24 Transforms and JPEG

24.1 Image Transforms

The concept of a transform is familiar to mathematicians. A transform is a standard mathematical tool that is employed to solve problems in many areas. The idea is to transform a mathematical quantity (a number, a vector, a function, or anything else) to another form, where it may look unfamiliar but may have useful properties. The transformed quantity is used to solve a problem or to perform a computation, and the result is then transformed back to its original form.

A simple, illustrative example is Roman numerals. The ancient Romans presumably knew how to operate on such numbers, but when we have to, say, multiply two Roman numerals, we may find it more convenient to transform them into modern (Arabic) notation, multiply, and then transform the result back into a Roman numeral. Here is a simple example:

$$\text{XCVI} \times \text{XII} \rightarrow 96 \times 12 = 1152 \rightarrow \text{MCLII}.$$

An image can be compressed by transforming its pixels (which are correlated) to a representation where they are *decorrelated*. Compression is achieved if the new values are smaller, on average, than the original values. Lossy compression can be achieved by quantizing the transformed values. The decoder inputs the transformed values from the compressed stream and reconstructs the (precise or approximate) original data by applying the inverse transform. The transforms discussed in this chapter are *orthogonal*. Chapter 25 discusses *subband transforms*. Reference [Rao and Yip 00] is an excellent reference on transforms and their applications to data compression.

The term *decorrelated* means that the transformed values are independent of one another. As a result, they can be encoded independently, which makes it simpler to construct a statistical model. An image can be compressed if its representation has redundancy. The redundancy in images stems mostly from pixel correlation. If we transform the image to a representation where the pixels are decorrelated, we have eliminated the redundancy and the image has been maximally compressed.

To illustrate orthogonal transforms, we start with a simple example where we scan an image in raster order and group pairs of adjacent pixels. Because the pixels are correlated, the two pixels (x, y) of a pair normally have similar values. We now consider each pair of pixels a point in two-dimensional space, and we plot the points. We know that all the points of the form (x, x) are located on the 45° line $y = x$, so we expect our points to be concentrated around this line. Figure 24.2a shows the results of plotting the pixels of a typical image—where a pixel has values in the interval $[0, 255]$—in such a way. Most points form a cloud around the 45° line, and only a few points are located away from it. We now transform the image by rotating all the points 45° clockwise about the origin, such that the 45° line now coincides with the x-axis (Figure 24.2b). This is done by the simple transformation (see Equation (4.4))

$$(x^*, y^*) = (x, y)\begin{pmatrix}\cos 45^\circ & -\sin 45^\circ\\ \sin 45^\circ & \cos 45^\circ\end{pmatrix} = (x, y)\frac{1}{\sqrt{2}}\begin{pmatrix}1 & -1\\ 1 & 1\end{pmatrix} = (x, y)\mathbf{R}, \tag{24.1}$$

where the rotation matrix $\mathbf{R}$ is orthonormal (i.e., the dot product of a row with itself is 1, the dot product of different rows is 0, and the same is true for columns). The inverse transformation is

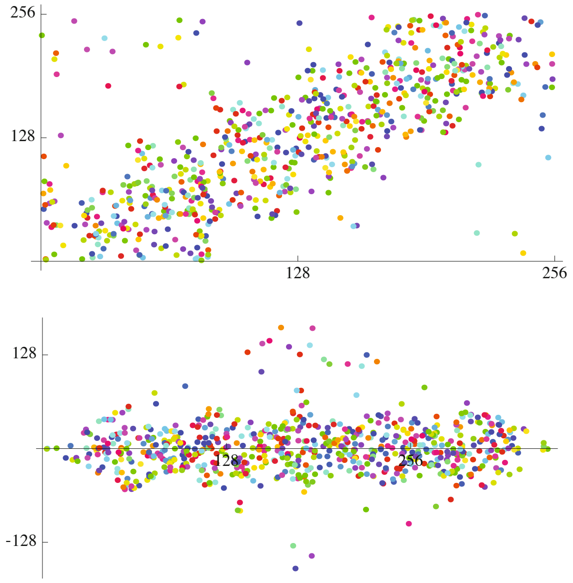
$$(x, y) = (x^*, y^*)\mathbf{R}^{-1} = (x^*, y^*)\mathbf{R}^T = (x^*, y^*)\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix}. \tag{24.2}$$

(The inverse of an orthonormal matrix is its transpose.)

It is obvious that most points end up with y coordinates that are zero or close to zero, while the x coordinates don't change much. Figure 24.3a,b shows the distributions of the x and y coordinates (i.e., the odd-numbered and even-numbered pixels) of the $128 \times 128 \times 8$ grayscale Lena image before the rotation. It is clear that the two distributions don't differ by much. Figure 24.3c,d shows that the distribution of the x coordinates stays about the same (with greater variance) but the y coordinates are concentrated around zero. The Matlab code that generated these results is also shown. (Figure 24.3d shows that the y coordinates are concentrated around 100, but this is because a few were as small as -101, so they had to be scaled by 101 to fit in a Matlab array, which always starts at index 1.)

```
p={{5,5},{6, 7},{12.1,13.2},{23,25},{32,29}};
rot={{0.7071,-0.7071},{0.7071,0.7071}};
Sum[p[[i,1]]p[[i,2]], {i,5}]
q=p.rot
Sum[q[[i,1]]q[[i,2]], {i,5}]
```

Figure 24.1: Code for Rotating Five Points.



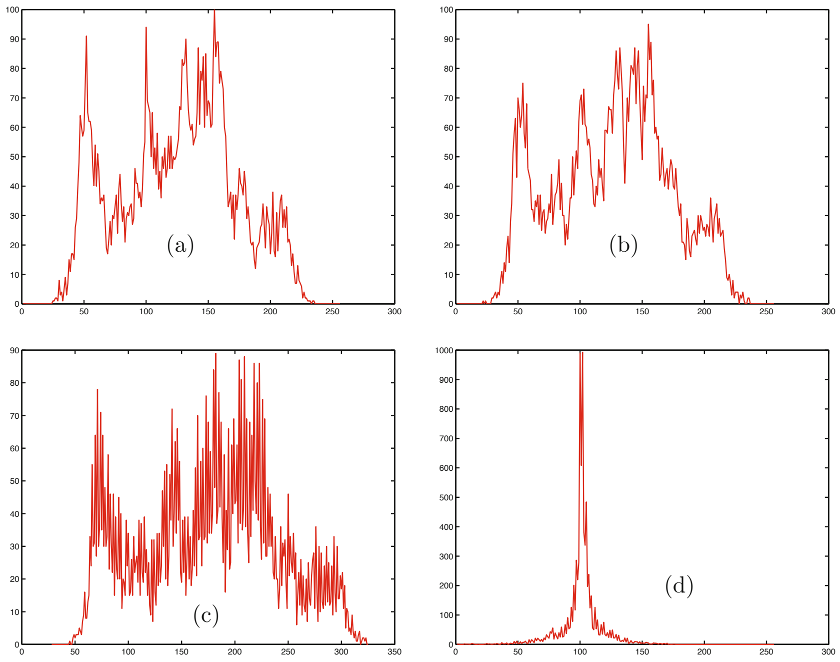
```
p=Table[Random[Real,{0,2}],{250}];
p=Flatten[Append[p,Table[Random[Real,{1,3}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{2,4}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{3,5}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{4,6}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{0,6}],{150}]]];
rot={{0.7071,-0.7071},{0.7071,0.7071}};
Graphics[Table[{Hue[RandomReal[]],Point[{p[[i]],p[[i+1]]}]},{i,1,1399,2}],
 Axes->True,AspectRatio->0.5,Ticks->{{{3,128},{6,256}},{{3,128},{6,256}}}]
Graphics[Table[{Hue[RandomReal[]],Point[{p[[i]],p[[i+1]]}.rot]},{i,1,1399,2}],
 Axes->True,AspectRatio->0.5,Ticks->{{{3,128},{6,256}},{{3,128},{-3,-128}}}]
```

Figure 24.2: Rotating a Cloud of Points.

Once the coordinates of points are known before and after the rotation, it is easy to measure the reduction in correlation. A simple measure is the sum $\sum_i x_i y_i$, also called the *cross-correlation* of points (x_i, y_i).

◇ **Exercise 24.1:** Given points $(5,5)$, $(6,7)$, $(12.1,13.2)$, $(23,25)$, and $(32,29)$, rotate them 45° clockwise and compute their cross-correlations before and after the rotation.

We can now compress the image by simply outputting the transformed pixels to become the compressed file. If lossy compression is acceptable, then all the pixels can be quantized (see [Salomon 09] for scalar and vector quantizations), resulting in even smaller numbers. We can also write all the odd-numbered pixels (those that make up the x coordinates of the pairs) on the compressed stream, followed by all the even-numbered pixels. These two sequences are called the *coefficient vectors* of the transform. The



```
filename='lena128'; dim=128;
xdist=zeros(256,1); ydist=zeros(256,1);
fid=fopen(filename,'r');
img=fread(fid,[dim,dim])';
for col=1:2:dim-1
 for row=1:dim
  x=img(row,col)+1; y=img(row,col+1)+1;
  xdist(x)=xdist(x)+1; ydist(y)=ydist(y)+1;
 end
end
figure(1), plot(xdist), colormap(gray) %dist of x&y values
figure(2), plot(ydist), colormap(gray) %before rotation
xdist=zeros(325,1); % clear arrays
ydist=zeros(256,1);
for col=1:2:dim-1
 for row=1:dim
  x=round((img(row,col)+img(row,col+1))*0.7071);
  y=round((-img(row,col)+img(row,col+1))*0.7071)+101;
  xdist(x)=xdist(x)+1; ydist(y)=ydist(y)+1;
 end
end
figure(3), plot(xdist), colormap(gray) %dist of x&y values
figure(4), plot(ydist), colormap(gray) %after rotation
```

Figure 24.3: Distribution of Image Pixels Before and After Rotation.

latter sequence consists of small numbers and may, after quantization, have runs of zeros, resulting in even better compression.

It can be shown that the total variance of the pixels does not change by the rotation, because a rotation matrix is orthonormal. However, since the variance of the new y coordinates is small, most of the variance is now concentrated in the x coordinates. The variance is sometimes called the *energy* of the distribution of pixels, so we can say that the rotation has concentrated (or compacted) the energy in the x coordinate and has created compression this way.

Concentrating the energy in one coordinate has another advantage. It makes it possible to quantize that coordinate more finely than the other coordinates. This type of quantization results in better (lossy) compression.

The following simple example illustrates the power of this basic transform. We start with the point $(4,5)$, whose two coordinates are similar. Using Equation (24.1) the point is transformed to $(4,5)\mathbf{R} = (9,1)/\sqrt{2} \approx (6.36396, 0.7071)$. The energies of the point and its transform are $4^2 + 5^2 = 41 = (9^2 + 1^2)/2$. If we delete the smaller coordinate (4) of the point, we end up with an error of $4^2/41 = 0.39$. If, on the other hand, we delete the smaller of the two transform coefficients (0.7071), the resulting error is just $0.7071^2/41 = 0.012$. Another way to obtain the same error is to consider the reconstructed point. Passing $\frac{1}{\sqrt{2}}(9,1)$ through the inverse transform (Equation (24.2)) results in the original point $(4,5)$. Doing the same with $\frac{1}{\sqrt{2}}(9,0)$ results in the approximate reconstructed point $(4.5, 4.5)$. The energy difference between the original and reconstructed points is the same small quantity

$$\frac{\left[(4^2+5^2)-(4.5^2+4.5^2)\right]}{4^2+5^2} = \frac{41-40.5}{41} = 0.0012.$$

This simple transform can easily be extended to any number of dimensions. Instead of selecting pairs of adjacent pixels we can select triplets. Each triplet becomes a point in three-dimensional space, and these points form a cloud concentrated around the line that forms equal (although not 45°) angles with the three coordinate axes. When this line is rotated such that it coincides with the x axis, the y and z coordinates of the transformed points become small numbers. The transformation is done by multiplying each point by a 3×3 rotation matrix, and such a matrix is, of course, orthonormal. The transformed points are then separated into three coefficient vectors, of which the last two consist of small numbers. For maximum compression each coefficient vector should be quantized separately.

This can be extended to more than three dimensions, with the only difference being that we cannot visualize spaces of dimensions higher than three. However, the mathematics can easily be extended. Some compression methods, such as JPEG, divide an image into blocks of 8×8 pixels each, and rotate first each row and then each column, by means of Equation (24.13), as shown in Section 24.3. This double rotation produces a set of 64 transformed values, of which the first—termed the "DC coefficient"—is large, and the other 63 (called the "AC coefficients") are normally small. Thus, this transform concentrates the energy in the first of 64 dimensions. The set of DC coefficients and each of the sets of 63 AC coefficients should, in principle, be quantized separately (JPEG does this a little differently, though; see Section 24.5.2).

24.2 Orthogonal Transforms

Image transforms are designed to have two properties: (1) reduce image redundancy by reducing the sizes of most pixels and (2) identify the less important parts of the image by isolating the various frequencies of the image. Thus, this section starts with a short discussion of frequencies. We intuitively associate a frequency with a wave. Water waves, sound waves, and electromagnetic waves have frequencies, but pixels in an image can also feature frequencies. Figure 24.4 shows a small, 5×8 bi-level image that illustrates this concept. The top row is uniform, so we can assign it zero frequency. The rows below it have increasing pixel frequencies as measured by the number of color changes along a row. The four waves on the right roughly correspond to the frequencies of the four top rows of the image.

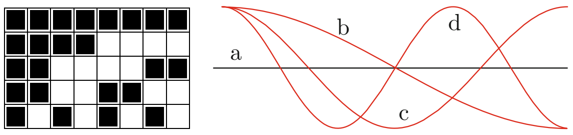


Figure 24.4: Image Frequencies.

Image frequencies are important because of the following basic fact: Low frequencies correspond to the important image features, while high frequencies correspond to the details of the image, which are less important. Thus, when a transform isolates the various image frequencies, transform coefficients that correspond to high frequencies can be quantized heavily, while transform coefficients that correspond to low frequencies should be quantized lightly or not at all. This is how a transform can compress an image very effectively by losing information, but only information associated with unimportant image details.

Practical image transforms should be fast and preferably also simple to implement. This suggests the use of *linear transforms*. In such a transform, each transformed value (or transform coefficient) c_i is a weighted sum of the data items (the pixels) d_j that are being transformed, where each item is multiplied by a weight w_{ij}. Thus, $c_i = \sum_j d_j w_{ij}$, for $i, j = 1, 2, \ldots, n$. For $n = 4$, this is expressed in matrix notation as follows:

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix}.$$

In the general case, we can write $\mathbf{C} = \mathbf{W}\cdot\mathbf{D}$. Each row of $\mathbf{W}$ is called a "basis vector."

The only quantities that have to be determined are the weights w_{ij}. The guiding principles for determining them are as follows:

1. Reducing redundancy. The first transform coefficient c_1 can be large, but the remaining values c_2, $c_3, \ldots$ should be small.

2. Isolating frequencies. The first transform coefficient c_1 should correspond to zero pixel frequency, and the remaining coefficients should correspond to higher and higher frequencies.

The key to determining the weights w_{ij} is the fact that our data items d_j are not arbitrary numbers but pixel values, which are nonnegative and correlated.

The basic relation $c_i = \sum_j d_j w_{ij}$ suggests that the first coefficient c_1 will be large if all the weights of the form w_{1j} are positive. To make the other coefficients c_i small, it is enough to make half the weights w_{ij} positive and the other half negative. A simple choice is to assign half the weights the value $+1$ and the other half the value -1. In the extreme case where all the pixels d_j are identical, this will result in $c_i = 0$. When the d_j's are similar, c_i will be small (positive or negative).

This choice of w_{ij} satisfies the first requirement: to reduce pixel redundancy by means of a transform. In order to satisfy the second requirement, the weights w_{ij} of row i should feature frequencies that get higher with i. Weights w_{1j} should have zero frequency; they should all be $+1$'s. Weights w_{2j} should have one sign change; i.e., they should be $+1, +1, \ldots + 1, -1, -1, \ldots, -1$. This continues until the last row of weights w_{nj} should have the highest frequency $+1, -1, +1, -1, \ldots, +1, -1$. The mathematical discipline of vector spaces coins the term "basis vectors" for our rows of weights.

In addition to isolating the various frequencies of pixels d_j, this choice results in basis vectors that are orthogonal. The basis vectors are the rows of matrix $\mathbf{W}$, which is why this matrix—and by implication, the entire transform—are also termed orthogonal.

These considerations are satisfied by the orthogonal matrix

$$\mathbf{W} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}. \tag{24.3}$$

The first basis vector (the top row of $\mathbf{W}$) consists of all 1's, so its frequency is zero. Each of the subsequent vectors has two $+1$'s and two -1's, so they produce small transformed values, and their frequencies (measured as the number of sign changes along the basis vector) get higher. This matrix is similar to the Walsh–Hadamard transform (Equation (24.4)).

To illustrate how this matrix identifies the frequencies in a data vector, we multiply it by four vectors as follows:

$$\mathbf{W}\cdot\begin{bmatrix}1\\0\\0\\1\end{bmatrix} = \begin{bmatrix}2\\0\\2\\0\end{bmatrix}, \quad \mathbf{W}\cdot\begin{bmatrix}0\\0.33\\-0.33\\-1\end{bmatrix} = \begin{bmatrix}0\\2.66\\0\\1.33\end{bmatrix}, \quad \mathbf{W}\cdot\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\1\\1\\1\end{bmatrix}, \quad \mathbf{W}\cdot\begin{bmatrix}1\\-0.8\\1\\-0.8\end{bmatrix} = \begin{bmatrix}0.4\\0\\0\\3.6\end{bmatrix}.$$

The results make sense when we discover how the four test vectors were determined

$$\begin{gathered}(1,0,0,1) = 0.5(1,1,1,1) + 0.5(1,-1,-1,1),\\ (1,0.33,-0.33,-1) = 0.66(1,1,-1,-1) + 0.33(1,-1,1,-1),\\ (1,0,0,0) = 0.25(1,1,1,1) + 0.25(1,1,-1,-1) + 0.25(1,-1,-1,1) + 0.25(1,-1,1,-1),\\ (1,-0.8,1,-0.8) = 0.1(1,1,1,1) + 0.9(1,-1,1,-1).\end{gathered}$$

The product of $\mathbf{W}$ and the first vector shows how that vector consists of equal amounts of the first and the third frequencies. Similarly, the transform $(0.4, 0, 0, 3.6)$ shows that vector $(1, -0.8, 1, -0.8)$ is a mixture of a small amount of the first frequency and nine times the fourth frequency.

It is also possible to modify this transform to conserve the energy of the data vector. All that's needed is to multiply the transformation matrix $\mathbf{W}$ by the scale factor 1/2. Thus, the product $(\mathbf{W}/2)\times(a, b, c, d)$ has the same energy $a^2 + b^2 + c^2 + d^2$ as the data vector (a, b, c, d). An example is the product of $\mathbf{W}/2$ and the correlated vector $(5, 6, 7, 8)$. It results in the transform coefficients $(13, -2, 0, -1)$, where the first coefficient is large and the remaining ones are smaller than the original data items. The energy of both $(5, 6, 7, 8)$ and $(13, -2, 0, -1)$ is 174, but whereas in the former vector the first component accounts for only 14% of the energy, in the transformed vector the first component accounts for 97% of the energy. This is how our simple orthogonal transform compacts the energy of the data vector.

Another advantage of $\mathbf{W}$ is that it also performs the inverse transform. The product $(\mathbf{W}/2)\cdot(13, -2, 0, -1)^T$ reconstructs the original data $(5, 6, 7, 8)$.

We are now in a position to appreciate the compression potential of this transform. We use matrix $\mathbf{W}/2$ to transform the (not very correlated) data vector $d = (4, 6, 5, 2)$. The result is $t = (8.5, 1.5, -2.5, 0.5)$. It's easy to transform t back to d, but t itself does not provide any compression. In order to achieve compression, we quantize the components of t, and the point is that even after heavy quantization, it is still possible to get back a vector very similar to the original d.

We first quantize t to the integers $(9, 1, -3, 0)$ and perform the inverse transform to get back $(3.5, 6.5, 5.5, 2.5)$. In a similar experiment, we completely delete the two smallest elements and inverse-transform the coarsely quantized vector $(8.5, 0, -2.5, 0)$. This produces the reconstructed data $(3, 5.5, 5.5, 3)$, still very close to the original values of d. The conclusion is that even this simple, intuitive transform is a powerful tool for "squeezing out" the redundancy in pixel data. More sophisticated transforms produce results that can be quantized coarsely and still be used to reconstruct the original data to a high degree.

24.2.1 Two-Dimensional Transforms

Given two-dimensional data such as the 4×4 matrix

$$\mathbf{D} = \begin{pmatrix} 5 & 6 & 7 & 4 \\ 6 & 5 & 7 & 5 \\ 7 & 7 & 6 & 6 \\ 8 & 8 & 8 & 8 \end{pmatrix},$$

where each of the four columns is highly correlated, we can apply our simple one-dimensional transform to the columns of $\mathbf{D}$. The result is

$$\mathbf{C}' = \mathbf{W}\cdot\mathbf{D} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}\cdot\mathbf{D} = \begin{pmatrix} 26 & 26 & 28 & 23 \\ -4 & -4 & 0 & -5 \\ 0 & 2 & 2 & 1 \\ -2 & 0 & -2 & -3 \end{pmatrix}.$$

Each column of $\mathbf{C}'$ is the transform of a column of $\mathbf{D}$. Notice how the top element of each column of $\mathbf{C}'$ is dominant, because the data in the corresponding column of $\mathbf{D}$ is correlated. Notice also that the rows of $\mathbf{C}'$ are still correlated. $\mathbf{C}'$ is the first stage in a two-stage process that produces the two-dimensional transform of matrix $\mathbf{D}$. The second stage should transform each *row* of $\mathbf{C}'$, and this is done by multiplying $\mathbf{C}'$ by the transpose $\mathbf{W}^T$. Our particular $\mathbf{W}$, however, is symmetric, so we end up with $\mathbf{C} = \mathbf{C}'\!\cdot\!\mathbf{W}^T = \mathbf{W}\!\cdot\!\mathbf{D}\!\cdot\!\mathbf{W}^T = \mathbf{W}\!\cdot\!\mathbf{D}\!\cdot\!\mathbf{W}$ or

$$\mathbf{C} = \begin{pmatrix} 26 & 26 & 28 & 23 \\ -4 & -4 & 0 & -5 \\ 0 & 2 & 2 & 1 \\ -2 & 0 & -2 & -3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 103 & 1 & -5 & 5 \\ -13 & -3 & -5 & 5 \\ 5 & -1 & -3 & -1 \\ -7 & 3 & -3 & -1 \end{pmatrix}.$$

The elements of $\mathbf{C}$ are decorrelated. The top-left element is dominant. It contains most of the total energy of the original $\mathbf{D}$. The elements in the top row and the leftmost column are somewhat large, while the remaining elements are smaller than the original data items. The double-stage, two-dimensional transformation has reduced the correlation in both the horizontal and vertical dimensions. As in the one-dimensional case, excellent compression can be achieved by quantizing the elements of $\mathbf{C}$, especially those that correspond to higher frequencies (i.e., located toward the bottom-right corner of $\mathbf{C}$).

This is the essence of orthogonal transforms. The next few sections discuss the following important transforms:

1. The Walsh–Hadamard transform (WHT, Section 24.2.2) is fast and easy to compute (it requires only additions and subtractions), but its performance, in terms of energy compaction, is lower than that of the DCT.
2. The Haar transform [Stollnitz et al. 96] is a simple, fast transform. It is the simplest wavelet transform and is discussed in Section 24.2.3 and in Chapter 25.
3. The Karhunen–Loève transform (KLT, Section 24.2.4) is the best one theoretically, in the sense of energy compaction (or, equivalently, pixel decorrelation). However, its coefficients are not fixed; they depend on the data to be compressed. Calculating these coefficients (the basis of the transform) is slow, as is the calculation of the transformed values themselves. Since the coefficients are data dependent, they have to be included in the compressed stream. For these reasons and because the DCT performs almost as well, the KLT is not generally used in practice.
4. The discrete cosine transform (DCT) is discussed in detail in Section 24.3. This important transform is almost as efficient as the KLT in terms of energy compaction, but it uses a fixed basis, independent of the data. There are also fast methods for computing the DCT. This method is used by JPEG and MPEG audio.

24.2.2 Walsh–Hadamard Transform

As mentioned earlier, this transform has low compression efficiency, which is why it is not used much in practice. It is, however, fast, because it can be computed with just additions, subtractions, and an occasional right shift (to replace a division by a power of 2).

