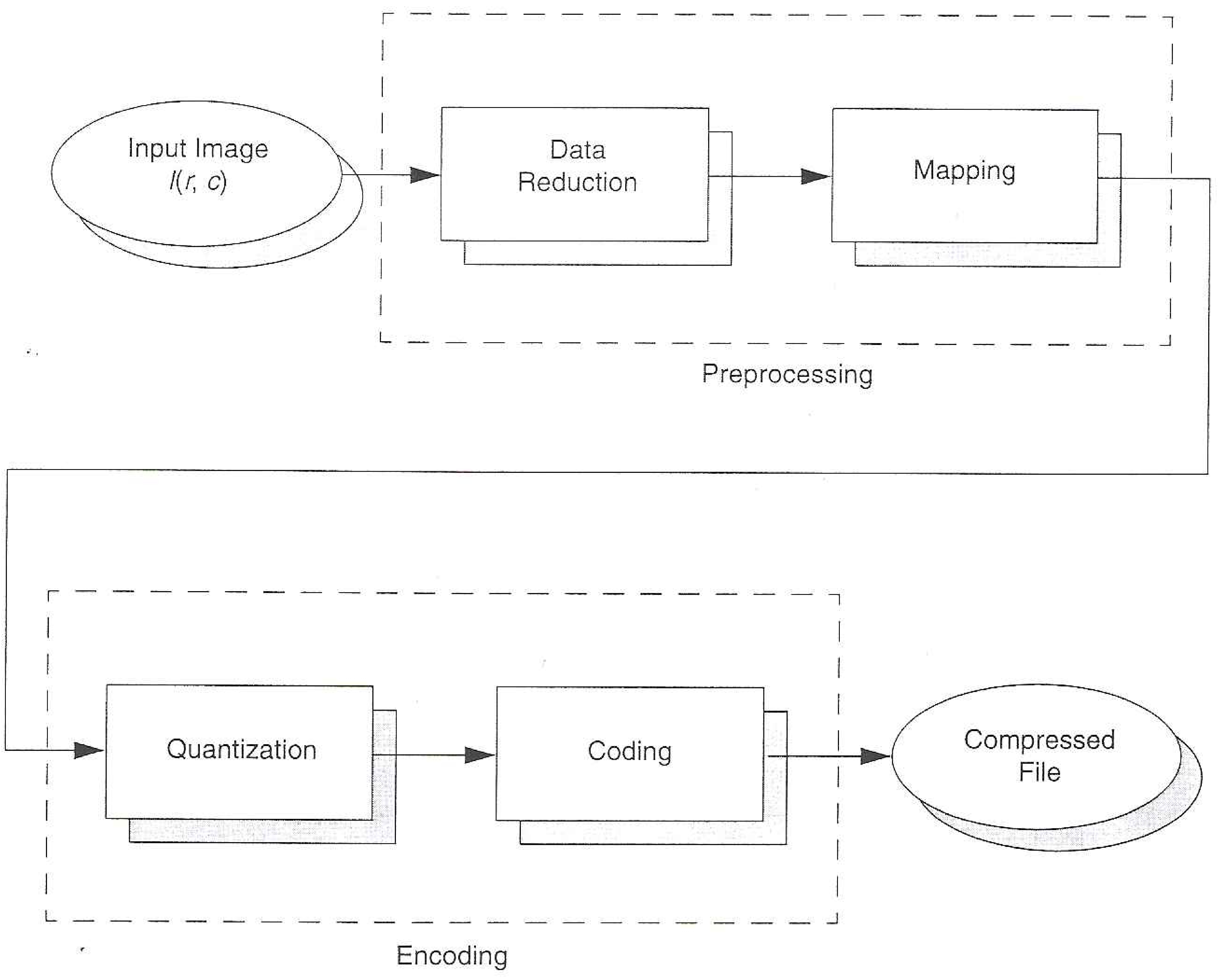
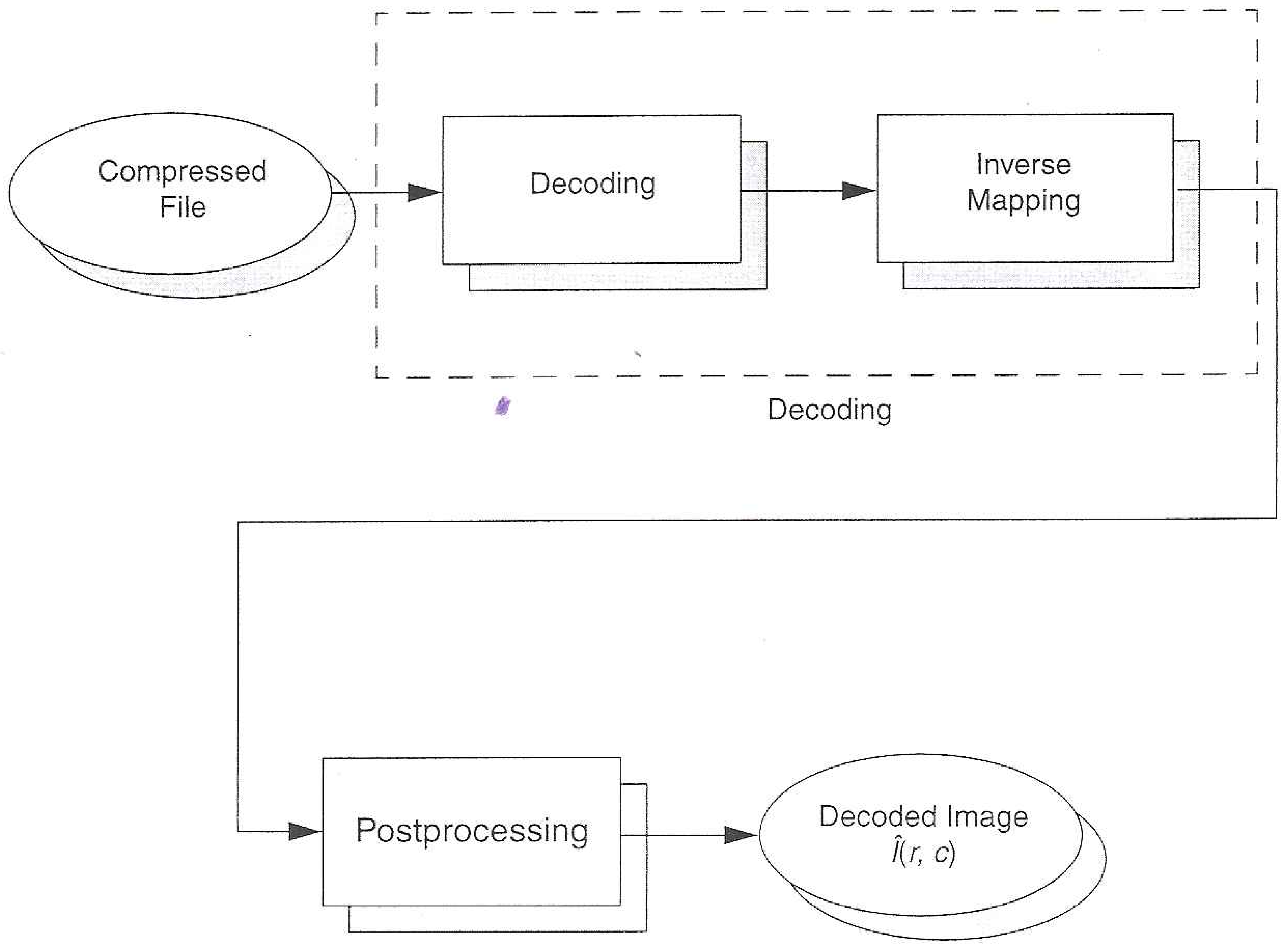


Figure 5.1-4 The Decompressor



to the original, quantized values. Next, these values are processed by a stage that performs an inverse mapping to reverse the original mapping process. Finally, the image may be postprocessed to enhance the look of the final image. In some cases this may be done to reverse any preprocessing, for example, enlarging an image that was shrunk in the data reduction process. In other cases the postprocessing may simply enhance the image to ameliorate any artifacts from the compression process itself.

The development of a compression algorithm is highly application specific. During the preprocessing stage of compression, processes such as enhancement, noise removal, or quantization are applied. The goal of preprocessing is to prepare the image for the encoding process by eliminating any irrelevant information, where *irrelevant* is defined by the application. For example, many images that are for viewing purposes only can be preprocessed by eliminating the lower bit planes, without losing any useful information. In Figure 5.1-5 are shown the eight bit planes corresponding to an 8-bit image. Each bit plane is shown as an image by using white if the corresponding bit is a 1 and black if the bit is a 0. Here we see that the lower bit planes contain little information and can be eliminated with no significant information loss.

The mapping process is important because image data tend to be highly correlated. What this means is that there is a lot of redundant information in the data itself. Specifically, if the value of one pixel is known, it is highly likely that the adjacent pixel value is similar. By finding a mapping equation that decorrelates the data, this type of data redundancy can be removed. One method to do this is to find the difference between adjacent pixels and encode these values; this is called *differential* result of vector quantization at $\frac{1}{2}$ bit/pixel. The block size used is 4×4 pixels, and the codebook in this example was designed by using a variation of the K-means algorithm discussed in Section 10.1.2. The training data used are all the blocks of four different images.

10.4 TRANSFORM IMAGE CODING

In transform image coding, an image is transformed to a domain significantly different from the image intensity domain, and the transform coefficients are then coded. In low bit rate applications (below 1 or 2 bits/pixel) such as video conferencing, transform coding techniques with scalar quantization typically perform significantly better than waveform coding techniques with scalar quantization. They are, however, more expensive computationally.

Transform coding techniques attempt to reduce the correlation that exists among image pixel intensities more fully than do waveform coding techniques. When the correlation is reduced, redundant information does not have to be coded repeatedly. Transform coding techniques also exploit the observation that for typical images a large amount of energy is concentrated in a small fraction of the transform coefficients. This is called the *energy compaction property*. Because of this property, it is possible to code only a fraction of the transform coefficients without seriously affecting the image. This allows us to code images at bit rates below 1 bit/pixel with a relatively small sacrifice in image quality and intelligibility.

In coding transform coefficients, we can use any of the quantization methods discussed in Section 10.1. However, the method that has been used most frequently is scalar quantization, because of its simplicity. In this section, we will assume scalar quantization unless otherwise specified.

10.4.1 Transforms

A schematic diagram of a transform image coder is shown in Figure 10.41. At the transmitter, the image $f(n_1, n_2)$ is transformed and the transform coefficients $T_f(k_1, k_2)$ are quantized. The quantized $\hat{T}_f(k_1, k_2)$ are then coded. At the receiver, the codewords are decoded and the resulting quantized transform coefficients $\hat{T}_f(k_1, k_2)$ are inverse transformed to obtain the reconstructed image $\hat{f}(n_1, n_2)$.

Several properties are desirable in the transform. Since we compute the transform at the transmitter and the inverse transform at the receiver, it is important to have efficient ways to compute them. Transforms which have been considered for image coding are linear transforms that can be expressed as

$$T_f(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} f(n_1, n_2) a(n_1, n_2; k_1, k_2) \tag{10.56a}$$

$$f(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} T_f(k_1, k_2) b(n_1, n_2; k_1, k_2) \tag{10.56b}$$

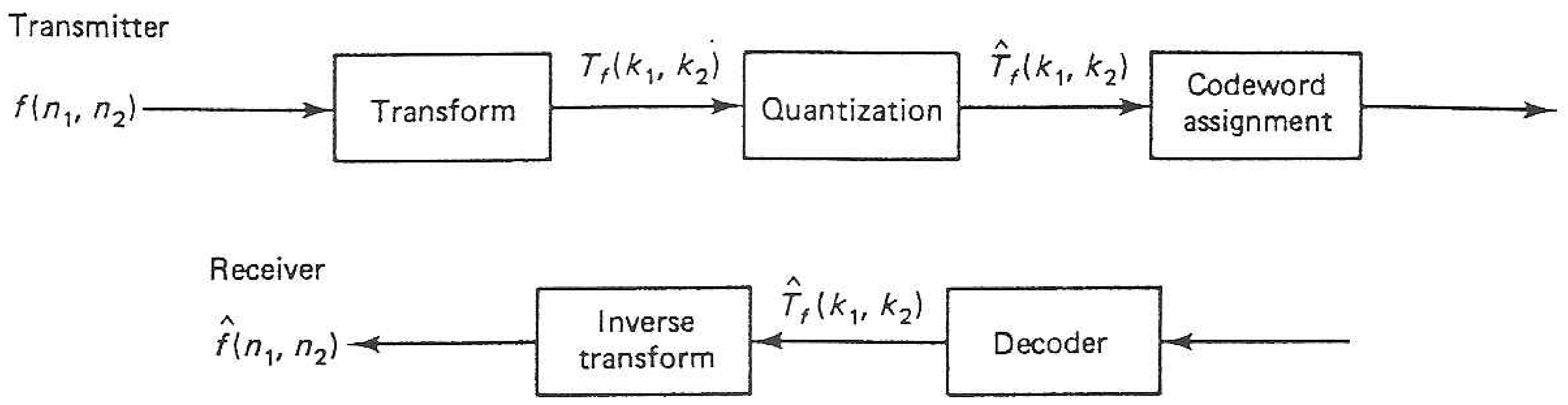


Figure 10.41 Transform image coder.

where $f(n_1, n_2)$ is an $N_1 \times N_2$-point sequence, $T_f(k_1, k_2)$ is also an $N_1 \times N_2$-point sequence and represents the transform coefficients, and $a(n_1, n_2; k_1, k_2)$ and $b(n_1, n_2; k_1, k_2)$ are the basis functions that satisfy (10.56). From (10.56), we can deduce that $f(n_1, n_2)$ is a linear combination of the basis functions $b(n_1, n_2; k_1, k_2)$ and that the transform coefficients $T_f(k_1, k_2)$ are the amplitudes of the basis functions in the linear combination. When the basis functions have some form of sinusoidal behavior, the transform coefficients can be interpreted as amplitudes of generalized spectral components. From computational considerations, the basis functions used in most transform image coders are separable, so (10.56) can be expressed as

$$T_f(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} f(n_1, n_2) a_R(n_1; k_1) a_C(n_2; k_2), \tag{10.57a}$$

$$f(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} T_f(k_1, k_2) b_R(n_1; k_1) b_C(n_2; k_2). \tag{10.57b}$$

An example of a transform in the form of (10.57) is the discrete Fourier transform (DFT) discussed in Section 3.2;

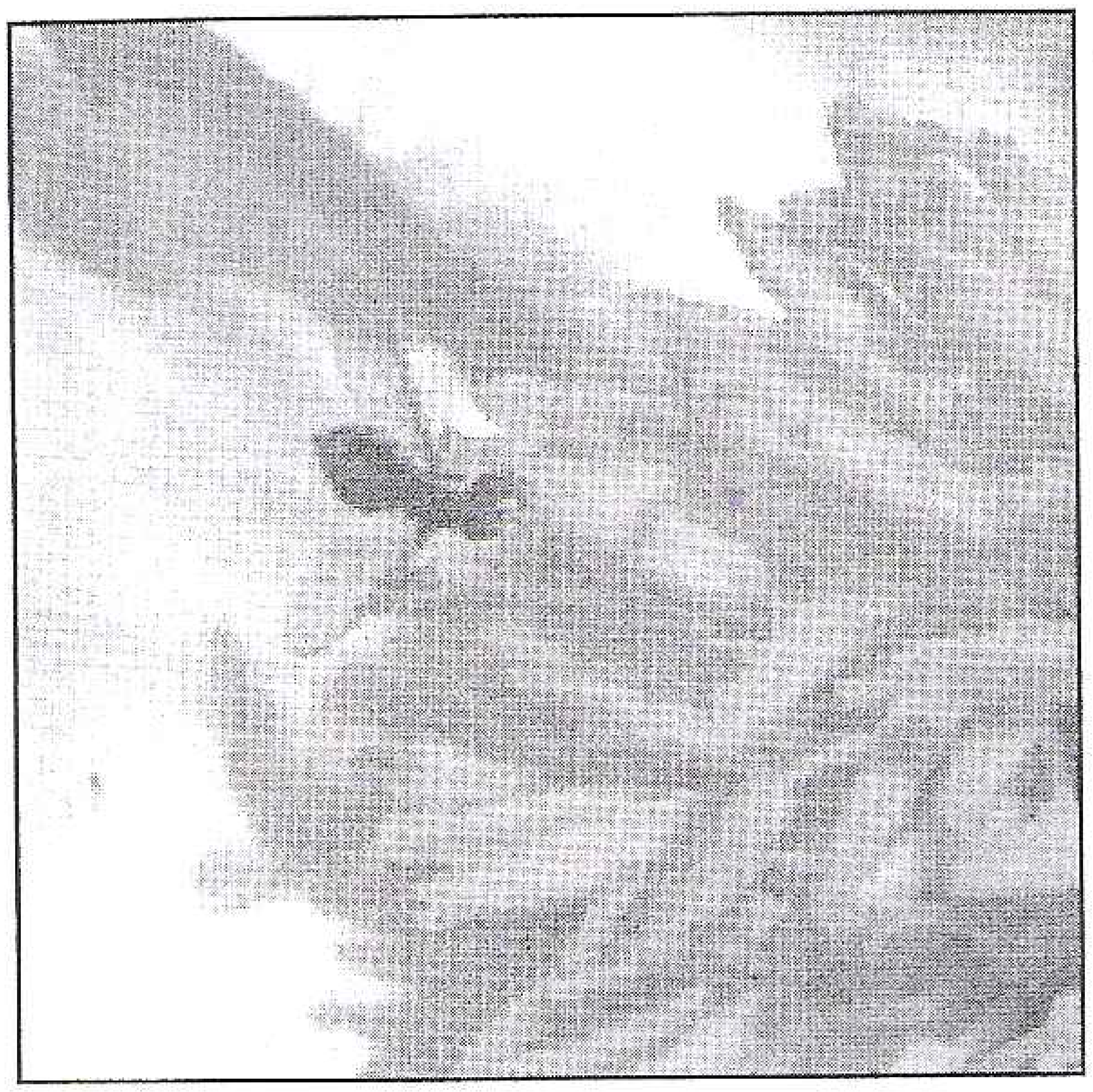
$$F(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} f(n_1, n_2) e^{-j(2\pi/N_1)k_1 n_1} e^{-j(2\pi/N_2)k_2 n_2} \tag{10.58a}$$

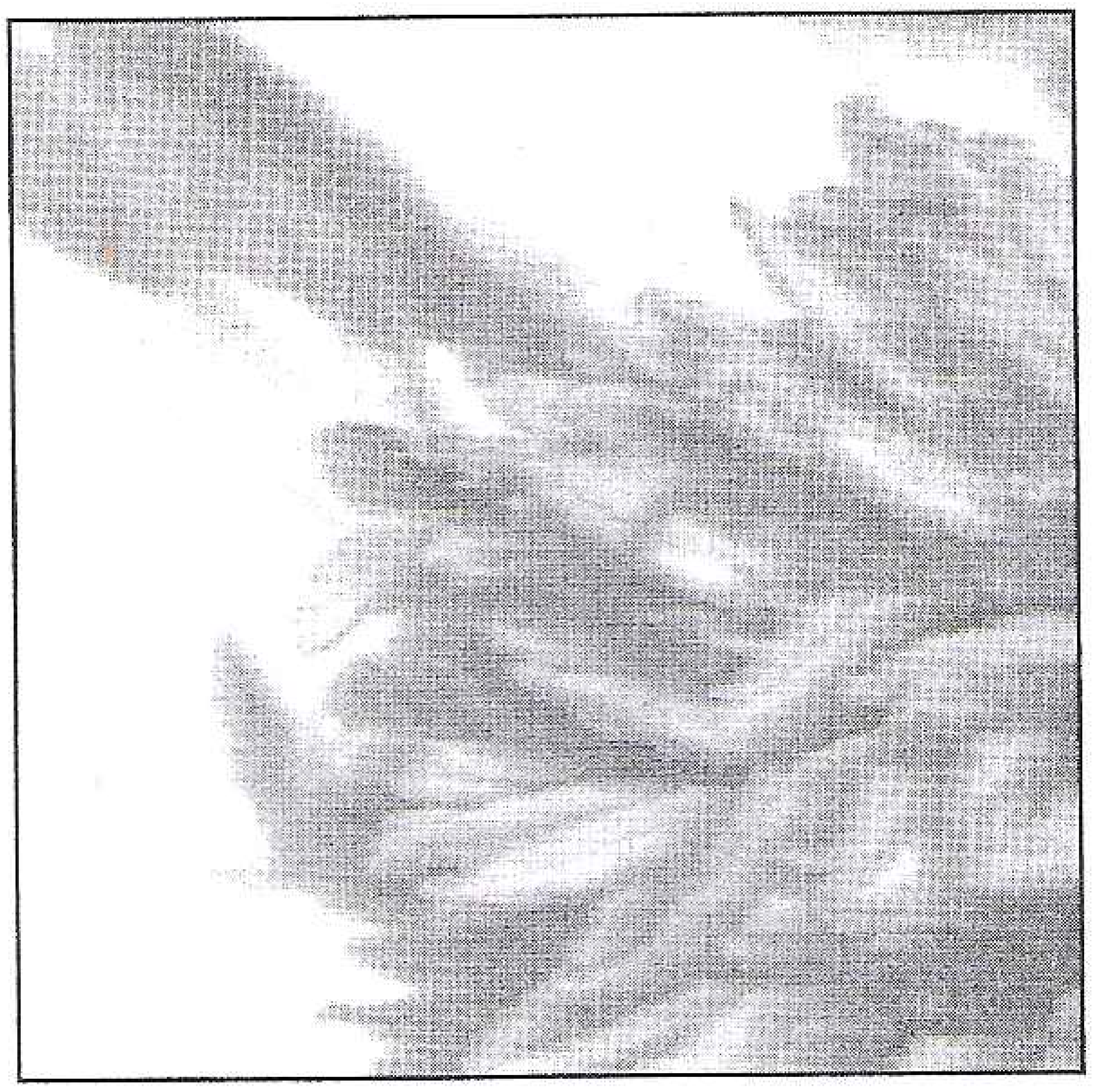
$$f(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} F(k_1, k_2) e^{j(2\pi/N_1)k_1 n_1} e^{j(2\pi/N_2)k_2 n_2}. \tag{10.58b}$$

When the basis functions are separable, the transform and inverse transform can be computed by row-column decomposition, in which 1-D transforms are computed row by row and then column by column. Row-column decomposition for computing a 2-D DFT was discussed in Section 3.4.1. Row-column decomposition can reduce the number of arithmetic operations by orders of magnitude compared to direct computation. In the case of a 512 × 512-point DFT computation, row-column decomposition alone reduces the number of arithmetic operations by a factor of several hundred (see Section 3.4.1). For transforms such as the DFT, additional computational savings can be obtained by exploiting the sinusoidal behavior of the basis functions.

Another desirable property of a transform is reduction of correlation among transform coefficients. We will refer to this property as the *correlation reduction*

Figure 5.3-11 (Continued)


d. Lloyd-Max quantizer, using 4 bits/pixel, normalized correlation = 0.90, with standard deviation = 10.


e. Truncation quantizer, using 4 bits/pixel, normalized correlation = 0.90.

5.3.5 Transform Coding

Transform coding is a form of block coding done in the transform domain. The image is divided into blocks, or subimages, and the transform is calculated for each block. Any of the previously defined transforms can be used, frequency (e.g., Fourier) or sequency (e.g., Walsh), but it has been determined that the discrete cosine transform (DCT) is optimal for most images. After the transform has been calculated, the transform coefficients are quantized and coded. The primary reason this method is effective is because the frequency/sequency transform of images efficiently puts most of the information into relatively few coefficients so that many of the high-frequency coefficients can be quantized to 0 (eliminated completely). This type of transform is really just a special type of mapping that uses spatial frequency concepts as a basis for the mapping. Remember that for image compression the whole idea of mapping the original data into another mathematical space is to pack the information (or energy) into as few coefficients as possible.

The simplest form of transform coding is achieved by filtering; we can simply eliminate some of the high-frequency coefficients. This alone will not provide much compression because the transform data are typically floating point and thus 4 or 8 bytes/pixel (compared to the original pixel data at 1 byte/pixel), so quantization and coding are applied to the reduced data. The quantization process is partially performed by what is referred as bit allocation. *Bit allocation* is determining the number of bits to be used to code each coefficient. Typically, more bits are used for lower-frequency components.

EXAMPLE 5-19

We have decided to use transform coding with a DCT on an image by dividing it into 4×4 blocks. The selected bit allocation can be represented by the following mask:

$$\begin{bmatrix} 8 & 6 & 4 & 1 \\ 6 & 4 & 1 & 0 \\ 4 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

where the numbers in the mask are the number of bits used to represent the corresponding transform coefficients (the upper-left corner corresponds to the zero-frequency coefficient, or average value, and the frequency increases to the right and down). This allows the lower frequencies less quantization (more resolution) because they have more bits to represent them.

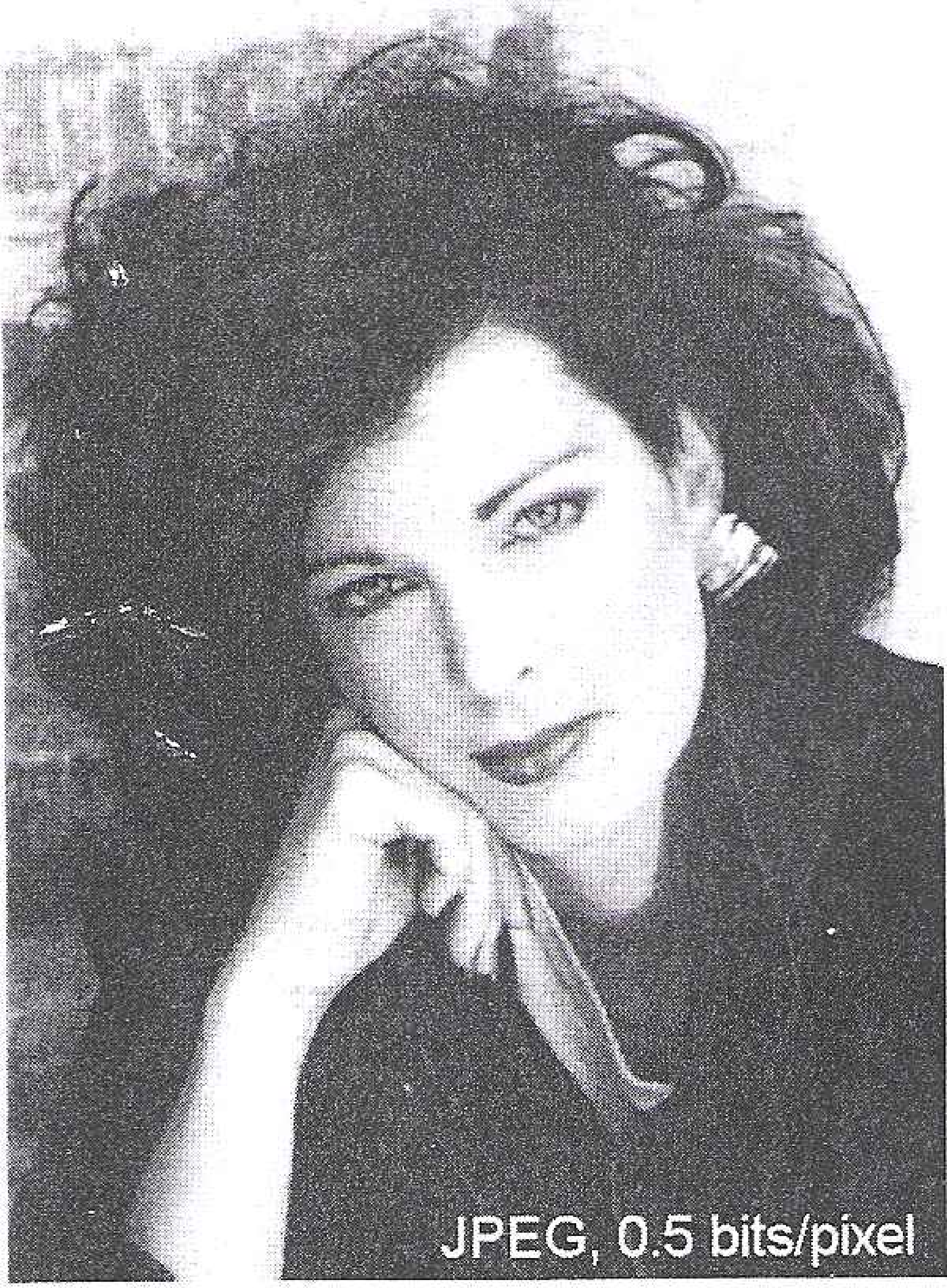
Next a quantization scheme, such as Lloyd-Max quantization, is applied. Because the zero-frequency coefficient for real images contains a large portion of the energy in the image and is always positive, it is typically treated differently than the higher-frequency coefficients. After they have been quantized, the coefficients can be coded using, for example, a Huffman coding method.

