The JPEG image compression technique consists of 5 functional stages.

1. an RGB to YCC color space conversion,
2. a spatial subsampling of the chrominance channels in YCC space,
3. the transformation of a blocked representation of the YCC spatial image data to a frequency domain representation using the discrete cosine transform,
4. a quantization of the blocked frequency domain data according to a user-defined quality factor, and finally
5. the coding of the frequency domain data, for storage, using Huffman coding.

The human visual system relies more on spatial content and acuity than it does on color for interpretation. For this reason, a color photograph, represented by a red, green, and blue image, is transformed to different color space that attempts to isolate these two components of image content; namely the YCC or luminance/chrominance-red/chrominance-blue color space. This color space transformation is performed on a pixel-by-pixel basis with the digital counts being converted according to the following rules

$$Y = 0.2990R + 0.5870G + 0.1140B$$
$$C_b = -0.1687R - 0.3313G + 0.5B + 2^{Bit\ Depth-1}$$
$$C_r = 0.5R - 0.4187G - 0.0813B + 2^{Bit\ Depth-1}$$

An example transformation is shown below.

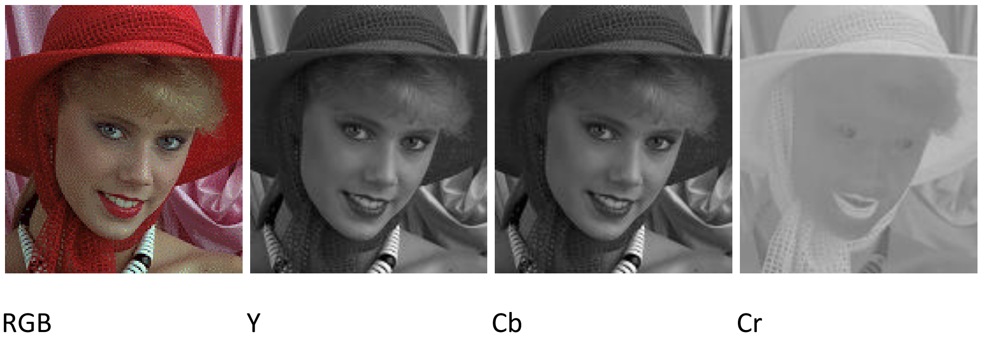


Figure 3. RGB to YCC Conversion - The original RGB image and the computed luminance (Y), chrominance-blue (Cb), and chrominance-red (Cr) images.

The luminance image carries the majority of the spatial information of the original image and is indeed just a weighted average of the original red, green, and blue digital count values for each pixel. The two chrominance images show very little spatial detail. This is fortuitous for the goal of compression.

The JPEG process subsamples the individual chrominance images before proceeding to half the number of individual rows and columns. Since there is little spatial detail in these channels, the subsampling does not discard much meaningful data. This results in one quarter of the number of pixels where in the original representations. The human visual system is however, easily fooled and the resulting true color image that is formed by inverting this subsampling/color space transformation process is virtually indistinguishable from the original unless viewed at very high magnifications.

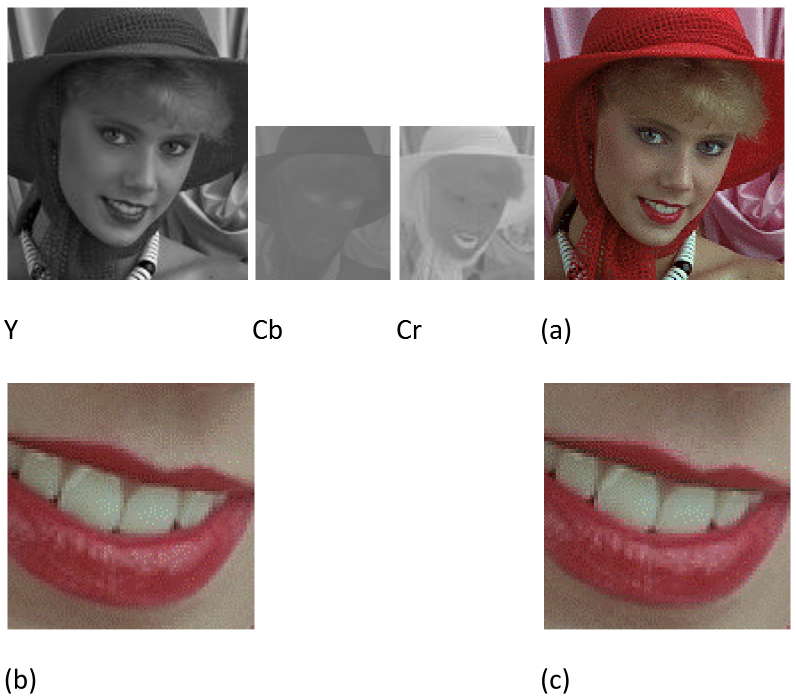


Figure 4. Inverse subsampling/color space transform - The result of the (a) inverse subsampling/color space transformation is virtually indistinguishable from the original image. The effects of subsampling can be seen in the magnified image subsections shown for the (b) original image and the resulting (c) inverse transformed image.

- 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.

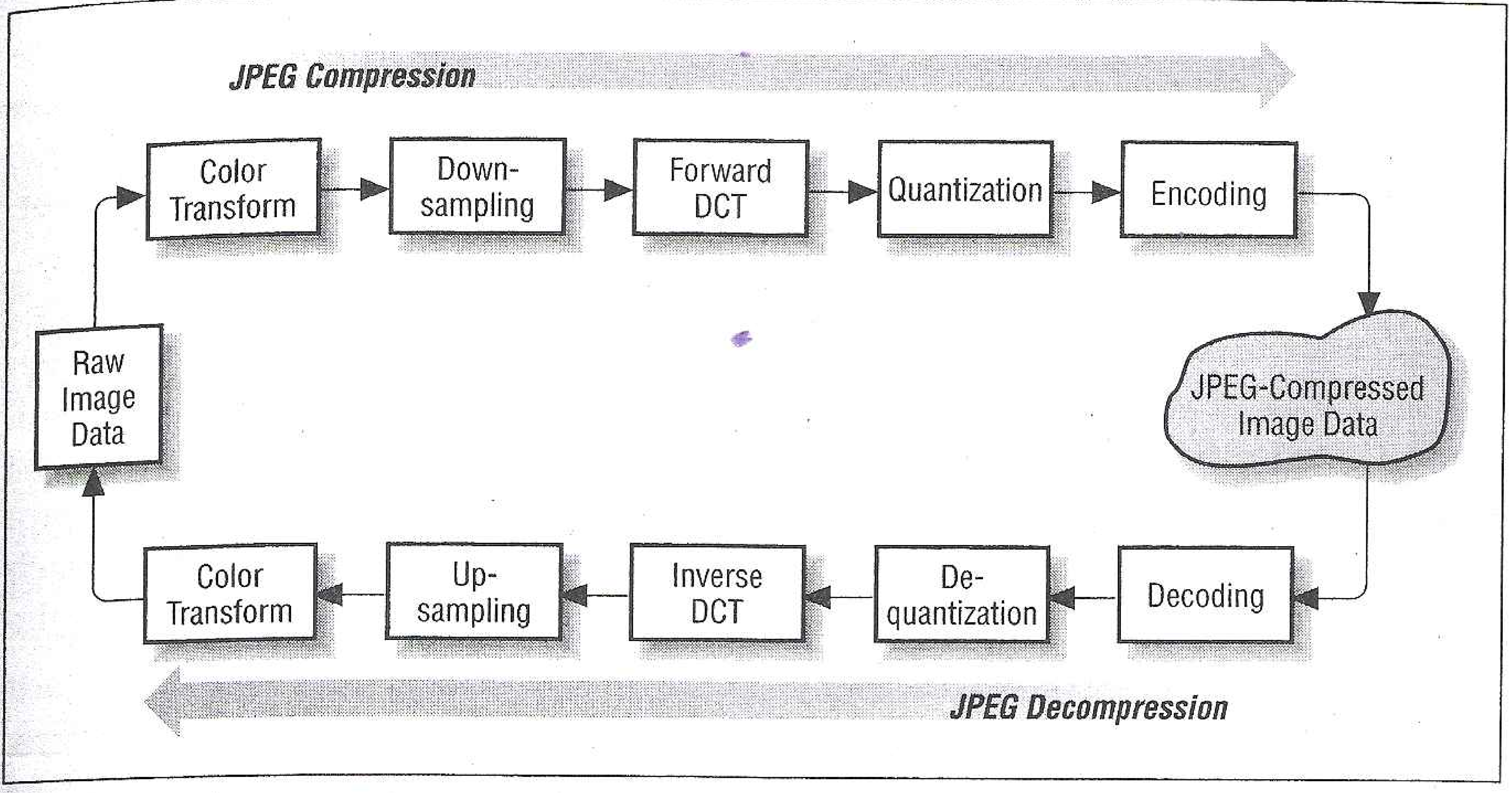


FIGURE 9-11: *JPEG compression and decompression*

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

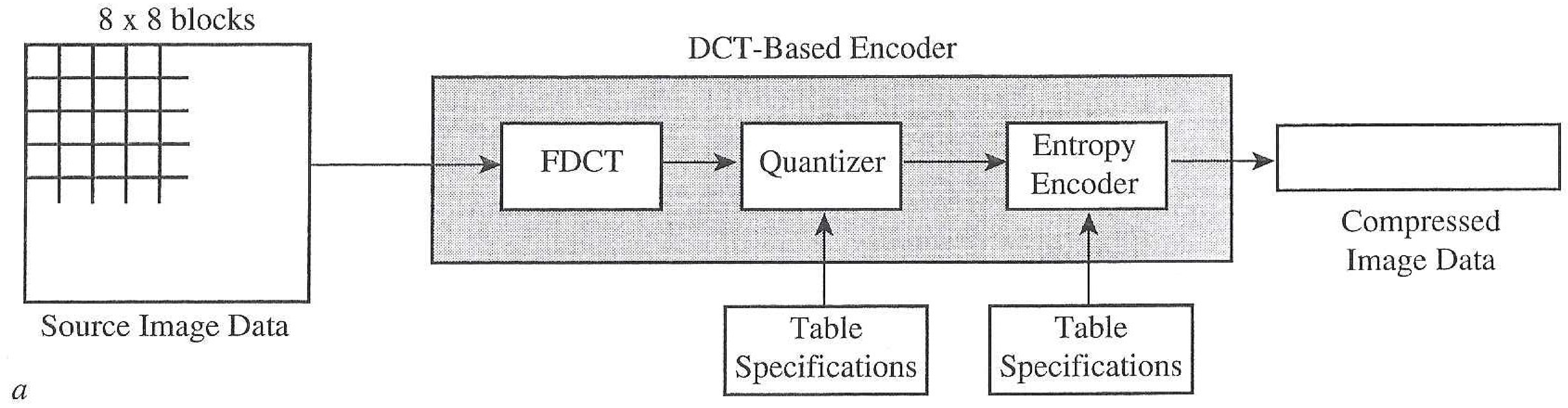
To discard an appropriate amount of information, the compressor divides each DCT output value by a "quantization coefficient" and rounds the result to an integer. The larger the quantization coefficient, the more data is lost, because the actual DCT value is represented less and less accurately. Each 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