Given an $N \times N$ block of pixels P_{xy} (where N must be a power of 2, $N = 2^n$), its two-dimensional WHT and inverse WHT are defined by Equations (24.4) and (24.5):

$$\begin{aligned} H(u,v) &= \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} p_{xy} g(x,y,u,v) \\ &= \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} p_{xy}(-1)^{\sum_{i=0}^{n-1}[b_i(x)p_i(u)+b_i(y)p_i(v)]}, \end{aligned} \tag{24.4}$$

$$\begin{aligned} P_{xy} &= \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} H(u,v) h(x,y,u,v) \\ &= \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} H(u,v)(-1)^{\sum_{i=0}^{n-1}[b_i(x)p_i(u)+b_i(y)p_i(v)]}, \end{aligned} \tag{24.5}$$

where $H(u,v)$ are the results of the transform (i.e., the WHT coefficients), the quantity $b_i(u)$ is bit i of the binary representation of the integer u, and $p_i(u)$ is defined in terms of the $b_j(u)$ by Equation (24.6):

$$\begin{aligned} p_0(u) &= b_{n-1}(u), \\ p_1(u) &= b_{n-1}(u) + b_{n-2}(u), \\ p_2(u) &= b_{n-2}(u) + b_{n-3}(u), \\ &\vdots \\ p_{n-1}(u) &= b_1(u) + b_0(u). \end{aligned} \tag{24.6}$$

(Recall that n is defined above by $N = 2^n$.) As an example, consider $u = 6 = 110_2$. Bits zero, one, and two of 6 are 0, 1, and 1, respectively, so $b_0(6) = 0$, $b_1(6) = 1$, and $b_2(6) = 1$.

The quantities $g(x,y,u,v)$ and $h(x,y,u,v)$ are called the *kernels* (or *basis images*) of the WHT. These matrices are identical. Their elements are just $+1$ and -1, and they are multiplied by the factor $\frac{1}{N}$. As a result, the WHT transform consists in multiplying each image pixel by $+1$ or -1, summing, and dividing the sum by N. Since $N = 2^n$ is a power of 2, dividing by it can be done by shifting n positions to the right.

The WHT kernels are shown, in graphical form, for $N = 4$, in Figure 24.5, where white denotes $+1$ and black denotes -1 (the factor $\frac{1}{N}$ is ignored). The rows and columns of blocks in this figure correspond to values of u and v from 0 to 3, respectively. The rows and columns inside each block correspond to values of x and y from 0 to 3, respectively. The number of sign changes across a row or a column of a matrix is called the *sequency* of the row or column. The rows and columns in the figure are ordered in increased sequency. Some authors show similar but unordered figures, because this transform was defined by Walsh and by Hadamard in slightly different ways (see [Gonzalez and Woods 92] for more information).

Compressing an image with the WHT is done similarly to the DCT, except that Equations (24.4) and (24.5) are used, instead of Equations (24.13) and (24.14).

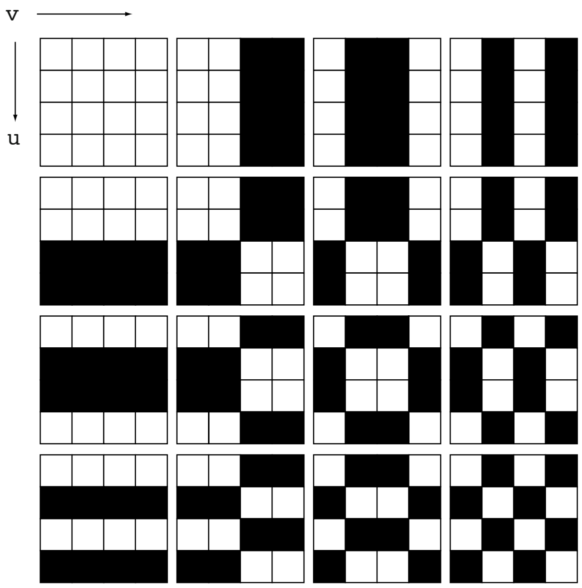


Figure 24.5: The Ordered WHT Kernel for $N = 4$.

◇ **Exercise 24.2:** Use appropriate mathematical software to compute and display the basis images of the WHT for $N = 8$.

24.2.3 Haar Transform

The Haar transform [Stollnitz et al. 96] is based on the Haar functions $h_k(x)$, which are defined for $x \in [0, 1]$ and for $k = 0, 1, \ldots, N-1$, where $N = 2^n$. Its application is also discussed in Chapter 25.

Before we discuss the actual transform, we have to mention that any integer k can be expressed as the sum $k = 2^p + q - 1$, where $0 \le p \le n-1$, $q = 0$ or 1 for $p = 0$, and $1 \le q \le 2^p$ for $p \ne 0$. For $N = 4 = 2^2$, for example, we get $0 = 2^0 + 0 - 1$, $1 = 2^0 + 1 - 1$, $2 = 2^1 + 1 - 1$, and $3 = 2^1 + 2 - 1$.

The Haar basis functions are now defined as

$$h_0(x) \stackrel{\text{def}}{=} h_{00}(x) = \frac{1}{\sqrt{N}}, \quad \text{for } 0 \le x \le 1, \tag{24.7}$$

and

$$h_k(x) \stackrel{\text{def}}{=} h_{pq}(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2}, & \frac{q-1}{2^p} \le x < \frac{q-1/2}{2^p}, \\ -2^{p/2}, & \frac{q-1/2}{2^p} \le x < \frac{q}{2^p}, \\ 0, & \text{otherwise for } x \in [0,1]. \end{cases} \tag{24.8}$$

The Haar transform matrix $\mathbf{A}_N$ of order $N \times N$ can now be constructed. A general element i, j of this matrix is the basis function $h_i(j)$, where $i = 0, 1, \ldots, N-1$ and $j = 0/N, 1/N, \ldots, (N-1)/N$. For example,

$$\mathbf{A}_2 = \begin{pmatrix} h_0(0/2) & h_0(1/2) \\ h_1(0/2) & h_1(1/2) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{24.9}$$

(recall that $i = 1$ implies $p = 0$ and $q = 1$). Figure 24.6 shows code to calculate this matrix for any N, and also the Haar basis images for $N = 8$.

◇ **Exercise 24.3:** Compute the Haar coefficient matrices $\mathbf{A}_4$ and $\mathbf{A}_8$.

Given an image block $\mathbf{P}$ of order $N \times N$ where $N = 2^n$, its Haar transform is the matrix product $\mathbf{A}_N \mathbf{P} \mathbf{A}_N$ (Section 25.1).

24.2.4 Karhunen–Loève Transform

The Karhunen–Loève transform (also called the Hotelling transform) has the best efficiency in the sense of energy compaction, but for the reasons mentioned earlier, it has more theoretical than practical value. Given an image, we break it up into k blocks of n pixels each, where n is typically 64 but can have other values, and k depends on the image size. We consider the blocks vectors and denote them by $\mathbf{b}^{(i)}$, for $i = 1, 2, \ldots, k$. The average vector is $\overline{\mathbf{b}} = (\sum_i \mathbf{b}^{(i)})/k$. A new set of vectors $\mathbf{v}^{(i)} = \mathbf{b}^{(i)} - \overline{\mathbf{b}}$ is defined, causing the average $(\sum \mathbf{v}^{(i)})/k$ to be zero. We denote the $n \times n$ KLT transform matrix that we are seeking by $\mathbf{A}$. The result of transforming a vector $\mathbf{v}^{(i)}$ is the weight vector $\mathbf{w}^{(i)} = \mathbf{A}\mathbf{v}^{(i)}$. The average of the $\mathbf{w}^{(i)}$ is also zero. We now construct a matrix $\mathbf{V}$ whose columns are the $\mathbf{v}^{(i)}$ vectors and another matrix $\mathbf{W}$ whose columns are the weight vectors $\mathbf{w}^{(i)}$:

$$\mathbf{V} = \left(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(k)}\right), \quad \mathbf{W} = \left(\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(k)}\right).$$

Matrices $\mathbf{V}$ and $\mathbf{W}$ have n rows and k columns each. From the definition of $\mathbf{w}^{(i)}$, we get $\mathbf{W} = \mathbf{A} \cdot \mathbf{V}$.

The n coefficient vectors $\mathbf{c}^{(j)}$ of the Karhunen–Loève transform are given by

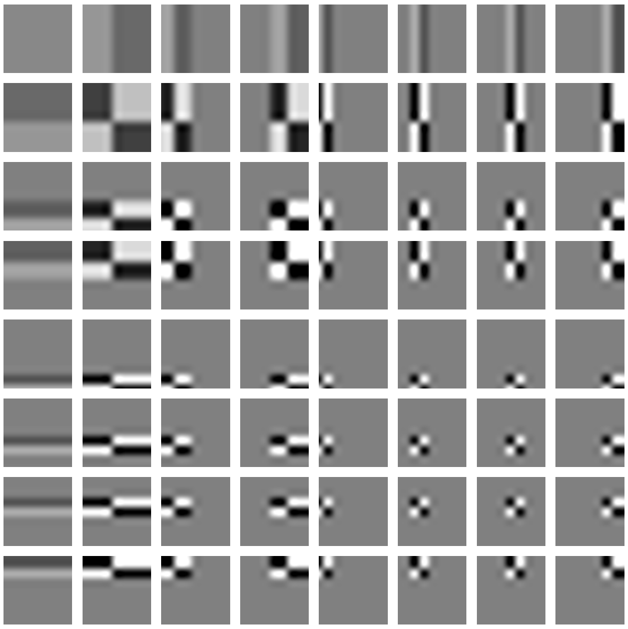
$$\mathbf{c}^{(j)} = \left(w_j^{(1)}, w_j^{(2)}, \ldots, w_j^{(k)}\right), \quad j = 1, 2, \ldots, n.$$

Thus, vector $\mathbf{c}^{(j)}$ consists of the jth elements of all the weight vectors $\mathbf{w}^{(i)}$, for $i = 1, 2, \ldots, k$ ($\mathbf{c}^{(j)}$ is the jth coordinate of the $\mathbf{w}^{(i)}$ vectors).

We now examine the elements of the matrix product $\mathbf{W} \cdot \mathbf{W}^T$ (this is an $n \times n$ matrix). A general element in row a and column b of this matrix is the sum of products:

$$\left(\mathbf{W} \cdot \mathbf{W}^T\right)_{ab} = \sum_{i=1}^{k} w_a^{(i)} w_b^{(i)} = \sum_{i=1}^{k} c_i^{(a)} c_i^{(b)} = \mathbf{c}^{(a)} \bullet \mathbf{c}^{(b)}, \quad \text{for } a, b \in [1, n]. \tag{24.10}$$

The fact that the average of each $\mathbf{w}^{(i)}$ is zero implies that a general diagonal element $(\mathbf{W} \cdot \mathbf{W}^T)_{jj}$ of the product matrix is the variance (up to a factor k) of the jth element



```
Needs["GraphicsImage`"] (* Draws 2D Haar Coefficients *)
n=8;
h[k_,x_]:=Module[{p,q}, If[k==0, 1/Sqrt[n],           (* h_0(x) *)
 p=0; While[2^p<=k ,p++]; p--; q=k-2^p+1; (* if k>0, calc. p, q *)
 If[(q-1)/(2^p)<=x && x<(q-.5)/(2^p),2^(p/2),
  If[(q-.5)/(2^p)<=x && x<q/(2^p),-2^(p/2),0]]]];
HaarMatrix=Table[h[k,x], {k,0,7}, {x,0,7/n,1/n}] //N;
HaarTensor=Array[Outer[Times, HaarMatrix[[#1]],HaarMatrix[[#2]]]&,
 {n,n}];
Show[GraphicsArray[Map[GraphicsImage[#, {-2,2}]&, HaarTensor,{2}]]]
```

Figure 24.6: The Basis Images of the Haar Transform for $n = 8$.

(or jth coordinate) of the $\mathbf{w}^{(i)}$ vectors. This, of course, is the variance of coefficient vector $\mathbf{c}^{(j)}$.

◇ **Exercise 24.4:** Explain why this is true.

The off-diagonal elements of $(\mathbf{W}\cdot\mathbf{W}^T)$ are the covariances of the $\mathbf{w}^{(i)}$ vectors such that element $\left(\mathbf{W}\cdot\mathbf{W}^T\right)_{ab}$ is the covariance of the ath and bth coordinates of the $\mathbf{w}^{(i)}$'s. Equation (24.10) shows that this is also the dot product $\mathbf{c}^{(a)}\cdot\mathbf{c}^{(b)}$. One of the main aims of image transform is to decorrelate the coordinates of the vectors, and probability theory tells us that two coordinates are decorrelated if their covariance is zero (the other aim is energy compaction, but the two goals go hand in hand). Thus, our aim is to find a transformation matrix $\mathbf{A}$ such that the product $\mathbf{W}\cdot\mathbf{W}^T$ will be diagonal.

From the definition of matrix $\mathbf{W}$ we get

$$\mathbf{W}\cdot\mathbf{W}^T = (\mathbf{AV})\cdot(\mathbf{AV})^T = \mathbf{A}(\mathbf{V}\cdot\mathbf{V}^T)\mathbf{A}^T.$$

Matrix $\mathbf{V}\cdot\mathbf{V}^T$ is symmetric, and its elements are the covariances of the coordinates of vectors $\mathbf{v}^{(i)}$, i.e.,

$$\left(\mathbf{V}\cdot\mathbf{V}^T\right)_{ab} = \sum_{i=1}^{k} v_a^{(i)} v_b^{(i)}, \quad \text{for } a, b \in [1, n].$$

Since $\mathbf{V}\cdot\mathbf{V}^T$ is symmetric, its eigenvectors are orthogonal. We therefore normalize these vectors (i.e., make them orthonormal) and choose them as the rows of matrix $\mathbf{A}$. This produces the result

$$\mathbf{W}\cdot\mathbf{W}^T = \mathbf{A}(\mathbf{V}\cdot\mathbf{V}^T)\mathbf{A}^T = \begin{pmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_n \end{pmatrix}.$$

This choice of $\mathbf{A}$ results in a diagonal matrix $\mathbf{W}\cdot\mathbf{W}^T$ whose diagonal elements are the eigenvalues of $\mathbf{V}\cdot\mathbf{V}^T$. Matrix $\mathbf{A}$ is the Karhunen–Loève transformation matrix; its rows are the basis vectors of the KLT, and the energies (variances) of the transformed vectors are the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of $\mathbf{V}\cdot\mathbf{V}^T$.

The basis vectors of the KLT are calculated from the original image pixels and are, therefore, data dependent. In a practical compression method, these vectors have to be included in the compressed stream, for the decoder's use, and this, combined with the fact that no fast method has been discovered for the calculation of the KLT, makes this transform less than ideal for practical applications.

24.3 The Discrete Cosine Transform

This important transform (DCT for short) was originated by [Ahmed et al. 74] and has been used and studied extensively since. Because of its importance for data compression, the DCT is treated here in detail. Section 24.3.1 introduces the mathematical expressions for the DCT in one dimension and two dimensions without any theoretical background or justifications. The use of the transform and its advantages for data compression are then demonstrated by several examples. Sections 24.3.2 and 24.3.3 cover the theory of the DCT and discuss its two interpretations as a rotation and as a basis of a vector space. Section 24.3.4 introduces the four DCT types, and Section 11.15.2 of [Salomon 09] discusses the three-dimensional DCT. Section 24.3.5 describes ways to speed up the computation of the DCT, and Section 24.3.7 is a short discussion of the symmetry of the DCT and how it can be exploited for a hardware implementation.

Several sections of important background material follow. Section 24.3.8 explains the QR decomposition of matrices. Section 24.3.9 introduces the concept of vector spaces and their bases. Section 24.3.10 shows how the rotation performed by the DCT relates to general rotations in three dimensions. Finally, the discrete sine transform is introduced in Section 24.3.11 together with the reasons that make it unsuitable for data compression.

For more information on this important transform, see [Ahmed et al. 74], [Rao and Yip 90], and [Britanak et al. 06].

24.3.1 Introduction

The DCT in one dimension is given by

$$G_f = \sqrt{\frac{2}{n}} C_f \sum_{t=0}^{n-1} p_t \cos\left[\frac{(2t+1)f\pi}{2n}\right], \tag{24.11}$$

where

$$C_f = \begin{cases} \frac{1}{\sqrt{2}}, & f = 0, \\ 1, & f > 0, \end{cases} \quad \text{for } f = 0, 1, \ldots, n-1.$$

The input is a set of n data values p_t (pixels, audio samples, or other data), and the output is a set of n DCT *transform coefficients* (or weights) G_f. The first coefficient G_0 is called the DC coefficient, and the rest are referred to as the AC coefficients (these terms have been inherited from electrical engineering, where they stand for "direct current" and "alternating current"). Notice that the coefficients are real numbers even if the input data consists of integers. Similarly, the coefficients may be positive or negative even if the input data consists of nonnegative numbers only. This computation is straightforward but slow (Section 24.3.5 discusses faster versions). The decoder inputs the DCT coefficients in sets of n and uses the *inverse* DCT (IDCT) to reconstruct the original data values (also in groups of n). The IDCT in one dimension is given by

$$p_t = \sqrt{\frac{2}{n}} \sum_{j=0}^{n-1} C_j G_j \cos\left[\frac{(2t+1)j\pi}{2n}\right], \quad \text{for } t = 0, 1, \ldots, n-1. \tag{24.12}$$

The important feature of the DCT, the feature that makes it so useful in data compression, is that it takes correlated input data and concentrates its energy in just the first few transform coefficients. If the input data consists of correlated quantities, then most of the n transform coefficients produced by the DCT are zeros or small numbers, and only a few are large (normally the first ones). We will see that the early coefficients contain the important (low-frequency) image information and the later coefficients contain the less-important (high-frequency) image information. Compressing data with the DCT is therefore done by quantizing the coefficients. The small ones are quantized coarsely (possibly all the way to zero), and the large ones can be quantized finely to the nearest integer. After quantization, the coefficients (or variable-length codes assigned to the coefficients) are written on the compressed stream. Decompression is done by performing the inverse DCT on the quantized coefficients. This results in data items that are not identical to the original ones but are not much different.

In practical applications, the data to be compressed is partitioned into sets of n items each and each set is DCT-transformed and quantized individually. The value of n is critical. Small values of n such as 3, 4, or 6 result in many small sets of data items. Such a small set is transformed to a small set of coefficients where the energy of the original data is concentrated in a few coefficients, but there are only a few coefficients in such a set! Thus, there are not enough small coefficients to quantize. Large values of n result in a few large sets of data. The problem in such a case is that the individual data items of a large set are normally not correlated and therefore result in a set of transform coefficients where all the coefficients are large. Experience indicates that $n = 8$ is a good value, and most data compression methods that employ the DCT use this value of n.

The following experiment illustrates the power of the DCT in one dimension. We start with the set of eight correlated data items $\mathbf{p} = (12, 10, 8, 10, 12, 10, 8, 11)$, apply the DCT in one dimension to them, and find that it results in the eight coefficients

$$28.6375,\ 0.571202,\ 0.46194,\ 1.757,\ 3.18198,\ -1.72956,\ 0.191342,\ -0.308709.$$

These can be fed to the IDCT and transformed by it to precisely reconstruct the original data (except for small errors caused by limited machine precision). Our goal, however, is to compress the data by quantizing the coefficients. We first quantize them to $28.6, 0.6, 0.5, 1.8, 3.2, -1.8, 0.2, -0.3$, and apply the IDCT to get back

$$12.0254,\ 10.0233,\ 7.96054,\ 9.93097,\ 12.0164,\ 9.99321,\ 7.94354,\ 10.9989.$$

We then quantize the coefficients even more, to $28, 1, 1, 2, 3, -2, 0, 0$, and apply the IDCT to get back

$$12.1883,\ 10.2315,\ 7.74931,\ 9.20863,\ 11.7876,\ 9.54549,\ 7.82865,\ 10.6557.$$

Finally, we quantize the coefficients to $28, 0, 0, 2, 3, -2, 0, 0$, and still get back from the IDCT the sequence

$$11.236,\ 9.62443,\ 7.66286,\ 9.57302,\ 12.3471,\ 10.0146,\ 8.05304,\ 10.6842,$$

where the largest difference between an original value (12) and a reconstructed one (11.236) is 0.764 (or 6.4% of 12). The code that does all that is listed in Figure 24.7.

```
n=8;
p={12.,10.,8.,10.,12.,10.,8.,11.};
c=Table[If[t==1, 0.7071, 1], {t,1,n}];
dct[i_]:=Sqrt[2/n]c[[i+1]]Sum[p[[t+1]]Cos[(2t+1)i Pi/16],{t,0,n-1}];
q=Table[dct[i],{i,0,n-1}] (* use exact DCT coefficients *)
q={28,0,0,2,3,-2,0,0}; (* or use quantized DCT coefficients *)
idct[t_]:=Sqrt[2/n]Sum[c[[j+1]]q[[j+1]]Cos[(2t+1)j Pi/16],{j,0,n-1}];
ip=Table[idct[t],{t,0,n-1}]
```

Figure 24.7: Experiments with the One-Dimensional DCT.

It seems magical that the eight original data items can be reconstructed to such high precision from just four transform coefficients. The explanation, however, relies on the following arguments instead of on magic: (1) The IDCT is given all eight transform coefficients, so it knows the positions, not just the values, of the nonzero coefficients. (2) The first few coefficients (the large ones) contain the important information of the original data items. The small coefficients, the ones that are quantized heavily, contain less important information (in the case of images, they contain the image details). (3) The original data is redundant because of pixel correlation.

The following experiment illustrates the performance of the DCT when applied to decorrelated data items. Given the eight decorrelated data items −12, 24, −181, 209, 57.8, 3, −184, and −250, their DCT produces

$$-117.803,\ 166.823,\ -240.83,\ 126.887,\ 121.198,\ 9.02198,\ -109.496,\ -185.206.$$

When these coefficients are quantized to $(-120., 170., -240., 125., 120., 9., -110., -185)$ and fed into the IDCT, the result is

$$-12.1249,\ 25.4974,\ -179.852,\ 208.237,\ 55.5898,\ 0.364874,\ -185.42,\ -251.701,$$

where the maximum difference (between 3 and 0.364874) is 2.63513 or 88% of 3. Obviously, even with such fine quantization the reconstruction is not as good as with correlated data.

◇ **Exercise 24.5:** Compute the one-dimensional DCT (Equation (24.11)) of the eight correlated values 11, 22, 33, 44, 55, 66, 77, and 88. Show how to quantize them, and compute their IDCT from Equation (24.12).

An important relative of the DCT is the Fourier transform, discussed in any text on digital signal processing, which also has a discrete version termed the DFT. The DFT has important applications, but it does not perform well in data compression because it assumes that the data to be transformed is periodic.

The following example illustrates the difference in performance between the DCT and the DFT. We start with the simple, highly-correlated sequence of eight numbers $(8, 16, 24, 32, 40, 48, 56, 64)$. It is displayed graphically in Figure 24.8a. Applying the DCT to it yields $(100, -52, 0, -5, 0, -2, 0, 0.4)$. When this is quantized to $(100, -52, 0, -5, 0, 0, 0, 0)$ and transformed back, it produces $(8, 15, 24, 32, 40, 48, 57, 63)$, a sequence almost identical to the original input. Applying the DFT to the same input, on the other hand, yields $(36, 10, 10, 6, 6, 4, 4, 4)$. When this is quantized to $(36, 10, 10, 6, 0, 0, 0, 0)$ and is transformed back, it produces $(24, 12, 20, 32, 40, 51, 59, 48)$. This output is shown in Figure 24.8b, and it illustrates the tendency of the Fourier transform to produce a periodic result.

The DCT in one dimension can be used to compress one-dimensional data, such as audio samples. This chapter, however, discusses image compression which is based on the two-dimensional correlation of pixels (a pixel tends to resemble all its near neighbors, not just those in its row). This is why practical image compression methods use the DCT in two dimensions. This version of the DCT is applied to small parts (data blocks) of the image. It is computed by applying the DCT in one dimension to each row of

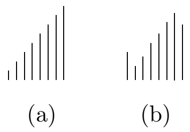


Figure 24.8: (a) One-Dimensional Input. (b) Its Inverse DFT.

a data block, and then to each column of the result. Because of the special way the DCT in two dimensions is computed, we say that it is separable in the two dimensions. Because it is applied to blocks of an image, we term it a "blocked transform." It is defined by

$$G_{ij}=\sqrt{\frac{2}{m}}\sqrt{\frac{2}{n}}C_iC_j\sum_{x=0}^{n-1}\sum_{y=0}^{m-1}p_{xy}\cos\left[\frac{(2y+1)j\pi}{2m}\right]\cos\left[\frac{(2x+1)i\pi}{2n}\right], \qquad (24.13)$$

for $0\le i\le n-1$ and $0\le j\le m-1$ and for C_i and C_j defined by Equation (24.11). The first coefficient G_{00} is again termed the "DC coefficient," and the remaining coefficients are called the "AC coefficients."