In addition to simple filtering, two particular types of transform coding have been widely explored: zonal and threshold coding. These two vary in the method they use for selecting the transform coefficients to retain (using ideal filters for transform coding selects the coefficients based on their location in the transform domain). *Zonal coding* involves selecting specific coefficients based on maximal variance, whereas *threshold coding* selects the coefficients above a specific value. In zonal coding, a zonal mask is determined for the entire image by finding the variance for each frequency component. This variance is calculated by using each subimage within the image as a separate sample and then finding the variance within this group of subimages (see Figure 5.3-13). The *zonal mask* is a bitmap of 1's and 0's, where the 1's correspond to the coefficients to retain, and the 0's correspond to the ones to eliminate. In practice, the zonal mask is often predetermined because the low-frequency terms tend to contain the most information, and hence exhibit the most variance. In threshold coding a different threshold mask is required for each block, which increases file size as well as algorithmic complexity.

One of the most commonly used image compression standards is primarily a form of transform coding. The Joint Photographic Experts Group met initially in 1987 under the auspices of the International Standards Organization (ISO) to devise an optimal still image compression standard. The result was a family of image compression methods for still images. The JPEG standard uses the DCT and 8×8 pixel blocks as the basis for compression. Before computing the DCT, the pixel values are level-shifted so that they are centered at zero.

EXAMPLE 5-20

A typical 8-bit image has a range of gray levels from 0 to 255. Level-shifting this range to be centered at zero involves subtracting 128 from each pixel value, so the resulting range is from -128 to 127.


JPEG, 0.5 bits/pixel

FFT-Real Data

- set $z = x + 0 * i$
  do FFT
  wasteful

- find FFT of two real vectors x,y
  Let

  $$z_j = x_j + iy_j$$

  Then

  $$\hat{x}_k = \frac{\hat{z}_k + \hat{z}_{N-k}}{2} \qquad \hat{y}_k = \frac{\hat{z}_k - \hat{z}_{N-k}}{2}$$

9 Inversion of the FFT of a real sequence

In this section we show how the symmetry property (5.6) of the FFT of a real sequence can be used to reduce by about one-half the computations involved in inverting the FFT. Although there are other methods of doing this than the method described below, this method is of interest because it uses no extra sines other than the set $\{S(m)\}_{m=0}^{(1/4)N}$ that was used for performing the FFT itself.

We will assume in this section that $W = e^{i2\pi/N}$. Let R_k and I_k stand for the real and imaginary parts of the FFT $\{F_k\}$. That is,

$$F_k = R_k + iI_k, \qquad (k = 0, 1, \ldots, N-1) \tag{9.1}$$

where F_k is defined by

$$F_k = \sum_{j=0}^{N-1} f_j W^{jk}.$$

Then, by the formula for DFT inversion,

$$f_j = \frac{1}{N} g_j, \qquad (j = 0, 1, \ldots, N-1) \tag{9.2}$$

where

$$g_j = \sum_{k=0}^{N-1} F_k (W^{jk})^*. \tag{9.3}$$

Using $W = e^{i2\pi/N}$ and (9.1) we have

$$\begin{aligned} g_j &= \sum_{j=0}^{N-1} (R_k + iI_k) e^{-i2\pi jk/N} \\ &= \sum_{k=0}^{N-1} (R_k + iI_k) \left(\cos \frac{2\pi jk}{N} - i \sin \frac{2\pi jk}{N} \right). \end{aligned} \tag{9.4}$$

Since $\{f_j\}$ consists of real numbers, so does $\{g_j\}$ because of (9.2). Consequently, only the real part of the sum in (9.4) is nonzero. Therefore, we must have

$$g_j = \sum_{k=0}^{N-1} R_k \cos \frac{2\pi jk}{N} + \sum_{k=0}^{N-1} I_k \sin \frac{2\pi jk}{N}. \tag{9.5}$$

Because of (5.6) we have

$$R_{N-k} = R_k, \qquad I_{N-k} = -I_k, \qquad (k = 1, \ldots, N-1). \tag{9.6}$$

It follows from (9.6) that, for each fixed j, the sequences

$$\left\{ R_k \cos \frac{2\pi jk}{N} \right\}_{k=1}^{N-1} \qquad \text{and} \qquad \left\{ I_k \sin \frac{2\pi jk}{N} \right\}_{k=1}^{N-1}$$

are even about $(1/2)N$. From formula (7.3) we obtain

$$\sum_{k=1}^{N-1} R_k \cos \frac{2\pi jk}{N} = R_{\frac{1}{2}N}(-1)^j + 2 \sum_{k=1}^{\frac{1}{2}N-1} R_k \cos \frac{2\pi jk}{N} \tag{9.7}$$

and

$$\sum_{k=1}^{N-1} I_k \sin \frac{2\pi jk}{N} = 2 \sum_{k=1}^{\frac{1}{2}N-1} I_k \sin \frac{2\pi jk}{N}. \tag{9.8}$$

Using (9.7) and (9.8) in (9.5) yields

$$g_j = R_0 + R_{\frac{1}{2}N}(-1)^j + 2 \sum_{k=1}^{\frac{1}{2}N-1} R_k \cos \frac{\pi jk}{\frac{1}{2}N} + 2 \sum_{k=1}^{\frac{1}{2}N-1} I_k \sin \frac{\pi jk}{\frac{1}{2}N}. \tag{9.9}$$

Formula (9.9) shows that $\{g_j\}_{j=0}^{N-1}$, and consequently $\{f_j\}_{j=0}^{N-1}$, can be generated from a $(1/2)N$-point fast cosine transform of

$$\{0, 2R_1, 2R_2, \ldots, 2R_{\frac{1}{2}N-1}\}$$

and a fast sine transform of

$$\{2I_1, 2I_2, \ldots, 2I_{\frac{1}{2}N-1}\}.$$

These fast cosine and fast sine transforms are even and odd about $(1/2)N$, respectively. Taking this symmetry into account, formula (9.9) applies to $j = 0$, $1, \ldots, N-1$.

As we noted above, an interesting feature of (9.9) is that to compute the fast cosine and fast sine transforms, *of order* $(1/2)N$, by the methods of Section 7, one needs only the *same* sines $\{S(m)\}_{m=0}^{(1/4)N}$ generated to calculate the FFT $\{F_k\}_{k=0}^{N-1}$. Therefore, inverting the FFT using (9.9) requires only these same sines. This is a useful memory savings.

Cosine Transform

Define

$$\alpha_0 = \sqrt{\frac{1}{N}}$$

$$\alpha_k = \sqrt{\frac{2}{N}} \; k = 1, 2, \ldots, N-1$$

Then

$$v_k = \alpha_k \sum_{n=0}^{N-1} u_n \cos\left[\frac{(2n+1)k\pi}{2N}\right] k = 0, 1, \ldots N-1$$

Note this uses **half** points in the interval.
Then the inverse is given by

$$u_n = \sum_{k=0}^{N-1} \alpha_k v_k \cos\left[\frac{(2n+1)k\pi}{2N}\right] \; n = 0, 1, \ldots N-1$$

To evaluate this fast we relate it to a FFT

$$v_k = \mathrm{Re}\left\{\alpha_k e^{-\frac{\pi i k}{2N}} \sum_{n=0}^{N-1} \tilde{u}_n e^{-\frac{2\pi i k n}{N}}\right\} = \mathrm{Re}\left\{\alpha_k w_{2N}^k DFT(u)\right\}$$

where

$$\tilde{u}_n = u_{2n}$$

$$\tilde{u}_{N-n-1} = u_{2n} + 1 \quad 0 \le n \le \frac{N}{2} - 1$$

Inverse

$$u_n = \sum_{k=0}^{N-1} \alpha_k v_k \cos\left[\frac{(2n+1)k\pi}{2N}\right] \; n = 0,1,\ldots N-1$$

Define

$$\tilde{u}_{2n} = \mathrm{Re}\left\{\left[\sum_{k=0}^{N-1} \alpha_k v_k e^{\frac{k\pi i}{2N}}\right] e^{\frac{2\pi i k n}{N}}\right\} \quad n = 0,1,\ldots \frac{N}{2} - 1$$

Then

$$u_{2n} = \tilde{u}_{2n}$$

$$u_{2n+1} = \tilde{u}_{2(N-1-n)}$$

Then

$$\hat{u}_k = \alpha_k \sum_{n=0}^{\frac{N}{2}-1} \left\{ u_{2n} \cos\left[\frac{(4n+1)k\pi}{2N}\right] + u_{2n+1} \cos\left[\frac{(4n+3)k\pi}{2N}\right] \right\}$$

$$= \alpha_k \sum_{n=0}^{\frac{N}{2}-1} \left\{ \overline{u}_n \cos\left[\frac{(4n+1)k\pi}{2N}\right] + \overline{u}_{N-n-1} \cos\left[\frac{(4n+3)k\pi}{2N}\right] \right\}$$

$$\text{Let} \quad n' = N - n - 1 \qquad n = N - n' - 1$$

$$= \alpha_k \sum_{n=0}^{\frac{N}{2}-1} \overline{u}_n \cos\left[\frac{(4n+1)k\pi}{2N}\right] + \sum_{n'=\frac{N}{2}}^{N-1} \overline{u}_{n'} \cos\left[\frac{(4(N-n')-1)k\pi}{2N}\right]$$

$$= \alpha_k \sum_{n=0}^{N-1} \overline{u}_{n'} \cos\left[\frac{(4n+1)k\pi}{2N}\right]$$

$$= \mathrm{Re}\left\{ \alpha_k e^{-\frac{2\pi ikn}{N}} \sum_{n=0}^{N-1} \overline{u}_{n'} \cos\left[\frac{(4n+1)k\pi}{2N}\right] \right\}$$

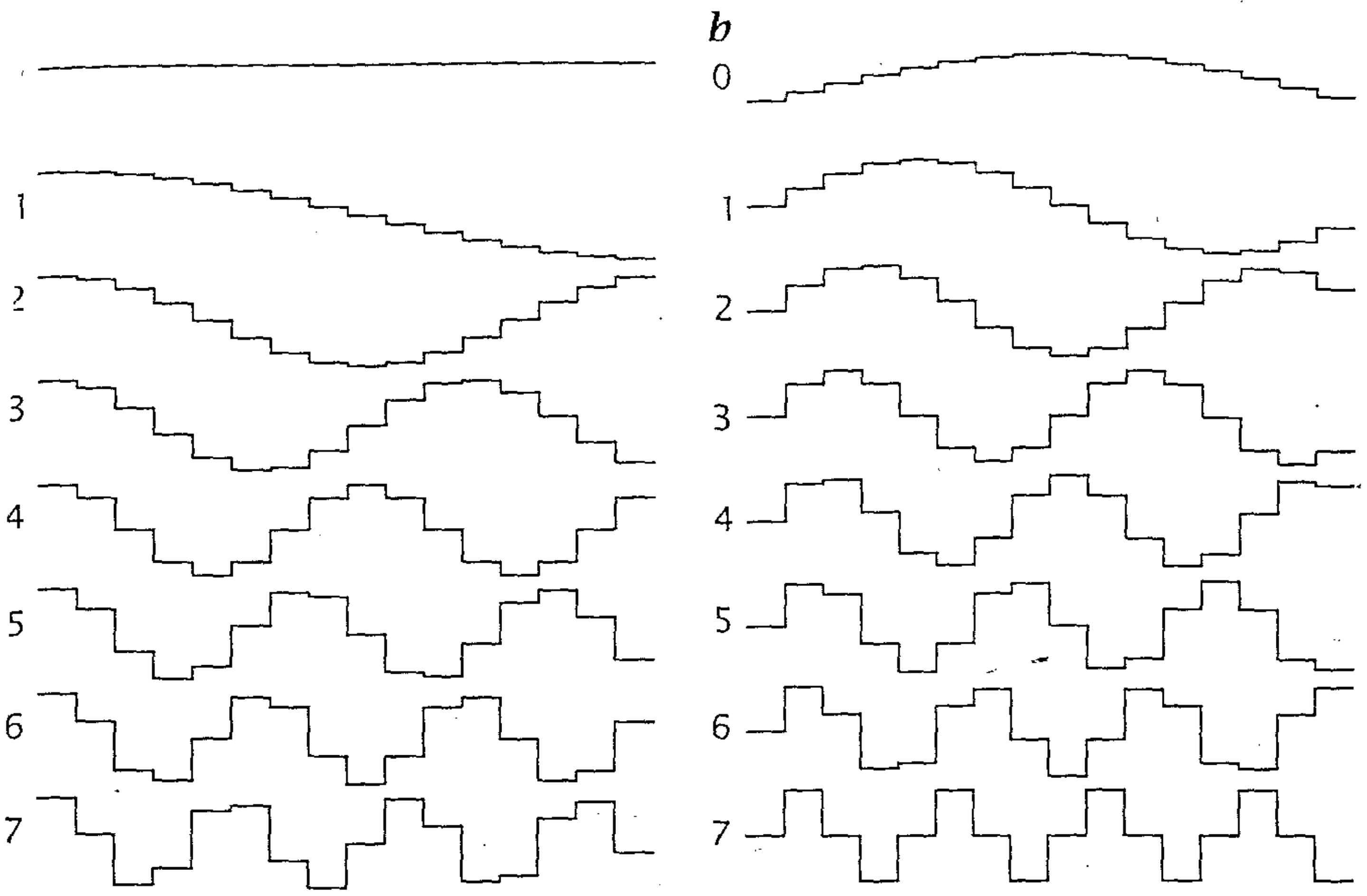
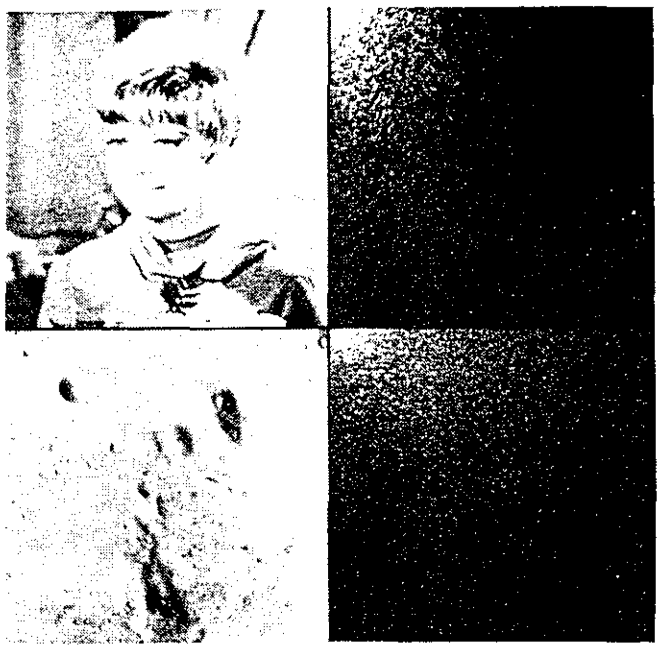


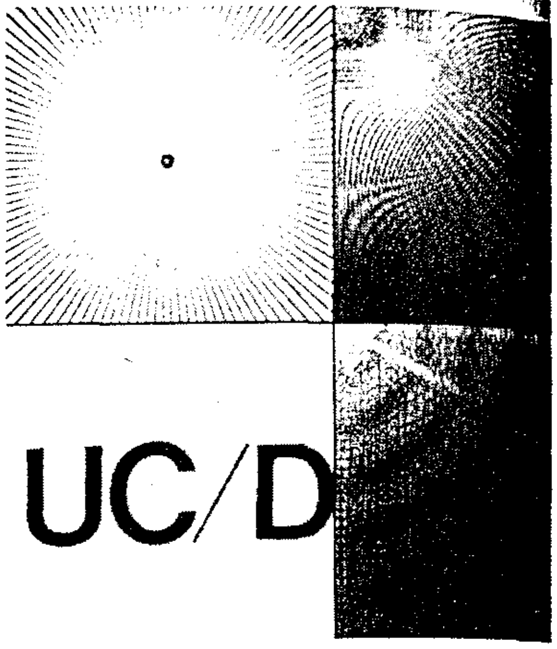
Figure 2.18: First 8 base functions of one-dimensional unitary transforms for $N = 16$*:* ***a*** *cosine transform and* ***b*** *sine transform.*

The cosine and sine functions only span the subspaces of the even and odd functions, respectively. Basis vectors with the missing symmetry can be generated, however, if trigonometric functions with half-integer wavelengths are added. This is equivalent to doubling the base wavelength. Consequently, the kernels for the *cosine* and *sine transforms* in an M-dimensional vector space are

$$\begin{aligned} C_{nv} &= \cos\left(\frac{\pi n v}{N}\right), \\ S_{nv} &= \sin\left(\frac{\pi n(v+1)}{N}\right). \end{aligned} \tag{2.44}$$

of the 1-D cosine and sine func-


(a) Cosine transform examples of monochrome images;



(b) Cosine transform examples of binary im

Figure 5.11

3. The cosine transform is a fast transform. The cosine transform of a vecto elements can be calculated in $O(N \log_2 N)$ operations via an N-poin [19]. To show this we define a new sequence $\tilde{u}(n)$ by reordering the eve odd elements of $u(n)$ as

$$\left.\begin{aligned}\tilde{u}(n) &= u(2n)\\ \tilde{u}(N-n-1) &= u(2n+1)\end{aligned}\right\}, \qquad 0 \le n \le \left(\frac{N}{2}\right) - 1$$

Now, we split the summation term in (5.87) into even and odd terms a (5.91) to obtain

$$\begin{aligned}v(k) &= \alpha(k)\left\{\sum_{n=0}^{(N/2)-1} u(2n)\cos\left[\frac{\pi(4n+1)k}{2N}\right]\right.\\ &\qquad \left.+\sum_{n=0}^{(N/2)-1} u(2n+1)\cos\left[\frac{\pi(4n+3)k}{2N}\right]\right\}\\ &= \alpha(k)\left\{\left[\sum_{n=0}^{(N/2)-1} \tilde{u}(n)\cos\left[\frac{\pi(4n+1)k}{2N}\right]\right.\right.\\ &\qquad \left.+\sum_{n=0}^{(N/2)-1} \tilde{u}(N-n-1)\cos\left[\frac{\pi(4n+3)k}{2N}\right]\right]\right\}\end{aligned}$$

Changing the index of summation in the second term to $n' = N - n -$ combining terms, we obtain

e Cosine Transform

The dct2 function in the Image Processing Toolbox computes the two-dimensional discrete cosine transform (DCT) of an image. The DCT has the property that, for a typical image, most of the visually significant information about the image is concentrated in just a few coefficients of the DCT. For this reason, the DCT is often used in image compression applications. For example, the DCT is at the heart of the the international standard lossy image compression algorithm known as JPEG. (The name comes from the working group that developed the standard: the Joint Photographic Experts Group.)

The two-dimensional DCT of an M-by-N matrix A is defined as follows:

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2n+1)q}{2N}, \quad \begin{matrix} 0 \le p \le M-1 \\ 0 \le q \le N-1 \end{matrix}$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p = 0 \\ \sqrt{2/M}, & 1 \le p \le M-1 \end{cases} \qquad \alpha_q = \begin{cases} 1/\sqrt{N}, & q = 0 \\ \sqrt{2/N}, & 1 \le q \le N-1 \end{cases}$$

The values B_{pq} are called the *DCT coefficients* of A. (Note that matrix indices in MATLAB always start at 1 rather than 0; therefore, the MATLAB matrix elements A(1,1) and B(1,1) correspond to the mathematical quantities A_{00} and B_{00}, respectively.)

The DCT is an invertible transform, and its inverse is given by:

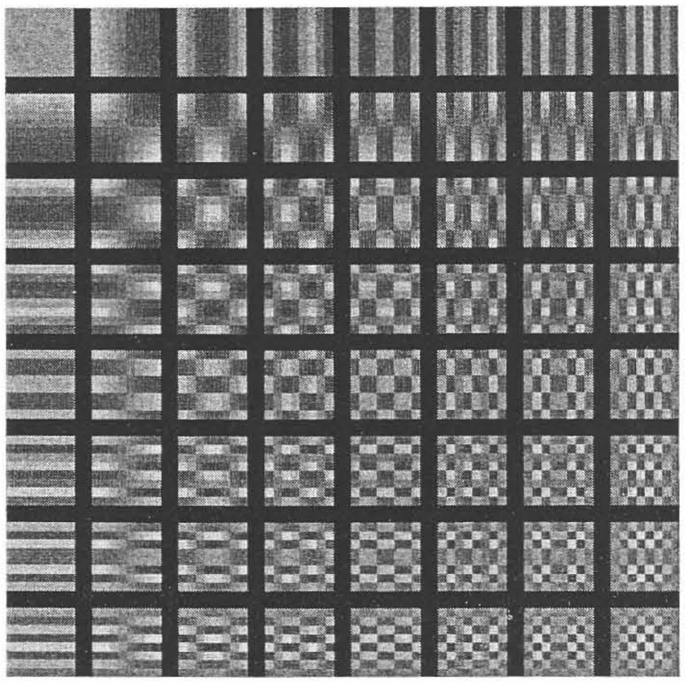
$$A_{mn} = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \alpha_p \alpha_q B_{pq} \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2n+1)q}{2N}, \quad \begin{matrix} 0 \le m \le M-1 \\ 0 \le n \le N-1 \end{matrix}$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p = 0 \\ \sqrt{2/M}, & 1 \le p \le M-1 \end{cases} \qquad \alpha_q = \begin{cases} 1/\sqrt{N}, & q = 0 \\ \sqrt{2/N}, & 1 \le q \le N-1 \end{cases}$$

The inverse DCT equation can be interpreted as meaning that any M-by-N matrix A can be written as a sum of MN functions of the form:

$$\alpha_p \alpha_q \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2n+1)q}{2N}, \qquad \begin{array}{l} 0 \le p \le M-1 \\ 0 \le q \le N-1 \end{array}$$

These functions are called the *basis functions* of the DCT. The DCT coefficients B_{pq}, then, can be regarded as the *weights* applied to each basis function. For 8-by-8 matrices, the 64 basis functions are illustrated by this image:



Horizontal frequencies increase from left to right, and vertical frequencies increase from top to bottom. The constant-valued basis function at the upper left is often called the *DC basis function*, and the corresponding DCT coefficient B_{00} is often called the *DC coefficient*.

The DCT Transform Matrix

The Image Processing Toolbox offers two different ways to compute the DCT. The first method is to use the function `dct2`. `dct2` uses an FFT-based algorithm for speedy computation with large inputs.

For small square inputs, such as 8-by-8 or 16-by-16, it may be more efficient to use the DCT *transform matrix*, which is returned by the function `dctmtx`. The M-by-M transform matrix `T` is given by:

$$T_{pq} = \begin{cases} \dfrac{1}{\sqrt{M}} & p = 0, \quad 0 \le q \le M-1 \\ \sqrt{\dfrac{2}{M}} \cos\dfrac{\pi(2q+1)p}{2M} & 1 \le p \le M-1, \quad 0 \le q \le M-1 \end{cases}$$

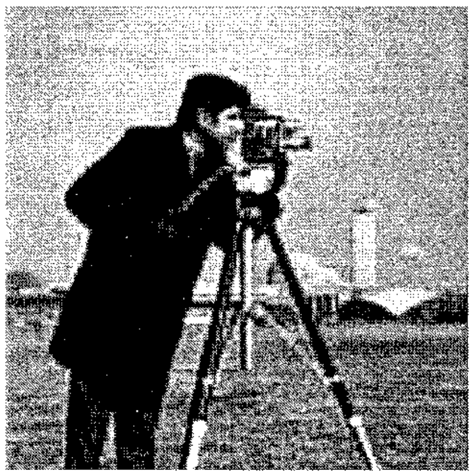
For an M-by-M matrix `A`, `T*A` is an M-by-M matrix whose columns contain the one-dimensional DCT of the columns of `A`. The two-dimensional DCT of `A` can be computed as `B=T*A*T'`. Since `T` is a real orthonormal matrix, its inverse is the same as its transpose. Therefore, the inverse two-dimensional DCT of `B` is given by `T'*B*T`.

The DCT and Image Compression

In the JPEG image compression algorithm, the input image is divided into 8-by-8 or 16-by-16 blocks, and the two-dimensional DCT is computed for each block. The DCT coefficients are then quantized, coded, and transmitted. The JPEG receiver (or JPEG file reader) decodes the quantized DCT coefficients, computes the inverse two-dimensional DCT of each block, and then puts the blocks back together into a single image. For typical images, many of the DCT coefficients have values close to zero; these coefficients can be discarded without seriously affecting the quality of the reconstructed image.

The example code below computes the two-dimensional DCT of 8-by-8 blocks in the input image; discards (sets to zero) all but 10 of the 64 DCT coefficients in each block; and then reconstructs the image using the two-dimensional inverse DCT of each block. The transform matrix computation method is used.

```
I = imread('cameraman.tif');
I = double(I)/255;
T = dctmtx(8);
B = blkproc(I,[8 8],'P1*x*P2',T,T');
mask = [1   1   1   1   0   0   0   0
        1   1   1   0   0   0   0   0
        1   1   0   0   0   0   0   0
        1   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
imshow(I), figure, imshow(I2)
```



Although there is some loss of quality in the reconstructed image, it is clearly recognizable, even though almost 85% of the DCT coefficients were discarded. To experiment with discarding more or fewer coefficients, and to apply this technique to other images, try running the demo function `dctdemo`.

To derive the inverse cosine transform relation, we relate $C_x(\omega_1, \omega_2)$ to $R(\omega_1, \omega_2)$ using (3.57), relate $R(\omega_1, \omega_2)$ to $r(n_1, n_2)$ using the inverse Fourier transform relation, and then relate $r(n_1, n_2)$ to $x(n_1, n_2)$ using (3.56). The result is

$$x(n_1, n_2) = \frac{1}{(2\pi)^2} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} C_x(\omega_1, \omega_2) \cos \omega_1(n_1 + \tfrac{1}{2}) \cos \omega_2(n_2 + \tfrac{1}{2})\, d\omega_1\, d\omega_2, \quad n_1 \ge 0, n_2 \ge 0. \tag{3.60}$$

From (3.58) and (3.60)

Cosine Transform Pair

$$C_x(\omega_1, \omega_2) = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} 4x(n_1, n_2) \cos \omega_1(n_1 + \tfrac{1}{2}) \cos \omega_2(n_2 + \tfrac{1}{2}). \tag{3.61a}$$

$$x(n_1, n_2) = \begin{cases} \dfrac{1}{(2\pi)^2} \displaystyle\int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} C_x(\omega_1, \omega_2) \cos \omega_1(n_1 + \tfrac{1}{2}) \cos \omega_2(n_2 + \tfrac{1}{2})\, d\omega_1\, d\omega_2 & n_1 \ge 0,\ n_2 \ge 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3.61b}$$

Many properties of the cosine transform can be derived from (3.61), or (3.55) and the Fourier transform properties. Some of the more important properties are listed in Table 3.4. From the symmetry properties, $C_x(\omega_1, \omega_2)$ is an even function and in addition is symmetric with respect to the ω_1 and ω_2 axes. When $x(n_1, n_2)$ is real, its cosine transform $C_x(\omega_1, \omega_2)$ is also real.