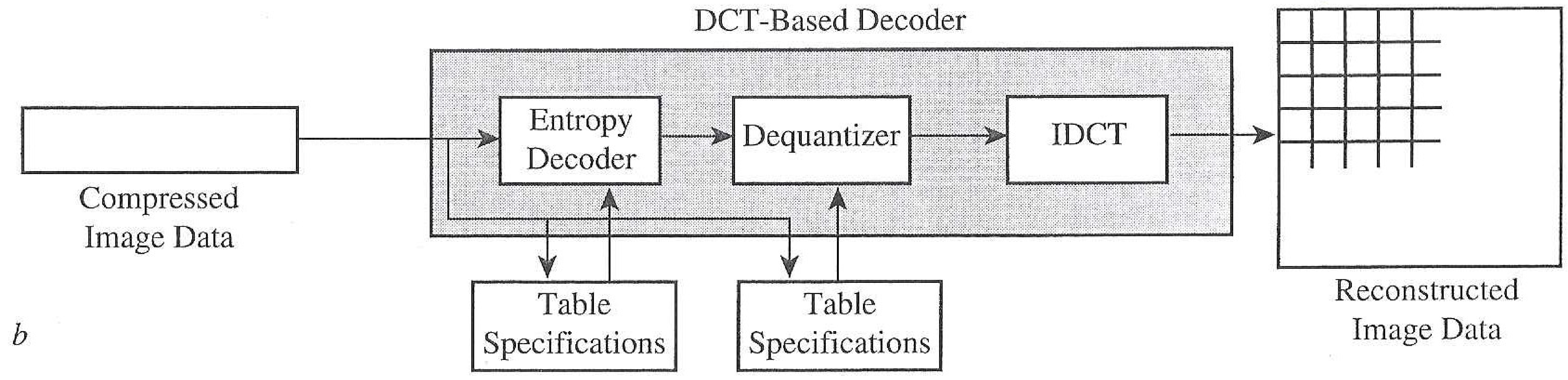


FIGURE 9.1 JPEG: *(a)* Encoder; *(b)* Decoder

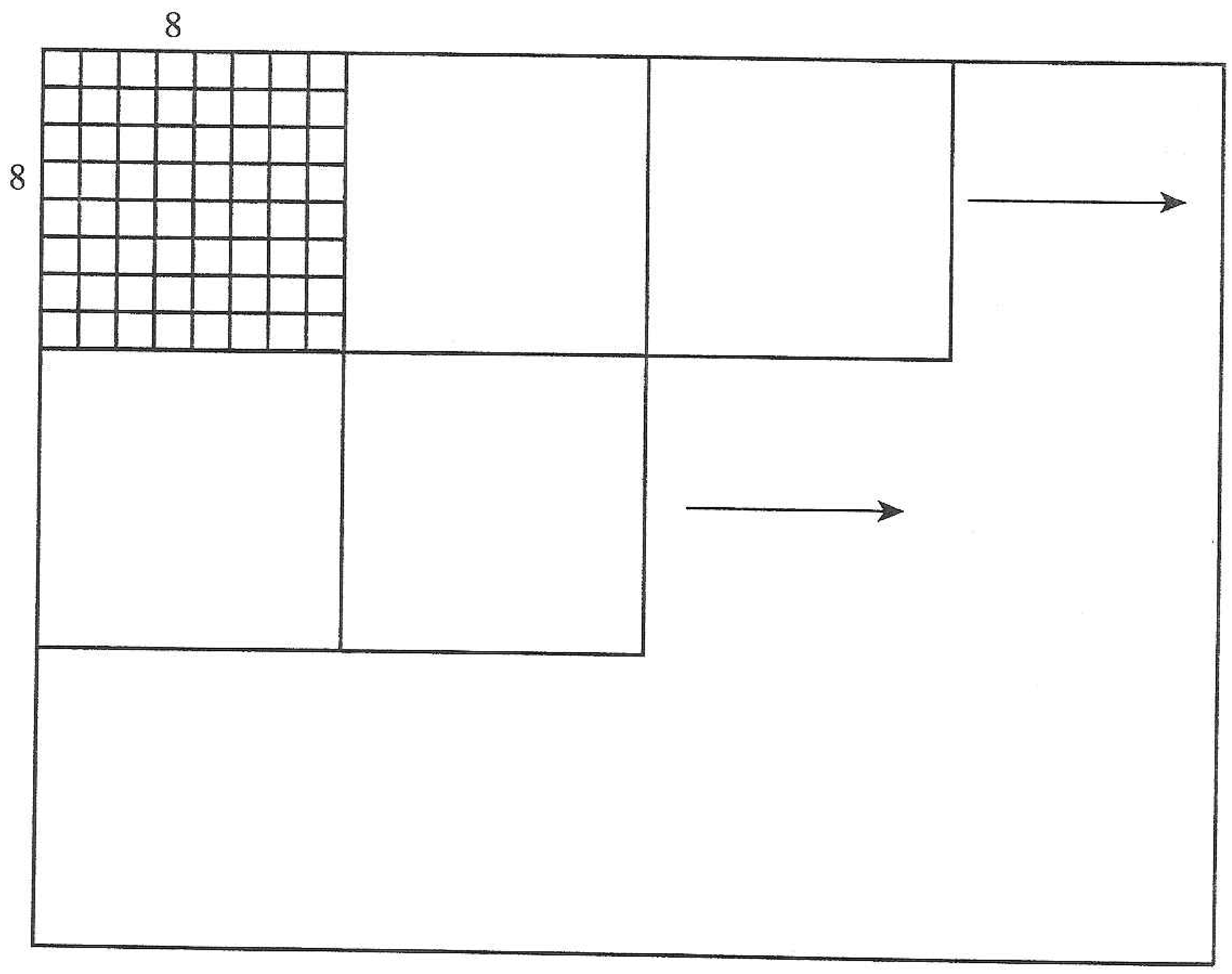


FIGURE 9.2 Partitioning of an Image into 8 × 8 Blocks of Pixels

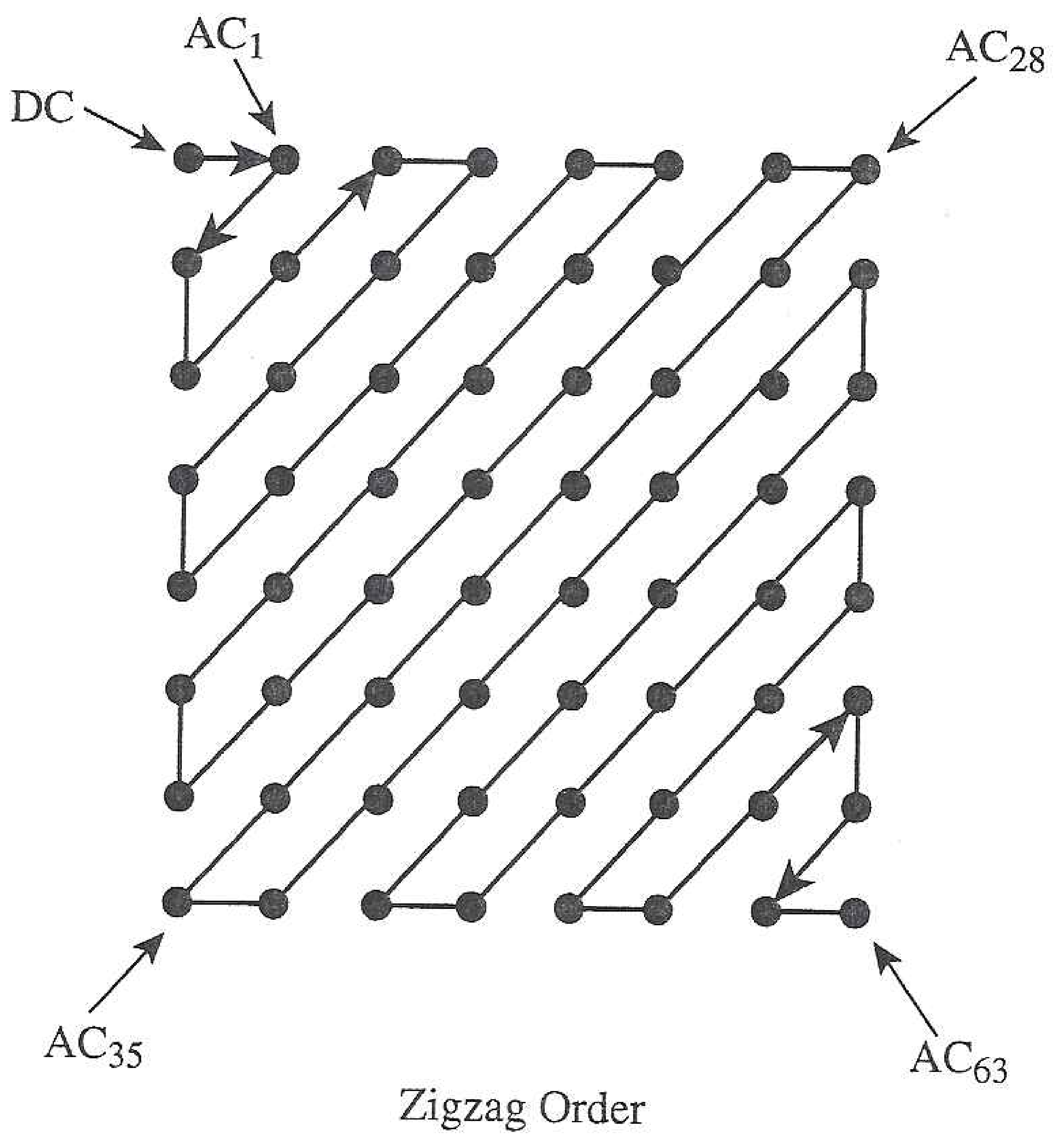


FIGURE 9.3 Zigzag Coefficient Ordering

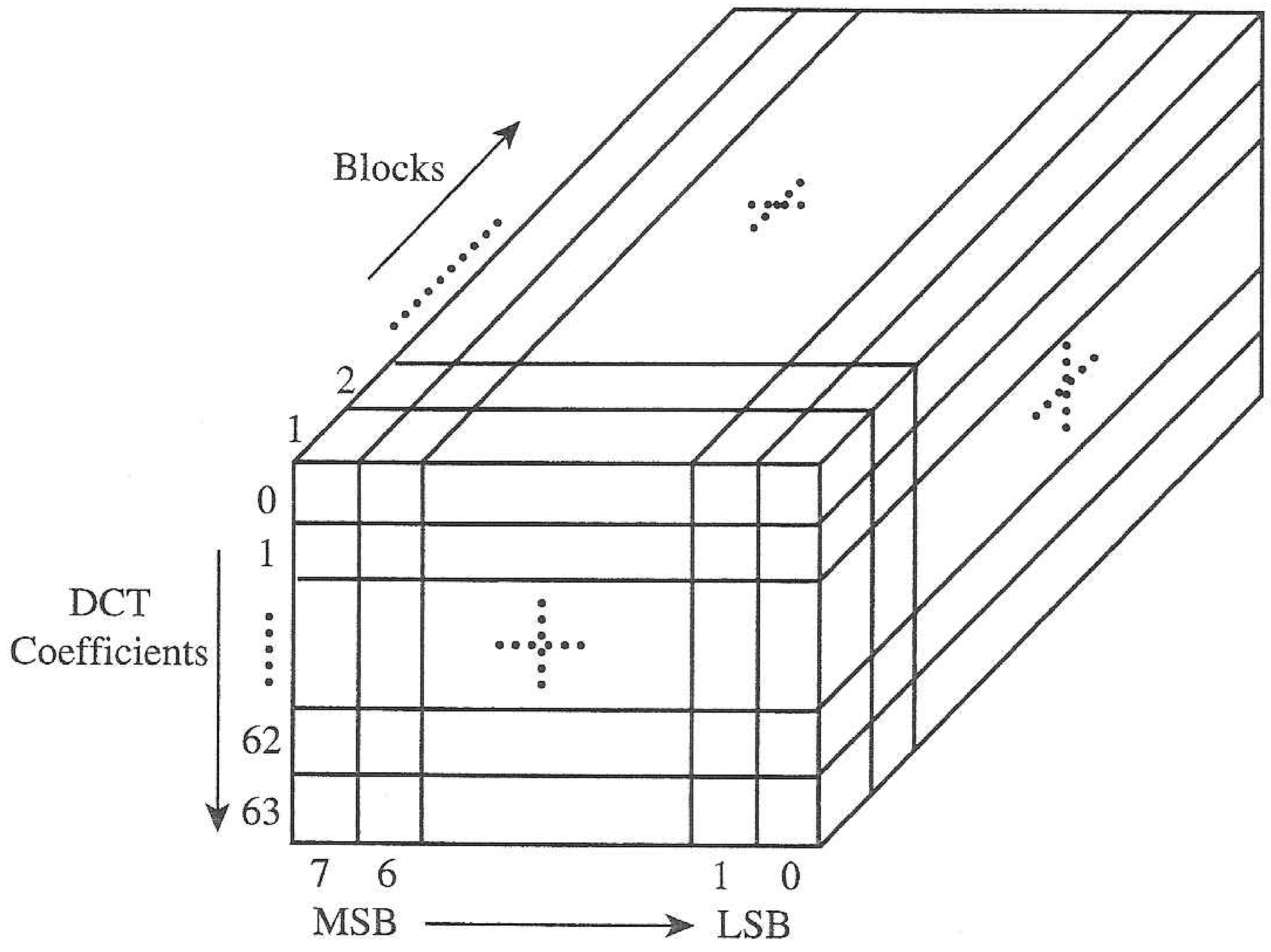


FIGURE 9.4 Sequential Lossy Encoding

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.

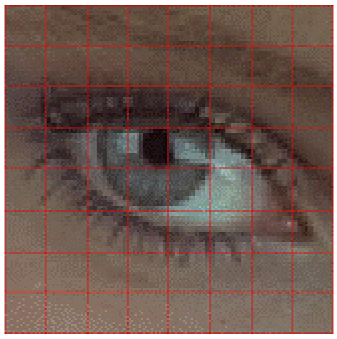


Figure 5. Image blocks - A small section of the image previously shown that has been segmented into blocks that are 8x8 pixels in size.

As the first two phases of the JPEG process attempt to take advantage of the weaknesses in the human visual system and reduce psycho-visual redundancy, the next phase attempts to exploit the inter-pixel redundancy present in most image data. If an image is broken up into small subsections or blocks, the likelihood that the pixels in these blocks will have similar digital count levels is high for the majority of the blocks throughout the image. Blocks that include high contrast image features such as edges will obviously not exhibit this behavior.