The image is broken up into blocks of $n\times m$ pixels p_{xy} (with $n=m=8$ typically), and Equation (24.13) is used to produce a block of $n\times m$ DCT coefficients G_{ij} for each block of pixels. The coefficients are then quantized, which results in lossy but highly efficient compression. The decoder reconstructs a block of quantized data values by computing the IDCT whose definition is

$$p_{xy}=\sqrt{\frac{2}{m}}\sqrt{\frac{2}{n}}\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}C_iC_jG_{ij}\cos\left[\frac{(2x+1)i\pi}{2n}\right]\cos\left[\frac{(2y+1)j\pi}{2m}\right], \quad (24.14)$$

where
$$C_f=\begin{cases}\frac{1}{\sqrt{2}}, & f=0\\ 1\ , & f>0,\end{cases}$$

for $0\le x\le n-1$ and $0\le y\le m-1$. We now show one way to compress an entire image with the DCT in several steps as follows:

1. The image is divided into k blocks of 8×8 pixels each. The pixels are denoted by p_{xy}. If the number of image rows (columns) is not divisible by 8, the bottom row (rightmost column) is duplicated as many times as needed.

2. The DCT in two dimensions (Equation (24.13)) is applied to each block B_i. The result is a block (we'll call it a vector) $W^{(i)}$ of 64 transform coefficients $w_j^{(i)}$ (where $j=0,1,\ldots,63$). The k vectors $W^{(i)}$ become the rows of matrix $\mathbf{W}$

$$\mathbf{W}=\begin{bmatrix} w_0^{(1)} & w_1^{(1)} & \ldots & w_{63}^{(1)}\\ w_0^{(2)} & w_1^{(2)} & \ldots & w_{63}^{(2)}\\ \vdots & \vdots & & \\ w_0^{(k)} & w_1^{(k)} & \ldots & w_{63}^{(k)}\end{bmatrix}.$$

3. The 64 columns of $\mathbf{W}$ are denoted by $C^{(0)}$, $C^{(1)}$, ..., $C^{(63)}$. The k elements of $C^{(j)}$ are $\left(w_j^{(1)}, w_j^{(2)}, \ldots, w_j^{(k)}\right)$. The first coefficient vector $C^{(0)}$ consists of the k DC coefficients.

4. Each vector $C^{(j)}$ is quantized separately to produce a vector $Q^{(j)}$ of quantized coefficients (JPEG does this differently; see Section 24.5.2). The elements of $Q^{(j)}$ are then written on the compressed stream. In practice, variable-length codes are assigned to the elements, and the codes, rather than the elements themselves, are written on the compressed stream. Sometimes, as in the case of JPEG, variable-length codes are assigned to runs of zero coefficients, to achieve better compression.

In practice, the DCT is used for lossy compression. For lossless compression (where the DCT coefficients are not quantized) the DCT is inefficient but can still be used, at least theoretically, because (1) most of the coefficients are small numbers and (2) there often are runs of zero coefficients. However, the small coefficients are real numbers, not integers, so it is not clear how to write them in full precision on the compressed stream and still have compression. Other image compression methods are better suited for lossless image compression.

The decoder reads the 64 quantized coefficient vectors $Q^{(j)}$ of k elements each, saves them as the columns of a matrix, and considers the k rows of the matrix weight vectors $W^{(i)}$ of 64 elements each (notice that these $W^{(i)}$'s are not identical to the original $W^{(i)}$'s because of the quantization). It then applies the IDCT (Equation (24.14)) to each weight vector, to reconstruct (approximately) the 64 pixels of block B_i. (Again, JPEG does this differently.)

We illustrate the performance of the DCT in two dimensions by applying it to two blocks of 8×8 values. The first block (Table 24.9a) has highly correlated integer values in the range $[8, 12]$, and the second block has random values in the same range. The first block results in a large DC coefficient, followed by small AC coefficients (including 20 zeros, Table 24.9b, where negative numbers are underlined). When the coefficients are quantized (Table 24.9c), the result, shown in Table 24.9d, is very similar to the original values. In contrast, the coefficients for the second block (Table 24.10b) include just one zero. When quantized (Table 24.10c) and transformed back, many of the 64 results are very different from the original values (Table 24.10d).

◇ **Exercise 24.6:** Explain why the 64 values of Table 24.9a are correlated.

The next example illustrates the difference in the performance of the DCT when applied to a continuous-tone image and to a discrete-tone image. We start with the highly correlated pattern of Table 24.11. This is an idealized example of a continuous-tone image, since adjacent pixels differ by a constant amount except the pixel (underlined) at row 7, column 7. The 64 DCT coefficients of this pattern are listed in Table 24.12. It is clear that there are only a few dominant coefficients. Table 24.13 lists the coefficients after they have been coarsely quantized, so that only four nonzero coefficients remain! The results of performing the IDCT on these quantized coefficients are shown in Table 24.14. It is obvious that the four nonzero coefficients have reconstructed the original pattern to a high degree. The only visible difference is in row 7, column 7, which has changed from 12 to 17.55 (marked in both figures). The Matlab code for this computation is listed in Figure 24.19.

12	10	8	10	12	10	8	11
11	12	10	8	10	12	10	8
8	11	12	10	8	10	12	10
10	8	11	12	10	8	10	12
12	10	8	11	12	10	8	10
10	12	10	8	11	12	10	8
8	10	12	10	8	11	12	10
10	8	10	12	10	8	11	12

(a) Original data

81	0	0	0	0	0	0	0
0	1.57	0.61	1.90	0.38	$\underline{1.81}$	0.20	$\underline{0.32}$
0	$\underline{0.61}$	0.71	0.35	0	0.07	0	0.02
0	1.90	$\underline{0.35}$	4.76	0.77	$\underline{3.39}$	0.25	$\underline{0.54}$
0	$\underline{0.38}$	0	$\underline{0.77}$	8.00	0.51	0	0.07
0	$\underline{1.81}$	$\underline{0.07}$	$\underline{3.39}$	$\underline{0.51}$	1.57	0.56	0.25
0	$\underline{0.20}$	0	$\underline{0.25}$	0	$\underline{0.56}$	$\underline{0.71}$	0.29
0	$\underline{0.32}$	$\underline{0.02}$	$\underline{0.54}$	$\underline{0.07}$	0.25	$\underline{0.29}$	$\underline{0.90}$

(b) DCT coefficients

81	0	0	0	0	0	0	0
0	2	1	2	0	$\underline{2}$	0	0
0	$\underline{1}$	1	0	0	0	0	0
0	2	0	5	1	$\underline{3}$	0	$\underline{1}$
0	0	0	$\underline{1}$	8	1	0	0
0	$\underline{2}$	0	$\underline{3}$	$\underline{1}$	2	1	0
0	0	0	0	0	$\underline{1}$	$\underline{1}$	0
0	0	0	$\underline{1}$	0	0	0	$\underline{1}$

(c) Quantized

12.29	10.26	7.92	9.93	11.51	9.94	8.18	10.97
10.90	12.06	10.07	7.68	10.30	11.64	10.17	8.18
7.83	11.39	12.19	9.62	8.28	10.10	11.64	9.94
10.15	7.74	11.16	11.96	9.90	8.28	10.30	11.51
12.21	10.08	8.15	11.38	11.96	9.62	7.68	9.93
10.09	12.10	9.30	8.15	11.16	12.19	10.07	7.92
7.87	9.50	12.10	10.08	7.74	11.39	12.06	10.26
9.66	7.87	10.09	12.21	10.15	7.83	10.90	12.29

(d) Reconstructed data (good)

Table 24.9: Two-Dimensional DCT of a Block of Correlated Values.

8	10	9	11	11	9	9	12
11	8	12	8	11	10	11	10
9	11	9	10	12	9	9	8
9	12	10	8	8	9	8	9
12	8	9	9	12	10	8	11
8	11	10	12	9	12	12	10
10	10	12	10	12	10	10	12
12	9	11	11	9	8	8	12

(a) Original data

79.12	0.98	0.64	$\underline{1.51}$	$\underline{0.62}$	$\underline{0.86}$	1.22	0.32
0.15	$\underline{1.64}$	$\underline{0.09}$	1.23	0.10	3.29	1.08	$\underline{2.97}$
$\underline{1.26}$	$\underline{0.29}$	$\underline{3.27}$	1.69	$\underline{0.51}$	1.13	1.52	1.33
$\underline{1.27}$	$\underline{0.25}$	$\underline{0.67}$	$\underline{0.15}$	1.63	$\underline{1.94}$	0.47	$\underline{1.30}$
$\underline{2.12}$	$\underline{0.67}$	$\underline{0.07}$	$\underline{0.79}$	0.13	$\underline{1.40}$	0.16	$\underline{0.15}$
$\underline{2.68}$	1.08	$\underline{1.99}$	$\underline{1.93}$	$\underline{1.77}$	$\underline{0.35}$	0	$\underline{0.80}$
1.20	2.10	$\underline{0.98}$	0.87	$\underline{1.55}$	$\underline{0.59}$	$\underline{0.98}$	2.76
$\underline{2.24}$	0.55	0.29	0.75	$\underline{2.40}$	$\underline{0.05}$	0.06	1.14

(b) DCT coefficients

79	1	1	$\underline{2}$	$\underline{1}$	$\underline{1}$	1	0
0	$\underline{2}$	0	1	0	3	1	$\underline{3}$
$\underline{1}$	0	$\underline{3}$	2	0	1	2	1
$\underline{1}$	0	$\underline{1}$	0	2	$\underline{2}$	0	$\underline{1}0$
$\underline{2}0$	$\underline{1}$	0	$\underline{1}$	0	$\underline{1}0$	0	0
$\underline{3}$	1	$\underline{2}$	$\underline{2}$	$\underline{2}$	0	0	$\underline{1}$
1	2	$\underline{1}$	1	$\underline{2}$	$\underline{1}$	$\underline{1}$	3
$\underline{2}$	1	0	1	$\underline{2}$	0	0	1

(c) Quantized

7.59	9.23	8.33	11.88	7.12	12.47	6.98	8.56
12.09	7.97	9.3	11.52	9.28	11.62	10.98	12.39
11.02	10.06	13.81	6.5	10.82	8.28	13.02	7.54
8.46	10.22	11.16	9.57	8.45	7.77	10.28	11.89
9.71	11.93	8.04	9.59	8.04	9.7	8.59	12.14
10.27	13.58	9.21	11.83	9.99	10.66	7.84	11.27
8.34	10.32	10.53	9.9	8.31	9.34	7.47	8.93
10.61	9.04	13.66	6.04	13.47	7.65	10.97	8.89

(d) Reconstructed data (bad)

Table 24.10: Two-Dimensional DCT of a Block of Random Values.

Tables 24.15 through 24.18 show the same process applied to a Y-shaped pattern, typical of a discrete-tone image. The quantization, shown in Table 24.17, is light. The coefficients have only been truncated to the nearest integer. It is easy to see that the reconstruction, shown in Table 24.18, isn't as good as before. Quantities that should have been 10 are between 8.96 and 10.11. Quantities that should have been zero are as big as 0.86. The conclusion is that the DCT performs well on continuous-tone images but is less efficient when applied to a discrete-tone image.

24.3.2 The DCT as a Basis

The discussion so far has concentrated on how to use the DCT for compressing one-dimensional and two-dimensional data. The aim of this section and the next one is to show why the DCT works the way it does and how Equations (24.11) and (24.13) have been derived. This topic is approached from two different directions. The first interpretation of the DCT is as a special basis of an n-dimensional vector space. We

00	10	20	30	30	20	10	00
10	20	30	40	40	30	20	10
20	30	40	50	50	40	30	20
30	40	50	60	60	50	40	30
30	40	50	60	60	50	40	30
20	30	40	50	50	40	30	20
10	20	30	40	40	30	<u>12</u>	10
00	10	20	30	30	20	10	00

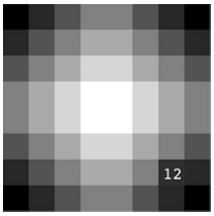


Table 24.11: A Continuous-Tone Pattern.

239	1.19	−89.76	−0.28	1.00	−1.39	−5.03	−0.79
1.18	−1.39	0.64	0.32	−1.18	1.63	−1.54	0.92
−89.76	0.64	−0.29	−0.15	0.54	−0.75	0.71	−0.43
−0.28	0.32	−0.15	−0.08	0.28	−0.38	0.36	−0.22
1.00	−1.18	0.54	0.28	−1.00	1.39	−1.31	0.79
−1.39	1.63	−0.75	−0.38	1.39	−1.92	1.81	−1.09
−5.03	−1.54	0.71	0.36	−1.31	1.81	−1.71	1.03
−0.79	0.92	−0.43	−0.22	0.79	−1.09	1.03	−0.62

Table 24.12: Its DCT Coefficients.

239	1	-90	0	0	0	0	0
0	0	0	0	0	0	0	0
-90	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 24.13: Quantized Heavily to Just Four Nonzero Coefficients.

0.65	9.23	21.36	29.91	29.84	21.17	8.94	0.30
9.26	17.85	29.97	38.52	38.45	29.78	17.55	8.91
21.44	30.02	42.15	50.70	50.63	41.95	29.73	21.09
30.05	38.63	50.76	59.31	59.24	50.56	38.34	29.70
30.05	38.63	50.76	59.31	59.24	50.56	38.34	29.70
21.44	30.02	42.15	50.70	50.63	41.95	29.73	21.09
9.26	17.85	29.97	38.52	38.45	29.78	<u>17.55</u>	8.91
0.65	9.23	21.36	29.91	29.84	21.17	8.94	0.30

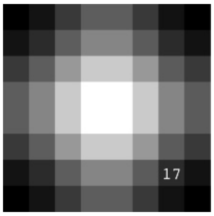


Table 24.14: Results of IDCT.

00	10	00	00	00	00	00	10
00	00	10	00	00	00	10	00
00	00	00	10	00	10	00	00
00	00	00	00	10	00	00	00
00	00	00	00	10	00	00	00
00	00	00	00	10	00	00	00
00	00	00	00	10	00	00	00
00	00	00	00	10	00	00	00

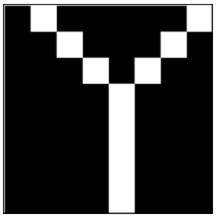


Table 24.15: A Discrete-Tone Image (Y).

13.75	−3.11	−8.17	2.46	3.75	−6.86	−3.38	6.59
4.19	−0.29	6.86	−6.85	−7.13	4.48	1.69	−7.28
1.63	0.19	6.40	−4.81	−2.99	−1.11	−0.88	−0.94
−0.61	0.54	5.12	−2.31	1.30	−6.04	−2.78	3.05
−1.25	0.52	2.99	−0.20	3.75	−7.39	−2.59	1.16
−0.41	0.18	0.65	1.03	3.87	−5.19	−0.71	−4.76
0.68	−0.15	−0.88	1.28	2.59	−1.92	1.10	−9.05
0.83	−0.21	−0.99	0.82	1.13	−0.08	1.31	−7.21

Table 24.16: Its DCT Coefficients.

13.75	−3	−8	2	3	−6	−3	6
4	−0	6	−6	−7	4	1	−7
1	0	6	−4	−2	−1	−0	−0
−0	0	5	−2	1	−6	−2	3
−1	0	2	−0	3	−7	−2	1
−0	0	0	1	3	−5	−0	−4
0	−0	−0	1	2	−1	1	−9
0	−0	−0	0	1	−0	1	−7

Table 24.17: Quantized Lightly by Truncating to Integer.

-0.13	8.96	0.55	-0.27	0.27	0.86	0.15	9.22
0.32	0.22	9.10	0.40	0.84	-0.11	9.36	-0.14
0.00	0.62	-0.20	9.71	-1.30	8.57	0.28	-0.33
-0.58	0.44	0.78	0.71	10.11	1.14	0.44	-0.49
-0.39	0.67	0.07	0.38	8.82	0.09	0.28	0.41
0.34	0.11	0.26	0.18	8.93	0.41	0.47	0.37
0.09	-0.32	0.78	-0.20	9.78	0.05	-0.09	0.49
0.16	-0.83	0.09	0.12	9.15	-0.11	-0.08	0.01

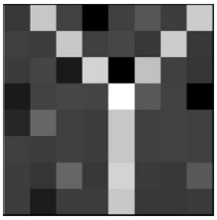


Table 24.18: The IDCT. Bad Results.

```
% 8x8 correlated values
n=8;
p=[00,10,20,30,30,20,10,00; 10,20,30,40,40,30,20,10; 20,30,40,50,50,40,30,20; ...
 30,40,50,60,60,50,40,30; 30,40,50,60,60,50,40,30; 20,30,40,50,50,40,30,20; ...
 10,20,30,40,40,30,12,10; 00,10,20,30,30,20,10,00];
figure(1), imagesc(p), colormap(gray), axis square, axis off
dct=zeros(n,n);
for j=0:7
 for i=0:7
  for x=0:7
   for y=0:7
dct(i+1,j+1)=dct(i+1,j+1)+p(x+1,y+1)*cos((2*y+1)*j*pi/16)*cos((2*x+1)*i*pi/16);
   end;
  end;
 end;
end;
dct=dct/4; dct(1,:)=dct(1,:)*0.7071; dct(:,1)=dct(:,1)*0.7071;
dct
quant=[239,1,-90,0,0,0,0,0; 0,0,0,0,0,0,0,0; -90,0,0,0,0,0,0,0; 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];
idct=zeros(n,n);
for x=0:7
 for y=0:7
  for i=0:7
if i==0 ci=0.7071; else ci=1; end;
   for j=0:7
 if j==0 cj=0.7071; else cj=1; end;
idct(x+1,y+1)=idct(x+1,y+1)+ ...
       ci*cj*quant(i+1,j+1)*cos((2*y+1)*j*pi/16)*cos((2*x+1)*i*pi/16);
   end;
  end;
 end;
end;
idct=idct/4;
idct
figure(2), imagesc(idct), colormap(gray), axis square, axis off
```

Figure 24.19: Code for Highly Correlated Pattern.

show that transforming a given data vector **p** by the DCT is equivalent to representing it by this special basis that isolates the various frequencies contained in the vector. Thus, the DCT coefficients resulting from the DCT transform of vector **p** indicate the various frequencies in the vector. The lower frequencies contain the important information in **p**, whereas the higher frequencies correspond to the details of the data in **p** and are therefore less important. This is why they can be quantized coarsely.

The second interpretation of the DCT is as a rotation, as shown intuitively for $n = 2$ (two-dimensional points) in Figure 24.2. This interpretation considers the DCT a rotation matrix that rotates an n-dimensional point with identical coordinates $(x, x, \ldots, x)$ from its original location to the x axis, where its coordinates become $(\alpha, \epsilon_2, \ldots, \epsilon_n)$ where the various ϵ_i are small numbers or zeros. Both interpretations are illustrated for the case $n = 3$, because this is the largest number of dimensions where it is possible to visualize geometric transformations.

For the special case $n = 3$, Equation (24.11) reduces to

$$G_f = \sqrt{\frac{2}{3}} C_f \sum_{t=0}^{2} p_t \cos\left[\frac{(2t+1)f\pi}{6}\right], \quad \text{for } f = 0, 1, 2.$$

Temporarily ignoring the normalization factors $\sqrt{2/3}$ and C_f, this can be written in matrix notation as

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \end{bmatrix} = \begin{bmatrix} \cos 0 & \cos 0 & \cos 0 \\ \cos\frac{\pi}{6} & \cos\frac{3\pi}{6} & \cos\frac{5\pi}{6} \\ \cos 2\frac{\pi}{6} & \cos 2\frac{3\pi}{6} & \cos 2\frac{5\pi}{6} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} = \mathbf{D}\cdot\mathbf{p}.$$

Thus, the DCT of the three data values $\mathbf{p} = (p_0, p_1, p_2)$ is obtained as the product of the DCT matrix $\mathbf{D}$ and the vector $\mathbf{p}$. We can therefore think of the DCT as the product of a DCT matrix and a data vector, where the matrix is constructed as follows: Select the three angles $\pi/6$, $3\pi/6$, and $5\pi/6$ and compute the three basis vectors $\cos(f\theta)$ for $f = 0$, 1, and 2, and for the three angles. The results are listed in Table 24.20 for the benefit of the reader.

θ	0.5236	1.5708	2.618
$\cos 0\theta$	1.	1.	1.
$\cos 1\theta$	0.866	0	−0.866
$\cos 2\theta$	0.5	−1	0.5

Table 24.20: The DCT Matrix for $n = 3$.

Because of the particular choice of the three angles, these vectors are orthogonal but not orthonormal. Their magnitudes are $\sqrt{3}$, $\sqrt{1.5}$, and $\sqrt{1.5}$, respectively. Normalizing them results in the three vectors $\mathbf{v}_1 = (0.5774, 0.5774, 0.5774)$, $\mathbf{v}_2 = (0.7071, 0, -0.7071)$, and $\mathbf{v}_3 = (0.4082, -0.8165, 0.4082)$. When stacked vertically, they produce the following 3×3 matrix

$$\mathbf{M} = \begin{bmatrix} 0.5774 & 0.5774 & 0.5774 \\ 0.7071 & 0 & -0.7071 \\ 0.4082 & -0.8165 & 0.4082 \end{bmatrix}. \tag{24.15}$$

(Equation (24.11) tells us how to normalize these vectors: Multiply each by $\sqrt{2/3}$, and then multiply the first by $1/\sqrt{2}$.) Notice that as a result of the normalization the columns of $\mathbf{M}$ have also become orthonormal, so $\mathbf{M}$ is an orthonormal matrix (such matrices have special properties).

The steps of computing the DCT matrix for an arbitrary n are as follows:

1. Select the n angles $\theta_j = (j+0.5)\pi/n$ for $j = 0, \ldots, n-1$. If we divide the interval $[0, \pi]$ into n equal-size segments, these angles are the centerpoints of the segments.

2. Compute the n vectors $\mathbf{v}_k$ for $k = 0, 1, 2, \ldots, n-1$, each with the n components $\cos(k\theta_j)$.

3. Normalize each of the n vectors and arrange them as the n rows of a matrix.

The angles selected for the DCT are $\theta_j = (j + 0.5)\pi/n$, so the components of each vector $\mathbf{v}_k$ are $\cos[k(j + 0.5)\pi/n]$ or $\cos[k(2j + 1)\pi/(2n)]$. Section 24.3.4 covers three other ways to select such angles. This choice of angles has the following two useful properties: (1) The resulting vectors are orthogonal, and (2) for increasing values of k, the n vectors $\mathbf{v}_k$ contain increasing frequencies (Figure 24.21). For $n = 3$, the top row of $\mathbf{M}$ (Equation (24.15)) corresponds to zero frequency, the middle row (whose elements become monotonically smaller) represents low frequency, and the bottom row (with three elements that first go down, then up) represents high frequency. Given a three-dimensional vector $\mathbf{v} = (v_1, v_2, v_3)$, the product $\mathbf{M}\cdot\mathbf{v}$ is a triplet whose components indicate the magnitudes of the various frequencies included in $\mathbf{v}$; they are *frequency coefficients*. (Strictly speaking, the product is $\mathbf{M}\cdot\mathbf{v}^T$, but we ignore the transpose in cases where the meaning is clear.) The following three extreme examples illustrate the meaning of this statement.

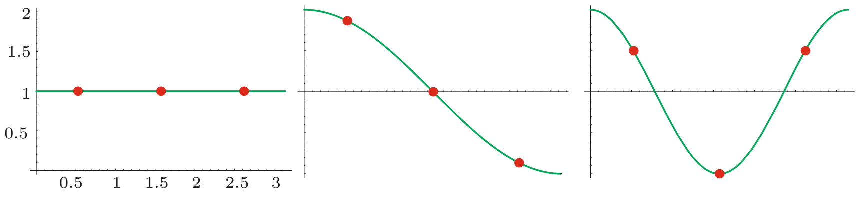


Figure 24.21: Increasing Frequencies.

The first example is $\mathbf{v} = (v, v, v)$. The three components of $\mathbf{v}$ are identical, so they correspond to zero frequency. The product $\mathbf{M}\cdot\mathbf{v}$ produces the frequency coefficients $(1.7322v, 0, 0)$, indicating no high frequencies. The second example is $\mathbf{v} = (v, 0, -v)$. The three components of $\mathbf{v}$ vary slowly from v to $-v$, so this vector contains a low frequency. The product $\mathbf{M}\cdot\mathbf{v}$ produces the coefficients $(0, 1.4142v, 0)$, confirming this result. The third example is $\mathbf{v} = (v, -v, v)$. The three components of $\mathbf{v}$ vary from v to $-v$ to v, so this vector contains a high frequency. The product $\mathbf{M}\cdot\mathbf{v}$ produces $(0, 0, 1.6329v)$, again indicating the correct frequency.

