DIGITAL COMPRESSION
for
MULTIMEDIA

Principles and Standards

Jerry D. Gibson	*Southern Methodist University*
Toby Berger	*Cornell University*
Tom Lookabaugh	*DiviCom*
Dave Lindbergh	*PictureTel Corporation*
Richard L. Baker	*PictureTel Corporation*

9 JPEG Still-Image Compression Standard

CHAPTER

9.1 Introduction

Although several standards for voice and facsimile compression have existed for a number of years, the JPEG still-image compression standard has to be one of the most widely recognized standards in existence today. The baseline JPEG compression method has truly become ubiquitous, with a wide variety of hardware and software implementations available for a host of applications. The broad goal of the Joint Photographic Experts Group (JPEG), meeting as a working group under the ISO but closely coordinated with CCITT SGVIII, was to develop a general-purpose still-image compression standard that would be applicable to virtually all continuous-tone still-image transmission and storage problems. To achieve this, JPEG draws upon years of prior research in image compression and complements this with numerous innovations and refinements that bring both performance and complexity to levels that allow it to have the wide utility that it has today.

The goals of JPEG are the following (Wallace 1991; Pennebaker and Mitchell 1993):

1. Achieve rate and reconstructed image quality "at or near the state of the art" with image fidelity classifiable as "very good" to "excellent"

2. Be useful for compressing almost any continuous-tone still image, including both gray-scale and color, any color space, and most image sizes

3. Have complexity that would allow software implementations on many common computing platforms and affordable hardware implementations
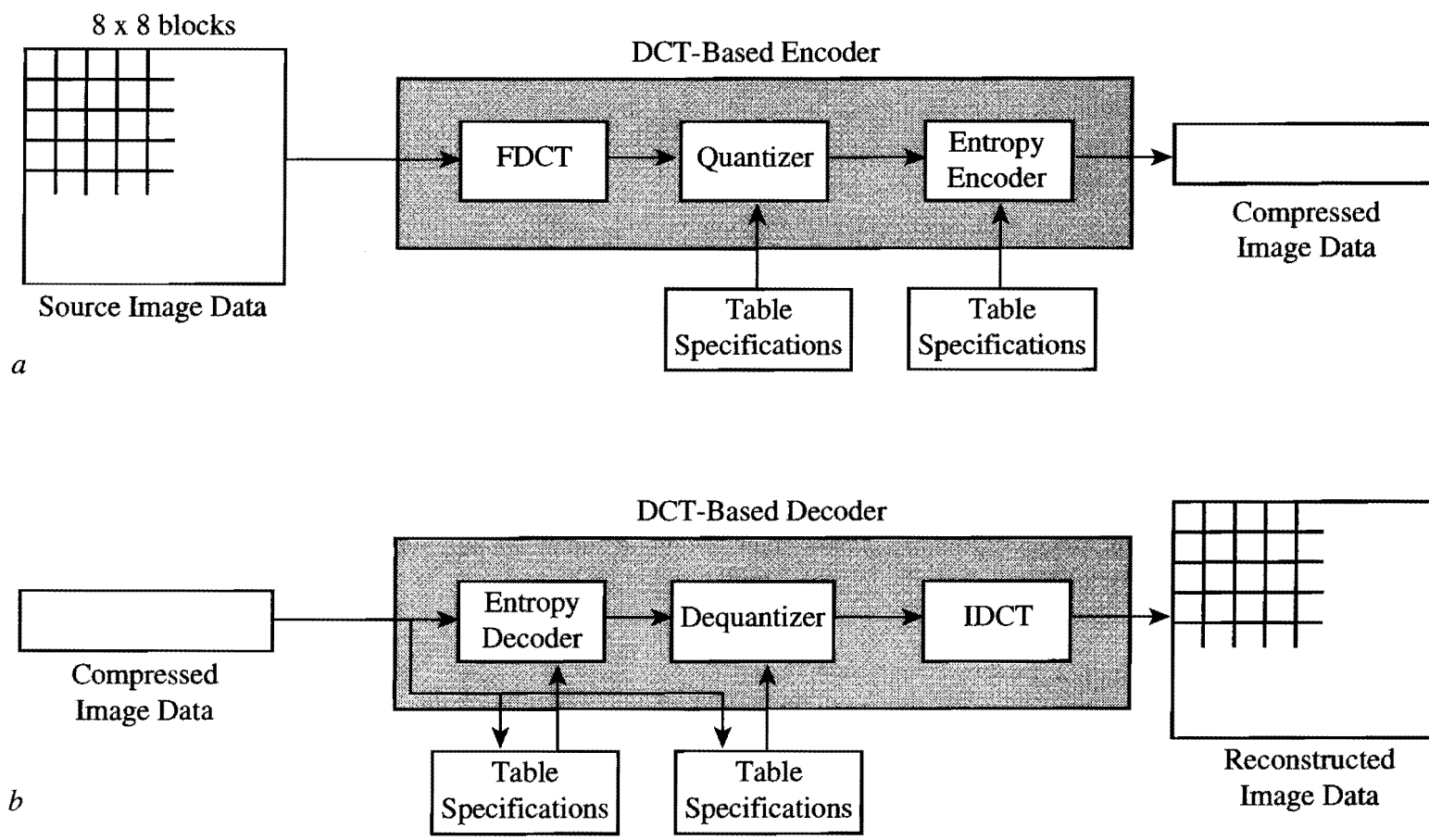
The JPEG standard has several lossy encoding modes, starting with the baseline sequential mode, and a lossless encoding mode. In the following sections, we provide an overview of the JPEG compression methods, a discussion of relative performance, and insights on what led to various parameter selections and approaches.