As can be seen in the previous figure, almost half of the blocks shown contain skin-toned pixels with very little high frequency information. The advantageous result of this fact is that the frequency-domain representation of the data in any one of these blocks that exhibits grey-level constancy in the luminance or chrominance will consist of relatively few non-zero or significant values. The frequency domain transformation chosen by the JPEG members is the discrete cosine transform (DCT). This was chosen over the more traditional Fourier transform since it produces real-valued rather than imaginary-valued transform coefficients that are more easily stored in a compact fashion in memory. The DCT coefficients are computed for a one-dimensional function as follows.

$$F(u) = c(u)\sum_{x=0}^{M-1} f(x)\cos\frac{(2x+1)u\pi}{2M} \quad \text{for} \quad u = 0,1,2,\ldots,M-1$$

$$c(u) = \begin{cases} \sqrt{\frac{1}{M}} & \text{where } u = 0 \\ \sqrt{\frac{2}{M}} & \text{otherwise} \end{cases}$$

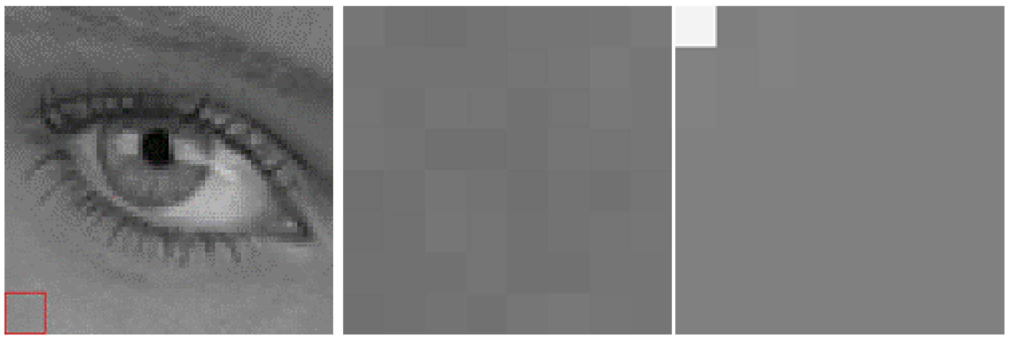
where M is the number of points in the function f(x). The DCT is performed on two-dimensional data sets as a series of consecutive one-dimensional transformations on the rows and subsequently the columns of the two dimensional array. The inverse transformation to take the frequency domain data back to the original spatial image data space is given by

$$f(x) = \sum_{u=0}^{M-1} c(u)F(u)\cos\frac{(2u+1)x\pi}{2M} \quad \text{for} \quad x = 0,1,2,\ldots,M-1$$

$$c(u) = \begin{cases} \sqrt{\frac{1}{M}} & \text{where } u = 0 \\ \sqrt{\frac{2}{M}} & \text{otherwise} \end{cases}$$

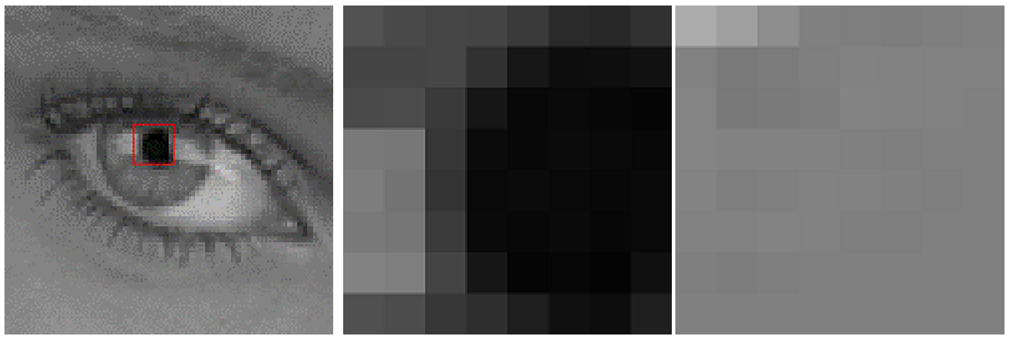
So for each 8x8 block of pixels in the original luminance and chrominance images, the DCT is computed. For the majority of blocks in the image, only some small number of the 64 pixels in the 8x8 block will have DCT coefficients that are significant in magnitude. The following figure illustrates the results of a DCT transformation on two blocks of varying degrees of grey-level constancy.

The DCT coefficients are computed for each 8x8 block of pixels in the image. To this point, the entire JPEG process is completely reversible except for the losses due to subsampling of the two chrominance channels.



925.5	-11.1	4.4	-1.8	-1.0	1.9	1.4	-0.3
4.3	-1.3	4.4	2.5	2.6	1.0	-0.7	0.6
4.0	-3.6	-0.7	3.5	1.7	-0.2	-1.6	1.0
-2.1	0.3	-0.1	-3.5	1.7	0.6	0.6	-0.6
-1.2	-0.6	0.2	3.3	1.3	-3.6	1.3	-0.8
-0.9	1.2	3.8	2.0	3.5	0.7	-0.9	-0.6
1.9	-1.0	-0.9	-1.3	0.1	0.9	1.4	1.2
-0.9	1.0	-0.8	-3.0	0.3	1.3	0.0	-0.4

(a)



350.5	251.0	109.0	0.8	-17.7	-26.1	-6.7	-3.7
7.0	-47.2	-43.6	-11.3	4.5	15.1	5.1	5.1
30.8	-58.4	-52.0	-26.3	25.0	14.2	6.4	-1.7
35.7	14.0	17.1	12.4	0.3	-6.1	-3.0	1.4
26.0	-19.7	-3.5	10.7	13.7	-3.1	-5.1	-1.1
20.0	18.6	20.0	7.5	-5.7	-6.5	-2.5	-3.1
-6.0	-23.6	-12.4	2.4	1.6	0.5	1.7	2.2
-3.0	-1.8	0.3	0.0	1.3	1.2	1.9	1.2

(b)

Figure 8. DCT transform - The discrete cosine transform coefficients represent the power of each frequency present in the sub-image blocks shown in the images to the left. The images in the center are a magnified version of the sub-image block shown. The images to the right represent a scaled visualization of the discrete cosine transform coefficients shown in the tables below each set of images. The data shown in (a) represent a smooth area in the original image while those shown in (b) represent a higher frequency region.

The next processing step in the chain of computations that make up JPEG image compression is the quantization of the DCT coefficients in each of the 8x8 blocks. It is at this step that the process is able to achieve the most compression; however, it is at the expense of image quality. The entire process becomes what is referred to in the image compression community as "lossy". The process is still reversible, however, it can no longer exactly reproduce the original image data.

As we have already seen, the DCT coefficients get smaller in magnitude as one moves away from the lowest frequency component (always located in the upper left hand corner of the 8x8 block). Quantization of the DCT coefficients scales each of the DCT coefficients by a prescribed, and unique factor, whose strength relies on the quality factor specified by the user. The JPEG committee prescribes for the luminance channel and for both chrominance channels the quantization factors. These scaling factors are used to divide, on a coefficient by coefficient manner, the DCT coefficients in each 8x8 block. Each element of the scaled coefficient values is then rounded off and converted to an integer value. The scaling factors are given in the following illustration.

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

(a)

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

(b)

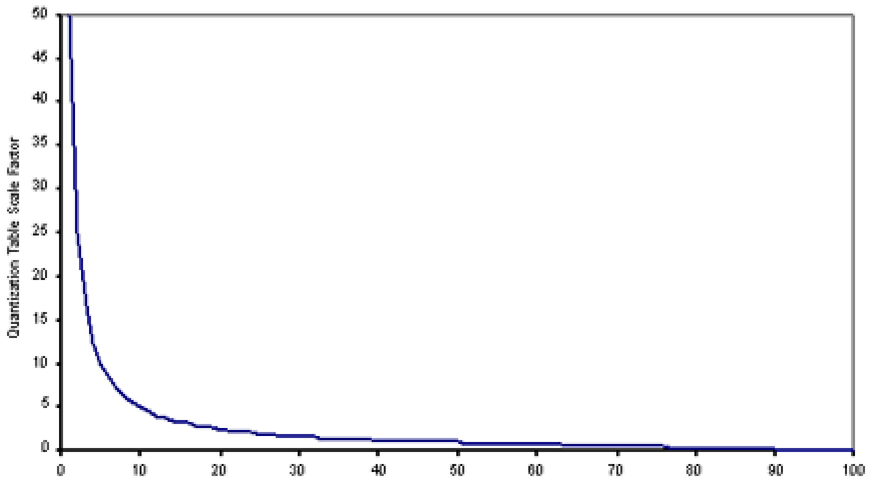
Figure 9. DCT coefficient quantization factors - The discrete cosine transform coefficients quantization factors are given (a) for use with the luminance channel and (b) for use with both the chrominance-blue and chrominance-red channels.

The quantized DCT coefficients are computed by applying the quantization factors, represented as Q, to the DCT coefficients as

$$DCT_{quantized} = ROUND\left(\frac{DCT_{coefficients}}{Q \cdot Scale_{Factor}}\right)$$

The factor, $Scale_{Factor}$, given in this equation is known as the scaling factor and is derived from a quality factor specified by the user. The quality factor is specified between 0 and 100 where 100 represents the best image quality (the least quantization). The relationship between the user-specified quality factor and the scaling factor is given by

$$Scale_{Factor} = \begin{cases} \dfrac{50}{Quality_{Factor}} & \text{for} \quad Quality_{Factor} < 50 \\ 2 - \dfrac{Quality_{Factor}}{50} & \text{for} \quad Quality_{Factor} \geq 50 \end{cases}$$



and is illustrated in the following plot

Figure 11. DCT quantization table scale factors - The DCT quantization scale factors are given as a function of the user specified quality factor.

Once the DCT coefficients have been scaled, quantized, and converted to integer values, the data is ready for coding and storage. As stated in the beginning of this essay, Shannon showed that it is most efficient to store a message by using the shortest codewords for the most frequently occurring symbols and longer codewords for less probable symbols. At this stage, we have conditioned the DCT coefficients in such a way that they are ready for coding redundancy reduction.

The final step in the JPEG process is to use Huffman coding to represent the conditioned DCT coefficients in as efficient manner as possible. As it is outside of the scope of this essay to give a complete description of Huffman coding, the reader is referred to the many web sites and textbooks that describe this topic in great detail.

Now some of the caveats of this widely used image compression technique must be mentioned.

1. This technique is best applied to photographs of natural scenes.
2. This technique works best when the assumption of grey-level constancy is valid in 8x8 image blocks. The technique will prove less effective providing less compression when this assumption is violated. Images with a lot of noise, for example those taking at high ISO film speed settings in low-light level situations with a digital camera, will not compress as well as those that receive plenty of exposure and contain less noise.
3. The use of JPEG for the storage of line art is not recommended. There are a lot of areas in images of line art that are constant grey level, and this is good, but the main content of interest in this type of image is the black lines on a white background, which the user expects to be crisp and sharp. The use of JPEG will result in unacceptable artifacts in blocks where there is high contrast.

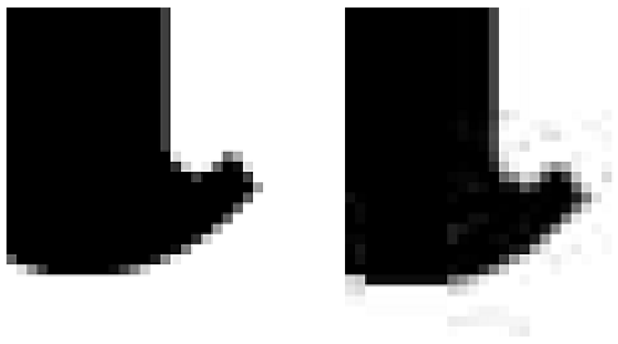


Figure 12. JPEG effects on line art - The image on the left is an original digital image of the serif on a lowercase "a". The image on the right shows the artifacts that are seen when this image is stored as a JPEG file with a user-specified quality factor of 20.

4. Never use JPEG as an intermediate storage format. JPEG should only be used to store the final image that results from your processing steps. If you take a picture, remove the red-eye, sharpen the edges, and then want to display it on the web; it is only for the final storage of the processed image for publishing that JPEG should be used.

5. One final note. As convenient as it is to store hundreds of images on the storage media in your digital camera, you might want to reconsider using JPEG as the default storage format (see the figure below). If you are going to be processing the images after you download them from your camera, you may want to consider a lossless format such as TIFF or RAW for use in your camera.