TABLE 3.4 PROPERTIES OF THE COSINE TRANSFORM

$x(n_1, n_2), y(n_1, n_2) = 0$ outside $n_1 \ge 0, n_2 \ge 0$
$x(n_1, n_2) \longleftrightarrow C_x(\omega_1, \omega_2)$
$y(n_1, n_2) \longleftrightarrow C_y(\omega_1, \omega_2)$

Property 1. *Linearity*

$$ax(n_1, n_2) + by(n_1, n_2) \longleftrightarrow aC_x(\omega_1, \omega_2) + bC_y(\omega_1, \omega_2)$$

Property 2. *Separable Sequence*

$$x(n_1, n_2) = x_1(n_1)x_2(n_2) \longleftrightarrow C_x(\omega_1, \omega_2) = C_{x1}(\omega_1)C_{x2}(\omega_2)$$

Property 3. *Energy Relationship*

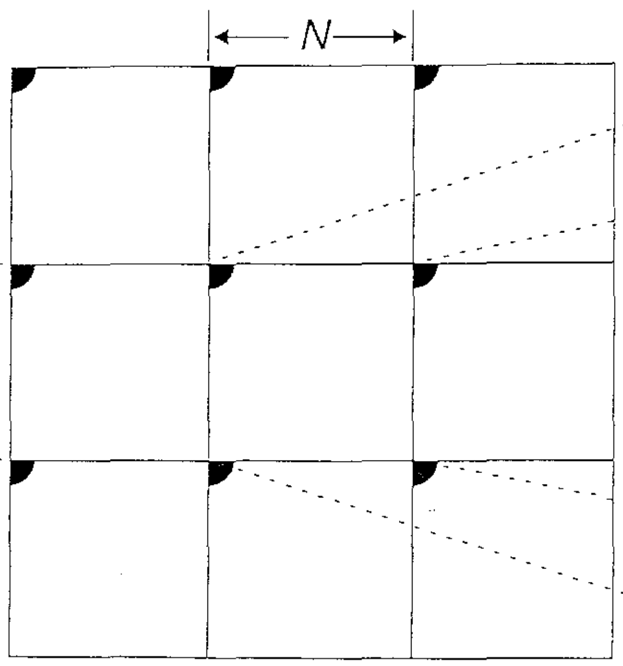
$$\sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} |x(n_1, n_2)|^2 = \frac{1}{4(2\pi)^2} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} |C_x(\omega_1, \omega_2)|^2\, d\omega_1\, d\omega_2$$

Property 4. *Symmetry Properties*

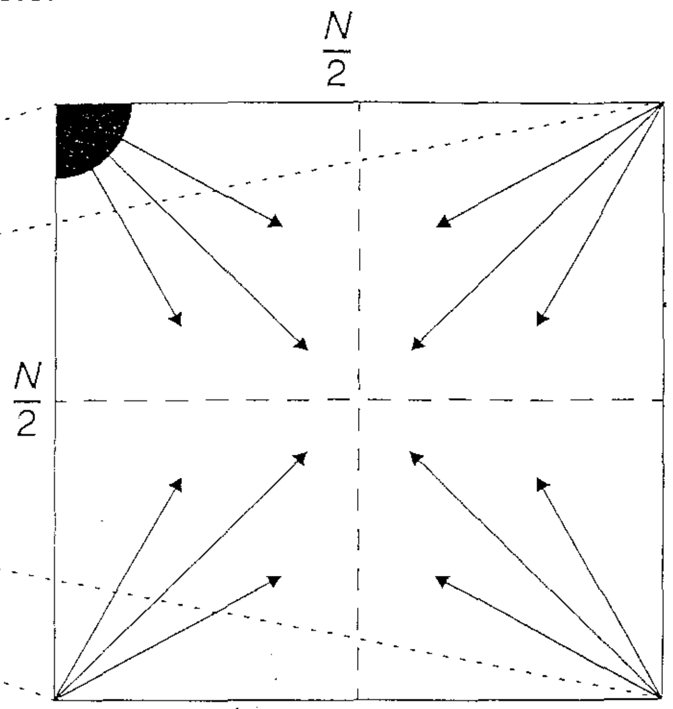
(a) $C_x(\omega_1, \omega_2) = C_x(-\omega_1, \omega_2) = C_x(\omega_1, -\omega_2) = C_x(-\omega_1, -\omega_2)$
(b) $x^*(n_1, n_2) \longleftrightarrow C_x^*(\omega_1, \omega_2)$
(c) real $x(n_1, n_2) \longleftrightarrow$ real $C_x(\omega_1, \omega_2)$

quency information. They are used for image compression or for hiding effects caused by noise. Visually they blur the image, although this blur is sometimes considered an enhancement because it imparts a softer effect to the image (see Figure 2.5-17). Lowpass filtering is performed by multiplying the spectrum by a filter and then applying the inverse transform to obtain the filtered image. The ideal filter function is shown in

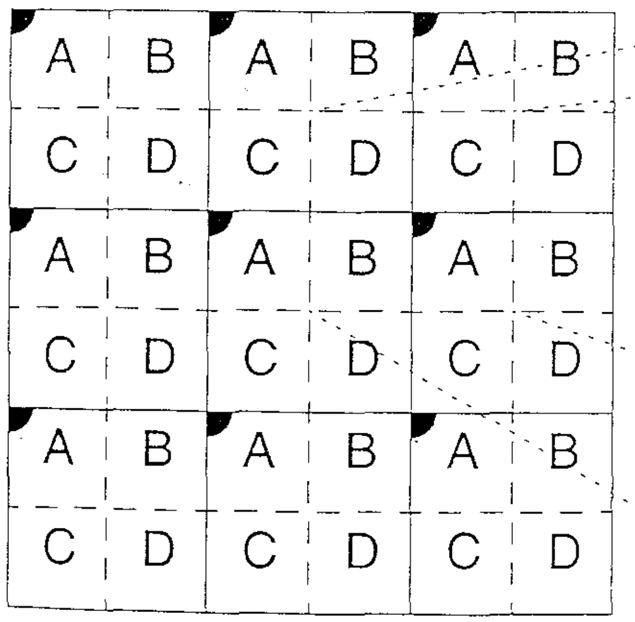
ure 2.5-13 Implied Symmetry for the Fourier Transform



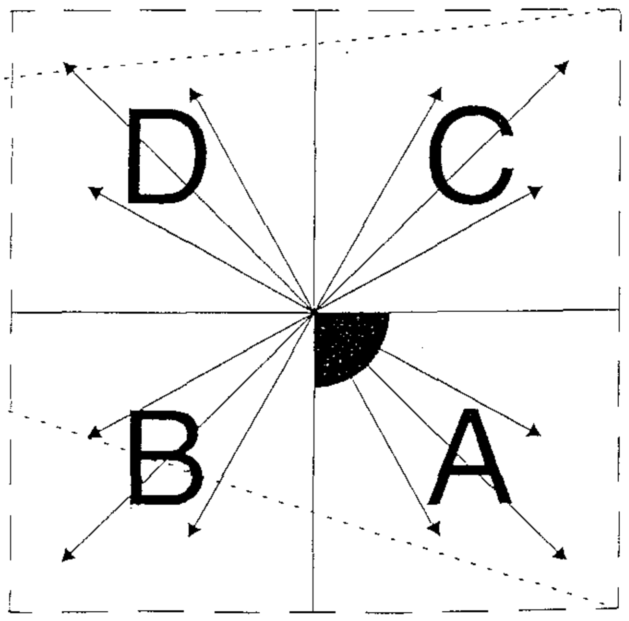
a. Implied symmetry with origin in upper-left corner. Each N × N block represents all the transform coefficients and is repeated infinitely in all directions.



b. Increasing frequency in direction of arrows.

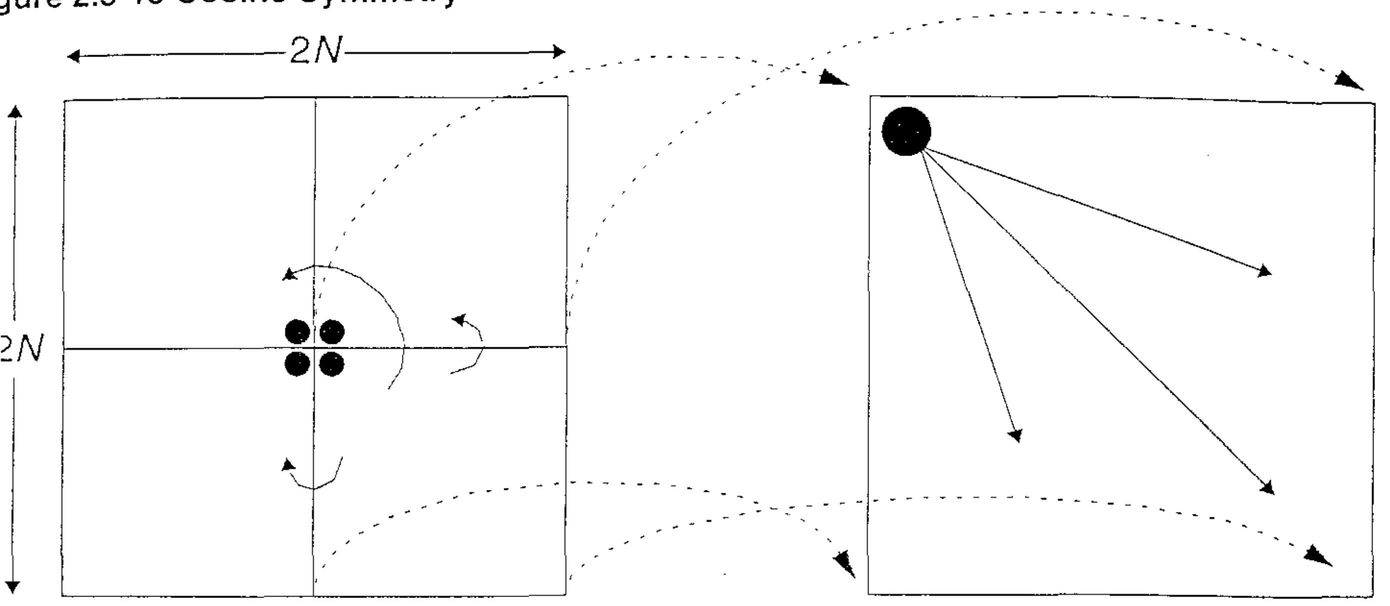


c. Periodic spectrum, with quadrants labeled A,B,C,D.



d. Spectrum shifted to center. Frequency increases in all directions as we move away from the origin.

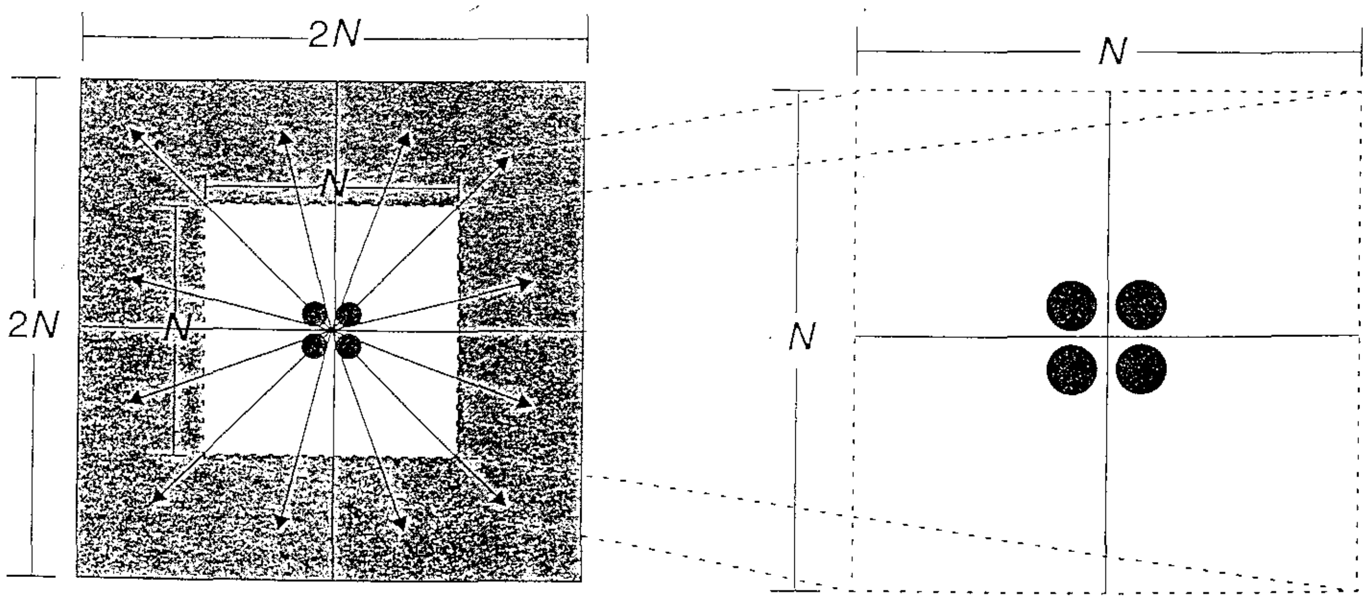
gure 2.5-15 Cosine Symmetry



a. Spectrum folded about origin, represented by the ●. The $2N \times 2N$ block is repeated infinitely in all directions.

b. Arrows indicate direction of increasing frequency for cosine spectrum.

jure 2.5-16 Cosine Spectrum Should Not Be Shifted to Center



a. Cosine spectrum with arrows in direction of increasing frequency.

b. Extracting the central $N \times N$ portion, we lose the high-frequency information.

□ Low frequencies ▨ High frequencies

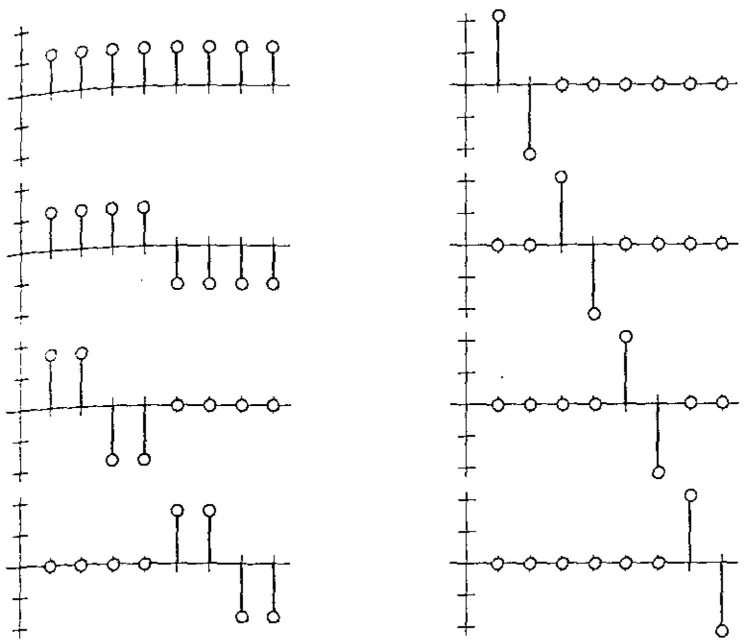


Figure 13–5 The Haar transform basis functions for $N = 8$

distinguishes it from the other transforms mentioned so far and establishes a starting point for wavelet transforms, which are introduced in the next chapter.

Basis Function Indexing. Since the Haar functions vary in two aspects (scale and position), they must be specified by a dual indexing scheme. The Haar functions are defined on the interval [0, 1] as follows. Let the integer $0 \le k \le N-1$ be specified (uniquely) by two other integers, p and q, as

$$k = 2^p + q - 1 \tag{47}$$

Notice that, under this construction, not only is k a function of p and q, but p and q are functions of k as well. For any value of $k > 0$, 2^p is the largest power of 2 such that $2^p \le k$, and $q - 1$ is the remainder.

The Haar functions are defined by

$$h_0(x) = \frac{1}{\sqrt{N}} \tag{48}$$

and

$$h_k(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2} & \dfrac{q-1}{2^p} \le x < \dfrac{q-\frac{1}{2}}{2^p} \\ -2^{p/2} & \dfrac{q-\frac{1}{2}}{2^p} \le x < \dfrac{q}{2^p} \\ 0 & \text{otherwise} \end{cases} \tag{49}$$

If we let $x = i/N$ for $i = 0, 1, \ldots, N-1$, this gives rise to a set of basis functions, each of which is an odd rectangular pulse pair, except for $k = 0$, which, as in the case of many of the other transforms discussed here, is constant. Further, the basis functions vary in both scale

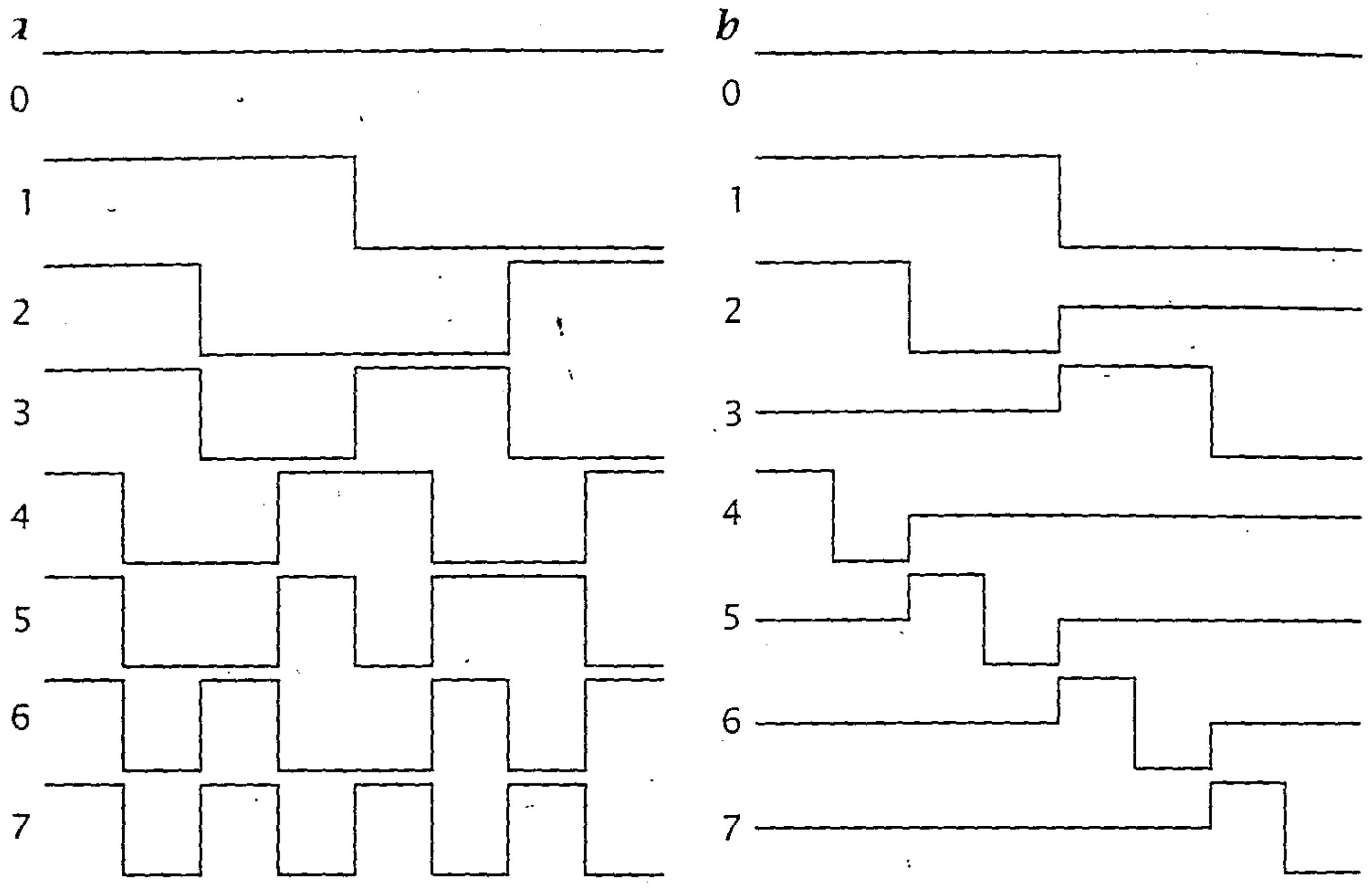


Figure 2.19: *First 8 base functions of one-dimensional unitary transforms for* $N = 16$*:* ***a*** *Hadamard transform and* ***b*** *Haar transform;*

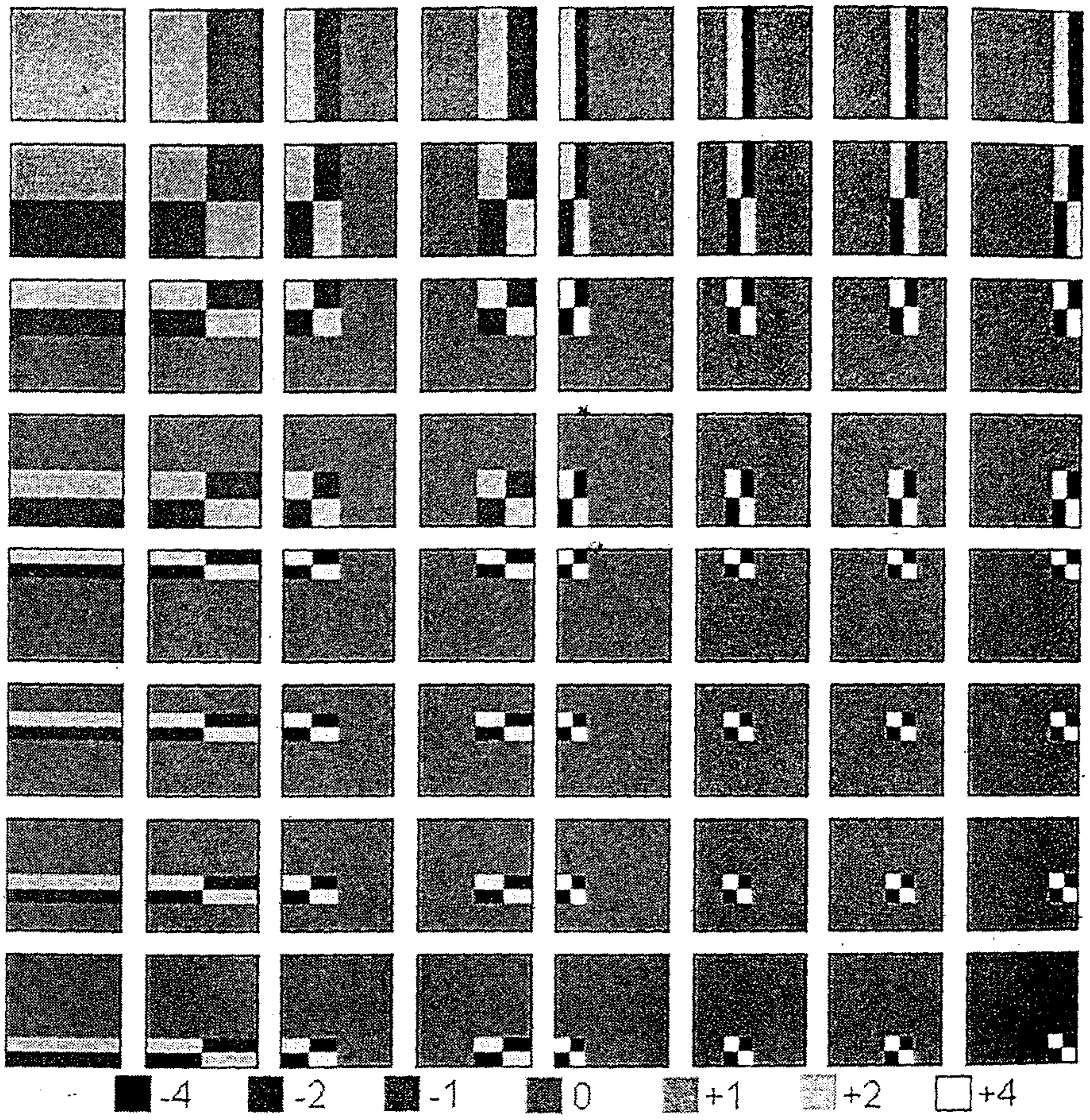


Figure 13–6 The Haar transform basis images for $N = 8$

ıt-by-eight unitary kernel matrix for the Haar transform is

$$\mathbf{Hr} = \frac{1}{\sqrt{8}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

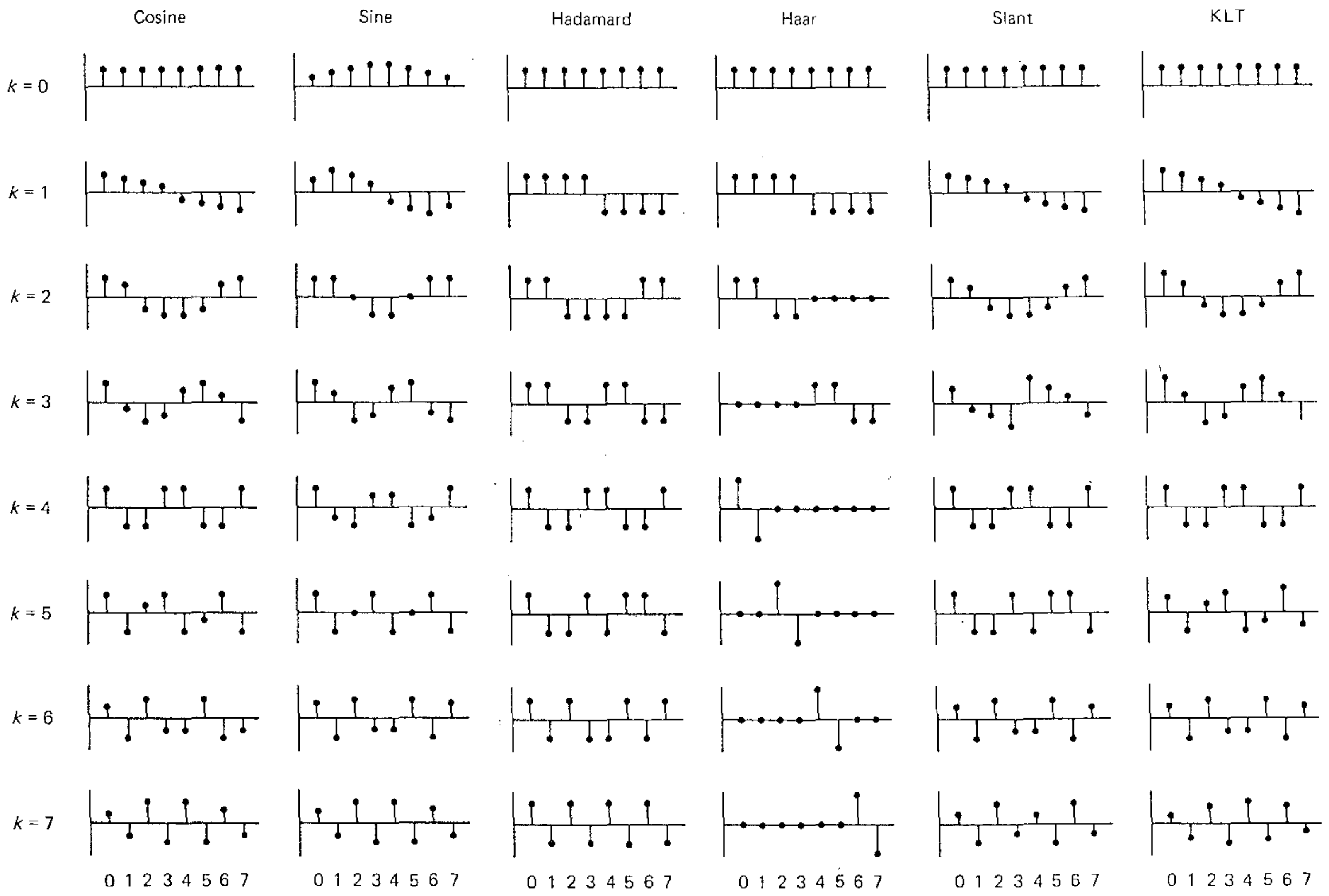
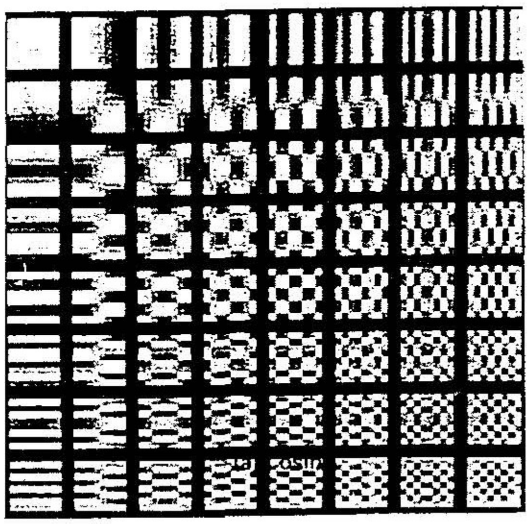
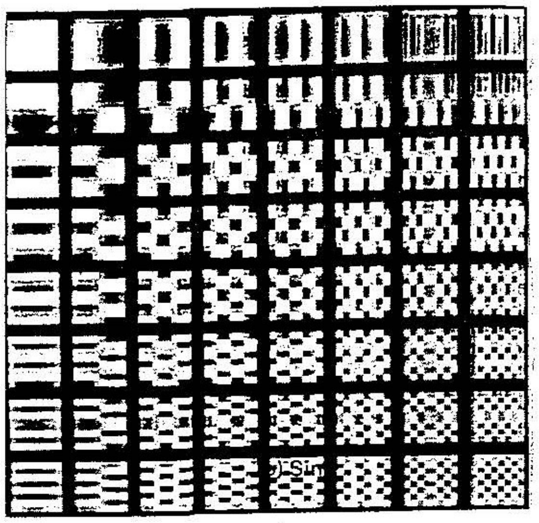


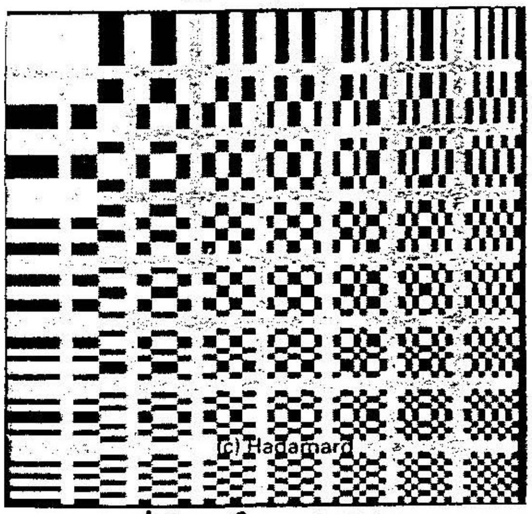
Figure 5.1 Basic vectors of the 8 × 8 transforms.