These examples are not very realistic because the vectors being tested are short, simple, and contain a single frequency each. Most vectors are more complex and contain several frequencies, which makes this method useful. A simple example of a vector with two frequencies is $\mathbf{v} = (1., 0.33, -0.34)$. The product $\mathbf{M}\cdot\mathbf{v}$ results in $(0.572, 0.948, 0)$ which indicates a large medium frequency, small zero frequency, and no high frequency. This makes sense once we realize that the vector being tested is the sum $0.33(1, 1, 1) + 0.67(1, 0, -1)$. A similar example is the sum $0.9(-1, 1, -1)+0.1(1, 1, 1) = (-0.8, 1, -0.8)$, which when multiplied by $\mathbf{M}$ produces $(-0.346, 0, -1.469)$. On the other hand, a vector with random components, such as $(1, 0, 0.33)$, typically contains roughly equal amounts of all three frequencies and produces three large frequency coefficients. The product $\mathbf{M}\cdot(1, 0, 0.33)$ produces $(0.77, 0.47, 0.54)$ because $(1, 0, 0.33)$ is the sum

$$0.33(1, 1, 1) + 0.33(1, 0, -1) + 0.33(1, -1, 1).$$

Notice that if $\mathbf{M}\cdot\mathbf{v} = \mathbf{c}$, then $\mathbf{M}^T\cdot\mathbf{c} = \mathbf{M}^{-1}\cdot\mathbf{c} = \mathbf{v}$. The original vector $\mathbf{v}$ can therefore be reconstructed from its frequency coefficients (up to small differences due to the limited precision of machine arithmetic). The inverse $\mathbf{M}^{-1}$ of $\mathbf{M}$ is also its transpose $\mathbf{M}^T$ because $\mathbf{M}$ is orthonormal.

A three-dimensional vector can have only the three frequencies zero, medium, and high. Similarly, an n-dimensional vector can have n different frequencies, which this method can identify. We concentrate on the case $n = 8$ and start with the DCT in one dimension. Figure 24.22 shows eight cosine waves of the form $\cos(f\theta_j)$, for $0 \le \theta_j \le \pi$, with frequencies $f = 0, 1, \ldots, 7$. Each wave is sampled at the eight points

$$\theta_j = \frac{\pi}{16},\quad \frac{3\pi}{16},\quad \frac{5\pi}{16},\quad \frac{7\pi}{16},\quad \frac{9\pi}{16},\quad \frac{11\pi}{16},\quad \frac{13\pi}{16},\quad \frac{15\pi}{16} \tag{24.16}$$

to form one basis vector $\mathbf{v}_f$, and the resulting eight vectors $\mathbf{v}_f$, $f = 0, 1, \ldots, 7$ (a total of 64 numbers) are shown in Table 24.23. They serve as the basis matrix of the DCT. Notice the similarity between this table and matrix $\mathbf{W}$ of Equation (24.3).

Because of the particular choice of the eight sample points, the $\mathbf{v}_i$'s are orthogonal. This is easy to check directly with appropriate mathematical software, but Section 24.3.4 describes a more elegant way of proving this property. After normalization, the $\mathbf{v}_i$'s can be considered either as an 8×8 transformation matrix (specifically, a rotation matrix, since it is orthonormal) or as a set of eight orthogonal vectors that constitute the basis of a vector space. Any vector $\mathbf{p}$ in this space can be expressed as a linear combination of the $\mathbf{v}_i$'s. As an example, we select the eight (correlated) numbers $\mathbf{p} = (0.6, 0.5, 0.4, 0.5, 0.6, 0.5, 0.4, 0.55)$ as our test data and express $\mathbf{p}$ as a linear combination $\mathbf{p} = \sum w_i\mathbf{v}_i$ of the eight basis vectors $\mathbf{v}_i$. Solving this system of eight equations yields the eight weights

$$\begin{gathered} w_0 = 0.506,\quad w_1 = 0.0143,\quad w_2 = 0.0115,\quad w_3 = 0.0439,\\ w_4 = 0.0795,\quad w_5 = -0.0432,\quad w_6 = 0.00478,\quad w_7 = -0.0077. \end{gathered}$$

Weight w_0 is not much different from the elements of $\mathbf{p}$, but the other seven weights are much smaller. This is how the DCT (or any other orthogonal transform) can lead to compression. The eight weights can be quantized and written on the compressed stream, where they occupy less space than the eight components of $\mathbf{p}$.

Figure 24.24 illustrates this linear combination graphically. Each of the eight $\mathbf{v}_i$'s is shown as a row of eight small, gray rectangles (a basis image) where a value of $+1$ is painted white and -1 is black. The eight elements of vector $\mathbf{p}$ are also displayed as a row of eight grayscale pixels.

To summarize, we interpret the DCT in one dimension as a set of basis images that have higher and higher frequencies. Given a data vector, the DCT separates the frequencies in the data and represents the vector as a linear combination (or a weighted sum) of the basis images. The weights are the DCT coefficients. This interpretation can be extended to the DCT in two dimensions. We apply Equation (24.13) to the case $n = 8$ to create 64 small basis images of 8×8 pixels each. The 64 images are then used as a basis of a 64-dimensional vector space. Any image B of 8×8 pixels can be expressed as a linear combination of the basis images, and the 64 weights of this linear combination are the DCT coefficients of B.

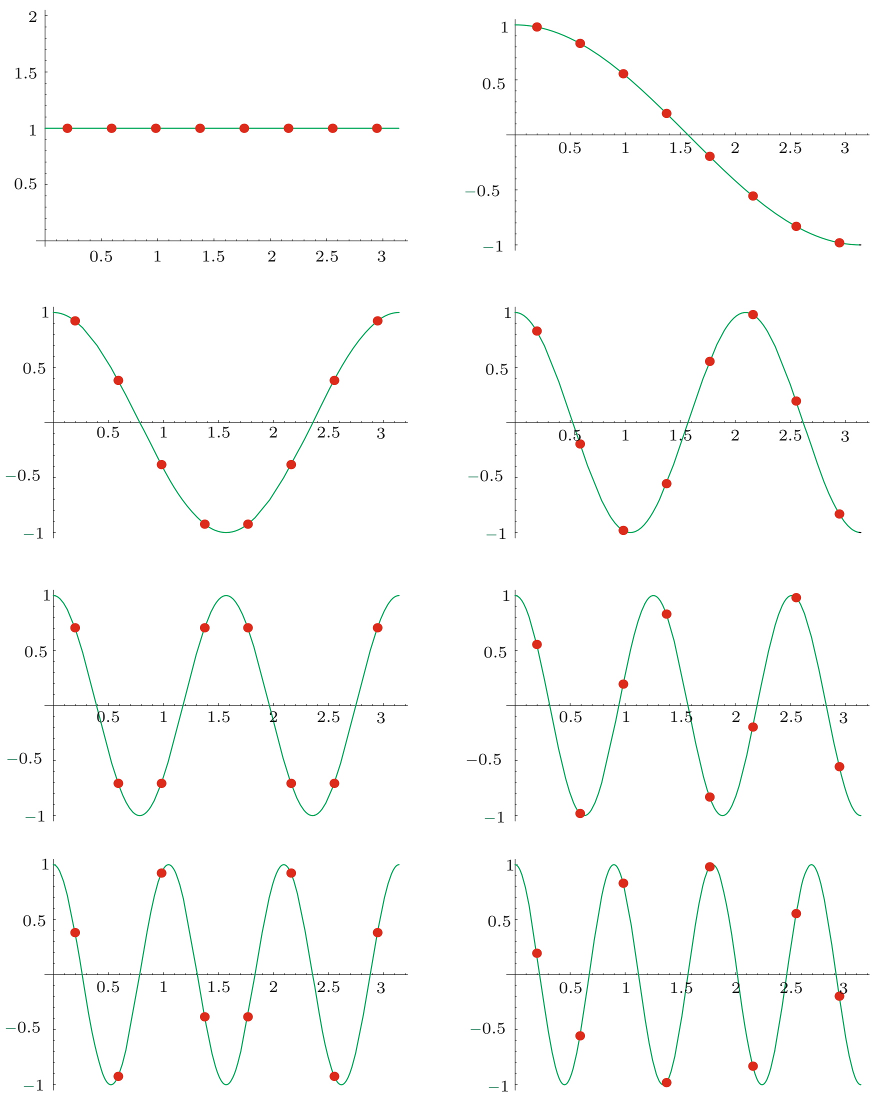


Figure 24.22: Angle and Cosine Values for an 8-Point DCT.

θ	0.196	0.589	0.982	1.374	1.767	2.160	2.553	2.945
$\cos 0\theta$	1.	1.	1.	1.	1.	1.	1.	1.
$\cos 1\theta$	0.981	0.831	0.556	0.195	−0.195	−0.556	−0.831	−0.981
$\cos 2\theta$	0.924	0.383	−0.383	−0.924	−0.924	−0.383	0.383	0.924
$\cos 3\theta$	0.831	−0.195	−0.981	−0.556	0.556	0.981	0.195	−0.831
$\cos 4\theta$	0.707	−0.707	−0.707	0.707	0.707	−0.707	−0.707	0.707
$\cos 5\theta$	0.556	−0.981	0.195	0.831	−0.831	−0.195	0.981	−0.556
$\cos 6\theta$	0.383	−0.924	0.924	−0.383	−0.383	0.924	−0.924	0.383
$\cos 7\theta$	0.195	−0.556	0.831	−0.981	0.981	−0.831	0.556	−0.195

Table 24.23: The Unnormalized DCT Matrix in One Dimension for $n = 8$.

```
Table[N[t],{t,Pi/16,15Pi/16,Pi/8}]
dctp[pw_]:=Table[N[Cos[pw t]],{t,Pi/16,15Pi/16,Pi/8}]
dctp[0]
dctp[1]
...
dctp[7]
```

Code for Table 24.23.

```
dct[pw_]:=Plot[Cos[pw t], {t,0,Pi}, DisplayFunction->Identity,
 AspectRatio->Automatic];
dcdot[pw_]:=ListPlot[Table[{t,Cos[pw t]},{t,Pi/16,15Pi/16,Pi/8}],
 DisplayFunction->Identity]
Show[dct[0],dcdot[0], Prolog->AbsolutePointSize[4],
 DisplayFunction->$DisplayFunction]
...
Show[dct[7],dcdot[7], Prolog->AbsolutePointSize[4],
 DisplayFunction->$DisplayFunction]
```

Code for Figure 24.22.

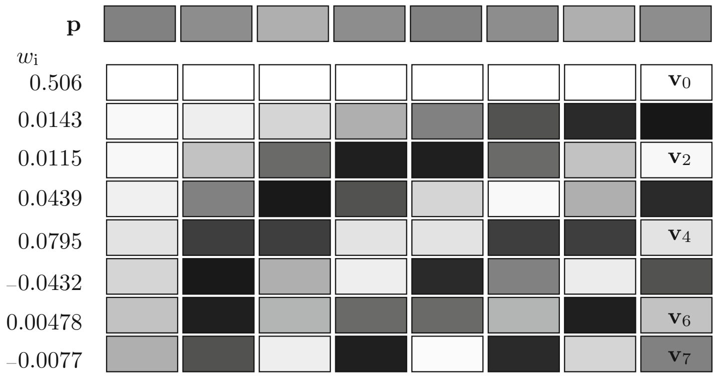


Figure 24.24: A Graphic Representation of the One-Dimensional DCT.

Figure 24.25 shows the graphic representation of the 64 basis images of the two-dimensional DCT for $n = 8$. A general element (i, j) in this figure is the 8×8 image obtained by calculating the product $\cos(i \cdot s)\cos(j \cdot t)$, where s and t are varied independently over the values listed in Equation (24.16) and i and j vary from 0 to 7. This figure can easily be generated by the *Mathematica* code shown with it. The alternative code shown is a modification of code in [Watson 94], and it requires the `GraphicsImage.m` package, which is not widely available.

Using appropriate software, it is easy to perform DCT calculations and display the results graphically. Figure 24.26a shows a random 8×8 data unit consisting of zeros and ones. The same unit is shown in Figure 24.26b graphically, with 1 as white and 0 as black. Figure 24.26c shows the weights by which each of the 64 DCT basis images has to be multiplied in order to reproduce the original data unit. In this figure, zero is shown in neutral gray, positive numbers are bright (notice how bright the DC weight is), and negative numbers are shown as dark. Figure 24.26d shows the weights numerically. The *Mathematica* code that does all that is also listed. Figure 24.27 is similar, but for a very regular data unit.

◇ **Exercise 24.7:** . Imagine an 8×8 block of values where all the odd-numbered rows consist of 1's and all the even-numbered rows contain zeros. What can we say about the DCT weights of this block?

24.3.3 The DCT as a Rotation

The second interpretation of matrix $\mathbf{M}$ (Equation (24.15)) is as a rotation. We already know that $\mathbf{M}\cdot(v, v, v)$ results in $(1.7322v, 0, 0)$ and this can be interpreted as a rotation of point (v, v, v) to the point $(1.7322v, 0, 0)$. The former point is located on the line that makes equal angles with the three coordinate axes, and the latter point is on the x axis. When considered in terms of adjacent pixels, this rotation has a simple meaning. Imagine three adjacent pixels in an image. They are normally similar, so we start by examining the case where they are identical. When three identical pixels are considered the coordinates of a point in three dimensions, that point is located on the line $x = y = z$. Rotating this line to the x axis brings our point to that axis where its x coordinate hasn't changed much and its y and z coordinates are zero. This is how such a rotation leads to compression. Generally, three adjacent pixels p_1, p_2, and p_3 are similar but not identical, which locates the point (p_1, p_2, p_3) somewhat off the line $x = y = z$. After the rotation, the point will end up near the x axis, where its y and z coordinates will be small numbers.

This interpretation of $\mathbf{M}$ as a rotation makes sense because $\mathbf{M}$ is orthonormal and any orthonormal matrix is a rotation matrix. However, the determinant of a rotation matrix is 1, whereas the determinant of our matrix is -1. An orthonormal matrix whose determinant is -1 performs an *improper rotation* (a rotation combined with a reflection). To get a better insight into the transformation performed by $\mathbf{M}$, we apply the QR matrix decomposition technique (Section 24.3.8) to decompose $\mathbf{M}$ into the matrix product $T1\times T2\times T3\times T4$, where

$$T1 = \begin{bmatrix} 0.8165 & 0 & -0.5774 \\ 0 & 1 & 0 \\ 0.5774 & 0 & 0.8165 \end{bmatrix}, \quad T2 = \begin{bmatrix} 0.7071 & -0.7071 & 0 \\ 0.7071 & 0.7071 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

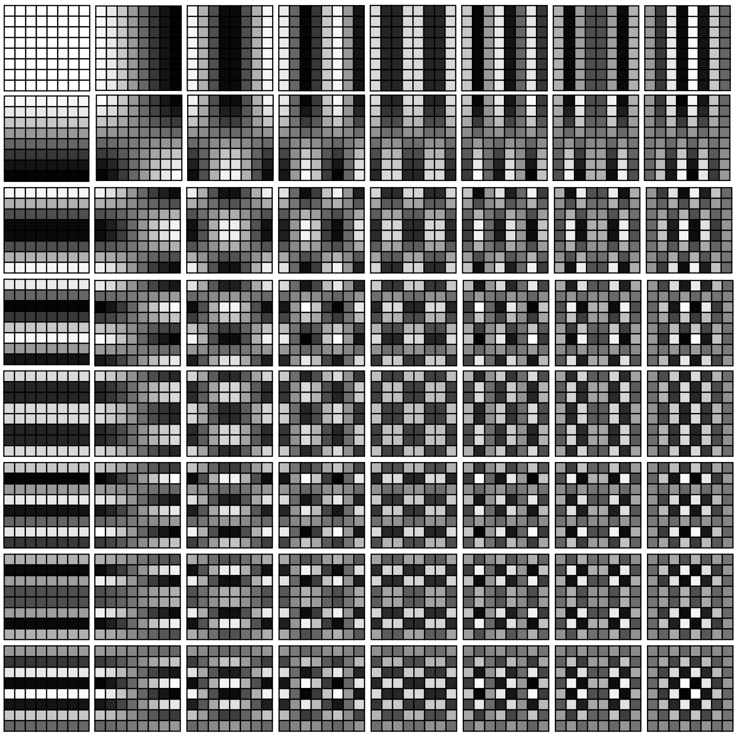


Figure 24.25: The 64 Basis Images of the Two-Dimensional DCT.

```
dctp[fs_,ft_]:=Table[SetAccuracy[N[(1.-Cos[fs s]Cos[ft t])/2],3],
 {s,Pi/16,15Pi/16,Pi/8},{t,Pi/16,15Pi/16,Pi/8}]//TableForm
dctp[0,0]
dctp[0,1]
...
dctp[7,7]
```

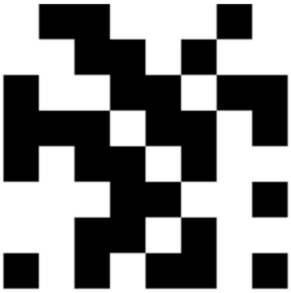
Code for Figure 24.25.

```
Needs["GraphicsImage`"] (* Draws 2D DCT Coefficients *)
DCTMatrix=Table[If[k==0,Sqrt[1/8],Sqrt[1/4]Cos[Pi(2j+1)k/16]],
 {k,0,7}, {j,0,7}] //N;
DCTTensor=Array[Outer[Times, DCTMatrix[[#1]],DCTMatrix[[#2]]]&,
 {8,8}];
Show[GraphicsArray[Map[GraphicsImage[#, {-.25,.25}]&, DCTTensor,{2}]]]
```

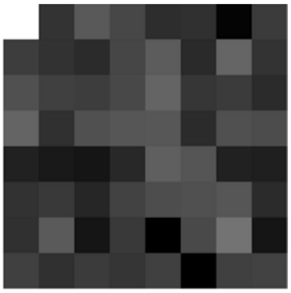
Alternative Code for Figure 24.25.

```
10011101
11001011
01100100
00010010
01001011
11100110
11001011
01010010
```

(a)



(b)



(c)

4.000	−0.133	0.637	0.272	−0.250	−0.181	−1.076	0.026
0.081	−0.178	−0.300	0.230	0.694	−0.309	0.875	−0.127
0.462	0.125	0.095	0.291	0.868	−0.070	0.021	−0.280
0.837	−0.194	0.455	0.583	0.588	−0.281	0.448	0.383
−0.500	−0.635	−0.749	−0.346	0.750	0.557	−0.502	−0.540
−0.167	0	−0.366	0.146	0.393	0.448	0.577	−0.268
−0.191	0.648	−0.729	−0.008	−1.171	0.306	1.155	−0.744
0.122	−0.200	0.038	−0.118	0.138	−1.154	0.134	0.148

(d)

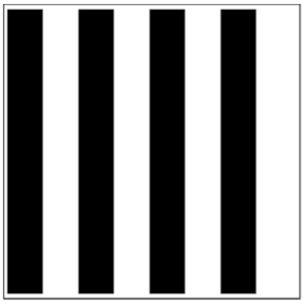
Figure 24.26: An Example of the DCT in Two Dimensions.

```
DCTMatrix=Table[If[k==0,Sqrt[1/8],Sqrt[1/4]Cos[Pi(2j+1)k/16]],
 {k,0,7}, {j,0,7}] //N;
DCTTensor=Array[Outer[Times, DCTMatrix[[#1]],DCTMatrix[[#2]]]&,
 {8,8}];
img={{1,0,0,1,1,1,0,1},{1,1,0,0,1,0,1,1},
{0,1,1,0,0,1,0,0},{0,0,0,1,0,0,1,0},
{0,1,0,0,1,0,1,1},{1,1,1,0,0,1,1,0},
{1,1,0,0,1,0,1,1},{0,1,0,1,0,0,1,0}};
ShowImage[Reverse[img]]
dctcoeff=Array[(Plus @@ Flatten[DCTTensor[[#1,#2]] img])&,{8,8}];
dctcoeff=SetAccuracy[dctcoeff,4];
dctcoeff=Chop[dctcoeff,.001];
MatrixForm[dctcoeff]
ShowImage[Reverse[dctcoeff]]
```

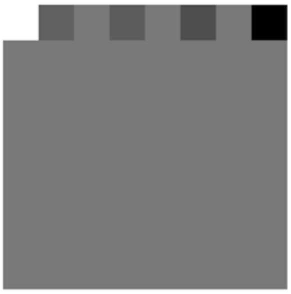
Code for Figure 24.26.

01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101

(a)



(b)



(c)

4.000	−0.721	0	−0.850	0	−1.273	0	−3.625
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	0
0	0	0	0	0	0	0	0

(d)

Figure 24.27: An Example of the DCT in Two Dimensions.

Some painters transform the sun into a yellow spot; others transform a yellow spot into the sun.

—Pablo Picasso.

```
DCTMatrix=Table[If[k==0,Sqrt[1/8],Sqrt[1/4]Cos[Pi(2j+1)k/16]],
 {k,0,7}, {j,0,7}] //N;
DCTTensor=Array[Outer[Times, DCTMatrix[[#1]],DCTMatrix[[#2]]]&,
 {8,8}];
img={{0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},
 {0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},
 {0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1}};
ShowImage[Reverse[img]]
dctcoeff=Array[(Plus @@ Flatten[DCTTensor[[#1,#2]] img])&,{8,8}];
dctcoeff=SetAccuracy[dctcoeff,4];
dctcoeff=Chop[dctcoeff,.001];
MatrixForm[dctcoeff]
ShowImage[Reverse[dctcoeff]]
```

Code for Figure 24.27.

$$T3=\begin{bmatrix}1&0&0\\0&0&1\\0&-1&0\end{bmatrix},\quad T4=\begin{bmatrix}1&0&0\\0&1&0\\0&0&-1\end{bmatrix}.$$

Each of matrices $T1$, $T2$, and $T3$ performs a rotation about one of the coordinate axes (these are called *Givens rotations*, Section 4.4.4) Matrix $T4$ is a reflection about the z axis. The transformation $\mathbf{M}\cdot(1,1,1)$ can now be written as $T1\times T2\times T3\times T4\times(1,1,1)$, where $T4$ reflects point $(1,1,1)$ to $(1,1,-1)$, $T3$ rotates $(1,1,-1)$ 90° about the x axis to $(1,-1,-1)$, which is rotated by $T2$ 45° about the z axis to $(1.4142,0,-1)$, which is rotated by $T1$ 35.26° about the y axis to $(1.7321,0,0)$.

(This particular sequence of transformations is a result of the order in which the individual steps of the QR decomposition have been performed. Performing the same steps in a different order results in different sequences of rotations. One example is (1) a reflection about the z axis that transforms $(1,1,1)$ to $(1,1,-1)$, (2) a rotation of $(1,1,-1)$ 135° about the x axis to $(1,-1.4142,0)$, and (3) a further rotation of 54.74° about the z axis to $(1.7321,0,0)$.)

For an arbitrary n, this interpretation is similar. We start with a vector of n adjacent pixels. They are considered the coordinates of a point in n-dimensional space. If the pixels are similar, the point is located near the line that makes equal angles with all the coordinate axes. Applying the DCT in one dimension (Equation (24.11)) rotates the point and brings it close to the x axis, where its first coordinate hasn't changed much and its remaining $n-1$ coordinates are small numbers. This is how the DCT in one dimension can be considered a single rotation in n-dimensional space. The rotation can be broken up into a reflection followed by $n-1$ Givens rotations, but a user of the DCT need not be concerned with these details.

The DCT in two dimensions is interpreted similarly as a double rotation. This interpretation starts with a block of $n\times n$ pixels (Figure 24.28a, where the pixels are labeled `L`). It first considers each row of this block as a point $(p_{x,0},p_{x,1},\ldots,p_{x,n-1})$ in n-dimensional space, and it rotates the point by means of the innermost sum

$$G1_{x,j}=\sqrt{\frac{2}{n}}C_j\sum_{y=0}^{n-1}p_{xy}\cos\left(\frac{(2y+1)j\pi}{2n}\right)$$

of Equation (24.13). This results in a block $G1_{x,j}$ of $n\times n$ coefficients where the first element of each row is dominant (labeled `L` in Figure 24.28b) and the remaining elements are small (labeled `S` in that figure). The outermost sum of Equation (24.13) is

$$G_{ij}=\sqrt{\frac{2}{n}}C_i\sum_{x=0}^{n-1}G1_{x,j}\cos\left(\frac{(2x+1)i\pi}{2n}\right).$$

Here, the *columns* of $G1_{x,j}$ are considered points in n-dimensional space and are rotated. The result is one large coefficient at the top-left corner of the block (`L` in Figure 24.28c) and n^2-1 small coefficients elsewhere (`S` and `s` in that figure). This interpretation considers the two-dimensional DCT as two separate rotations in n dimensions; the first one rotates each of the n rows, and the second one rotates each of the n columns. It is

```
L L L L L L L L     L S S S S S S S     L S S S S S S S
L L L L L L L L     L S S S S S S S     S s s s s s s s
L L L L L L L L     L S S S S S S S     S s s s s s s s
L L L L L L L L     L S S S S S S S     S s s s s s s s
L L L L L L L L     L S S S S S S S     S s s s s s s s
L L L L L L L L     L S S S S S S S     S s s s s s s s
L L L L L L L L     L S S S S S S S     S s s s s s s s
L L L L L L L L     L S S S S S S S     S s s s s s s s

      (a)                 (b)                 (c)
```

Figure 24.28: The Two-Dimensional DCT as a Double Rotation.

interesting to observe that $2n$ rotations in n dimensions are faster than one rotation in n^2 dimensions, since the latter requires an $n^2 \times n^2$ rotation matrix.