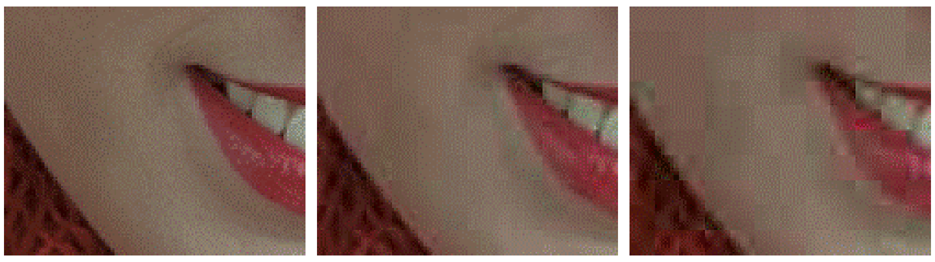


Figure 13. Effects of JPEG - The effects of using the JPEG file format are shown. The image shown in (a) is the original image. Image (b) shows the results of JPEG compression using a "medium" quality setting on a digital camera. Image (c) shows the results of JPEG compression using a "low" quality setting on a digital camera. While higher quality setting will not result in such objectionable images, the photographer should always be cognizant that these 8x8 blocking artifacts will always be present in any image saved as a JPEG file from a digital camera or saved as a processed product from an image processing program.

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.

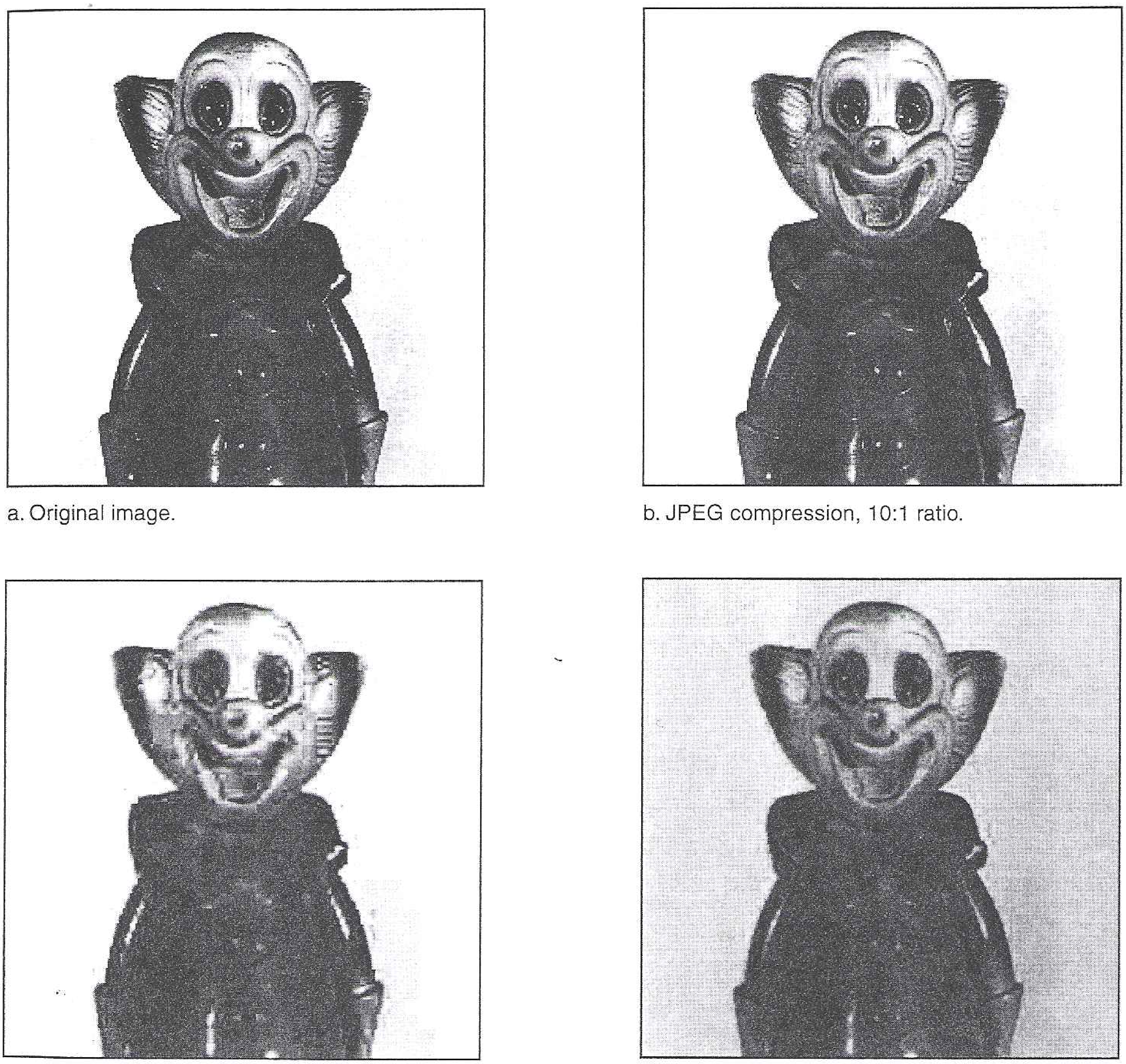
Arithmetic encoding has several drawbacks:

- 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 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.

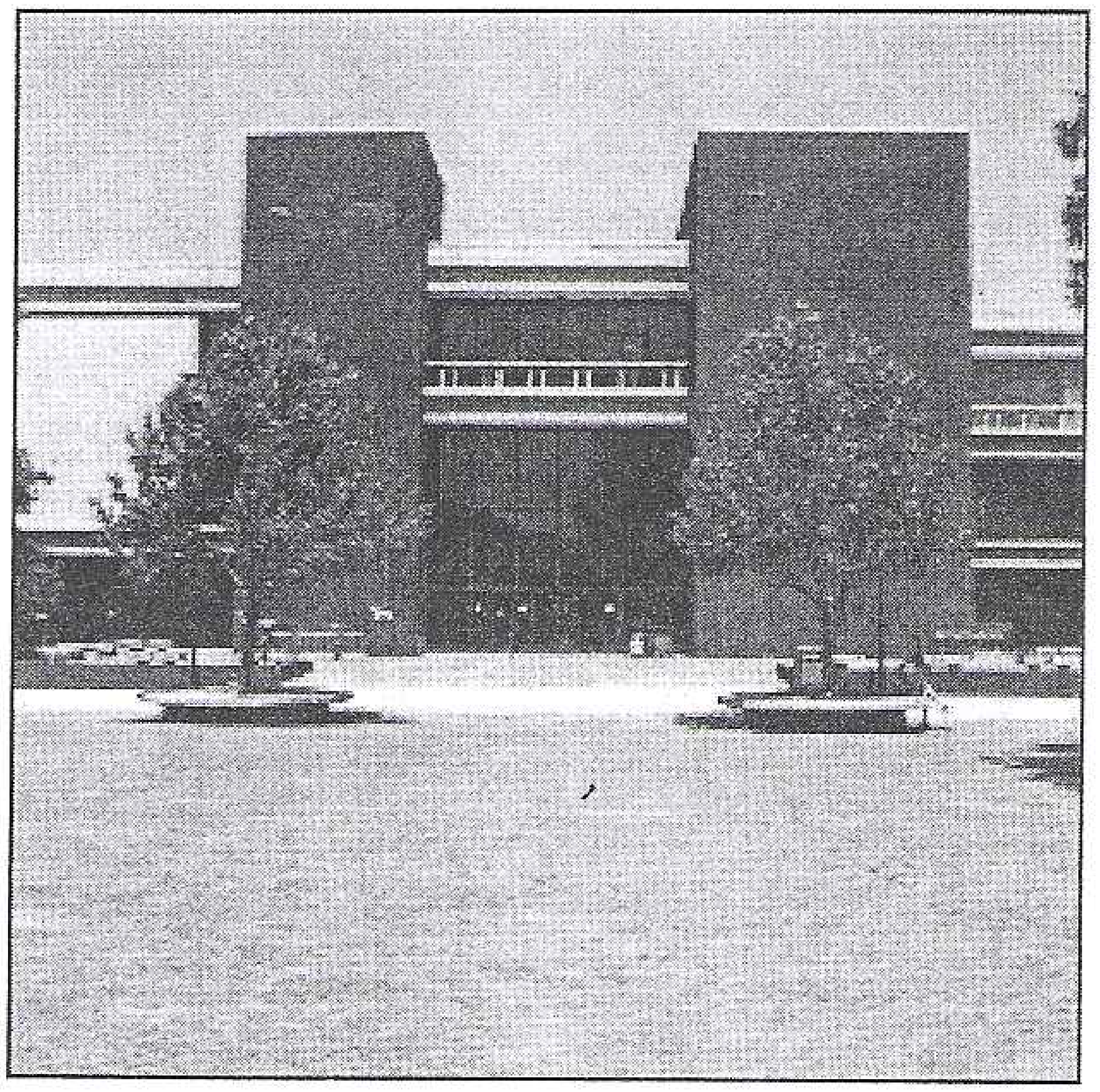
5.3.1 Gray-Level Run-Length Coding

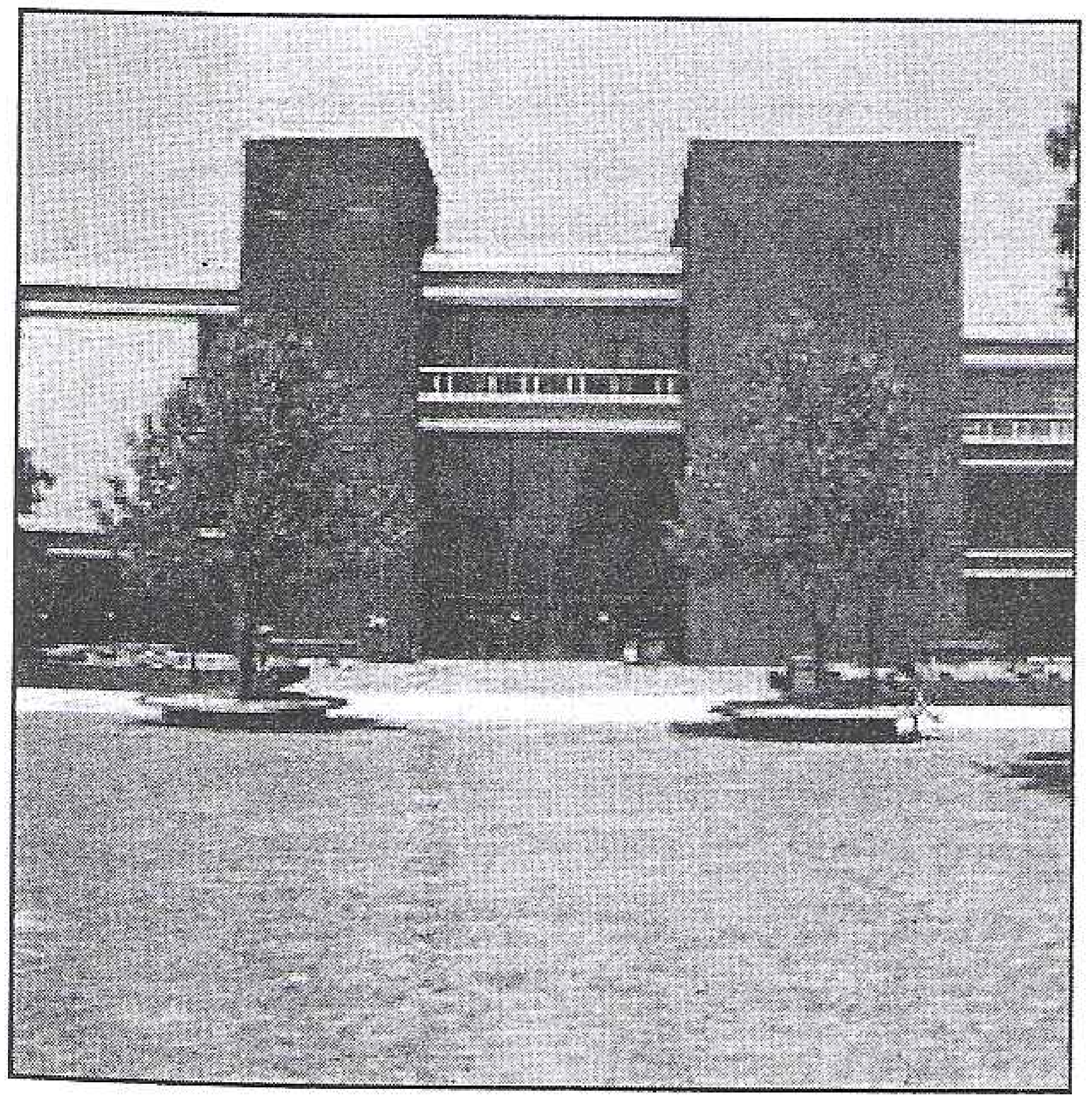
In Section 5.2.2 on lossless compression we discussed methods of extending basic run-length coding to gray-level images, by using bit-plane coding. The RLC technique

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.

