13

Transform Coding

13.1 Overview

In this chapter we will describe a technique in which the source output is decomposed, or transformed, into components that are then coded according to their individual characteristics. We will then look at a number of different transforms, including the popular discrete cosine transform, and discuss the issues of quantization and coding of the transformed coefficients. This chapter concludes with a description of the baseline sequential JPEG image-coding algorithm and some of the issues involved with transform coding of audio signals.

13.2 Introduction

In the previous chapter we developed a number of tools that can be used to transform a given sequence into different representations. If we take a sequence of inputs and transform them into another sequence in which most of the information is contained in only a few elements, we can then encode and transmit those elements, along with their location in the new sequence, resulting in data compression. In our discussion, we will use the terms "variance" and "information" interchangeably. The justification for this is shown in the results in Chapter 7. For example, recall that for a Gaussian source the differential entropy is given as $\frac{1}{2}\log 2\pi e\sigma^2$. Thus, an increase in the variance results in an increase in the entropy, which is a measure of the information contained in the source output.

To begin our discussion of transform coding, consider the following example.

Example 13.2.1:

Let's revisit Example 8.5.1. In Example 8.5.1, we studied the encoding of the output of a source that consisted of a sequence of pairs of numbers. Each pair of numbers corresponds to the height and weight of an individual. In particular, let's look at the sequence of outputs shown in Table 13.1.

If we look at the height and weight as the coordinates of a point in two-dimensional space, the sequence can be shown graphically as in Figure 13.1. Notice that the output

TABLE 13.1 Original sequence.

Height	Weight
65	170
75	188
60	150
70	170
56	130
80	203
68	160
50	110
40	80
50	153
69	148
62	140
76	164
64	120

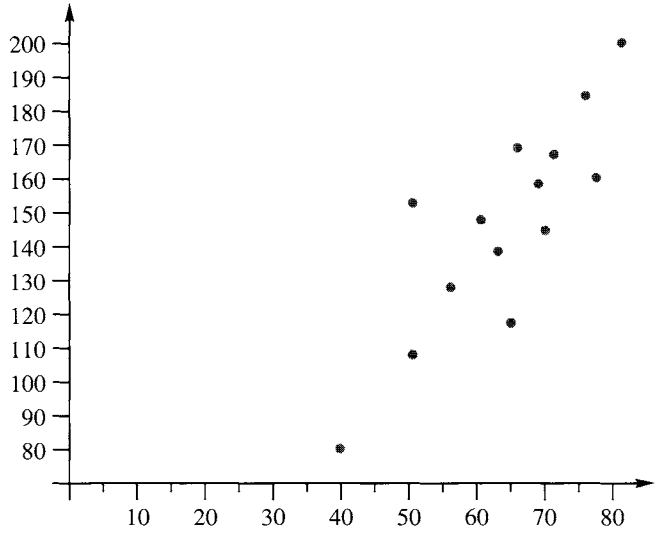

FIGURE 13.1 Source output sequence.

values tend to cluster around the line $y = 2.5x$. We can rotate this set of values by the transformation

$$\theta = \mathbf{A}\mathbf{x} \tag{13.1}$$

where $\mathbf{x}$ is the two-dimensional source output vector

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \tag{13.2}$$

x_0 corresponds to height and x_1 corresponds to weight, A is the rotation matrix

$$\mathbf{A} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \tag{13.3}$$

ϕ is the angle between the x-axis and the $y = 2.5x$ line, and

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \tag{13.4}$$

is the rotated or transformed set of values. For this particular case the matrix $\mathbf{A}$ is

$$\mathbf{A} = \begin{bmatrix} 0.37139068 & 0.92847669 \\ -0.92847669 & 0.37139068 \end{bmatrix} \tag{13.5}$$

and the transformed sequence (rounded to the nearest integer) is shown in Table 13.2. (For a brief review of matrix concepts, see Appendix B.)

Notice that for each pair of values, almost all the energy is compacted into the first element of the pair, while the second element of the pair is significantly smaller. If we plot this sequence in pairs, we get the result shown in Figure 13.2. Note that we have rotated the original values by an angle of approximately 68 degrees (arctan 2.5).

TABLE 13.2 Transformed sequence.

First Coordinate	Second Coordinate
182	3
202	0
162	0
184	−2
141	−4
218	1
174	−4
121	−6
90	−7
161	10
163	−9
153	−6
181	−9
135	−15

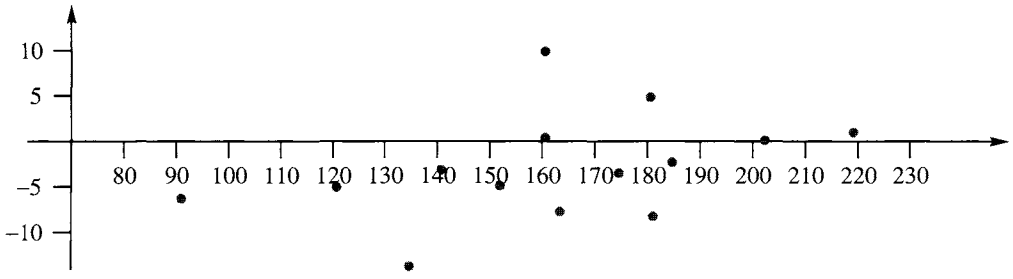

FIGURE 13.2 The transformed sequence.

Suppose we set all the second elements of the transformation to zero, that is, the second coordinates of the sequence shown in Table 13.2. This reduces the number of elements that need to be encoded by half. What is the effect of throwing away half the elements of the sequence? We can find that out by taking the inverse transform of the reduced sequence. The inverse transform consists of reversing the rotation. We can do this by multiplying the blocks of two of the transformed sequences with the second element in each block set to zero with the matrix

$$\mathbf{A}^{-1} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \tag{13.6}$$

and obtain the reconstructed sequence shown in Table 13.3. Comparing this to the original sequence in Table 13.1, we see that, even though we transmitted only half the number of elements present in the original sequence, this "reconstructed" sequence is very close to the original. The reason there is so little error introduced in the sequence $\{x_n\}$ is that for this

TABLE 13.3 Reconstructed sequence.

Height	Weight
68	169
75	188
60	150
68	171
53	131
81	203
65	162
45	112
34	84
60	150
61	151
57	142
67	168
50	125

particular transformation the error introduced into the $\{x_n\}$ sequence is equal to the error introduced into the $\{\theta_n\}$ sequence. That is,

$$\sum_{i=0}^{N-1}(x_i-\hat{x}_i)^2=\sum_{i=0}^{N-1}(\theta_i-\hat{\theta}_i)^2 \tag{13.7}$$

where $\{\hat{x}_n\}$ is the reconstructed sequence, and

$$\hat{\theta}_i=\begin{cases}\theta_i & i=0,2,4,\ldots\\ 0 & \text{otherwise}\end{cases} \tag{13.8}$$

(see Problem 1). The error introduced in the $\{\theta_n\}$ sequence is the sum of squares of the θ_ns that are set to zero. The magnitudes of these elements are quite small, and therefore the total error introduced into the reconstructed sequence is quite small also. ♦

We could reduce the number of samples we needed to code because most of the information contained in each pair of values was put into one element of each pair. As the other element of the pair contained very little information, we could discard it without a significant effect on the fidelity of the reconstructed sequence. The transform in this case acted on pairs of values; therefore, the maximum reduction in the number of significant samples was a factor of two. We can extend this idea to longer blocks of data. By compacting most of the information in a source output sequence into a few elements of the transformed sequence using a reversible transform, and then discarding the elements of the sequence that do not contain much information, we can get a large amount of compression. This is the basic idea behind transform coding.

In Example 13.2.1 we have presented a geometric view of the transform process. We can also examine the transform process in terms of the changes in statistics between the original and transformed sequences. It can be shown that we can get the maximum amount of compaction if we use a transform that decorrelates the input sequence; that is, the sample-to-sample correlation of the transformed sequence is zero. The first transform to provide decorrelation for discrete data was presented by Hotelling [179] in the *Journal of Educational Psychology* in 1933. He called his approach the *method of principal components*. The analogous transform for continuous functions was obtained by Karhunen [180] and Loéve [181]. This decorrelation approach was first utilized for compression, in what we now call transform coding, by Kramer and Mathews [182], and Huang and Schultheiss [183].

Transform coding consists of three steps. First, the data sequence $\{x_n\}$ is divided into blocks of size N. Each block is mapped into a transform sequence $\{\theta_n\}$ using a reversible mapping in a manner similar to that described in Example 13.2.1. As shown in the example, different elements of each block of the transformed sequence generally have different statistical properties. In Example 13.2.1, most of the energy of the block of two input values was contained in the first element of the block of two transformed values, while very little of the energy was contained in the second element. This meant that the second element of each block of the transformed sequence would have a small magnitude, while the magnitude of the first element could vary considerably depending on the magnitude of the elements in the input block. The second step consists of quantizing the transformed sequence. The quantization strategy used will depend on three main factors: the desired average bit rate, the statistics

of the various elements of the transformed sequence, and the effect of distortion in the transformed coefficients on the reconstructed sequence. In Example 13.2.1, we could take all the bits available to us and use them to quantize the first coefficient. In more complex situations, the strategy used may be very different. In fact, we may use different techniques, such as differential encoding and vector quantization [118], to encode the different coefficients.