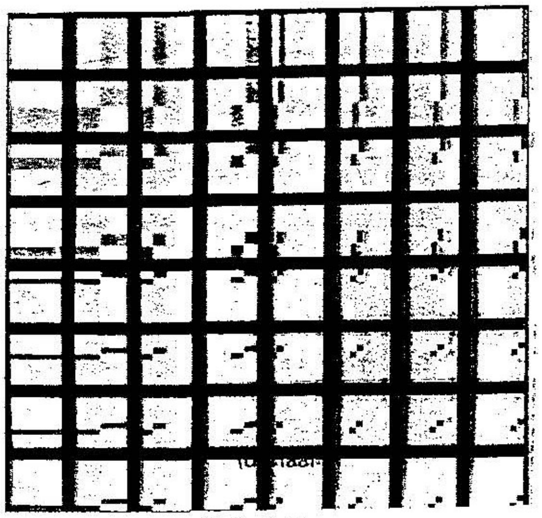
COSINE



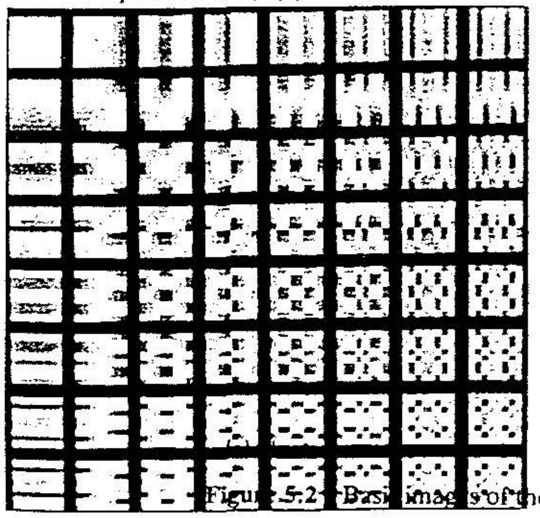
SINE



HADAMARD



HAAR



SLANT

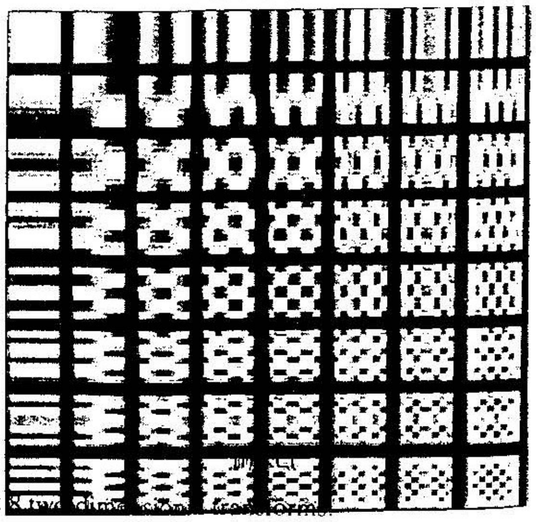
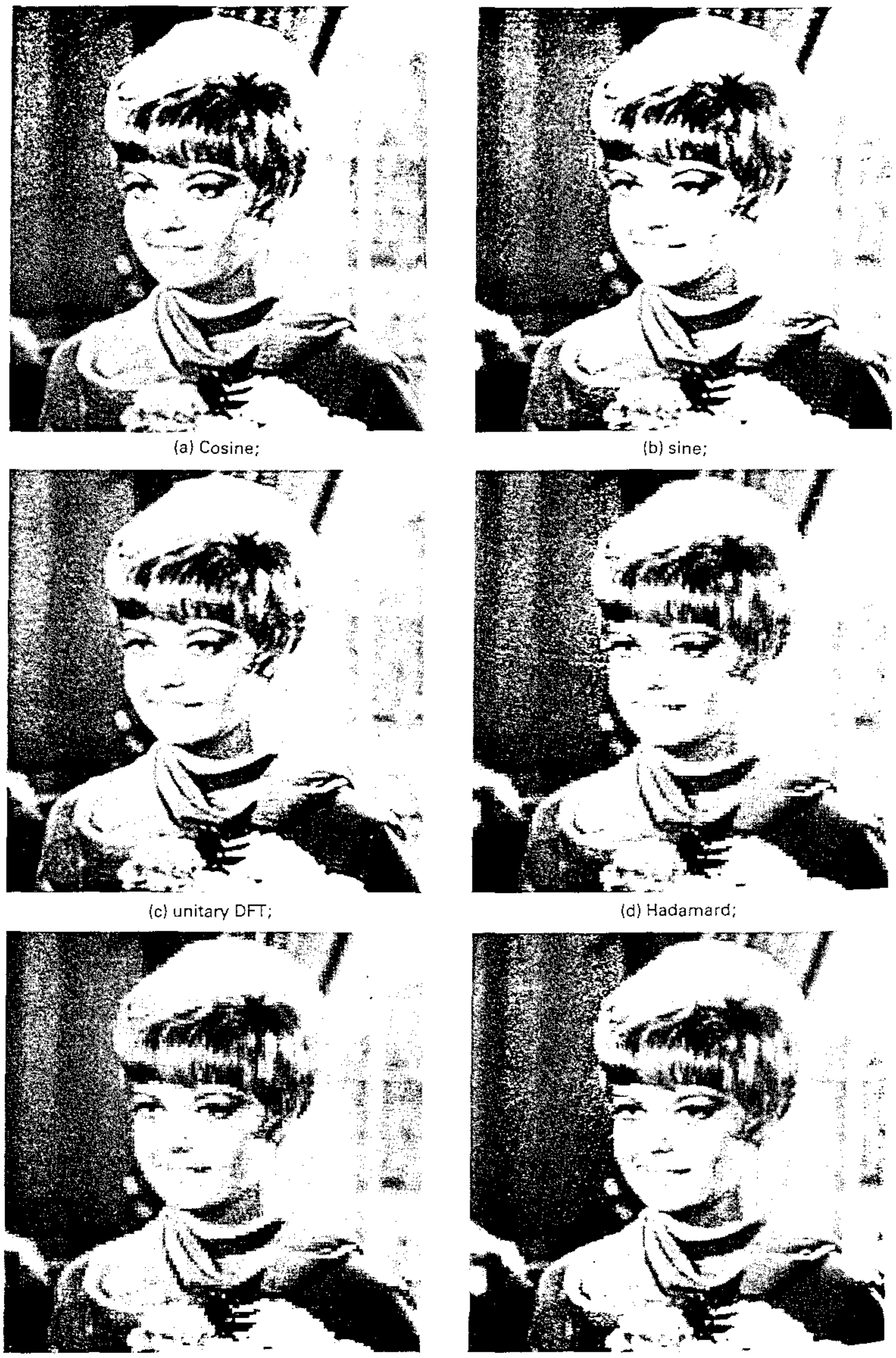


Figure 5.2 Basis images of the 8 × 8 two-dimensional transforms.

KL

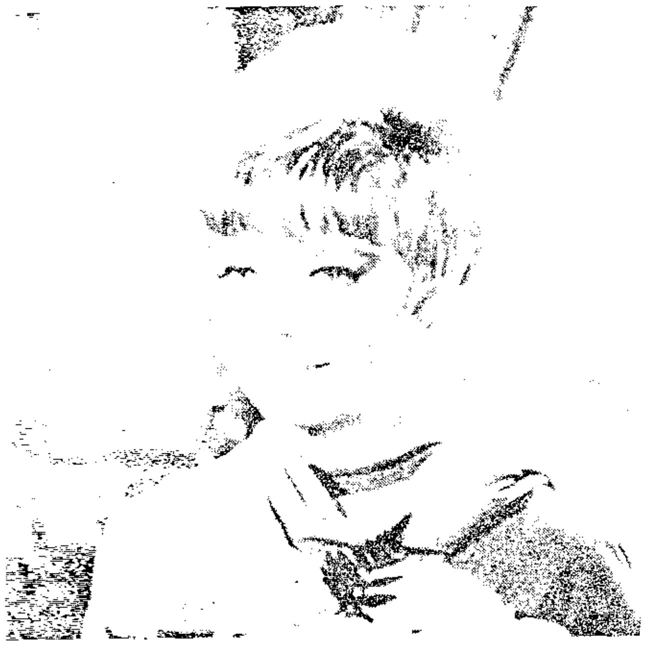
8×8 2D

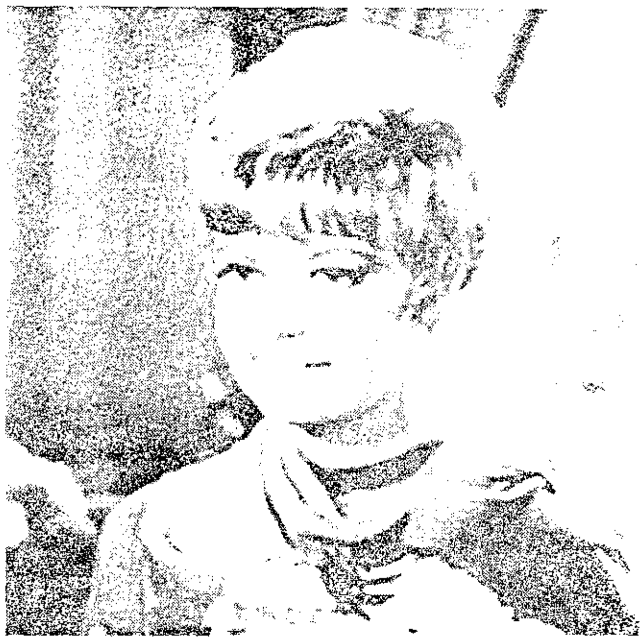


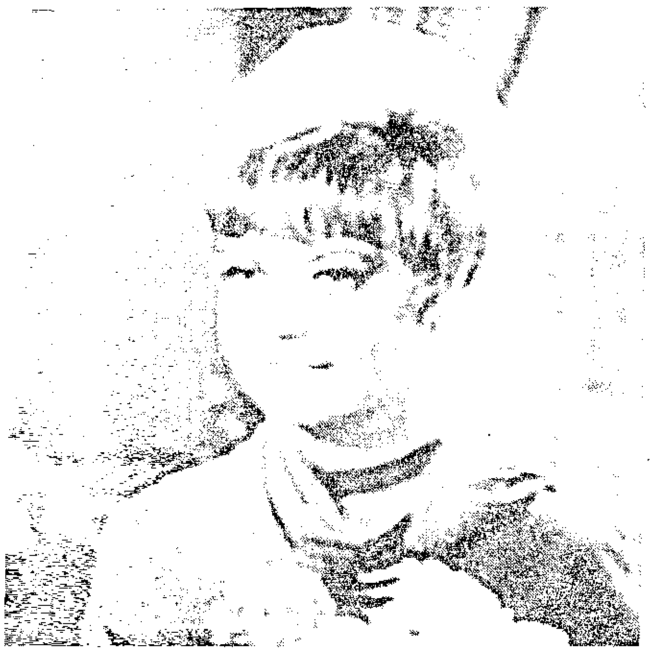
(a) Cosine; (b) sine; (c) unitary DFT; (d) Hadamard; (e) Haar; (f) Slant.

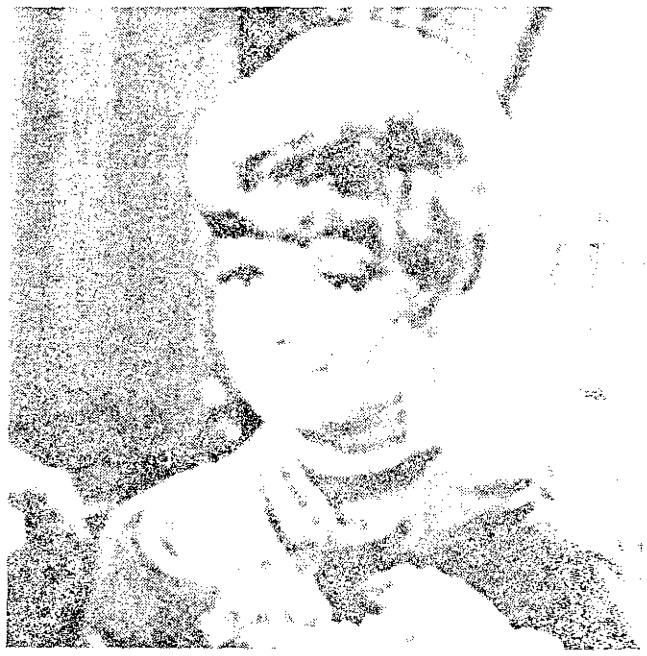
Figure 5.22 Basis restriction zonal filtering using different transforms with 4 : 1 sample reduction.

[illegible] we find the cosine transform to have the best performance.


(a) Original;


(b) 4 : 1 sample reduction;


(c) 8 : 1 sample reduction;


(d) 16 : 1 sample reduction.

Sine Transform

$$v_k = \sqrt{\frac{2}{N+1}} \sum_{n=0}^{N-1} u_n \sin\left[\frac{(n+1)(k+1)\pi}{N+1}\right] k = 0,1,\ldots N-1$$

and

$$u_n = \sqrt{\frac{2}{N+1}} \sum_{k=0}^{N-1} v_k \sin\left[\frac{(n+1)(k+1)\pi}{N+1}\right] k = 0,1,\ldots N-1$$

To evaluate this fast we relate it to a FFT

$$v_k = \operatorname{Re}\left\{\alpha_k e^{-\frac{\pi i k}{2N}} \sum_{n=0}^{N-1} \tilde{u}_n e^{-\frac{2\pi i k n}{N}}\right\} = \operatorname{Re}\left\{\alpha_k w_{2N}^{k} DFT(u)\right\}$$

where

$$\tilde{u}_n = u_{2n}$$

$$\tilde{u}_{N-n-1} = u_{2n} + 1 \quad 0 \le n \le \frac{N}{2} - 1$$

Inverse

```
%                                                                    1/7/99

% Y = DCT(X) returns the discrete cosine transform of X. The
% vector Y is the same size as X and contains the discrete
% cosine transform coefficients.
%
% Author(s): C. Thompson, 2-12-93
%            S. Eddins, 10-26-94, revised
% Copyright 1993-1998 The MathWorks, Inc.  All Rights Reserved.
% $Revision: 5.3 $  $Date: 1997/11/24 16:21:02 $

% References:
%    1) A. K. Jain, "Fundamentals of Digital Image
%       Processing", pp. 150-153.
%    2) Wallace, "The JPEG Still Picture Compression Standard",
%       Communications of the ACM, April 1991.

if rem(n,2)==1 | ~isreal(a), % odd case
% Form intermediate even-symmetric matrix.

else % even case

% Re-order the elements of the columns of x
y = [ aa(1:2:n,:); aa(n:-2:2,:) ];

% Compute weights to multiply DFT coefficients
ww = 2*exp(-i*(0:n-1)'*pi/(2*n))/sqrt(2*n);
ww(1) = ww(1) / sqrt(2);
W = ww(:,ones(1,m));

% Compute DCT using equation (5.92) in Jain
b = W .* fft(y);
end

if isreal(a), b = real(b); end
if do_trans, b = b.'; end
```

```
%

% X = IDCT(Y) inverts the DCT transform, returning the original
% vector if Y was obtained using Y = DCT(X).
%
% Author(s): C. Thompson, 2-12-93
%         S. Eddins, 10-26-94, revised
% Copyright 1993-1998 The MathWorks, Inc. All Rights Reserved.
% $Revision: 5.3 $ $Date: 1997/11/24 16:21:02 $

% References:
%    1) A. K. Jain, "Fundamentals of Digital Image
%       Processing", pp. 150-153.
%    2) Wallace, "The JPEG Still Picture Compression Standard",
%       Communications of the ACM, April 1991.

if rem(n,2)==1 | ~isreal(b), % odd case
% Form intermediate even-symmetric matrix.

else % even case
% Compute precorrection factor
ww = sqrt(2*n) * exp(j*pi*(0:n-1)/(2*n)).';
ww(1) = ww(1)/sqrt(2);
W = ww(:,ones(1,m));

% Compute x tilde using equation (5.93) in Jain
y = ifft(W.*bb);

% Re-order elements of each column according to equations (5.93) and
% (5.94) in Jain
a = zeros(n,m);
a(1:2:n,:) = y(1:n/2,:);
a(2:2:n,:) = y(n:-1:n/2+1,:);
end

if isreal(b), a = real(a); end
if do_trans, a = a.'; end
```

all symmetrizable

DCT 1 $\quad \cos \frac{jk\pi}{N-1}$ $\quad \begin{pmatrix} 2 & -2 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -2 & 2 \end{pmatrix}$

DCT 2 $\quad \cos \frac{(j+\frac{1}{2})k\pi}{N}$ $\quad \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}$

DCT 3 $\quad \cos \frac{j(k+\frac{1}{2})\pi}{N}$ $\quad \begin{pmatrix} 2 & -2 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$

DCT 4 $\quad \cos \frac{(j+\frac{1}{2})(k+\frac{1}{2})\pi}{N}$ $\quad \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 3 \end{pmatrix}$

ical Analysis of the DCT Coefficient Distributions for Images

. Lam, *Member, IEEE,* and Joseph W. Goodman, *Fellow, IEEE*

des, there have been various
)CT coefficients for images.
ıly on fitting the empirical
ith a variety of well-known
ıparing their goodness-of-fit.
dominant choice balancing
to the empirical data. Yet,
has been no mathematical
o this distribution. In this
ical analysis using a doubly
iich not only provides the
t also leads to insights about
literature. This model also
ıanges in the image statistics
butions.

ısforms, Gaussian distribu-
robability statistics.

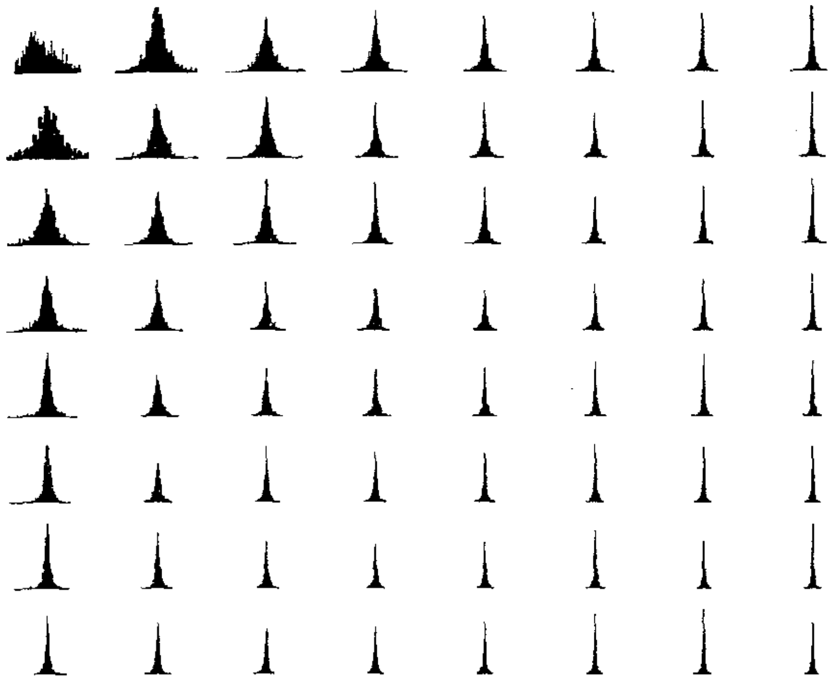


Fig. 1. Histogram of DCT coefficients of "bridge."

:ION

:t and multimedia systems,
l widespread popularity for
ontinuous-tone images. In
vided into nonoverlapping
h block is then subjected to
before quantization and en-
he compression algorithm,
d by various researchers.
n understanding the distri-
e more than 20 years ago.
have performed the DCT
the corresponding coeffi-
ng statistical distribution?
: instance, in quantizer de-
ıhancement [1], [2]. Fig. 1
s of the DCT coefficients.
picture shown in Fig. 2(a)
library. The upper left co-

experimental results like Fig. 1 indicated that they resemble Laplacian distributions when the Kolmogorov–Smirnov goodness-of-fit test is used [4]. The probability density function of a Laplacian distribution can be written as

$$p(x) = \frac{\mu}{2} \exp\{-\mu|x|\}. \quad (1)$$

This is sometimes also referred to as the double exponential distribution. Since then, different researchers have tried a variety of fitting methods, such as χ^2, Kurtosis, and Watson tests. Many other possible distributions of the coefficients have also been proposed, including Cauchy, generalized Gaussian, and even a sum of Gaussians [5]–[9]. Using different pictures for the experiments, they often differ in opinion as to what distribution model is the most suitable, although the Laplacian distribution remains a popular choice balancing simplicity of the model and fidelity to the empirical data. Yet, none of them provided any analytic justification for their choices of distributions. In this paper, we investigate this problem in two steps: first, we derive the distri-

- JPEG is not trivial to implement. It is not likely you will be able to sit down and write your own JPEG encoder/decoder in a few evenings. We recommend that you obtain a third-party JPEG library, rather than writing your own.
- JPEG is not supported by very many file formats. The formats that do support JPEG are all fairly new and can be expected to be revised at frequent intervals.

Baseline JPEG

The JPEG specification defines a minimal subset of the standard called baseline JPEG, which all JPEG-aware applications are required to support. This baseline uses an encoding scheme based on the Discrete Cosine Transform (DCT) to achieve compression. DCT is a generic name for a class of operations identified and published some years ago. DCT-based algorithms have since made their way into various compression methods.

DCT-based encoding algorithms are always lossy by nature. DCT algorithms are capable of achieving a high degree of compression with only minimal loss of data. This scheme is effective only for compressing continuous-tone images in which the differences between adjacent pixels are usually small. In practice, JPEG works well only on images with depths of at least four or five bits per color channel. The baseline standard actually specifies eight bits per input sample. Data of lesser bit depth can be handled by scaling it up to eight bits per sample, but the results will be bad for low-bit-depth source data, because of the large jumps between adjacent pixel values. For similar reasons, colormapped source data does not work very well, especially if the image has been dithered.

The JPEG compression scheme is divided into the following stages:

1. Transform the image into an optimal color space.
2. Downsample chrominance components by averaging groups of pixels together.
3. Apply a Discrete Cosine Transform (DCT) to blocks of pixels, thus removing redundant image data.
4. Quantize each block of DCT coefficients using weighting functions optimized for the human eye.
5. Encode the resulting coefficients (image data) using a Huffman variable word-length algorithm to remove redundancies in the coefficients.

Figure 9-11 summarizes these steps, and the following subsections look at each of them in turn. Note that JPEG decoding performs the reverse of these steps.

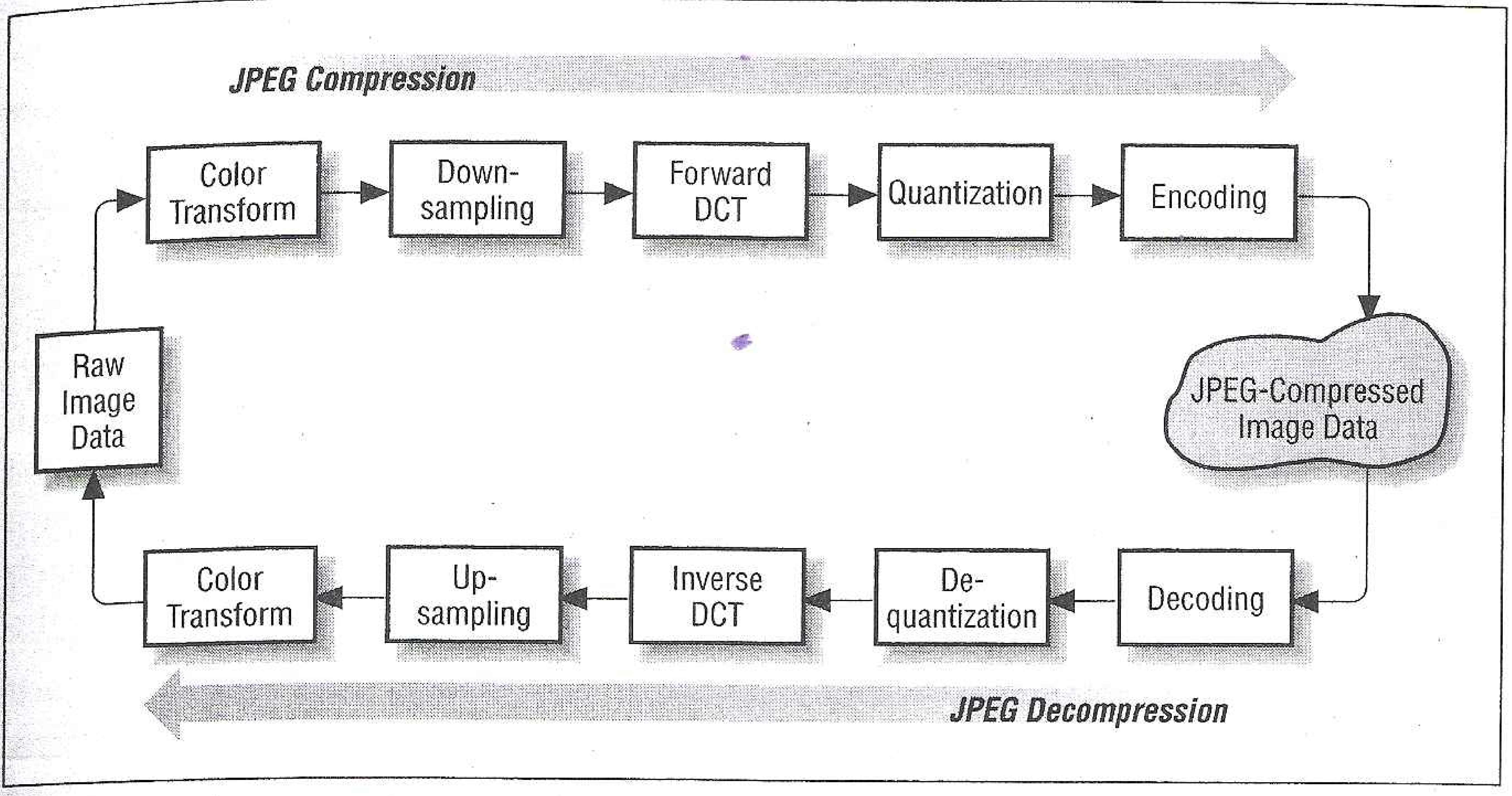


FIGURE 9-11: *JPEG compression and decompression*

Transform the image

The JPEG algorithm is capable of encoding images that use any type of color space. JPEG itself encodes each component in a color model separately, and it is completely independent of any color-space model, such as RGB, HSI, or CMY. The best compression ratios result if a luminance/chrominance color space, such as YUV or YCbCr, is used. (See Chapter 2 for a description of these color spaces.)

