Lossy Compression Algorithms

8

In this chapter, we consider *lossy* compression methods. Since information loss implies some tradeoff between error and bitrate, we first consider measures of *distortion*—e.g., squared error. Different quantizers are introduced, each of which has a different distortion behavior. A discussion of transform coding leads into an introduction to the Discrete Cosine Transform used in JPEG compression (see Chap. 9) and the Karhunen Loève transform. Another transform scheme, wavelet-based coding, is then set out.

Sayood [1] deals extensively with the subject of lossy data compression in a well-organized and easy-to-understand manner. The mathematical foundation for the development of many lossy data compression algorithms is the study of *stochastic processes*. Stark and Woods [2] is an excellent textbook on this subject.

8.1 Introduction

As discussed in Chap. 7, the *compression ratio* for image data using lossless compression techniques (e.g., Huffman Coding, Arithmetic Coding, LZW) is low when the image histogram is relatively flat. For image compression in multimedia applications, where a higher compression ratio is required, lossy methods are usually adopted. In lossy compression, the compressed image is usually not the same as the original image but is meant to form a close approximation to the original image *perceptually*. To quantitatively describe how close the approximation is to the original data, some form of distortion measure is required.

8.2 Distortion Measures

A *distortion measure* is a mathematical quantity that specifies how close an approximation is to its original, using some distortion criteria. When looking at compressed data, it is natural to think of the distortion in terms of the numerical difference

Z.-N. Li et al., *Fundamentals of Multimedia*,
Texts in Computer Science, DOI: 10.1007/978-3-319-05290-8_8,

between the original data and the reconstructed data. However, when the data to be compressed is an image, such a measure may not yield the intended result.

For example, if the reconstructed image is the same as original image except that it is shifted to the right by one vertical scan line, an average human observer would have a hard time distinguishing it from the original and would therefore conclude that the distortion is small. However, when the calculation is carried out numerically, we find a large distortion, because of the large changes in individual pixels of the reconstructed image. The problem is that we need a measure of *perceptual distortion*, not a more naive numerical approach. However, the study of perceptual distortions is beyond the scope of this book.

Of the many numerical distortion measures that have been defined, we present the three most commonly used in image compression. If we are interested in the average pixel difference, the *mean square error* (MSE) σ^2 is often used. It is defined as

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - y_n)^2 \tag{8.1}$$

where x_n, y_n, and N are the input data sequence, reconstructed data sequence, and length of the data sequence, respectively.

If we are interested in the size of the error relative to the signal, we can measure the signal-to-noise ratio (SNR) by taking the ratio of the average square of the original data sequence and the mean square error (MSE), as discussed in Chap. 6. In decibel units (dB), it is defined as

$$SNR = 10\log_{10}\frac{\sigma_x^2}{\sigma_d^2} \tag{8.2}$$

where σ_x^2 is the average square value of the original data sequence and σ_d^2 is the MSE. Another commonly used measure for distortion is the *peak-signal-to-noise ratio* (PSNR), which measures the size of the error relative to the peak value of the signal x_{peak}. It is given by

$$PSNR = 10\log_{10}\frac{x_{\text{peak}}^2}{\sigma_d^2}. \tag{8.3}$$

8.3 The Rate-Distortion Theory

Lossy compression always involves a tradeoff between rate and distortion. Rate is the average number of bits required to represent each source symbol. Within this framework, the tradeoff between rate and distortion is represented in the form of a rate-distortion function $R(D)$.

Intuitively, for a given source and a given distortion measure, if D is a tolerable amount of distortion, $R(D)$ specifies the lowest rate at which the source data can be encoded while keeping the distortion bounded above by D. It is easy to see that when $D = 0$, we have a lossless compression of the source. The rate-distortion function

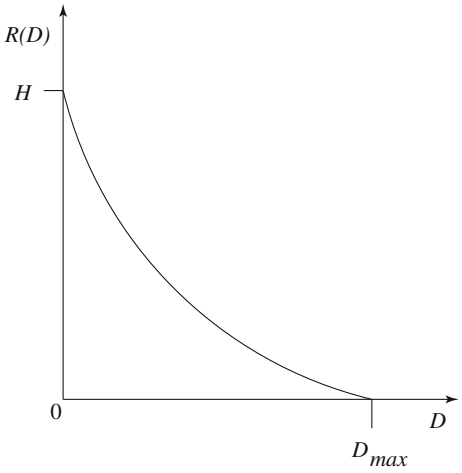

Fig. 8.1 Typical rate-distortion function

is meant to describe a fundamental limit for the performance of a coding algorithm and so can be used to evaluate the performance of different algorithms.

Figure 8.1 shows a typical rate-distortion function. Notice that the minimum possible rate at $D = 0$, no loss, is the entropy of the source data. The distortion corresponding to a rate $R(D) \equiv 0$ is the maximum amount of distortion incurred when "nothing" is coded.

Finding a closed-form analytic description of the rate-distortion function for a given source is difficult, if not impossible. Gyorgy [3] presents analytic expressions of the rate-distortion function for various sources. For sources for which an analytic solution cannot be readily obtained, the rate-distortion function can be calculated numerically, using algorithms developed by Arimoto [4] and Blahut [5].

8.4 Quantization

Quantization in some form is the heart of any lossy scheme. Without quantization, we would indeed be losing little information. Here, we embark on a more detailed discussion of quantization than in Sect. 6.3.2.

The source we are interested in compressing may contain a large number of distinct output values (or even infinite, if analog). To efficiently represent the source output, we have to reduce the number of distinct values to a much smaller set, via quantization.

Each algorithm (each *quantizer*) can be uniquely determined by its partition of the input range, on the encoder side, and the set of output values, on the decoder side. The input and output of each quantizer can be either scalar values or vector values,

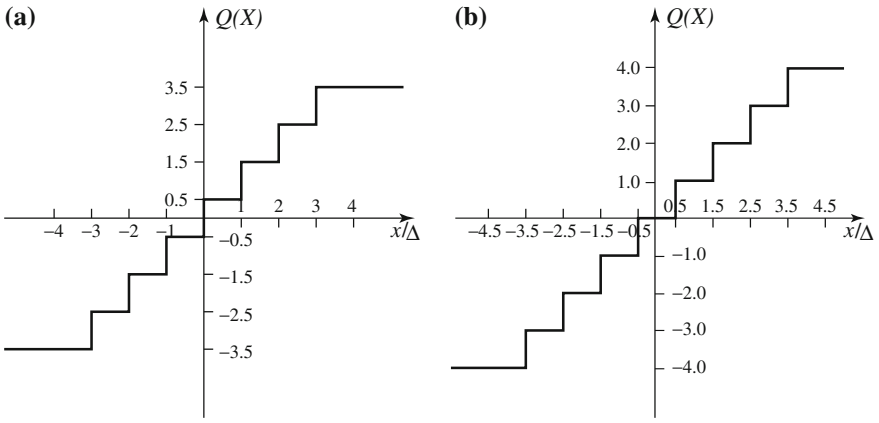

Fig. 8.2 Uniform scalar quantizers: **a** midrise; **b** midtread

thus leading to *scalar quantizers* and *vector quantizers*. In this section, we examine the design of both uniform and nonuniform scalar quantizers and briefly introduce the topic of *vector quantization* (VQ).

8.4.1 Uniform Scalar Quantization

A uniform scalar quantizer partitions the domain of input values into equally spaced intervals, except possibly at the two outer intervals. The endpoints of partition intervals are called the quantizer's *decision boundaries*. The output or reconstruction value corresponding to each interval is taken to be the midpoint of the interval. The length of each interval is referred to as the *step size*, denoted by the symbol Δ.

Uniform scalar quantizers are of two types: *midrise* and *midtread*. A midrise quantizer is used with an even number of output levels, and a midtread quantizer with an odd number. The midrise quantizer has a partition interval that brackets zero (see Fig. 8.2). The midtread quantizer has zero as one of its output values, hence, it is also known as *dead-zone* quantizer, because it turns a range of nonzero input values into the zero output.

The midtread quantizer is important when source data represents the zero value by fluctuating between small positive and negative numbers. Applying the midtread quantizer in this case would produce an accurate and steady representation of the value zero. For the special case $\Delta = 1$, we can simply compute the output values for these quantizers as

$$Q_{\text{midrise}}(x) = \lceil x \rceil - 0.5 \tag{8.4}$$

$$Q_{\text{midtread}}(x) = \lfloor x + 0.5 \rfloor. \tag{8.5}$$

The goal for the design of a successful uniform quantizer is to minimize the distortion for a given source input with a desired number of output values. This can be done by adjusting the step size Δ to match the input statistics.

Let's examine the performance of an M level quantizer. Let $B = \{b_0, b_1, \ldots, b_M\}$ be the set of decision boundaries and $Y = \{y_1, y_2, \ldots, y_M\}$ be the set of reconstruction or output values. Suppose the input is uniformly distributed in the interval $[-X_{\max}, X_{\max}]$. The rate of the quantizer is

$$R = \lceil \log_2 M \rceil. \tag{8.6}$$

That is, R is the number of bits required to code M things—in this case, the M output levels.

The step size Δ is given by

$$\Delta = \frac{2X_{\max}}{M} \tag{8.7}$$

since the entire range of input values is from $-X_{\max}$ to $X_{\max}$. For bounded input, the quantization error caused by the quantizer is referred to as *granular distortion*. If the quantizer replaces a whole range of values, from a maximum value to ∞, and similarly for negative values, that part of the distortion is called the *overload* distortion.

To get an overall figure for granular distortion, notice that decision boundaries b_i for a midrise quantizer are $[(i-1)\Delta, i\Delta]$, $i = 1 .. M/2$, covering positive data X (and another half for negative X values). Output values y_i are the midpoints $i\Delta - \Delta/2, i = 1 .. M/2$, again just considering positive data. The total distortion is twice the sum over the positive data, or

$$D_{\text{gran}} = 2\sum_{i=1}^{\frac{M}{2}} \int_{(i-1)\Delta}^{i\Delta} \left(x - \frac{2i-1}{2}\Delta\right)^2 \frac{1}{2X_{\max}}\, \mathrm{d}x \tag{8.8}$$

where we divide by the range of X to normalize to a value of at most 1.

Since the reconstruction values y_i are the midpoints of each interval, the quantization error must lie within the values $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$. Figure 8.3 is a graph of quantization error for a uniformly distributed source. The quantization error in this case is also uniformly distributed. Therefore, the average squared error is the same as the variance σ_d^2 of the quantization error calculated from just the interval $[0, \Delta]$ with error values in $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$. The error value at x is $e(x) = x - \Delta/2$, so the variance of errors is given by

$$\begin{aligned} \sigma_d^2 &= \frac{1}{\Delta}\int_0^{\Delta} (e(x) - \bar{e})^2\, \mathrm{d}x \\ &= \frac{1}{\Delta}\int_0^{\Delta} \left(x - \frac{\Delta}{2} - 0\right)^2 \mathrm{d}x \\ &= \frac{\Delta^2}{12}. \end{aligned} \tag{8.9}$$

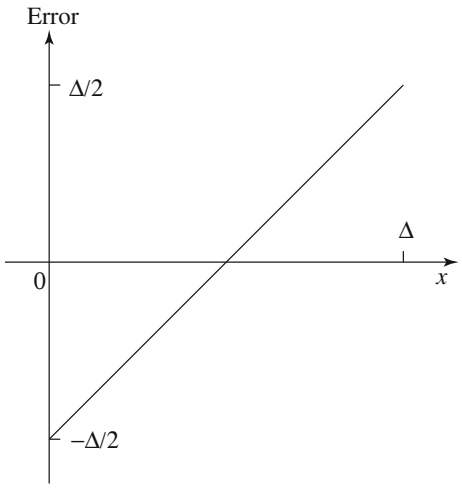

Fig. 8.3 Quantization error of a uniformly distributed source

Similarly, the *signal* variance is $\sigma_x^2 = (2X_{\max})^2/12$ for a random signal, so if the quantizer is n bits, $M = 2^n$, then from Eq. (8.2) we have

$$\begin{aligned} SQNR &= 10\log_{10}\left(\frac{\sigma_x^2}{\sigma_d^2}\right) \\ &= 10\log_{10}\left(\frac{(2X_{\max})^2}{12}\cdot\frac{12}{\Delta^2}\right) \\ &= 10\log_{10}\left(\frac{(2X_{\max})^2}{12}\cdot\frac{12}{\left(\frac{2X_{\max}}{M}\right)^2}\right) \\ &= 10\log_{10} M^2 = 20\,n\,\log_{10} 2 \qquad (8.10) \\ &= 6.02\,n\ (\text{dB}). \qquad (8.11) \end{aligned}$$

Hence, we have rederived the formula (6.3) derived more simply in Sect. 6.1. From Eq. (8.11), we have the important result that increasing one bit in the quantizer increases the signal-to-quantization noise ratio by 6.02 dB. In Sect. 6.1.5 we also showed that if we know the signal probability density function we can get a more accurate figure for the SQNR: there we assumed a sinusoidal signal and derived a more exact SQNR Eq. (6.4). As well, more sophisticated estimates of D result from more sophisticated models of the probability distribution of errors.

8.4.2 Nonuniform Scalar Quantization

If the input source is not uniformly distributed, a uniform quantizer may be inefficient. Increasing the number of decision levels within the region where the source is densely

distributed can effectively lower granular distortion. In addition, without having to increase the total number of decision levels, we can enlarge the region in which the source is sparsely distributed. Such *nonuniform quantizers* thus have nonuniformly defined decision boundaries.

There are two common approaches for nonuniform quantization: the Lloyd–Max quantizer and the companded quantizer, both introduced in Chap. 6.

Lloyd–Max Quantizer*

For a uniform quantizer, the total distortion is equal to the granular distortion, as in Eq. (8.8). If the source distribution is not uniform, we must explicitly consider its probability distribution (*probability density function*) $f_X(x)$. Now we need the correct decision boundaries b_i and reconstruction values y_i, by solving for both simultaneously. To do so, we plug variables b_i, y_i into a total distortion measure

$$D_{\text{gran}} = \sum_{j=1}^{M} \int_{b_{j-1}}^{b_j} (x - y_j)^2 \frac{1}{X_{\max}} f_X(x)\, \mathrm{d}x. \tag{8.12}$$

Then we can minimize the total distortion by setting the derivative of Eq. (8.12) to zero. Differentiating with respect to y_j yields the set of reconstruction values

$$y_j = \frac{\int_{b_{j-1}}^{b_j} x f_X(x)\, \mathrm{d}x}{\int_{b_{j-1}}^{b_j} f_X(x)\, \mathrm{d}x}. \tag{8.13}$$

This says that the optimal reconstruction value is the weighted centroid of the x interval. Differentiating with respect to b_j and setting the result to zero yields

$$b_j = \frac{y_{j+1} + y_j}{2}. \tag{8.14}$$

This gives a decision boundary b_j at the midpoint of two adjacent reconstruction values. Solving these two equations simultaneously is carried out by iteration. The result is termed the Lloyd–Max quantizer.

Starting with an initial guess of the optimal reconstruction levels, the algorithm above iteratively estimates the optimal boundaries, based on the current estimate of the reconstruction levels. It then updates the current estimate of the reconstruction levels, using the newly computed boundary information. The process is repeated until the reconstruction levels converge. For an example of the algorithm in operation, see Exercise 3.

Companded Quantizer

In companded quantization, the input is mapped by a *compressor function* G and then quantized using a uniform quantizer. After transmission, the quantized values

Algorithm 8.1 (Lloyd–Max Quantization)

```
BEGIN
  Choose initial level set y_0
  i = 0
  Repeat
    Compute b_i using Equation 8.14
    i = i + 1
    Compute y_i using Equation 8.13
  Until |y_i − y_{i−1}| < ϵ
END
```

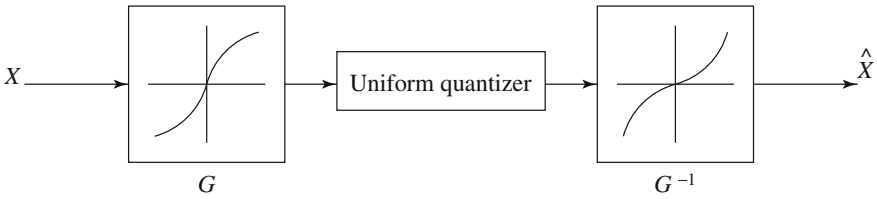

Fig. 8.4 Companded quantization

are mapped back using an *expander function* G^{-1}. The block diagram for the companding process is shown in Fig. 8.4, where $\hat{X}$ is the quantized version of X. If the input source is bounded by $x_{\max}$, then any nonuniform quantizer can be represented as a companded quantizer. The two commonly used companders are the μ-law and A-law companders (Sect. 6.1).

8.4.3 Vector Quantization

One of the fundamental ideas in Shannon's original work on information theory is that any compression system performs better if it operates on vectors or groups of samples rather than on individual symbols or samples. We can form vectors of input samples by concatenating a number of consecutive samples into a single vector. For example, an input vector might be a segment of a speech sample, a group of consecutive pixels in an image, or a chunk of data in any other format.

The idea behind *vector quantization* (VQ) is similar to that of scalar quantization but extended into multiple dimensions. Instead of representing values within an interval in one-dimensional space by a reconstruction value, as in scalar quantization, in VQ an n-component *code vector* represents vectors that lie within a region in n-dimensional space. A collection of these code vectors forms the *codebook* for the vector quantizer.

Since there is no implicit ordering of code vectors, as there is in the one-dimensional case, an index set is also needed to index into the codebook. Figure 8.5 shows the basic vector quantization procedure. In the diagram, the encoder finds

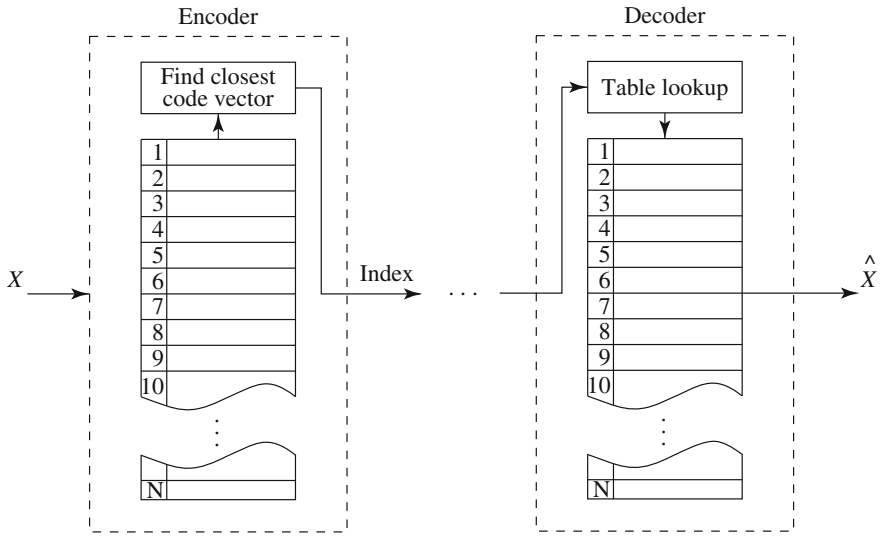

Fig. 8.5 Basic vector quantization procedure

the closest code vector to the input vector and outputs the associated index. On the decoder side, exactly the same codebook is used. When the coded index of the input vector is received, a simple table lookup is performed to determine the reconstruction vector.