Finally, the quantized value needs to be encoded using some binary encoding technique. The binary coding may be as simple as using a fixed-length code or as complex as a combination of run-length coding and Huffman or arithmetic coding. We will see an example of the latter when we describe the JPEG algorithm.

The various quantization and binary coding techniques have been described at some length in previous chapters, so we will spend the next section describing various transforms. We will then discuss quantization and coding strategies in the context of these transforms.

13.3 The Transform

All the transforms we deal with will be linear transforms; that is, we can get the sequence $\{\theta_n\}$ from the sequence $\{x_n\}$ as

$$\theta_n = \sum_{i=0}^{N-1} x_i a_{n,i}. \tag{13.9}$$

This is referred to as the *forward transform*. For the transforms that we will be considering, a major difference between the transformed sequence $\{\theta_n\}$ and the original sequence $\{x_n\}$ is that the characteristics of the elements of the θ sequence are determined by their position within the sequence. For example, in Example 13.2.1 the first element of each pair of the transformed sequence was more likely to have a large magnitude compared to the second element. In general, we cannot make such statements about the source output sequence $\{x_n\}$. A measure of the differing characteristics of the different elements of the transformed sequence $\{\theta_n\}$ is the variance σ_n^2 of each element. These variances will strongly influence how we encode the transformed sequence. The size of the block N is dictated by practical considerations. In general, the complexity of the transform grows more than linearly with N. Therefore, beyond a certain value of N, the computational costs overwhelm any marginal improvements that might be obtained by increasing N. Furthermore, in most real sources the statistical characteristics of the source output can change abruptly. For example, when we go from a silence period to a voiced period in speech, the statistics change drastically. Similarly, in images, the statistical characteristics of a smooth region of the image can be very different from the statistical characteristics of a busy region of the image. If N is large, the probability that the statistical characteristics change significantly within a block increases. This generally results in a larger number of the transform coefficients with large values, which in turn leads to a reduction in the compression ratio.

The original sequence $\{x_n\}$ can be recovered from the transformed sequence $\{\theta_n\}$ via the *inverse transform*:

$$x_n = \sum_{i=0}^{N-1} \theta_i b_{n,i}. \tag{13.10}$$

The transforms can be written in matrix form as

$$\theta = \mathbf{A}\mathbf{x} \tag{13.11}$$

$$\mathbf{x} = \mathbf{B}\theta \tag{13.12}$$

where $\mathbf{A}$ and $\mathbf{B}$ are $N \times N$ matrices and the (i, j)th element of the matrices is given by

$$[\mathbf{A}]_{i,j} = a_{i,j} \tag{13.13}$$

$$[\mathbf{B}]_{i,j} = b_{i,j}. \tag{13.14}$$

The forward and inverse transform matrices $\mathbf{A}$ and $\mathbf{B}$ are inverses of each other; that is, $\mathbf{AB} = \mathbf{BA} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

Equations (13.9) and (13.10) deal with the transform coding of one-dimensional sequences, such as sampled speech and audio sequences. However, transform coding is one of the most popular methods used for image compression. In order to take advantage of the two-dimensional nature of dependencies in images, we need to look at two-dimensional transforms.

Let $X_{i,j}$ be the (i, j)th pixel in an image. A general linear two-dimensional transform for a block of size $N \times N$ is given as

$$\Theta_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{i,j} a_{i,j,k,l}. \tag{13.15}$$

All two-dimensional transforms in use today are *separable* transforms; that is, we can take the transform of a two-dimensional block by first taking the transform along one dimension, then repeating the operation along the other direction. In terms of matrices, this involves first taking the (one-dimensional) transform of the rows, and then taking the column-by-column transform of the resulting matrix. We can also reverse the order of the operations, first taking the transform of the columns, and then taking the row-by-row transform of the resulting matrix. The transform operation can be represented as

$$\Theta_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_{k,i} X_{i,j} a_{i,j} \tag{13.16}$$

which in matrix terminology would be given by

$$\Theta = \mathbf{A}\mathbf{X}\mathbf{A}^T. \tag{13.17}$$

The inverse transform is given as

$$\mathbf{X} = \mathbf{B}\Theta\mathbf{B}^T. \tag{13.18}$$

All the transforms we deal with will be *orthonormal transforms*. An orthonormal transform has the property that the inverse of the transform matrix is simply its transpose because the rows of the transform matrix form an orthonormal basis set:

$$\mathbf{B} = \mathbf{A}^{-1} = \mathbf{A}^T. \tag{13.19}$$

For an orthonormal transform, the inverse transform will be given as

$$\mathbf{X} = \mathbf{A}^T \Theta \mathbf{A}. \tag{13.20}$$

Orthonormal transforms are energy preserving; that is, the sum of the squares of the transformed sequence is the same as the sum of the squares of the original sequence. We can see this most easily in the case of the one-dimensional transform:

$$\sum_{i=0}^{N-1} \theta_i^2 = \theta^T \theta \tag{13.21}$$

$$= (\mathbf{A}\mathbf{x})^T \mathbf{A}\mathbf{x} \tag{13.22}$$

$$= \mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x}. \tag{13.23}$$

If $\mathbf{A}$ is an orthonormal transform, $\mathbf{A}^T\mathbf{A} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$, then

$$\mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x} = \mathbf{x}^T \mathbf{x} \tag{13.24}$$

$$= \sum_{n=0}^{N-1} x_n^2 \tag{13.25}$$

and

$$\sum_{i=0}^{N-1} \theta_i^2 = \sum_{n=0}^{N-1} x_n^2. \tag{13.26}$$

The efficacy of a transform depends on how much energy compaction is provided by the transform. One way of measuring the amount of energy compaction afforded by a particular orthonormal transform is to take a ratio of the arithmetic mean of the variances of the transform coefficient to their geometric means [123]. This ratio is also referred to as the transform coding gain G_{TC}:

$$G_{TC} = \frac{\frac{1}{N}\sum_{i=0}^{N-1} \sigma_i^2}{(\prod_{i=0}^{N-1} \sigma_i^2)^{\frac{1}{N}}} \tag{13.27}$$

where σ_i^2 is the variance of the ith coefficient θ_i.

Transforms can be interpreted in several ways. We have already mentioned a geometric interpretation and a statistical interpretation. We can also interpret them as a decomposition of the signal in terms of a basis set. For example, suppose we have a two-dimensional orthonormal transform $\mathbf{A}$. The inverse transform can be written as

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \theta_0 \begin{bmatrix} a_{00} \\ a_{01} \end{bmatrix} + \theta_1 \begin{bmatrix} a_{10} \\ a_{11} \end{bmatrix} \tag{13.28}$$

We can see that the transformed values are actually the coefficients of an expansion of the input sequence in terms of the rows of the transform matrix. The rows of the transform matrix are often referred to as the basis vectors for the transform because they form an orthonormal basis set, and the elements of the transformed sequence are often called the transform coefficients. By characterizing the basis vectors in physical terms we can get a physical interpretation of the transform coefficients.

Example 13.3.1:

Consider the following transform matrix:

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{13.29}$$

We can verify that this is indeed an orthonormal transform.

Notice that the first row of the matrix would correspond to a "low-pass" signal (no change from one component to the next), while the second row would correspond to a "high-pass" signal. Thus, if we tried to express a sequence in which each element has the same value in terms of these two rows, the second coefficient should be zero. Suppose the original sequence is (α, α). Then

$$\begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{2}\alpha \\ 0 \end{bmatrix} \tag{13.30}$$

The "low-pass" coefficient has a value of $\sqrt{2}\alpha$, while the "high-pass" coefficient has a value of 0. The "low-pass" and "high-pass" coefficients are generally referred to as the low-frequency and high-frequency coefficients.

Let us take two sequences in which the components are not the same and the degree of variation is different. Consider the two sequences (3, 1) and (3, −1). In the first sequence the second element differs from the first by 2; in the second sequence, the magnitude of the difference is 4. We could say that the second sequence is more "high pass" than the first sequence. The transform coefficients for the two sequences are $(2\sqrt{2}, \sqrt{2})$ and $(\sqrt{2}, 2\sqrt{2})$, respectively. Notice that the high-frequency coefficient for the sequence in which we see a larger change is twice that of the high-frequency coefficient for the sequence with less change. Thus, the two coefficients do seem to behave like the outputs of a low-pass filter and a high-pass filter.