24.3.4 The Four DCT Types

There are four ways to select n equally-spaced angles that generate orthogonal vectors of cosines. They correspond (after the vectors are normalized by scale factors) to four discrete cosine transforms designated DCT-1 through DCT-4. The most useful is DCT-2, which is normally referred to as *the* DCT. Equation (24.17) lists the definitions of the four types. The actual angles of the DCT-1 and DCT-2 are listed (for $n = 8$) in Table 24.29. Note that DCT-3 is the transpose of DCT-2 and DCT-4 is a shifted version of DCT-1. Notice that the DCT-1 has $n+1$ vectors of $n+1$ cosines each. In each of the four types, the n (or $n+1$) DCT vectors are orthogonal and become normalized after they are multiplied by the proper scale factor. Figure 24.30 lists *Mathematica* code to generate the normalized vectors of the four types and test for normalization.

$$
\begin{aligned}
\mathrm{DCT}^1_{k,j} &= \sqrt{\frac{2}{n}} C_k C_j \cos\left[\frac{k\,j\,\pi}{n}\right], \quad k,\, j = 0, 1, \ldots, n,\\
\mathrm{DCT}^2_{k,j} &= \sqrt{\frac{2}{n}} C_k \cos\left[\frac{k(j+\frac{1}{2})\pi}{n}\right], \quad k,\, j = 0, 1, \ldots, n-1,\\
\mathrm{DCT}^3_{k,j} &= \sqrt{\frac{2}{n}} C_j \cos\left[\frac{(k+\frac{1}{2})j\pi}{n}\right], \quad k,\, j = 0, 1, \ldots, n-1,\\
\mathrm{DCT}^4_{k,j} &= \sqrt{\frac{2}{n}} \cos\left[\frac{(k+\frac{1}{2})(j+\frac{1}{2})\pi}{n}\right], \quad k,\, j = 0, 1, \ldots, n-1,
\end{aligned}
\tag{24.17}
$$

where the scale factor C_x is defined by

$$
C_x = \begin{cases} 1/\sqrt{2}, & \text{if } x = 0 \text{ or } x = n,\\ 1, & \text{otherwise.} \end{cases}
$$

Orthogonality can be proved either directly, by multiplying pairs of different vectors, or indirectly. The latter approach is discussed in detail in [Strang 99] and it proves that the DCT vectors are orthogonal by showing that they are the eigenvectors of certain symmetric matrices. In the case of the DCT-2, for example, the symmetric matrix is

k	scale	DCT-1 Angles (9×9)								
0	$\frac{1}{2\sqrt{2}}$	0*	0	0	0	0	0	0	0	0*
1	$\frac{1}{2}$	0	$\frac{\pi}{8}$	$\frac{2\pi}{8}$	$\frac{3\pi}{8}$	$\frac{4\pi}{8}$	$\frac{5\pi}{8}$	$\frac{6\pi}{8}$	$\frac{7\pi}{8}$	$\frac{8\pi}{8}$
2	$\frac{1}{2}$	0	$\frac{2\pi}{8}$	$\frac{4\pi}{8}$	$\frac{6\pi}{8}$	$\frac{8\pi}{8}$	$\frac{10\pi}{8}$	$\frac{12\pi}{8}$	$\frac{14\pi}{8}$	$\frac{16\pi}{8}$
3	$\frac{1}{2}$	0	$\frac{3\pi}{8}$	$\frac{6\pi}{8}$	$\frac{9\pi}{8}$	$\frac{12\pi}{8}$	$\frac{15\pi}{8}$	$\frac{18\pi}{8}$	$\frac{21\pi}{8}$	$\frac{24\pi}{8}$
4	$\frac{1}{2}$	0	$\frac{4\pi}{8}$	$\frac{8\pi}{8}$	$\frac{12\pi}{8}$	$\frac{16\pi}{8}$	$\frac{20\pi}{8}$	$\frac{24\pi}{8}$	$\frac{28\pi}{8}$	$\frac{32\pi}{8}$
5	$\frac{1}{2}$	0	$\frac{5\pi}{8}$	$\frac{10\pi}{8}$	$\frac{15\pi}{8}$	$\frac{20\pi}{8}$	$\frac{25\pi}{8}$	$\frac{30\pi}{8}$	$\frac{35\pi}{8}$	$\frac{40\pi}{8}$
6	$\frac{1}{2}$	0	$\frac{6\pi}{8}$	$\frac{12\pi}{8}$	$\frac{18\pi}{8}$	$\frac{24\pi}{8}$	$\frac{30\pi}{8}$	$\frac{36\pi}{8}$	$\frac{42\pi}{8}$	$\frac{48\pi}{8}$
7	$\frac{1}{2}$	0	$\frac{7\pi}{8}$	$\frac{14\pi}{8}$	$\frac{21\pi}{8}$	$\frac{28\pi}{8}$	$\frac{35\pi}{8}$	$\frac{42\pi}{8}$	$\frac{49\pi}{8}$	$\frac{56\pi}{8}$
8	$\frac{1}{2\sqrt{2}}$	0*	$\frac{8\pi}{8}$	$\frac{16\pi}{8}$	$\frac{24\pi}{8}$	$\frac{32\pi}{8}$	$\frac{40\pi}{8}$	$\frac{48\pi}{8}$	$\frac{56\pi}{8}$	$\frac{64\pi}{8}$*

* the scale factor for these four angles is 4.

k	scale	DCT-2 Angles							
0	$\frac{1}{2\sqrt{2}}$	0	0	0	0	0	0	0	0
1	$\frac{1}{2}$	$\frac{\pi}{16}$	$\frac{3\pi}{16}$	$\frac{5\pi}{16}$	$\frac{7\pi}{16}$	$\frac{9\pi}{16}$	$\frac{11\pi}{16}$	$\frac{13\pi}{16}$	$\frac{15\pi}{16}$
2	$\frac{1}{2}$	$\frac{2\pi}{16}$	$\frac{6\pi}{16}$	$\frac{10\pi}{16}$	$\frac{14\pi}{16}$	$\frac{18\pi}{16}$	$\frac{22\pi}{16}$	$\frac{26\pi}{16}$	$\frac{30\pi}{16}$
3	$\frac{1}{2}$	$\frac{3\pi}{16}$	$\frac{9\pi}{16}$	$\frac{15\pi}{16}$	$\frac{21\pi}{16}$	$\frac{27\pi}{16}$	$\frac{33\pi}{16}$	$\frac{39\pi}{16}$	$\frac{45\pi}{16}$
4	$\frac{1}{2}$	$\frac{4\pi}{16}$	$\frac{12\pi}{16}$	$\frac{20\pi}{16}$	$\frac{28\pi}{16}$	$\frac{36\pi}{16}$	$\frac{44\pi}{16}$	$\frac{52\pi}{16}$	$\frac{60\pi}{16}$
5	$\frac{1}{2}$	$\frac{5\pi}{16}$	$\frac{15\pi}{16}$	$\frac{25\pi}{16}$	$\frac{35\pi}{16}$	$\frac{45\pi}{16}$	$\frac{55\pi}{16}$	$\frac{65\pi}{16}$	$\frac{75\pi}{16}$
6	$\frac{1}{2}$	$\frac{6\pi}{16}$	$\frac{18\pi}{16}$	$\frac{30\pi}{16}$	$\frac{42\pi}{16}$	$\frac{54\pi}{16}$	$\frac{66\pi}{16}$	$\frac{78\pi}{16}$	$\frac{90\pi}{16}$
7	$\frac{1}{2}$	$\frac{7\pi}{16}$	$\frac{21\pi}{16}$	$\frac{35\pi}{16}$	$\frac{49\pi}{16}$	$\frac{63\pi}{16}$	$\frac{77\pi}{16}$	$\frac{91\pi}{16}$	$\frac{105\pi}{16}$

Table 24.29: Angle Values for the DCT-1 and DCT-2.

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \vdots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{bmatrix}. \qquad \text{For } n = 3, \text{ matrix} \qquad \mathbf{A}_3 = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

has eigenvectors $(0.5774, 0.5774, 0.5774)$, $(0.7071, 0, -0.7071)$, $(0.4082, -0.8165, 0.4082)$ with eigenvalues 0, 1, and 3, respectively. Recall that these eigenvectors are the rows of matrix $\mathbf{M}$ of Equation (24.15).

24.3.5 Practical DCT

Equation (24.13) can be coded directly in any higher-level language. Since this equation is the basis of several compression methods such as JPEG and MPEG, its fast calculation is essential. It can be speeded up considerably by making several improvements, and this section offers some ideas.

```
(* DCT-1. Notice (n+1)x(n+1) *)
Clear[n, nor, kj, DCT1, T1];
n=8; nor=Sqrt[2/n];
kj[i_]:=If[i==0 || i==n, 1/Sqrt[2], 1];
DCT1[k_]:=Table[nor kj[j] kj[k] Cos[j k Pi/n], {j,0,n}]
T1=Table[DCT1[k], {k,0,n}]; (* Compute nxn cosines *)
MatrixForm[T1] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T1[[i]].T1[[j]]]], {i,1,n}, {j,1,n}]]

(* DCT-2 *)
Clear[n, nor, kj, DCT2, T2];
n=8; nor=Sqrt[2/n];
kj[i_]:=If[i==0 || i==n, 1/Sqrt[2], 1];
DCT2[k_]:=Table[nor kj[k] Cos[(j+1/2)k Pi/n], {j,0,n-1}]
T2=Table[DCT2[k], {k,0,n-1}]; (* Compute nxn cosines *)
MatrixForm[T2] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T2[[i]].T2[[j]]]], {i,1,n}, {j,1,n}]]

(* DCT-3. This is the transpose of DCT-2 *)
Clear[n, nor, kj, DCT3, T3];
n=8; nor=Sqrt[2/n];
kj[i_]:=If[i==0 || i==n, 1/Sqrt[2], 1];
DCT3[k_]:=Table[nor kj[j] Cos[(k+1/2)j Pi/n], {j,0,n-1}]
T3=Table[DCT3[k], {k,0,n-1}]; (* Compute nxn cosines *)
MatrixForm[T3] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T3[[i]].T3[[j]]]], {i,1,n}, {j,1,n}]]

(* DCT-4. This is DCT-1 shifted *)
Clear[n, nor, DCT4, T4];
n=8; nor=Sqrt[2/n];
DCT4[k_]:=Table[nor Cos[(k+1/2)(j+1/2) Pi/n], {j,0,n-1}]
T4=Table[DCT4[k], {k,0,n-1}]; (* Compute nxn cosines *)
MatrixForm[T4] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T4[[i]].T4[[j]]]], {i,1,n}, {j,1,n}]]
```

Figure 24.30: Code for Four DCT Types.

From the dictionary

Exegete (EK-suh-jeet), noun: A person who explains or interprets difficult parts of written works.

1. Regardless of the image size, only 32 cosine functions are involved. They can be precomputed once and used as needed to calculate all the 8×8 data units. Calculating the expression

$$p_{xy} \cos\left[\frac{(2x+1)i\pi}{16}\right] \cos\left[\frac{(2y+1)j\pi}{16}\right]$$

now amounts to performing two multiplications. The double sum of Equation (24.13) therefore requires $64 \times 2 = 128$ multiplications and 63 additions.

◇ **Exercise 24.8:** Why are only 32 different cosine functions needed for the DCT?

2. A little algebraic tinkering shows that the double sum of Equation (24.13) can be written as the matrix product $\mathbf{CPC}^T$, where $\mathbf{P}$ is the 8×8 matrix of the pixels, $\mathbf{C}$ is the matrix defined by

$$C_{ij} = \begin{cases} \frac{1}{\sqrt{8}}, & i = 0 \\ \frac{1}{2}\cos\left[\frac{(2j+1)i\pi}{16}\right], & i > 0, \end{cases} \tag{24.18}$$

and $\mathbf{C}^T$ is the transpose of $\mathbf{C}$. (The product of two matrices $\mathbf{A}_{mp}$ and $\mathbf{B}_{pn}$ is a matrix $\mathbf{C}_{mn}$ defined by

$$C_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

For other properties of matrices, see any text on linear algebra.)

Calculating one matrix element of the product $\mathbf{CP}$ therefore requires eight multiplications and seven (but for simplicity let's say eight) additions. Multiplying the two 8×8 matrices $\mathbf{C}$ and $\mathbf{P}$ requires $64\times 8 = 8^3$ multiplications and the same number of additions. Multiplying the product $\mathbf{CP}$ by $\mathbf{C}^T$ requires the same number of operations, so the DCT of one 8×8 data unit requires 2×8^3 multiplications (and the same number of additions). Assuming that the entire image consists of $n\times n$ pixels and that $n = 8q$, there are $q\times q$ data units, so the DCT of all the data units requires $2q^2 8^3$ multiplications (and the same number of additions). In comparison, performing one DCT for the entire image would require $2n^3 = 2q^3 8^3 = (2q^2 8^3)q$ operations. By dividing the image into data units, we reduce the number of multiplications (and also of additions) by a factor of q. Unfortunately, q cannot be too large, because that would mean very small data units.

Recall that a color image consists of three components (often RGB, but sometimes YCbCr or YPbPr). In JPEG, the DCT is applied to each component separately, bringing the total number of arithmetic operations to $3\times 2q^2 8^3 = 3{,}072q^2$. For a 512×512-pixel image, this implies $3072 \times 64^2 = 12{,}582{,}912$ multiplications (and the same number of additions).

3. Another way to speed up the DCT is to perform all the arithmetic operations on fixed-point (scaled integer) rather than on floating-point numbers. On many computers, operations on fixed-point numbers require (somewhat) sophisticated programming techniques, but they are considerably faster than floating-point operations (except on supercomputers, which are optimized for floating-point arithmetic).

The DCT algorithm with smallest currently-known number of arithmetic operations is described in [Feig and Linzer 90]. Today, there are also various VLSI chips that perform this calculation efficiently.

24.3.6 The LLM Method

This section describes the Loeffler–Ligtenberg–Moschytz (LLM) method for the DCT in one dimension [Loeffler et al. 89]. Developed in 1989 by Christoph Loeffler, Adriaan Ligtenberg, and George S. Moschytz, this algorithm computes the DCT in one dimension with a total of 29 additions and 11 multiplications. Recall that the DCT in one dimension involves multiplying a row vector by a matrix. For $n = 8$, multiplying the row by one column of the matrix requires eight multiplications and seven additions, so the total number of operations required for the entire operation is 64 multiplications and 56 additions. Reducing the number of multiplications from 64 to 11 represents a savings of 83% and reducing the number of additions from 56 to 29 represents a savings of 49%—very significant!

Only the final result is listed here and the interested reader is referred to the original publication for the details. We start with the double sum of Equation (24.13) and claim that a little algebraic tinkering reduces it to the form $\mathbf{CPC}^T$, where $\mathbf{P}$ is the 8×8 matrix of the pixels, $\mathbf{C}$ is the matrix defined by Equation (24.18), and $\mathbf{C}^T$ is the transpose of $\mathbf{C}$. In the one-dimensional case, only one matrix multiplication, namely $\mathbf{PC}$, is needed. The originators of this method show that matrix $\mathbf{C}$ can be written (up to a factor of $\sqrt{8}$) as the product of seven simple matrices, as shown in Figure 24.32.

Even though the number of matrices has been increased, the problem has been simplified, because our seven matrices are sparse and contain mostly 1's and -1's. Multiplying by 1 or by -1 does not require a multiplication, and multiplying something by 0 saves an addition. Table 24.31 summarizes the total number of arithmetic operations required to multiply a row vector by the seven matrices.

Matrix	Additions	Multiplications
$\mathbf{C}_1$	0	0
$\mathbf{C}_2$	8	12
$\mathbf{C}_3$	4	0
$\mathbf{C}_4$	2	0
$\mathbf{C}_5$	0	2
$\mathbf{C}_6$	4	0
$\mathbf{C}_7$	8	0
Total	26	14

Table 24.31: Number of Arithmetic Operations.

These surprisingly small numbers can be reduced further by the following observation. We notice that matrix $\mathbf{C}_2$ has three groups of four cosines each. One of the groups consists of (we ignore the $\sqrt{2}$) two $\cos\frac{6}{16}\pi$ and two $\cos\frac{2}{16}\pi$ (one with a negative sign). We use the trigonometric identity $\cos(\frac{\pi}{2}-\alpha) = \sin\alpha$ to replace the two $\pm\cos\frac{2}{16}\pi$ with $\pm\sin\frac{6}{16}\pi$. Multiplying any matrix by $\mathbf{C}_2$ now results in products of the form $A\cos(\frac{6}{16}\pi) - B\sin(\frac{6}{16}\pi)$ and $B\cos(\frac{6}{16}\pi) + A\sin(\frac{6}{16}\pi)$. It seems that computing

$$
\mathbf{C} = \begin{bmatrix}
1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0\\
0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&1
\end{bmatrix}
$$

$$
\times \begin{bmatrix}
1&1&0&0&0&0&0&0\\
1&-1&0&0&0&0&0&0\\
0&0&\sqrt{2}\cos\frac{6\pi}{16}&\sqrt{2}\cos\frac{2\pi}{16}&0&0&0&0\\
0&0&-\sqrt{2}\cos\frac{2\pi}{16}&\sqrt{2}\cos\frac{6\pi}{16}&0&0&0&0\\
0&0&0&0&\sqrt{2}\cos\frac{7\pi}{16}&0&0&\sqrt{2}\cos\frac{\pi}{16}\\
0&0&0&0&0&\sqrt{2}\cos\frac{3\pi}{16}&\sqrt{2}\cos\frac{5\pi}{16}&0\\
0&0&0&0&0&-\sqrt{2}\cos\frac{5\pi}{16}&\sqrt{2}\cos\frac{3\pi}{16}&0\\
0&0&0&0&-\sqrt{2}\cos\frac{\pi}{16}&0&0&\sqrt{2}\cos\frac{7\pi}{16}
\end{bmatrix}
$$

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

$$
\times \begin{bmatrix}
1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0\\
0&0&0&0&0&1/\sqrt{2}&0&0\\
0&0&0&0&0&0&1/\sqrt{2}&0\\
0&0&0&0&0&0&0&1
\end{bmatrix}
\times \begin{bmatrix}
1&0&0&1&0&0&0&0\\
0&1&1&0&0&0&0&0\\
0&1&-1&0&0&0&0&0\\
1&0&0&-1&0&0&0&0\\
0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1
\end{bmatrix}
$$

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

$$= \mathbf{C_1C_2C_3C_4C_5C_6C_7}.$$

Figure 24.32: Product of Seven Matrices.

these two elements requires four multiplications and two additions (assuming that a subtraction takes the same time to execute as an addition). The following computation, however, yields the same result with three additions and three multiplications:

$$T = (A+B)\cos\alpha, \quad T - B(\cos\alpha - \sin\alpha), \quad -T + A(\cos\alpha + \sin\alpha).$$

Thus, the three groups now require nine additions and nine multiplications instead of the original six additions and 12 multiplications (two more additions are needed for the other nonzero elements of $\mathbf{C}_2$), which brings the totals of Table 24.31 down to 29 additions and 11 multiplications.

There is no national science just as there is no national multiplication table; what is national is no longer science.

—Anton Chekhov.

24.3.7 Hardware Implementation of the DCT

Table 24.29 lists the 64 angle values of the DCT-2 for $n = 8$. When the cosines of those angles are computed, we find that because of the symmetry of the cosine function, there are only six distinct nontrivial cosine values. They are summarized in Table 24.33, where $a = 1/\sqrt{2}$, $b_i = \cos(i\pi/16)$, and $c_i = \cos(i\pi/8)$. The six nontrivial values are b_1, b_3, b_5, b_7, c_1, and c_3.

1	1	1	1	1	1	1	1
b_1	b_3	b_5	b_7	$-b_7$	$-b_5$	$-b_3$	$-b_1$
c_1	c_3	$-c_3$	$-c_1$	$-c_1$	$-c_3$	c_3	c_1
b_3	$-b_7$	$-b_1$	$-b_5$	b_5	b_1	b_7	$-b_3$
a	$-a$	$-a$	a	a	$-a$	$-a$	a
b_5	$-b_1$	b_7	b_3	$-b_3$	$-b_7$	b_1	$-b_5$
c_3	$-c_1$	c_1	$-c_3$	$-c_3$	c_1	$-c_1$	c_3
b_7	$-b_5$	b_3	$-b_1$	b_1	$-b_3$	b_5	$-b_7$

Table 24.33: Six Distinct Cosine Values for the DCT-2.

This feature can be exploited in a fast software implementation of the DCT or to make a simple hardware device to compute the DCT coefficients G_i for eight pixel values p_i. Figure 24.34 shows how such a device may be organized in two parts, each computing four of the eight coefficients G_i. Part I is based on a 4×4 symmetric matrix whose elements are the four distinct b_i's. The eight pixels are divided into four groups of two pixels each. The two pixels of each group are subtracted, and the four differences become a row vector that's multiplied by the four columns of the matrix to produce the four DCT coefficients G_1, G_3, G_5, and G_7. Part II is based on a similar 4×4 matrix whose nontrivial elements are the two c_i's. The computations are similar except that the two pixels of each group are added instead of subtracted.

$$\big[(p_0-p_7),(p_1-p_6),(p_2-p_5),(p_3-p_4)\big]\begin{bmatrix} b_1 & b_3 & b_5 & b_7 \\ b_3 & -b_7 & -b_1 & -b_5 \\ b_5 & -b_1 & b_7 & b_3 \\ b_7 & -b_5 & b_3 & -b_1 \end{bmatrix} \rightarrow [G_1, G_3, G_5, G_7], \tag{I}$$

$$\big[(p_0+p_7),(p_1+p_6),(p_2+p_5),(p_3+p_4)\big]\begin{bmatrix}1 & c_1 & a & c_3\\ 1 & c_3 & -a & -c_1\\ 1 & -c_3 & -a & c_1\\ 1 & -c_1 & a & -c_3\end{bmatrix}\rightarrow[G_0,G_2,G_4,G_6]. \tag{II}$$

Figure 24.34: A Hardware Implementation of the DCT-2.

Figure 24.35 (after [Chen et al. 77]) illustrates how such a device can be constructed out of simple adders, complementors, and multipliers. Notation such as $c(3\pi/16)$ in the figure refers to the cosine function.

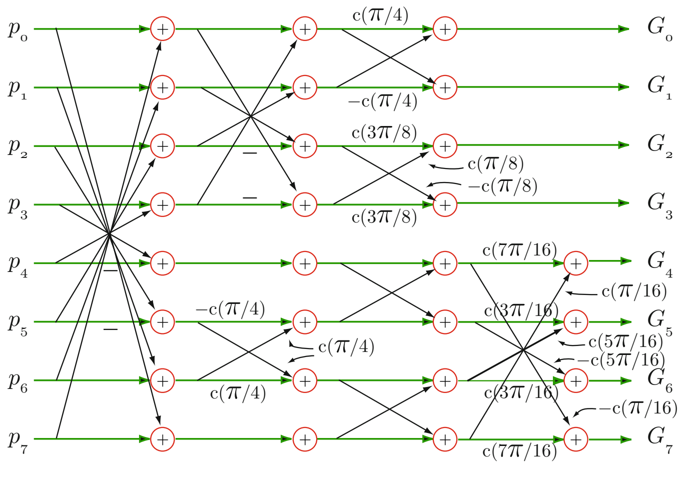


Figure 24.35: A Hardware Implementation of the DCT-2.

24.3.8 QR Matrix Decomposition

This section provides background material on the technique of QR matrix decomposition. It is intended for those already familiar with matrices who want to master this method.

Any matrix $\mathbf{A}$ can be factored into the matrix product $\mathbf{Q}\times\mathbf{R}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is upper triangular. If $\mathbf{A}$ is also orthogonal, then $\mathbf{R}$ will also be orthogonal. However, an upper triangular matrix that's also orthogonal must be diagonal. The orthogonality of $\mathbf{R}$ implies $\mathbf{R}^{-1}=\mathbf{R}^T$ and its being diagonal implies $\mathbf{R}^{-1}\times\mathbf{R}=\mathbf{I}$. The conclusion is that if $\mathbf{A}$ is orthogonal, then $\mathbf{R}$ must satisfy $\mathbf{R}^T\times\mathbf{R}=\mathbf{I}$, which means that its diagonal elements must be $+1$ or -1. If $\mathbf{A}=\mathbf{Q}\times\mathbf{R}$ and $\mathbf{R}$ has this form, then $\mathbf{A}$ and $\mathbf{Q}$ are identical, except that columns i of $\mathbf{A}$ and $\mathbf{Q}$ will have opposite signs for all values of i where $\mathbf{R}_{i,i}=-1$.