Finding the appropriate codebook and searching for the closest code vector at the encoder end may require considerable computational resources. However, the decoder can execute quickly, since only a constant time operation is needed to obtain the reconstruction. Because of this property, VQ is attractive for systems with a lot of resources at the encoder end while the decoder has only limited resources, and the need is for quick execution time. Most multimedia applications fall into this category.

Gersho and Gray [6] cover quantization, especially vector quantization, comprehensively. In addition to the basic theory, this book provides a nearly exhaustive description of available VQ methods.

8.5 Transform Coding

From basic principles of information theory, we know that coding vectors is more efficient than coding scalars (see Sect. 7.4.2). To carry out such an intention, we need to group blocks of consecutive samples from the source input into vectors.

Let $\mathbf{X} = \{x_1, x_2, \ldots, x_k\}^T$ be a vector of samples. Whether our input data is an image, a piece of music, an audio or video clip, or even a piece of text, there is a good chance that a substantial amount of correlation is inherent among neighboring samples x_i. The rationale behind transform coding is that if $\mathbf{Y}$ is the result of a linear

transform **T** of the input vector **X** in such a way that the components of **Y** are much less correlated, then **Y** can be coded more efficiently than **X**.

For example, if most information in an RGB image is contained in a main axis, rotating so that this direction is the first component means that luminance can be compressed differently from color information. This will approximate the luminance channel in the eye.

In higher dimensions than three, if most information is accurately described by the first few components of a transformed vector, the remaining components can be coarsely quantized, or even set to zero, with little signal distortion. The more *decorrelated*—that is, the less effect one dimension has on another (the more orthogonal the axes), the more chance we have of dealing differently with the axes that store relatively minor amounts of information without affecting reasonably accurate reconstruction of the signal from its quantized or truncated transform coefficients.

Generally, the transform **T** itself does not compress any data. The compression comes from the processing and *quantization* of the components of **Y**. In this section, we will study the Discrete Cosine Transform (DCT) as a tool to decorrelate the input signal. We will also examine the Karhunen–Loève Transform (KLT), which *optimally* decorrelates the components of the input **X**.

8.5.1 Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT), a widely used transform coding technique, is able to perform decorrelation of the input signal in a data-independent manner [7,8]. Because of this, it has gained tremendous popularity. We will examine the definition of the DCT and discuss some of its properties, in particular the relationship between it and the more familiar Discrete Fourier Transform (DFT).

Definition of DCT

Let's start with the two-dimensional DCT. Given a function $f(i, j)$ over two integer variables i and j (a piece of an image), the 2D DCT transforms it into a new function $F(u, v)$, with integer u and v running over the same range as i and j. The general definition of the transform is

$$F(u,v) = \frac{2\,C(u)\,C(v)}{\sqrt{MN}} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \cos\frac{(2i+1)u\pi}{2M} \cos\frac{(2j+1)v\pi}{2N} f(i,j) \quad (8.15)$$

where $i, u = 0, 1, \ldots, M-1$, $j, v = 0, 1, \ldots, N-1$, and the constants $C(u)$ and $C(v)$ are determined by

$$C(\xi) = \begin{cases} \frac{\sqrt{2}}{2} & \text{if } \xi = 0, \\ 1 & \text{otherwise.} \end{cases} \quad (8.16)$$

In the JPEG image compression standard (see Chap. 9), an image block is defined to have dimension $M = N = 8$. Therefore, the definitions for the 2D DCT and its inverse (IDCT) in this case are as follows:

2D Discrete Cosine Transform (2D DCT)

$$F(u,v) = \frac{C(u)\,C(v)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} f(i,j), \quad (8.17)$$

where $i, j, u, v = 0, 1, \ldots, 7$, and the constants $C(u)$ and $C(v)$ are determined by Eq. (8.16).

2D Inverse Discrete Cosine Transform (2D IDCT)

The inverse function is almost the same, with the roles of $f(i,j)$ and $F(u,v)$ reversed, except that now $C(u)C(v)$ must stand inside the sums:

$$\tilde{f}(i,j) = \sum_{u=0}^{7} \sum_{v=0}^{7} \frac{C(u)\,C(v)}{4} \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} F(u,v) \quad (8.18)$$

where $i, j, u, v = 0, 1, \ldots, 7$, and the constants $C(u)$ and $C(v)$ are determined by Eq. (8.16).

The 2D transforms are applicable to 2D signals, such as digital images. As shown below, the 1D version of the DCT and IDCT is similar to the 2D version.

1D Discrete Cosine Transform (1D DCT)

$$F(u) = \frac{C(u)}{2} \sum_{i=0}^{7} \cos \frac{(2i+1)u\pi}{16} f(i), \quad (8.19)$$

where $i = 0, 1, \ldots, 7$, $u = 0, 1, \ldots, 7$, and the constant $C(u)$ is the same as in Eq. (8.16).

1D Inverse Discrete Cosine Transform (1D-IDCT)

$$\tilde{f}(i) = \sum_{u=0}^{7} \frac{C(u)}{2} \cos \frac{(2i+1)u\pi}{16} F(u), \quad (8.20)$$

where $i = 0, 1, \ldots, 7$, $u = 0, 1, \ldots, 7$, and the constant $C(u)$ is the same as in Eq. (8.16).

One-Dimensional DCT

Let's examine the DCT for a one-dimensional signal; almost all concepts are readily extensible to the 2D DCT.

An electrical signal with constant magnitude is known as a DC (direct current) signal. A common example is a battery that carries 1.5 or 9 volts DC. An electrical signal that changes its magnitude periodically at a certain frequency is known as an AC (alternating current) signal. A good example is the household electric power circuit, which carries electricity with sinusoidal waveform at 110 volts AC, 60 Hz (or 220 volts, 50 Hz in many other countries).

Most real signals are more complex. Speech signals or a row of gray-level intensities in a digital image are examples of such 1D signals. However, any signal can be expressed as a sum of multiple signals that are sine or cosine waveforms at various amplitudes and frequencies. This is known as Fourier analysis. The terms DC and AC, originating in electrical engineering, are carried over to describe these components of a signal (usually) composed of one DC and several AC components.

If a cosine function is used, the process of determining the amplitudes of the AC and DC components of the signal is called a *Cosine Transform*, and the integer indices make it a *Discrete Cosine Transform*. When $u = 0$, Eq. (8.19) yields the DC coefficient; when $u = 1$, or 2,..., up to 7, it yields the first or second, etc., up to the seventh AC coefficient.

Equation (8.20) shows the *Inverse Discrete Cosine Transform*. This uses a sum of the products of the DC or AC coefficients and the cosine functions to reconstruct (recompose) the function $f(i)$. Since computing the DCT and IDCT involves some loss, $f(i)$ is now denoted by $\tilde{f}(i)$.

In short, the role of the DCT is to decompose the original signal into its DC and AC components; the role of the IDCT is to reconstruct (recompose) the signal. The DCT and IDCT use the same set of cosine functions; they are known as *basis functions*. Figure 8.6 shows the family of eight 1D DCT basis functions: $u = 0 .. 7$.

The DCT enables a new means of signal processing and analysis in the *frequency domain*. In the original *Signal Processing* that deals with electrical and electronic signals (e.g., electricity, speech), $f(i)$ usually represents a signal that changes with time i (we will not be bothered here by the convention that time is usually denoted as t). The 1D DCT transforms $f(i)$, which is in the *time domain*, to $F(u)$, which is in the *frequency domain*. The coefficients $F(u)$ are known as the *frequency responses* and form the *frequency spectrum* of $f(i)$. In *Image Processing*, the image content $f(i, j)$ changes with the spatial indices i and j in the *space domain*. The 2D DCT now transforms $f(i, j)$ to $F(u, v)$, which is in the *spatial frequency domain*. For the convenience of discussion, we sometimes use 1D images and 1D DCT as examples.

Let's use some examples to illustrate frequency responses.

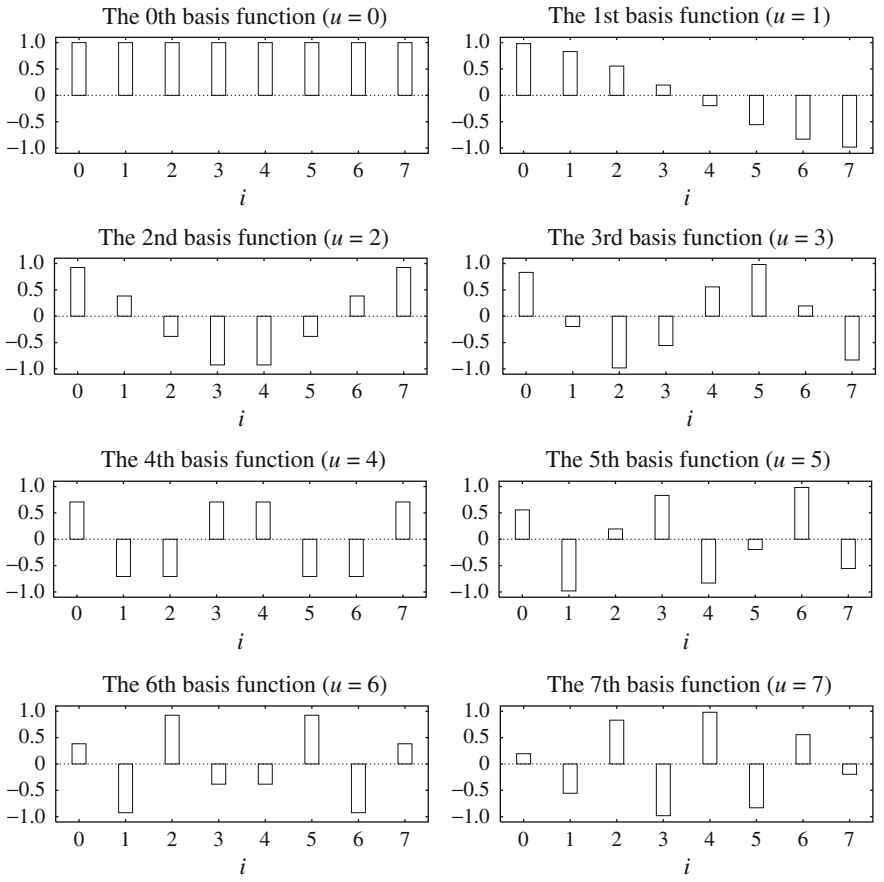

Fig. 8.6 The 1D DCT basis functions

Example 8.1

The left side of Fig. 8.7a shows a DC signal with a magnitude of 100, i.e., $f_1(i) = 100$. Since we are examining the *Discrete* Cosine Transform, the input signal is discrete, and its domain is [0, 7].

When $u = 0$, regardless of the i value, all the cosine terms in Eq. (8.19) become $\cos 0$, which equals 1. Taking into account that $C(0) = \sqrt{2}/2$, $F_1(0)$ is given by

$$\begin{aligned} F_1(0) &= \frac{\sqrt{2}}{2 \cdot 2} \cdot (1 \cdot 100 + 1 \cdot 100 + 1 \cdot 100 + 1 \cdot 100 \\ &\quad + 1 \cdot 100 + 1 \cdot 100 + 1 \cdot 100 + 1 \cdot 100) \\ &\approx 283 \end{aligned}$$

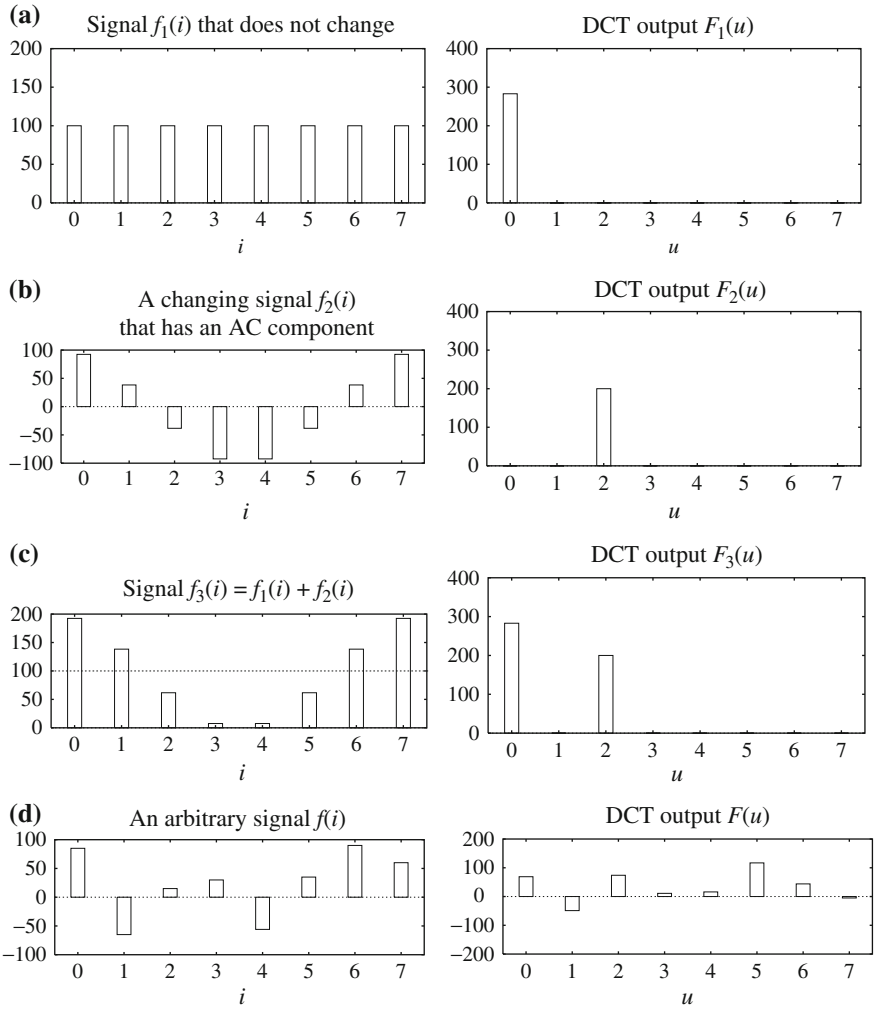

Fig. 8.7 Examples of 1D Discrete Cosine Transform: **a** a DC signal $f_1(i)$; **b** an AC signal $f_2(i)$; **c** $f_3(i) = f_1(i) + f_2(i)$; and **d** an arbitrary signal $f(i)$

when $u = 1$, $F_1(u)$ is as below. Because $\cos \frac{\pi}{16} = -\cos \frac{15\pi}{16}$, $\cos \frac{3\pi}{16} = -\cos \frac{13\pi}{16}$, etc. and $C(1) = 1$, we have

$$\begin{aligned} F_1(1) = \frac{1}{2} \cdot (&\cos \frac{\pi}{16} \cdot 100 + \cos \frac{3\pi}{16} \cdot 100 + \cos \frac{5\pi}{16} \cdot 100 + \cos \frac{7\pi}{16} \cdot 100 \\ &+ \cos \frac{9\pi}{16} \cdot 100 + \cos \frac{11\pi}{16} \cdot 100 + \cos \frac{13\pi}{16} \cdot 100 + \cos \frac{15\pi}{16} \cdot 100) \\ = 0. \end{aligned}$$

Similarly, it can be shown that $F_1(2) = F_1(3) = ... = F_1(7) = 0$. The 1D-DCT result $F_1(u)$ for this DC signal $f_1(i)$ is depicted on the right side of Fig. 8.7a—only a DC (i.e., first) component of F is nonzero.

Example 8.2

The left side of Fig. 8.7b shows a discrete cosine signal $f_2(i)$. Incidentally (or, rather, purposely), it has the same frequency and phase as the second cosine basis function, and its amplitude is 100.

When $u = 0$, again, all the cosine terms in Eq. (8.19) equal 1. Because $\cos\frac{\pi}{8} = -\cos\frac{7\pi}{8}$, $\cos\frac{3\pi}{8} = -\cos\frac{5\pi}{8}$, and so on, we have

$$\begin{aligned} F_2(0) = \frac{\sqrt{2}}{2\cdot 2}\cdot 1\cdot(&100\cos\frac{\pi}{8} + 100\cos\frac{3\pi}{8} + 100\cos\frac{5\pi}{8} + 100\cos\frac{7\pi}{8} \\ &+ 100\cos\frac{9\pi}{8} + 100\cos\frac{11\pi}{8} + 100\cos\frac{13\pi}{8} + 100\cos\frac{15\pi}{8}) \\ = 0. \end{aligned}$$

To calculate $F_2(u)$, we first note that when $u = 2$, because $\cos\frac{3\pi}{8} = \sin\frac{\pi}{8}$, we have

$$\cos^2\frac{\pi}{8} + \cos^2\frac{3\pi}{8} = \cos^2\frac{\pi}{8} + \sin^2\frac{\pi}{8} = 1.$$

Similarly,

$$\begin{aligned} \cos^2\frac{5\pi}{8} + \cos^2\frac{7\pi}{8} &= 1 \\ \cos^2\frac{9\pi}{8} + \cos^2\frac{11\pi}{8} &= 1 \\ \cos^2\frac{13\pi}{8} + \cos^2\frac{15\pi}{8} &= 1. \end{aligned}$$

Then we end up with

$$\begin{aligned} F_2(2) = \frac{1}{2}\cdot(&\cos\frac{\pi}{8}\cdot\cos\frac{\pi}{8} + \cos\frac{3\pi}{8}\cdot\cos\frac{3\pi}{8} + \cos\frac{5\pi}{8}\cdot\cos\frac{5\pi}{8} \\ &+ \cos\frac{7\pi}{8}\cdot\cos\frac{7\pi}{8} + \cos\frac{9\pi}{8}\cdot\cos\frac{9\pi}{8} + \cos\frac{11\pi}{8}\cdot\cos\frac{11\pi}{8} \\ &+ \cos\frac{13\pi}{8}\cdot\cos\frac{13\pi}{8} + \cos\frac{15\pi}{8}\cdot\cos\frac{15\pi}{8})\cdot 100 \\ = \frac{1}{2}\cdot(1+1+1+1)\cdot 100 &= 200. \end{aligned}$$

We will not show the other derivations in detail. It turns out that $F_2(1) = F_2(3) = F_2(4) = \cdots = F_2(7) = 0$.

Example 8.3

In the third row of Fig. 8.7 the input signal to the DCT is now the sum of the previous two signals—that is, $f_3(i) = f_1(i) + f_2(i)$. The output $F(u)$ values are

$$F_3(0) = 283,$$
$$F_3(2) = 200,$$
$$F_3(1) = F_3(3) = F_3(4) = \cdots = F_3(7) = 0.$$

Thus we discover that $F_3(u) = F_1(u) + F_2(u)$.

Example 8.4

The fourth row of the figure shows an arbitrary (or at least relatively complex) input signal $f(i)$ and its DCT output $F(u)$:

$f(i)(i = 0..7)$:	85	−65	15	30	−56	35	90	60
$F(u)(u = 0..7)$:	69	−49	74	11	16	117	44	−5.

Note that in this more general case, all the DCT coefficients $F(u)$ are nonzero and some are negative.