Finally, notice that in every case the sum of the squares of the original sequence is the same as the sum of the squares of the transform coefficients; that is, the transform is energy preserving, as it must be, since **A** is orthonormal. ♦

We can interpret one-dimensional transforms as an expansion in terms of the rows of the transform matrix. Similarly, we can interpret two-dimensional transforms as expansions in terms of matrices that are formed by the outer product of the rows of the transform matrix. Recall that the outer product is given by

$$\mathbf{x}\mathbf{x}^T = \begin{bmatrix} x_0x_0 & x_0x_1 & \cdots & x_0x_{N-1} \\ x_1x_0 & x_1x_1 & \cdots & x_1x_{N-1} \\ \vdots & \vdots & & \vdots \\ x_{N-1}x_0 & x_{N-1}x_1 & \cdots & x_{N-1}x_{N-1} \end{bmatrix} \tag{13.31}$$

To see this more clearly, let us use the transform introduced in Example 13.3.1 for a two-dimensional transform.

Example 13.3.2:

For an $N \times N$ transform $\mathbf{A}$, let $\alpha_{i,j}$ be the outer product of the ith and jth rows:

$$\alpha_{i,j} = \begin{bmatrix} a_{i0} \\ a_{i1} \\ \vdots \\ a_{iN-1} \end{bmatrix} \begin{bmatrix} a_{j0} & a_{j1} & \cdots & a_{jN-1} \end{bmatrix} \tag{13.32}$$

$$= \begin{bmatrix} a_{i0}a_{j0} & a_{i0}a_{j1} & \cdots & a_{i0}a_{jN-1} \\ a_{i1}a_{j0} & a_{i1}a_{j1} & \cdots & a_{i1}a_{jN-1} \\ \vdots & \vdots & & \vdots \\ a_{iN-1}a_{j0} & a_{iN-1}a_{j1} & \cdots & a_{iN-1}a_{jN-1} \end{bmatrix} \tag{13.33}$$

For the transform of Example 13.3.1, the outer products are

$$\alpha_{0,0} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \alpha_{0,1} = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \tag{13.34}$$

$$\alpha_{1,0} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \alpha_{1,1} = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{13.35}$$

From (13.20), the inverse transform is given by

$$\begin{bmatrix} x_{01} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} \theta_{00} & \theta_{01} \\ \theta_{10} & \theta_{11} \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{13.36}$$

$$= \frac{1}{2}\begin{bmatrix} \theta_{00}+\theta_{01}+\theta_{10}+\theta_{11} & \theta_{00}-\theta_{01}+\theta_{10}-\theta_{11} \\ \theta_{00}+\theta_{01}-\theta_{10}-\theta_{11} & \theta_{00}-\theta_{01}-\theta_{10}+\theta_{11} \end{bmatrix} \tag{13.37}$$

$$= \theta_{00}\alpha_{0,0} + \theta_{01}\alpha_{0,1} + \theta_{10}\alpha_{1,0} + \theta_{11}\alpha_{1,1}. \tag{13.38}$$

The transform values θ_{ij} can be viewed as the coefficients of the expansion of $\mathbf{x}$ in terms of the matrices $\alpha_{i,j}$. The matrices $\alpha_{i,j}$ are known as the *basis* matrices.

For historical reasons, the coefficient θ_{00}, corresponding to the basis matrix $\alpha_{0,0}$, is called the DC coefficient, while the coefficients corresponding to the other basis matrices are called AC coefficients. DC stands for direct current, which is current that does not change with time. AC stands for alternating current, which does change with time. Notice that all the elements of the basis matrix $\alpha_{0,0}$ are the same, hence the DC designation. ♦

In the following section we will look at some of the variety of transforms available to us, then at some of the issues involved in quantization and coding. Finally, we will describe in detail two applications, one for image coding and one for audio coding.

13.4 Transforms of Interest

In Example 13.2.1, we constructed a transform that was specific to the data. In practice, it is generally not feasible to construct a transform for the specific situation, for several

reasons. Unless the characteristics of the source output are stationary over a long interval, the transform needs to be recomputed often, and it is generally burdensome to compute a transform for every different set of data. Furthermore, the overhead required to transmit the transform itself might negate any compression gains. Both of these problems become especially acute when the size of the transform is large. However, there are times when we want to find out the best we can do with transform coding. In these situations, we can use data-dependent transforms to obtain an idea of the best performance available. The best-known data-dependent transform is the discrete Karhunen-Loéve transform (KLT). We will describe this transform in the next section.

13.4.1 Karhunen-Loéve Transform

The rows of the discrete Karhunen-Loéve transform [184], also known as the Hotelling transform, consist of the eigenvectors of the autocorrelation matrix. The autocorrelation matrix for a random process X is a matrix whose (i, j)th element $[R]_{i,j}$ is given by

$$[R]_{i,j} = E[X_n X_{n+|i-j|}]. \tag{13.39}$$

We can show [123] that a transform constructed in this manner will minimize the geometric mean of the variance of the transform coefficients. Hence, the Karhunen-Loéve transform provides the largest transform coding gain of any transform coding method.

If the source output being compressed is nonstationary, the autocorrelation function will change with time. Thus, the autocorrelation matrix will change with time, and the KLT will have to be recomputed. For a transform of any reasonable size, this is a significant amount of computation. Furthermore, as the autocorrelation is computed based on the source output, it is not available to the receiver. Therefore, either the autocorrelation or the transform itself has to be sent to the receiver. The overhead can be significant and remove any advantages to using the optimum transform. However, in applications where the statistics change slowly and the transform size can be kept small, the KLT can be of practical use [185].

Example 13.4.1:

Let us see how to obtain the KLT transform of size two for an arbitrary input sequence. The autocorrelation matrix of size two for a stationary process is

$$\mathbf{R} = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) \\ R_{xx}(1) & R_{xx}(0) \end{bmatrix} \tag{13.40}$$

Solving the equation $|\lambda \mathbf{I} - \mathbf{R}| = 0$, we get the two eigenvalues $\lambda_1 = R_{xx}(0) + R_{xx}(1)$, and $\lambda_2 = R_{xx}(0) - R_{xx}(1)$. The corresponding eigenvectors are

$$V_1 = \begin{bmatrix} \alpha \\ \alpha \end{bmatrix} \qquad V_2 = \begin{bmatrix} \beta \\ -\beta \end{bmatrix} \tag{13.41}$$

where α and β are arbitrary constants. If we now impose the orthonormality condition, which requires the vectors to have a magnitude of 1, we get

$$\alpha = \beta = \frac{1}{\sqrt{2}}$$

and the transform matrix $\mathbf{K}$ is

$$\mathbf{K} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{13.42}$$

Notice that this matrix is not dependent on the values of $R_{xx}(0)$ and $R_{xx}(1)$. This is only true of the 2×2 KLT. The transform matrices of higher order are functions of the autocorrelation values. ♦

Although the Karhunen-Loéve transform maximizes the transform coding gain as defined by (13.27), it is not practical in most circumstances. Therefore, we need transforms that do not depend on the data being transformed. We describe some of the more popular transforms in the following sections.

13.4.2 Discrete Cosine Transform

The discrete cosine transform (DCT) gets its name from the fact that the rows of the $N \times N$ transform matrix $\mathbf{C}$ are obtained as a function of cosines.

$$[\mathbf{C}]_{i,j} = \begin{cases} \sqrt{\frac{1}{N}} \cos \frac{(2j+1)i\pi}{2N} & i = 0, j = 0, 1, \ldots, N-1 \\ \sqrt{\frac{2}{N}} \cos \frac{(2j+1)i\pi}{2N} & i = 1, 2, \ldots, N-1, j = 0, 1, \ldots, N-1. \end{cases} \tag{13.43}$$

The rows of the transform matrix are shown in graphical form in Figure 13.3. Notice how the amount of variation increases as we progress down the rows; that is, the frequency of the rows increases as we go from top to bottom.

The outer products of the rows are shown in Figure 13.4. Notice that the basis matrices show increased variation as we go from the top-left matrix, corresponding to the θ_{00} coefficient, to the bottom-right matrix, corresponding to the $\theta_{(N-1)(N-1)}$ coefficient.

The DCT is closely related to the discrete Fourier transform (DFT) mentioned in Chapter 11, and in fact can be obtained from the DFT. However, in terms of compression, the DCT performs better than the DFT.