9.2   Baseline JPEG

The basic JPEG encoder and decoder structures are illustrated in Figure 9.1; FDCT stands for forward DCT and IDCT stands for inverse DCT. The baseline JPEG encoding method is called *sequential encoding*. First the image is partitioned into 8 × 8 blocks of pixels that are ordered according to a "rasterlike" left-to-right, top-to-bottom scan, as depicted in Figure 9.2. The FDCT is computed on each of the 8 × 8 blocks of pixels, and the resulting 64 DCT coefficients are scalar quantized using uniform quantization tables based upon psychovisual experiments (Lohscheller 1984). These scalar quantization tables are provided as part of the standard but are not a requirement. After the DCT coefficients are quantized, the coefficients within the block are ordered according to the zigzag scan in Figure 9.3, the resulting bitstream is runlength coded to generate an intermediate symbol sequence, and then these symbols are Huffman coded for transmission or storage.

Figure 9.4 illustrates the transmission sequence for all blocks in an image. Each set of DCT coefficients for an 8 × 8 block is shown as a square slice, with the rows corresponding to the bits representing the coefficients, starting in the top row with the DC coefficient, followed in subsequent rows by the AC coefficients arranged following the zigzag ordering. The zigzag scan increases the runlengths, thereby increasing the efficiency of the lossless coding step. The DC coefficients from block to block are actually differenced before scalar quantization because the DC values of adjacent blocks are often very close to being the same.

Luminance and chrominance quantization tables drawn from the CCIR-601 experiments by Lohscheller (1984) are shown in Figure 9.5; the step size for the differenced DC coefficient is shown in the upper left, followed by the step size for the first horizontal harmonic to the right, and so on. Note that the step size is presented rather than the bit allocation, so that a large step size indicates a coarse quantization, or a low bit allocation, and a small step size represents a higher bit allocation. The perceptually designed quantization tables take advantage of
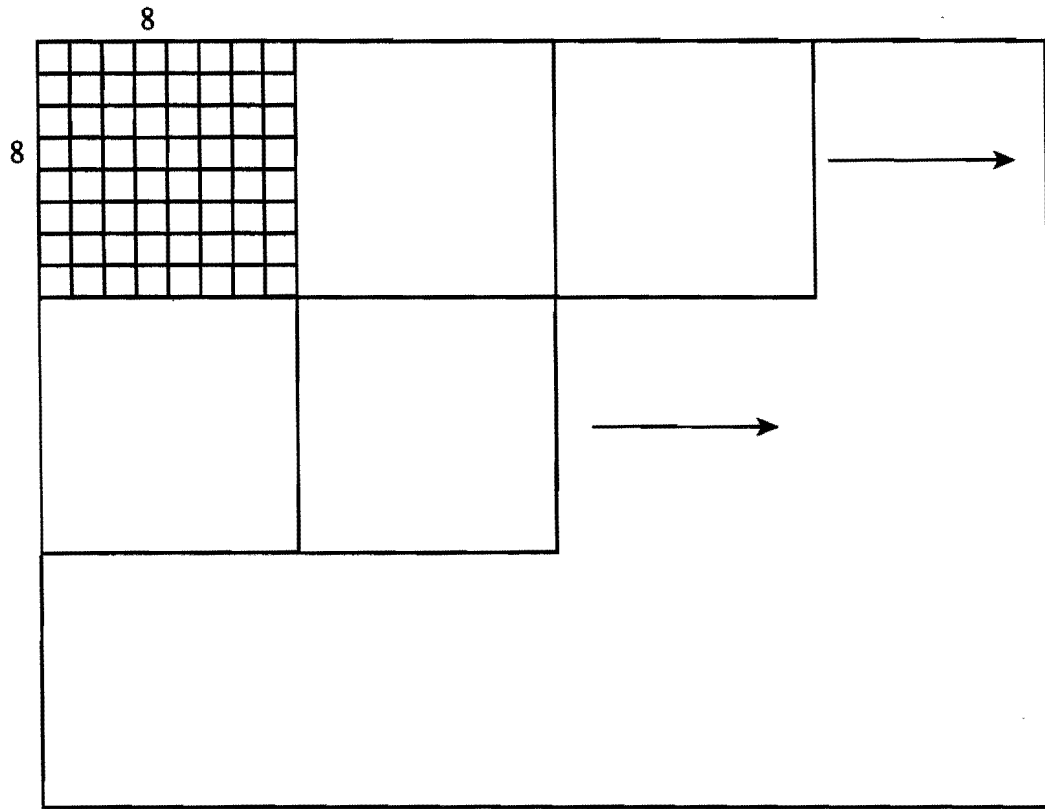
8 x 8 blocks