Most of the visual information to which human eyes are most sensitive is found in the high-frequency, gray-scale, luminance component (Y) of the YCbCr color space. The other two chrominance components (Cb and Cr) contain high-frequency color information to which the human eye is less sensitive. Most of this information can therefore be discarded.

In comparison, the RGB, HSI, and CMY color models spread their useful visual image information evenly across each of their three color components, making the selective discarding of information very difficult. All three color components would need to be encoded at the highest quality, resulting in a poorer compression ratio. Gray-scale images do not have a color space as such and therefore do not require transforming.

Downsample chrominance components

The simplest way of exploiting the eye's lesser sensitivity to chrominance information is simply to use fewer pixels for the chrominance channels. For example, in an image nominally 1000×1000 pixels, we might use a full 1000×1000 luminance pixels but only 500×500 pixels for each chrominance component. In this representation, each chrominance pixel covers the same area as a 2×2 block of luminance pixels. We store a total of six pixel values for each 2×2 block (four luminance values, one each for the two chrominance channels), rather than the twelve values needed if each component is represented at full resolution. Remarkably, this 50 percent reduction in data volume has almost no effect on the perceived quality of most images. Equivalent savings are not possible with conventional color models such as RGB, because in RGB each color channel carries some luminance information and so any loss of resolution is quite visible.

When the uncompressed data is supplied in a conventional format (equal resolution for all channels), a JPEG compressor must reduce the resolution of the chrominance channels by *downsampling*, or averaging together groups of pixels. The JPEG standard allows several different choices for the sampling ratios, or relative sizes, of the downsampled channels. The luminance channel is always left at full resolution (1:1 sampling). Typically both chrominance channels are downsampled 2:1 horizontally and either 1:1 or 2:1 vertically, meaning that a chrominance pixel covers the same area as either a 2×1 or a 2×2 block of luminance pixels. JPEG refers to these downsampling processes as 2h1v and 2h2v sampling, respectively.

Another notation commonly used is 4:2:2 sampling for 2h1v and 4:2:0 sampling for 2h2v; this notation derives from television customs (color transformation and downsampling have been in use since the beginning of color TV transmission). 2h1v sampling is fairly common because it corresponds to National Television Standards Committee (NTSC) standard TV practice, but it offers less compression than 2h2v sampling, with hardly any gain in perceived quality.

Apply a Discrete Cosine Transform

The image data is divided up into 8×8 blocks of pixels. (From this point on, each color component is processed independently, so a "pixel" means a single value, even in a color image.) A DCT is applied to each 8×8 block. DCT converts the spatial image representation into a frequency map: the low-order or "DC" term represents the average value in the block, while successive higher-order ("AC") terms represent the strength of more and more rapid changes across the width or height of the block. The highest AC term represents the strength of a cosine wave alternating from maximum to minimum at adjacent pixels.

The DCT calculation is fairly complex; in fact, this is the most costly step in JPEG compression. The point of doing it is that we have now separated out the high- and low-frequency information present in the image. We can discard high-frequency data easily without losing low-frequency information. The DCT step itself is lossless except for roundoff errors.

Quantize each block

To discard an appropriate amount of information, the compressor divides each DCT output value by a "quantization coefficient" and rounds the result to an integer. The larger the quantization coefficient, the more data is lost, because the actual DCT value is represented less and less accurately. Each of the 64 positions of the DCT output block has its own quantization coefficient, with the higher-order terms being quantized more heavily than the low-order terms (that is, the higher-order terms have larger quantization coefficients). Furthermore, separate quantization tables are employed for luminance and chrominance data, with the chrominance data being quantized more heavily than the luminance data. This allows JPEG to exploit further the eye's differing sensitivity to luminance and chrominance.

It is this step that is controlled by the "quality" setting of most JPEG compressors. The compressor starts from a built-in table that is appropriate for a medium-quality setting and increases or decreases the value of each table entry in inverse proportion to the requested quality. The complete quantization tables actually used are recorded in the compressed file so that the decompressor will know how to (approximately) reconstruct the DCT coefficients.

Selection of an appropriate quantization table is something of a black art. Most existing compressors start from a sample table developed by the ISO JPEG committee. It is likely that future research will yield better tables that provide more compression for the same perceived image quality. Implementation of improved tables should not cause any compatibility problems, because decompressors merely read the tables from the compressed file; they don't care how the table was picked.

Encode the resulting coefficients

The resulting coefficients contain a significant amount of redundant data. Huffman compression will losslessly remove the redundancies, resulting in smaller JPEG data. An optional extension to the JPEG specification allows arithmetic encoding to be used instead of Huffman for an even greater compression ratio. (See the section called "JPEG Extensions (Part 1)" below.) At this point, the JPEG data stream is ready to be transmitted across a communications channel or encapsulated inside an image file format.

JPEG Extensions (Part 1)

What we have examined thus far is only the baseline specification for JPEG. A number of extensions have been defined in Part 1 of the JPEG specification that provide progressive image buildup, improved compression ratios using arithmetic encoding, and a lossless compression scheme. These features are beyond the needs of most JPEG implementations and have therefore been defined as "not required to be supported" extensions to the JPEG standard.

Progressive image buildup

Progressive image buildup is an extension for use in applications that need to receive JPEG data streams and display them on the fly. A baseline JPEG image can be displayed only after all of the image data has been received and decoded. But some applications require that the image be displayed after only some of the data is received. Using a conventional compression method, this means displaying the first few scan lines of the image as it is decoded. In this case, even if the scan lines were interlaced, you would need at least 50 percent of the image data to get a good clue as to the content of the image. The progressive buildup extension of JPEG offers a better solution.

Progressive buildup allows an image to be sent in layers rather than scan lines. But instead of transmitting each bitplane or color channel in sequence (which wouldn't be very useful), a succession of images built up from approximations of the original image are sent. The first scan provides a low-accuracy representation of the entire image—in effect, a very low-quality JPEG compressed image. Subsequent scans gradually refine the image by increasing the effective quality factor. If the data is displayed on the fly, you would first see a crude, but recognizable, rendering of the whole image. This would appear very quickly because only a small amount of data would need to be transmitted to produce it. Each subsequent scan would improve the displayed image's quality one block at a time.

A limitation of progressive JPEG is that each scan takes essentially a full JPEG decompression cycle to display. Therefore, with typical data transmission rates, a very fast JPEG decoder (probably specialized hardware) would be needed to make effective use of progressive transmission.

The JPEG standard does offer a separate lossless mode. This mode has nothing in common with the regular DCT-based algorithms, and it is currently implemented only in a few commercial applications. JPEG lossless is a form of Predictive Lossless Coding using a 2D Differential Pulse Code Modulation (DPCM) scheme. The basic premise is that the value of a pixel is combined with the values of up to three neighboring pixels to form a predictor value. The predictor value is then subtracted from the original pixel value. When the entire bitmap has been processed, the resulting predictors are compressed using either the Huffman or the binary arithmetic entropy encoding methods described in the JPEG standard.

Lossless JPEG works on images with 2 to 16 bits per pixel, but performs best on images with 6 or more bits per pixel. For such images, the typical compression ratio achieved is 2:1. For image data with fewer bits per pixels, other compression schemes do perform better.

JPEG Extensions (Part 3)

The following JPEG extensions are described in Part 3 of the JPEG specification.

Variable quantization

Variable quantization is an enhancement available to the quantization procedure of DCT-based processes. This enhancement may be used with any of the DCT-based processes defined by JPEG with the exception of the baseline process.

The process of quantization used in JPEG quantizes each of the 64 DCT coefficients using a corresponding value from a quantization table. Quantization values may be redefined prior to the start of a scan but must not be changed once they are within a scan of the compressed data stream.

Variable quantization allows the scaling of quantization values within the compressed data stream. At the start of each 8×8 block is a quantizer scale factor used to scale the quantization table values within an image component and to match these values with the AC coefficients stored in the compressed data. Quantization values may then be located and changed as needed.

Variable quantization allows the characteristics of an image to be changed to control the quality of the output based on a given model. The variable quantizer can constantly adjust during decoding to provide optimal output.

The amount of output data can also be decreased or increased by raising or lowering the quantizer scale factor. The maximum size of the resulting JPEG file or data stream may be imposed by constant adaptive adjustments made by the variable quantizer.

A related JPEG extension provides for hierarchical storage of the same image at multiple resolutions. For example, an image might be stored at 250×250, 500×500, 1000×1000, and 2000×2000 pixels, so that the same image file could support display on low-resolution screens, medium-resolution laser printers, and high-resolution imagesetters. The higher-resolution images are stored as differences from the lower-resolution ones, so they need less space than they would need if they were stored independently. This is not the same as a progressive series, because each image is available in its own right at the full desired quality.

Arithmetic encoding

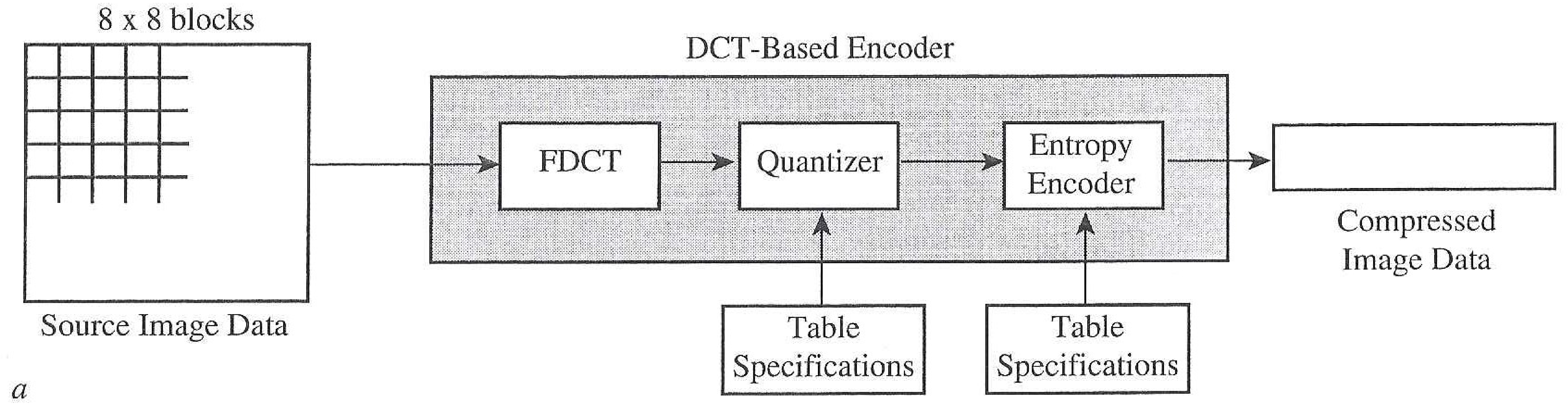
The baseline JPEG standard defines Huffman compression as the final step in the encoding process. A JPEG extension replaces the Huffman engine with a binary arithmetic entropy encoder. The use of an arithmetic coder reduces the resulting size of the JPEG data by a further 10 percent to 15 percent over the results that would be achieved by the Huffman coder. With no change in resulting image quality, this gain could be of importance in implementations where enormous quantities of JPEG images are archived.

Arithmetic encoding has several drawbacks:

- Not all JPEG decoders support arithmetic decoding. Baseline JPEG decoders are required to support only the Huffman algorithm.
- The arithmetic algorithm is slower in both encoding and decoding than Huffman.
- The arithmetic coder used by JPEG (called a *Q-coder*) is owned by IBM and AT&T. (Mitsubishi also holds patents on arithmetic coding.) You must obtain a license from the appropriate vendors if their Q-coders are to be used as the back end of your JPEG implementation.

Lossless JPEG compression

A question that commonly arises is "At what Q factor does JPEG become lossless?" The answer is "never." Baseline JPEG is a lossy method of compression regardless of adjustments you may make in the parameters. In fact, DCT-based encoders are always lossy, because roundoff errors are inevitable in the color conversion and DCT steps. You can suppress deliberate information loss in the downsampling and quantization steps, but you still won't get an exact recreation of the original bits. Further, this minimum-loss setting is a very inefficient way to use lossy JPEG.



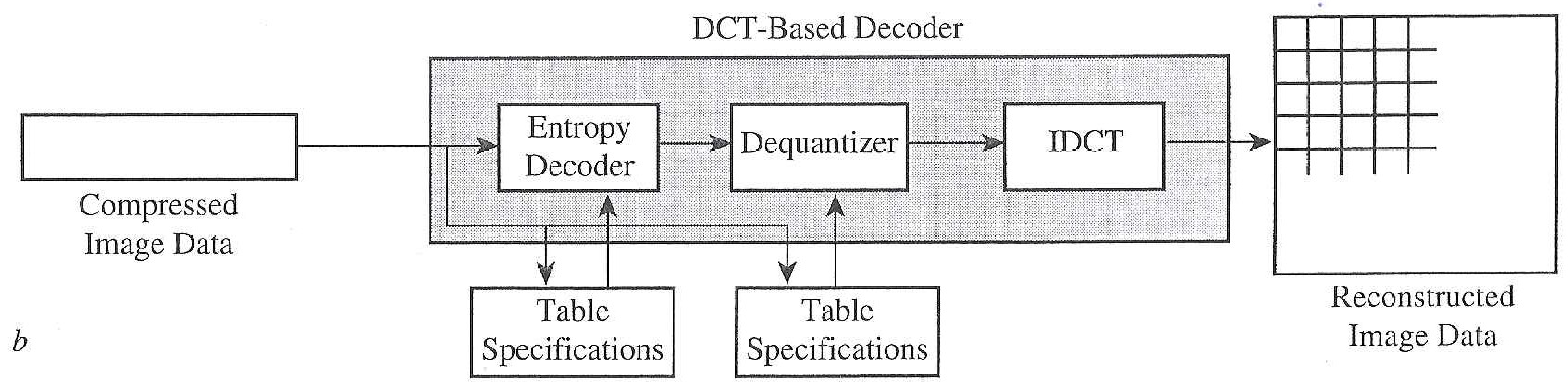


FIGURE 9.1 JPEG: *(a)* Encoder; *(b)* Decoder

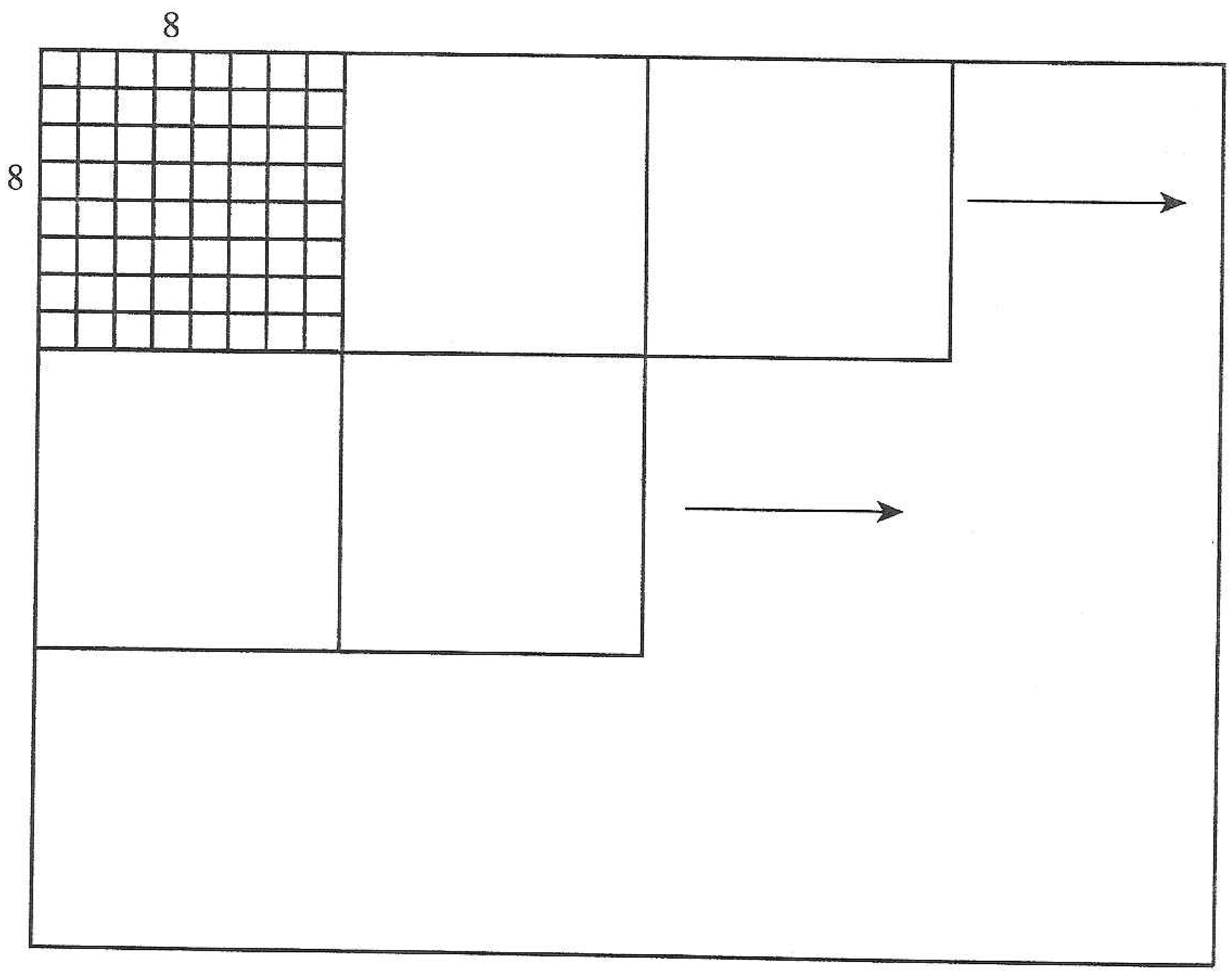


FIGURE 9.2 Partitioning of an Image into 8 × 8 Blocks of Pixels

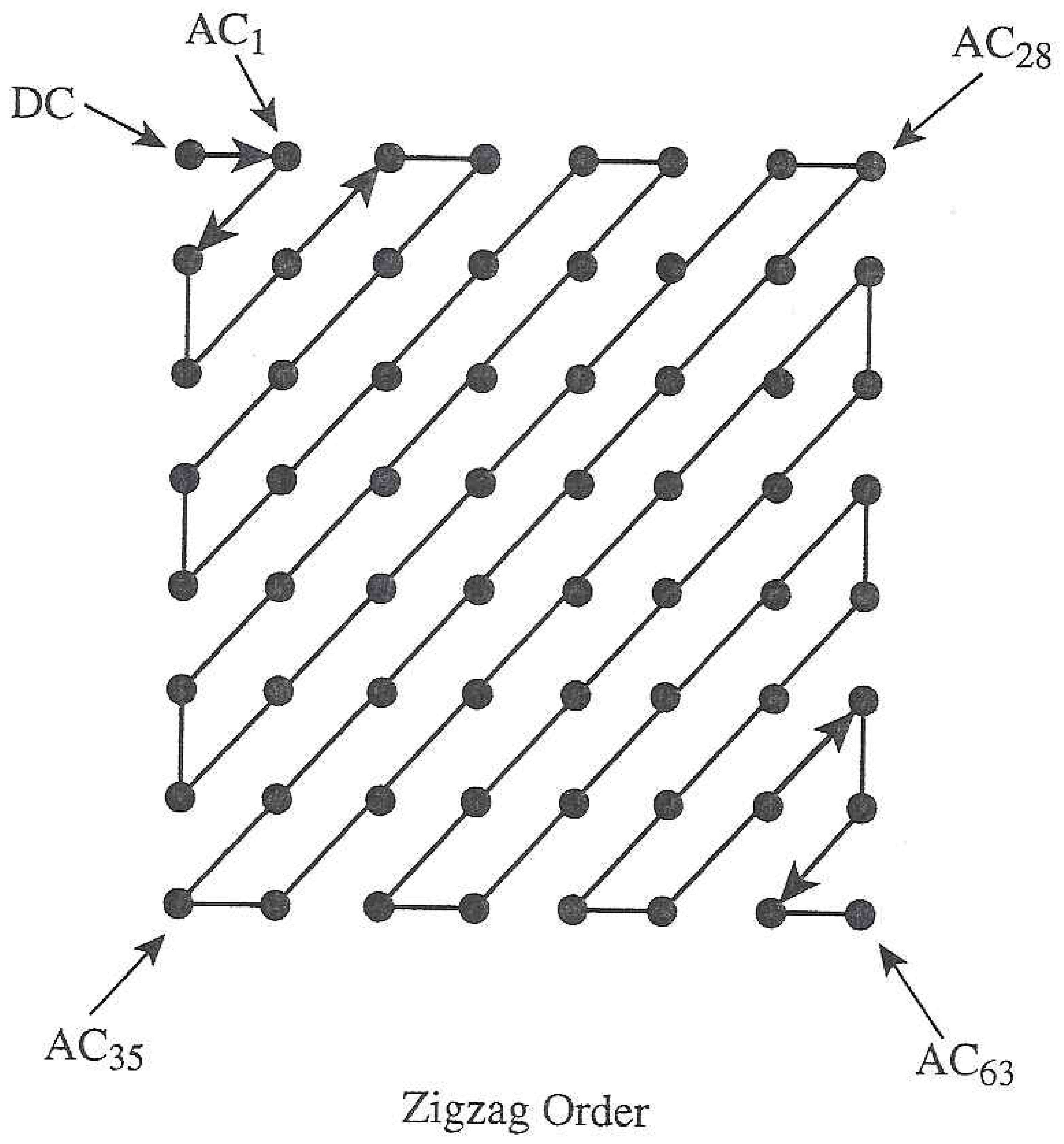


FIGURE 9.3 Zigzag Coefficient Ordering

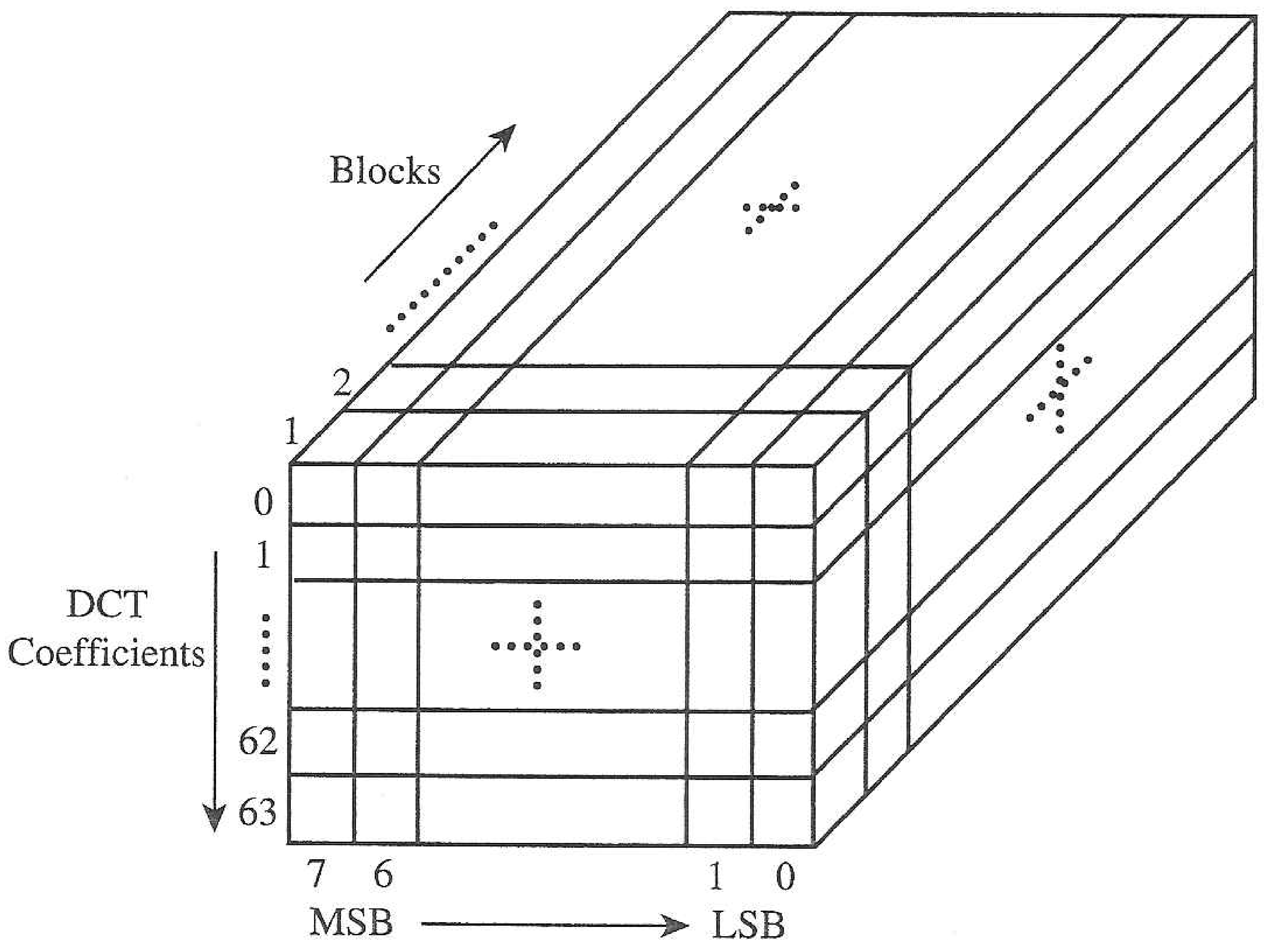


FIGURE 9.4 Sequential Lossy Encoding

Luminance quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Chrominance quantization table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

FIGURE 9.5 Suggested Step Sizes for CCIR-601

masking properties of the eye and zero out many small coefficient values, which shows up in Figure 9.5 as large step size values.

The JPEG sequential baseline encoder accommodates only 8-bit sample inputs and has two Huffman tables each for the DC and AC coefficients. The entropy coding methods are detailed in Section 9.5.