From the above examples, the characteristics of the DCT can be summarized as follows:

1. The DCT produces the spatial frequency spectrum $F(u)$ corresponding to the spatial signal $f(i)$.
   In particular, the 0th DCT coefficient $F(0)$ is the DC component of the signal $f(i)$. Up to a constant factor (i.e., $\frac{1}{2} \cdot \frac{\sqrt{2}}{2} \cdot 8 = 2 \cdot \sqrt{2}$ in the 1D DCT and $\frac{1}{4} \cdot \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2}}{2} \cdot 64 = 8$ in the 2D DCT), $F(0)$ equals the average magnitude of the signal. In Fig. 8.7a, the average magnitude of the DC signal is obviously 100, and $F(0) = 2\sqrt{2} \times 100$; in Fig. 8.7b, the average magnitude of the AC signal is 0, and so is $F(0)$; in Fig. 8.7c, the average magnitude of $f_3(i)$ is apparently 100, and again we have $F(0) = 2\sqrt{2} \times 100$.
   The other seven DCT coefficients reflect the various changing (i.e., AC) components of the signal $f(i)$ at different frequencies. If we denote $F(1)$ as AC1, $F(2)$ as AC2, ..., $F(7)$ as AC7, then AC1 is the first AC component, which completes half a cycle as a cosine function over [0, 7]; AC2 completes a full cycle; AC3 completes one and one-half cycles; ..., and AC7, three and a half cycles. All these are, of course, due to the cosine basis functions, which are arranged in exactly this manner. In other words, the second basis function corresponds to AC1, the third corresponds to AC2, and so on. In the example in Fig. 8.7b, since the signal $f_2(i)$ and the third basis function have exactly the same cosine waveform, with identical frequency and phase, they will reach the maximum (positive) and minimum (negative) values synchronously. As a result, their products are always positive, and the sum of their products ($F_2(2)$ or AC2) is large. It turns out that all other AC coefficients are zero, since $f_2(i)$ and all the other basis functions happen to be orthogonal. (We will discuss orthogonality later in this chapter.)
   It should be pointed out that the DCT coefficients can easily take on negative values. For DC, this occurs when the average of $f(i)$ is less than zero. (For an image, this never happens so the DC is nonnegative.) For AC, a special case occurs when $f(i)$ and some basis function have the same frequency but one

of them happens to be half a cycle behind—this yields a negative coefficient, possibly with a large magnitude.

In general, signals will look more like the one in Fig. 8.7d. Then $f(i)$ will produce many nonzero AC components, with the ones toward AC7 indicating higher frequency content. A signal will have large (positive or negative) response in its high-frequency components only when it alternates rapidly within the small range [0, 7].

As an example, if AC7 is a large positive number, this indicates that the signal $f(i)$ has a component that alternates synchronously with the eighth basis function—three and half cycles. According to the Nyqist theorem, this is the highest frequency in the signal that can be sampled with eight discrete values without significant loss and aliasing.

2. The DCT is a *linear transform*.
   In general, a transform $\mathcal{T}$ (or function) is *linear*, iff

$$\mathcal{T}(\alpha p + \beta q) = \alpha \mathcal{T}(p) + \beta \mathcal{T}(q), \tag{8.21}$$

where α and β are constants, and p and q are any functions, variables or constants.

From the definition in Eq. (8.19), this property can readily be proven for the DCT, because it uses only simple arithmetic operations.

One-Dimensional Inverse DCT

Let's finish the example in Fig. 8.7d by showing its inverse DCT (IDCT). Recall that $F(u)$ contains the following:

$$F(u)(u = 0..7): \quad 69 \quad -49 \quad 74 \quad 11 \quad 16 \quad 117 \quad 44 \quad -5.$$

The 1D IDCT, as indicated in Eq. (8.20), can readily be implemented as a loop with eight iterations, as illustrated in Fig. 8.8.

$$\text{Iteration 0:} \quad \tilde{f}(i) = \frac{C(0)}{2} \cdot \cos 0 \cdot F(0) = \frac{\sqrt{2}}{2 \cdot 2} \cdot 1 \cdot 69 \approx 24.3.$$

$$\begin{aligned}\text{Iteration 1:} \quad \tilde{f}(i) &= \frac{C(0)}{2} \cdot \cos 0 \cdot F(0) + \frac{C(1)}{2} \cdot \cos \frac{(2i+1)\pi}{16} \cdot F(1) \\ &\approx 24.3 + \frac{1}{2} \cdot (-49) \cdot \cos \frac{(2i+1)\pi}{16} \approx 24.3 - 24.5 \cdot \cos \frac{(2i+1)\pi}{16}.\end{aligned}$$

$$\begin{aligned}\text{Iteration 2:} \quad \tilde{f}(i) &= \frac{C(0)}{2} \cdot \cos 0 \cdot F(0) + \frac{C(1)}{2} \cdot \cos \frac{(2i+1)\pi}{16} \cdot F(1) + \frac{C(2)}{2} \cdot \cos \frac{(2i+1)\pi}{8} \cdot F(2) \\ &\approx 24.3 - 24.5 \cdot \cos \frac{(2i+1)\pi}{16} + 37 \cdot \cos \frac{(2i+1)\pi}{8}.\end{aligned}$$

After iteration 0, $\tilde{f}(i)$ has a constant value of approximately 24.3, which is the recovery of the DC component in $f(i)$; after iteration 1, $\tilde{f}(i) \approx 24.3 - 24.5 \cdot \cos \frac{(2i+1)\pi}{16}$, which is the sum of the DC and first AC component; after iteration 2, $\tilde{f}(i)$ reflects the sum of DC and AC1 and AC2; and so on. As shown, the process of the sum-of-product in IDCT eventually reconstructs (recomposes) the function $f(i)$, which is approximately

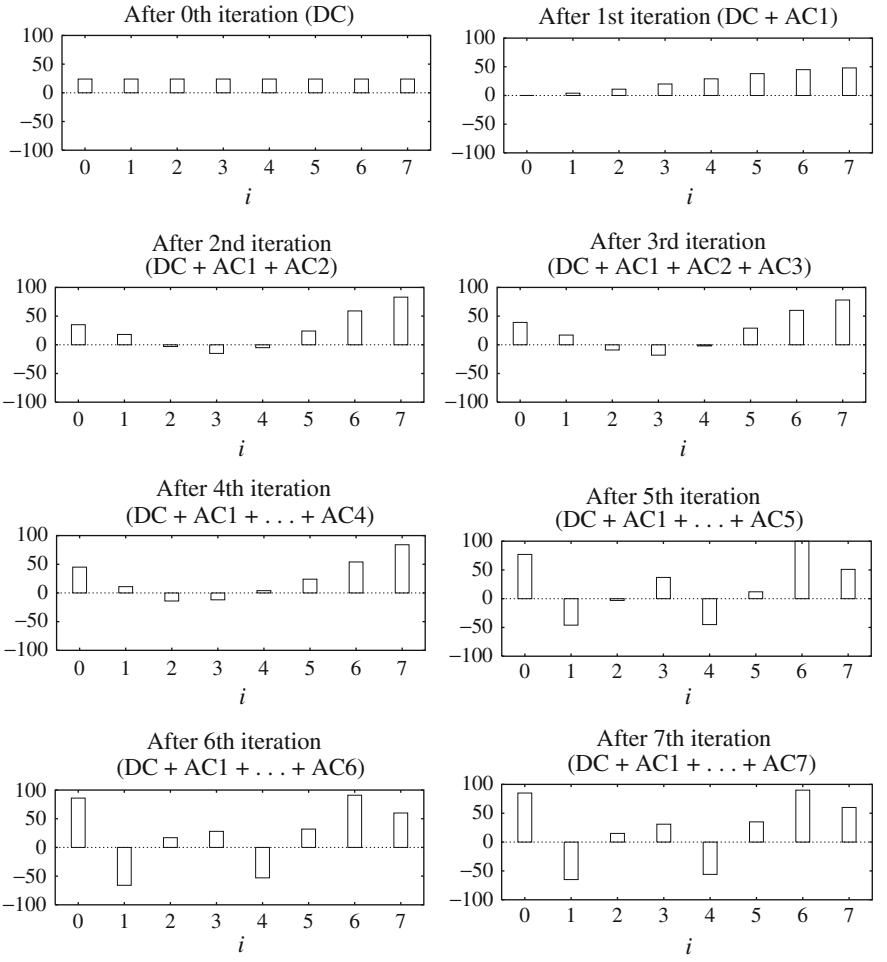

Fig. 8.8 An example of 1D IDCT

$$\tilde{f}(i)(i = 0\,..\,7): \quad 85 \quad -65 \quad 15 \quad 30 \quad -56 \quad 35 \quad 90 \quad 60.$$

As it happens, even though we went from integer to integer via intermediate *floats*, we recovered the signal exactly. This is not always true, but the answer is always close.

The Cosine Basis Functions

For a better decomposition, the basis functions should be *orthogonal*, so as to have the least redundancy among them.

Functions $B_p(i)$ and $B_q(i)$ are orthogonal if

$$\sum_i [B_p(i) \cdot B_q(i)] = 0 \quad \text{if } p \neq q. \tag{8.22}$$

Functions $B_p(i)$ and $B_q(i)$ are *orthonormal* if they are orthogonal and

$$\sum_i [B_p(i) \cdot B_q(i)] = 1 \quad \text{if } p = q. \tag{8.23}$$

The orthonormal property is desirable. With this property, the signal is not amplified during the transform. When the same basis function is used in both the transformation and its inverse (sometimes called *forward transform* and *backward transform*), we will get (approximately) the same signal back.

It can be shown that

$$\sum_{i=0}^{7} \left[\cos \frac{(2i+1) \cdot p\pi}{16} \cdot \cos \frac{(2i+1) \cdot q\pi}{16} \right] = 0 \quad \text{if } p \neq q$$

$$\sum_{i=0}^{7} \left[\frac{C(p)}{2} \cos \frac{(2i+1) \cdot p\pi}{16} \cdot \frac{C(q)}{2} \cos \frac{(2i+1) \cdot q\pi}{16} \right] = 1 \quad \text{if } p = q.$$

The cosine basis functions in the DCT are indeed orthogonal. With the help of constants $C(p)$ and $C(q)$ they are also orthonormal. (Now we understand why constants $C(u)$ and $C(v)$ in the definitions of DCT and IDCT seemed to have taken some arbitrary values.)

Recall that because of the orthogonality, for $f_2(i)$ in Fig. 8.7b, only $F_2(2)$ (for $u = 2$) has a nonzero output whereas all other DCT coefficients are zero. This is desirable for some signal processing and analysis in the frequency domain, since we are now able to precisely identify the frequency components in the original signal.

The cosine basis functions are analogous to the basis vectors $\vec{x}$, $\vec{y}$, $\vec{z}$ for the 3D Cartesian space, or the so-called *3D vector space*. The vectors are orthonormal, because

$$\begin{aligned}
\vec{x} \cdot \vec{y} &= (1, 0, 0) \cdot (0, 1, 0) = 0 \\
\vec{x} \cdot \vec{z} &= (1, 0, 0) \cdot (0, 0, 1) = 0 \\
\vec{y} \cdot \vec{z} &= (0, 1, 0) \cdot (0, 0, 1) = 0 \\
\vec{x} \cdot \vec{x} &= (1, 0, 0) \cdot (1, 0, 0) = 1 \\
\vec{y} \cdot \vec{y} &= (1, 0, 0) \cdot (1, 0, 0) = 1 \\
\vec{z} \cdot \vec{z} &= (1, 0, 0) \cdot (1, 0, 0) = 1.
\end{aligned}$$

Any point $P = (x_p, y_p, z_p)$ can be represented by a vector $\vec{OP} = (x_p, y_p, z_p)$, where O is the origin, which can in turn be decomposed into $x_p \cdot \vec{x} + y_p \cdot \vec{y} + z_p \cdot \vec{z}$.

If we view the sum-of-products operation in Eq. (8.19) as the dot product of one of the discrete cosine basis functions (for a specified u) and the signal $f(i)$, then the analogy between the DCT and the Cartesian projection is remarkable. Namely, to get the x-coordinate of point P, we simply project P onto the x axis. Mathematically, this

is equivalent to a dot product $\vec{x} \cdot \vec{OP} = x_p$. Obviously, the same goes for obtaining y_p and z_p.

Now, compare this to the example in Fig. 8.7b, for a point $P = (0, 5, 0)$ in the Cartesian space. Only its projection onto the y axis is $y_p = 5$ and its projections onto the x and z axes are both 0.

Finally, for reconstruction of P, use the dot product of (x_p, y_p, z_p) and $(\vec{x}, \vec{y}, \vec{z})$ to obtain $x_p \cdot \vec{x} + y_p \cdot \vec{y} + z_p \cdot \vec{z}$.

2D Basis Functions

For two-dimensional DCT functions, we use the basis depicted as 8×8 *images* in Fig. 8.9, where u and v indicate their spatial frequencies, white indicates positive values and black indicates negative. For a particular pair of u and v, the respective basis function is:

$$\cos \frac{(2i+1) \cdot u\pi}{16} \cdot \cos \frac{(2j+1) \cdot v\pi}{16}, \tag{8.24}$$

where i and j are their row and column indices.

For example, for the enlarged block shown in Fig. 8.9, where $u = 1$ and $v = 2$, it is:

$$\cos \frac{(2i+1) \cdot 1\pi}{16} \cdot \cos \frac{(2j+1) \cdot 2\pi}{16}.$$

To obtain DCT coefficients, we essentially just form the inner product of each of these 64 basis functions with an 8×8 block from an original image. Again, we are talking about an original signal indexed by space, not time. The 64 products we calculate make up an 8×8 *spatial frequency* response $F(u, v)$. We do this for each 8×8 image block.

2D Separable Basis

Of course, for speed, most software implementations use fixed point arithmetic to calculate the DCT transform. Just as there is a mathematically derived Fast Fourier Transform, there is also a Fast DCT. Some fast implementations approximate coefficients so that all multiplies are shifts and adds. Moreover, a much simpler mechanism is used to produce 2D DCT coefficients—*factorization* into two 1D DCT transforms.

The 2D DCT can be *separated* into a sequence of two 1D DCT steps. First, we calculate an intermediate function $G(u, j)$ by performing a 1D DCT in each column—in this way, we have taken care of the 1D transform vertically, i.e., replacing the row index by its frequency counterpart u. When the block size is 8×8:

$$G(u, j) = \frac{1}{2} C(u) \sum_{i=0}^{7} \cos \frac{(2i+1)u\pi}{16} f(i, j). \tag{8.25}$$

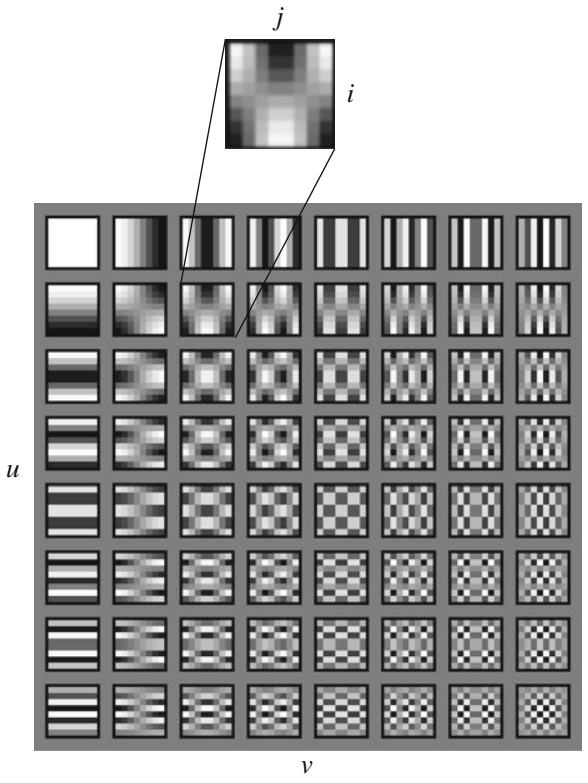

Fig. 8.9 Graphical illustration of 8×8 2D DCT basis

Then we calculate another 1D DCT horizontally in each row, this time replacing the column index j by its frequency counterpart v:

$$F(u, v) = \frac{1}{2}C(v)\sum_{j=0}^{7}\cos\frac{(2j+1)v\pi}{16}G(u, j). \tag{8.26}$$

This is possible because the 2D DCT basis functions are *separable* (multiply separate functions of i and j). It is straightforward to see that this simple change saves many arithmetic steps. The number of iterations required is reduced from 8×8 to $8 + 8$.

2D DCT-Matrix Implementation

The above factorization of a 2D DCT into two 1D DCTs can be implemented by two consecutive matrix multiplications, i.e.,

$$F(u, v) = \mathbf{T} \cdot f(i, j) \cdot \mathbf{T}^T. \tag{8.27}$$

We will name $\mathbf{T}$ the *DCT-matrix*.

$$\mathbf{T}[i,\, j] = \begin{cases} \frac{1}{\sqrt{N}}, & \text{if } i = 0 \\ \sqrt{\frac{2}{N}} \cdot \cos\frac{(2j+1)\cdot i\pi}{2N}, & \text{if } i > 0 \end{cases} \tag{8.28}$$

where $i = 0, ..., N-1$ and $j = 0, ..., N-1$ are the row and column indices, and the block size is $N \times N$.

When $N = 8$, we have:

$$\mathbf{T_8}[i,\, j] = \begin{cases} \frac{1}{2\sqrt{2}}, & \text{if } i = 0 \\ \frac{1}{2} \cdot \cos\frac{(2j+1)\cdot i\pi}{16}, & \text{if } i > 0. \end{cases} \tag{8.29}$$

Hence,

$$\mathbf{T_8} = \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \cdots & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\cdot\cos\frac{\pi}{16} & \frac{1}{2}\cdot\cos\frac{3\pi}{16} & \frac{1}{2}\cdot\cos\frac{5\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{15\pi}{16} \\ \frac{1}{2}\cdot\cos\frac{\pi}{8} & \frac{1}{2}\cdot\cos\frac{3\pi}{8} & \frac{1}{2}\cdot\cos\frac{5\pi}{8} & \cdots & \frac{1}{2}\cdot\cos\frac{15\pi}{8} \\ \frac{1}{2}\cdot\cos\frac{3\pi}{16} & \frac{1}{2}\cdot\cos\frac{9\pi}{16} & \frac{1}{2}\cdot\cos\frac{15\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{45\pi}{16} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{2}\cdot\cos\frac{7\pi}{16} & \frac{1}{2}\cdot\cos\frac{21\pi}{16} & \frac{1}{2}\cdot\cos\frac{35\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{105\pi}{16} \end{bmatrix}. \tag{8.30}$$

A closer look at the DCT-matrix will reveal that each row of the matrix is basically a 1D DCT basis function, ranging from DC to AC1, AC2, ..., AC7. Compared to the functions in Fig. 8.6, the only difference is that we have added some constants and taken care of the orthonormal aspect of the DCT basis functions. Indeed, the constants and basis functions in Eqs. (8.19) and (8.29) are exactly the same. (We will leave it as an exercise (see Exercise 7) to verify that the rows and columns of $\mathbf{T_8}$ are orthonormal vectors, i.e., $\mathbf{T_8}$ is an Orthogonal Matrix.)

In summary, the implementation of the 2D DCT is now a simple matter of applying two matrix multiplications as in Eq. (8.27). The first multiplication applies 1D DCT vertically (for each column), and the second applies 1D DCT horizontally (for each row). What has been achieved is exactly the two steps as indicated in Eqs. (8.25) and (8.26).

2D IDCT Matrix Implementation

In this section, we will show how to reconstruct $f(i, j)$ from $F(u, v)$ losslessly by matrix multiplications. In the next several chapters, when we discuss lossy compression of images and videos, quantization steps will usually be applied to the DCT coefficients $F(u, v)$ before the IDCT.