To see why, recall that when we find the Fourier coefficients for a sequence of length N, we assume that the sequence is periodic with period N. If the original sequence is as shown in Figure 13.5a, the DFT assumes that the sequence outside the interval of interest behaves in the manner shown in Figure 13.5b. This introduces sharp discontinuities, at the beginning and the end of the sequence. In order to represent these sharp discontinuities, the DFT needs nonzero coefficients for the high-frequency components. Because these components are needed only at the two endpoints of the sequence, their effect needs to be canceled out at other points in the sequence. Thus, the DFT adjusts other coefficients accordingly. When we discard the high-frequency coefficients (which should not have been there anyway) during

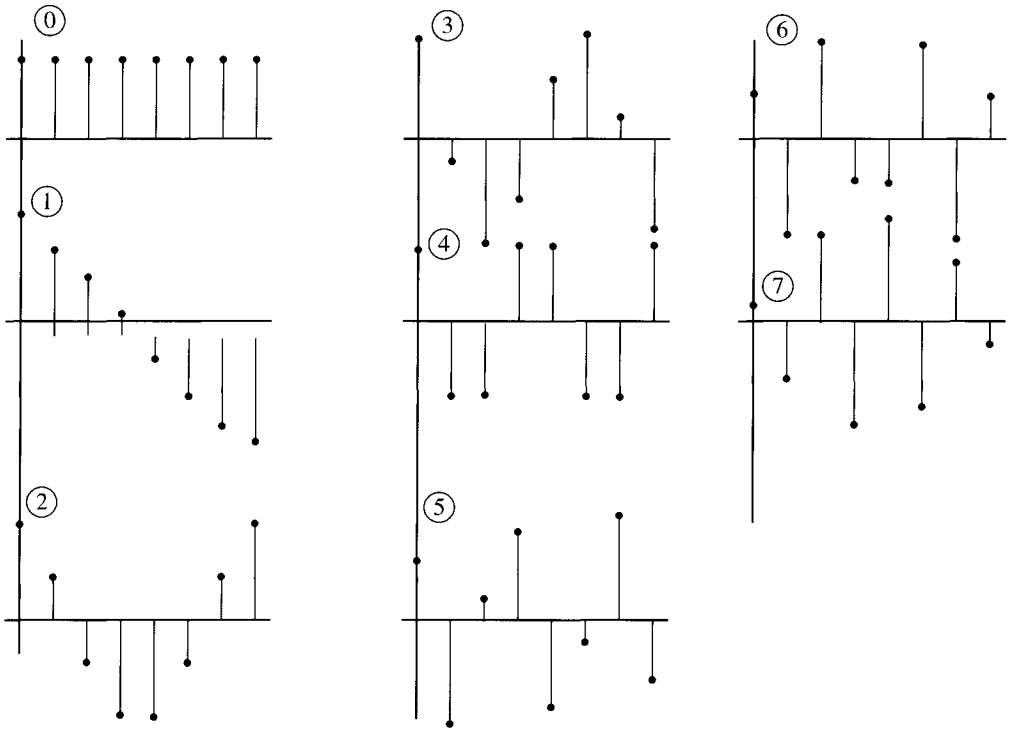

FIGURE 13.3 **Basis set for the discrete cosine transform. The numbers in the circles correspond to the row of the transform matrix.**

the compression process, the coefficients that were canceling out the high-frequency effect in other parts of the sequence result in the introduction of additional distortion.

The DCT can be obtained using the DFT by mirroring the original N-point sequence to obtain a $2N$-point sequence, as shown in Figure 13.6b. The DCT is simply the first N points of the resulting $2N$-point DFT. When we take the DFT of the $2N$-point mirrored sequence, we again have to assume periodicity. However, as we can see from Figure 13.6c, this does not introduce any sharp discontinuities at the edges.

The DCT is substantially better at energy compaction for most correlated sources when compared to the DFT [123]. In fact, for Markov sources with high correlation coefficient ρ,

$$\rho = \frac{E[x_n x_{n+1}]}{E[x_n^2]}, \tag{13.44}$$

the compaction ability of the DCT is very close to that of the KLT. As many sources can be modeled as Markov sources with high values for ρ, this superior compaction ability has made the DCT the most popular transform. It is a part of many international standards, including JPEG, MPEG, and CCITT H.261, among others.

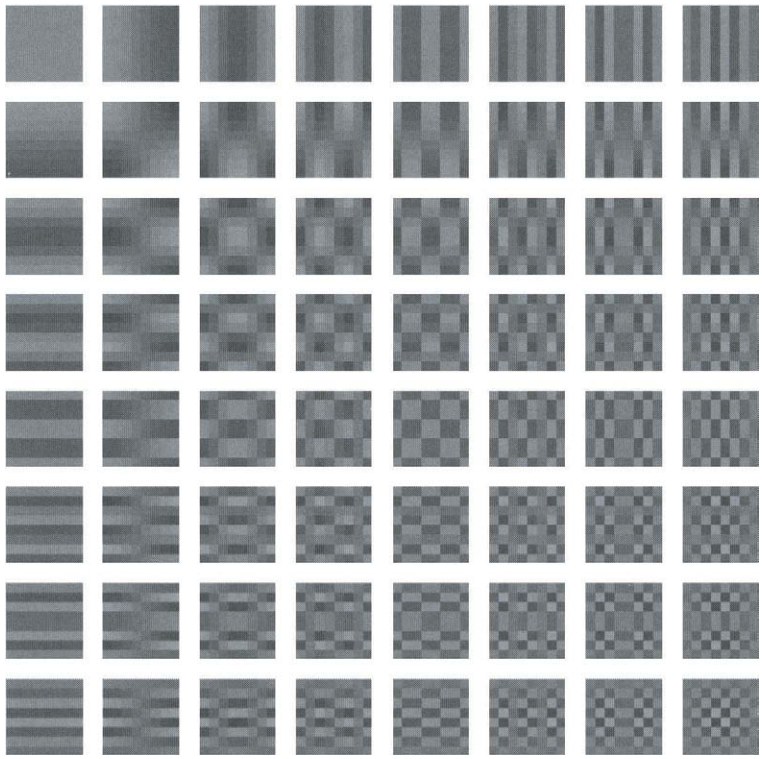

FIGURE 13.4 The basis matrices for the DCT.

13.4.3 Discrete Sine Transform

The discrete sine transform (DST) is a complementary transform to the DCT. Where the DCT provides performance close to the optimum KLT when the correlation coefficient ρ is large, the DST performs close to the optimum KLT in terms of compaction when the magnitude of ρ is small. Because of this property, it is often used as the complementary transform to DCT in image [186] and audio [187] coding applications.

The elements of the transform matrix for an $N \times N$ DST are

$$[\mathrm{S}]_{ij} = \sqrt{\frac{2}{N+1}} \sin \frac{\pi(i+1)(j+1)}{N+1} \qquad i, j = 0, 1, \ldots, N-1. \tag{13.45}$$

13.4.4 Discrete Walsh-Hadamard Transform

A transform that is especially simple to implement is the discrete Walsh-Hadamard transform (DWHT). The DWHT transform matrices are rearrangements of discrete Hadamard matrices, which are of particular importance in coding theory [188]. A Hadamard matrix of order N is defined as an $N \times N$ matrix H, with the property that $HH^T = NI$, where I is the $N \times N$

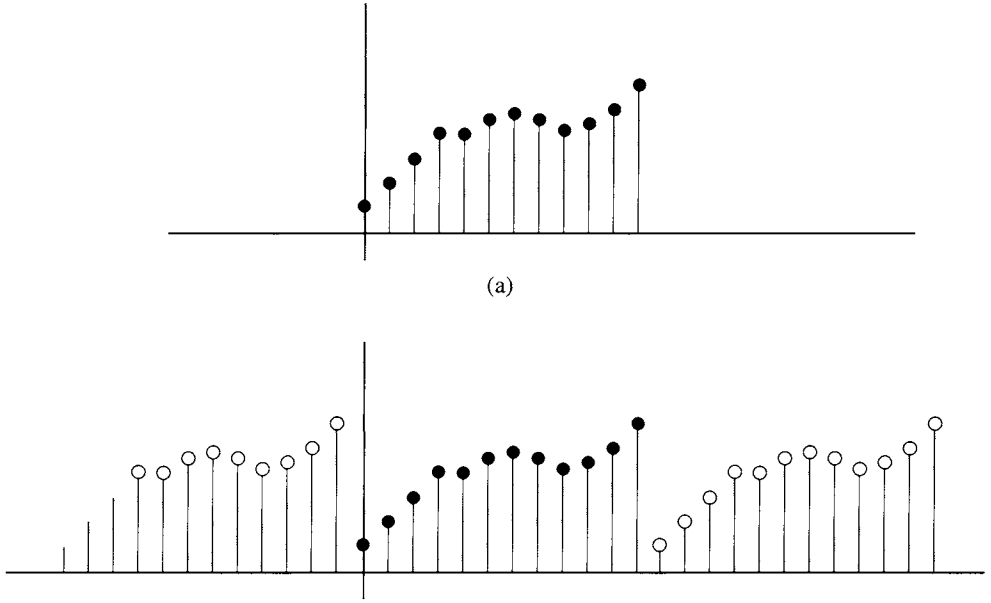

FIGURE 13. 5 Taking the discrete Fourier transform of a sequence.