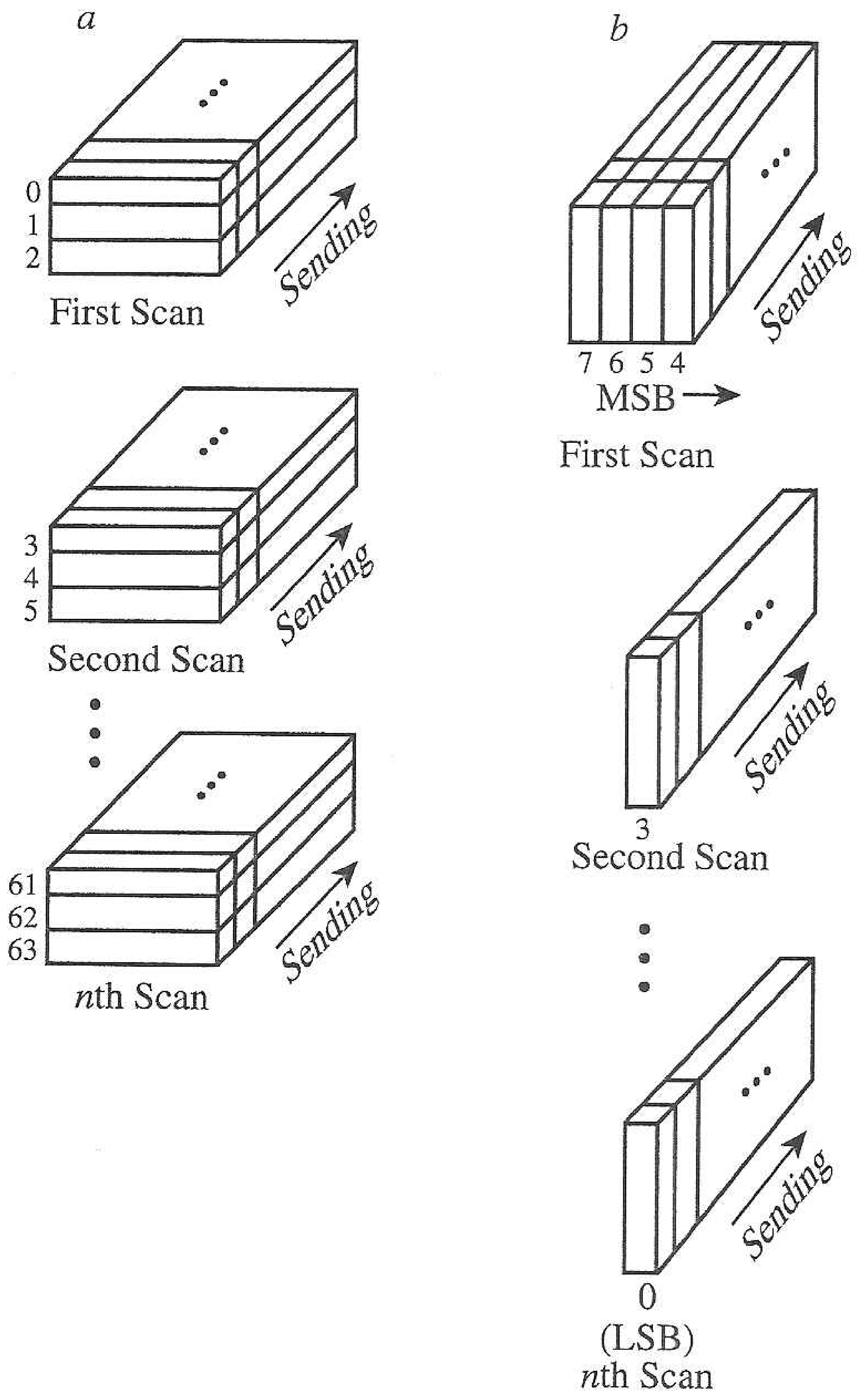


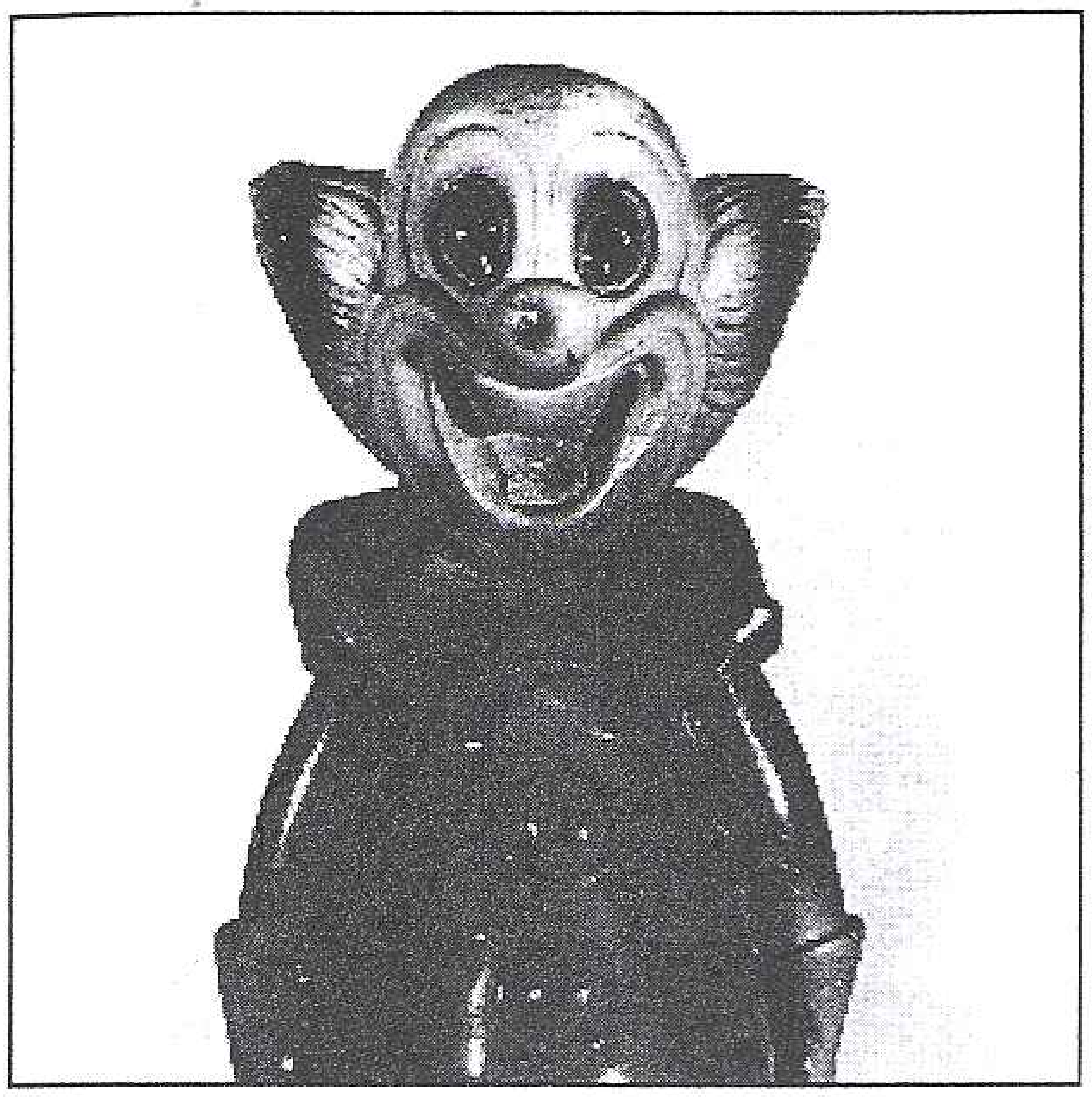
FIGURE 9.6 Progressive Encoding: *(a)* Spectral Selection; *(b)* Successive Approximation

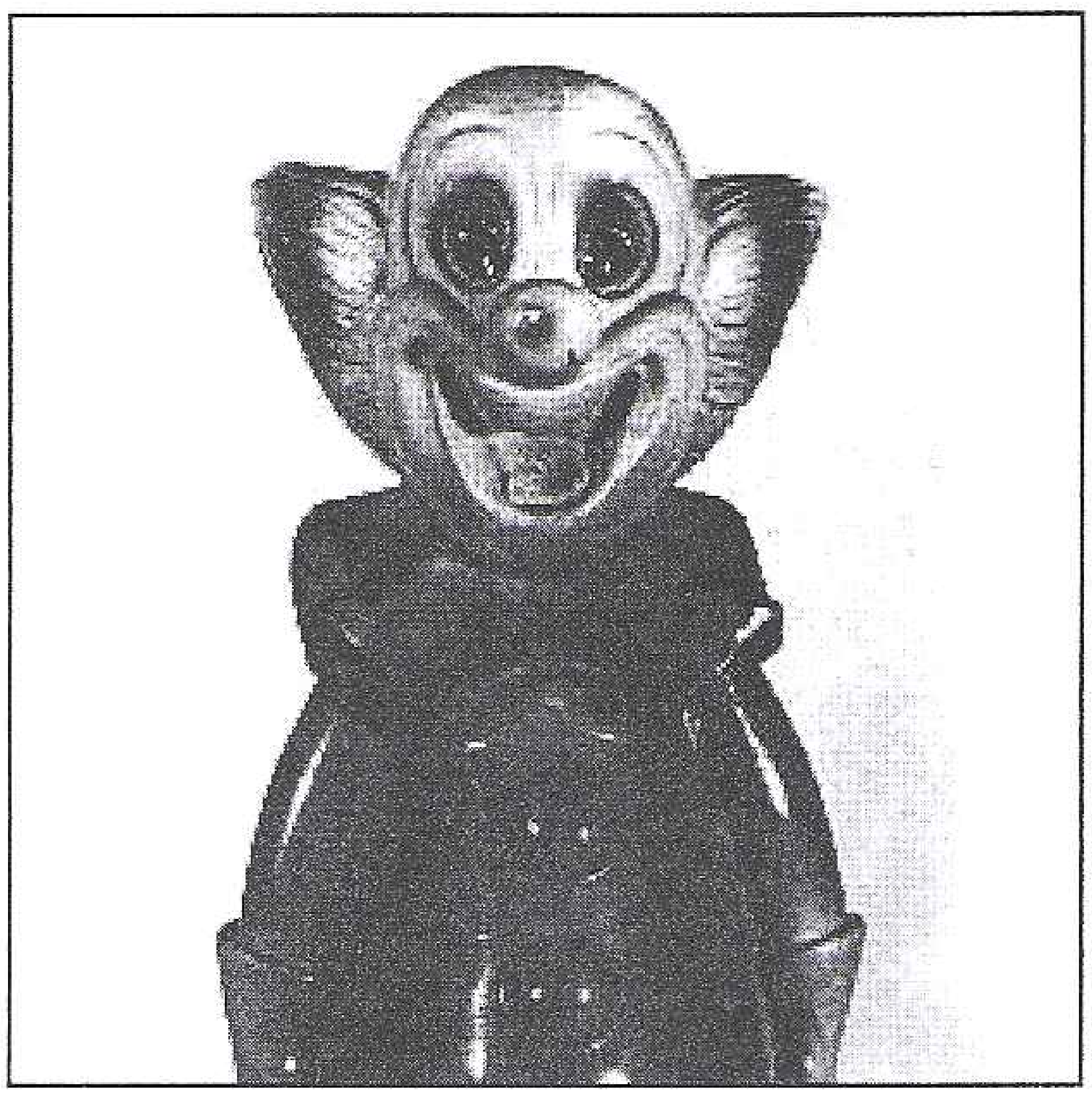
distortion above cutoff, while successive approximation gives more of a constant distortion across spatial frequencies.

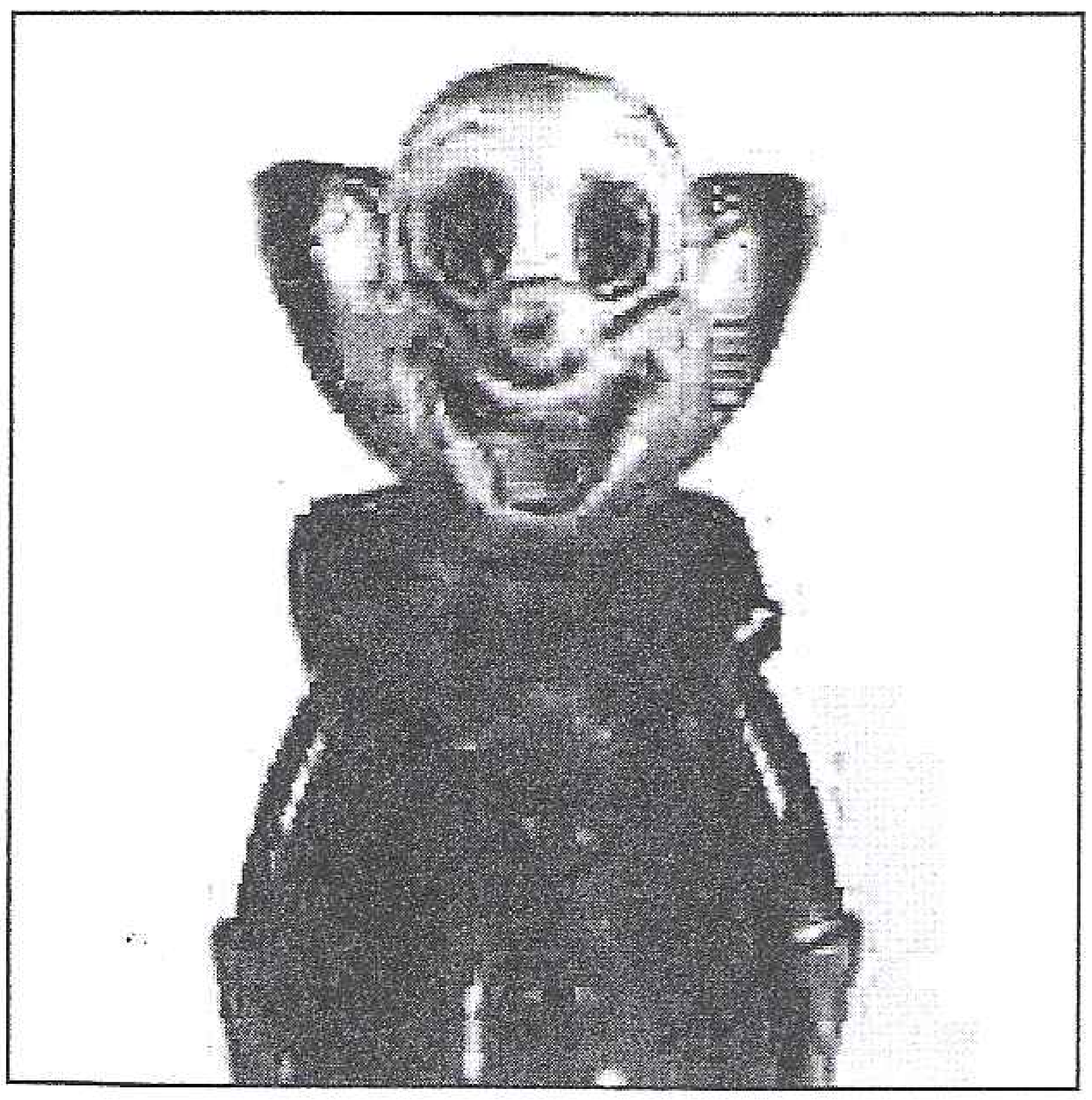
9.4 Hierarchical (Pyramidal) Encoding

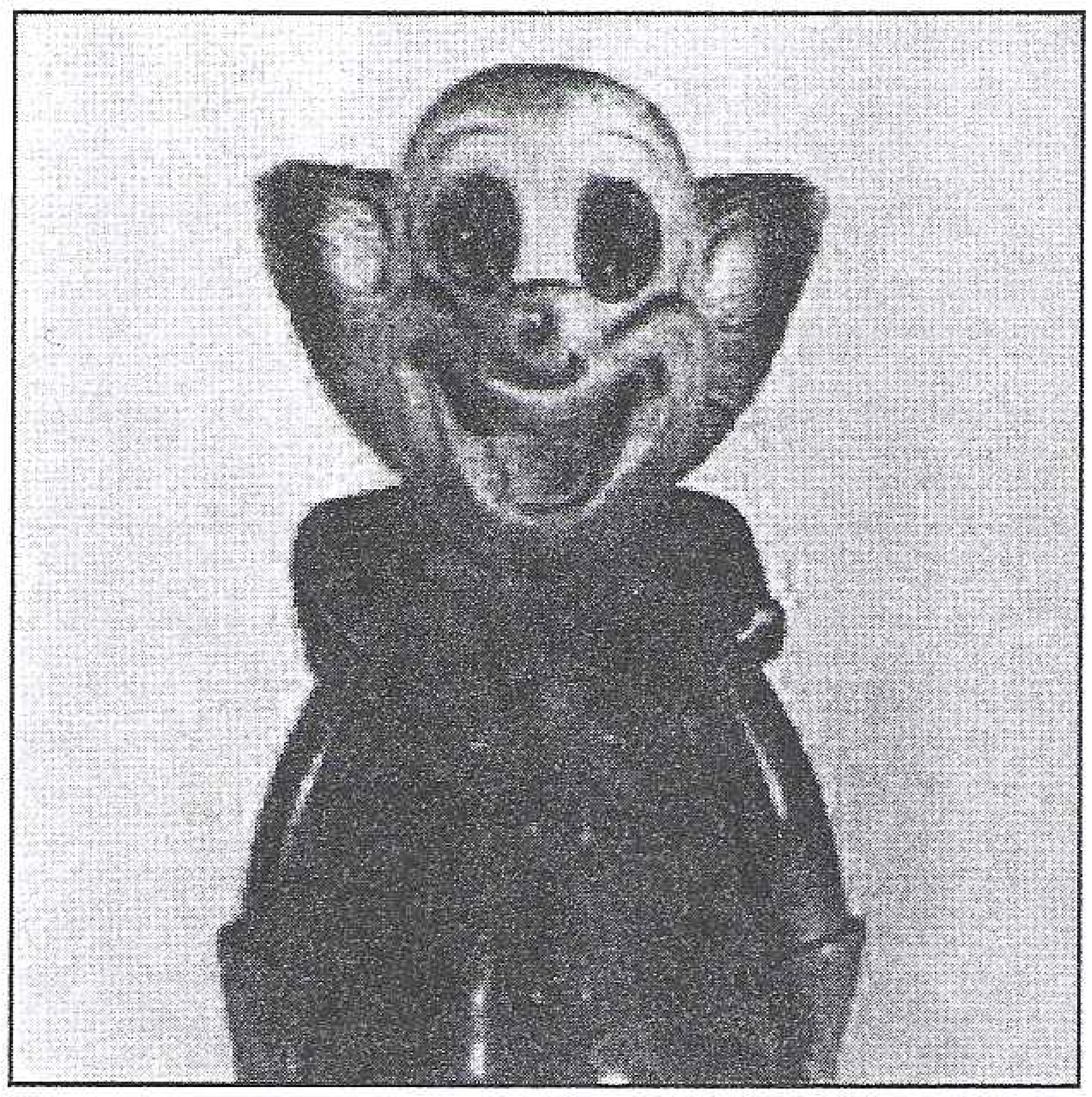
It may sometimes be necessary to view a high-resolution image on a lower-resolution display device; in these situations, it would be inefficient to transmit the DCT coefficients for the entire high-resolution image to the low-resolution

Figure 5.3-1 Lossy Image Compression


a. Original image.


b. JPEG compression, 10:1 ratio.


c. JPEG compression, 48:1 ratio.


d. Wavelet/vector quantization compression, 36:1 ratio.

dictive coding (DPC), block truncation coding (BTC), and vector quantization (VQ). In the transform domain we will discuss filtering, zonal coding, threshold coding, and the JPEG algorithm. We will also look at techniques for combining these methods into hybrid compression algorithms, which use both the spatial and transform domains.

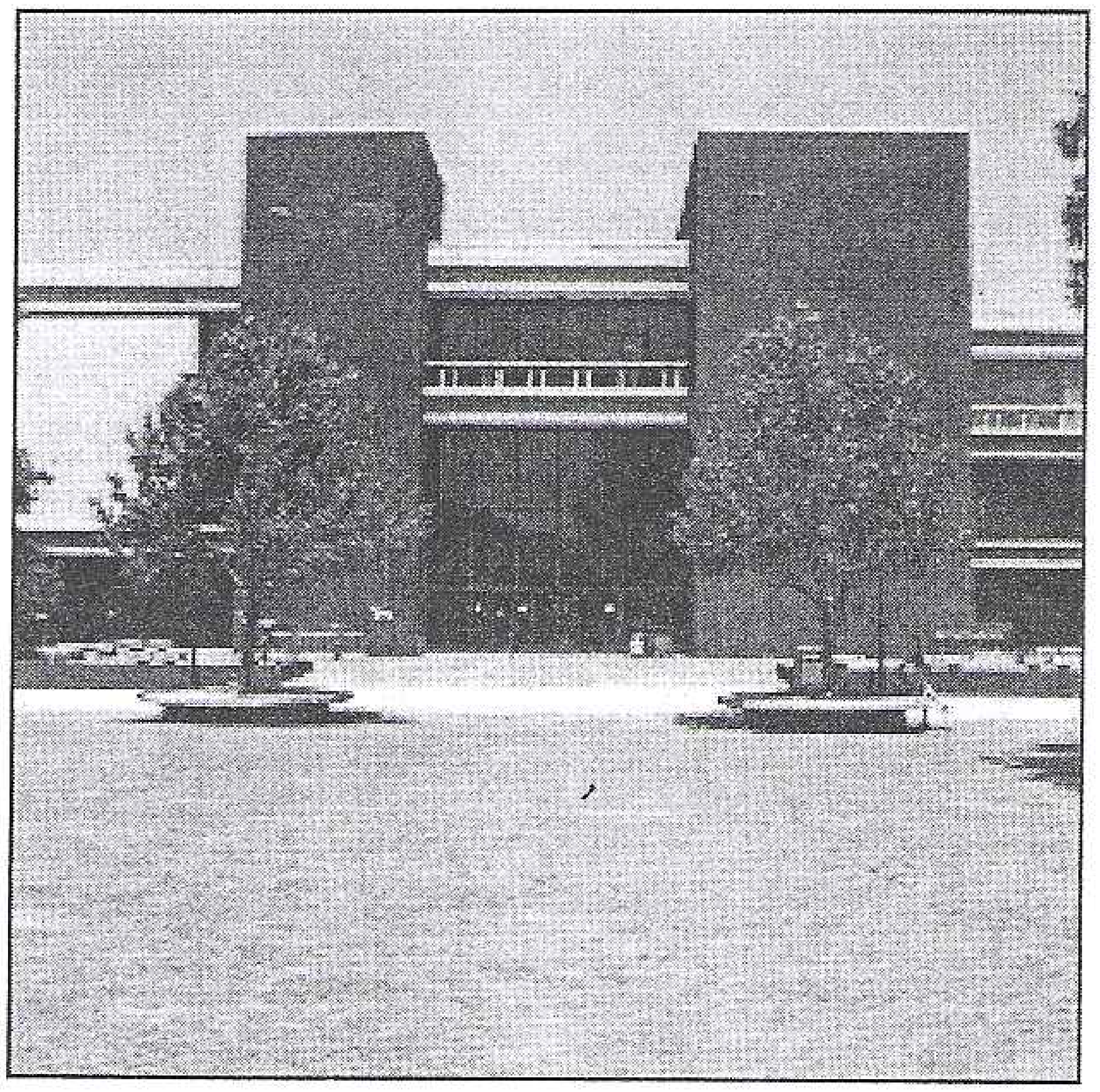
5.3.1 Gray-Level Run-Length Coding

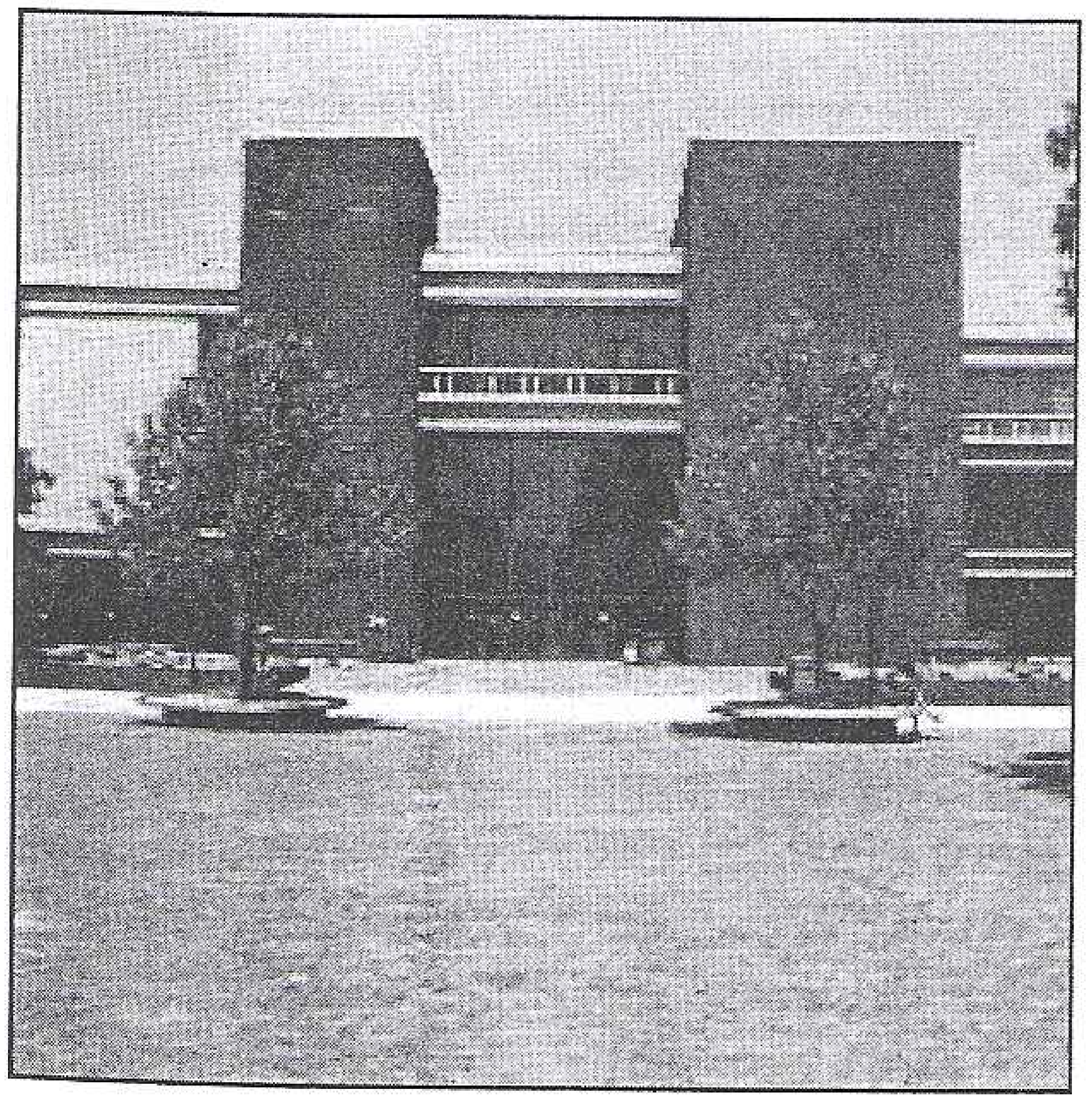
In Section 5.2.2 on lossless compression we discussed methods of extending basic run-length coding to gray-level images, by using bit-plane coding. The RLC technique

5.3.6 Hybrid Methods

Hybrid compression methods use both the spatial domain and the transform domain. For example, the original image (spatial domain) can be differentially mapped, and then this differential image can be transform coded. Alternately, a one-dimensional transform can be performed on the rows, and this transformed data can undergo differential predictive coding along the columns. These methods are often used for compression of analog video signals. For digital images these techniques can be applied to blocks (subimages), as well as rows or columns.

igure 5.3-14 JPEG Compression


a. Original image.


b. JPEG compression = 10:1.