DCT-Based Encoder

FDCT → Quantizer → Entropy Encoder

Table Specifications

Table Specifications

Source Image Data

Compressed Image Data

a

DCT-Based Decoder

Entropy Decoder → Dequantizer → IDCT

Table Specifications

Table Specifications

Compressed Image Data

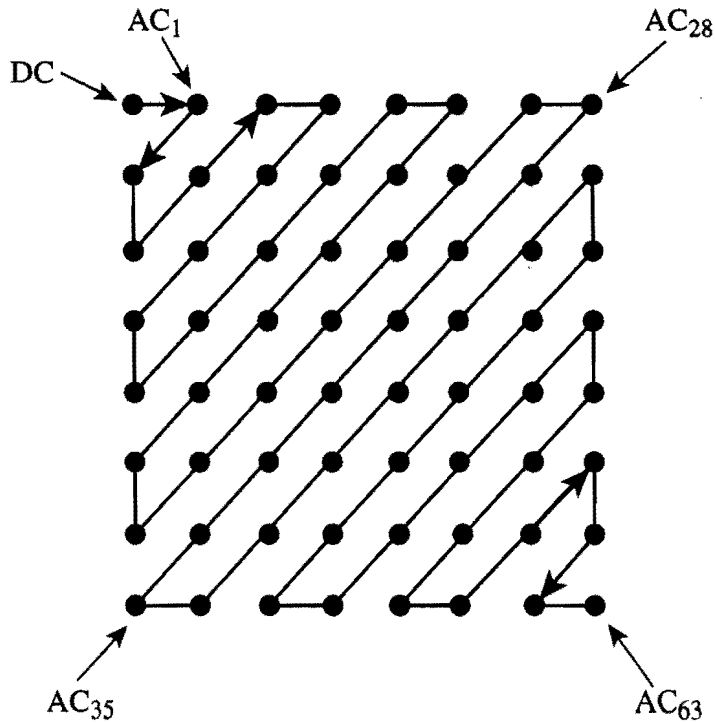Reconstructed Image Data

b

FIGURE

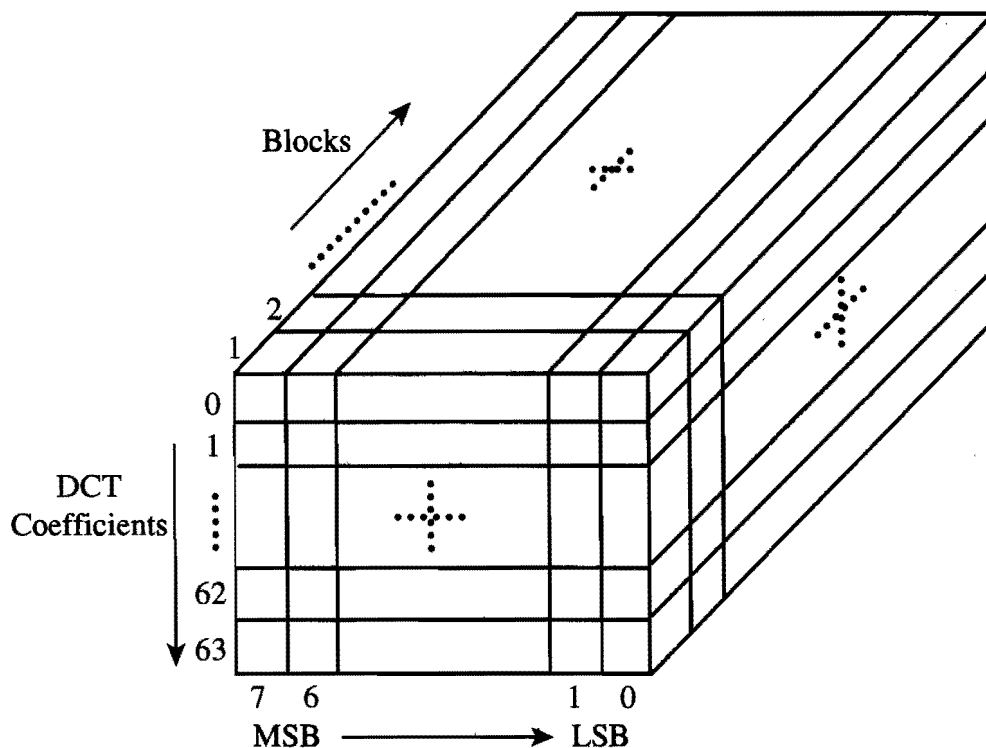9.1 JPEG: *(a)* Encoder; *(b)* Decoder

FIGURE

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

Zigzag Order

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.
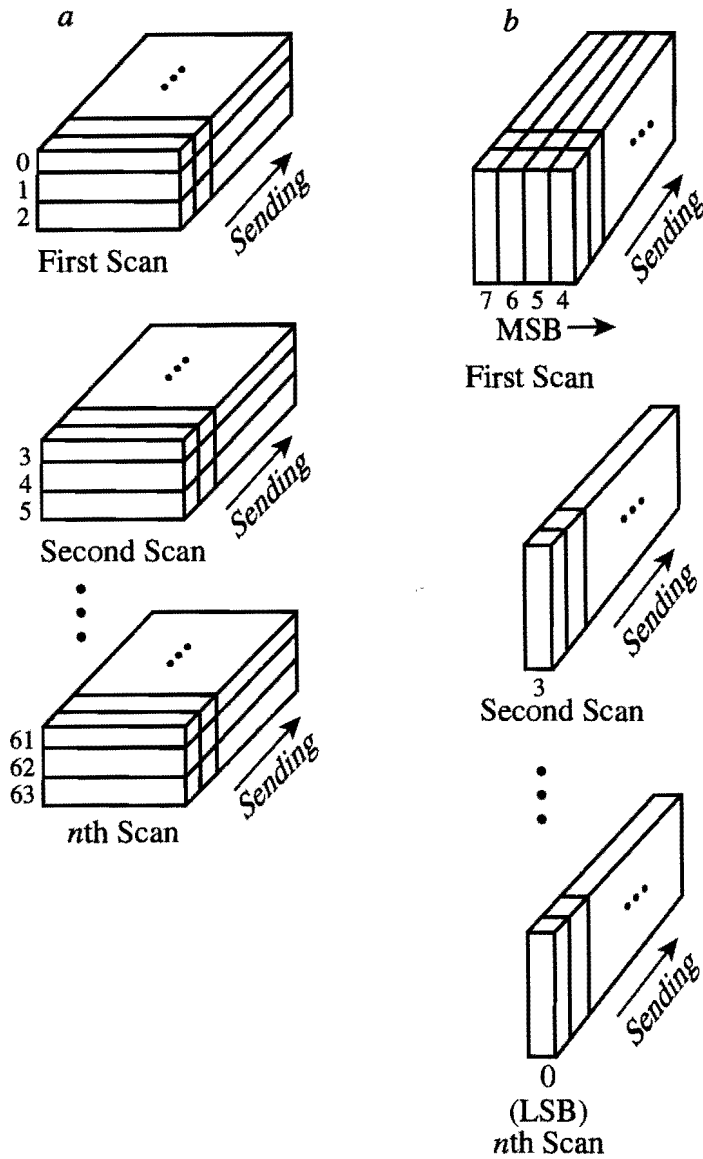
9.3   Progressive Encoding