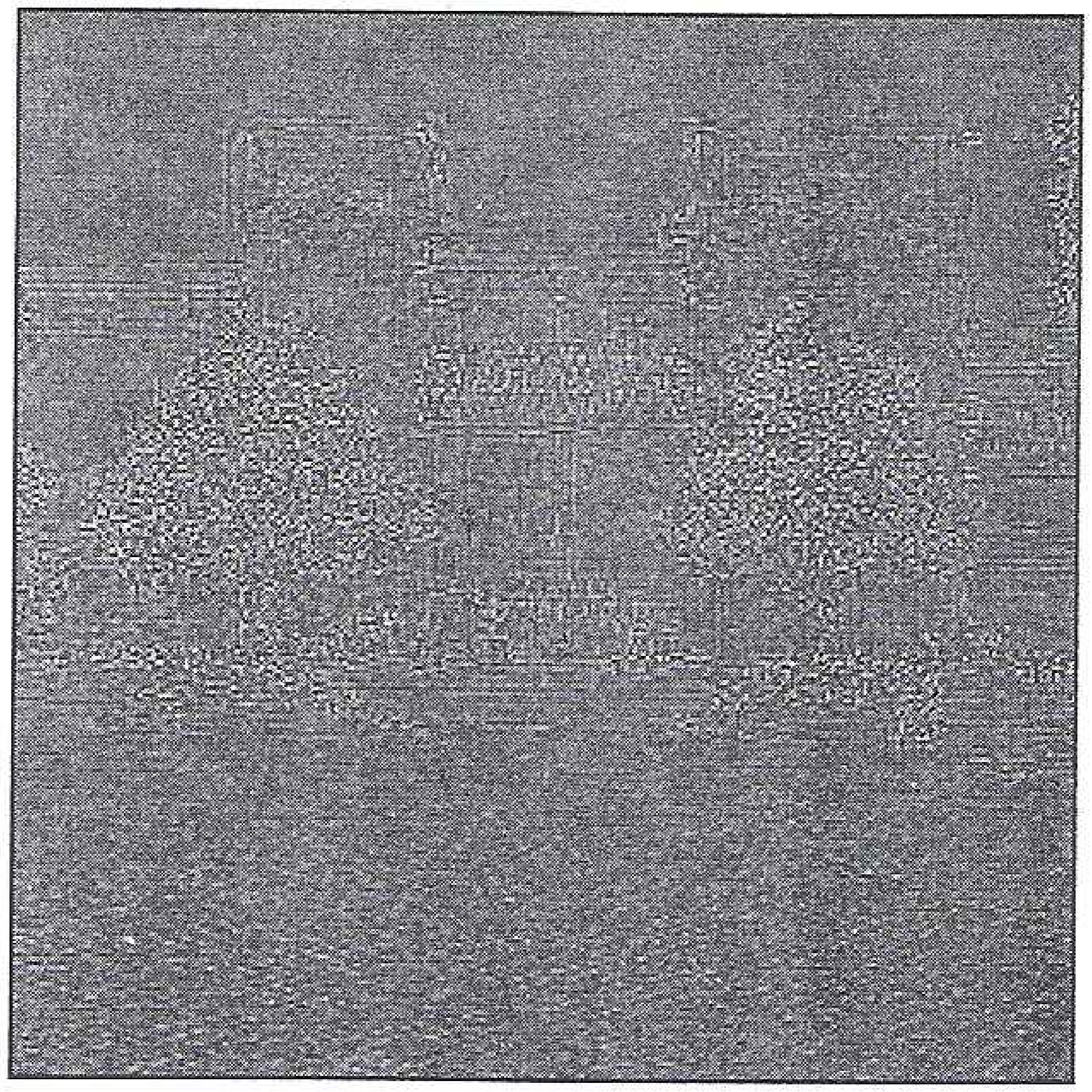
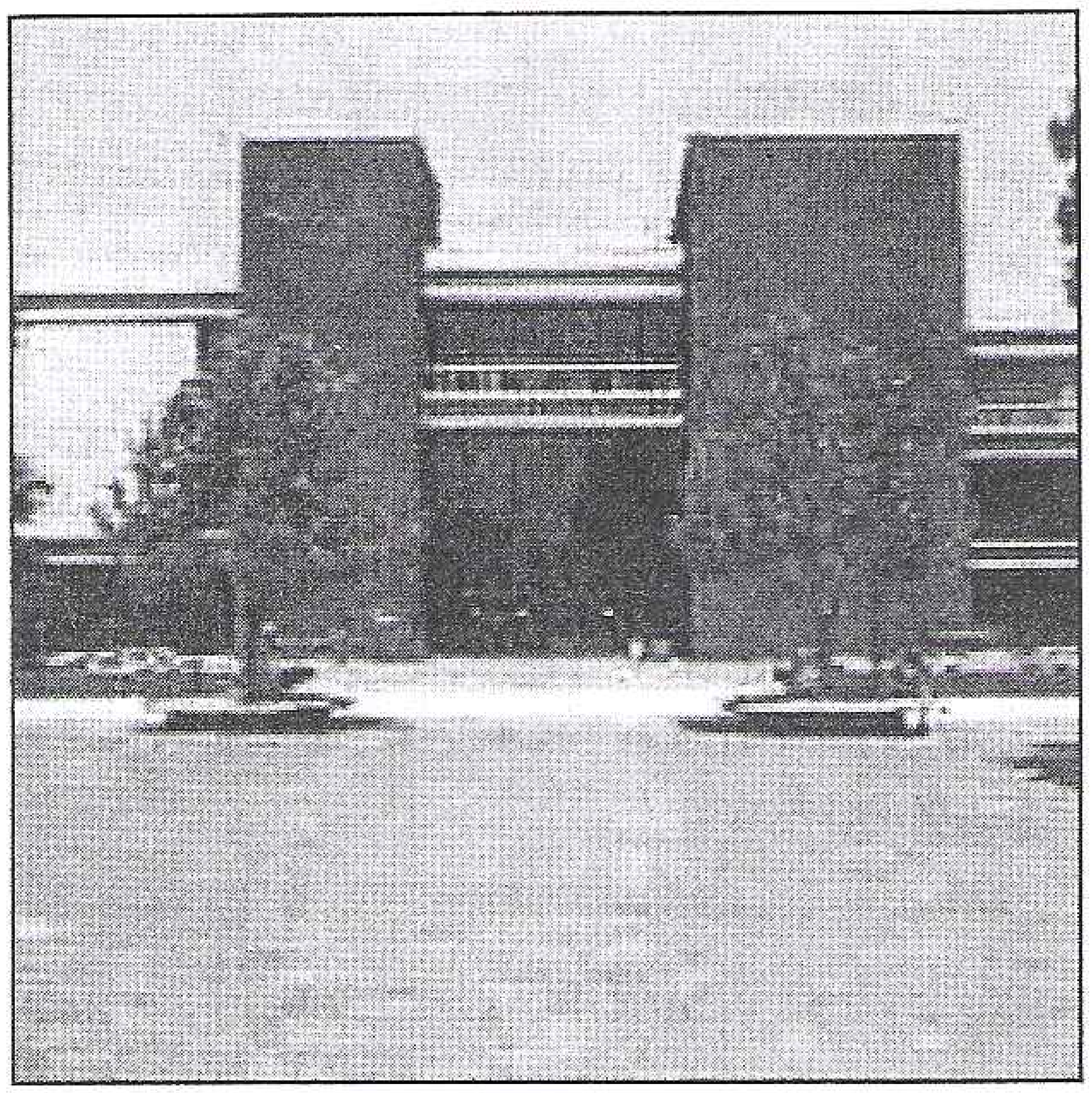
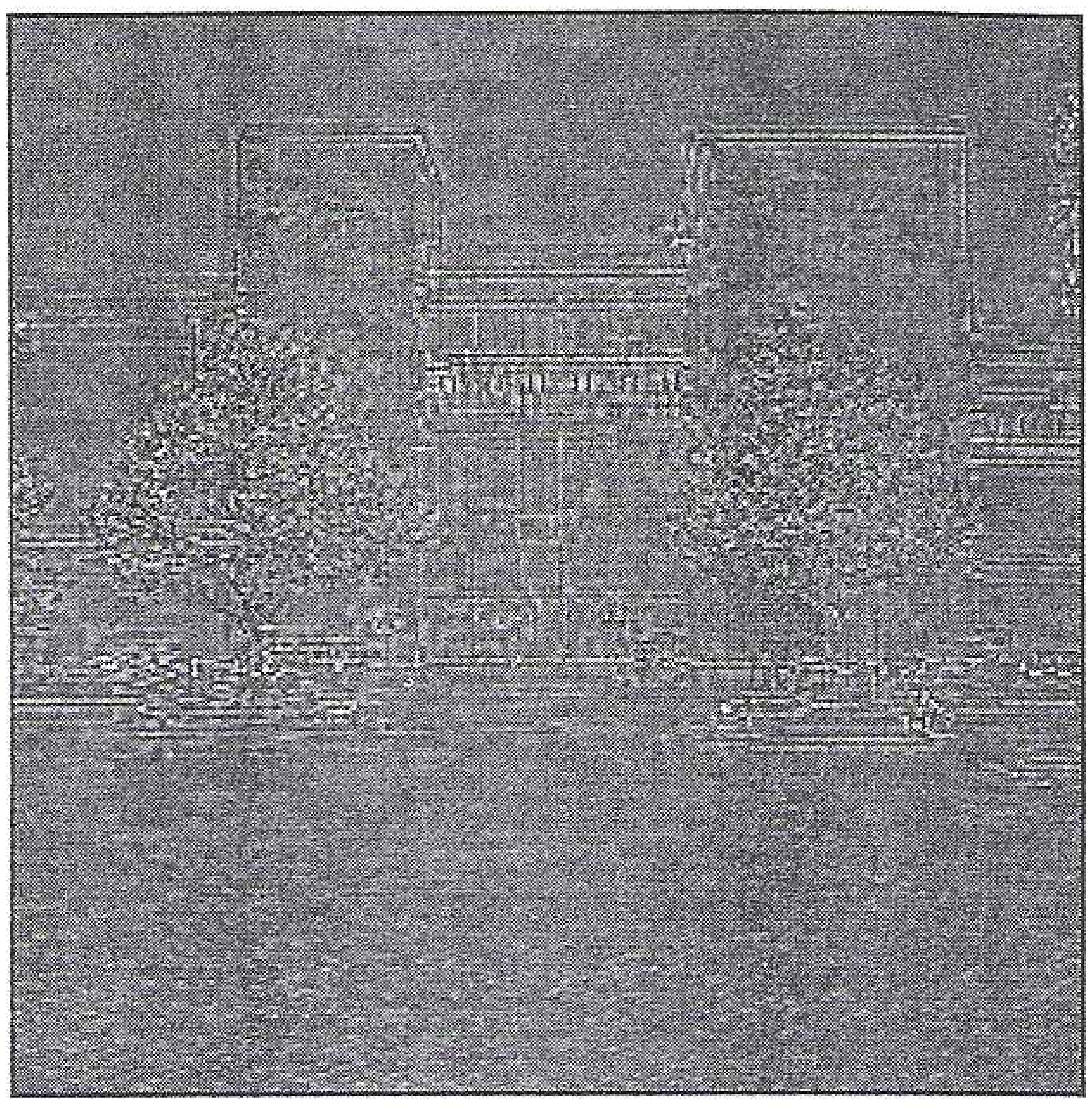
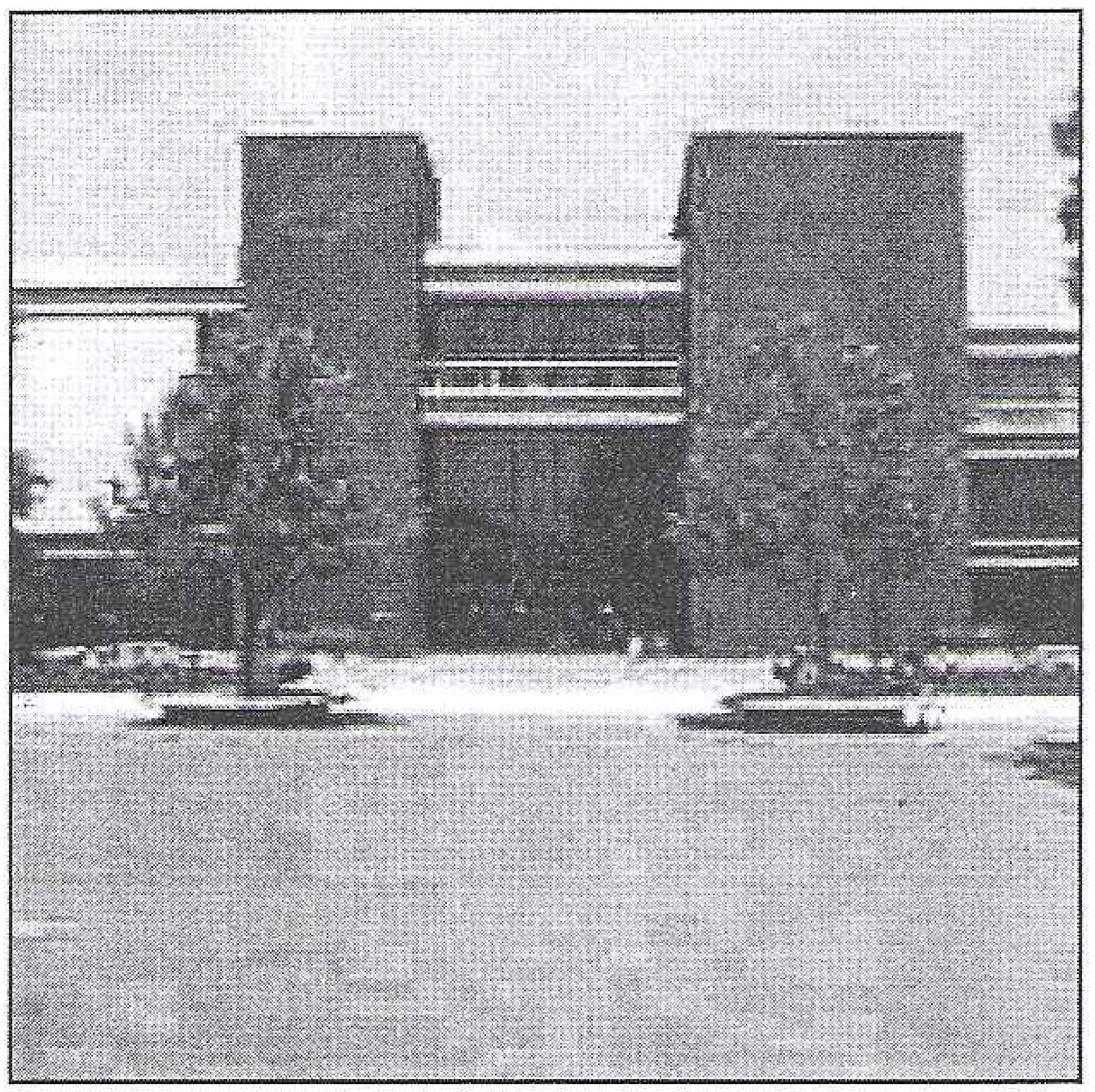
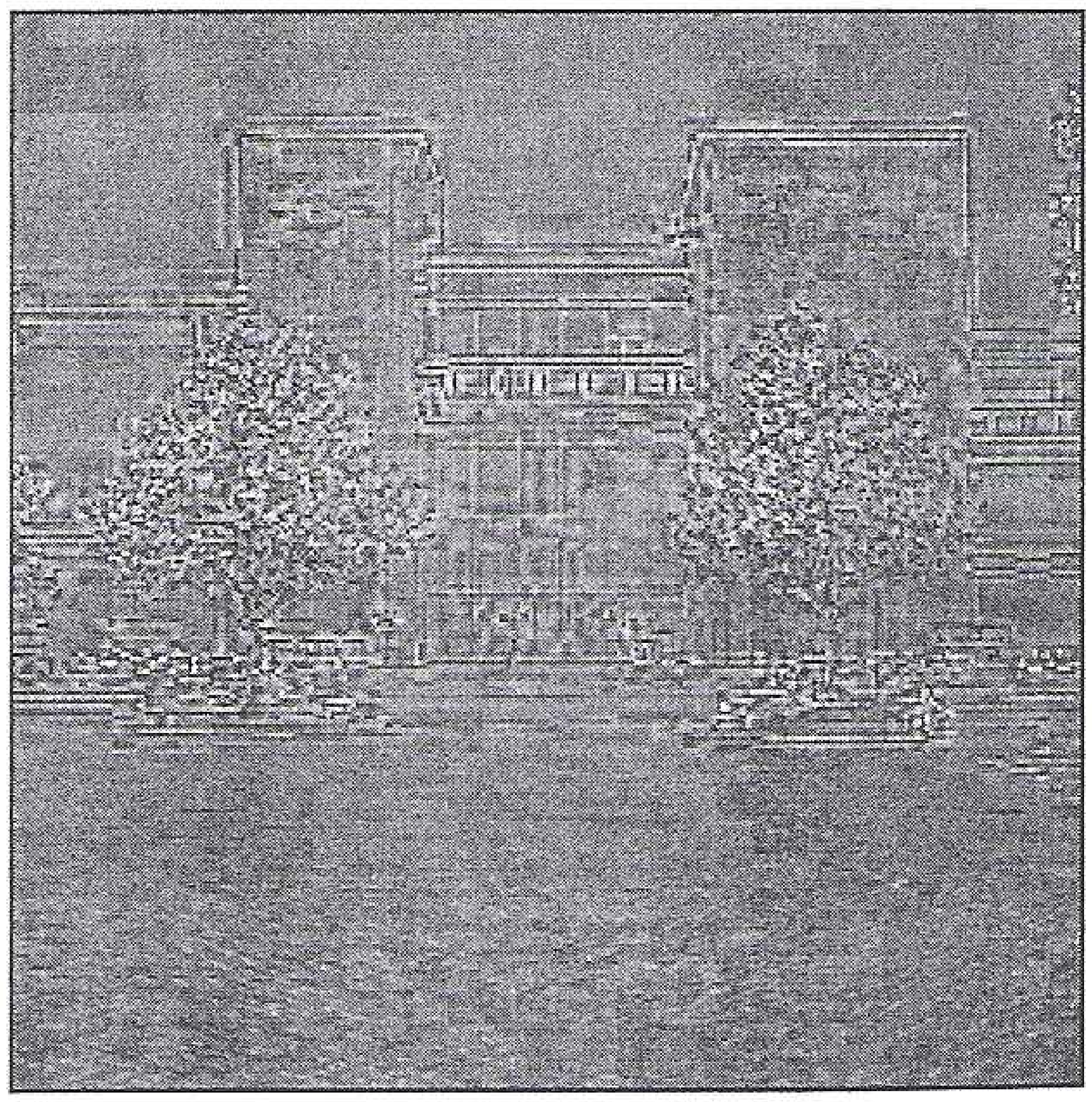