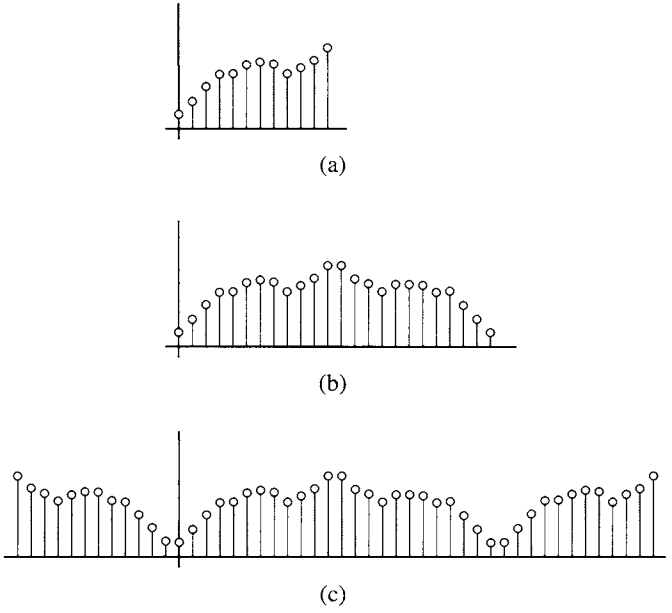

FIGURE 13. 6 Taking the discrete cosine transform of a sequence.

identity matrix. Hadamard matrices whose dimensions are a power of two can be constructed in the following manner:

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix} \tag{13.46}$$

with $H_1 = [1]$. Therefore,

$$H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{13.47}$$

$$H_4 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{13.48}$$

$$H_8 = \begin{bmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \tag{13.49}$$

The DWHT transform matrix H can be obtained from the Hadamard matrix by multiplying it by a normalizing factor so that $HH^T = I$ instead of NI, and by reordering the rows in increasing *sequency* order. The sequency of a row is half the number of sign changes in that row. In H_8 the first row has sequency 0, the second row has sequency 7/2, the third row has sequency 3/2, and so on. Normalization involves multiplying the matrix by $\frac{1}{\sqrt{N}}$. Reordering the H_8 matrix in increasing sequency order, we get

$$H = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} \tag{13.50}$$

Because the matrix without the scaling factor consists of ± 1, the transform operation consists simply of addition and subtraction. For this reason, this transform is useful in situations where minimizing the amount of computations is very important. However, the amount of energy compaction obtained with this transform is substantially less than the compaction obtained by the use of the DCT. Therefore, where sufficient computational power is available, DCT is the transform of choice.

13.5 Quantization and Coding of Transform Coefficients

If the amount of information conveyed by each coefficient is different, it makes sense to assign differing numbers of bits to the different coefficients. There are two approaches to assigning bits. One approach relies on the average properties of the transform coefficients, while the other approach assigns bits as needed by individual transform coefficients.

In the first approach, we first obtain an estimate of the variances of the transform coefficients. These estimates can be used by one of two algorithms to assign the number of bits used to quantize each of the coefficients. We assume that the relative variance of the coefficients corresponds to the amount of information contained in each coefficient. Thus, coefficients with higher variance are assigned more bits than coefficients with smaller variance.

Let us find an expression for the distortion, then find the bit allocation that minimizes the distortion. To perform the minimization we will use the method of Lagrange [189]. If the average number of bits per sample to be used by the transform coding system is R, and the average number of bits per sample used by the kth coefficient is R_k, then

$$R = \frac{1}{M}\sum_{k=1}^{M} R_k \tag{13.51}$$

where M is the number of transform coefficients. The reconstruction error variance for the kth quantizer $\sigma_{r_k}^2$ is related to the kth quantizer input variance $\sigma_{\theta_k}^2$ by the following:

$$\sigma_{r_k}^2 = \alpha_k 2^{-2R_k}\sigma_{\theta_k}^2 \tag{13.52}$$

where α_k is a factor that depends on the input distribution and the quantizer.

The total reconstruction error is given by

$$\sigma_r^2 = \sum_{k=1}^{M} \alpha_k 2^{-2R_k}\sigma_{\theta_k}^2. \tag{13.53}$$

The objective of the bit allocation procedure is to find R_k to minimize (13.53) subject to the constraint of (13.51). If we assume that α_k is a constant α for all k, we can set up the minimization problem in terms of Lagrange multipliers as

$$J = \alpha\sum_{k=1}^{M} 2^{-2R_k}\sigma_{\theta_k}^2 - \lambda\left(R - \frac{1}{M}\sum_{k=1}^{M} R_k\right). \tag{13.54}$$

Taking the derivative of J with respect to R_k and setting it equal to zero, we can obtain this expression for R_k:

$$R_k = \frac{1}{2}\log_2\left(2\alpha \ln 2\sigma_{\theta_k}^2\right) - \frac{1}{2}\log_2\lambda. \tag{13.55}$$

Substituting this expression for R_k in (13.51), we get a value for λ:

$$\lambda = \prod_{k=1}^{M}\left(2\alpha \ln 2\sigma_{\theta_k}^2\right)^{\frac{1}{M}} 2^{-2R}. \tag{13.56}$$

Substituting this expression for λ in (13.55), we finally obtain the individual bit allocations:

$$R_k = R + \frac{1}{2}\log_2 \frac{\sigma_{\theta_k}^2}{\prod_{k=1}^{M}(\sigma_{\theta_k}^2)^{\frac{1}{M}}}. \tag{13.57}$$

Although these values of R_k will minimize (13.53), they are not guaranteed to be integers, or even positive. The standard approach at this point is to set the negative R_ks to zero. This will increase the average bit rate above the constraint. Therefore, the nonzero R_ks are uniformly reduced until the average rate is equal to R.

The second algorithm that uses estimates of the variance is a recursive algorithm and functions as follows:

1. Compute $\sigma_{\theta_k}^2$ for each coefficient.
2. Set $R_k = 0$ for all k and set $R_b = MR$, where R_b is the total number of bits available for distribution.
3. Sort the variances $\{\sigma_{\theta_k}^2\}$. Suppose $\sigma_{\theta_l}^2$ is the maximum.
4. Increment R_l by 1, and divide $\sigma_{\theta_l}^2$ by 2.
5. Decrement R_b by 1. If $R_b = 0$, then stop; otherwise, go to 3.

If we follow this procedure, we end up allocating more bits to the coefficients with higher variance.

This form of bit allocation is called *zonal sampling*. The reason for this name can be seen from the example of a bit allocation map for the 8×8 DCT of an image shown in Table 13.4. Notice that there is a zone of coefficients that roughly comprises the right lower diagonal of the bit map that has been assigned zero bits. In other words, these coefficients are to be discarded. The advantage to this approach is its simplicity. Once the bit allocation has been obtained, every coefficient at a particular location is always quantized using the same number of bits. The disadvantage is that, because the bit allocations are performed based on average value, variations that occur on the local level are not reconstructed properly. For example, consider an image of an object with sharp edges in front of a relatively plain background. The number of pixels that occur on edges is quite small compared to the total number of pixels. Therefore, if we allocate bits based on average variances, the coefficients that are important for representing edges (the high-frequency coefficients) will get few or

TABLE 13.4 Bit allocation map for an 8 × 8 transform.

8	7	5	3	1	1	0	0
7	5	3	2	1	0	0	0
4	3	2	1	1	0	0	0
3	3	2	1	1	0	0	0
2	1	1	1	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

no bits assigned to them. This means that the reconstructed image will not contain a very good representation of the edges.

This problem can be avoided by using a different approach to bit allocation known as *threshold coding* [190, 93, 191]. In this approach, which coefficient to keep and which to discard is not decided a priori. In the simplest form of threshold coding, we specify a threshold value. Coefficients with magnitude below this threshold are discarded, while the other coefficients are quantized and transmitted. The information about which coefficients have been retained is sent to the receiver as side information. A simple approach described by Pratt [93] is to code the first coefficient on each line regardless of the magnitude. After this, when we encounter a coefficient with a magnitude above the threshold value, we send two codewords: one for the quantized value of the coefficient, and one for the count of the number of coefficients since the last coefficient with magnitude greater than the threshold. For the two-dimensional case, the block size is usually small, and each "line" of the transform is very short. Thus, this approach would be quite expensive. Chen and Pratt [191] suggest scanning the block of transformed coefficients in a zigzag fashion, as shown in Figure 13.7. If we scan an 8×8 block of quantized transform coefficients in this manner, we will find that in general a large section of the tail end of the scan will consist of zeros. This is because