Progressive encoding modes are provided to allow a coarse version of an image to be transmitted at a low rate and then progressively improved by subsequent transmissions. This is convenient for browsing applications so that the user does not have to wait for an entire image to be received before being able to view its contents. Implementation of progressive encoding requires the availability of a sufficient buffer to store the quantized DCT coefficients for an entire image. The entropy coding then transmits selected sets of these coefficients.

The two progressive encoding modes in the JPEG standard, spectral selection and successive approximation, are illustrated in Figure 9.6. Spectral selection involves sending sets of DCT coefficients starting with lower frequencies and progressing to higher frequencies. For example, the first scan might include the DC coefficient and the two AC coefficients corresponding to the first harmonic in the horizontal and vertical directions, followed by a second scan that includes the next three AC coefficients, and so on, according to the zigzag scan in Figure 9.3. This scheme is simple to implement, but all high-frequency information is postponed to the later scans. The result is that the reconstructed images from the early scans are blurred.

Successive approximation encoding improves on spectral selection by sending the DCT coefficients corresponding to all frequencies but keeping the transmission rate down by sending only the n_1 most significant bits of each coefficient first ($n_1 = 4$ in Figure 9.6 (b)), followed by the n_2 most significant bits ($n_2 = 1$ in Figure 9.6 (b)), and so on. This method gives very good reconstructed image quality, even for the very early scans. Figure 9.6 contrasts the two progressive encoding methods.

A combination of spectral selection and successive approximation can yield very efficient compression results with good reconstruction quality. Pennebaker and Mitchell (1993) report that using all bits of the DC coefficients and reducing precision on all of the AC coefficients for a rate of 0.24 bit/pixel produces quality slightly superior to spectral selection at 0.36 bit/pixel with the DC and the first five AC coefficients sent with full precision. This is an intuitive result: the DC coefficient is often the highest energy transform coefficient in a block, and DC mismatch in adjacent blocks is a common cause of blocking artifacts, so the DC coefficient requires an accurate representation. Further, the representation of higher frequencies is impossible in the early scans of spectral selection, and even a coarse representation makes high-frequency spectral content evident to the viewer. Pennebaker and Mitchell (1993) summarize this by saying that the spectral selection mode produces no distortion below cutoff, but maximum

a

0
1
2
First Scan

3
4
5
Second Scan

61
62
63
*n*th Scan

b

7 6 5 4
MSB →
First Scan

3
Second Scan

0
(LSB)
*n*th Scan

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
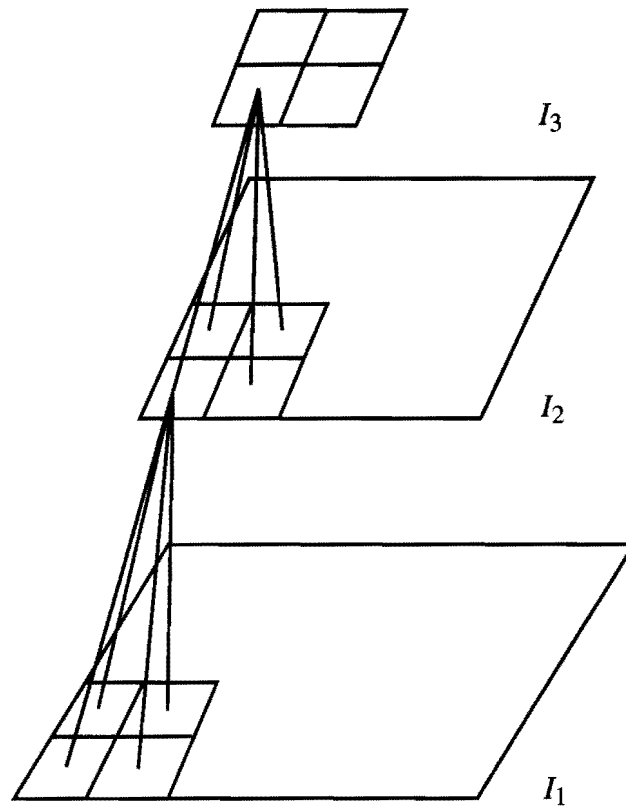
FIGURE

9.7   Pyramidal Filtering and Decimation of an Image

display. The JPEG standard accommodates these applications by specifying a hierarchical encoding mode based upon pyramidal encoding techniques, where each resolution can differ from adjacent ones by a factor of two in either the horizontal or vertical directions, or both. The hierarchical (pyramidal) encoding procedure is as follows.