c. Error image for (b), multiplied by 8 to show detail.

Figure 5.3-14 (Continued)


d. JPEG compression = 20:1.


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

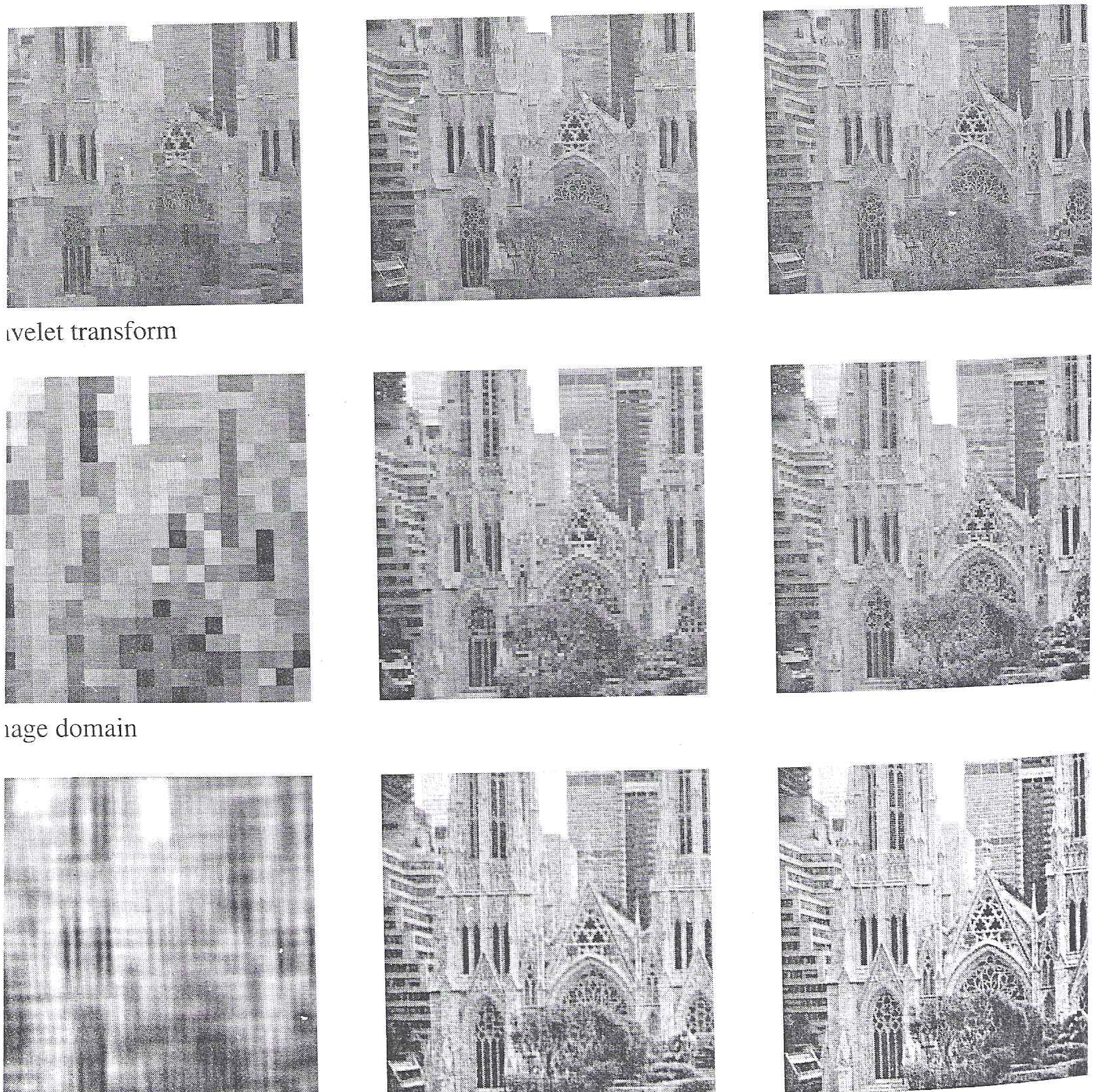

f. JPEG compression = 30:1.


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.

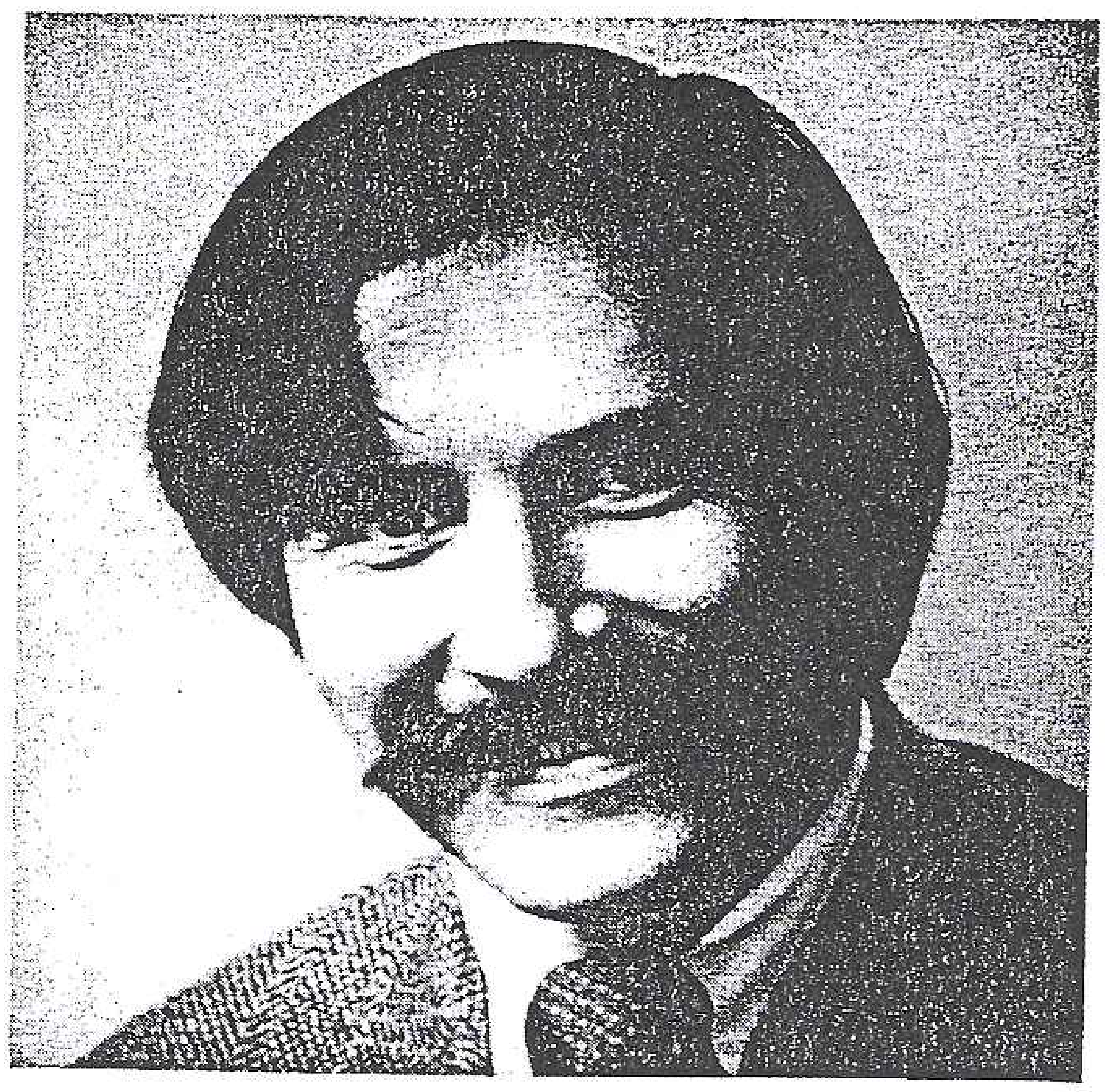
images, superior for progressive transmission. A simple comparison is shown i Figure 26.5 which shows three progressive sequences, a wavelet series, a sequenc generated by resampling the image at progressively higher resolution and a JPE series. The series is for a single, three and six updates containing 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
rogression and
s.