The QR decomposition of matrix $\mathbf{A}$ into $\mathbf{Q}$ and $\mathbf{R}$ is done by a loop where each iteration converts one element of $\mathbf{A}$ to zero. When all the below-diagonal elements of $\mathbf{A}$ have been zeroed, it becomes the upper triangular matrix $\mathbf{R}$. Each element $\mathbf{A}_{i,j}$ is zeroed by multiplying $\mathbf{A}$ by a **Givens rotation** matrix $T_{i,j}$ (Section 4.4.4). This is an antisymmetric matrix where the two diagonal elements $T_{i,i}$ and $T_{j,j}$ are set to the cosine of a certain angle θ, and the two off-diagonal elements $T_{j,i}$ and $T_{i,j}$ are set to the sine and negative sine, respectively, of the same θ. The sine and cosine of θ are defined as

$$\cos\theta = \frac{\mathbf{A}_{j,j}}{D}, \quad \sin\theta = \frac{\mathbf{A}_{i,j}}{D}, \quad \text{where } D = \sqrt{\mathbf{A}_{j,j}^2 + \mathbf{A}_{i,j}^2}.$$

Following are some examples of Givens rotation matrices:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & 0 & s \\ 0 & 0 & 1 & 0 \\ 0 & -s & 0 & c \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & s & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -s & 0 & c & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{24.19}$$

Those familiar with rotation matrices will recognize that a Givens matrix [Givens 58] rotates a point through an angle whose sine and cosine are the s and c of Equation (24.19). In two dimensions, the rotation is done about the origin. In three dimensions, it is done about one of the coordinate axes (the x axis in Equation (24.19)). In four dimensions, the rotation is about two of the four coordinate axes (the first and third in Equation (24.19)) and cannot be visualized. In general, an $n\times n$ Givens matrix rotates a point about $n-2$ coordinate axes of an n-dimensional space.

Figure 24.36 is a Matlab function for the QR decomposition of a matrix **A**. Notice how **Q** is obtained as the product of the individual Givens matrices and how the double loop zeros all the below-diagonal elements column by column from the bottom up.

```
function [Q,R]=QRdecompose(A);
% Computes the QR decomposition of matrix A
% R is an upper triangular matrix and Q
% an orthogonal matrix such that A=Q*R.
[m,n]=size(A);  % determine the dimens of A
Q=eye(m);  % Q starts as the mxm identity matrix
R=A;
for p=1:n
 for q=(1+p):m
  w=sqrt(R(p,p)^2+R(q,p)^2);
  s=-R(q,p)/w; c=R(p,p)/w;
  U=eye(m);  % Construct a U matrix for Givens rotation
  U(p,p)=c; U(q,p)=-s; U(p,q)=s; U(q,q)=c;
  R=U'*R;  % one Givens rotation
  Q=Q*U;
 end
end
```

Figure 24.36: A Matlab Function for the QR Decomposition of a Matrix.

"Computer!' shouted Zaphod, "rotate angle of vision through oneeighty degrees and don't talk about it!'

—Douglas Adams, *The Hitchhikers Guide to the Galaxy.*

24.3.9 Vector Spaces

The discrete cosine transform can also be interpreted as a change of basis in a vector space from the standard basis to the DCT basis, so this section is a short discussion of vector spaces, their relation to data compression and to the DCT, their bases, and the important operation of change of basis.

An n-dimensional vector space is the set of all vectors of the form $(v_1, v_2, \ldots, v_n)$. We limit the discussion to the case where the v_i's are real numbers. The attribute of vector spaces that makes them important to us is the existence of *bases.* Any vector (a, b, c) in three dimensions can be written as the linear combination

$$(a, b, c) = a(1, 0, 0) + b(0, 1, 0) + c(0, 0, 1) = a\mathbf{i} + b\mathbf{j} + c\mathbf{k},$$

so we say that the set of three vectors $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ forms a basis of the three-dimensional vector space. Notice that the three basis vectors are orthogonal; the dot product of any two of them is zero. They are also orthonormal; the dot product of each with itself is 1. It is convenient to have an orthonormal basis, but this is not a requirement. The basis does not even have to be orthogonal.

The set of three vectors $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ can be extended to any number of dimensions. A basis for an n-dimensional vector space may consist of the n vectors $\mathbf{v}_i$ for $i = 1, 2, \ldots, n$, where element j of vector $\mathbf{v}_i$ is the Kronecker delta function δ_{ij}. This simple basis is the *standard basis* of the n-dimensional vector space. In addition to this basis, the n-dimensional vector space can have other bases. We illustrate two other bases for $n = 8$.

God made the integers, all else is the work of man.
—Leopold Kronecker.

The DCT (unnormalized) basis consists of the eight vectors

$$\begin{gathered}
(1,1,1,1,1,1,1,1), \quad (1,1,1,1,-1,-1,-1,-1),\\
(1,1,-1,-1,-1,-1,1,1), \quad (1,-1,-1,-1,1,1,1,-1),\\
(1,-1,-1,1,1,-1,-1,1), \quad (1,-1,1,1,-1,-1,1,-1),\\
(1,-1,1,-1,-1,1,-1,1), \quad (1,-1,1-1,1,-1,1,-1).
\end{gathered}$$

Notice how their elements correspond to higher and higher frequencies. The (unnormalized) Haar wavelet basis (Section 25.1) consists of the eight vectors

$$\begin{gathered}
(1,1,1,1,1,1,1,1), \quad (1,1,1,1,-1,-1,-1,-1),\\
(1,1,-1,-1,0,0,0,0), \quad (0,0,0,0,1,1,-1,-1),\\
(1,-1,0,0,0,0,0,0), \quad (0,0,1,-1,0,0,0,0),\\
(0,0,0,0,1,-1,0,0), \quad (0,0,0,0,0,0,1,-1).
\end{gathered}$$

To understand why these bases are useful for data compression, recall that our data vectors are images or parts of images. The pixels of an image are normally correlated, but the standard basis takes no advantage of this. The vector of all 1's, on the other hand, is included in the above bases because this single vector is sufficient to express any uniform image. Thus, a group of identical pixels $(v, v, \ldots, v)$ can be represented as the single coefficient v times the vector of all 1's. (The discrete sine transform of Section 24.3.11 is unsuitable for data compression mainly because it does not include this uniform vector.) Basis vector $(1, 1, 1, 1, -1, -1, -1, -1)$ can represent the energy of a group of pixels that's half dark and half bright. Thus, the group $(v, v, \ldots, v, -v, -v, \ldots, -v)$ of pixels is represented by the single coefficient v times this basis vector. Successive basis vectors represent higher-frequency images, up to vector $(1, -1, 1, -1, 1, -1, 1, -1)$. This basis vector resembles a checkerboard and therefore isolates the high-frequency details of an image. Those details are normally the least important and can be heavily quantized or even zeroed to achieve better compression.

The vector members of a basis don't have to be orthogonal. In order for a set S of vectors to be a basis, it has to have the following two properties: (1) The vectors have to be linearly independent and (2) it should be possible to express any member of the vector space as a linear combination of the vectors of S. For example, the three vectors $(1, 1, 1)$, $(0, 1, 0)$, and $(0, 0, 1)$ are not orthogonal but form a basis for the three-dimensional vector space. (1) They are linearly independent because none of them can be expressed as a linear combination of the other two. (2) Any vector (a, b, c) can be expressed as the linear combination $a(1, 1, 1) + (b - a)(0, 1, 0) + (c - a)(0, 0, 1)$.

Once we realize that a vector space may have many bases, we start looking for good bases. A good basis for data compression is one where the inverse of the basis matrix is easy to compute and where the energy of a data vector becomes concentrated in a few coefficients. The bases discussed so far are simple, being based on zeros and ones. The orthogonal bases have the added advantage that the inverse of the basis matrix is simply its transpose. Being fast is not enough, because the fastest thing we could do is to stay with the original standard basis. The reason for changing a basis is to get compression. The DCT base has the added advantage that it concentrates the energy of a vector of correlated values in a few coefficients. Without this property, there would be no reason to change the coefficients of a vector from the standard basis to the DCT basis. After changing to the DCT basis, many coefficients can be quantized, sometimes even zeroed, with a loss of only the least-important image information. If we quantize the original pixel values in the standard basis, we also achieve compression, but we lose image information that may be important.

Once a basis has been selected, it is easy to express any given vector in terms of the basis vectors. Assuming that the basis vectors are $\mathbf{b}_i$ and given an arbitrary vector $\mathbf{P} = (p_1, p_2, \ldots, p_n)$, we write $\mathbf{P}$ as a linear combination $\mathbf{P} = c_1\mathbf{b}_1 + c_2\mathbf{b}_2 + \cdots + c_n\mathbf{b}_n$ of the $\mathbf{b}_i$'s with unknown coefficients c_i. Using matrix notation, this is written $\mathbf{P} = \mathbf{c} \cdot \mathbf{B}$, where $\mathbf{c}$ is a row vector of the coefficients and $\mathbf{B}$ is the matrix whose rows are the basis vectors. The unknown coefficients can be computed by $\mathbf{c} = \mathbf{P} \cdot \mathbf{B}^{-1}$ and this is the reason why a good basis is one where the inverse of the basis matrix is easy to compute.

A simple example is the coefficients of a vector under the standard basis. We have seen that vector (a, b, c) can be written as the linear combination $a(1, 0, 0) + b(0, 1, 0) + c(0, 0, 1)$. Thus, when the standard basis is used, the coefficients of a vector $\mathbf{P}$ are simply its original elements. If we now want to compress the vector by changing to the DCT basis, we need to compute the coefficients under the new basis. This is an example of the important operation of *change of basis*.

Given two bases $\mathbf{b}_i$ and $\mathbf{v}_i$ and assuming that a given vector $\mathbf{P}$ can be expressed as $\sum c_i\mathbf{b}_i$ and also as $\sum w_i\mathbf{v}_i$, the problem of change of basis is to express one set of coefficients in terms of the other. Since the vectors $\mathbf{v}_i$ constitute a basis, any vector can be expressed as a linear combination of them. Specifically, any $\mathbf{b}_j$ can be written $\mathbf{b}_j = \sum_i t_{ij}\mathbf{v}_i$ for some numbers t_{ij}. We now construct a matrix $\mathbf{T}$ from the t_{ij} and observe that it satisfies $\mathbf{b}_i\mathbf{T} = \mathbf{v}_i$ for $i = 1, 2, \ldots, n$. Thus, $\mathbf{T}$ is a *linear transformation* that transforms basis $\mathbf{b}_i$ to $\mathbf{v}_i$. The numbers t_{ij} are the elements of $\mathbf{T}$ in basis $\mathbf{v}_i$.

For our vector $\mathbf{P}$, we can now write $(\sum c_i\mathbf{b}_i)\mathbf{T} = \sum c_i\mathbf{v}_i$, which implies

$$\sum_{j=1}^{n} w_j\mathbf{v}_j = \sum_j w_j\mathbf{b}_j\mathbf{T} = \sum_j w_j\sum_i \mathbf{v}_i t_{ij} = \sum_i\Big(\sum_j w_j t_{ij}\Big)\mathbf{v}_i.$$

This shows that $c_i = \sum_j t_{ij}w_j$; in other words, a basis is changed by means of a linear transformation $\mathbf{T}$ and the same transformation also relates the elements of a vector in the old and new bases.

Once we switch to a new basis in a vector space, every vector has new coordinates and every transformation has a different matrix.

A linear transformation $\mathbf{T}$ operates on an n-dimensional vector and produces an m-dimensional vector. Thus, $\mathbf{T}(\mathbf{v})$ is a vector $\mathbf{u}$. If $m = 1$, the transformation produces a scalar. If $m = n-1$, the transformation is a projection. Linear transformations satisfy the two important properties $\mathbf{T}(\mathbf{u}+\mathbf{v}) = \mathbf{T}(\mathbf{u}) + \mathbf{T}(\mathbf{v})$ and $\mathbf{T}(c\mathbf{v}) = c\mathbf{T}(\mathbf{v})$. In general, the linear transformation of a linear combination $\mathbf{T}(c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n)$ equals the linear combination of the individual transformations $c_1\mathbf{T}(\mathbf{v}_1) + c_2\mathbf{T}(\mathbf{v}_2) + \cdots + c_n\mathbf{T}(\mathbf{v}_n)$. This implies that the zero vector is transformed to itself under any linear transformation.

Examples of linear transformations are projection, reflection, rotation, and differentiating a polynomial. The derivative of $c_1 + c_2x + c_3x^2$ is $c_2 + 2c_3x$. This is a transformation from the basis (c_1, c_2, c_3) in three-dimensional space to basis (c_2, c_3) in two-dimensional space. The transformation matrix satisfies $(c_1, c_2, c_3)\mathbf{T} = (c_2, 2c_3)$, so it is given by

$$\mathbf{T} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}.$$

Examples of nonlinear transformations are translation, the length of a vector, and adding a constant vector v_0. The latter is nonlinear because if $\mathbf{T}(\mathbf{v}) = \mathbf{v} + \mathbf{v}_0$ and we double the size of $\mathbf{v}$, then $\mathbf{T}(2\mathbf{v}) = 2\mathbf{v} + \mathbf{v}_0$ is different from $2\mathbf{T}(\mathbf{v}) = 2(\mathbf{v} + \mathbf{v}_0)$. Transforming a vector $\mathbf{v}$ to its length $||v||$ is also nonlinear because $\mathbf{T}(-\mathbf{v}) \neq -\mathbf{T}(\mathbf{v})$. Translation is nonlinear because it transforms the zero vector to a nonzero vector.

In general, a linear transformation is performed by multiplying the transformed vector $\mathbf{v}$ by the transformation matrix $\mathbf{T}$. Thus, $\mathbf{u} = \mathbf{v}\cdot\mathbf{T}$ or $\mathbf{T}(\mathbf{v}) = \mathbf{v}\cdot\mathbf{T}$. Notice that we denote by $\mathbf{T}$ both the transformation and its matrix.

In order to describe a transformation uniquely, it is enough to describe what it does to the vectors of a basis. To see why this is true, we observe the following. If for a given vector $\mathbf{v}_1$ we know what $\mathbf{T}(\mathbf{v}_1)$ is, then we know what $\mathbf{T}(a\mathbf{v}_1)$ is for any a. Similarly, if for a given $\mathbf{v}_2$ we know what $\mathbf{T}(\mathbf{v}_2)$ is, then we know what $\mathbf{T}(b\mathbf{v}_1)$ is for any b and also what $\mathbf{T}(a\mathbf{v}_1 + b\mathbf{v}_2)$ is. Thus, we know how $\mathbf{T}$ transforms any vector in the plane containing $\mathbf{v}_1$ and $\mathbf{v}_2$. This argument shows that if we know what $\mathbf{T}(\mathbf{v}_i)$ is for all the vectors $\mathbf{v}_i$ of a basis, then we know how $\mathbf{T}$ transforms any vector in the vector space.

Given a basis $\mathbf{b}_i$ for a vector space, we consider the special transformation that affects the magnitude of each vector but not its direction. Thus, $\mathbf{T}(\mathbf{b}_i) = \lambda_i \mathbf{b}_i$ for some number λ_i. The basis $\mathbf{b}_i$ is the *eigenvector* basis of transformation $\mathbf{T}$. Since we know $\mathbf{T}$ for the entire basis, we also know it for any other vector. Any vector $\mathbf{v}$ in the vector space can be expressed as a linear combination $\mathbf{v} = \sum_i c_i \mathbf{b}_i$. If we apply our transformation to both sides and use the linearity property, we end up with

$$\mathbf{T}(\mathbf{v}) = \mathbf{v} \cdot \mathbf{T} = \sum_i c_i \mathbf{b}_i \cdot \mathbf{T}. \tag{24.20}$$

In the special case where $\mathbf{v}$ is the basis vector $\mathbf{b}_1$, Equation (24.20) implies $\mathbf{T}(\mathbf{b}_1) = \sum_i c_i \mathbf{b}_i \cdot \mathbf{T}$. On the other hand, $\mathbf{T}(\mathbf{b}_1) = \lambda_1 \mathbf{b}_1$. We therefore conclude that $c_1 = \lambda_1$ and, in general, that the transformation matrix $\mathbf{T}$ is diagonal with λ_i in position i of its diagonal.

In the eigenvector basis, the transformation matrix is diagonal, so this is the perfect basis. We would love to have it in compression, but it is data dependent. It is called the Karhunen–Loève transform (KLT) and is described in Section 24.2.4.

24.3.10 Rotations in Three Dimensions

For those exegetes who want the complete story, the following paragraphs show how a proper rotation matrix (with a determinant of +1) that rotates a point (v, v, v) to the x axis can be derived from the general rotation matrix in three dimensions (Section 4.4.3).

A general rotation in three dimensions is fully specified by (1) an axis $\mathbf{u}$ of rotation, (2) the angle θ of rotation, and (3) the direction (clockwise or counterclockwise as viewed from the origin) of the rotation about $\mathbf{u}$. Given a unit vector $\mathbf{u} = (u_x, u_y, u_z)$, matrix $\mathbf{M}$ of Equation (24.21) performs a rotation of $\theta°$ about $\mathbf{u}$. The rotation appears clockwise to an observer looking from the origin in the direction of $\mathbf{u}$. If $\mathbf{P} = (x, y, z)$ is an arbitrary point, its position after the rotation is given by the product $\mathbf{P} \cdot \mathbf{M}$.

$$\mathbf{M} = \tag{24.21}$$
$$\begin{pmatrix} u_x^2 + \cos\theta(1 - u_x^2) & u_x u_y(1 - \cos\theta) - u_z \sin\theta & u_x u_z(1 - \cos\theta) + u_y \sin\theta \\ u_x u_y(1 - \cos\theta) + u_z \sin\theta & u_y^2 + \cos\theta(1 - u_y^2) & u_y u_z(1 - \cos\theta) - u_x \sin\theta \\ u_x u_z(1 - \cos\theta) - u_y \sin\theta & u_y u_z(1 - \cos\theta) + u_x \sin\theta & u_z^2 + \cos\theta(1 - u_z^2) \end{pmatrix}.$$

The general rotation of Equation (24.21) can now be applied to our problem, which is to rotate the vector $\mathbf{D} = (1, 1, 1)$ to the x axis. The rotation should be done about the vector $\mathbf{u}$ that's perpendicular to both $\mathbf{D}$ and $(1, 0, 0)$. This vector is computed by the cross-product $\mathbf{u} = \mathbf{D} \times (1, 0, 0) = (0, 1, -1)$. Normalizing it yields $\mathbf{u} = (0, \alpha, -\alpha)$, where $\alpha = 1/\sqrt{2}$.

The next step is to compute the angle θ between $\mathbf{D}$ and the x axis. This is done by normalizing $\mathbf{D}$ and computing the dot product of it and the x axis (recall that the dot product of two unit vectors is the cosine of the angle between them). The normalized $\mathbf{D}$ is (β,β,β), where $\beta = 1/\sqrt{3}$, and the dot product results in $\cos\theta = \beta$, which also produces $\sin\theta = -\sqrt{1-\beta^2} = -\sqrt{2/3} = -\beta/\alpha$. The reason for the negative sign is that a rotation from $(1,1,1)$ to $(1,0,0)$ about $\mathbf{u}$ appears counterclockwise to an observer looking from the origin in the direction of positive $\mathbf{u}$. The rotation matrix of Equation (24.21) was derived for the opposite direction of rotation. Also, $\cos\theta = \beta$ implies that $\theta = 54.76°$. This angle, not 45°, is the angle made by vector $\mathbf{D}$ with each of the three coordinate axes. (As an aside, when the number of dimensions increases, the angle between vector $(1,1,\ldots,1)$ and any of the coordinate axes approaches 90°.)

Substituting $\mathbf{u}$, $\sin\theta$, and $\cos\theta$ in Equation (24.21) and using the relations $\alpha^2 + \beta(1-\alpha^2) = (\beta+1)/2$ and $-\alpha^2(1-\beta) = (\beta-1)/2$ yields the simple rotation matrix

$$\mathbf{M} = \begin{bmatrix} \beta & -\beta & -\beta \\ \beta & \alpha^2+\beta(1-\alpha^2) & -\alpha^2(1-\beta) \\ \beta & -\alpha^2(1-\beta) & \alpha^2+\beta(1-\alpha^2) \end{bmatrix} = \begin{bmatrix} \beta & -\beta & -\beta \\ \beta & (\beta+1)/2 & (\beta-1)/2 \\ \beta & (\beta-1)/2 & (\beta+1)/2 \end{bmatrix}$$
$$\approx \begin{bmatrix} 0.5774 & -0.5774 & -0.5774 \\ 0.5774 & 0.7886 & -0.2115 \\ 0.5774 & -0.2115 & 0.7886 \end{bmatrix}.$$

It is now easy to see that a point on the line $x = y = z$, with coordinates (v,v,v) is rotated by $\mathbf{M}$ to $(v,v,v)\mathbf{M} = (1.7322v, 0, 0)$. Notice that the determinant of $\mathbf{M}$ equals $+1$, so $\mathbf{M}$ is a rotation matrix, in contrast to the matrix of Equation (24.15), which generates improper rotations.

24.3.11 Discrete Sine Transform

Readers who have made it to this point may raise the question of why the cosine function, and not the sine, is used in the transform? Is it possible to use the sine function in a similar way to the DCT to create a discrete sine transform? Is there a DST, and if not, why? This short section discusses the differences between the sine and cosine functions and shows why these differences lead to a very ineffective discrete sine transform.

A function $f(x)$ that satisfies $f(x) = -f(-x)$ is called *odd*. Similarly, a function for which $f(x) = f(-x)$ is called *even*. For an odd function, it is always true that $f(0) = -f(-0) = -f(0)$, so $f(0)$ must be 0. Most functions are neither odd nor even, but the trigonometric functions sine and cosine are important examples of odd and even functions, respectively. Figure 24.37 shows that even though the only difference between them is phase (i.e., the cosine is a shifted version of the sine), this difference is enough to reverse their parity. When the (odd) sine curve is shifted, it becomes the (even) cosine curve, which has the same shape.

To understand the difference between the DCT and the DST, we examine the one-dimensional case. The DCT in one dimension, Equation (24.11), employs the function $\cos[(2t+1)f\pi/16]$ for $f = 0,1,\ldots,7$. For the first term, where $f = 0$, this function becomes $\cos(0)$, which is 1. This term is the familiar and important DC coefficient, which is proportional to the average of the eight data values being transformed. The DST is similarly based on the function $\sin[(2t+1)f\pi/16]$, resulting in a zero first term [since $\sin(0) = 0$]. The first term contributes nothing to the transform, so the DST does not have a DC coefficient.

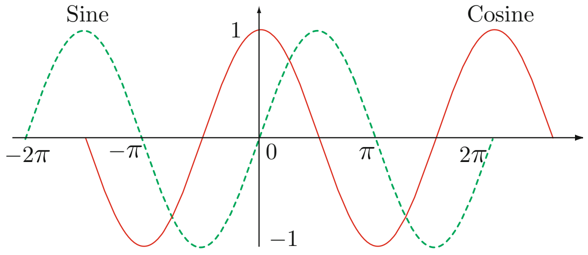


Figure 24.37: The Sine and Cosine as Odd and Even Functions, Respectively.

The disadvantage of this can be seen when we consider the example of eight identical data values being transformed by the DCT and by the DST. Identical values are, of course, perfectly correlated. When plotted, they become a horizontal line. Applying the DCT to these values produces just a DC coefficient: All the AC coefficients are zero. The DCT compacts all the energy of the data into the single DC coefficient whose value is identical to the values of the data items. The IDCT can reconstruct the eight values perfectly (except for minor differences resulting from limited machine precision). Applying the DST to the same eight values, on the other hand, results in seven AC coefficients whose sum is a wave function that passes through the eight data points but oscillates between the points. This behavior, illustrated by Figure 24.38, has three disadvantages, namely (1) the energy of the original data values is not compacted, (2) the seven coefficients are not decorrelated (since the data values are perfectly correlated), and (3) quantizing the seven coefficients may greatly reduce the quality of the reconstruction done by the inverse DST.