It turns out that the 2D IDCT matrix implementation is simply:

$$f(i, j) = \mathbf{T}^T \cdot F(u, v) \cdot \mathbf{T}. \tag{8.31}$$

Its derivation is as follows:

First, because $\mathbf{T} \cdot \mathbf{T}^{-1} = \mathbf{T}^{-1} \cdot \mathbf{T} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, we can simply rewrite $f(i, j)$ as:

$$f(i, j) = \mathbf{T}^{-1} \cdot \mathbf{T} \cdot f(i, j) \cdot \mathbf{T}^T \cdot (\mathbf{T}^T)^{-1}.$$

According to Eq. (8.27),

$$F(u, v) = \mathbf{T} \cdot f(i, j) \cdot \mathbf{T}^T.$$

Hence,

$$f(i, j) = \mathbf{T}^{-1} \cdot F(u, v) \cdot (\mathbf{T}^T)^{-1}.$$

As stated above, the DCT-matrix $\mathbf{T}$ is orthogonal, therefore,

$$\mathbf{T}^T = \mathbf{T}^{-1}.$$

It follows,

$$f(i, j) = \mathbf{T}^T \cdot F(u, v) \cdot \mathbf{T}.$$

Comparison of DCT and DFT

The discrete cosine transform is a close counterpart to the *Discrete Fourier Transform* (*DFT*) [9], and in the world of signal processing, the latter is likely the more common. We have started off with the DCT instead because it is simpler and is also much used in multimedia. Nevertheless, we should not entirely ignore the DFT.

For a continuous signal, we define the continuous Fourier transform $\mathcal{F}$ as follows:

$$\mathcal{F}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} \, \mathrm{d}t. \tag{8.32}$$

Using Euler's formula, we have

$$e^{ix} = \cos(x) + i \sin(x). \tag{8.33}$$

Thus, the continuous Fourier transform is composed of an infinite sum of sine and cosine terms. Because digital computers require us to discretize the input signal, we define a DFT that operates on eight samples of the input signal $\{f_0, f_1, \ldots, f_7\}$ as

$$F_\omega = \sum_{x=0}^{7} f_x \cdot e^{-\frac{2\pi i \omega x}{8}}. \tag{8.34}$$

Writing the sine and cosine terms explicitly, we have

$$F_\omega = \sum_{x=0}^{7} f_x \cos\left(\frac{2\pi\omega x}{8}\right) - i \sum_{x=0}^{7} f_x \sin\left(\frac{2\pi\omega x}{8}\right). \tag{8.35}$$

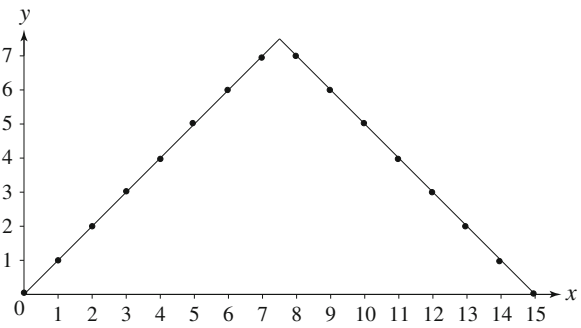

Fig. 8.10 Symmetric extension of the ramp function

Table 8.1 DCT and DFT coefficients of the ramp function

Ramp	DCT	DFT
0	9.90	28.00
1	−6.44	−4.00
2	0.00	9.66
3	−0.67	−4.00
4	0.00	4.00
5	−0.20	−4.00
6	0.00	1.66
7	−0.51	−4.00

Even without giving an explicit definition of the DCT, we can guess that the DCT is likely a transform that involves only the real part of the DFT. The intuition behind the formulation of the DCT that allows it to use only the cosine basis functions of the DFT is that we can cancel out the imaginary part of the DFT by making a symmetric copy of the original input signal.

This works because sine is an odd function; thus, the contributions from the sine terms cancel each other out when the signal is symmetrically extended. Therefore, the DCT of eight input samples corresponds to the DFT of 16 samples made up of the original eight input samples and a symmetric copy of these, as in Fig. 8.10.

With the symmetric extension, the DCT is now working on a triangular wave, whereas the DFT tries to code the repeated ramp. Because the DFT is trying to model the artificial discontinuity created between each copy of the samples of the ramp function, a lot of high-frequency components are needed. (Refer to [9] for a thorough discussion and comparison of DCT and DFT.)

Table 8.1 shows the calculated DCT and DFT coefficients. We can see that more energy is concentrated in the first few coefficients in the DCT than in the DFT. If we try to approximate the original ramp function using only three terms of both the DCT and DFT, we notice that the DCT approximation is much closer. Figure 8.11 shows the comparison.

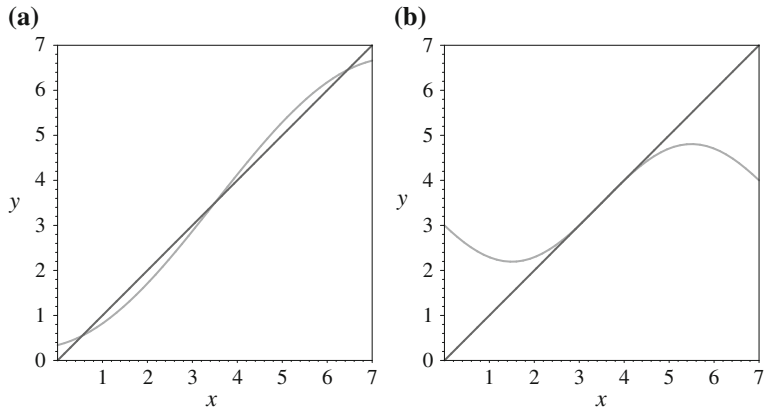

Fig. 8.11 Approximation of the ramp function: **a** three-term DCT approximation; **b** three-term DFT approximation

8.5.2 Karhunen–Loève Transform*

The Karhunen–Loève Transform (KLT) is a reversible linear transform that exploits the statistical properties of the vector representation. Its primary property is that it optimally decorrelates the input. To do so, it fits an n-dimensional ellipsoid around the (mean-subtracted) data. The main ellipsoid axis is the major direction of change in the data.

Think of a cigar that has unfortunately been stepped on. Cigar data consists of a cloud of points in 3-space giving the coordinates of positions of measured points in the cigar. The long axis of the cigar will be identified by a statistical program as the first KLT axis. The second most important axis is the horizontal axis across the squashed cigar, perpendicular to the first axis. The third axis is orthogonal to both and is in the vertical, thin direction. A KLT component program carries out just this analysis.

To understand the optimality of the KLT, consider the autocorrelation matrix $\mathbf{R_X}$ of the set of k input vectors $\mathbf{X}$, defined in terms of the expectation value $E(\cdot)$ as

$$\mathbf{R_X} = E[\mathbf{X}\mathbf{X}^T] \tag{8.36}$$

$$= \begin{bmatrix} R_X(1,1) & R_X(1,2) & \cdots & R_X(1,k) \\ R_X(2,1) & R_X(2,2) & \cdots & R_X(2,k) \\ \vdots & \vdots & \ddots & \vdots \\ R_X(k,1) & R_X(k,2) & \cdots & R_X(k,k) \end{bmatrix} \tag{8.37}$$

where $R_X(t,s) = E[X_t X_s]$ is the autocorrelation function. Our goal is to find a transform $\mathbf{T}$ such that the components of the output $\mathbf{Y}$ are uncorrelated—that is, $E[Y_t Y_s] = 0$, if $t \neq s$. Thus, the autocorrelation matrix of $\mathbf{Y}$ takes on the form of a positive diagonal matrix.

Since any autocorrelation matrix is symmetric and nonnegative definite, there are k orthogonal eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$ and k corresponding real and nonnegative eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 0$. We define the Karhunen-Loève transform as

$$\mathbf{T} = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k]^T. \tag{8.38}$$

Then, the autocorrelation matrix of $\mathbf{Y}$ becomes

$$\begin{aligned} \mathbf{R_Y} &= E[\mathbf{Y}\mathbf{Y}^T] && (8.39)\\ &= E[\mathbf{T}\mathbf{X}\mathbf{X}^T\mathbf{T}] && (8.40)\\ &= \mathbf{T}\mathbf{R_X}\mathbf{T}^T && (8.41)\\ &= \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ 0 & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \lambda_k \end{bmatrix}. && (8.42) \end{aligned}$$

Clearly, we have the required autocorrelation matrix for $\mathbf{Y}$. Therefore, the KLT is optimal, in the sense that it completely decorrelates the input. In addition, since the KLT depends on the computation of the autocorrelation matrix of the input vector, it is data-dependent: it has to be computed for every dataset.

Example 8.5

To illustrate the mechanics of the KLT, consider the four 3D input vectors $\mathbf{x}_1 = (4, 4, 5)$, $\mathbf{x}_2 = (3, 2, 5)$, $\mathbf{x}_3 = (5, 7, 6)$, and $\mathbf{x}_4 = (6, 7, 7)$. To find the required transform, we must first estimate the autocorrelation matrix of the input. The mean of the four input vectors is

$$\mathbf{m}_x = \frac{1}{4}\begin{bmatrix} 18 \\ 20 \\ 23 \end{bmatrix}.$$

We can estimate the autocorrelation matrix using the formula

$$\mathbf{R_X} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^T - \mathbf{m}_x\mathbf{m}_x^T \tag{8.43}$$

where n is the number of input vectors, which is 4. From this equation, we obtain

$$\mathbf{R_X} = \begin{bmatrix} 1.25 & 2.25 & 0.88 \\ 2.25 & 4.50 & 1.50 \\ 0.88 & 1.50 & 0.69 \end{bmatrix}.$$

We are trying to diagonalize matrix $\mathbf{R_X}$, which is the same as forming an eigenvector-eigenvalue decomposition (from linear algebra). That is, we want to rewrite $\mathbf{R_X}$ as $\mathbf{R_X} = \mathbf{T}\mathbf{D}\mathbf{T}^{-1}$, where the matrix $\mathbf{D}$ is diagonal, with off-diagonal values equaling zero: $\mathbf{D} = diag(\lambda_1, \lambda_2, \lambda_3)$, with the λ values called the eigenvalues and the columns of matrix $\mathbf{T}$ called the eigenvectors. These are easy to calculate using various math libraries; in Matlab, function `eig` will do the job.

Here, the eigenvalues of $\mathbf{R_X}$ are $\lambda_1 = 6.1963$, $\lambda_2 = 0.2147$, and $\lambda_3 = 0.0264$. Clearly, the first component is by far the most important. The corresponding eigenvectors are

$$\mathbf{u}_1 = \begin{bmatrix} 0.4385 \\ 0.8471 \\ 0.3003 \end{bmatrix} \quad \mathbf{u}_2 = \begin{bmatrix} 0.4460 \\ -0.4952 \\ 0.7456 \end{bmatrix} \quad \mathbf{u}_3 = \begin{bmatrix} -0.7803 \\ 0.1929 \\ 0.5949 \end{bmatrix}.$$

Therefore, the KLT is given by the matrix

$$\mathbf{T} = \begin{bmatrix} 0.4385 & 0.8471 & 0.3003 \\ 0.4460 & -0.4952 & 0.7456 \\ -0.7803 & 0.1929 & 0.5949 \end{bmatrix}.$$

Subtracting the mean vector from each input vector and applying the KLT, we have

$$\mathbf{y}_1 = \begin{bmatrix} -1.2916 \\ -0.2870 \\ -0.2490 \end{bmatrix} \quad \mathbf{y}_2 = \begin{bmatrix} -3.4242 \\ 0.2573 \\ 0.1453 \end{bmatrix}$$

$$\mathbf{y}_3 = \begin{bmatrix} 1.9885 \\ -0.5809 \\ 0.1445 \end{bmatrix} \quad \mathbf{y}_4 = \begin{bmatrix} 2.7273 \\ 0.6107 \\ -0.0408 \end{bmatrix}.$$

Since the rows of $\mathbf{T}$ are orthonormal vectors, the inverse transform is just the transpose: $\mathbf{T}^{-1} = \mathbf{T}^T$. We can obtain the original vectors from the transform coefficients using the inverse relation

$$\mathbf{x} = \mathbf{T}^T\mathbf{y} + \mathbf{m}_x. \tag{8.44}$$

In terms of the transform coefficients $\mathbf{y}_i$, the magnitude of the first few components is usually considerably larger than that of the other components. In general, after the KLT, most of the "energy" of the transform coefficients is concentrated within the first few components. This is the *energy compaction* property of the KLT.

For an input vector $\mathbf{x}$ with n components, if we coarsely quantize the output vector $\mathbf{y}$ by setting its last k components to zero, calling the resulting vector $\hat{\mathbf{y}}$, the KLT minimizes the mean squared error between the original vector and its reconstruction.

8.6 Wavelet-Based Coding

8.6.1 Introduction

Decomposing the input signal into its constituents allows us to apply coding techniques suitable for each constituent, to improve compression performance. Consider again a time-dependent signal $f(t)$ (it is best to base discussion on continuous functions to start with). The traditional method of signal decomposition is the Fourier transform. Above, in our discussion of the DCT, we considered a special cosine-based transform. If we carry out analysis based on both sine and cosine, then a concise notation assembles the results into a function $\mathcal{F}(\omega)$, a complex-valued function

of real-valued frequency ω given in Eq. (8.32). Such decomposition results in very fine resolution in the *frequency* domain. However, since a sinusoid is theoretically infinite in extent in time, such a decomposition gives no *temporal* resolution.

Another method of decomposition that has gained a great deal of popularity in recent years is the *wavelet transform*. It seeks to represent a signal with good resolution in *both* time and frequency, by using a set of basis functions called wavelets.

There are two types of wavelet transforms: the *Continuous Wavelet Transform* (CWT) and the *Discrete Wavelet Transform* (DWT). We assume that the CWT is applied to the large class of functions $f(x)$ that are square integrable on the real line—that is, $\int [f(x)]^2 \, dx < \infty$. In mathematics, this is written as $f(x) \in \mathbf{L}^2(R)$.

The other kind of wavelet transform, the DWT, operates on discrete samples of the input signal. The DWT resembles other discrete linear transforms, such as the DFT or the DCT, and is very useful for image processing and compression.

Before we begin a discussion of the theory of wavelets, let's develop an intuition about this approach by going through an example using the simplest wavelet transform, the so-called *Haar Wavelet Transform*, to form averages and differences of a sequence of `float` values.

If we repeatedly take averages and differences and keep results for every step, we effectively create a *multiresolution analysis* of the sequence. For images, this would be equivalent to creating smaller and smaller summary images, one-quarter the size for each step, and keeping track of differences from the average as well. Mentally stacking the full-size image, the quarter-size image, the sixteenth size image, and so on, creates a *pyramid*. The full set, along with difference images, is the multiresolution decomposition.

Example 8.6 (A Simple Wavelet Transform)

The objective of the wavelet transform is to decompose the input signal, for compression purposes, into components that are easier to deal with, have special interpretations, or have some components that can be thresholded away. Furthermore, we want to be able to at least approximately reconstruct the original signal, given these components. Suppose we are given the following input sequence:

$$\{x_{n,i}\} = \{10, 13, 25, 26, 29, 21, 7, 15\} \tag{8.45}$$

here, $i \in [0 \,..\, 7]$ indexes "pixels", and n stands for the level of a *pyramid* we are on. At the top, $n = 3$ for this sequence, and we shall form three more sequences, for $n = 2$, 1, and 0. At each level, less information will be retained in the beginning elements of the transformed signal sequence. When we reach pyramid level $n = 0$, we end up with the sequence average stored in the first element. The remaining elements store detail information.

Consider the transform that replaces the original sequence with its pairwise *average* $x_{n-1,i}$ and *difference* $d_{n-1,i}$, defined as follows:

$$x_{n-1,i} = \frac{x_{n,2i} + x_{n,2i+1}}{2} \tag{8.46}$$

$$d_{n-1,i} = \frac{x_{n,2i} - x_{n,2i+1}}{2} \tag{8.47}$$

where now $i \in [0\,..\,3]$. Notice that the averages and differences are applied only on consecutive *pairs* of input sequences whose first element has an even index. Therefore, the number of elements in each set $\{x_{n-1,i}\}$ and $\{d_{n-1,i}\}$ is exactly half the number of elements in the original sequence. We can form a new sequence having length equal to that of the original sequence by concatenating the two sequences $\{x_{n-1,i}\}$ and $\{d_{n-1,i}\}$. The resulting sequence is thus

$$\{x_{n-1,i}, d_{n-1,i}\} = \{11.5, 25.5, 25, 11, -1.5, -0.5, 4, -4\} \tag{8.48}$$

where we are now at level $n - 1 = 2$. This sequence has exactly the same number of elements as the input sequence—the transform did not increase the amount of data. Since the first half of the above sequence contains averages from the original sequence, we can view it as a coarser approximation to the original signal.

The second half of this sequence can be viewed as the details or approximation errors of the first half. Most of the values in the detail sequence are much smaller than those of the original sequence. Thus, most of the energy is effectively concentrated in the first half. Therefore, we can potentially store $\{d_{n-1,i}\}$ using fewer bits.

It is easily verified that the original sequence can be reconstructed from the transformed sequence, using the relations

$$\begin{aligned} x_{n,2i} &= x_{n-1,i} + d_{n-1,i} \\ x_{n,2i+1} &= x_{n-1,i} - d_{n-1,i}. \end{aligned} \tag{8.49}$$

This transform is the discrete Haar wavelet transform. Averaging and differencing can be carried out by applying a so-called *scaling function* and *wavelet function* along the signal. Figure 8.12 shows the Haar version of these functions.

We can further apply the same transform to $\{x_{n-1,i}\}$, to obtain another level of approximation $x_{n-2,i}$ and detail $d_{n-2,i}$:

$$\{x_{n-2,i}, d_{n-2,i}, d_{n-1,i}\} = \{18.5, 18, -7, 7, -1.5, -0.5, 4, -4\}. \tag{8.50}$$

This is an essential idea of multiresolution analysis. We can now study the input signal in three different scales, along with the details needed to go from one scale to another. This process can continue n times, until only one element is left in the approximation sequence. In this case, $n = 3$, and the final sequence is given below:

$$\{x_{n-3,i}, d_{n-3,i}, d_{n-2,i}, d_{n-1,i}\} = \{18.25, 0.25, -7, 7, -1.5, -0.5, 4, -4\}. \tag{8.51}$$

Now we realize that n was 3 because only three resolution changes were available until we reached the final form.

The value 18.25, corresponding to the coarsest approximation to the original signal, is the average of all the elements in the original sequence. From this example, it is easy to see that the cost of computing this transform is proportional to the number of elements N in the input sequence—that is, $O(N)$.

Extending the one-dimensional Haar wavelet transform into two dimensions is relatively easy: we simply apply the one-dimensional transform to the rows and columns of the two-dimensional input separately. We will demonstrate the two-dimensional Haar transform applied to the 8×8 input image shown in Fig. 8.13.