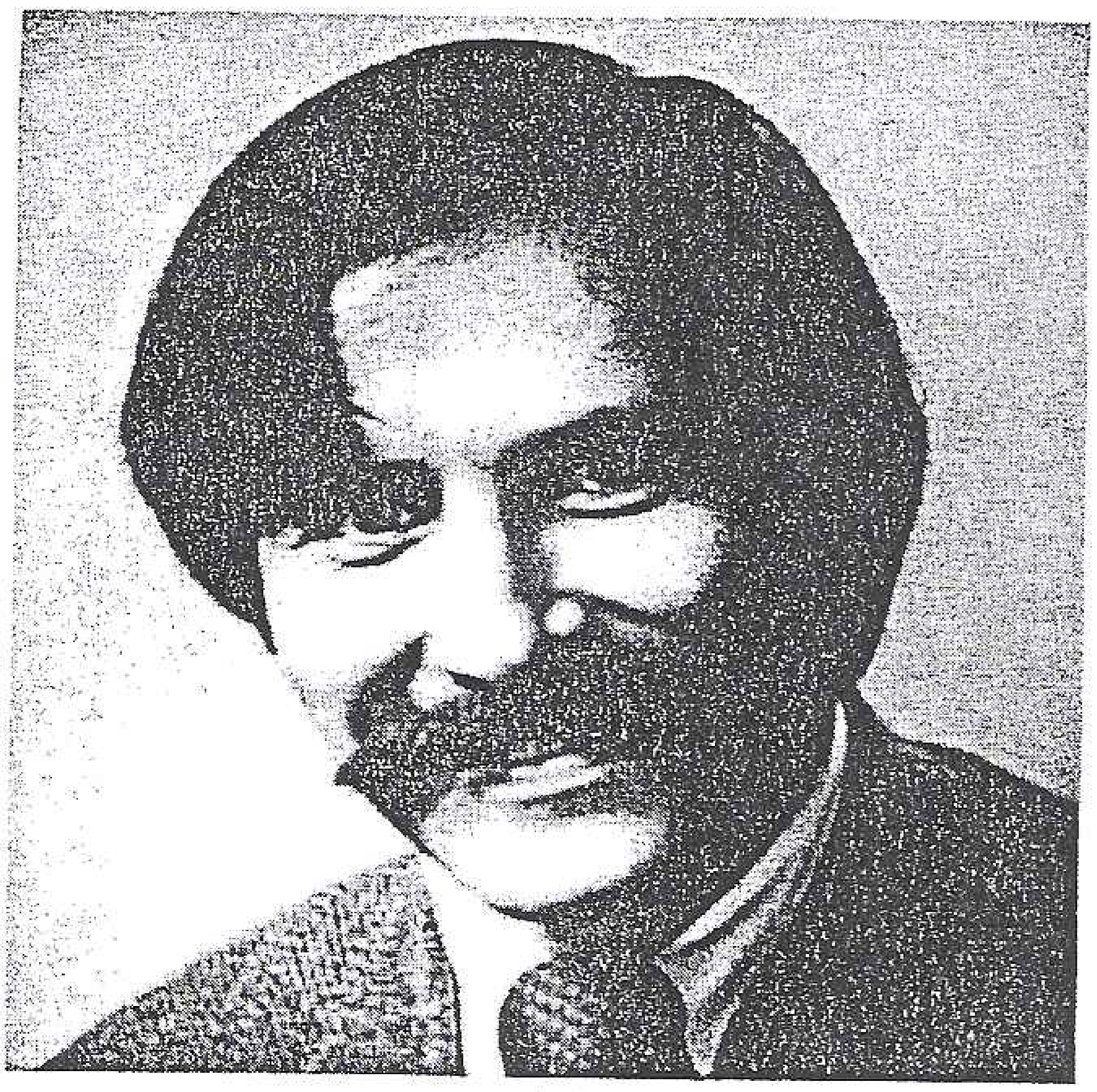


Degradations due to transform coding. Quantization noise in the reconstructed image manifests itself in transform image coding differently from waveform image coding. In general, the effect of quantization noise is less localized in transform image coding. Quantization of one transform coefficient affects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

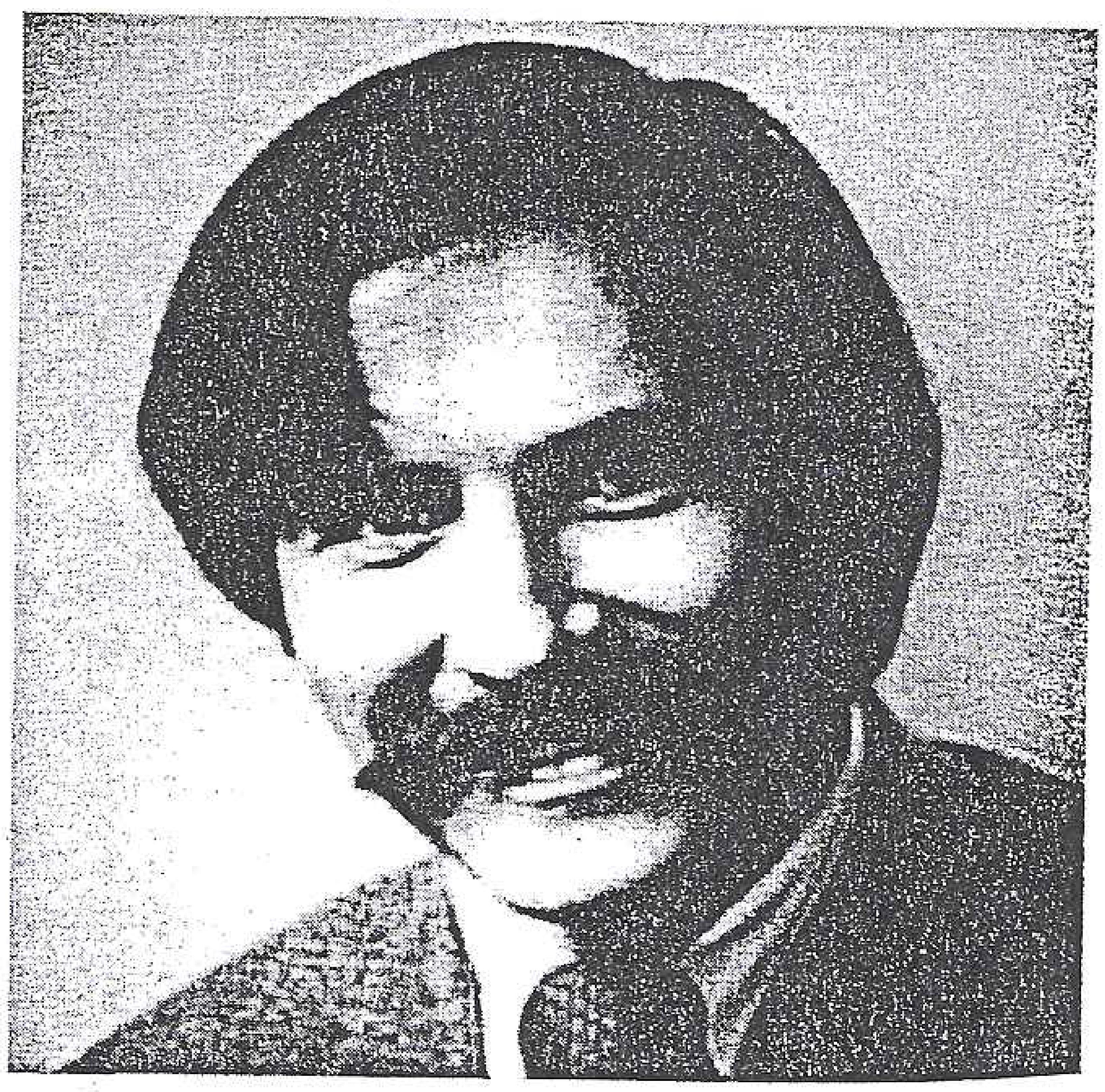
Several types of image degradation result from quantization noise in transform image coding. One type is loss of spatial resolution. In the DCT coding of images, the discarded transform coefficients are typically high-frequency components. The result is loss of detail in the image. An example of this 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.

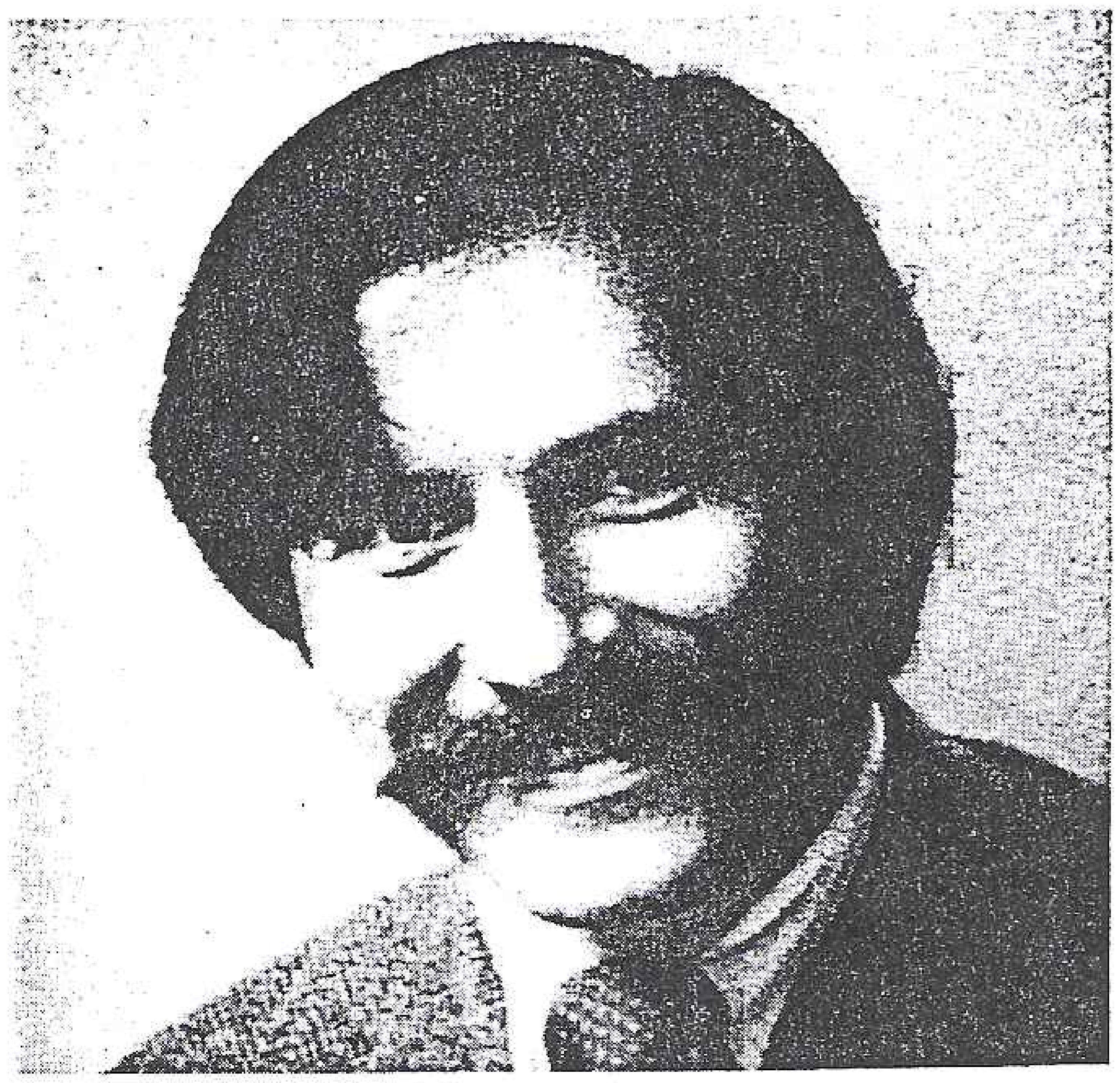


Figure 10.46 Illustration of graininess increase due to quantization of DCT coefficients. A 2-bit/pixel uniform quantizer was used to quantize each DCT coefficient retained to reconstruct the image in Figure 10.45(b).