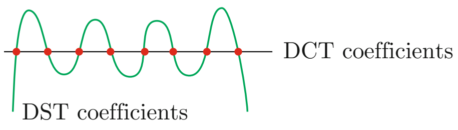
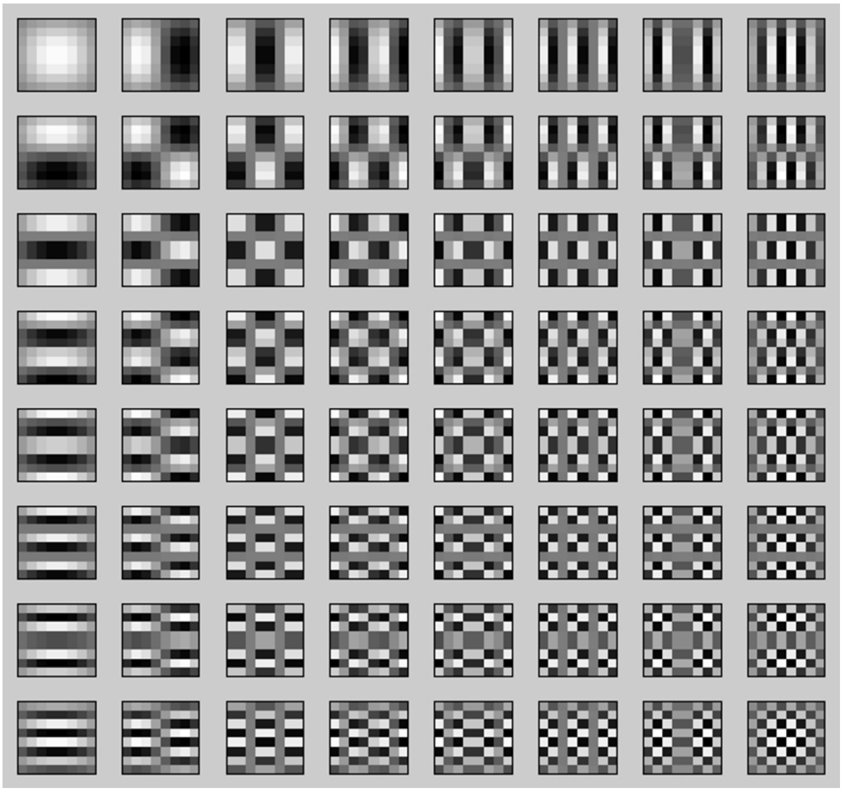


Figure 24.38: The DCT and DST of Eight Identical Data Values.

Example: Applying the DST to the eight identical values 100 results in the eight coefficients $(0, 256.3, 0, 90, 0, 60.1, 0, 51)$. Using these coefficients, the IDST can reconstruct the original values, but it is easy to see that the AC coefficients do not behave like those of the DCT. They are not getting smaller, and there are no runs of zeros among them. Applying the DST to the eight highly correlated values 11, 22, 33, 44, 55, 66, 77, and 88 results in the even worse set of coefficients

$$(0, 126.9, -57.5, 44.5, -31.1, 29.8, -23.8, 25.2).$$

There is no energy compaction at all.



```
N=8;
m=[1:N]'*ones(1,N); n=m';
% can also use cos instead of sin
%A=sqrt(2/N)*cos(pi*(2*(n-1)+1).*(m-1)/(2*N));
A=sqrt(2/N)*sin(pi*(2*(n-1)+1).*(m-1)/(2*N));
A(1,:)=sqrt(1/N);
C=A';
for row=1:N
  for col=1:N
    B=C(:,row)*C(:,col).'; %tensor product
    subplot(N,N,(row-1)*N+col)
    imagesc(B)
    drawnow
  end
end
```

Figure 24.39: The 64 Basis Images of the DST in Two Dimensions.

These arguments and examples, together with the fact (discussed in [Ahmed et al. 74] and [Rao and Yip 90]) that the DCT produces highly decorrelated coefficients, argue strongly for the use of the DCT as opposed to the DST in data compression.

◇ **Exercise 24.9:** Use mathematical software to compute and display the 64 basis images of the DST in two dimensions for $n = 8$.

We are the wisp of straw, the plaything of the winds. We think that we are making for a goal deliberately chosen; destiny drives us towards another. Mathematics, the exaggerated preoccupation of my youth, did me hardly any service; and animals, which I avoided as much as ever I could, are the consolation of my old age. Nevertheless, I bear no grudge against the sine and the cosine, which I continue to hold in high esteem. They cost me many a pallid hour at one time, but they always afforded me some first rate entertainment: they still do so, when my head lies tossing sleeplessly on its pillow.

—J. Henri Fabre, *The Life of the Fly.*

24.4 Test Images

New image compression methods that are developed and implemented have to be tested. Testing different methods on the same data makes it possible to compare their performance both in compression efficiency and in speed. This is why there are standard collections of test data, such as the Calgary Corpus [Calgary 11], the Canterbury Corpus [Canterbury 11], and the ITU-T set of eight training documents for fax compression [funet 11].

In addition to these sets of test data, there currently exist collections of still images commonly used by researchers and implementors in the fields of image compression and image processing. Three of the four images shown here, namely *Lena*, *mandril*, and *peppers*, are arguably the most well-known of them. They are continuous-tone images, although *Lena* has some features of a discrete-tone image.

Each image is accompanied by a detail, showing individual pixels (see also Figure 21.13). It is easy to see why the *peppers* image is continuous-tone. Adjacent pixels that differ much in color are fairly rare in this image. Most neighboring pixels are very similar. In contrast, the *mandril* image, even though natural, is a bad example of a continuous-tone image. The detail (showing part of the right eye and the area around it) shows that many pixels differ considerably from their immediate neighbors because of the animal's facial hair in this area. This image compresses badly under any compression method. However, the nose area, with mostly blue and red, is continuous-tone. The *Lena* image is mostly pure continuous-tone, especially the wall and the bare skin areas. The hat is good continuous-tone, whereas the hair and the plume on the hat are bad continuous-tone. The straight lines on the wall and the curved parts of the mirror are features of a discrete-tone image.

The *Lena* image is widely used by the image processing community, in addition to being popular in image compression. Because of the interest in it, its origin and history have been researched and are well documented. This image is part of the *Playboy*

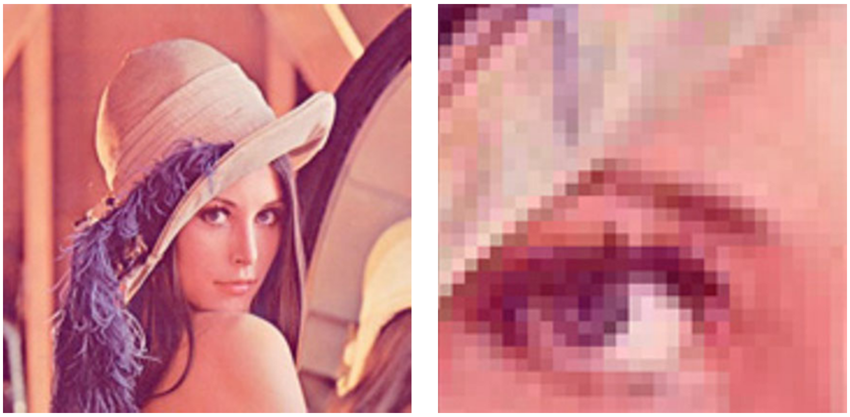


Figure 24.40: Lena and Detail.

centerfold for November, 1972. It features the Swedish playmate Lena Soderberg (née Sjooblom), and it was discovered, clipped, and scanned in the early 1970s by Alexander Sawchuk, an assistant professor at the University of Southern California for use as a test image for his image compression research. It has since become the most important, well-known, and commonly used image in the history of imaging and electronic communications. As a result, Lena is currently considered by many the First Lady of the Internet. *Playboy*, which normally prosecutes unauthorized users of its images, has found out about the unusual use of one of its copyrighted images, but decided to give its blessing to this particular "application."

Lena herself currently lives in Sweden. She was told of her "fame" in 1988, was surprised and amused by it, and was invited to attend the 50th Anniversary IS&T (the society for Imaging Science and Technology) conference in Boston in May 1997. At the conference she autographed her picture, posed for new pictures (available on the www), and gave a presentation (about herself, not image processing).

The three images are widely available for downloading on the Internet.

Figure 24.44 shows a typical discrete-tone image, with a detail shown in Figure 24.45. Notice the straight lines and the text, where certain characters appear several times (a source of redundancy). This particular image has few colors, but in general, a discrete-tone image may have many colors.

Lena, Illinois, is a community of approximately 2,900 people. Lena is considered to be a clean and safe community located centrally to larger cities that offer other interests when needed. Lena is 2-1/2 miles from Lake Le-Aqua-Na State Park. The park offers hiking trails, fishing, swimming beach, boats, cross country skiing, horse back riding trails, as well as picnic and camping areas. It is a beautiful well-kept park that has free admission to the public. A great place for sledding and ice skating in the winter! (From `http://www.villageoflena.com/`)

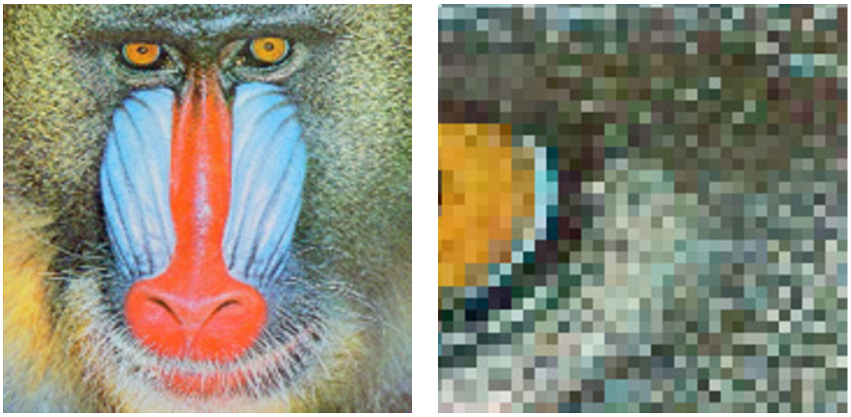


Figure 24.41: Mandril and Detail.

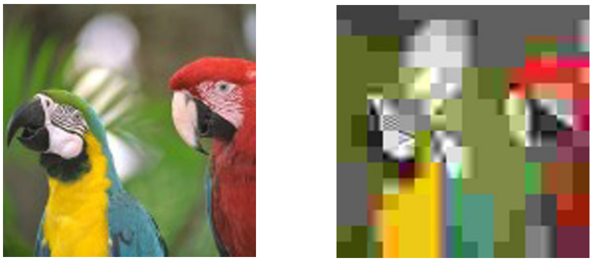


Figure 24.42: JPEG Blocking Artifacts.

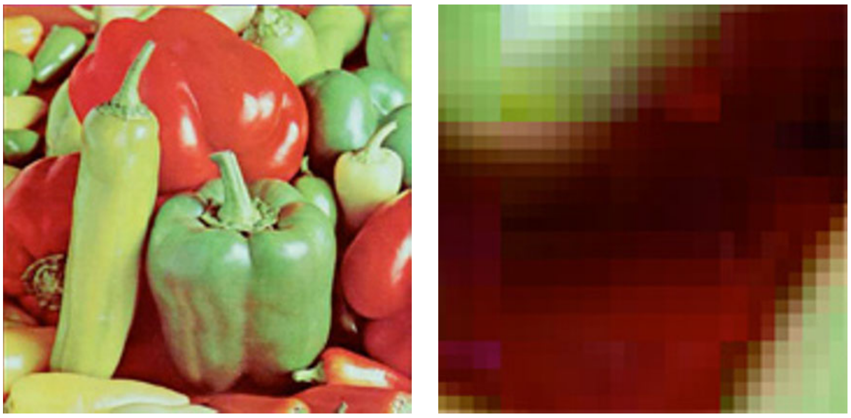


Figure 24.43: Peppers and Detail.

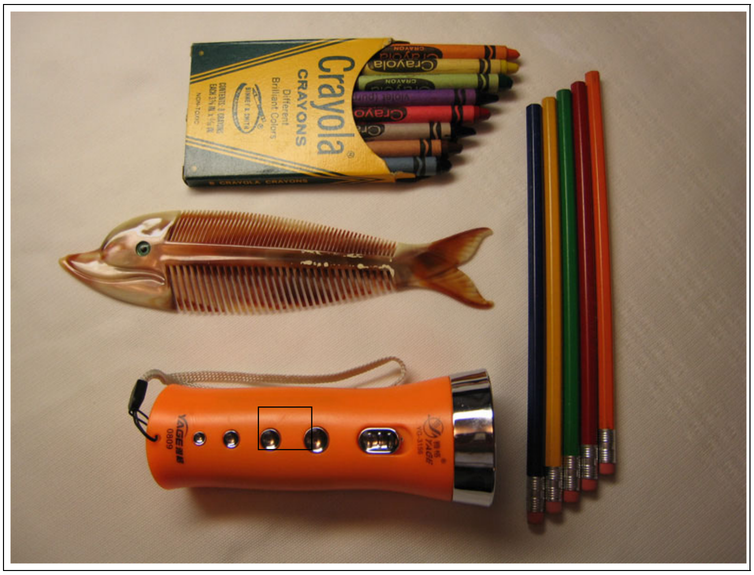


Figure 24.44: A Discrete-Tone Image.

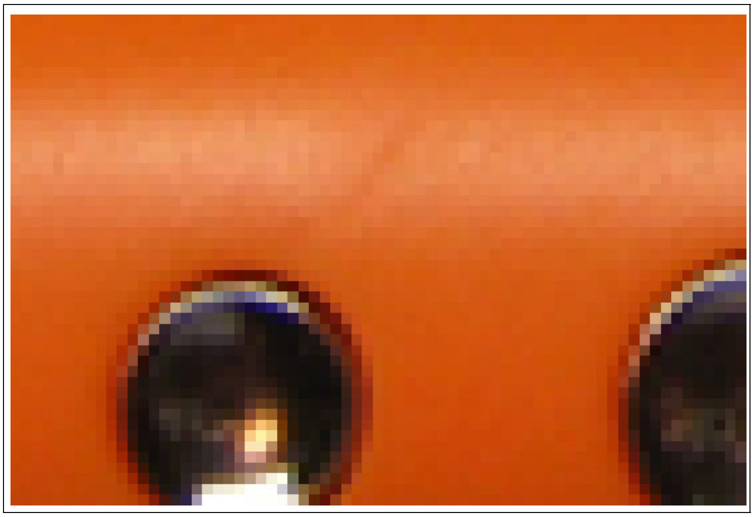


Figure 24.45: A Discrete-Tone Image (Detail).

24.5 JPEG

JPEG is a sophisticated lossy/lossless compression method for color or grayscale still images (not videos). It does not handle bi-level (black and white) images very well. It also performs best on continuous-tone images, where adjacent pixels have similar colors. An important feature of JPEG is its use of many parameters, allowing the user to adjust the amount of the data lost (and thus also the compression ratio) over a very wide range. Often, the eye cannot see any image degradation even at compression factors of 10 or 20. There are two operating modes, lossy (also called baseline) and lossless (which typically produces compression ratios of around 0.5). Most implementations support just the lossy mode. This mode includes progressive and hierarchical coding. A few of the many references to JPEG are [Pennebaker and Mitchell 92], [Wallace 91], and [Zhang 90].

JPEG is a compression method, not a complete standard for image representation. This is why it does not specify image features such as pixel aspect ratio, color space, or interleaving of bitmap rows.

JPEG has been designed as a compression method for continuous-tone images. The main goals of JPEG compression are the following:

1. High compression ratios, especially in cases where image quality is judged as very good to excellent.
2. The use of many parameters, allowing knowledgeable users to experiment and achieve the desired compression/quality trade-off.
3. Obtaining good results with any kind of continuous-tone image, regardless of image dimensions, color spaces, pixel aspect ratios, or other image features.
4. A sophisticated, but not too complex compression method, allowing software and hardware implementations on many platforms.
5. Several modes of operation: (a) A sequential mode where each image component (color) is compressed in a single left-to-right, top-to-bottom scan; (b) a progressive mode where the image is compressed in multiple blocks (known as "scans") to be viewed from coarse to fine detail; (c) a lossless mode that is important in cases where the user decides that no pixels should be lost (the trade-off is low compression ratio compared to the lossy modes); and (d) a hierarchical mode where the image is compressed at multiple resolutions allowing lower-resolution blocks to be viewed without first having to decompress the following higher-resolution blocks.

The name JPEG is an acronym that stands for Joint Photographic Experts Group. This was a joint effort by the CCITT and the ISO (the International Standards Organization) that started in June 1987 and produced the first JPEG draft proposal in 1991. The JPEG standard has proved successful and has become widely used for image compression, especially in Web pages.

The main JPEG compression steps are outlined here, and each step is then described in detail later.

1. Color images are transformed from RGB into a luminance-chrominance color space (Section 21.12; this step is skipped for grayscale images). The eye is sensitive to small changes in luminance but not in chrominance, so the chrominance part can later lose much data, and thus be highly compressed, without visually impairing the overall image quality much. This step is optional but important because the remainder of the algo-

rithm works on each color component separately. Without transforming the color space, none of the three color components will tolerate much loss, leading to worse compression.

2. Color images are downsampled by creating low-resolution pixels from the original ones (this step is used only when hierarchical compression is selected; it is always skipped for grayscale images). The downsampling is not done for the luminance component. Downsampling is done either at a ratio of 2:1 both horizontally and vertically (the so-called `2h2v` or 4:1:1 sampling) or at ratios of 2:1 horizontally and 1:1 vertically (`2h1v` or 4:2:2 sampling). Since this is done on two of the three color components, `2h2v` reduces the image to $1/3 + (2/3) \times (1/4) = 1/2$ its original size, while `2h1v` reduces it to $1/3 + (2/3) \times (1/2) = 2/3$ its original size. Since the luminance component is not touched, there is no noticeable loss of image quality. Grayscale images don't go through this step.

3. The pixels of each color component are organized in groups of 8×8 pixels called *data units*, and each data unit is compressed separately. If the number of image rows or columns is not a multiple of 8, the bottom row and the rightmost column are duplicated as many times as necessary. In the noninterleaved mode, the encoder handles all the data units of the first image component, then the data units of the second component, and finally those of the third component. In the interleaved mode the encoder processes the three top-left data units of the three image components, then the three data units to their right, and so on. The fact that each data unit is compressed separately is one of the downsides of JPEG. If the user asks for maximum compression, the decompressed image may exhibit blocking artifacts due to differences between blocks. Figure 24.42 is an extreme example of this effect.

4. The discrete cosine transform (DCT, Section 24.3) is then applied to each data unit to create an 8×8 map of frequency components (Section 24.5.1). They represent the average pixel value and successive higher-frequency changes within the group. This prepares the image data for the crucial step of losing information. Since DCT involves the transcendental function cosine, it must involve some loss of information due to the limited precision of computer arithmetic. This means that even without the main lossy step (step 5 below), there will be some loss of image quality, but it is normally small.

5. Each of the 64 frequency components in a data unit is divided by a separate number called its *quantization coefficient* (QC), and then rounded to an integer (Section 24.5.2). This is where information is irretrievably lost. Large QCs cause more loss, so the high-frequency components typically have larger QCs. Each of the 64 QCs is a JPEG parameter and can, in principle, be specified by the user. In practice, most JPEG implementations use the QC tables recommended by the JPEG standard for the luminance and chrominance image components (Table 24.47).

6. The 64 quantized frequency coefficients (which are now integers) of each data unit are encoded using a combination of RLE and Huffman coding (Section 24.5.3). An arithmetic coding variant known as the QM coder (see [Salomon 09]) can optionally be used instead of Huffman coding.

7. The last step adds headers and all the required JPEG parameters, and outputs the result. The compressed file may be in one of three formats (1) the *interchange* format, in which the file contains the compressed image and all the tables needed by the decoder (mostly quantization tables and tables of Huffman codes), (2) the *abbreviated* format for compressed image data, where the file contains the compressed image and may contain

no tables (or just a few tables), and (3) the *abbreviated* format for table-specification data, where the file contains just tables, and no compressed image. The second format makes sense in cases where the same encoder/decoder pair is used, and they have the same tables built in. The third format is used in cases where many images have been compressed by the same encoder, using the same tables. When those images need to be decompressed, they are sent to a decoder preceded by one file with table-specification data.

The JPEG decoder performs the reverse steps. (Thus, JPEG is a symmetric compression method.)

The progressive mode is a JPEG option. In this mode, higher-frequency DCT coefficients are written on the compressed stream in blocks called "scans." Each scan that is read and processed by the decoder results in a sharper image. The idea is to use the first few scans to quickly create a low-quality, blurred preview of the image, and then either input the remaining scans or stop the process and reject the image. The trade-off is that the encoder has to save all the coefficients of all the data units in a memory buffer before they are sent in scans, and also go through all the steps for each scan, slowing down the progressive mode.

Figure 24.46a shows an example of an image with resolution 1024×512. The image is divided into $128 \times 64 = 8192$ data units, and each is transformed by the DCT, becoming a set of 64 8-bit numbers. Figure 24.46b is a block whose depth corresponds to the 8,192 data units, whose height corresponds to the 64 DCT coefficients (the DC coefficient is the top one, numbered 0), and whose width corresponds to the eight bits of each coefficient.

After preparing all the data units in a memory buffer, the encoder writes them on the compressed stream in one of two methods, *spectral selection* or *successive approximation* (Figure 24.46c,d). The first scan in either method is the set of DC coefficients. If spectral selection is used, each successive scan consists of several consecutive (a *band* of) AC coefficients. If successive approximation is used, the second scan consists of the four most-significant bits of all AC coefficients, and each of the following four scans, numbers 3 through 6, adds one more significant bit (bits 3 through 0, respectively).

In the hierarchical mode, the encoder stores the image several times in the output stream, at several resolutions. However, each high-resolution part uses information from the low-resolution parts of the output stream, so the total amount of information is less than that required to store the different resolutions separately. Each hierarchical part may use the progressive mode.

The hierarchical mode is useful in cases where a high-resolution image needs to be output in low resolution. Today, in 2011, it is difficult to come up with an example of a low-resolution output device, but there may be places where a few old, obsolete dot-matrix printers are still in use.

The lossless mode of JPEG (Section 24.5.4) calculates a "predicted" value for each pixel, generates the difference between the pixel and its predicted value, and encodes the difference using the same method (i.e., Huffman or arithmetic coding) employed by step 5 above. The predicted value is calculated using values of pixels above and to the left of the current pixel (pixels that have already been input and encoded). The following sections discuss the steps in more detail:

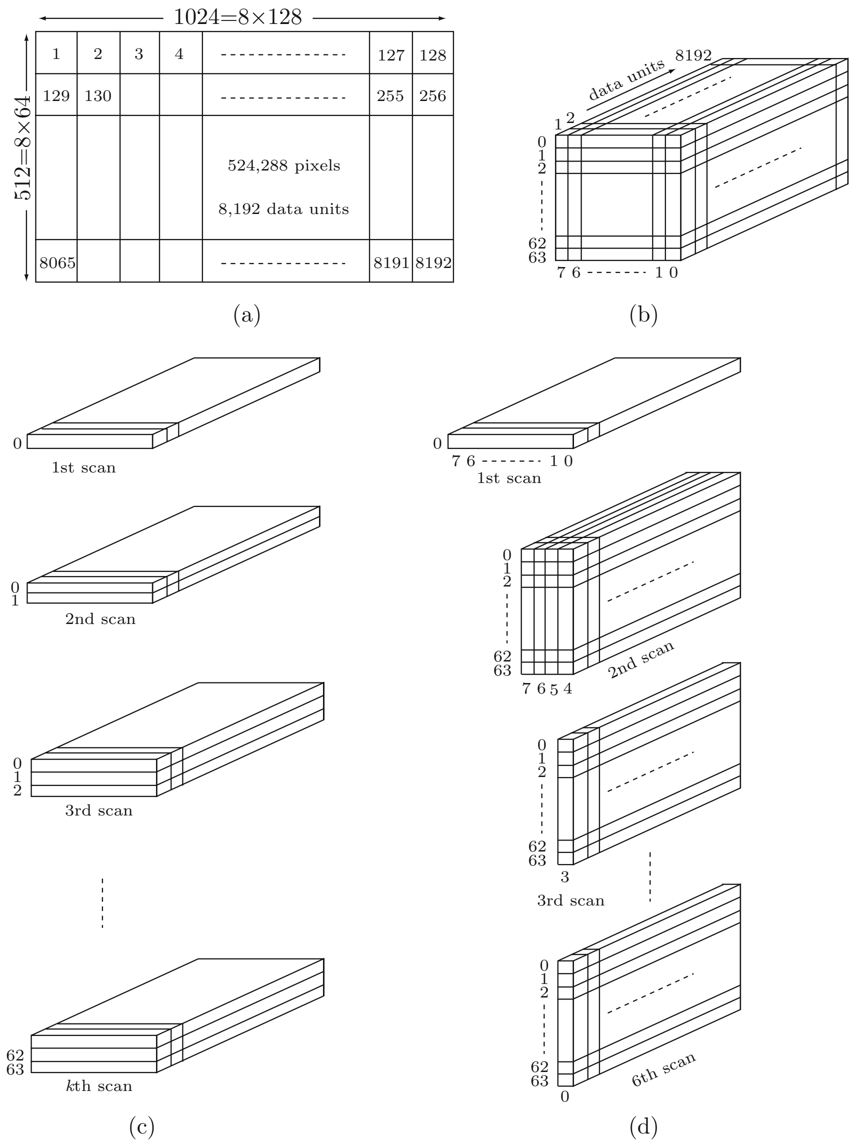


Figure 24.46: Scans in the JPEG Progressive Mode.

24.5.1 DCT

The general concept of a transform is discussed in Section 24.1. The discrete cosine transform is discussed in much detail in Section 24.3. Other examples of important transforms are the Fourier transform and the wavelet transform (Chapter 25). Both have applications in many areas and also have discrete versions (DFT and DWT).