The original image is filtered and downsampled by the required multiples of two, and the low-resolution image is coded using any of the lossy modes or even the lossless encoding mode. The compressed image is then decoded and upsampled, and subtracted from the next-higher-resolution image. The resulting difference image is then encoded by one of the encoding modes. This procedure is continued until all resolutions are coded. Note that all encodings may be lossy, lossless, or lossy followed by a final lossless encoding.

Figure 9.7 shows the generation of the pyramidal image structure by filtering and decimation. Figure 9.8 shows a block diagram of a hierarchical or pyramidal encoder that has the images in Figure 9.7 as input. Note that this figure represents the encoder only, yet decoders for all but the base-level image must be implemented in the encoder because the decoded and interpolated im-
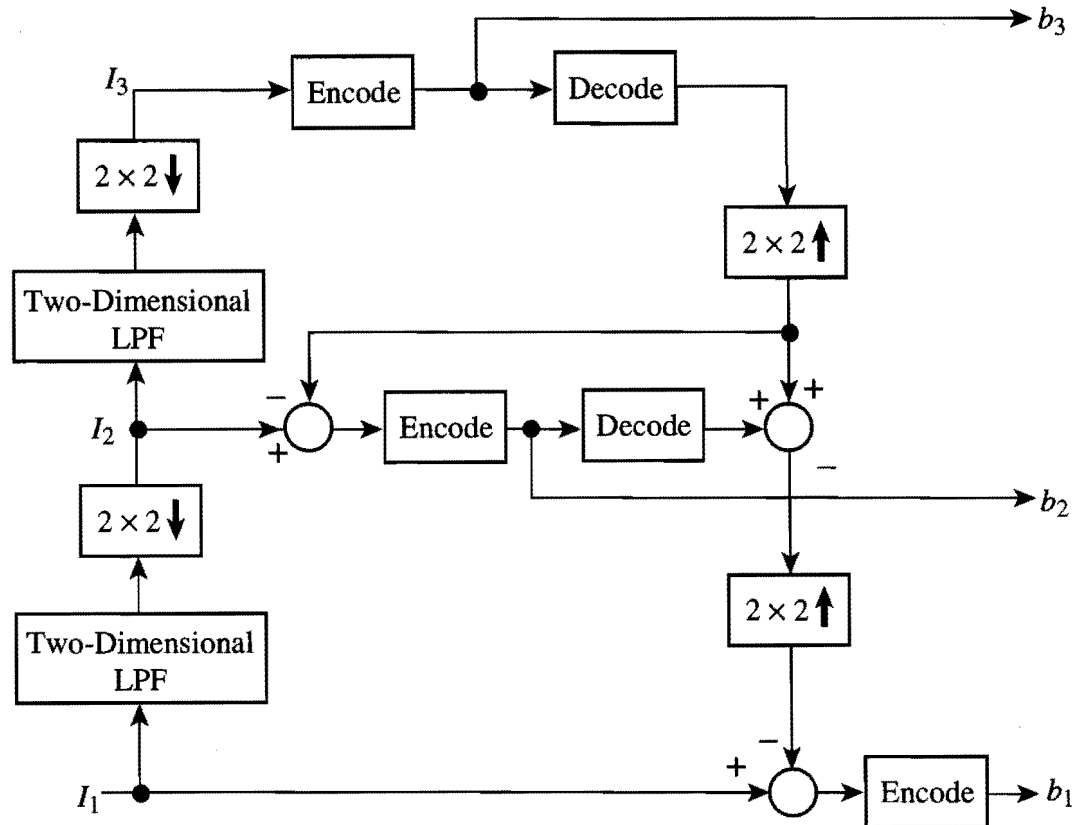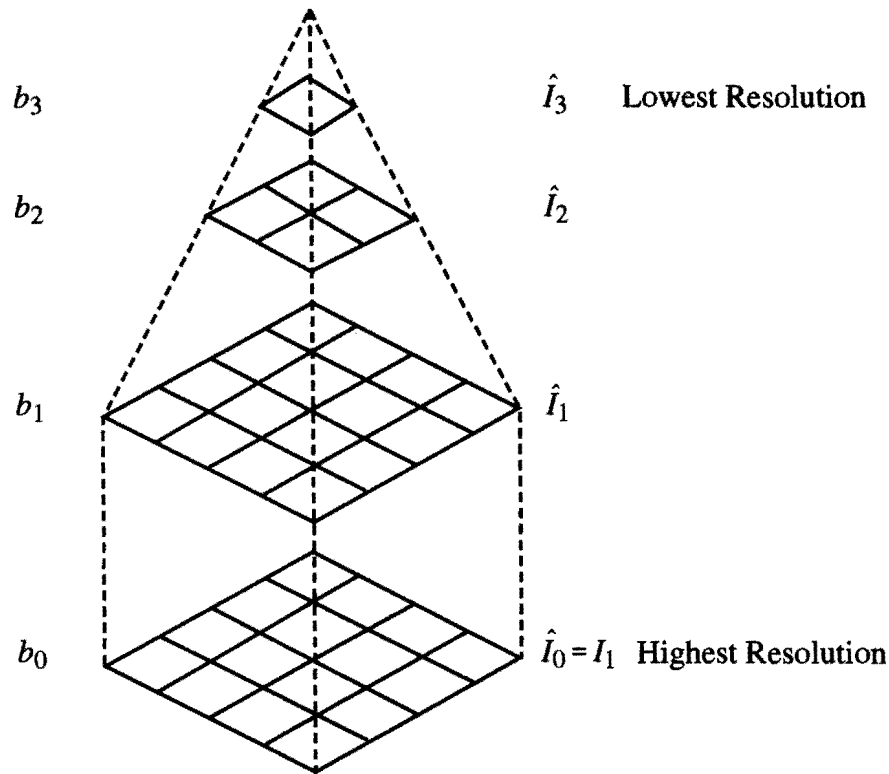
FIGURE

9.8   Encoder for Three-Level Pyramidal Coding

ages from the upper levels are needed to encode all lower levels. The original filtered and decimated images cannot be used directly because encoding errors from upper levels will accumulate at lower levels. In Figure 9.8, b_1, b_2, and b_3 are the transmitted data streams needed at the decoder.