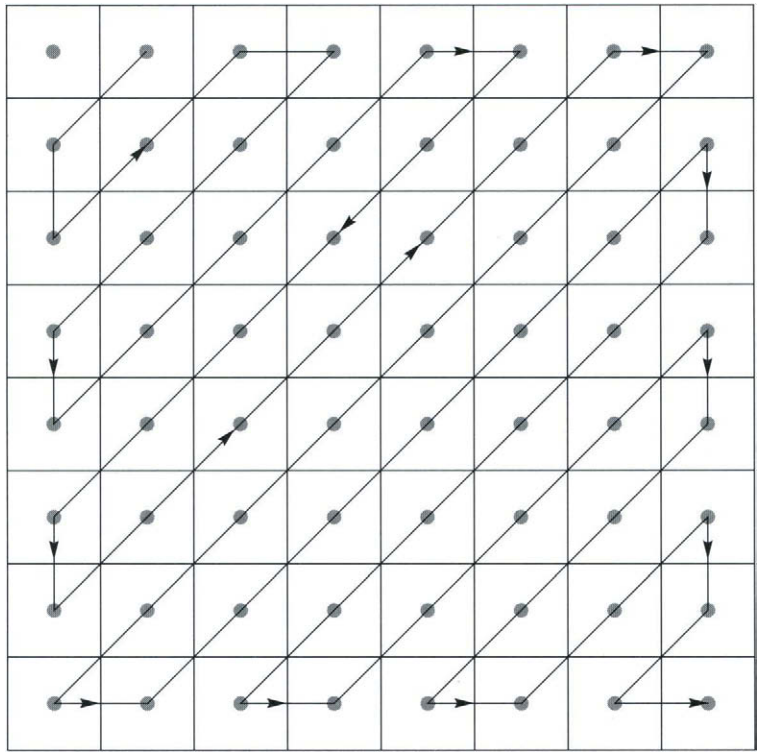

FIGURE 13.7 The zigzag scanning pattern for an 8 x 8 transform.

generally the higher-order coefficients have smaller amplitude. This is reflected in the bit allocation table shown in Table 13.4. As we shall see later, if we use midtread quantizers (quantizers with a zero output level), combined with the fact that the step sizes for the higher-order coefficients are generally chosen to be quite large, this means that many of these coefficients will be quantized to zero. Therefore, there is a high probability that after a few coefficients along the zigzag scan, all coefficients will be zero. In this situation, Chen and Pratt suggest the transmission of a special *end-of-block* (EOB) symbol. Upon reception of the EOB signal, the receiver would automatically set all remaining coefficients along the zigzag scan to zero.

The algorithm developed by the Joint Photographic Experts Group (JPEG), described in the next section, uses a rather clever variation of this approach.

13.6 Application to Image Compression—JPEG

The JPEG standard is one of the most widely known standards for lossy image compression. It is a result of the collaboration of the International Standards Organization (ISO), which is a private organization, and what was the CCITT (now ITU-T), a part of the United Nations. The approach recommended by JPEG is a transform coding approach using the DCT. The approach is a modification of the scheme proposed by Chen and Pratt [191]. In this section we will briefly describe the baseline JPEG algorithm. In order to illustrate the various components of the algorithm, we will use an 8×8 block of the Sena image, shown in Table 13.5. For more details, see [10].

13.6.1 The Transform

The transform used in the JPEG scheme is the DCT transform described earlier. The input image is first "level shifted" by 2^{P-1}; that is, we subtract 2^{P-1} from each pixel value, where P is the number of bits used to represent each pixel. Thus, if we are dealing with 8-bit images whose pixels take on values between 0 and 255, we would subtract 128 from each pixel so that the value of the pixel varies between -128 and 127. The image is divided into blocks of size 8×8, which are then transformed using an 8×8 forward DCT. If any dimension of the image is not a multiple of eight, the encoder replicates the last column or row until the

TABLE 13.5 An 8 x 8 block from the Sena image.

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156

TABLE 13.6 The DCT coefficients corresponding to the block of data from the Sena image after level shift.

39.88	6.56	−2.24	1.22	−0.37	−1.08	0.79	1.13
−102.43	4.56	2.26	1.12	0.35	−0.63	−1.05	−0.48
37.77	1.31	1.77	0.25	−1.50	−2.21	−0.10	0.23
−5.67	2.24	−1.32	−0.81	1.41	0.22	−0.13	0.17
−3.37	−0.74	−1.75	0.77	−0.62	−2.65	−1.30	0.76
5.98	−0.13	−0.45	−0.77	1.99	−0.26	1.46	0.00
3.97	5.52	2.39	−0.55	−0.051	−0.84	−0.52	−0.13
−3.43	0.51	−1.07	0.87	0.96	0.09	0.33	0.01

final size is a multiple of eight. These additional rows or columns are removed during the decoding process. If we take the 8×8 block of pixels shown in Table 13.5, subtract 128 from it, and take the DCT of this level-shifted block, we obtain the DCT coefficients shown in Table 13.6. Notice that the lower-frequency coefficients in the top-left corner of the table have larger values than the higher-frequency coefficients. This is generally the case, except for situations in which there is substantial activity in the image block.

13.6.2 Quantization

The JPEG algorithm uses uniform midtread quantization to quantize the various coefficients. The quantizer step sizes are organized in a table called the *quantization table* and can be viewed as the fixed part of the quantization. An example of a quantization table from the JPEG recommendation [10] is shown in Table 13.7. Each quantized value is represented by a label. The label corresponding to the quantized value of the transform coefficient θ_{ij} is obtained as

$$l_{ij} = \left\lfloor \frac{\theta_{ij}}{Q_{ij}} + 0.5 \right\rfloor \tag{13.58}$$

TABLE 13.7 Sample 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

TABLE 13.8 The quantizer labels obtained by using the quantization table on the coefficients.

2	1	0	0	0	0	0	0
−9	0	0	0	0	0	0	0
3	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

where Q_{ij} is the (i, j)th element of the quantization table, and $\lfloor x \rfloor$ is the largest integer smaller than x. Consider the θ_{00} coefficient from Table 13.6. The value of θ_{00} is 39.88. From Table 13.7, Q_{00} is 16. Therefore,

$$l_{00} = \left\lfloor \frac{39.88}{16} + 0.5 \right\rfloor = \lfloor 2.9925 \rfloor = 2. \tag{13.59}$$

The reconstructed value is obtained from the label by multiplying the label with the corresponding entry in the quantization table. Therefore, the reconstructed value of θ_{00} would be $l_{00} \times Q_{00}$, which is $2 \times 16 = 32$. The quantization error in this case is $39.88 - 32 = -7.88$. Similarly, from Tables 13.6 and 13.7, θ_{01} is 6.56 and Q_{01} is 11. Therefore,

$$l_{01} = \left\lfloor \frac{6.56}{11} + 0.5 \right\rfloor = \lfloor 1.096 \rfloor = 1. \tag{13.60}$$

The reconstructed value is 11, and the quantization error is $11 - 6.56 = 4.44$. Continuing in this fashion, we obtain the labels shown in Table 13.8.

From the sample quantization table shown in Table 13.7, we can see that the step size generally increases as we move from the DC coefficient to the higher-order coefficients. Because the quantization error is an increasing function of the step size, more quantization error will be introduced in the higher-frequency coefficients than in the lower-frequency coefficients. The decision on the relative size of the step sizes is based on how errors in these coefficients will be perceived by the human visual system. Different coefficients in the transform have widely different perceptual importance. Quantization errors in the DC and lower AC coefficients are more easily detectable than the quantization error in the higher AC coefficients. Therefore, we use larger step sizes for perceptually less important coefficients.

Because the quantizers are all midtread quantizers (that is, they all have a zero output level), the quantization process also functions as the thresholding operation. All coefficients with magnitudes less than half the corresponding step size will be set to zero. Because the step sizes at the tail end of the zigzag scan are larger, the probability of finding a long run of zeros increases at the end of the scan. This is the case for the 8×8 block of labels shown in Table 13.8. The entire run of zeros at the tail end of the scan can be coded with an EOB code after the last nonzero label, resulting in substantial compression.

Furthermore, this effect also provides us with a method to vary the rate. By making the step sizes larger, we can reduce the number of nonzero values that need to be transmitted, which translates to a reduction in the number of bits that need to be transmitted.

13.6.3 Coding

Chen and Pratt [191] used separate Huffman codes for encoding the label for each coefficient and the number of coefficients since the last nonzero label. The JPEG approach is somewhat more complex but results in higher compression. In the JPEG approach, the labels for the DC and AC coefficients are coded differently.

From Figure 13.4 we can see that the basis matrix corresponding to the DC coefficient is a constant matrix. Thus, the DC coefficient is some multiple of the average value in the 8×8 block. The average pixel value in any 8×8 block will not differ substantially from the average value in the neighboring 8×8 block; therefore, the DC coefficient values will be quite close. Given that the labels are obtained by dividing the coefficients with the corresponding entry in the quantization table, the labels corresponding to these coefficients will be closer still. Therefore, it makes sense to encode the differences between neighboring labels rather than to encode the labels themselves.