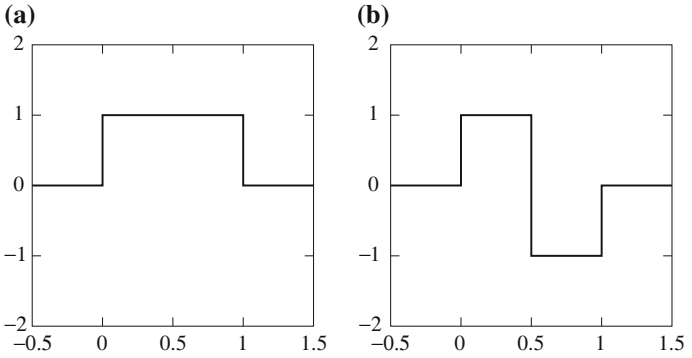

Fig. 8.12 Haar Wavelet Transform: **a** scaling function; **b** wavelet function

(a)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	63	127	127	63	0	0
0	0	127	255	255	127	0	0
0	0	127	255	255	127	0	0
0	0	63	127	127	63	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b)

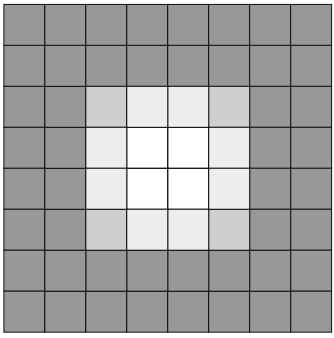

Fig. 8.13 Input image for the 2D Haar Wavelet Transform: **a** pixel values; **b** an 8×8 image

Example 8.7 (2D Haar Transform)

This example of the 2D Haar transform not only serves to illustrate how the wavelet transform is applied to two-dimensional inputs but also points out useful interpretations of the transformed coefficients. However, it is intended only to provide the reader with an intuitive feeling of the kinds of operations involved in performing a general 2D wavelet transform. Subsequent sections provide more detailed description of the forward and inverse 2D wavelet transform algorithms, as well as a more elaborate example using a more complex wavelet.

2D Haar Wavelet Transform

We begin by applying a one-dimensional Haar wavelet transform to each row of the input. The first and last two rows of the input are trivial. After performing the averaging and differencing operations on the remaining rows, we obtain the intermediate output shown in Fig. 8.14.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	95	95	0	0	–32	32	0
0	191	191	0	0	–64	64	0
0	191	191	0	0	–64	64	0
0	95	95	0	0	–32	32	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 8.14 Intermediate output of the 2D Haar Wavelet Transform

0	0	0	0	0	0	0	0
0	143	143	0	0	–48	48	0
0	143	143	0	0	–48	48	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	–48	–48	0	0	16	–16	0
0	48	48	0	0	–16	16	0
0	0	0	0	0	0	0	0

Fig. 8.15 Output of the first level of the 2D Haar Wavelet Transform

We continue by applying the same 1D Haar transform to each column of the intermediate output. This step completes one level of the 2D Haar transform. Figure 8.15 gives the resulting coefficients.

We can naturally divide the result into four quadrants. The upper left quadrant contains the averaged coefficients from both the horizontal and vertical passes. Therefore, it can be viewed as a low-pass-filtered version of the original image, in the sense that higher frequency edge information is lost, while low spatial frequency smooth information is retained.

The upper right quadrant contains the vertical averages of the horizontal differences and can be interpreted as information about the *vertical edges* within the original image. Similarly, the lower left quadrant contains the vertical differences of the horizontal averages and represents the *horizontal edges* in the original image. The lower right quadrant contains the differences from both the horizontal and vertical passes. The coefficients in this quadrant represent *diagonal edges*.

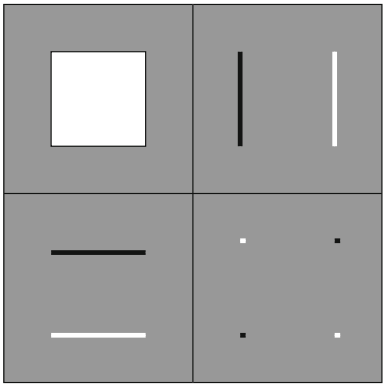

Fig. 8.16 A simple graphical illustration of the Wavelet Transform

These interpretations are shown more clearly as images in Fig. 8.16 where bright pixels code are positive and dark pixels code are negative image values.

The inverse of the 2D Haar transform can be calculated by first inverting the columns using Eq. (8.49), and then inverting the resulting rows.

8.6.2 Continuous Wavelet Transform*

We noted that the motivation for the use of wavelets is to provide a set of basis functions that decompose a signal in time over parameters in the frequency domain and the time domain simultaneously. A Fourier transform aims to pin down only the frequency content of a signal, in terms of spatially varying rather than time varying signals. What wavelets aim to do is pin down the frequency content at different parts of the image.

For example, one part of the image may be "busy" with texture and thus high-frequency content, while another part may be smooth, with little high-frequency content. Naturally, one can think of obvious ways to consider frequencies for localized areas of an image: divide an image into parts and fire away with Fourier analysis. The time-sequence version of that idea is called the *Short-Term* (or *Windowed*) *Fourier Transform*. And other ideas have also arisen. However, it turns out that wavelets, a much newer development, have neater characteristics.

To further motivate the subject, we should consider the *Heisenberg uncertainty principle*, from physics. In the context of signal processing, this says that there is a tradeoff between accuracy in pinning down a function's frequency, and its extent in time. We cannot do both accurately, in general, and still have a useful basis function. For example, a sine wave is exact in terms of its frequency but infinite in extent.

As an example of a function that dies away quickly and also has limited frequency content, suppose we start with a Gaussian function,

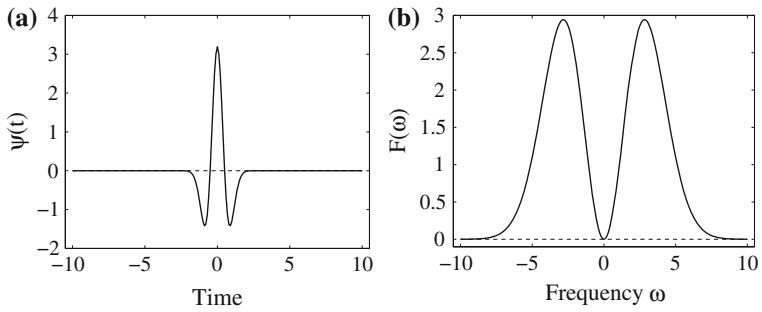

Fig. 8.17 A Mexican-hat Wavelet: **a** $\sigma = 0.5$; **b** its Fourier transform

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-t^2}{2\sigma^2}}. \tag{8.52}$$

The parameter σ expresses the *scale* of the Gaussian (bell-shaped) function.

The second derivative of this function, called $\psi(t)$, looks like a Mexican hat, as in Fig. 8.17a. Clearly, the function $\psi(t)$ is limited in time. Its equation is as follows:

$$\psi(t) = \frac{1}{\sigma^3\sqrt{2\pi}} \left[e^{\frac{-t^2}{2\sigma^2}} \left(\frac{t^2}{\sigma^2} - 1 \right) \right]. \tag{8.53}$$

We can explore the frequency content of function $\psi(t)$ by taking its Fourier transform. This turns out to be given by

$$\mathcal{F}(\omega) = \omega^2 e^{-\frac{\sigma^2\omega^2}{2}}. \tag{8.54}$$

Figure 8.17b displays this function: the candidate wavelet (8.53) is indeed limited in frequency as well.

In general, a wavelet is a function $\psi \in \mathbf{L}^2(R)$ with a zero average,

$$\int_{-\infty}^{+\infty} \psi(t)\, \mathrm{d}t = 0 \tag{8.55}$$

that satisfies some conditions that ensure it can be utilized in a multiresolution decomposition. The conditions ensure that we can use the decomposition for zooming in locally in some part of an image, much as we might be interested in closer or farther views of some neighborhood in a map.

The constraint (8.55) is called the *admissibility condition* for wavelets. A function that sums to zero must oscillate around zero. Also, from (8.32), we see that the DC value, the Fourier transform of $\psi(t)$ for $\omega = 0$, is zero. Another way to state this is that the 0th moment M_0 of $\psi(t)$ is zero. The pth moment is defined as

$$M_p = \int_{-\infty}^{\infty} t^p \psi(t)\, \mathrm{d}t. \tag{8.56}$$

The function ψ is normalized with $\|\psi\| = 1$ and centered in the neighborhood of $t = 0$. We can obtain a *family* of wavelet functions by scaling and translating the *mother wavelet* ψ as follows:

$$\psi_{s,u}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right). \tag{8.57}$$

If $\psi(t)$ is normalized, so is $\psi_{s,u}(t)$.

The *Continuous Wavelet Transform* (CWT) of $f \in \mathbf{L}^2(R)$ at time u and scale s is defined as

$$\mathcal{W}(f, s, u) = \int_{-\infty}^{+\infty} f(t)\psi_{s,u}(t)\,\mathrm{d}t \tag{8.58}$$

The CWT of a 1D signal is a 2D function—a function of both *scale* s and *shift* u.

A very important issue is that, in contradistinction to (8.32), where the Fourier analysis function is stipulated to be the sinusoid, here (8.58) does not state what $\psi(t)$ actually is! Instead, we create a set of rules such functions must obey and then invent useful functions that obey these rules—different functions for different uses.

Just as we defined the DCT in terms of products of a function with a set of basis functions, here the transform $\mathcal{W}$ is written in terms of inner products with basis functions that are a scaled and shifted version of the mother wavelet $\psi(t)$.

The mother wavelet $\psi(t)$ is a *wave*, since it must be an oscillatory function. Why is it wave*let*? The spatial-frequency analyzer parameter in (8.58) is s, the scale. We choose some scale s and see how much content the signal has around that scale. To make the function decay rapidly, away from the chosen s, we have to choose a mother wavelet $\psi(t)$ that decays as fast as some power of s.

It is actually easy to show, from (8.58), that if all moments of $\psi(t)$ up to the nth are zero (or quite small, practically speaking), then the CWT coefficient $\mathcal{W}(f, s, u)$ has a Taylor expansion around $u = 0$ that is of order s^{n+2} (see Exercise 12). This is the localization in frequency we desire in a good mother wavelet.

We derive wavelet coefficients by applying wavelets at different scales over many locations of the signal. Excitingly, if we shrink the wavelets down small enough that they cover a part of the function $f(t)$ that is a polynomial of degree n or less, the coefficient for that wavelet and all smaller ones will be zero. The condition that the wavelet should have vanishing moments up to some order is one way of characterizing mathematical *regularity conditions* on the mother wavelet.

The inverse of the continuous wavelet transform is:

$$f(t) = \frac{1}{C_\psi}\int_0^{+\infty}\int_{-\infty}^{+\infty}\mathcal{W}(f, s, u)\frac{1}{\sqrt{s}}\,\psi\left(\frac{t-u}{s}\right)\frac{1}{s^2}\,\mathrm{d}u\,\mathrm{d}s \tag{8.59}$$

where

$$C_\psi = \int_0^{+\infty}\frac{|\Psi(\omega)|^2}{\omega}\,\mathrm{d}\omega < +\infty \tag{8.60}$$

and $\Psi(\omega)$ is the Fourier transform of $\psi(t)$. Eq. (8.60) is another phrasing of the admissibility condition.

The trouble with the CWT is that (8.58) is nasty: most wavelets are not analytic but result simply from numerical calculations. The resulting infinite set of scaled and

shifted functions is not necessary for the analysis of *sampled* functions, such as the ones arise in image processing. For this reason, we apply the ideas that pertain to the CWT to the discrete domain.

8.6.3 Discrete Wavelet Transform*

Discrete wavelets are again formed from a mother wavelet, but with scale and shift in discrete steps.

Multiresolution Analysis and the Discrete Wavelet Transform

The connection between wavelets in the continuous time domain and *filter banks* in the discrete time domain is multiresolution analysis; we discuss the DWT within this framework. Mallat [10] showed that it is possible to construct wavelets ψ such that the dilated and translated family

$$\left\{\psi_{j,n}(t) = \frac{1}{\sqrt{2^j}}\,\psi\left(\frac{t-2^j n}{2^j}\right)\right\}_{(j,n)\in\mathbf{Z}^2} \tag{8.61}$$

is an *orthonormal basis* of $\mathbf{L}^2(R)$, where $\mathbf{Z}$ represents the set of integers. This is known as "dyadic" scaling and translation and corresponds to the notion of zooming out in a map by factors of 2. (If we draw a cosine function $\cos(t)$ from time 0 to 2π and then draw $\cos(t/2)$, we see that while $\cos(t)$ goes over a whole cycle, $\cos(t/2)$ has only a half cycle: the function $\cos(2^{-1}t)$ is a *wider* function and thus is at a broader scale.)

Note that, we change the scale of translations along with the overall scale 2^j, so as to keep movement in the lower resolution image in proportion. Notice also that the notation used says that a larger index j corresponds to a coarser version of the image.

Multiresolution analysis provides the tool to *adapt signal resolution to only relevant details* for a particular task. The *octave decomposition* introduced by Mallat [11] initially decomposes a signal into an approximation component and a detail component. The approximation component is then recursively decomposed into approximation and detail at successively coarser scales. Wavelets are set up such that the approximation at resolution 2^{-j} contains all the necessary information to compute an approximation at coarser resolution $2^{-(j+1)}$.

Wavelets are used to characterize detail information. The averaging information is formally determined by a kind of dual to the mother wavelet, called the *scaling function* $\phi(t)$.

The main idea in the theory of wavelets is that at a particular level of resolution j, the set of *translates* indexed by n form a basis at that level. Interestingly, the set of translates forming the basis at the $j+1$ next level, a coarser level, can all be written as a sum of weights times the level-j basis. The scaling function is chosen such that the coefficients of its translates are all necessarily bounded (less than infinite).

The scaling function, along with its translates, forms a basis at the coarser level $j + 1$ (say 3, or the 1/8 level) but not at level j (say 2, or the 1/4 level). Instead, at level j the set of translates of the scaling function ϕ *along with the set of translates of the mother wavelet* ϕ do form a basis. We are left with the situation that the scaling function describes smooth, or approximation, information, and the wavelet describes what is left over—detail information.

Since the set of translates of the scaling function ϕ at a coarser level can be written exactly as a weighted sum of the translates at a finer level, the scaling function must satisfy the so-called *dilation equation* [12]:

$$\phi(t) = \sum_{n \in \mathbf{Z}} \sqrt{2} h_0[n] \phi(2t - n) \tag{8.62}$$

The square brackets come from the theory of *filters*, and their use is carried over here. The dilation equation is a recipe for finding a function that can be built from a sum of copies of itself that are first scaled, translated, and dilated. Equation (8.62) expresses a condition that a function must satisfy to be a scaling function and at the same time forms a definition of the *scaling vector* h_0.

Not only is the scaling function expressible as a sum of translates, but as well the *wavelet* at the coarser level is also expressible as such:

$$\psi(t) = \sum_{n \in \mathbf{Z}} \sqrt{2} h_1[n] \phi(2t - n) \tag{8.63}$$

Below, we'll show that the set of coefficients h_1 for the wavelet can in fact be derived from the scaling function ones h_0 [Eq. (8.65) below], so we also have that the wavelet can be derived from the scaling function, once we have one. The equation reads

$$\psi(t) = \sum_{n \in \mathbf{Z}} (-1)^n h_0[1 - n] \phi(2t - n) \tag{8.64}$$

So the condition on a wavelet is similar to that on the scaling function, Eq. (8.62), and in fact uses the same coefficients, only in the opposite order and with alternating signs.

Clearly, for efficiency, we would like the sums in (8.62) and (8.63) to be as few as possible, so we choose wavelets that have as few vector entries h_0 and h_1 as possible. The effect of the scaling function is a kind of smoothing, or filtering, operation on a signal. Therefore, it acts as a low-pass filter, screening out high-frequency content. The vector values $h_0[n]$ are called the low-pass filter *impulse response* coefficients, since they describe the effect of the filtering operation on a signal consisting of a single spike with magnitude unity (an impulse) at time $t = 0$. A complete discrete signal is made of a set of such spikes, shifted in time from 0 and weighted by the magnitudes of the discrete samples.

Hence, to specify a DWT, only the discrete low-pass filter impulse response $h_0[n]$ is needed. These specify the approximation filtering, given by the scaling function. The discrete *high-pass* impulse response $h_1[n]$, describing the details using the wavelet function, can be derived from $h_0[n]$ using the following equation:

$$h_1[n] = (-1)^n h_0[1 - n] \tag{8.65}$$

Table 8.2 Orthogonal wavelet filters

Wavelet	Number of taps	Start index	Coefficients
Haar	2	0	[0.707, 0.707]
Daubechies 4	4	0	[0.483, 0.837, 0.224, −0.129]
Daubechies 6	6	0	[0.332, 0.807, 0.460, −0.135, −0.085, 0.0352]
Daubechies 8	8	0	[0.230, 0.715, 0.631, −0.028, −0.187, 0.031, 0.033, −0.011]

The number of coefficients in the impulse response is called the number of *taps* in the filter. If $h_0[n]$ has only a finite number of nonzero entries, the resulting wavelet is said to have *compact support*. Additional constraints, such as orthonormality and regularity, can be imposed on the coefficients $h_0[n]$. The vectors $h_0[n]$ and $h_1[n]$ are called the low-pass and high-pass *analysis* filters.

To *reconstruct* the original input, an inverse operation is needed. The inverse filters are called *synthesis* filters. For orthonormal wavelets, the forward transform and its inverse are transposes of each other, and the analysis filters are identical to the synthesis filters.

Without orthogonality, the wavelets for analysis and synthesis are called *biorthogonal*, a weaker condition. In this case, the synthesis filters are not identical to the analysis filters. We denote them as $\tilde{h}_0[n]$ and $\tilde{h}_1[n]$. To specify a biorthogonal wavelet transform, we require both $h_0[n]$ and $\tilde{h}_0[n]$. As before, we can compute the discrete high-pass filters in terms of sums of the low-pass ones:

$$h_1[n] = (-1)^n \tilde{h}_0[1-n] \tag{8.66}$$

$$\tilde{h}_1[n] = (-1)^n h_0[1-n] \tag{8.67}$$

Tables 8.2 and 8.3 (cf. [13]) give some commonly used orthogonal and biorthogonal wavelet filters. The "start index" columns in these tables refer to the starting value of the index n used in Eqs. (8.66) and (8.67).

Figure 8.18 shows a block diagram for the 1D dyadic wavelet transform. Here, $x[n]$ is the discrete sampled signal. The box $\boxed{\downarrow 2}$ means subsampling by taking every second element, and the box $\boxed{\uparrow 2}$ means upsampling by replication. The reconstruction phase yields series $y[n]$.

For analysis, at each level we transform the series $x[n]$ into another series of the same length, in which the first half of the elements is approximation information and the second half consists of detail information. For an N-tap filter, this is simply the series

$$\{x[n]\} \;\rightarrow\; y[n] = \left\{ \sum_j x[j] h_0[n-j] \;;\; \sum_j x[j] h_1[n-j] \right\} \tag{8.68}$$

where for each half, the odd-numbered results are discarded. The summation over shifted coefficients in (8.68) is referred to as a *convolution*.