The encoded and transmitted images are depicted in Figure 9.9 by the labels on the left of the pyramid, or alternatively, this pyramid can represent the decoded images, as labeled to the right of the pyramid. If lossy coding of the upper three levels is employed, lossless reconstruction is still possible by sending an appropriate additional data sequence, indicated in the figure by b_0. This bitstream would be generated at the encoder to represent the difference between I_1 and $\hat{I}_1$. Completely lossless transmission of all levels in the pyramid is also possible with appropriate lossless encoders in Figure 9.8. Notice that hierarchical encoding can also be viewed as a version of progressive transmission where the progression is in spatial resolution not image reconstruction quality.

Since scaling of images on a platform can consume substantial CPU time, hierarchical coding can be a nice enhancement. Of course, it is generated at the expense of greater storage requirements at the encoder and an increase in

b_3 $\hat{I}_3$ Lowest Resolution

b_2 $\hat{I}_2$

b_1 $\hat{I}_1$

b_0 $\hat{I}_0 = I_1$ Highest Resolution

FIGURE

9.9   Encoded Levels and Decoded Image Pyramid

data rate of up to 33%. This increase in data rate is implied by the increased number of pixels in all of the levels combined. Specifically, if the original image is $N \times N$, then with repeated factor-of-two decimation, the next levels will be $N/2 \times N/2$, $N/4 \times N/4$, and so on, and the total number of pixels in any number of levels is upper bounded by

$$N^2 \left[1 + \frac{1}{4} + \frac{1}{4^2} + \cdots \right] \le \frac{4N^2}{3}$$

Performance of the hierarchical mode is better than other JPEG modes at very low bit rates, so this might also be an attraction for some applications. Consistent with the JPEG decoder syntax approach, the downsampling filter is not specified, but the upsampling filter is. The interpolated value between two pixels from the lower-resolution image is a truncated average of the two pixels. The left column and the top row of the upsampled and lower-resolution image are aligned, and the rightmost column and bottom row of the lower-resolution image are replicated to produce the necessary interpolations in the upsampled image. A suggested compatible downsampling filter is given in the standard.

9.5   Entropy Coding

Entropy coding in the JPEG standard is accomplished in two steps. First, the coefficients are converted into an intermediate sequence of symbols, and then these symbols are coded using Huffman coding or arithmetic coding. Since the DC value of an 8 × 8 block is differentially encoded, intermediate symbol sequence generation is slightly different for the AC coefficients than for the differenced DC coefficients, and since the statistics of the DC and AC coefficients are quite different, they use separate Huffman tables. We begin with a description of the intermediate sequence generation and coding for the AC coefficients.

The baseline sequential JPEG only allows 8-bit integer pixel inputs, but the AC coefficients can be 3 bits larger, so the AC amplitudes fall in the range [−1023, 1023]. The zigzag ordering of the AC coefficients is mapped into an intermediate sequence of what are called "symbol-1" and "symbol-2" pairs. Symbol-1 consists of (RUNLENGTH, SIZE), where RUNLENGTH is the length of zero values preceding the next nonzero AC coefficient and SIZE is the number of bits needed to code the next nonzero coefficient amplitude, which is symbol-2. So, symbol-2 is (AMPLITUDE), which is the value of the AC coefficient. RUNLENGTH takes the values 0 to 15, and a symbol-1 pair of (15, 0) represents a runlength of 16 zero-valued AC coefficients. Since there can be long runs of zero-valued coefficients, up to three consecutive (15, 0) pairs followed by a symbol-1 that completes the runlength and a single symbol-2 are allowed. There are 63 possible AC coefficients in the 8 × 8 block, and this coding method allows for the possibility of 63 zero values. If the last run of zeros includes the last AC coefficient in the block, no symbol-2 is sent after the final run of zeros, but instead, the (0, 0) end-of-block (EOB) symbol is sent that denotes the end of the 8 × 8 block.

The symbol-1 sequence is entropy coded, Huffman coded in the baseline, but symbol-2 is assigned its direct binary representation if positive, or the one's complement representation if it is negative. Thus, only the symbol-1 sequence is Huffman compressed. Note that direct Huffman coding of all of the AC coefficient amplitudes would require 2047 entries, but by Huffman coding only the (RUNLENGTH, SIZE) pairs, only 16 (RUNLENGTH values) ×10 (SIZE bits)+ EOB = 161 entries, are needed. A partial sample table for this Huffman code is given in Table 9.1, and a table for the SIZE classifications is given in Table 9.2. Only the first 12 categories are used here.