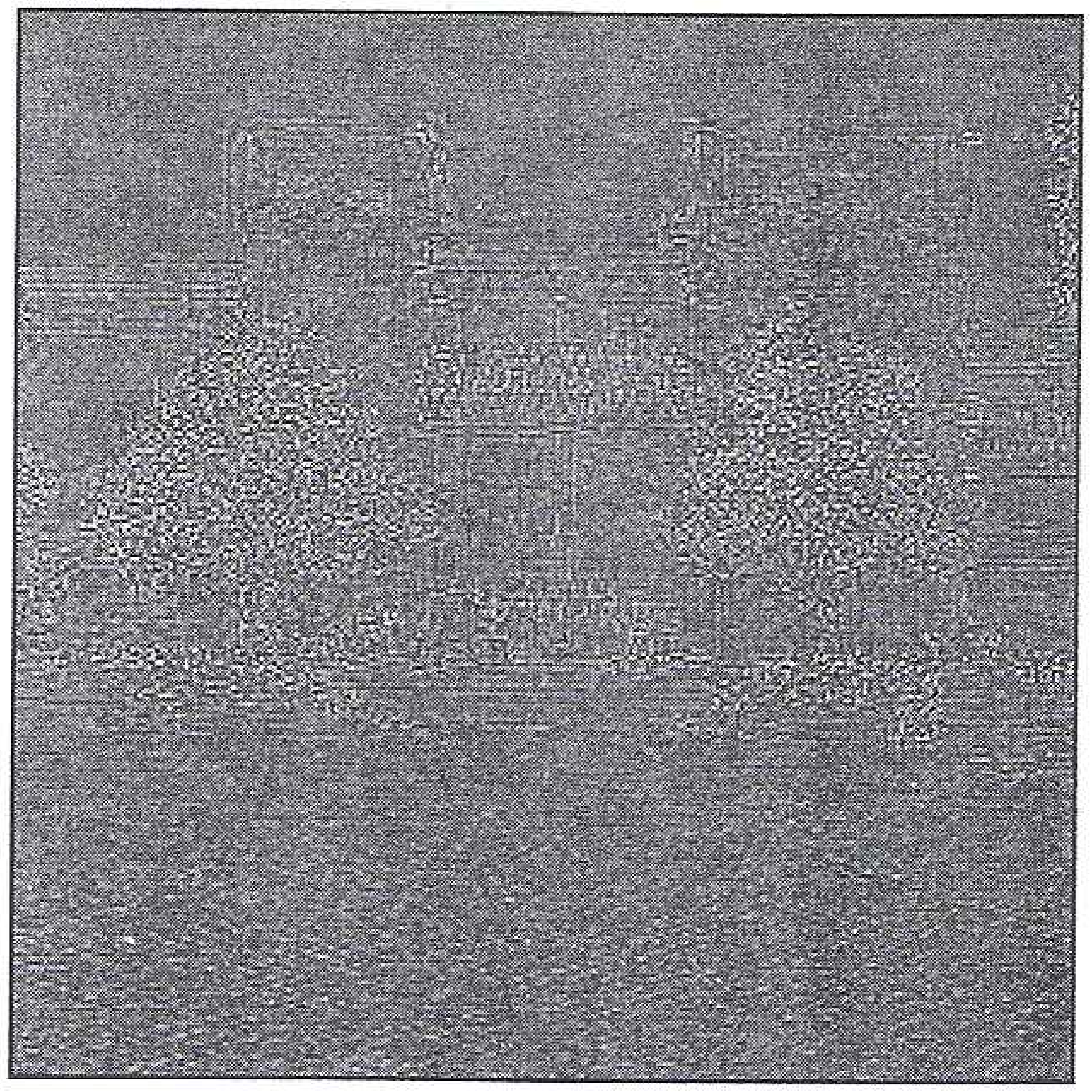
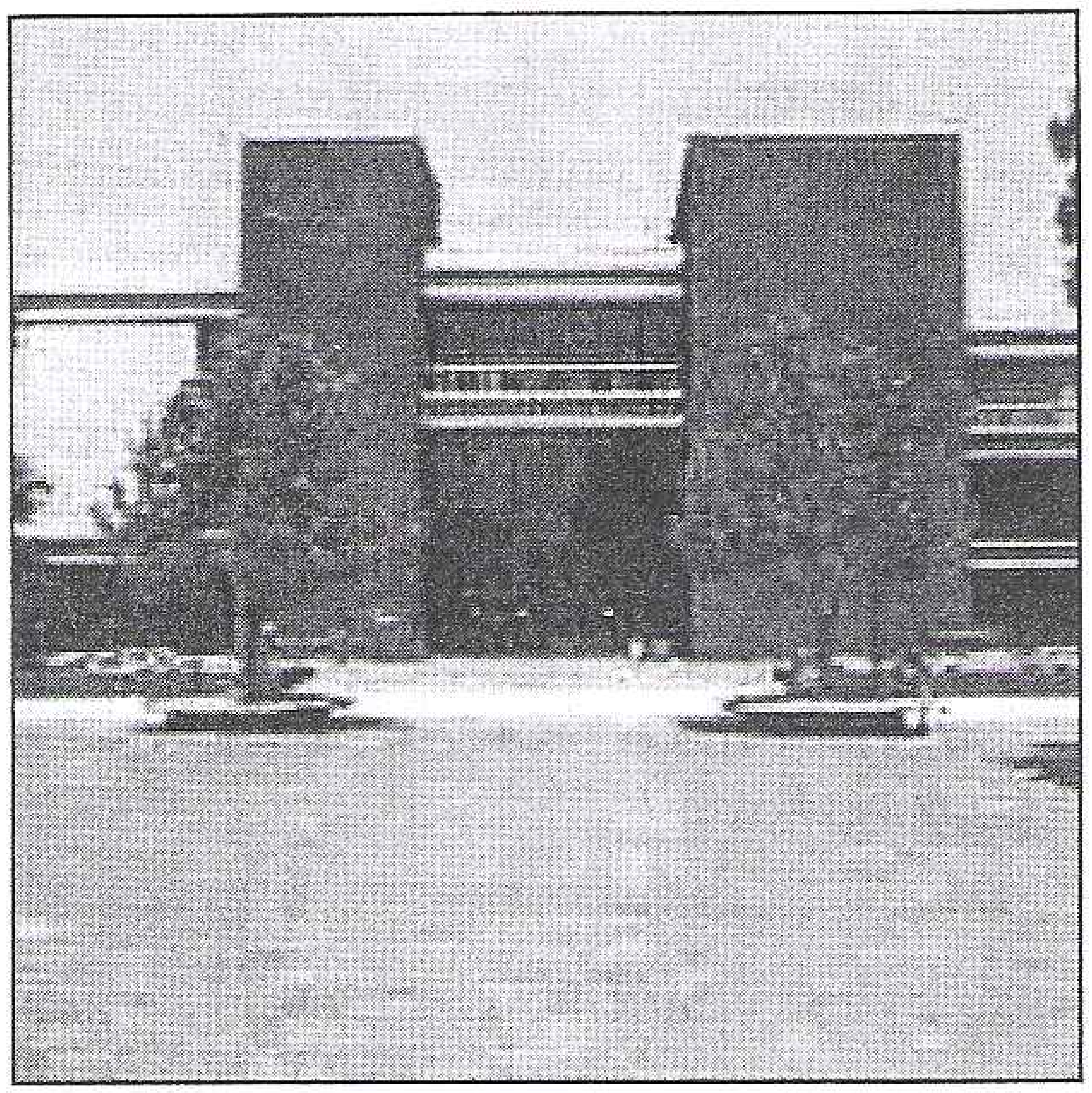
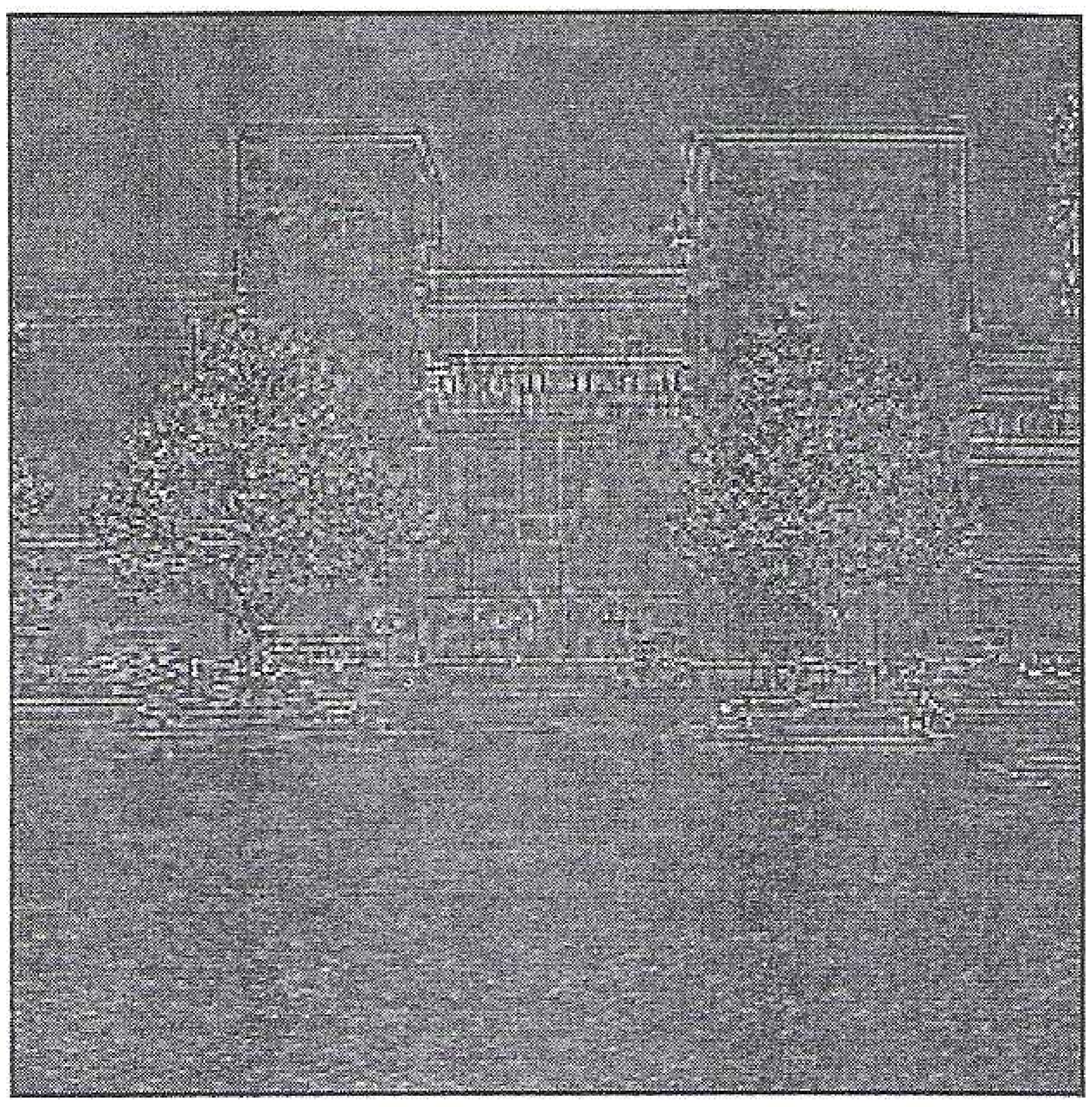
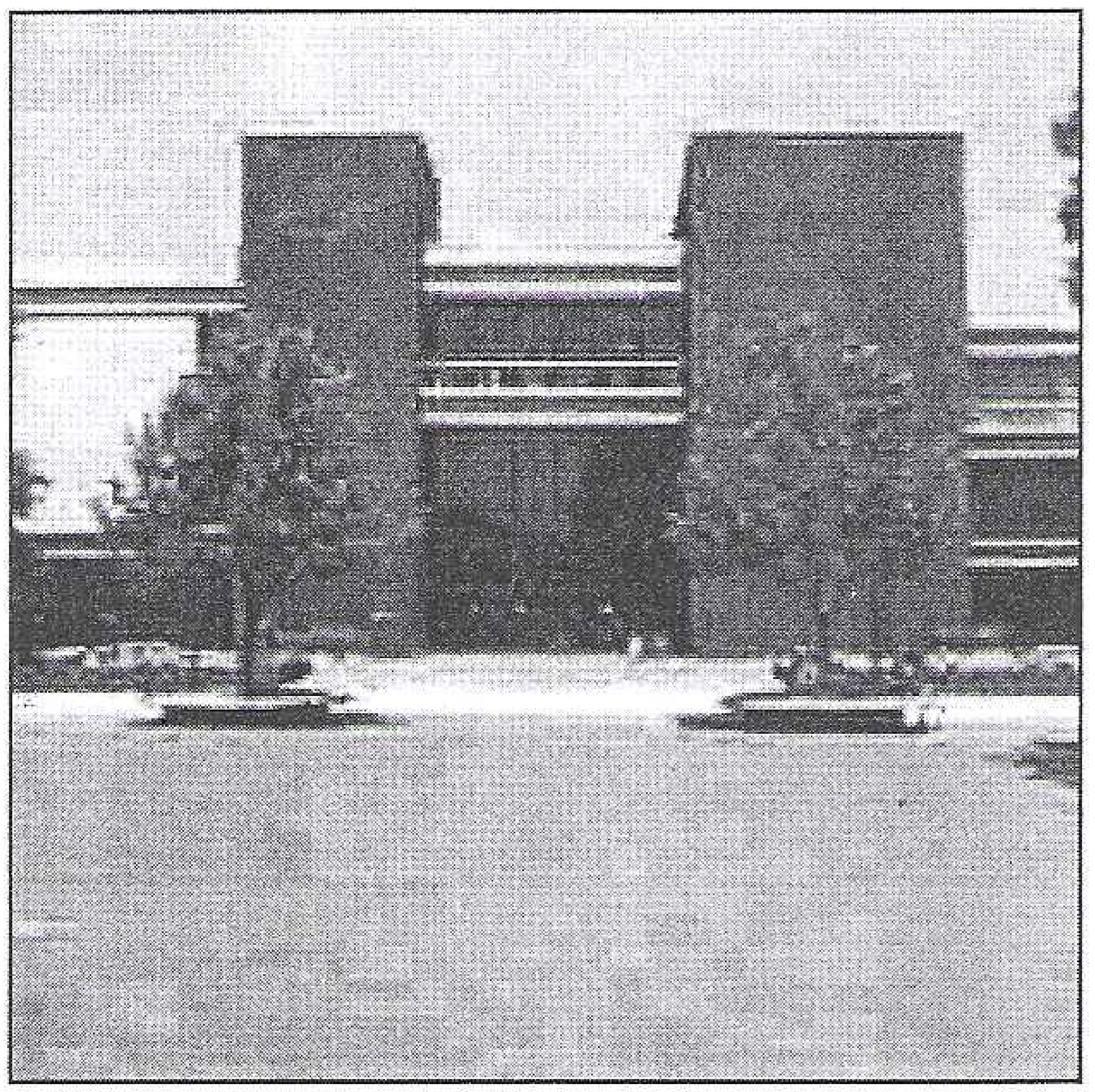
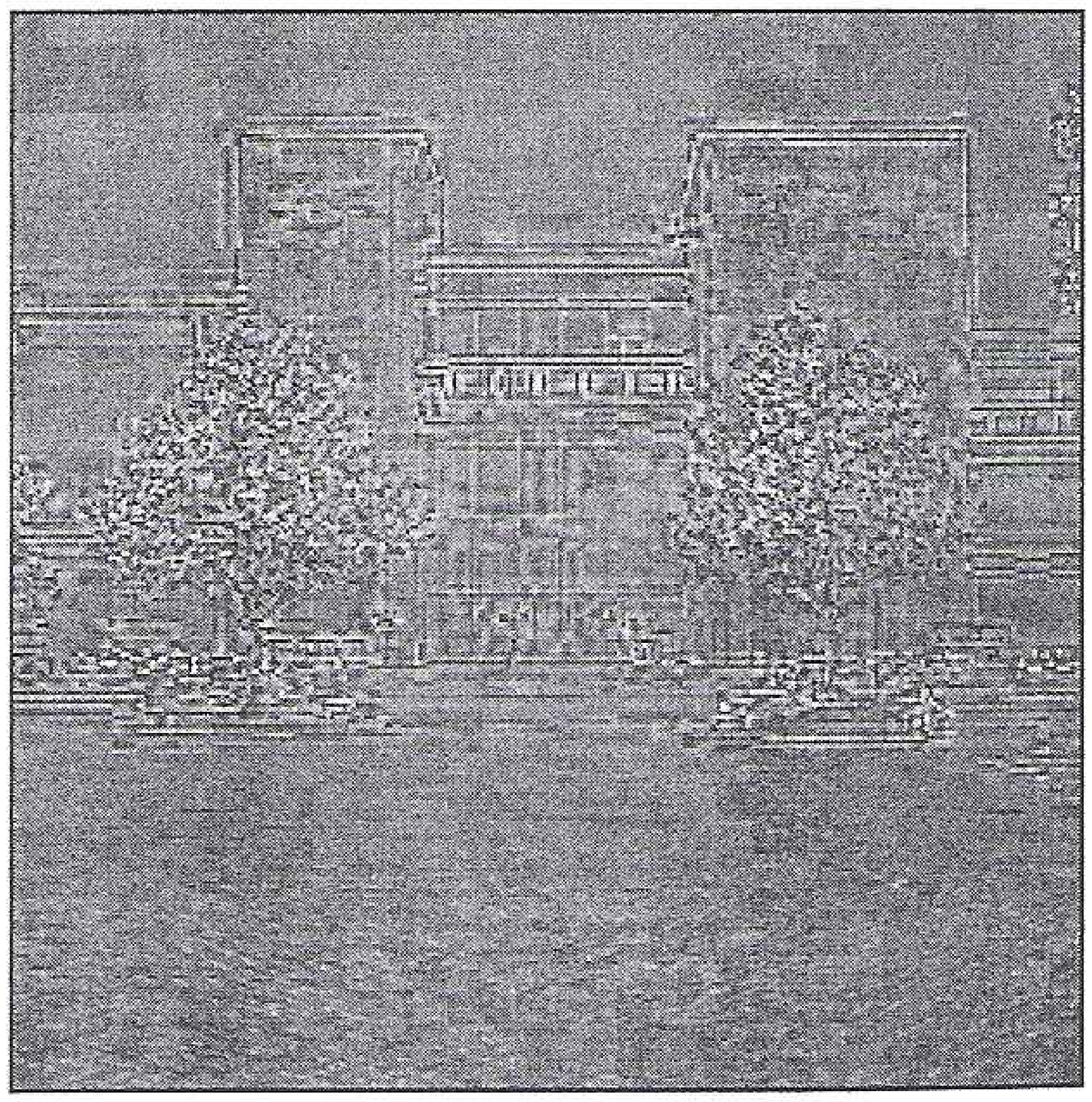

c. Error image for (b), multiplied by 8 to show detail.

Figure 5.3-14 (Continued)


d. JPEG compression = 20:1.

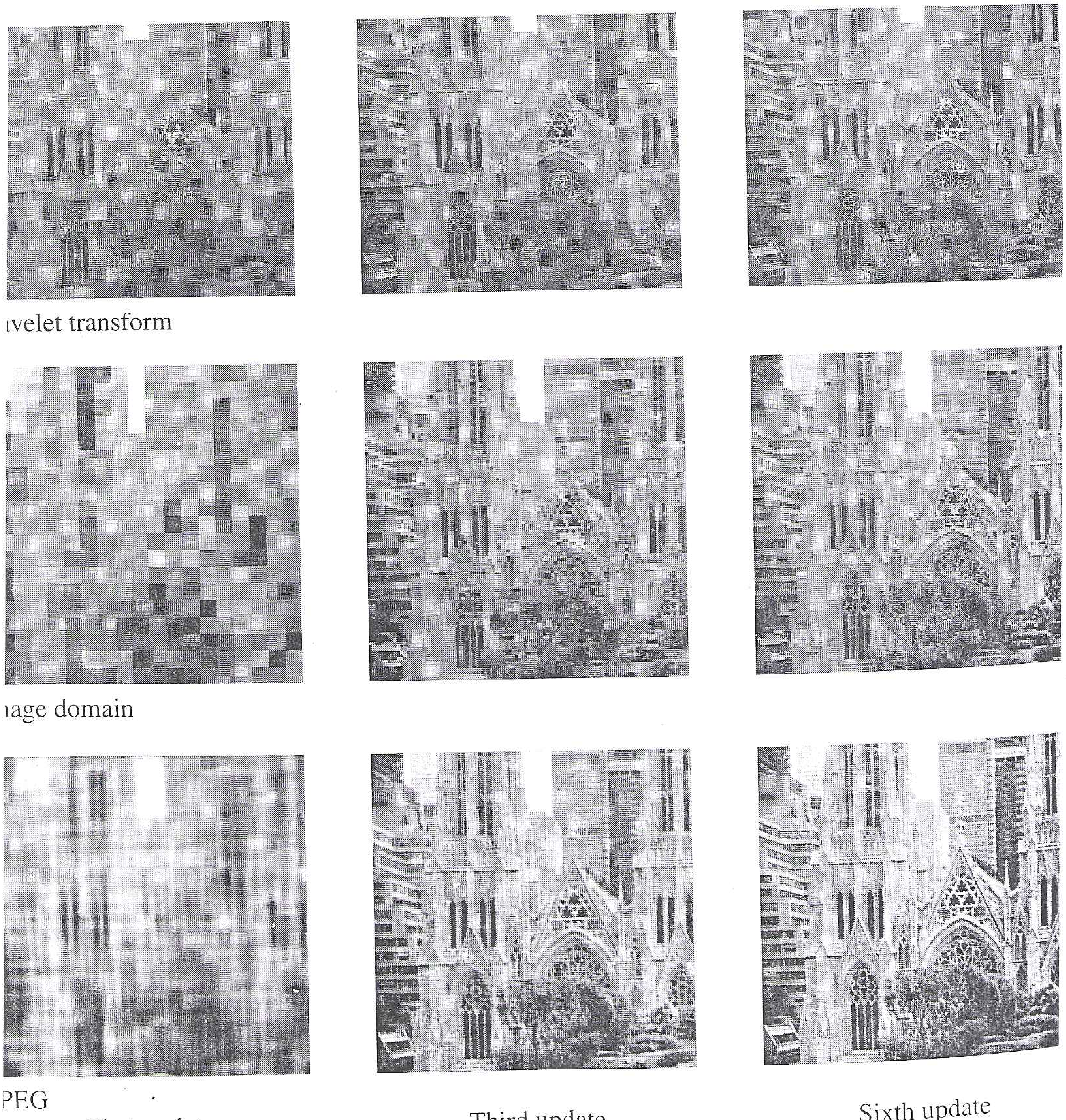

e. Error image for (d), multiplied by 8 to show detail.


f. JPEG compression = 30:1.


g. Error image for (f), multiplied by 8 to show detail.

Model-based image compression can be considered a hybrid method, althoug the transform used may be an object-based transform. *Model-based compressio* works by finding models for objects within the image and using model parameters fc the compressed file. The objects are often defined by lines or shapes (boundaries), so Hough transform may be used, whereas the object interiors can be defined by statist cal texture modeling. Methods have also been developed that use texture modeling i the wavelet domain. The model-based methods can achieve very high compressio ratios, but the decompressed images often have an artificial look to them.

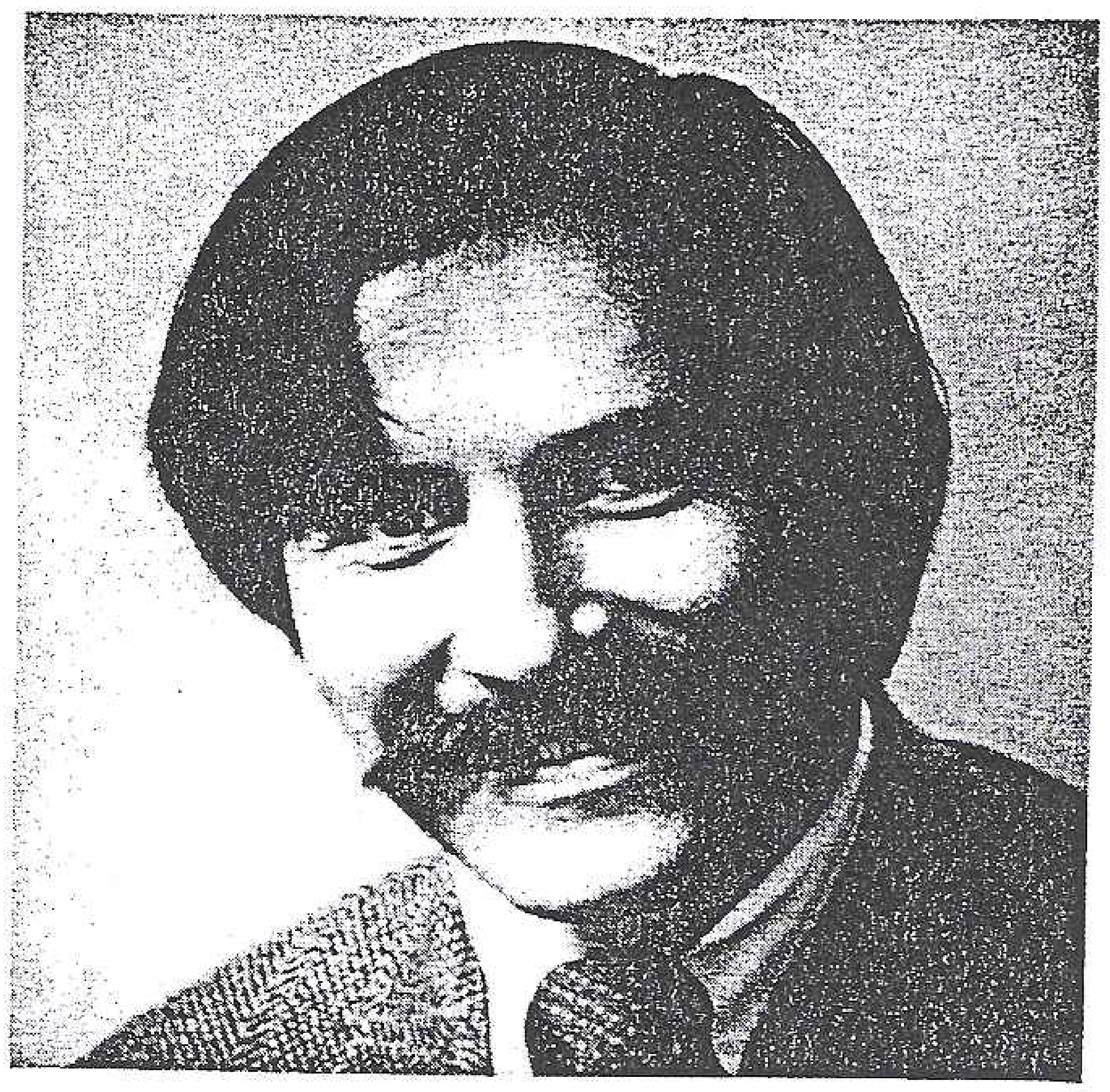
images, superior for progressive transmission. A simple comparison is shown i Figure 26.5 which shows three progressive sequences, a wavelet series, a sequenc generated by resampling the image at progressively higher resolution and a JPE(series. The series is for a single, three and six updates containing 0.14%, 4.12% and 8.24% respectively. The algorithm used to generate the wavelet series is no described. The progressive mode of JPEG is described in Section 26.3.3.