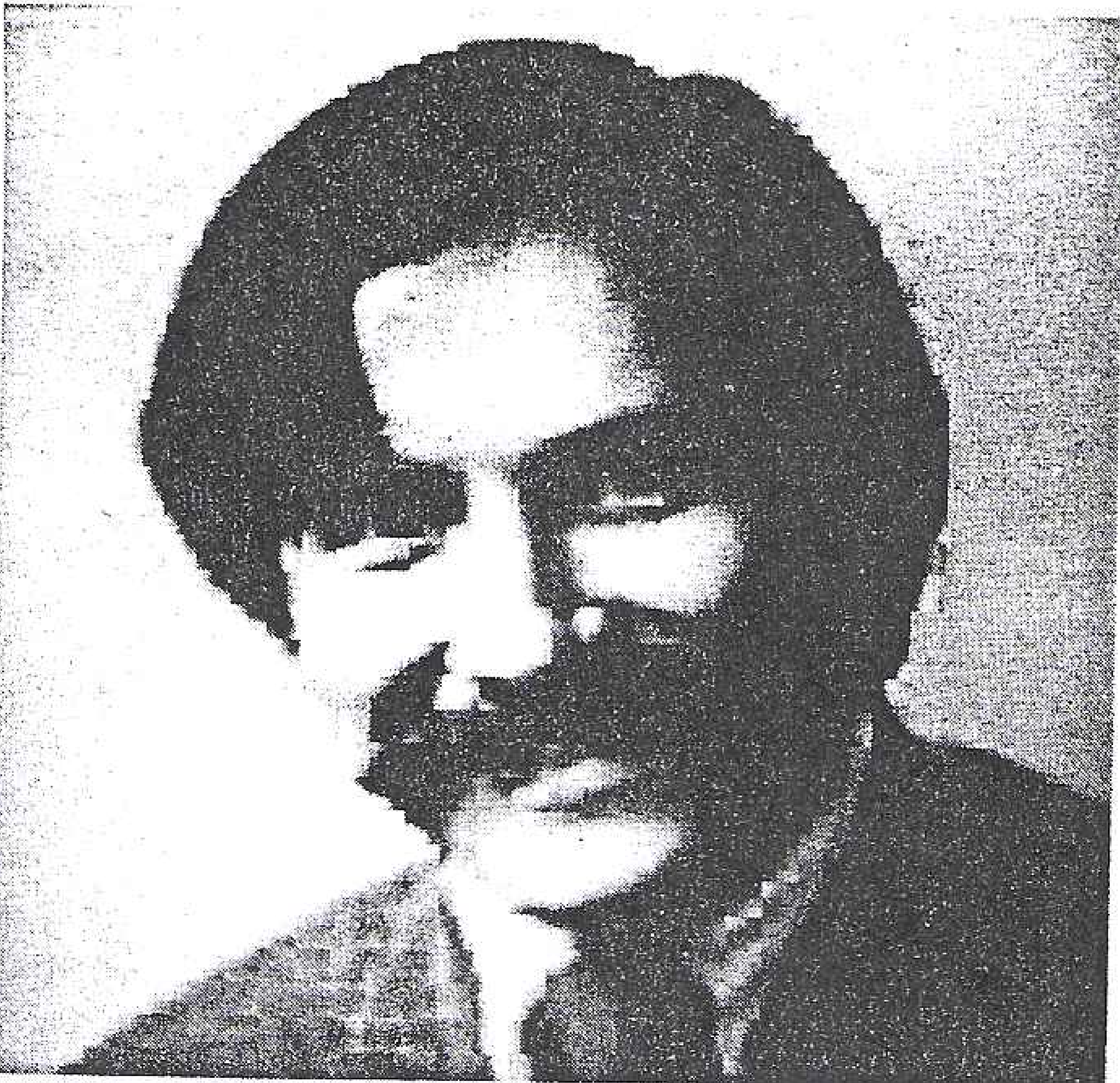
and are selected from a zone of triangular shape shown in Figure 10.43(a). From Figure 10.45, it is clear that the reconstructed image appears more blurry as we retain a smaller number of coefficients. It is also clear that an image reconstructed from only a small fraction of the transform coefficients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

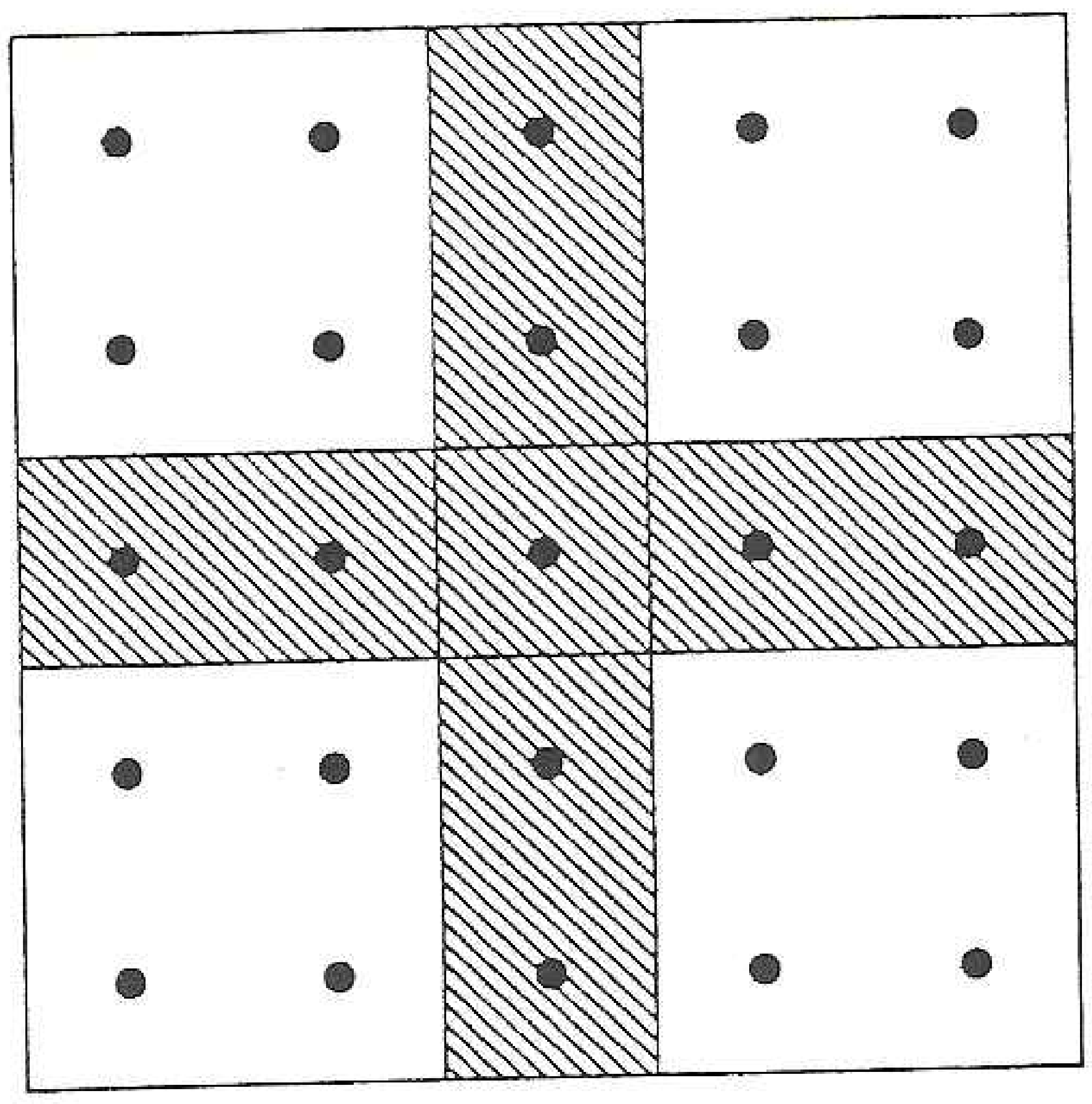


Figure 10.49 Example of one-pixel overlap in the overlap method for reducing blocking effect.

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

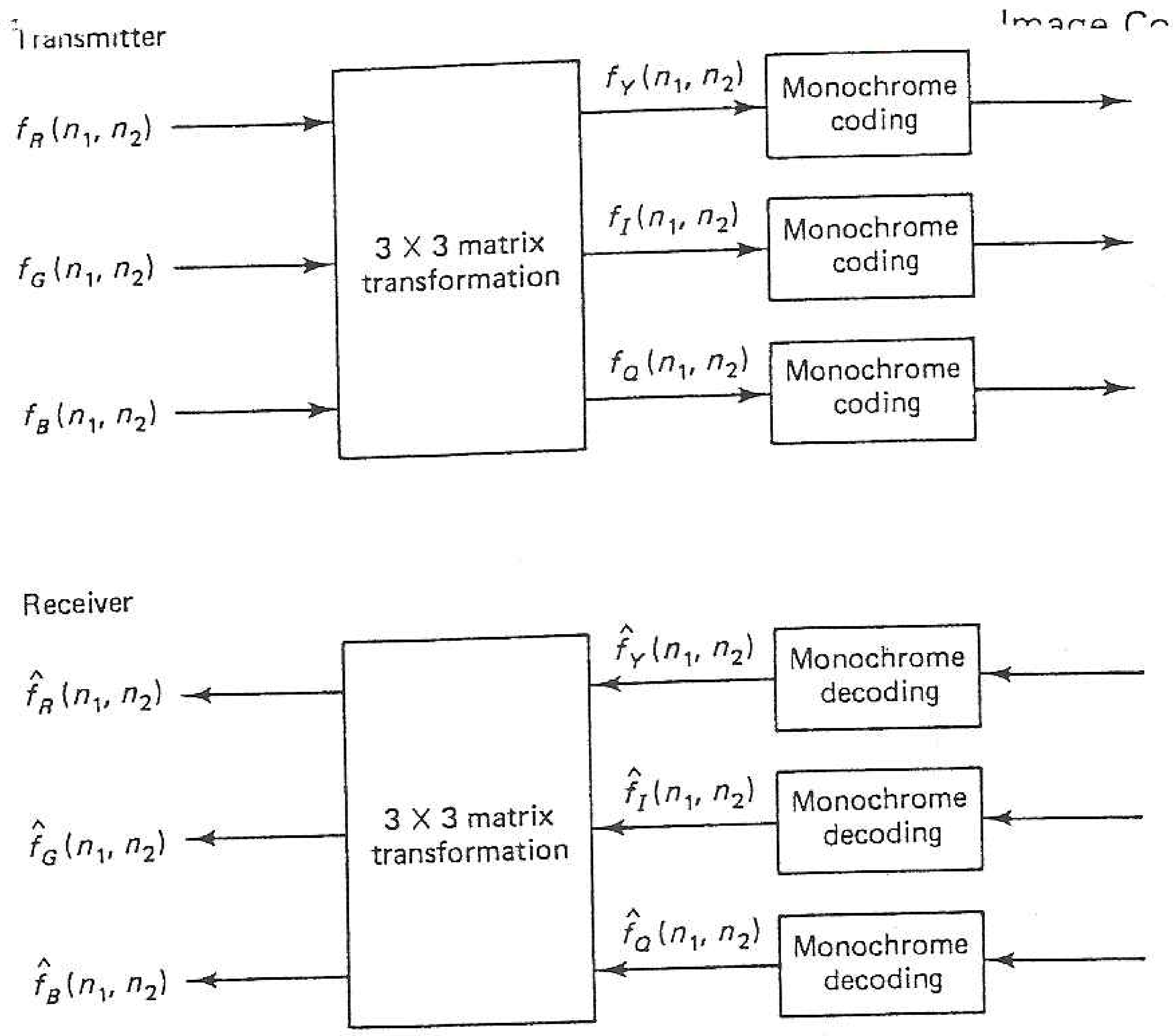


Figure 10.57. Color image coding in the *YIQ* space.

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 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