The differenced DC coefficient stream covers the range [−2047, 2047] because of the differencing, and the symbol-1 sequence now just consists of SIZE. The

T A B L E 9.1 Partial Huffman Code for AC Coefficient (RUNLENGTH, SIZE) Pairs

Run/Size	Code Length	Codeword
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110

T A B L E   SIZE Categories

9.2

SIZE	DC Difference	Code
0	0	—
1	−1, 1	0, 1
2	−3, −2, 2, 3	00, 01, 10, 11
3	−7, . . . , −4, 4, . . . , 7	000, . . . , 011, 100, . . . , 111
4	−15, . . . , −8, 8, . . . , 15	0000, . . . , 0111, 1000, . . . , 1111
⋮	⋮	⋮
16	32,768	—

T A B L E   Huffman Code for the Luminance Difference DC SIZE

9.3

SIZE Category	Code Length	Codeword
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

symbol-2 sequence for the differenced DC values is again AMPLITUDE. As for the AC coefficients, only the symbol-1 sequence is entropy coded. Thus, rather than requiring a Huffman code with 4095 entries, only 12 entries are needed to Huffman code the SIZE information. Table 9.3 gives a typical Huffman code assignment for the differenced DC SIZE.

9.5.1 Example of DCT Coefficient Encoding

Let the DCT coefficients in a particular 8 × 8 image block be as shown below:

```
 48  12   0   0   0   0   0   0
-10   8   0   0   0   0   0   0
  2   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
  0   0   0   0   0   0   0   0
```

If we assume that the DC coefficient in the previous block was 40, the difference is thus 8. To encode this differenced DC value, we first find the SIZE in Table 9.2, refer to Table 9.3 to find the Huffman code for this SIZE, then append the code for the AMPLITUDE in this SIZE category. Thus, for the differenced DC value of 8, the assigned binary code is 1011000. To code the AC coefficients, we arrange them in order using the zigzag scan, find the correct combinations of RUNLENGTH and SIZE, look up the Huffman code for this pair in Table 9.1, and then append the appropriate code from Table 9.2. Thus, the ordered coefficients are 12, −10, 2, and 8, followed by 58 zeros to complete the block. To get the code for 12, we have no zeros before it and it falls in SIZE category 4, so from Table 9.1, we find the code for (0, 4) to be 1011 and then the 4-bit binary code for 12 is 1100, so that the complete codeword for 12 is 10111100. For −10, we have no preceding zeros and it falls into SIZE category 4, so the code for −10 is 10110101. Proceeding in this fashion, we code 2 as 0110 and 8 as 10111000. Since there are no intervening nonzero coefficients before the end of the block, the 58 zeros are encoded as EOB or 1010.

9.6   Image Data Conventions

While JPEG's goal was interoperability, there are a number of details that are not specified, including the image resolution, the color space for natural images, and methods for converting pixels with other than 8- or 12-bit accuracy. JPEG does have specifications for how to handle multicomponent images and flexibility in how a user might want to encode and play back the compressed image data. Certainly, since JPEG is intended to address a wide range of applications, it has to encompass images with several color components and varying resolutions.

JPEG provides for a maximum of 255 image components, each component represented by a rectangular array of pixels, denoted here by x_i horizontal pixels and y_i vertical pixels for the ith component. Each of these pixels has two levels of precision for lossy DCT coding, 8 and 12 bits; for lossless coding, the pixels can have 2 to 16 bits accuracy. The components are also allowed to have differing resolutions, which are expressed in JPEG as a fraction of a maximum resolution by ratios of sampling factors. That is, if we let $X = \max(x_i)$ and $Y = \max(y_i)$ denote the maximum horizontal and vertical resolutions, respectively, then x_i and y_i for the ith component can be expressed as

$$x_i = [X \times H_i/H_{max}] \tag{9.1}$$

$$y_i = [Y \times V_i/V_{max}] \tag{9.2}$$

where H_{max} and V_{max} are the maximum horizontal and vertical relative sampling factors, and H_i and V_i are the relative sampling factors of the ith components.

For example, if we have a YCbCr color space image that is 512×512, the baseline JPEG would support 8-bit precision pixels, and the luminance component would have $x_1 = y_1 = 512$. Since the chrominance components are usually subsampled by a factor of two, $x_2 = y_2 = x_3 = y_3 = 256$, so we get $X = \max(512, 256) = 512 = Y$, with relative sampling factors $H_1 = V_1 = 2$ and $H_2 = V_2 = 1$, so that $H_{max} = V_{max} = 2$. Thus, we can calculate

$$x_1 = [512 \times 2/2] = 512 = y_1$$

$$x_2 = [512 \times 1/2] = 256 = y_2 = x_3 = y_3$$

from equations (9.1) and (9.2).

With this information, it is still necessary to consider how a user might wish to encode and decode a multiple-component image. For instance, with a three-component image, it is possible that you might want to encode and store the luminance component for the entire image first, then encode and store the Cb chrominance for the entire image, then the Cr chrominance. Alternatively, you might wish to interleave coded versions of the three components for subsets of the image, which would then allow all components for a small portion of the image to be retrieved and decoded without waiting for this process for the complete image.

JPEG supports this interleaving by defining terms called *data units* and *minimum coded units* (MCUs). A data unit is the smallest possible unit of data that can be encoded and stored in JPEG; it is different for lossy and lossless coding modes. For lossy coding, the data unit is a block of 8×8 pixels processed by the DCT; for lossless coding, the data unit is a single sample or just one pixel.

The MCU specifies the number of data units per interleaved component when interleaving is used. For our previous example of a 512 × 512 image coded into YCbCr components with 2:1 subsampling of both chrominance components, we might have an 8 × 8 luminance data unit followed by one 4 × 4 Cb data unit and one 4 × 4 Cr data unit as one MCU.