Table 8.3 Biorthogonal wavelet filters

Wavelet	Filter	Number of taps	Start index	Coefficients
Antonini 9/7	$h_0[n]$	9	−4	[0.038, −0.024, −0.111, 0.377, 0.853, 0.377, −0.111, −0.024, 0.038]
	$\tilde{h}_0[n]$	7	−3	[−0.065, −0.041, 0.418, 0.788, 0.418, −0.041, −0.065]
Villa 10/18	$h_0[n]$	10	−4	[0.029, 0.0000824, −0.158, 0.077, 0.759, 0.759, 0.077, −0.158, 0.0000824, 0.029]
	$\tilde{h}_0[n]$	18	−8	[0.000954, −0.00000273, −0.009, −0.003, 0.031, −0.014, −0.086, 0.163, 0.623, 0.623, 0.163, −0.086, −0.014, 0.031, −0.003, −0.009, −0.00000273, 0.000954]
Brislawn	$h_0[n]$	10	−4	[0.027, −0.032, −0.241, 0.054, 0.900, 0.900, 0.054, −0.241, −0.032, 0.027]
	$\tilde{h}_0[n]$	10	−4	[0.020, 0.024, −0.023, 0.146, 0.541, 0.541, 0.146, −0.023, 0.024, 0.020]

2D Discrete Wavelet Transform

The extension of the wavelet transform to two dimensions is quite straightforward. A two-dimensional scaling function is said to be *separable* if it can be factored into a product of two one-dimensional scaling functions. That is,

$$\phi(x, y) = \phi(x)\phi(y) \tag{8.69}$$

For simplicity, only separable wavelets are considered in this section. Furthermore, let's assume that the width and height of the input image are powers of 2.

For an N by N input image, the two-dimensional DWT proceeds as follows:

1. Convolve each row of the image with $h_0[n]$ and $h_1[n]$, discard the odd-numbered columns of the resulting arrays, and concatenate them to form a transformed row.
2. After all rows have been transformed, convolve each column of the result with $h_0[n]$ and $h_1[n]$. Again discard the odd-numbered rows and concatenate the result.

After the above two steps, one stage of the DWT is complete. The transformed image now contains four subbands LL, HL, LH, and HH, standing for low-low, high-low, and so on, as Fig. 8.19a shows. As in the one-dimensional transform, the LL subband can be further decomposed to yield yet another level of decomposition. This process can be continued until the desired number of decomposition levels is reached or the LL component only has a single element left. A two-level decomposition is shown in Fig. 8.19b.

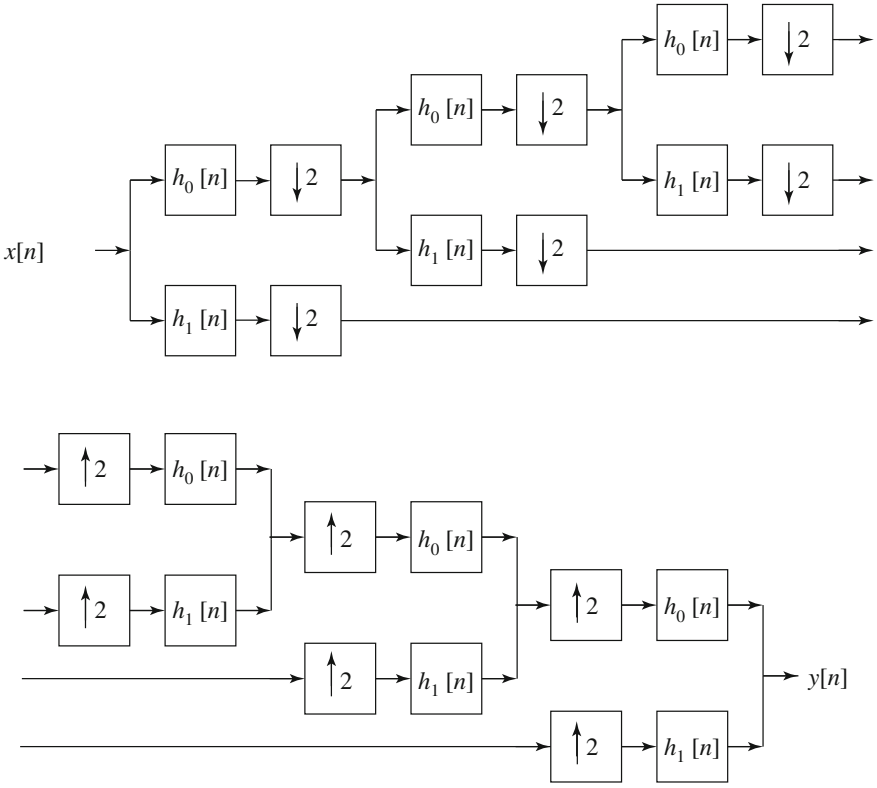

Fig. 8.18 Block diagram of the 1D dyadic wavelet transform

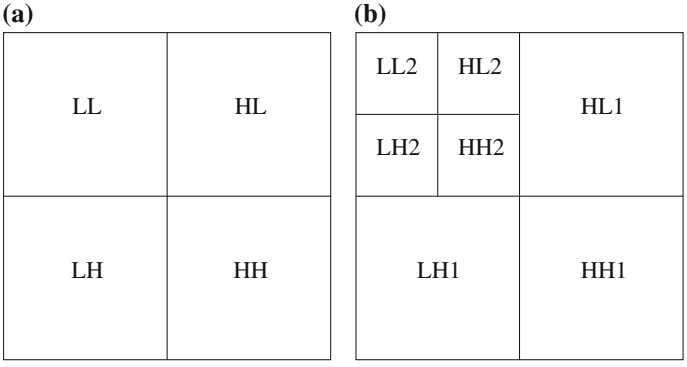

Fig. 8.19 The two-dimensional discrete wavelet transform: **a** one-level transform; **b** two-level transform

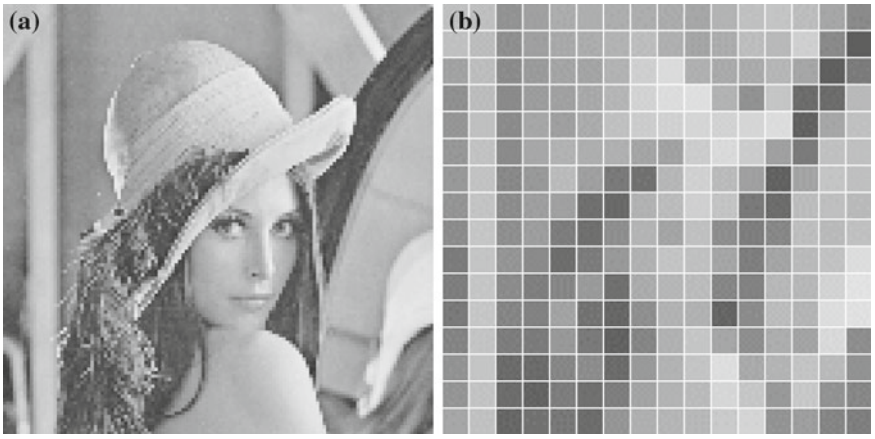

Fig. 8.20 Lena: **a** original 128 × 128 image; **b** 16 × 16 subsampled image

The inverse transform simply reverses the steps of the forward transform.

1. For each stage of the transformed image, starting with the last, separate each column into low-pass and high-pass coefficients. Upsample each of the low-pass and high-pass arrays by inserting a zero after each coefficient.
2. Convolve the low-pass coefficients with $h_0[n]$ and high-pass coefficients with $h_1[n]$ and add the two resulting arrays.
3. After all columns have been processed, separate each row into low-pass and high-pass coefficients and upsample each of the two arrays by inserting a zero after each coefficient.
4. Convolve the low-pass coefficients with $h_0[n]$ and high-pass coefficients with $h_1[n]$ and add the two resulting arrays.

If biorthogonal filters are used for the forward transform, we must replace the $h_0[n]$ and $h_1[n]$ above with $\tilde{h}_0[n]$ and $\tilde{h}_1[n]$ in the inverse transform.

Example 8.8

The input image is a subsampled version of the image `Lena`, as shown in Fig. 8.20. The size of the input is 16 × 16. The filter used in the example is the Antonini 9/7 filter set given in Table 8.3.

Before we begin, we need to compute the analysis and synthesis high-pass filters using Eqs. (8.66) and (8.67). The resulting filter coefficients are

$$\begin{aligned} h_1[n] &= [-0.065, 0.041, 0.418, -0.788, 0.418, 0.041, -0.065] \\ \tilde{h}_1[n] &= [-0.038, -0.024, 0.111, 0.377, -0.853, 0.377, 0.111, -0.024, -0.038] \end{aligned} \tag{8.70}$$

The input image in numerical form is

$$I_{00}(x, y) = \begin{bmatrix} 158 & 170 & 97 & 104 & 123 & 130 & 133 & 125 & 132 & 127 & 112 & 158 & 159 & 144 & 116 & 91 \\ 164 & 153 & 91 & 99 & 124 & 152 & 131 & 160 & 189 & 116 & 106 & 145 & 140 & 143 & 227 & 53 \\ 116 & 149 & 90 & 101 & 118 & 118 & 131 & 152 & 202 & 211 & 84 & 154 & 127 & 146 & 58 & 58 \\ 95 & 145 & 88 & 105 & 188 & 123 & 117 & 182 & 185 & 204 & 203 & 154 & 153 & 229 & 46 & 147 \\ 101 & 156 & 89 & 100 & 165 & 113 & 148 & 170 & 163 & 186 & 144 & 194 & 208 & 39 & 113 & 159 \\ 103 & 153 & 94 & 103 & 203 & 136 & 146 & 92 & 66 & 192 & 188 & 103 & 178 & 47 & 167 & 159 \\ 102 & 146 & 106 & 99 & 99 & 121 & 39 & 60 & 164 & 175 & 198 & 46 & 56 & 56 & 156 & 156 \\ 99 & 146 & 95 & 97 & 144 & 61 & 103 & 107 & 108 & 111 & 192 & 62 & 65 & 128 & 153 & 154 \\ 99 & 140 & 103 & 109 & 103 & 124 & 54 & 81 & 172 & 137 & 178 & 54 & 43 & 159 & 149 & 174 \\ 84 & 133 & 107 & 84 & 149 & 43 & 158 & 95 & 151 & 120 & 183 & 46 & 30 & 147 & 142 & 201 \\ 58 & 153 & 110 & 41 & 94 & 213 & 71 & 73 & 140 & 103 & 138 & 83 & 152 & 143 & 128 & 207 \\ 56 & 141 & 108 & 58 & 92 & 51 & 55 & 61 & 88 & 166 & 58 & 103 & 146 & 150 & 116 & 211 \\ 89 & 115 & 188 & 47 & 113 & 104 & 56 & 67 & 128 & 155 & 187 & 71 & 153 & 134 & 203 & 95 \\ 35 & 99 & 151 & 67 & 35 & 88 & 88 & 128 & 140 & 142 & 176 & 213 & 144 & 128 & 214 & 100 \\ 89 & 98 & 97 & 51 & 49 & 101 & 47 & 90 & 136 & 136 & 157 & 205 & 106 & 43 & 54 & 76 \\ 44 & 105 & 69 & 69 & 68 & 53 & 110 & 127 & 134 & 146 & 159 & 184 & 109 & 121 & 72 & 113 \end{bmatrix}$$

I represents the pixel values. The first subscript of I indicates the current stage of the transform, while the second subscript indicates the current step within a stage. We start by convolving the first row with both $h_0[n]$ and $h_1[n]$ and discarding the values with odd-numbered index. The results of these two operations are

$$(I_{00}(:, 0) * h_0[n]) \downarrow 2 = [245, 156, 171, 183, 184, 173, 228, 160]$$
$$(I_{00}(:, 0) * h_1[n]) \downarrow 2 = [-30, 3, 0, 7, -5, -16, -3, 16]$$

where the colon in the first index position indicates that we are showing a whole row. If you like, you can verify these operations using MATLAB's `conv` function.

Next, we form the transformed output row by concatenating the resulting coefficients. The first row of the transformed image is then

$$[245, 156, 171, 183, 184, 173, 228, 160, -30, 3, 0, 7, -5, -16, -3, 16].$$

Similar to the simple one-dimensional Haar transform examples, most of the energy is now concentrated on the first half of the transformed image. We continue the same process for the remaining rows and obtain the following result:

$$I_{11}(x, y) =$$

$$\begin{bmatrix}
245 & 156 & 171 & 183 & 184 & 173 & 228 & 160 & -30 & 3 & 0 & 7 & -5 & -16 & -3 & 16 \\
239 & 141 & 181 & 197 & 242 & 158 & 202 & 229 & -17 & 5 & -20 & 3 & 26 & -27 & 27 & 141 \\
195 & 147 & 163 & 177 & 288 & 173 & 209 & 106 & -34 & 2 & 2 & 19 & -50 & -35 & -38 & -1 \\
180 & 139 & 226 & 177 & 274 & 267 & 247 & 163 & -45 & 29 & 24 & -29 & -2 & 30 & -101 & -78 \\
191 & 145 & 197 & 198 & 247 & 230 & 239 & 143 & -49 & 22 & 36 & -11 & -26 & -14 & 101 & -54 \\
192 & 145 & 237 & 184 & 135 & 253 & 169 & 192 & -47 & 38 & 36 & 4 & -58 & 66 & 94 & -4 \\
176 & 159 & 156 & 77 & 204 & 232 & 51 & 196 & -31 & 9 & -48 & 30 & 11 & 58 & 29 & 4 \\
179 & 148 & 162 & 129 & 146 & 213 & 92 & 217 & -39 & 18 & 50 & -10 & 33 & 51 & -23 & 8 \\
169 & 159 & 163 & 97 & 204 & 202 & 85 & 234 & -29 & 1 & -42 & 23 & 37 & 41 & -56 & -5 \\
155 & 153 & 149 & 159 & 176 & 204 & 65 & 236 & -32 & 32 & 85 & 39 & 38 & 44 & -54 & -31 \\
145 & 148 & 158 & 148 & 164 & 157 & 188 & 215 & -55 & 59 & -110 & 28 & 26 & 48 & -1 & -64 \\
134 & 152 & 102 & 70 & 153 & 126 & 199 & 207 & -47 & 38 & 13 & 10 & -76 & 3 & -7 & -76 \\
127 & 203 & 130 & 94 & 171 & 218 & 171 & 228 & 12 & 88 & -27 & 15 & 1 & 76 & 24 & 85 \\
70 & 188 & 63 & 144 & 191 & 257 & 215 & 232 & -5 & 24 & -28 & -9 & 19 & -46 & 36 & 91 \\
129 & 124 & 87 & 96 & 177 & 236 & 162 & 77 & -2 & 20 & -48 & 1 & 17 & -56 & 30 & -24 \\
103 & 115 & 85 & 142 & 188 & 234 & 184 & 132 & -37 & 0 & 27 & -4 & 5 & -35 & -22 & -33
\end{bmatrix}.$$

We now go on and apply the filters to the columns of the above resulting image. As before, we apply both $h_0[n]$ and $h_1[n]$ to each column and discard the odd indexed results:

$$(I_{11}(0, :) * h_0[n]) \downarrow 2 = [353, 280, 269, 256, 240, 206, 160, 153]^T$$
$$(I_{11}(0, :) * h_1[n]) \downarrow 2 = [-12, 10, -7, -4, 2, -1, 43, 16]^T.$$

Concatenating the above results into a single column and applying the same procedure to each of the remaining columns, we arrive at the final transformed image:

$$I_{12}(x, y) =$$

$$\begin{bmatrix}
353 & 212 & 251 & 272 & 281 & 234 & 308 & 289 & -33 & 6 & -15 & 5 & 24 & -29 & 38 & 120 \\
280 & 203 & 254 & 250 & 402 & 269 & 297 & 207 & -45 & 11 & -2 & 9 & -31 & -26 & -74 & 23 \\
269 & 202 & 312 & 280 & 316 & 353 & 337 & 227 & -70 & 43 & 56 & -23 & -41 & 21 & 82 & -81 \\
256 & 217 & 247 & 155 & 236 & 328 & 114 & 283 & -52 & 27 & -14 & 23 & -2 & 90 & 49 & 12 \\
240 & 221 & 226 & 172 & 264 & 294 & 113 & 330 & -41 & 14 & 31 & 23 & 57 & 60 & -78 & -3 \\
206 & 204 & 201 & 192 & 230 & 219 & 232 & 300 & -76 & 67 & -53 & 40 & 4 & 46 & -18 & -107 \\
160 & 275 & 150 & 135 & 244 & 294 & 267 & 331 & -2 & 90 & -17 & 10 & -24 & 49 & 29 & 89 \\
153 & 189 & 113 & 173 & 260 & 342 & 256 & 176 & -20 & 18 & -38 & -4 & 24 & -75 & 25 & -5 \\
-12 & 7 & -9 & -13 & -6 & 11 & 12 & -69 & -10 & -1 & 14 & 6 & -38 & 3 & -45 & -99 \\
10 & 3 & -31 & 16 & -1 & -51 & -10 & -30 & 2 & -12 & 0 & 24 & -32 & -45 & 109 & 42 \\
-7 & 5 & -44 & -35 & 67 & -10 & -17 & -15 & 3 & -15 & -28 & 0 & 41 & -30 & -18 & -19 \\
-4 & 9 & -1 & -37 & 41 & 6 & -33 & 2 & 9 & -12 & -67 & 31 & -7 & 3 & 2 & 0 \\
2 & -3 & 9 & -25 & 2 & -25 & 60 & -8 & -11 & -4 & -123 & -12 & -6 & -4 & 14 & -12 \\
-1 & 22 & 32 & 46 & 10 & 48 & -11 & 20 & 19 & 32 & -59 & 9 & 70 & 50 & 16 & 73 \\
43 & -18 & 32 & -40 & -13 & -23 & -37 & -61 & 8 & 22 & 2 & 13 & -12 & 43 & -8 & -45 \\
16 & 2 & -6 & -32 & -7 & 5 & -13 & -50 & 24 & 7 & -61 & 2 & 11 & -33 & 43 & 1
\end{bmatrix}.$$

This completes one stage of the Discrete Wavelet Transform. We can perform another stage by applying the same transform procedure to the upper left 8×8 DC image of $I_{12}(x, y)$. The resulting two-stage transformed image is

$I_{22}(x, y) =$

$$\begin{bmatrix}
558 & 451 & 608 & 532 & 75 & 26 & 94 & 25 & -33 & 6 & -15 & 5 & 24 & -29 & 38 & 120 \\
463 & 511 & 627 & 566 & 66 & 68 & -43 & 68 & -45 & 11 & -2 & 9 & -31 & -26 & -74 & 23 \\
464 & 401 & 478 & 416 & 14 & 84 & -97 & -229 & -70 & 43 & 56 & -23 & -41 & 21 & 82 & -81 \\
422 & 335 & 477 & 553 & -88 & 46 & -31 & -6 & -52 & 27 & -14 & 23 & -2 & 90 & 49 & 12 \\
14 & 33 & -56 & 42 & 22 & -43 & -36 & 1 & -41 & 14 & 31 & 23 & 57 & 60 & -78 & -3 \\
-13 & 36 & 54 & 52 & 12 & -21 & 51 & 70 & -76 & 67 & -53 & 40 & 4 & 46 & -18 & -107 \\
25 & -20 & 25 & -7 & -35 & 35 & -56 & -55 & -2 & 90 & -17 & 10 & -24 & 49 & 29 & 89 \\
46 & 37 & -51 & 51 & -44 & 26 & 39 & -74 & -20 & 18 & -38 & -4 & 24 & -75 & 25 & -5 \\
-12 & 7 & -9 & -13 & -6 & 11 & 12 & -69 & -10 & -1 & 14 & 6 & -38 & 3 & -45 & -99 \\
10 & 3 & -31 & 16 & -1 & -51 & -10 & -30 & 2 & -12 & 0 & 24 & -32 & -45 & 109 & 42 \\
-7 & 5 & -44 & -35 & 67 & -10 & -17 & -15 & 3 & -15 & -28 & 0 & 41 & -30 & -18 & -19 \\
-4 & 9 & -1 & -37 & 41 & 6 & -33 & 2 & 9 & -12 & -67 & 31 & -7 & 3 & 2 & 0 \\
2 & -3 & 9 & -25 & 2 & -25 & 60 & -8 & -11 & -4 & -123 & -12 & -6 & -4 & 14 & -12 \\
-1 & 22 & 32 & 46 & 10 & 48 & -11 & 20 & 19 & 32 & -59 & 9 & 70 & 50 & 16 & 73 \\
43 & -18 & 32 & -40 & -13 & -23 & -37 & -61 & 8 & 22 & 2 & 13 & -12 & 43 & -8 & -45 \\
16 & 2 & -6 & -32 & -7 & 5 & -13 & -50 & 24 & 7 & -61 & 2 & 11 & -33 & 43 & 1
\end{bmatrix}.$$

Notice that I_{12} corresponds to the subband diagram shown in Fig. 8.19a, and I_{22} corresponds to Fig. 8.19b. At this point, we may apply *different levels of quantization* to each subband according to some preferred bit allocation algorithm, given a desired bitrate. *This is the basis for a simple wavelet-based compression algorithm.* However, since in this example we are illustrating the mechanics of the DWT, here we will simply bypass the quantization step and perform an inverse transform to reconstruct the input image.