Depending on the number of bits used to encode the pixel values, the number of values that the labels, and hence the differences, can take on may become quite large. A Huffman code for such a large alphabet would be quite unmanageable. The JPEG recommendation resolves this problem by partitioning the possible values that the differences can take on into categories. The size of these categories grows as a power of two. Thus, category 0 has only one member (0), category 1 has two members (-1 and 1), category 2 has four members (-3, -2, 2, 3), and so on. The category numbers are then Huffman coded. The number of codewords in the Huffman code is equal to the base two logarithm of the number of possible values that the label differences can take on. If the differences can take on 4096 possible values, the size of the Huffman code is $\log_2 4096 = 12$. The elements within each category are specified by tacking on extra bits to the end of the Huffman code for that category. As the categories are different sizes, we need a differing number of bits to identify the value in each category. For example, because category 0 contains only one element, we need no additional bits to specify the value. Category 1 contains two elements, so we need 1 bit tacked on to the end of the Huffman code for category 1 to specify the particular element in that category. Similarly, we need 2 bits to specify the element in category 2, 3 bits for category 3, and n bits for category n.

The categories and the corresponding difference values are shown in Table 13.9. For example, if the difference between two labels was 6, we would send the Huffman code for category 3. As category 3 contains the eight values $\{-7, -6, -5, -4, 4, 5, 6, 7\}$, the Huffman code for category 3 would be followed by 3 bits that would specify which of the eight values in category 3 was being transmitted.

The binary code for the AC coefficients is generated in a slightly different manner. The category C that a nonzero label falls in and the number of zero-valued labels Z since the last nonzero label form a pointer to a specific Huffman code as shown in Table 13.10. Thus, if the label being encoded falls in category 3, and there have been 15 zero-valued labels prior to this nonzero label in the zigzag scan, then we form the pointer $F/3$, which points

TABLE 13.9 Coding of the differences of the DC labels.

0			0			
1			−1	1		
2		−3	−2	2	3	
3	−7	···	−4	4	···	7
4	−15	···	−8	8	···	15
5	−31	···	−16	16	···	31
6	−63	···	−32	32	···	63
7	−127	···	−64	64	···	127
8	−255	···	−128	128	···	255
9	−511	···	−256	256	···	511
10	−1,023	···	−512	512	···	1,023
11	−2,047	···	−1,024	1,024	···	2,047
12	−4,095	···	−2,048	2,048	···	4,095
13	−8,191	···	−4,096	4,096	···	8,191
14	−16,383	···	−8,192	8,192	···	16,383
15	−32,767	···	−16,384	16,384	···	32,767
16			32,768			

TABLE 13.10 Sample table for obtaining the Huffman code for a given label value and run length. The values of Z are represented in hexadecimal.

Z/C	Codeword	Z/C	Codeword	···	Z/C	Codeword
0/0 (EOB)	1010			···	F/0 (ZRL)	11111111001
0/1	00	1/1	1100	···	F/1	1111111111110101
0/2	01	1/2	11011	···	F/2	1111111111110110
0/3	100	1/3	1111001	···	F/3	1111111111110111
0/4	1011	1/4	111110110	···	F/4	1111111111111000
0/5	11010	1/5	11111110110	···	F/5	1111111111111001
⋮	⋮	⋮	⋮		⋮	

to the codeword 1111111111110111. Because the label falls in category 3, we follow this codeword with 3 bits that indicate which of the eight possible values in category 3 is the value that the label takes on.

There are two special codes shown in Table 13.10. The first is for the end-of-block (EOB). This is used in the same way as in the Chen and Pratt [191] algorithm; that is, if a particular label value is the last nonzero value along the zigzag scan, the code for it is immediately followed by the EOB code. The other code is the ZRL code, which is used when the number of consecutive zero values along the zigzag scan exceeds 15.

To see how all of this fits together, let's encode the labels in Table 13.8. The label corresponding to the DC coefficient is coded by first taking the difference between the value of the quantized label in this block and the quantized label in the previous block. If we assume that the corresponding label in the previous block was −1, then the difference would be 3. From Table 13.9 we can see that this value falls in category 2. Therefore, we

would send the Huffman code for category 2 followed by the 2-bit sequence 11 to indicate that the value in category 2 being encoded was 3, and not −3, −2, or 2. To encode the AC coefficients, we first order them using the zigzag scan. We obtain the sequence

$$1\ -9\ 3\ 0\ 0\ 0\cdots 0$$

The first value, 1, belongs to category 1. Because there are no zeros preceding it, we transmit the Huffman code corresponding to 0/1, which from Table 13.10 is 00. We then follow this by a single bit 1 to indicate that the value being transmitted is 1 and not −1. Similarly, −9 is the seventh element in category 4. Therefore, we send the binary string 1011, which is the Huffman code for 0/4, followed by 0110 to indicate that −9 is the seventh element in category 4. The next label is 3, which belongs to category 2, so we send the Huffman code 01 corresponding to 0/2, followed by the 2 bits 11. All the labels after this point are 0, so we send the EOB Huffman code, which in this case is 1010. If we assume that the Huffman code for the DC coefficient was 2 bits long, we have sent a grand total of 21 bits to represent this 8×8 block. This translates to an average $\frac{21}{64}$ bits per pixel.

To obtain a reconstruction of the original block, we perform the dequantization, which simply consists of multiplying the labels in Table 13.8 with the corresponding values in Table 13.7. Taking the inverse transform of the quantized coefficients shown in Table 13.11 and adding 128, we get the reconstructed block shown in Table 13.12. We can see that in spite of going from 8 bits per pixel to $\frac{9}{32}$ bits per pixel, the reproduction is remarkably close to the original.

TABLE 13.11 The quantized values of the coefficients.

32	11	0	0	0	0	0	0
−108	0	0	0	0	0	0	0
42	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

TABLE 13.12 The reconstructed block.

123	122	122	121	120	120	119	119
121	121	121	120	119	118	118	118
121	121	120	119	119	118	117	117
124	124	123	122	122	121	120	120
130	130	129	129	128	128	128	127
141	141	140	140	139	138	138	137
152	152	151	151	150	149	149	148
159	159	158	157	157	156	155	155

FIGURE 13.8 Sinan image coded at 0.5 bits per pixel using the JPEG algorithm.

If we wanted an even more accurate reproduction, we could do so at the cost of increased bit rate by multiplying the step sizes in the quantization table by one-half and using these values as the new step sizes. Using the same assumptions as before, we can show that this will result in an increase in the number of bits transmitted. We can go in the other direction by multiplying the step sizes with a number greater than one. This will result in a reduction in bit rate at the cost of increased distortion.

Finally, we present some examples of JPEG-coded images in Figures 13.8 and 13.9. These were coded using shareware generated by the Independent JPEG Group (organizer, Dr. Thomas G. Lane). Notice the high degree of "blockiness" in the lower-rate image (Figure 13.8). This is a standard problem of most block-based techniques, and specifically of the transform coding approach. A number of solutions have been suggested for removing this blockiness, including postfiltering at the block edges as well as transforms that overlap the block boundaries. Each approach has its own drawbacks. The filtering approaches tend to reduce the resolution of the reconstructions, while the overlapped approaches increase the complexity. One particular overlapped approach that is widely used in audio compression is the modified DCT (MDCT), which is described in the next section.

13.7 Application to Audio Compression—The MDCT

As mentioned in the previous section, the use of the block based transform has the unfortunate effect of causing distortion at the block boundaries at low rates. A number of techniques that use overlapping blocks have been developed over the years [192]. One that has gained

FIGURE 13. 9 Sinan image coded at 0.25 bits per pixel using the JPEG algorithm.

wide acceptance in audio compression is a transform based on the discrete cosine transform called the modified discrete cosine transform (MDCT). It is used in almost all popular audio coding standards from *mp3* and AAC to Ogg Vorbis.

The MDCT used in these algorithms uses 50% overlap. That is, each block overlaps half of the previous block and half of the next block of data. Consequently, each audio sample is part of two blocks. If we were to keep all the frequency coefficients we would end up with twice as many coefficients as samples. Reducing the number of frequency coefficients results in the introduction of distortion in the inverse transform. The distortion is referred to as time domain aliasing [193]. The reason for the name is evident if we consider that the distortion is being introduced by subsampling in the frequency domain. Recall that sampling at less than the Nyquist frequency in the time domain leads to an overlap of replicas of the frequency spectrum, or frequency aliasing. The lapped transforms are successful because they are constructed in such a way that while the inverse transform of each block results in time-domain aliasing, the aliasing in consecutive blocks cancel each other out.

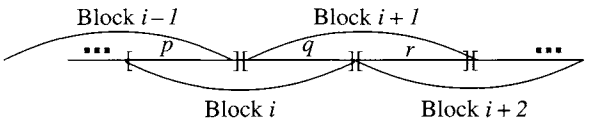

FIGURE 13. 10 Source output sequence.