9.7   Lossless Encoding Mode

Since lossless encoding with the DCT is difficult to specify, the lossless encoding mode chosen by the JPEG committee consists of a very simple predictive encoder that does not involve the DCT at all. Figure 9.10 illustrates the JPEG lossless predictive encoding operations. In this figure, the pixel to be predicted, denoted by X, can be predicted using the three pixels already available in the scanned image, labeled as A, B, and C. There are eight possible choices for the
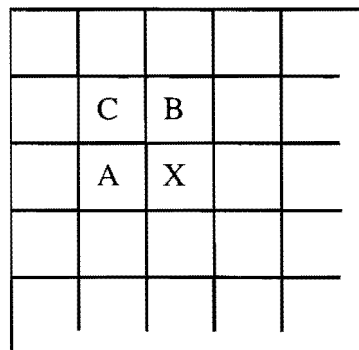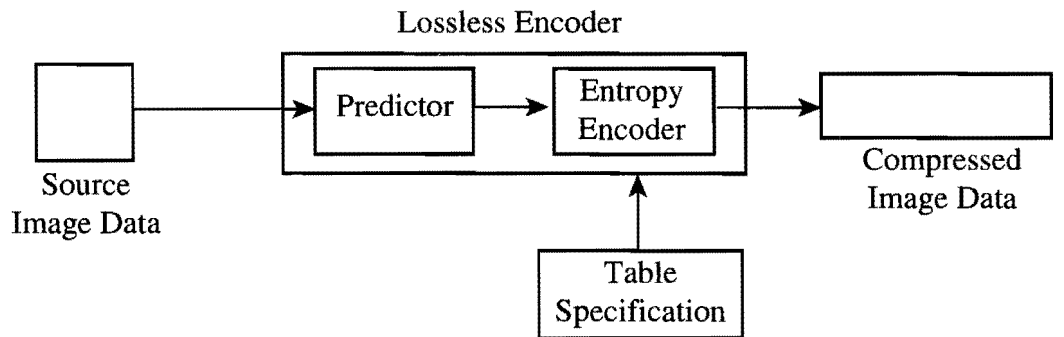
Predictors for Lossless Coding	
Selection Value	Prediction
0	No Prediction
1	A
2	B
3	C
4	A + B − C
5	A + ((B − C)/2)
6	B + ((A − C)/2)
7	(A + B)/2

FIGURE

9.10   Predictive Lossless JPEG Encoding

predictor, as shown in the lower right of Figure 9.10, and the particular predictor used by the encoder is specified in the header sent to the decoder. The "no prediction" selection is only available for hierarchical encoding. Depending upon the image, one of the predictors may outperform the others; however, various image coding experiments in the literature indicate that their performance can be relatively close on the average.

Similar to the coding methods used for the DCT coefficients, the prediction residual is encoded by representing each value as a SIZE category and AMPLITUDE. The input sample can have an accuracy from 2^2 to 2^{16}, and the prediction residual may be any value up to 2^{16}. The SIZE category symbol is taken from Table 9.2 and specifies the number of bits used to represent the amplitude of the prediction residual. The AMPLITUDE symbol is the actual encoded value of the residual. The SIZE category symbols are entropy coded. Thus, if the prediction residual error is 128, the SIZE category is 8 from Table 9.2; the binary codeword for an amplitude of 128 is 10000000. The SIZE category symbols are entropy coded.

9.8   Summary

An overview of the several encoding options available with JPEG is given in Table 9.4. From the table we can see that baseline JPEG supports 8-bit input samples and has default Huffman coding tables, but does not support arithmetic coding. Extended modes include twice as many Huffman tables as the baseline and four sets of arithmetic coding tables.

JPEG performance, in terms of quality versus bit rate, is really quite impressive, especially considering the relatively low complexity and the fact that these approaches have been around for a relatively long time. The lossy mode performance is summarized in Table 9.5 for 8-bit color photographs. The rate given is in total bits per pixel. This number basically comes from the fact that the luminance component has 8-bit accuracy and the two chrominance components have 8-bit accuracy each but half the sample rate. Thus, there is an average of 16 bits/pixel in the input image, and a rate of 0.5 bit/pixel is a 32:1 compression. The 2-bit/pixel images are considered "indistinguishable" from the original; the 0.25-bit/pixel performance is only classified as "fair." A rate of 0.083 bit/pixel was tested in the JPEG evaluations, but the quality of the image at that rate need only be "recognizable," and this rate is not specified as a part of the baseline.

The entropy coding step can have an impact on coder performance. As noted by Pennebaker and Mitchell (1993) for experiments on standard test images,

T A B L E   JPEG Encoding Options

9.4

	Baseline (All DCT Decoders)	Extended Processes	Lossless Processes
DCT-based	yes	yes	
Predictive-based			yes
8-bit samples	yes	yes	
12-bit samples		yes	
2-bit to 16-bit samples			yes
Sequential	yes	yes	yes
Progressive		yes	
Maximum Huffman tables	2 DC + 2 AC	4 DC + 4 AC	4 DC
Maximum Arithmetic tables	0	4 DC + 4 AC	4 DC
Decodes 4 comps	yes	yes	yes
Interleaved scans	yes	yes	yes
Noninterleaved	yes	yes	yes

T A B L E   JPEG Lossy Mode Performance 8-Bit Color Photographs

9.5

Rate (bits/pixel)	Perceived Quality	Applications
0.25	fair to good	few
0.50	good to very good	some
0.75	very good to excellent	many
1.50	excellent to indistinguishable	most
above 2.00	indistinguishable	nearly all

custom Huffman tables provide a rate reduction of 2.4–7.9% over the fixed Huffman tables, and the arithmetic coding allows an average reduction of 2.5% over the custom Huffman tables. Lossless coder performance yields about a 2:1 compression.