We refer to the top left 8×8 block of values as the innermost stage in correspondence with Fig. 8.19. Starting with the innermost stage, we extract the first column and separate the low-pass and high-pass coefficients. The low-pass coefficient is simply the first half of the column, and the high-pass coefficients are the second half. Then, we upsample them by appending a zero after each coefficient. The two resulting arrays are

$$\vec{a} = [558, 0, 463, 0, 464, 0, 422, 0]^T$$
$$\vec{b} = [14, 0, -13, 0, 25, 0, 46, 0]^T.$$

Since we are using biorthogonal filters, we convolve $\vec{a}$ and $\vec{b}$ with $\tilde{h}_0[n]$ and $\tilde{h}_1[n]$ respectively. The results of the two convolutions are then added to form a single 8×1 array. The resulting column is

$$[414, 354, 323, 338, 333, 294, 324, 260]^T.$$

All columns in the innermost stage are processed in this manner. The resulting image is

$$I'_{21}(x, y) =$$

$$\begin{bmatrix}
414 & 337 & 382 & 403 & 70 & -16 & 48 & 12 & -33 & 6 & -15 & 5 & 24 & -29 & 38 & 120 \\
354 & 322 & 490 & 368 & 39 & 59 & 63 & 55 & -45 & 11 & -2 & 9 & -31 & -26 & -74 & 23 \\
323 & 395 & 450 & 442 & 62 & 25 & -26 & 90 & -70 & 43 & 56 & -23 & -41 & 21 & 82 & -81 \\
338 & 298 & 346 & 296 & 23 & 77 & -117 & -131 & -52 & 27 & -14 & 23 & -2 & 90 & 49 & 12 \\
333 & 286 & 364 & 298 & 4 & 67 & -75 & -176 & -41 & 14 & 31 & 23 & 57 & 60 & -78 & -3 \\
294 & 279 & 308 & 350 & -2 & 17 & 12 & -53 & -76 & 67 & -53 & 40 & 4 & 46 & -18 & -107 \\
324 & 240 & 326 & 412 & -96 & 54 & -25 & -45 & -2 & 90 & -17 & 10 & -24 & 49 & 29 & 89 \\
260 & 189 & 382 & 359 & -47 & 14 & -63 & 69 & -20 & 18 & -38 & -4 & 24 & -75 & 25 & -5 \\
-12 & 7 & -9 & -13 & -6 & 11 & 12 & -69 & -10 & -1 & 14 & 6 & -38 & 3 & -45 & -99 \\
10 & 3 & -31 & 16 & -1 & -51 & -10 & -30 & 2 & -12 & 0 & 24 & -32 & -45 & 109 & 42 \\
-7 & 5 & -44 & -35 & 67 & -10 & -17 & -15 & 3 & -15 & -28 & 0 & 41 & -30 & -18 & -19 \\
-4 & 9 & -1 & -37 & 41 & 6 & -33 & 2 & 9 & -12 & -67 & 31 & -7 & 3 & 2 & 0 \\
2 & -3 & 9 & -25 & 2 & -25 & 60 & -8 & -11 & -4 & -123 & -12 & -6 & -4 & 14 & -12 \\
-1 & 22 & 32 & 46 & 10 & 48 & -11 & 20 & 19 & 32 & -59 & 9 & 70 & 50 & 16 & 73 \\
43 & -18 & 32 & -40 & -13 & -23 & -37 & -61 & 8 & 22 & 2 & 13 & -12 & 43 & -8 & -45 \\
16 & 2 & -6 & -32 & -7 & 5 & -13 & -50 & 24 & 7 & -61 & 2 & 11 & -33 & 43 & 1
\end{bmatrix}.$$

We are now ready to process the rows. For each row of the upper left 8×8 sub-image, we again separate them into low-pass and high-pass coefficients. Then we upsample both by adding a zero after each coefficient. The results are convolved with the appropriate $\tilde{h}_0[n]$ and $\tilde{h}_1[n]$ filters. After these steps are completed for all rows, we have

$$I'_{12}(x, y) =$$

$$\begin{bmatrix}
353 & 212 & 251 & 272 & 281 & 234 & 308 & 289 & -33 & 6 & -15 & 5 & 24 & -29 & 38 & 120 \\
280 & 203 & 254 & 250 & 402 & 269 & 297 & 207 & -45 & 11 & -2 & 9 & -31 & -26 & -74 & 23 \\
269 & 202 & 312 & 280 & 316 & 353 & 337 & 227 & -70 & 43 & 56 & -23 & -41 & 21 & 82 & -81 \\
256 & 217 & 247 & 155 & 236 & 328 & 114 & 283 & -52 & 27 & -14 & 23 & -2 & 90 & 49 & 12 \\
240 & 221 & 226 & 172 & 264 & 294 & 113 & 330 & -41 & 14 & 31 & 23 & 57 & 60 & -78 & -3 \\
206 & 204 & 201 & 192 & 230 & 219 & 232 & 300 & -76 & 67 & -53 & 40 & 4 & 46 & -18 & -107 \\
160 & 275 & 150 & 135 & 244 & 294 & 267 & 331 & -2 & 90 & -17 & 10 & -24 & 49 & 29 & 89 \\
153 & 189 & 113 & 173 & 260 & 342 & 256 & 176 & -20 & 18 & -38 & -4 & 24 & -75 & 25 & -5 \\
-12 & 7 & -9 & -13 & -6 & 11 & 12 & -69 & -10 & -1 & 14 & 6 & -38 & 3 & -45 & -99 \\
10 & 3 & -31 & 16 & -1 & -51 & -10 & -30 & 2 & -12 & 0 & 24 & -32 & -45 & 109 & 42 \\
-7 & 5 & -44 & -35 & 67 & -10 & -17 & -15 & 3 & -15 & -28 & 0 & 41 & -30 & -18 & -19 \\
-4 & 9 & -1 & -37 & 41 & 6 & -33 & 2 & 9 & -12 & -67 & 31 & -7 & 3 & 2 & 0 \\
2 & -3 & 9 & -25 & 2 & -25 & 60 & -8 & -11 & -4 & -123 & -12 & -6 & -4 & 14 & -12 \\
-1 & 22 & 32 & 46 & 10 & 48 & -11 & 20 & 19 & 32 & -59 & 9 & 70 & 50 & 16 & 73 \\
43 & -18 & 32 & -40 & -13 & -23 & -37 & -61 & 8 & 22 & 2 & 13 & -12 & 43 & -8 & -45 \\
16 & 2 & -6 & -32 & -7 & 5 & -13 & -50 & 24 & 7 & -61 & 2 & 11 & -33 & 43 & 1
\end{bmatrix}.$$

We then repeat the same inverse transform procedure on $I'_{12}(x, y)$, to obtain $I'_{00}(x, y)$. Notice that $I'_{00}(x, y)$ is not exactly the same as $I_{00}(x, y)$, but the difference is small.

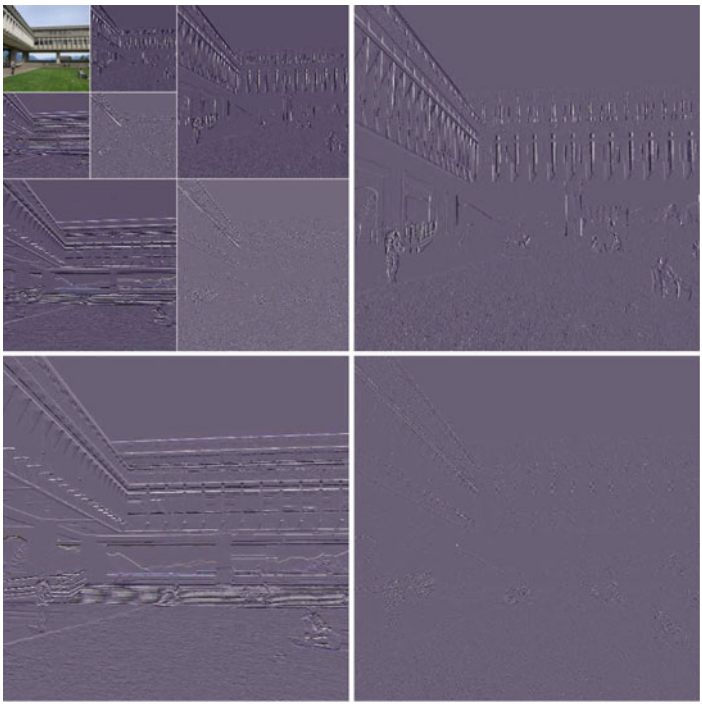

Fig. 8.21 Haar Wavelet Decomposition

These small differences are caused by round-off errors during the forward and inverse transform, and truncation errors when converting from floating point numbers to integer grayscale values.

$$I'_{00}(x, y) =$$

$$\begin{bmatrix}
158 & 170 & 97 & 103 & 122 & 129 & 132 & 125 & 132 & 126 & 111 & 157 & 159 & 144 & 116 & 91 \\
164 & 152 & 90 & 98 & 123 & 151 & 131 & 159 & 188 & 115 & 106 & 145 & 140 & 143 & 227 & 52 \\
115 & 148 & 89 & 100 & 117 & 118 & 131 & 151 & 201 & 210 & 84 & 154 & 127 & 146 & 58 & 58 \\
94 & 144 & 88 & 104 & 187 & 123 & 117 & 181 & 184 & 203 & 202 & 153 & 152 & 228 & 45 & 146 \\
100 & 155 & 88 & 99 & 164 & 112 & 147 & 169 & 163 & 186 & 143 & 193 & 207 & 38 & 112 & 158 \\
103 & 153 & 93 & 102 & 203 & 135 & 145 & 91 & 66 & 192 & 188 & 103 & 177 & 46 & 166 & 158 \\
102 & 146 & 106 & 99 & 99 & 121 & 39 & 60 & 164 & 175 & 198 & 46 & 56 & 56 & 156 & 156 \\
99 & 146 & 95 & 97 & 143 & 60 & 102 & 106 & 107 & 110 & 191 & 61 & 65 & 128 & 153 & 154 \\
98 & 139 & 102 & 109 & 103 & 123 & 53 & 80 & 171 & 136 & 177 & 53 & 43 & 158 & 148 & 173 \\
84 & 133 & 107 & 84 & 148 & 42 & 157 & 94 & 150 & 119 & 182 & 45 & 29 & 146 & 141 & 200 \\
57 & 152 & 109 & 41 & 93 & 213 & 70 & 72 & 139 & 102 & 137 & 82 & 151 & 143 & 128 & 207 \\
56 & 141 & 108 & 58 & 91 & 50 & 54 & 60 & 87 & 165 & 57 & 102 & 146 & 149 & 116 & 211 \\
89 & 114 & 187 & 46 & 113 & 104 & 55 & 66 & 127 & 154 & 186 & 71 & 153 & 134 & 203 & 94 \\
35 & 99 & 150 & 66 & 34 & 88 & 88 & 127 & 140 & 141 & 175 & 212 & 144 & 128 & 213 & 100 \\
88 & 97 & 96 & 50 & 49 & 101 & 47 & 90 & 136 & 136 & 156 & 204 & 105 & 43 & 54 & 76 \\
43 & 104 & 69 & 69 & 68 & 53 & 110 & 127 & 134 & 145 & 158 & 183 & 109 & 121 & 72 & 113
\end{bmatrix}.$$

Figure 8.21 shows a three-level image decomposition using the Haar wavelet.

8.7 Wavelet Packets

Wavelet packets can be viewed as a generalization of wavelets. They were first introduced by Coifman, Meyer, Quake, and Wickerhauser [14] as a family of orthonormal bases for discrete functions of $\mathbf{R}^N$. A complete subband decomposition can be viewed as a decomposition of the input signal, using an analysis tree of depth $\log N$.

In the usual dyadic wavelet decomposition, only the low-pass-filtered subband is recursively decomposed and thus can be represented by a logarithmic tree structure. However, a wavelet packet decomposition allows the decomposition to be represented by any pruned subtree of the full tree topology. Therefore, this representation of the decomposition topology is isomorphic to all permissible subband topologies [15]. The leaf nodes of each pruned subtree represent one permissible orthonormal basis.

The wavelet packet decomposition offers a number of attractive properties, including

- Flexibility, since a best wavelet basis in the sense of some cost metric can be found within a large library of permissible bases.
- Favorable localization of wavelet packets in both frequency and space.
- Low computational requirement for wavelet packet decomposition, because each decomposition can be computed in the order of $N \log N$ using fast filter banks.

Wavelet packets are currently being applied to solve various practical problems such as image compression, signal de-noising, fingerprint identification, and so on.

8.8 Embedded Zerotree of Wavelet Coefficients

So far, we have described a wavelet-based scheme for image decomposition. However, aside from referring to the idea of quantizing away small coefficients, we have not really addressed how to code the wavelet transform values—how to form a bitstream. This problem is precisely what is dealt with in terms of a new data structure, the Embedded Zerotree.

The *Embedded Zerotree Wavelet* (EZW) algorithm introduced by Shapiro [16] is an effective and computationally efficient technique in image coding. This work has inspired a number of refinements to the initial EZW algorithm, the most notable being Said and Pearlman's *Set Partitioning in Hierarchical Trees* (SPIHT) algorithm [17] and Taubman's *Embedded Block Coding with Optimized Truncation* (EBCOT) algorithm [18], which is adopted into the JPEG2000 standard.

The EZW algorithm addresses two problems: obtaining the best image quality for a given bitrate and accomplishing this task in an embedded fashion. An *embedded* code is one that contains all lower rate codes "embedded" at the beginning of the bitstream. The bits are effectively ordered by importance in the bitstream. An embedded code allows the encoder to terminate the encoding at any point and thus meet any target bitrate exactly. Similarly, a decoder can cease to decode at any point and produce reconstructions corresponding to all lower rate encodings.

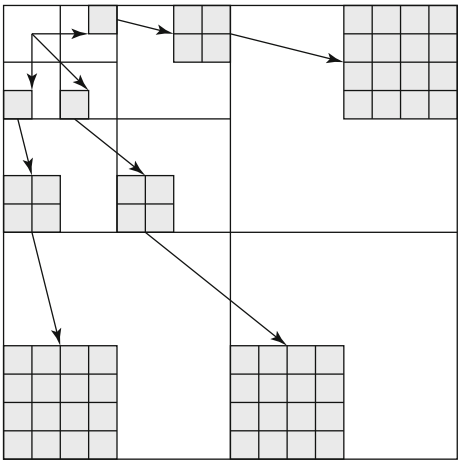

Fig. 8.22 Parent–child relationship in a zerotree

To achieve this goal, the EZW algorithm takes advantage of an important aspect of low-bitrate image coding. When conventional coding methods are used to achieve low bitrates, using scalar quantization followed by entropy coding, say, the most likely symbol, after quantization, is zero. It turns out that a large fraction of the bit budget is spent encoding the *significance map*, which flags whether input samples (in the case of the 2D discrete wavelet transform, the transform coefficients) have a zero or nonzero quantized value. The EZW algorithm exploits this observation to turn any significant improvement in encoding the significance map into a corresponding gain in compression efficiency. The EZW algorithm consists of two central components: the zerotree data structure and the method of successive approximation quantization.

8.8.1 The Zerotree Data Structure

The coding of the significance map is achieved using a new data structure called the *zerotree*. A wavelet coefficient x is said to be *insignificant* with respect to a given threshold T if $|x| < T$. The zerotree operates under the hypothesis that if a wavelet coefficient at a coarse scale is insignificant with respect to a given threshold T, all wavelet coefficients of the same orientation in the same spatial location at finer scales are likely to be insignificant with respect to T. Using the hierarchical wavelet decomposition presented in this chapter, we can relate every coefficient at a given scale to a set of coefficients at the next finer scale of similar orientation.

Figure 8.22 provides a pictorial representation of the zerotree on a three-stage wavelet decomposition. The coefficient at the coarse scale is called the *parent* while all corresponding coefficients are the next finer scale of the same spatial location and similar orientation are called *children*. For a given parent, the set of all coefficients at all finer scales are called *descendants*. Similarly, for a given child, the set of all coefficients at all coarser scales are called *ancestors*.

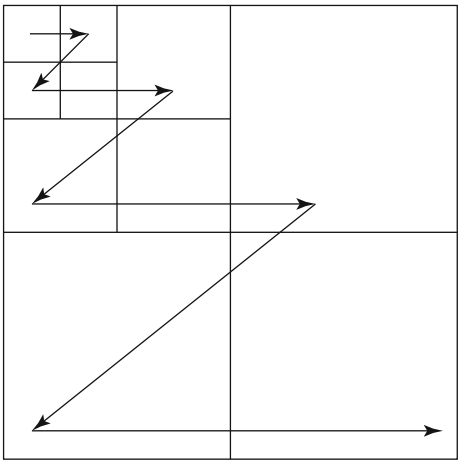

Fig. 8.23 EZW scanning order

The scanning of the coefficients is performed in such a way that no child node is scanned before its parent. Figure 8.23 depicts the scanning pattern for a three-level wavelet decomposition.

Given a threshold T, a coefficient x is an element of the zerotree if it is insignificant and all its descendants are insignificant as well. An element of a zerotree is a *zerotree root* if it is not the descendant of a previously found zerotree root. The significance map is coded using the zerotree with a four-symbol alphabet. The four symbols are

- **The zerotree root**. The root of the zerotree is encoded with a special symbol indicating that the insignificance of the coefficients at finer scales is completely predictable.
- **Isolated zero**. The coefficient is insignificant but has some significant descendants.
- **Positive significance**. The coefficient is significant with a positive value.
- **Negative significance**. The coefficient is significant with a negative value.

The cost of encoding the significance map is substantially reduced by employing the zerotree. The zerotree works by exploiting self-similarity on the transform coefficients. The underlying justification for the success of the zerotree is that even though the image has been transformed using a decorrelating transform, the occurrences of insignificant coefficients are not independent events.