The JPEG committee elected to use the DCT because of its excellent performance, because it does not assume anything about the structure of the data (the DFT, for example, assumes that the data to be transformed is periodic), and because there are ways to speed it up (Section 24.3.5).

The JPEG standard calls for applying the DCT not to the entire image but to data units (blocks) of 8×8 pixels. The reasons for this are: (1) Applying DCT to large blocks involves many arithmetic operations and is therefore slow. Applying DCT to small data units is faster. (2) Experience shows that, in a continuous-tone image, correlations between pixels are short range. A pixel in such an image has a value (color component or shade of gray) that's close to those of its near neighbors, but has nothing to do with the values of far neighbors. The JPEG DCT is therefore executed by Equation (24.13), duplicated here for $n=8$

$$\begin{aligned} G_{ij} &= \frac{1}{4}C_iC_j\sum_{x=0}^{7}\sum_{y=0}^{7}p_{xy}\cos\left(\frac{(2x+1)i\pi}{16}\right)\cos\left(\frac{(2y+1)j\pi}{16}\right),\\ &\text{where } C_f=\begin{cases}\frac{1}{\sqrt{2}}, & f=0,\\ 1, & f>0,\end{cases}\quad \text{and } 0\le i,j\le 7. \end{aligned} \tag{24.13}$$

The DCT is JPEG's key to lossy compression. The unimportant image information is reduced or removed by quantizing the 64 DCT coefficients, especially the ones located toward the lower-right. If the pixels of the image are correlated, quantization does not degrade the image quality much. For best results, each of the 64 coefficients is quantized by dividing it by a different quantization coefficient (QC). All 64 QCs are parameters that can be controlled, in principle, by the user (Section 24.5.2).

The JPEG decoder works by computing the inverse DCT (IDCT), Equation (24.14), duplicated here for $n=8$

$$\begin{aligned} p_{xy} &= \frac{1}{4}\sum_{i=0}^{7}\sum_{j=0}^{7}C_iC_jG_{ij}\cos\left(\frac{(2x+1)i\pi}{16}\right)\cos\left(\frac{(2y+1)j\pi}{16}\right),\\ &\text{where } C_f=\begin{cases}\frac{1}{\sqrt{2}}, & f=0;\\ 1, & f>0.\end{cases} \end{aligned} \tag{24.14}$$

It takes the 64 quantized DCT coefficients and calculates 64 pixels p_{xy}. If the QCs are the right ones, the new 64 pixels will be very similar to the original ones. Mathematically, the DCT is a one-to-one mapping of 64-point vectors from the image domain to the frequency domain. The IDCT is the reverse mapping. If the DCT and IDCT could be calculated with infinite precision and if the DCT coefficients were not quantized, the original 64 pixels would be exactly reconstructed.

24.5.2 Quantization

After each 8×8 data unit of DCT coefficients G_{ij} is computed, it is quantized. This is the step where information is lost (except for some unavoidable loss because of finite precision calculations in other steps). Each number in the DCT coefficients matrix is divided by the corresponding number from the particular "quantization table" used, and the result is rounded to the nearest integer. As has already been mentioned, three such tables are needed, for the three color components. The JPEG standard allows for up to four tables, and the user can select any of the four for quantizing each color component. The 64 numbers that constitute each quantization table are all JPEG parameters. In principle, they can all be specified and fine-tuned by the user for maximum compression. In practice, few users have the patience or expertise to experiment with so many parameters, so JPEG software normally uses the following two approaches:

1. Default quantization tables. Two such tables, for the luminance (grayscale) and the chrominance components, are the result of many experiments performed by the JPEG committee. They are included in the JPEG standard and are reproduced here as Table 24.47. It is easy to see how the QCs in the table generally grow as we move from the upper left corner to the bottom right corner. This is how JPEG reduces the DCT coefficients with high spatial frequencies.
2. A simple quantization table Q is computed, based on one parameter R specified by the user. A simple expression such as $Q_{ij} = 1 + (i + j) \times R$ guarantees that QCs start small at the upper-left corner and get bigger toward the lower-right corner. Table 24.48 shows an example of such a table with $R = 2$.

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

Luminance

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

Chrominance

Table 24.47: Recommended Quantization Tables.

If the quantization is done correctly, very few nonzero numbers will be left in the DCT coefficients matrix, and they will typically be concentrated in the upper-left region. These numbers are the output of JPEG, but they are further compressed before being written on the output stream. In the JPEG literature this compression is called "entropy coding," and Section 24.5.3 shows in detail how it is done. Three techniques are used by entropy coding to compress the 8×8 matrix of integers:

1. The 64 numbers are collected by scanning the matrix in zigzags (Figure 23.9). This produces a string of 64 numbers that starts with some nonzeros and typically ends with many consecutive zeros. Only the nonzero numbers are output (after further compressing

1	3	5	7	9	11	13	15
3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29

Table 24.48: The Quantization Table $1 + (i + j) \times 2$.

them) and are followed by a special end-of block (EOB) code. This way there is no need to output the trailing zeros (we can say that the EOB is the run-length encoding of all the trailing zeros). The interested reader should also consult chapter 11 of [Salomon 09] for other methods to compress binary strings with many consecutive zeros.

◇ **Exercise 24.10:** Propose a practical way to write a loop that traverses an 8×8 matrix in zigzag.

2. The nonzero numbers are compressed using Huffman coding (Section 24.5.3).
3. The first of those numbers (the DC coefficient, Page 1083) is treated differently from the others (the AC coefficients).

She had just succeeded in curving it down into a graceful zigzag, and was going to dive in among the leaves, which she found to be nothing but the tops of the trees under which she had been wandering, when a sharp hiss made her draw back in a hurry.

—Lewis Carroll, *Alice in Wonderland* (1865).

24.5.3 Coding

We first discuss point 3 above. Each 8×8 matrix of quantized DCT coefficients contains one DC coefficient (at position $(0, 0)$, the top left corner) and 63 AC coefficients. The DC coefficient is a measure of the average value of the 64 original pixels, constituting the data unit. Experience shows that in a continuous-tone image, adjacent data units of pixels are normally correlated in the sense that the average values of the pixels in adjacent data units are close. We already know that the DC coefficient of a data unit is a multiple of the average of the 64 pixels constituting the unit. This implies that the DC coefficients of adjacent data units don't differ much. JPEG outputs the first one (encoded), followed by *differences* (also encoded) of the DC coefficients of consecutive data units.

Example: If the first three 8×8 data units of an image have quantized DC coefficients of 1118, 1114, and 1119, then the JPEG output for the first data unit is 1118 (Huffman encoded, see below) followed by the 63 (encoded) AC coefficients of that data unit. The output for the second data unit will be $1114 - 1118 = -4$ (also Huffman encoded), followed by the 63 (encoded) AC coefficients of that data unit, and the output for the third data unit will be $1119 - 1114 = 5$ (also Huffman encoded), again followed by the 63 (encoded) AC coefficients of that data unit. This way of handling the DC coefficients is worth the extra trouble, because the differences are small.

Coding the DC differences is done with Table 24.49, so first here are a few words about this table. Each row has a row number (on the left), the unary code for the row (on the right), and several columns in between. Each row contains greater numbers (and also more numbers) than its predecessor but not the numbers contained in previous rows. Row i contains the range of integers $[-(2^i-1), +(2^i-1)]$ but is missing the middle range $[-(2^{i-1}-1), +(2^{i-1}-1)]$. Thus, the rows get very long, which means that a simple two-dimensional array is not a good data structure for this table. In fact, there is no need to store these integers in a data structure, since the program can figure out where in the table any given integer x is supposed to reside by analyzing the bits of x.

The first DC coefficient to be encoded in our example is 1118. It resides in row 11 column 930 of the table (column numbering starts at zero), so it is encoded as 111111111110|01110100010 (the unary code for row 11, followed by the 11-bit binary value of 930). The second DC difference is -4. It resides in row 3 column 3 of Table 24.49, so it is encoded as 1110|011 (the unary code for row 3, followed by the 3-bit binary value of 3).

◇ **Exercise 24.11:** How is the third DC difference, 5, encoded?

Point 2 above has to do with the precise way the 63 AC coefficients of a data unit are compressed. It uses a combination of RLE and either Huffman or arithmetic coding. The idea is that the sequence of AC coefficients normally contains just a few nonzero numbers, with runs of zeros between them, and with a long run of trailing zeros. For each nonzero number x, the encoder (1) finds the number Z of consecutive zeros preceding x; (2) finds x in Table 24.49 and prepares its row and column numbers (R and C); (3) the pair (R, Z) (that's (R,Z), not (R,C)) is used as row and column numbers for Table 24.52; and (4) the Huffman code found in that position in the table is concatenated to C (where C is written as an R-bit number) and the result is (finally) the code emitted by the JPEG encoder for the AC coefficient x and all the consecutive zeros preceding it.

0:	0										0
1:	-1	1									10
2:	-3	-2	2	3							110
3:	-7	-6	-5	-4	4	5	6	7			1110
4:	-15	-14	. . .	-9	-8	8	9	10	. . .	15	11110
5:	-31	-30	-29	. . .	-17	-16	16	17	. . .	31	111110
6:	-63	-62	-61	. . .	-33	-32	32	33	. . .	63	1111110
7:	-127	-126	-125	. . .	-65	-64	64	65	. . .	127	11111110
⋮				⋮							
14:	-16383	-16382	-16381	. . .	-8193	-8192	8192	8193	. . .	16383	111111111111110
15:	-32767	-32766	-32765	. . .	-16385	-16384	16384	16385	. . .	32767	1111111111111110
16:	32768										1111111111111111

Table 24.49: Coding the Differences of DC Coefficients.

The Huffman codes in Table 24.52 are not the ones recommended by the JPEG standard. The standard recommends the use of Tables 24.50 and 24.51 and says that up to four Huffman code tables can be used by a JPEG codec, except that the baseline mode can use only two such tables. The actual codes in Table 24.52 are thus arbitrary. The reader should notice the EOB code at position $(0,0)$ and the ZRL code at position $(0,15)$. The former indicates end-of-block, and the latter is the code emitted for 15 consecutive zeros when the number of consecutive zeros exceeds 15. These codes are the ones recommended for the luminance AC coefficients of Table 24.50. The EOB and ZRL codes recommended for the chrominance AC coefficients of Table 24.51 are 00 and 1111111010, respectively.

As an example consider the sequence

$$1118, 2, 0, -2, \underbrace{0, \ldots, 0}_{13}, -1, 0, \ldots .$$

The first AC coefficient 2 has no zeros preceding it, so $\mathrm{Z} = 0$. It is found in Table 24.49 in row 2, column 2, so $\mathrm{R} = 2$ and $\mathrm{C} = 2$. The Huffman code in position $(\mathrm{R}, \mathrm{Z}) = (2, 0)$ of Table 24.52 is 01, so the final code emitted for 2 is 01|10. The next nonzero coefficient, -2, has one zero preceding it, so $\mathrm{Z} = 1$. It is found in Table 24.49 in row 2, column 1, so $\mathrm{R} = 2$ and $\mathrm{C} = 1$. The Huffman code in position $(\mathrm{R}, \mathrm{Z}) = (2, 1)$ of Table 24.52 is 11011, so the final code emitted for 2 is 11011|01.

◇ **Exercise 24.12:** What code is emitted for the last nonzero AC coefficient, -1?

Finally, the sequence of trailing zeros is encoded as 1010 (EOB), so the output for the above sequence of AC coefficients is 01101101110111010101010. We saw earlier that the DC coefficient is encoded as 111111111110|1110100010, so the final output for the entire 64-pixel data unit is the 46-bit number

$$1111111111100111010001001101101110111010101010.$$

These 46 bits encode one color component of the 64 pixels of a data unit. Let's assume that the other two color components are also encoded into 46-bit numbers. If each pixel originally consists of 24 bits, then this corresponds to a compression factor of $64 \times 24/(46 \times 3) \approx 11.13$; very impressive!

(Notice that the DC coefficient of 1118 has contributed 23 of the 46 bits. Subsequent data units code differences of their DC coefficient, which may take fewer than 10 bits instead of 23. They may feature much higher compression factors as a result.)

The same tables (Tables 24.49 and 24.52) used by the encoder should, of course, be used by the decoder. The tables may be predefined and used by a JPEG codec as defaults, or they may be specifically calculated for a given image in a special pass preceding the actual compression. The JPEG standard does not specify any code tables, so any JPEG codec must use its own.

Readers who feel that this coding scheme is complex should take a look at the much more complex CAVLC coding method that is employed by H.264 [Salomon 09] to encode a similar sequence of 8×8 DCT transform coefficients.

Some JPEG variants use a particular version of arithmetic coding, called the QM coder, that is specified in the JPEG standard. This version of arithmetic coding is adaptive, so it does not need Tables 24.49 and 24.52. It adapts its behavior to the image statistics as it goes along. Using arithmetic coding may produce 5–10% better compression than Huffman for a typical continuous-tone image. However, it is more

	R				
Z	1 6	2 7	3 8	4 9	5 A
0	00 1111000	01 11111000	100 1111110110	1011 1111111110000010	11010 1111111110000011
1	1100 1111111110000100	11011 1111111110000101	11110001 1111111110000110	111110110 1111111110000111	11111110110 1111111110001000
2	11100 111111110001010	11111001 111111110001011	1111110111 111111110001100	111111110100 111111110001101	111111110001001 111111110001110
3	111010 1111111110010001	111110111 1111111110010010	111111110101 1111111110010011	1111111110001111 1111111110010100	1111111110010000 1111111110010101
4	111011 1111111110011001	1111111000 1111111110011010	1111111110010110 1111111110011011	1111111110010111 1111111110011100	1111111110011000 1111111110011101
5	1111010 1111111110100001	11111110111 1111111110100010	1111111110011110 1111111110100011	1111111110011111 1111111110100100	1111111110100000 1111111110100101
6	1111011 1111111110101001	111111110110 1111111110101010	1111111110100110 1111111110101011	1111111110100111 1111111110101100	1111111110101000 1111111110101101
7	11111010 1111111110110001	111111110111 1111111110110010	1111111110101110 1111111110110011	1111111110101111 1111111110110100	1111111110110000 1111111110110101
8	111111000 1111111110111001	111111111000000 1111111110111010	1111111110110110 1111111110111011	1111111110110111 1111111110111100	1111111110111000 1111111110111101
9	111111001 1111111111000010	1111111110111110 1111111111000011	1111111110111111 1111111111000100	1111111111000000 1111111111000101	1111111111000001 1111111111000110
A	111111010 1111111111001011	1111111111000111 1111111111001100	1111111111001000 1111111111001101	1111111111001001 1111111111001110	1111111111001010 1111111111001111
B	1111111001 1111111111010100	1111111111010000 1111111111010101	1111111111010001 1111111111010110	1111111111010010 1111111111010111	1111111111010011 1111111111011000
C	1111111010 1111111111011101	1111111111011001 1111111111011110	1111111111011010 1111111111011111	1111111111011011 1111111111100000	1111111111011100 1111111111100001
D	11111111000 1111111111100110	1111111111100010 1111111111100111	1111111111100011 1111111111101000	1111111111100100 1111111111101001	1111111111100101 1111111111101010
E	1111111111101011 1111111111110000	1111111111101100 1111111111110001	1111111111101101 1111111111110010	1111111111101110 1111111111110011	1111111111101111 1111111111110100
F	11111111001 1111111111111001	1111111111110101 1111111111111010	1111111111110110 1111111111111011	1111111111110111 1111111111111101	1111111111111000 1111111111111110

Table 24.50: Recommended Huffman Codes for Luminance AC Coefficients.

	R				
Z	1	2	3	4	5
	6	7	8	9	A
0	01	100	1010	11000	11001
	111000	1111000	111110100	1111110110	111111110100
1	1011	111001	11110110	111110101	11111110110
	111111110101	111111110001000	111111110001001	111111110001010	111111110001011
2	11010	11110111	1111110111	111111110110	111111111000010
	1111111110001100	1111111110001101	1111111110001110	1111111110001111	1111111110010000
3	11011	11111000	1111111000	111111110111	1111111110010001
	1111111110010010	1111111110010011	1111111110010100	1111111110010101	1111111110010110
4	111010	111110110	1111111110010111	1111111110011000	1111111110011001
	1111111110011010	1111111110011011	1111111110011100	1111111110011101	1111111110011110
5	111011	1111111001	1111111110011111	1111111110100000	1111111110100001
	1111111110100010	1111111110100011	1111111110100100	1111111110100101	1111111110100110
6	1111001	11111110111	1111111110100111	1111111110101000	1111111110101001
	1111111110101010	1111111110101011	1111111110101100	1111111110101101	1111111110101110
7	1111010	11111111000	1111111110101111	1111111110110000	1111111110110001
	1111111110110010	1111111110110011	1111111110110100	1111111110110101	1111111110110110
8	11111001	1111111110110111	1111111110111000	1111111110111001	1111111110111010
	1111111110111011	1111111110111100	1111111110111101	1111111110111110	1111111110111111
9	111110111	1111111111000000	1111111111000001	1111111111000010	1111111111000011
	1111111111000100	1111111111000101	1111111111000110	1111111111000111	1111111111001000
A	111111000	1111111111001001	1111111111001010	1111111111001011	1111111111001100
	1111111111001101	1111111111001110	1111111111001111	1111111111010000	1111111111010001
B	111111001	1111111111010010	1111111111010011	1111111111010100	1111111111010101
	1111111111010110	1111111111010111	1111111111011000	1111111111011001	1111111111011010
C	111111010	1111111111011011	1111111111011100	1111111111011101	1111111111011110
	1111111111011111	1111111111100000	1111111111100001	1111111111100010	1111111111100011
D	11111111001	1111111111100100	1111111111100101	1111111111100110	1111111111100111
	1111111111101000	1111111111101001	1111111111101010	1111111111101011	1111111111101100
E	11111111100000	1111111111101101	1111111111101110	1111111111101111	1111111111110000
	1111111111110001	1111111111110010	1111111111110011	1111111111110100	1111111111110101
F	111111111000011	111111111010110	1111111111110111	1111111111111000	1111111111111001
	1111111111111010	1111111111111011	1111111111111100	1111111111111101	1111111111111110

Table 24.51: Recommended Huffman Codes for Chrominance AC Coefficients.

R Z:	0	1	...	15
0:	1010			11111111001(ZRL)
1:	00	1100	...	1111111111110101
2:	01	11011	...	1111111111110110
3:	100	1111001	...	1111111111110111
4:	1011	111110110	...	1111111111111000
5:	11010	11111110110	...	1111111111111001
⋮	⋮			

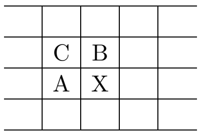
Table 24.52: Coding AC Coefficients.

complex to implement than Huffman coding, so in practice it is rare to find a JPEG codec that uses it.

24.5.4 Lossless Mode

The lossless mode of JPEG uses differencing to reduce the values of pixels before they are compressed. This particular form of differencing is called *predicting*. The values of some near neighbors of a pixel are subtracted from the pixel to get a small number, which is then compressed further using Huffman or arithmetic coding. Figure 24.53a shows a pixel X and three neighbor pixels A, B, and C. Figure 24.53b shows eight possible ways (predictions) to combine the values of the three neighbors. In the lossless mode, the user can select one of these predictions, and the encoder then uses it to combine the three neighbor pixels and subtract the combination from the value of X. The result is normally a small number, which is then entropy-coded in a way very similar to that described for the DC coefficient in Section 24.5.3.

Predictor 0 is used only in the hierarchical mode of JPEG. Predictors 1, 2, and 3 are called one-dimensional. Predictors 4, 5, 6, and 7 are two dimensional.



(a)

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	$(A + B)/2$

(b)

Figure 24.53: Pixel Prediction in the Lossless Mode.

It should be noted that the lossless mode of JPEG has never been very successful. It produces typical compression factors of 2, and is therefore inferior to other lossless image compression methods. Because of this, many JPEG implementations do not even implement this mode. Even the lossy (baseline) mode of JPEG does not perform well when asked to limit the amount of loss to a minimum. As a result, some JPEG implementations do not allow parameter settings that result in minimum loss. The strength of JPEG is in its ability to generate highly compressed images that when decompressed are indistinguishable from the original. Recognizing this, the ISO has decided to come up with another standard for lossless compression of continuous-tone images. This standard is now commonly known as JPEG-LS and is described in [Salomon 09].

24.5.5 The Compressed File

A JPEG encoder outputs a compressed file that includes parameters, markers, and the compressed data units. The parameters are either four bits (these always come in pairs), one byte, or two bytes long. The markers serve to identify the various parts of the file. Each is two bytes long, where the first byte is `X'FF'` and the second one is not 0 or `X'FF'`. A marker may be preceded by a number of bytes with `X'FF'`. Table 24.55 lists all the JPEG markers (the first four groups are start-of-frame markers). The compressed data units are combined into MCUs (minimal coded unit), where an MCU is either a single data unit (in the noninterleaved mode) or three data units from the three image components (in the interleaved mode).

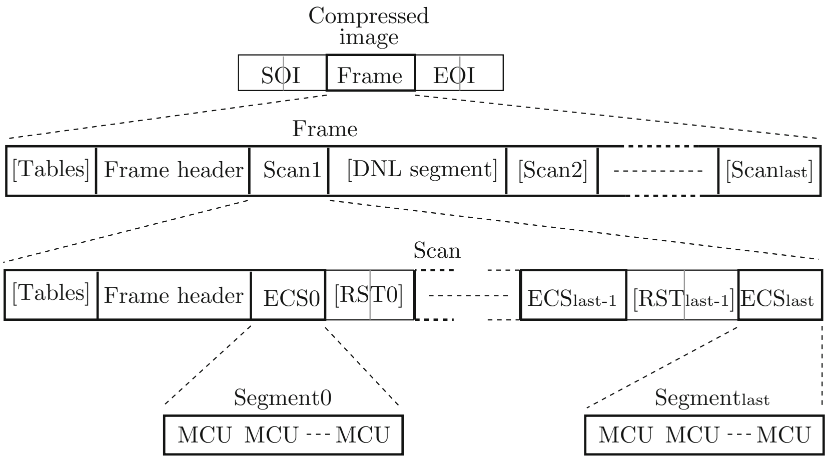


Figure 24.54: JPEG File Format.

Figure 24.54 shows the main parts of the JPEG compressed file (parts in square brackets are optional). The file starts with the SOI marker and ends with the EOI marker. In between these markers, the compressed image is organized in frames. In the hierarchical mode there are several frames, and in all other modes there is only one frame. In each frame the image information is contained in one or more scans, but the frame also contains a header and optional tables (which, in turn, may include markers). The first scan may be followed by an optional DNL segment (define number

Value	Name	Description
Nondifferential, Huffman coding		
FFC0	SOF$_0$	Baseline DCT
FFC1	SOF$_1$	Extended sequential DCT
FFC2	SOF$_2$	Progressive DCT
FFC3	SOF$_3$	Lossless (sequential)
Differential, Huffman coding		
FFC5	SOF$_5$	Differential sequential DCT
FFC6	SOF$_6$	Differential progressive DCT
FFC7	SOF$_7$	Differential lossless (sequential)
Nondifferential, arithmetic coding		
FFC8	JPG	Reserved for extensions
FFC9	SOF$_9$	Extended sequential DCT
FFCA	SOF$_{10}$	Progressive DCT
FFCB	SOF$_{11}$	Lossless (sequential)
Differential, arithmetic coding		
FFCD	SOF$_{13}$	Differential sequential DCT
FFCE	SOF$_{14}$	Differential progressive DCT
FFCF	SOF$_{15}$	Differential lossless (sequential)
Huffman table specification		
FFC4	DHT	Define Huffman table
Arithmetic coding conditioning specification		
FFCC	DAC	Define arith coding conditioning(s)
Restart interval termination		
FFD0–FFD7	RST$_m$	Restart with modulo 8 count m
Other markers		
FFD8	SOI	Start of image
FFD9	EOI	End of image
FFDA	SOS	Start of scan
FFDB	DQT	Define quantization table(s)
FFDC	DNL	Define number of lines
FFDD	DRI	Define restart interval
FFDE	DHP	Define hierarchical progression
FFDF	EXP	Expand reference component(s)
FFE0–FFEF	APP$_n$	Reserved for application segments
FFF0–FFFD	JPG$_n$	Reserved for JPEG extensions
FFFE	COM	Comment
Reserved markers		
FF01	TEM	For temporary private use
FF02–FFBF	RES	Reserved

Table 24.55: JPEG Markers.

of lines), which starts with the DNL marker and contains the number of lines in the image that's represented by the frame. A scan starts with optional tables, followed by the scan header, followed by several entropy-coded segments (ECS), which are separated by (optional) restart markers (RST). Each ECS contains one or more MCUs, where an MCU is, as explained earlier, either a single data unit or three such units.

I think he be transform'd into a beast;
For I can nowhere find him like a man.
—William Shakespeare, *As You Like It* (1601)