Consider the scenario shown in Figure 13.10. Let's look at the coding for block i and block $i+1$. The inverse transform of the coefficients resulting from both these blocks will result in the audio samples in the subblock q. We assume that the blocksize is N and therefore the subblock size is $N/2$. The forward transform can be represented by an $N/2 \times N$ matrix P. Let us partition the matrix into two $N/2 \times N/2$ blocks, A and B. Thus

$$P = [A|B]$$

Let $x_i = [p|q]$, then the forward transform Px_i can be written in terms of the subblocks as

$$X_i = [A|B]\begin{bmatrix} p \\ q \end{bmatrix}$$

The inverse transform matrix Q can be represented by an $N \times N/2$, which can be partitioned into two $N/2 \times N/2$ blocks, C and D.

$$Q = \begin{bmatrix} C \\ D \end{bmatrix}$$

Applying the inverse transform, we get the reconstruction values $\hat{x}$

$$\hat{x}_i = QX_i = QPx_i = \begin{bmatrix} C \\ D \end{bmatrix}[A|B]\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} CAp + CBq \\ DAp + DBq \end{bmatrix}$$

Repeating the process for block $i+1$ we get

$$\hat{x}_{i+1} = QX_{i+1} = QPx_{i+1} = \begin{bmatrix} C \\ D \end{bmatrix}[A|B]\begin{bmatrix} q \\ r \end{bmatrix} = \begin{bmatrix} CAq + CBr \\ DAq + DBr \end{bmatrix}$$

To cancel out the aliasing in the second half of the block we need

$$CAq + CBr + DAp + DBq = q$$

From this we can get the requirements on the transform

$$CB = 0 \tag{13.61}$$

$$DA = 0 \tag{13.62}$$

$$CA + DB = I \tag{13.63}$$

Note that the same requirements will help cancel the aliasing in the first half of block i by using the second half of the inverse transform of block $i-1$. One selection that satisfies the last condition is

$$CA = \frac{1}{2}(I - J) \tag{13.64}$$

$$DB = \frac{1}{2}(I + J) \tag{13.65}$$

The forward modified discrete transform is given by the following equation:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi}{N}(k+\frac{1}{2})(n+\frac{1}{2}+\frac{N}{4})\right) \tag{13.66}$$

where x_n are the audio samples and X_k are the frequency coefficients. The inverse MDCT is given by

$$y_n = \frac{2}{N}\sum_{n=0}^{\frac{N}{2}-1} X_k \cos\left(\frac{2\pi}{N}(k+\frac{1}{2})(n+\frac{1}{2}+\frac{N}{4})\right) \tag{13.67}$$

or in terms of our matrix notation,

$$[P]_{i,j} = \cos\left(\frac{2\pi}{N}(i+\frac{1}{2})(j+\frac{1}{2}+\frac{N}{4})\right) \tag{13.68}$$

$$[Q]_{i,j} = \frac{2}{N}\cos\left(\frac{2\pi}{N}(i+\frac{1}{2})(j+\frac{1}{2}+\frac{N}{4})\right) \tag{13.69}$$

It is easy to verify that, given a value of N, these matrices satisfy the conditions for alias cancellation.

Thus, while the inverse transform for any one block will contain aliasing, by using the inverse transform of neighboring blocks the aliasing can be canceled. What about blocks that do not have neighbors—that is, the first and last blocks? One way to resolve this problem is to pad the sampled audio sequence with $N/2$ zeros at the beginning and end of the sequence. In practice, this is not necessary, because the data to be transformed is windowed prior to the transform. For the first and last blocks we use a special window that has the same effect as introducing zeros. For information on the design of windows for the MDCT, see [194]. For more on how the MDCT is used in audio compression techniques, see Chapter 16.

13.8 Summary

In this chapter we have described the concept of transform coding and provided some of the details needed for the investigation of this compression scheme. The basic encoding scheme works as follows:

- Divide the source output into blocks. In the case of speech or audio data, they will be one-dimensional blocks. In the case of images, they will be two-dimensional blocks. In image coding, a typical block size is 8×8. In audio coding the blocks are generally overlapped by 50%.
- Take the transform of this block. In the case of one-dimensional data, this involves pre-multiplying the N vector of source output samples by the transform matrix. In the case of image data, for the transforms we have looked at, this involves pre-multiplying the $N \times N$ block by the transform matrix and post-multiplying the result with the

transpose of the transform matrix. Fast algorithms exist for performing the transforms described in this chapter (see [195]).

- Quantize the coefficients. Various techniques exist for the quantization of these coefficients. We have described the approach used by JPEG. In Chapter 16 we describe the quantization techniques used in various audio coding algorithms.
- Encode the quantized value. The quantized value can be encoded using a fixed-length code or any of the different variable-length codes described in earlier chapters. We have described the approach taken by JPEG.

The decoding scheme is the inverse of the encoding scheme for image compression. For the overlapped transform used in audio coding the decoder adds the overlapped portions of the inverse transform to cancel aliasing.

The basic approach can be modified depending on the particular characteristics of the data. We have described some of the modifications used by various commercial algorithms for transform coding of audio signals.

Further Reading

1. For detailed information about the JPEG standard, *JPEG Still Image Data Compression Standard*, by W.B. Pennebaker and J.L. Mitchell [10], is an invaluable reference. This book also contains the entire text of the official draft JPEG recommendation, ISO DIS 10918-1 and ISO DIS 10918-2.
2. For a detailed discussion of the MDCT and how it is used in audio coding, an excellent source is *Introduction to Digital Audio Coding Standards*, by M. Bosi and R.E. Goldberg [194]
3. Chapter 12 in *Digital Coding of Waveforms*, by N.S. Jayant and P. Noll [123], provides a more mathematical treatment of the subject of transform coding.
4. A good source for information about transforms is *Fundamentals of Digital Image Processing*, by A.K. Jain [196]. Another one is *Digital Image Processing*, by R.C. Gonzales and R.E. Wood [96]. This book has an especially nice discussion of the Hotelling transform.
5. The bit allocation problem and its solutions are described in *Vector Quantization and Signal Compression*, by A. Gersho and R.M. Gray [5].
6. A very readable description of transform coding of images is presented in *Digital Image Compression Techniques*, by M. Rabbani and P.W. Jones [80].
7. *The Data Compression Book*, by M. Nelson and J.-L. Gailly [60], provides a very readable discussion of the JPEG algorithm.

13.9 Projects and Problems

1. A square matrix $\mathbf{A}$ has the property that $\mathbf{A}^T\mathbf{A} = \mathbf{A}\mathbf{A}^T = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. If X_1 and X_2 are two N-dimensional vectors and

$$\Theta_1 = \mathbf{A}X_1$$
$$\Theta_2 = \mathbf{A}X_2$$

then show that

$$|X_1 - X_2|^2 = |\Theta_1 - \Theta_2|^2 \tag{13.70}$$

2. Consider the following sequence of values:

10	11	12	11	12	13	12	11
10	−10	8	−7	8	−8	7	−7

(a) Transform each row separately using an eight-point DCT. Plot the resulting 16 transform coefficients.

(b) Combine all 16 numbers into a single vector and transform it using a 16-point DCT. Plot the 16 transform coefficients.

(c) Compare the results of (a) and (b). For this particular case would you suggest a block size of 8 or 16 for greater compression? Justify your answer.

3. Consider the following "image":

4	3	2	1
3	2	1	1
2	1	1	1
1	1	1	1

(a) Obtain the two-dimensional DWHT transform by first taking the one-dimensional transform of the rows, then taking the column-by-column transform of the resulting matrix.

(b) Obtain the two-dimensional DWHT transform by first taking the one-dimensional transform of the columns, then taking the row-by-row transform of the resulting matrix.

(c) Compare and comment on the results of (a) and (b).

4. (This problem was suggested by P.F. Swaszek.) Let us compare the energy compaction properties of the DCT and the DWHT transforms.

(a) For the Sena image, compute the mean squared value of each of the 64 coefficients using the DCT. Plot these values.

(b) For the Sena image, compute the mean squared value of each of the 64 coefficients using the DWHT. Plot these values.

(c) Compare the results of (a) and (b). Which transform provides more energy compaction? Justify your answer.

5. Implement the transform and quantization portions of the JPEG standard. For coding the labels use an arithmetic coder instead of the modified Huffman code described in this chapter.

   (a) Encode the Sena image using this transform coder at rates of (approximately) 0.25, 0.5, and 0.75 bits per pixel. Compute the mean squared error at each rate and plot the rate versus the mse.

   (b) Repeat part (a) using one of the public domain implementations of JPEG.

   (c) Compare the plots obtained using the two coders and comment on the relative performance of the coders.

6. One of the extensions to the JPEG standard allows for the use of multiple quantization matrices. Investigate the issues involved in designing a set of quantization matrices. Should the quantization matrices be similar or dissimilar? How would you measure their similarity? Given a particular block, do you need to quantize it with each quantization matrix to select the best? Or is there a computationally more efficient approach? Describe your findings in a report.