In addition, the zerotree coding technique is based on the observation that it is much easier to predict insignificance than to predict significant details across scales. This technique focuses on reducing the cost of encoding the significance map so that more bits will be available to encode the expensive significant coefficients.

8.8.2 Successive Approximation Quantization

Embedded coding in the EZW coder is achieved using a method called *Successive Approximation Quantization* (SAQ). One motivation for developing this method is to

produce an embedded code that provides a coarse-to-fine, multiprecision logarithmic representation of the scale space corresponding to the wavelet-transformed image. Another motivation is to take further advantage of the efficient encoding of the significance map using the zerotree data structure, by allowing it to encode more significance maps.

The SAQ method sequentially applies a sequence of thresholds $T_0, \ldots, T_{N-1}$ to determine the significance of each coefficient. The thresholds are chosen such that $T_i = T_{i-1}/2$. The initial threshold T_0 is chosen so that $|x_j| < 2T_0$ for all transform coefficients x_j. A *dominant list* and a *subordinate list* are maintained during the encoding and decoding process. The dominant list contains the coordinates of the coefficients that have not yet been found to be significant in the same relative order as the initial scan.

Using the scan ordering shown in Fig. 8.23, all coefficients in a given subband appear on the initial dominant list prior to coefficients in the next subband. The subordinate list contains the magnitudes of the coefficients that have been found to be significant. Each list is scanned only once for each threshold.

During a dominant pass, coefficients having their coordinates on the dominant list implies that they are not yet significant. These coefficients are compared to the threshold T_i to determine their significance. If a coefficient is found to be significant, its magnitude is appended to the subordinate list, and the coefficient in the wavelet transform array is set to zero to enable the possibility of a zerotree occurring on future dominant passes at smaller thresholds. The resulting significance map is zerotree-coded.

The dominant pass is followed by a subordinate pass. All coefficients on the subordinate list are scanned, and their magnitude, as it is made available to the decoder, is refined to an additional bit of precision. Effectively, the width of the uncertainty interval for the true magnitude of the coefficients is cut in half. For each magnitude on the subordinate list, the refinement can be encoded using a binary alphabet with a 1 indicating that the true value falls in the upper half of the uncertainty interval and a 0 indicating that it falls in the lower half. The string of symbols from this binary alphabet is then entropy-coded. After the subordinate pass, the magnitudes on the subordinate list are sorted in decreasing order to the extent that the decoder can perform the same sort.

The process continues to alternate between the two passes, with the threshold halved before each dominant pass. The encoding stops when some target stopping criterion has been met.

8.8.3 EZW Example

The following example demonstrates the concept of zerotree coding and successive approximation quantization. Shapiro [16] presents an example of EZW coding in his paper for an 8×8 three-level wavelet transform. However, unlike the example given by Shapiro, we will complete the encoding and decoding process and show the output bitstream up to the point just before entropy coding.

57	–37	39	–20	3	7	9	10
–29	30	17	33	8	2	1	6
14	6	15	13	9	–4	2	3
10	19	–7	9	–7	14	12	–9
12	15	33	20	–2	3	1	0
0	7	2	4	4	–1	1	1
4	1	10	3	2	0	1	0
5	6	0	0	3	1	2	1

Fig. 8.24 Coefficients of a three-stage wavelet transform used as input to the EZW algorithm

Figure 8.24 shows the coefficients of a three-stage wavelet transform that we attempt to code using the EZW algorithm. We will use the symbols p, n, t, and z to denote positive significance, negative significance, zerotree root, and isolated zero respectively.

Since the largest coefficient is 57, we will choose the initial threshold T_0 to be 32. At the beginning, the dominant list contains the coordinates of all the coefficients. We begin scanning in the order shown in Fig. 8.23 and determine the significance of the coefficients. The following is the list of coefficients visited, in the order of the scan:

$$\{57, -37, -29, 30, 39, -20, 17, 33, 14, 6, 10, 19, 3, 7, 8, 2, 2, 3, 12, -9, 33, 20, 2, 4\}$$

With respect to the threshold $T_0 = 32$, it is easy to see that the coefficients 57 and -37 are significant. Thus, we output a p and an n to represent them. The coefficient -29 is insignificant but contains a significant descendant, 33, in LH1. Therefore, it is coded as z. The coefficient 30 is also insignificant, and all its descendants are insignificant with respect to the current threshold, so it is coded as t.

Since we have already determined the insignificance of 30 and all its descendants, the scan will bypass them, and no additional symbols will be generated. Continuing in this manner, the dominant pass outputs the following symbols:

$$D_0 : pnztpttptzttttttttttptttt$$

Five coefficients are found to be significant: 57, -37, 39, 33, and another 33. Since we know that no coefficients are greater than $2T_0 = 64$, and the threshold used in the first dominant pass is 32, the uncertainty interval is thus $[32, 64)$. Therefore, we know that the value of significant coefficients lie somewhere inside this uncertainty interval.

The subordinate pass following the dominant pass refines the magnitude of these coefficients by indicating whether they lie in the first half or the second half of the uncertainty interval. The output is 0 if the values lie in $[32, 48)$ and 1 for values

within [48, 64). According to the order of the scan, the subordinate pass outputs the following bits:

$$S_0 : 10000$$

Now the dominant list contains the coordinates of all the coefficients except those found to be significant, and the subordinate list contains the values {57, 37, 39, 33, 33}. After the subordinate pass is completed, we attempt to rearrange the values in the subordinate list such that larger coefficients appear before smaller ones, with the constraint that the decoder is able do exactly the same.

Since the subordinate pass halves the uncertainty interval, the decoder is able to distinguish values from [32, 48) and [48, 64). Since 39 and 37 are not distinguishable in the decoder, their order will not be changed. Therefore, the subordinate list remains the same after the reordering operation.

Before we move on to the second round of dominant and subordinate passes, we need to set the values of the significant coefficients to 0 in the wavelet transform array so that they do not prevent the emergence of a new zerotree.

The new threshold for a second dominant pass is $T_1 = 16$. Using the same procedure as above, the dominant pass outputs the following symbols. Note that the coefficients in the dominant list will not be scanned.

$$D_1 : zznptnpttztpttttttttttttptttttt \tag{8.71}$$

The subordinate list is now {57, 37, 39, 33, 33, 29, 30, 20, 17, 19, 20}. The subordinate pass that follows will halve each of the three current uncertainty intervals [48, 64), [32, 48), and [16, 32). The subordinate pass outputs the following bits:

$$S_1 : 10000110000$$

Now we set the value of the coefficients found to be significant to 0 in the wavelet transform array.

The output of the subsequent dominant and subordinate passes is shown below:

$$\begin{aligned}
D_2 &: zzzzzzzzptpzpptnttptpptttpttpttpnpptttttttptttttttttttttt \\
S_2 &: 01100111001101100000110110 \\
D_3 &: zzzzzzztzpztztnttptttttptnnttttpttpppttpttttt \\
S_3 &: 00100010001110100110001001111101100010 \\
D_4 &: zzzzzttztztzztzzpttpppttttpttpttnpttptptttpt \\
S_4 &: 11111010011010110000010111011011000100100101010 \\
D_5 &: zzzztztttztzzzzttpttpttttnptpptttpptp
\end{aligned}$$

Since the length of the uncertainty interval in the last pass is 1, the last subordinate pass is unnecessary.

On the decoder side, suppose we received information only from the first dominant and subordinate passes. We can reconstruct a lossy version of the transform coefficients by reversing the encoding process. From the symbols in D_0 we can obtain the position of the significant coefficients. Then, using the bits decoded from S_0, we can

56	–40	40	0	0	0	0	0
0	0	0	40	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	40	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

Fig. 8.25 Reconstructed transform coefficients from the first dominant and subordinate passes

58	–38	38	–22	0	0	12	12
–30	30	18	34	12	0	0	0
12	0	12	12	12	0	0	0
12	20	0	12	0	12	12	–12
12	12	34	22	0	0	0	0
0	0	0	0	0	0	0	0
0	0	12	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 8.26 Reconstructed transform coefficients from D_0, S_0, D_1, S_1, D_2, and the first 10 bits of S_2

reconstruct the value of these coefficients using the center of the uncertainty interval. Figure 8.25 shows the resulting reconstruction.

It is evident that we can stop the decoding process at any point to reconstruct a coarser representation of the original input coefficients. Figure 8.26 shows the reconstruction if the decoder received only D_0, S_0, D_1, S_1, D_2, and only the first 10 bits of S_2. The coefficients that were not refined during the last subordinate pass appear as if they were quantized using a coarser quantizer than those that were.

In fact, the reconstruction value used for these coefficients is the center of the uncertainty interval from the previous pass. The heavily shaded coefficients in the figure are those that were refined, while the lightly shaded coefficients are those that were not refined. As a result, it is not easy to see where the decoding process ended, and this eliminates much of the visual artifact contained in the reconstruction.

8.9 Set Partitioning in Hierarchical Trees (SPIHT)

SPIHT is a revolutionary extension of the EZW algorithm. Based on EZW's underlying principles of partial ordering of transformed coefficients, ordered bitplane transmission of refinement bits, and the exploitation of self-similarity in the transformed wavelet image, the SPIHT algorithm significantly improves the performance of its predecessor by changing the ways subsets of coefficients are partitioned and refinement information is conveyed.

A unique property of the SPIHT bitstream is its compactness. The resulting bitstream from the SPIHT algorithm is so compact that passing it through an entropy coder would produce only marginal gain in compression at the expense of much more computation. Therefore, a fast SPIHT coder can be implemented without any entropy coder or possibly just a simple patent-free Huffman coder.

Another signature of the SPIHT algorithm is that no ordering information is explicitly transmitted to the decoder. Instead, the decoder reproduces the execution path of the encoder and recovers the ordering information. A desirable side effect of this is that the encoder and decoder have similar execution times, which is rarely the case for other coding methods. Said and Pearlman [17] gives a full description of this algorithm.

8.10 Exercises

1. Assume we have an unbounded source we wish to quantize using an M-bit midtread uniform quantizer. Derive an expression for the total distortion if the step size is 1.
2. Suppose the domain of a uniform quantizer is $[-b_M, b_M]$. We define the loading fraction as
$$\gamma = \frac{b_M}{\sigma}$$
where σ is the standard deviation of the source. Write a simple program to quantize a Gaussian distributed source having zero mean and unit variance using a 4-bit uniform quantizer. Plot the SNR against the loading fraction and estimate the optimal step size that incurs the least amount of distortion from the graph.
3. Suppose the input source is Gaussian distributed with zero mean and unit variance—that is, the probability density function is defined as
$$f_X(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}. \tag{8.72}$$
We wish to find a four-level Lloyd–Max quantizer. Let $\mathbf{y}_i = [y_i^0, \ldots, y_i^3]$ and $\mathbf{b}_i = [b_i^0, \ldots, b_i^3]$. The initial reconstruction levels are set to $\mathbf{y}_0 = [-2, -1, 1, 2]$. This source is unbounded, so the outer two boundaries are $+\infty$ and $-\infty$.
Follow the Lloyd–Max algorithm in this chapter: the other boundary values

are calculated as the midpoints of the reconstruction values. We now have $\mathbf{b}_0 = [-\infty, -1.5, 0, 1.5, \infty]$. Continue one more iteration for $i = 1$, using Eq. (8.13) and find y_0^1, y_1^1, y_2^1, y_3^1, using numerical integration. Also calculate the squared error of the difference between $\mathbf{y}_1$ and $\mathbf{y}_0$.
Iteration is repeated until the squared error between successive estimates of the reconstruction levels are below some predefined threshold ϵ. Write a small program to implement the Lloyd–Max quantizer described above.

4. If the block size for a 2D DCT transform is 8×8, and we use only the DC components to create a thumbnail image, what fraction of the original pixels would we be using?
5. When the block size is 8, the definition of the DCT is given in Eq. (8.17).
   (a) If an 8×8 grayscale image is in the range 0 .. 255, what is the largest value a DCT coefficient could be, and for what input image? (Also, state *all* the DCT coefficient values for that image.)
   (b) If we first subtract the value 128 from the whole image and then carry out the DCT, what is the exact effect on the DCT value $F[2, 3]$?
   (c) Why would we carry out that subtraction? Does the subtraction affect the number of bits we need to code the image?
   (d) Would it be possible to invert that subtraction, in the IDCT? If so, how?
6. Write a simple program or refer to the sample DCT program `dct_1D.c` in the book's web site to verify the results in Example 8.2 of the 1D DCT example in this chapter.
7. Write a program to verify that the DCT-matrix $\mathbf{T_8}$ as defined in Eqs. 8.29 and 8.30 is an Orthogonal Matrix, i.e., all its rows and columns are orthogonal unit vectors (orthonormal vectors).
8. Write a program to verify that the 2D DCT and IDCT matrix implementations as defined in Eqs. 8.27 and 8.31 are lossless, i.e., they can transform any 8×8 values $f(i, j)$ to $F(u, v)$ and back to $f(i.j)$. (Here, we are not concerned with possible/tiny floating point calculation errors.)
9. We could use a similar DCT scheme for *video streams* by using a 3D version of DCT. Suppose one color component of a video has pixels f_{ijk} at position (i, j) and time k. How could we define its 3D DCT transform?
10. Suppose a uniformly colored sphere is illuminated and has shading varying smoothly across its surface, as in Fig. 8.27.
    (a) What would you expect the DCT coefficients for its image to look like?
    (b) What would be the effect on the DCT coefficients of having a checkerboard of colors on the surface of the sphere?
    (c) For the uniformly colored sphere again, describe the DCT values for a block that straddles the top edge of the sphere, where it meets the black background.
    (d) Describe the DCT values for a block that straddles the left edge of the sphere.
11. The Haar wavelet has a scaling function which is defined as follows:

$$\phi(t) = \begin{cases} 1 & 0 \leq t \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{8.73}$$

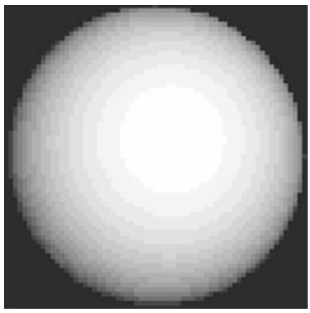

Fig. 8.27 Sphere shaded by a light

and its scaling vector is $h_0[0] = h_0[1] = 1/\sqrt{2}$.

(a) Draw the scaling function, then verify that its dilated translates $\phi(2t)$ and $\phi(2t-1)$ satisfy the dilation equation (8.62). Draw the combination of these functions that makes up the full function $\phi(t)$.

(b) Derive the wavelet vector $h_1[0]$, $h_1[1]$ from Eq. (8.65) and then derive and draw the Haar wavelet function $\psi(t)$ from Eq. (8.63).

12. Suppose the mother wavelet $\psi(t)$ has vanishing moments M_p up to and including M_n. Expand $f(t)$ in a Taylor series around $t = 0$, up to the nth derivative of f [i.e., up to leftover error of order $O(n+1)$]. Evaluate the summation of integrals produced by substituting the Taylor series into (8.58) and show that the result is of order $O(s^{n+2})$.
13. The program `wavelet_compression.c` on this book's web site is in fact simple to implement as a MATLAB function (or similar fourth-generation language). The advantage in doing so is that the `imread` function can input image formats of a great many types, and `imwrite` can output as desired. Using the given program as a template, construct a MATLAB program for wavelet-based image reduction, with perhaps the number of wavelet levels being a function parameter.
14. It is interesting to find the Fourier transform of functions, and this is easy if you have available a symbolic manipulation system such as MAPLE. In that language, you can just invoke the `fourier` function and view the answer directly! As an example, try the following code fragment:

```
with('inttrans');
f := 1;
F := fourier(f,t,w);
```

The answer should be $2\pi\delta(w)$. Let's try a Gaussian:

```
f := exp(-t^2);
F := fourier(f,t,w);
```

Now the answer should be $\sqrt{\pi}e^{(-w^2/4)}$: the Fourier transform of a Gaussian is simply another Gaussian.

15. Suppose we define the wavelet function

$$\psi(t) \;=\; exp(-t^{1/4})\sin(t^4)\;, \quad t \geq 0 \tag{8.74}$$

This function oscillates about the value 0. Use a plotting package to convince yourself that the function has a zero moment M_p for any value of p.

16. Implement both a DCT-based and a wavelet-based image coder. Design your user interface so that the compressed results from both coders can be seen side by side for visual comparison. The PSNR for each coded image should also be shown, for quantitative comparisons.
    Include a slider bar that controls the target bitrate for both coders. As you change the target bitrate, each coder should compress the input image in real time and show the compressed results immediately on your user interface.
    Discuss both qualitative and quantitative compression results observed from your program at target bitrates of 4, 1, and 0.25 bpp.

References

1. K. Sayood, *Introduction to Data Compression*, 4th edn. (Morgan Kaufmann, San Francisco, 2012)
2. H. Stark, J.W. Woods, *Probability and Random Processes with Application to Signal Processing,* 3rd edn. (Prentice Hall, Upper Saddle River, 2002)
3. A. György. On the theoretical limits of lossy source coding, 1998. Tudományos Diákkör (TDK) Conf. (Hungarian Scientific Student's Conf.) at Technical University of Budapest
4. S. Arimoto, An algorithm for calculating the capacity of an arbitrary discrete memoryless channel. IEEE Trans. Inform. Theory **18**, 14–20 (1972)
5. R. Blahut, Computation of channel capacity and rate-distortion functions. IEEE Trans. Inform. Theory **18**, 460–473 (1972)
6. A. Gersho, R.M. Gray, *Vector Quantization and Signal Compression.* (Springer, Boston, 1991)
7. A.K. Jain, *Fundamentals of Digital Image Processing* (Prentice-Hall, Englewood Cliffs, 1988)
8. K.R. Rao, P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications* (Academic Press, Boston, 1990)
9. J.F. Blinn, What's the deal with the DCT? IEEE Comput. Graphics Appl. **13**(4), 78–83 (1993)
10. S. Mallat, *A Wavelet Tour of Signal Processing,* 3rd edn. (Academic Press, San Diego, 2008)
11. S. Mallat, A theory for multiresolution signal decomposition: the wavelet representation. IEEE Trans. Pattern Anal. Mach. Intell. **11**, 674–693 (1989)
12. R.C. Gonzalez, R.E. Woods, *Digital Image Processing,* 3rd edn. (Prentice-Hall, Upper Saddle River, 2007)
13. B.E. Usevitch, A tutorial on modern lossy wavelet image compression: foundations of JPEG 2000. IEEE Signal Process. Mag. **18**(5), 22–35 (2001)
14. R. Coifman, Y. Meyer, S. Quake, V. Wickerhauser, *Signal Processing and Compression with Wavelet packets.* (Yale University, Numerical Algorithms Research Group, 1990)
15. K. Ramachandran, M. Vetterli, Best wavelet packet basis in a rate-distortion sense. IEEE Trans. Image Processing **2**, 160–173 (1993)
16. J. Shapiro, Embedded image coding using zerotrees of wavelet coefficients. IEEE Trans. Signal Processing, 41(12), 3445–3462 (1993)
17. A. Said, W.A. Pearlman, A new, fast, and efficient image codec based on set partitioning in hierarchical trees. IEEE Trans. CSVT **6**(3), 243–249 (1996)
18. D. Taubman, High performance scalable image compression with EBCOT. IEEE Trans. Image Processing **9**(7), 1158–1170 (2000)