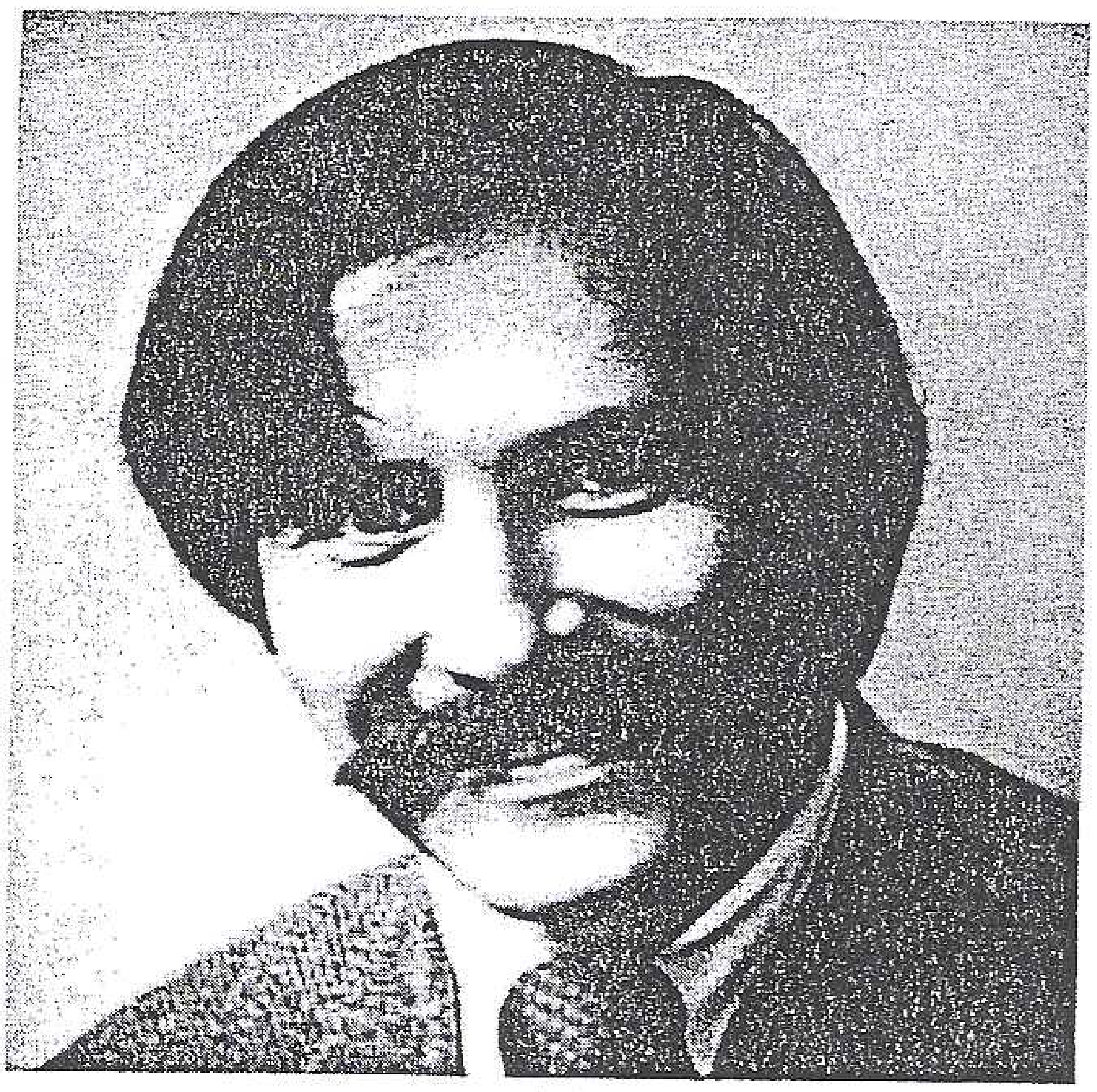


26.5
ɔrogression and
s.

Degradations due to transform coding. Quantization noise in the reconstructed image manifests itself in transform image coding differently from waveform image coding. In general, the effect of quantization noise is less localized in transform image coding. Quantization of one transform coefficient affects all the image intensities within the subimage.

Several types of image degradation result from quantization noise in transform image coding. One type is loss of spatial resolution. In the DCT coding of images, the discarded transform coefficients are typically high-frequency components. The result is loss of detail in the image. An example of this type of degradation is shown in Figure 10.45. Figure 10.45(a) shows an original image of 512 × 512 pixels. Figures 10.45(b) and (c) show images reconstructed by retaining 14% and 8% of the DCT coefficients in each subimage, respectively. The subimage size used is 16 × 16 pixels. The transform coefficients retained are not quantized,


(a)


(b)

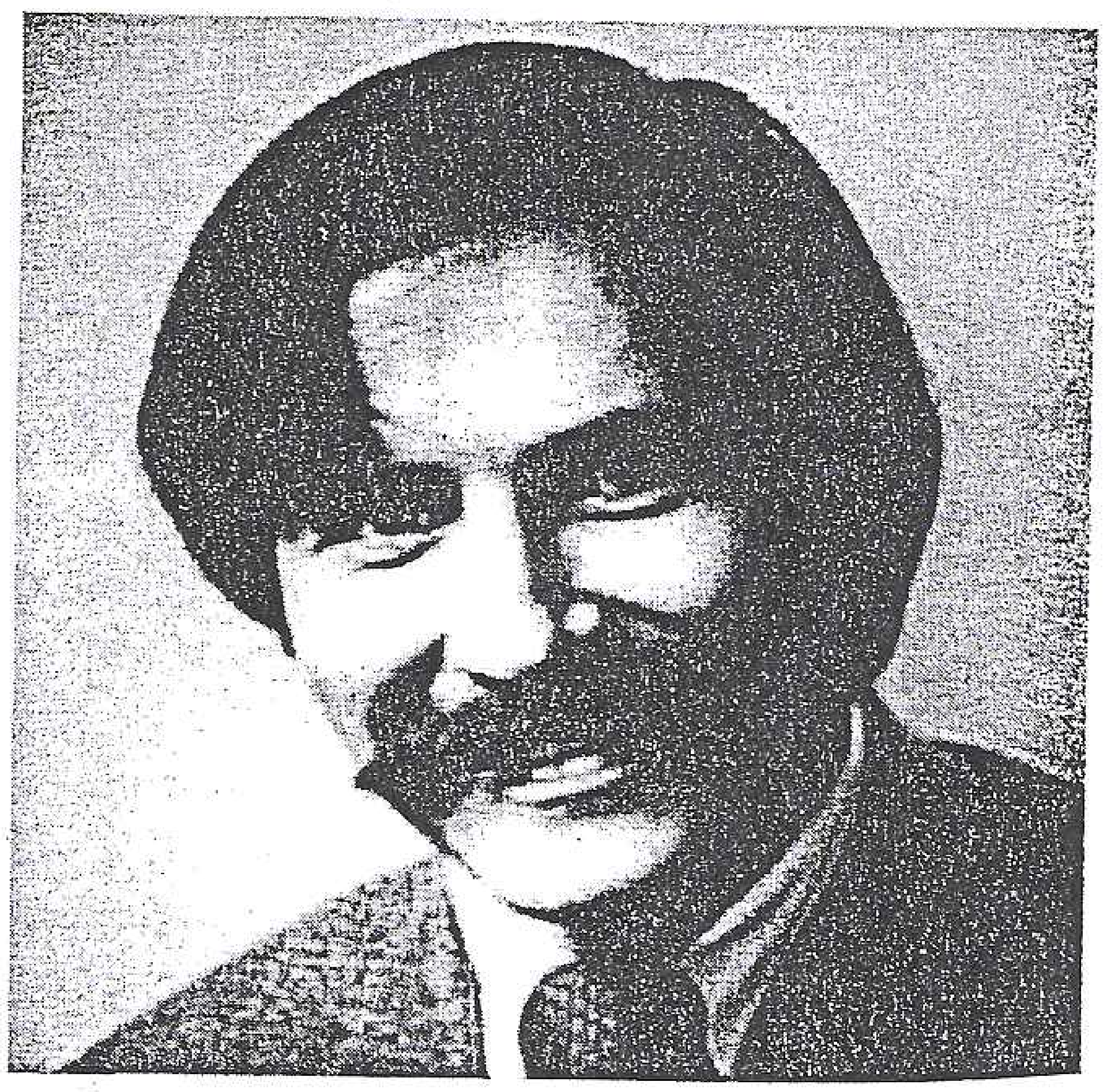

(c)

Figure 10.45 Illustration of spatial resolution loss due to discarding discrete cosine transform (DCT) coefficients.(a) Original image of 512 × 512 pixels. (b) reconstructed image with 14% of DCT coefficients retained. NMSE = 0.8%, SNR = 21.1 dB. (c) reconstructed image with 8% of DCT coefficients retained. NMSE = 1.2%, SNR = 19.3 dB.

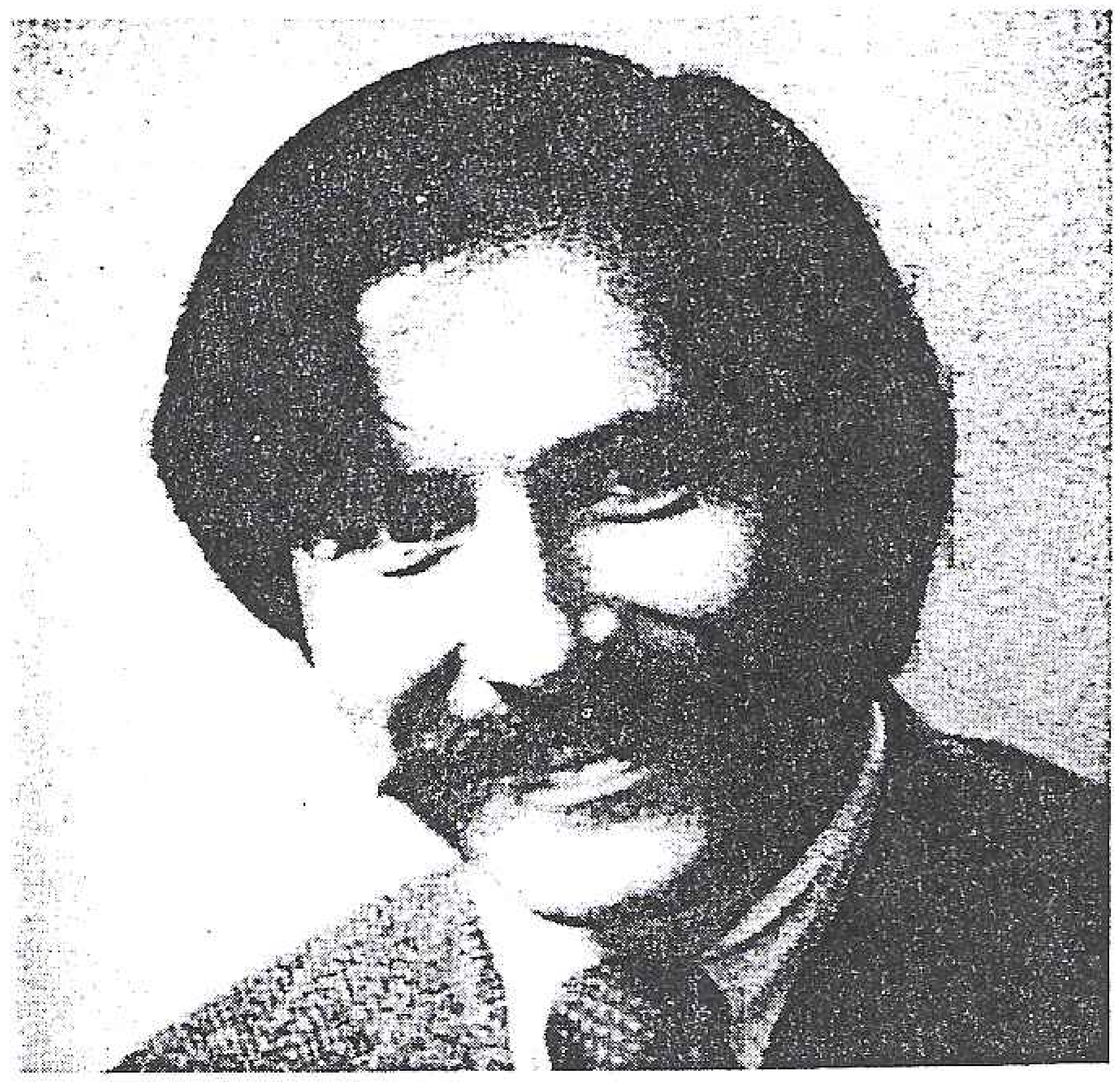


Figure 10.46 Illustration of graininess increase due to quantization of DCT coefficients. A 2-bit/pixel uniform quantizer was used to quantize each DCT coefficient retained to reconstruct the image in Figure 10.45(b).

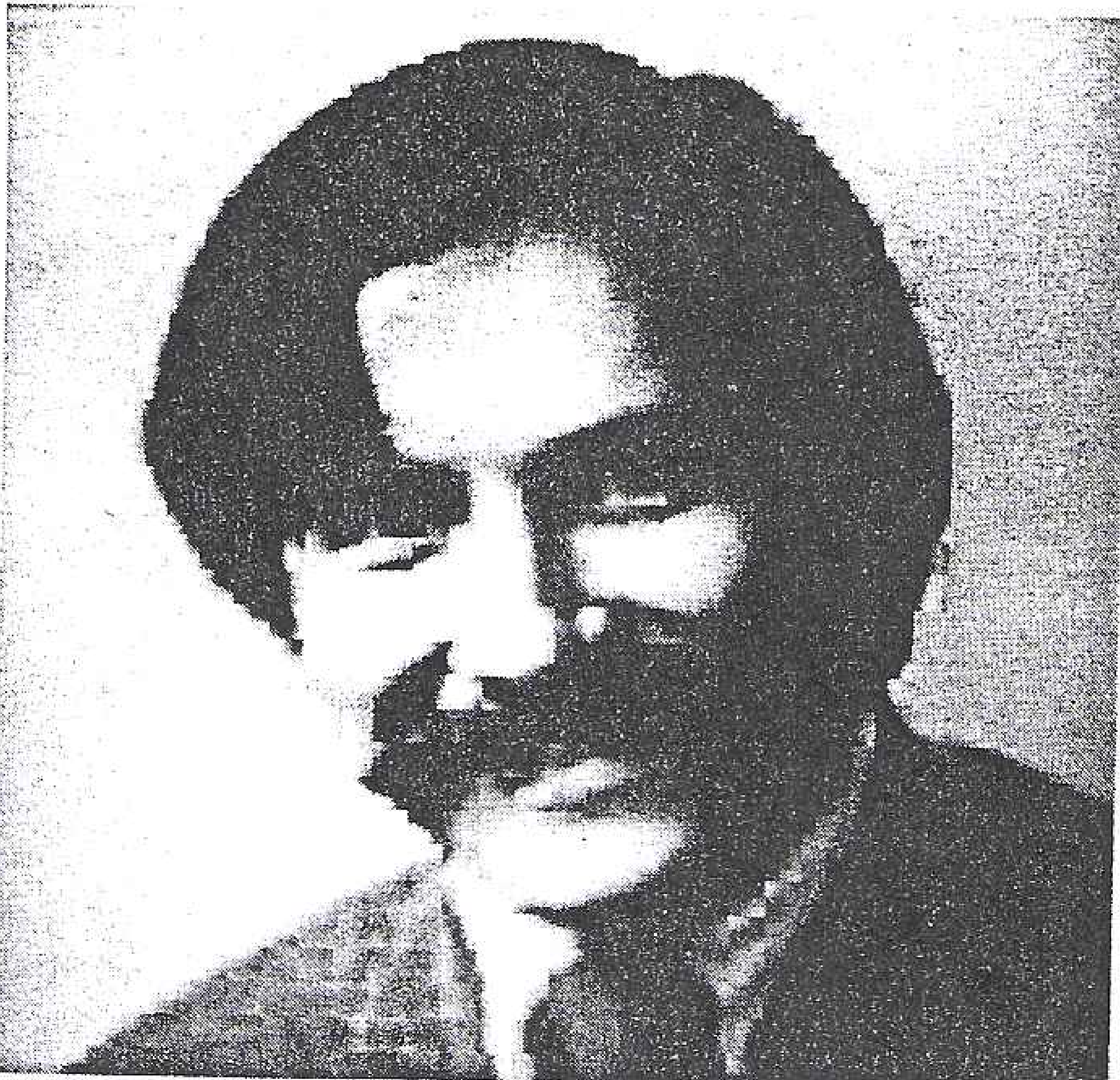
and are selected from a zone of triangular shape shown in Figure 10.43(a). From Figure 10.45, it is clear that the reconstructed image appears more blurry as we retain a smaller number of coefficients. It is also clear that an image reconstructed from only a small fraction of the transform coefficients looks quite good, illustrating the energy compaction property.

Another type of degradation results from quantization of the retained transform coefficients. The degradation in this case typically appears as graininess in the image. Figure 10.46 shows the result of coarse quantization of transform coefficients. This example is obtained by using a 2-bit uniform quantizer for each retained coefficient to reconstruct the image in Figure 10.45(b).

A third type of degradation arises from subimage-by-subimage coding. Since each subimage is coded independently, the pixels at the subimage boundaries may have artificial intensity discontinuities. This is known as the *blocking effect*, and is more pronounced as the bit rate decreases. An image with a visible blocking effect is shown in Figure 10.47. A DCT with zonal coding, a subimage of 16×16 pixels, and a bit rate of 0.15 bit/pixel were used to generate the image in Figure 10.47.

Examples. To design a transform coder at a given bit rate, different types of image degradation due to quantization have to be carefully balanced by a proper choice of various design parameters. As was discussed, these parameters include the transform used, subimage size, selection of which coefficients will be retained, bit allocation, and selection of quantization levels. If one type of degradation dominates other types of degradation, the performance of the coder can usually be improved by decreasing the dominant degradation at the expense of some increase in other types of degradation.

Figure 10.48 shows examples of transform image coding. Figure 10.48(a)



10.4.3 Reduction of Blocking Effect

When the bit rate is sufficiently low, the blocking effect, which results from independent coding of each subimage, becomes highly visible. Reconstructed images exhibiting blocking effects can be very unpleasant visually, and blocking effects that are clearly visible often become the dominant degradation.

Two general approaches to reducing the blocking effect have been considered. In one approach, the blocking effect is dealt with at the source. An example of this approach is the overlap method, which modifies the image segmentation process. A typical segmentation procedure divides an image into mutually exclusive regions. In the overlap method, the subimages are obtained with a slight overlap around the perimeter of each subimage. The pixels at the perimeter are coded in two or more regions. In reconstructing the image, a pixel that is coded more than once can be assigned an intensity that is the average of the coded values. Thus, abrupt boundary discontinuities caused by coding are reduced because the reconstructed subimages are woven together. An example of the overlap method is shown in Figure 10.49. In the figure, a 5 × 5-pixel image is divided into four 3 × 3-pixel subimages by using a one-pixel overlap scheme. The shaded area indicates pixels that are coded more than once. The overlap method reduces blocking effects well. However, some pixels are coded more than once, and this increases the number of pixels coded. The increase is about 13% when an image of 256 × 256 pixels is divided into 16 × 16-pixel subimages with a one-pixel overlap. This increase shows why overlap of two or more pixels is not very useful. It also shows a difference between image coding and other image processing applications such as image restoration in dealing with blocking effects. As was discussed in Section 9.2.3, a blocking effect can occur in any subimage-by-subimage processing environment. In image restoration, the cost of overlapping subimages is primarily an increase in the number of computations. An overlap of 50% of the subimage size is common in subimage-by-subimage restoration. In image coding, however, the cost of overlapping subimages is an increase in the number of computations and, more seriously, a potential increase in the required bit rate. An overlap of more than one pixel is thus seldom considered in DCT image coding.

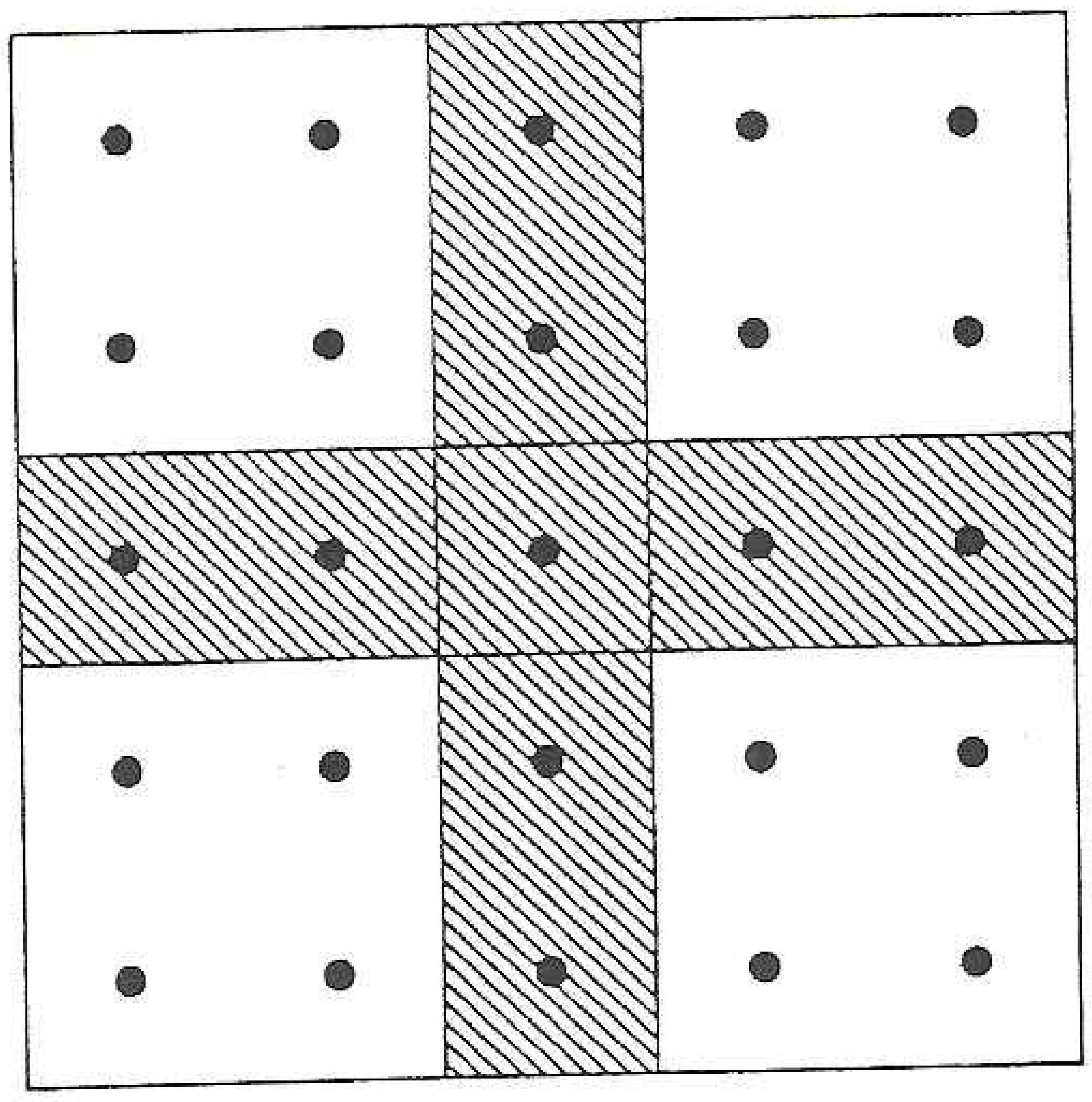


Figure 10.49 Example of one-pixel overlap in the overlap method for reducing blocking effect.

10.6.2 Color Image Coding

Thus far we considered the problem of coding monochrome images. Many of the monochrome image coding methods can be extended directly to color image coding. As discussed in Section 7.1.4, a color image can be viewed as three monochrome images $f_R(n_1, n_2)$, $f_G(n_1, n_2)$, and $f_B(n_1, n_2)$, representing the red, green, and blue components. Each of the three components can be considered a monochrome image, and the coding methods we discussed can be applied directly.

The three components $f_R(n_1, n_2)$, $f_G(n_1, n_2)$, and $f_B(n_1, n_2)$ are highly correlated with each other, and coding each component separately is not very efficient. One approach that exploits the correlation is to transform the three components to another set of three components with less correlation. One such set is $f_Y(n_1, n_2)$, $f_I(n_1, n_2)$, and $f_Q(n_1, n_2)$, where $f_Y(n_1, n_2)$ is the luminance component and $f_I(n_1, n_2)$ and $f_Q(n_1, n_2)$ are the two chrominance components. The linear 3×3 matrix transformation between R, G, and B components and Y, I, and Q components was discussed in Section 7.1.4. The approach to transform the R, G, and B components to the Y, I, and Q components and then code the Y, I, and Q components is illustrated in Figure 10.57.

One advantage of coding Y, I, and Q components rather than R, G, and B components is that most high-frequency components of a color image are concentrated in the Y component. Therefore, the chrominance components I and Q can be undersampled by a factor of 2×2 to 4×4 in waveform coding without seriously

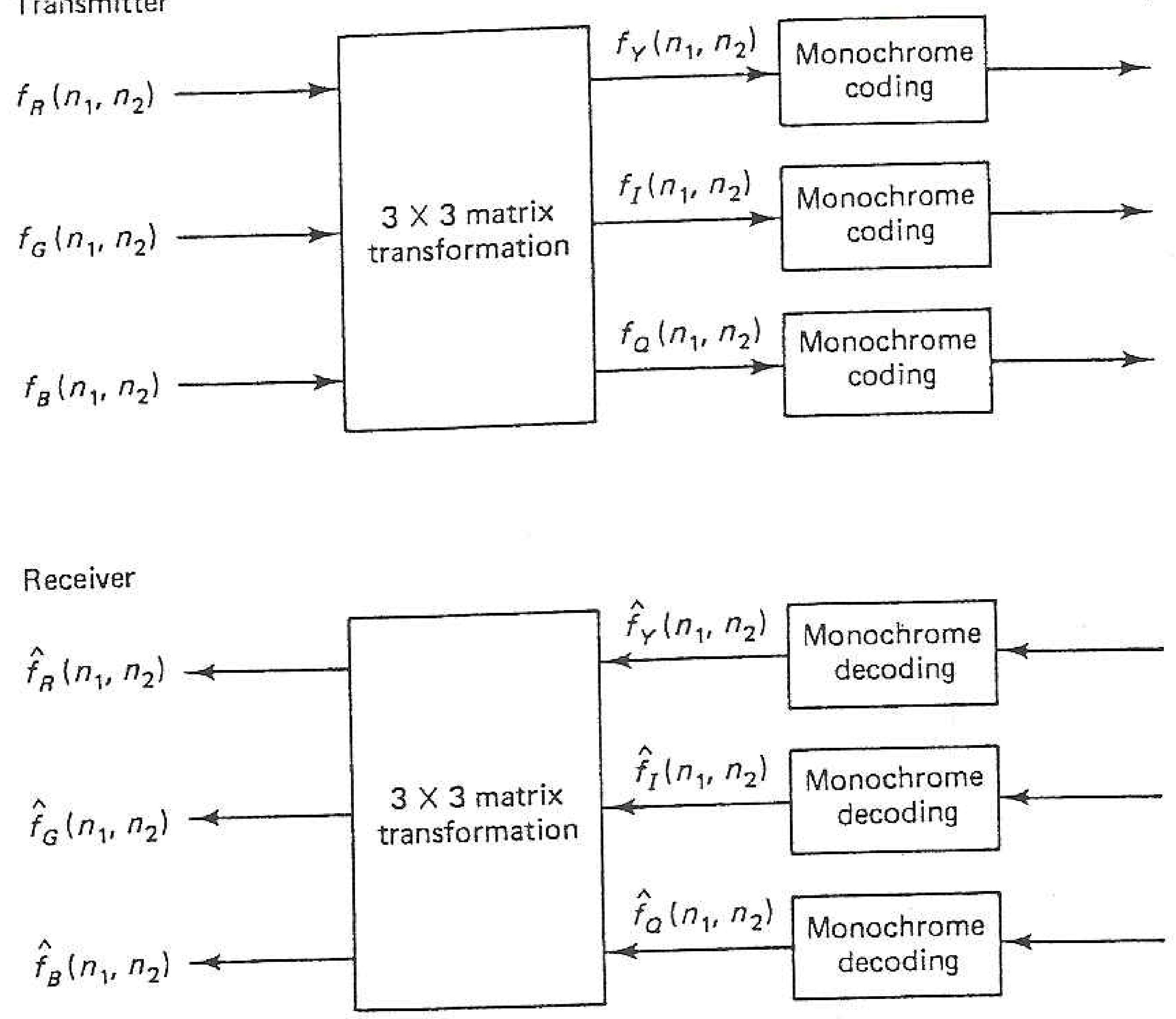


Figure 10.57 . Color image coding in the *YIQ* space.

affecting the high-frequency details of a color image. In transform coding, a smaller number of coefficients needs to be coded for the I and Q components than for the Y component. Typically, the total number of bits assigned to both the I and Q components is only approximately half the number of bits used to code the Y component; adding color does not increase the bit rate by a factor of three. In addition, the aesthetics of color lead to a more visually pleasant reconstructed image and tend to hide degradations in the image. A color image coded at a given bit rate typically looks better than a black-and-white image obtained by coding only the Y component of the color image at the same total bit rate.

Two examples of color image coding are shown in Figures 10.58 and 10.59 [illegible] Figure 10.58(a) shows an original color image of 512×512

Fractal image compression

The most commonly used forms of image compression use combinations of classical techniques, like JPEG, discussed in the previous section, to effect a lossy

	a	b	c	d	e	f
p	0.85	0.04	–0.04	0.85	0.02	0.08
q	–0.13	0.24	0.22	0.20	0.12	–0.27
r	0.18	–0.24	0.21	0.20	–0.12	–0.30
s	0.0	0.0	0.0	0.16	0.0	–0.42