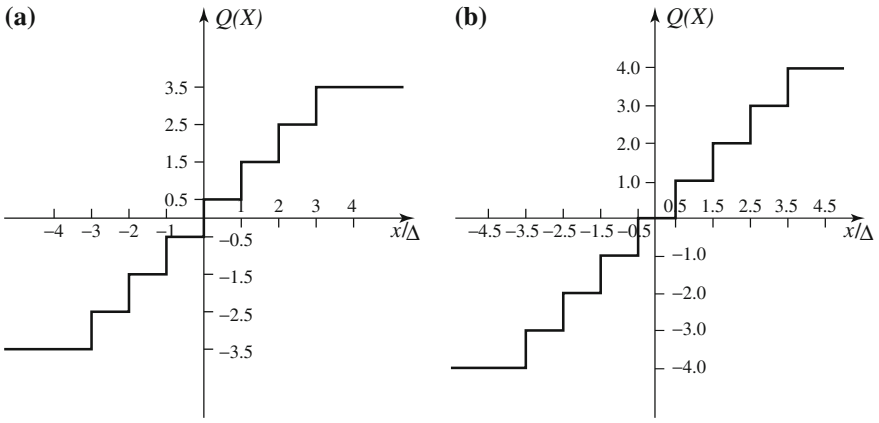

Fig. 8.2 Uniform scalar quantizers: **a** midrise; **b** midtread

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

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

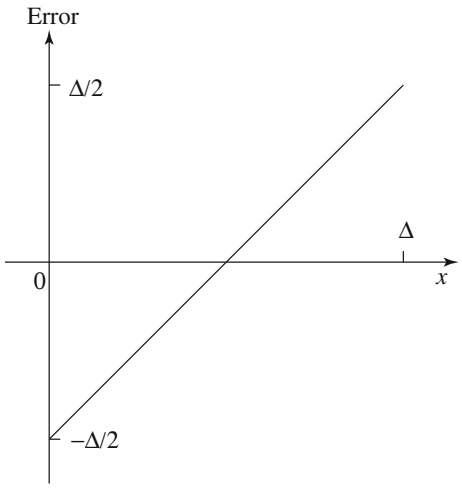

Fig. 8.3 Quantization error of a uniformly distributed source

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
```

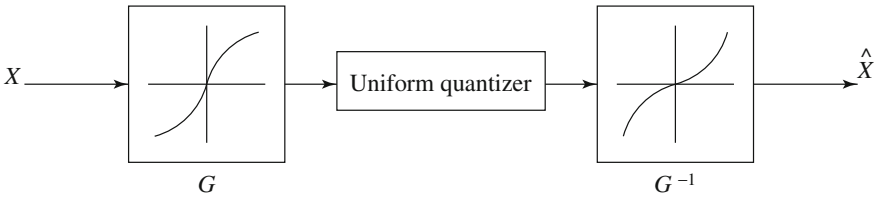

Fig. 8.4 Companded quantization

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

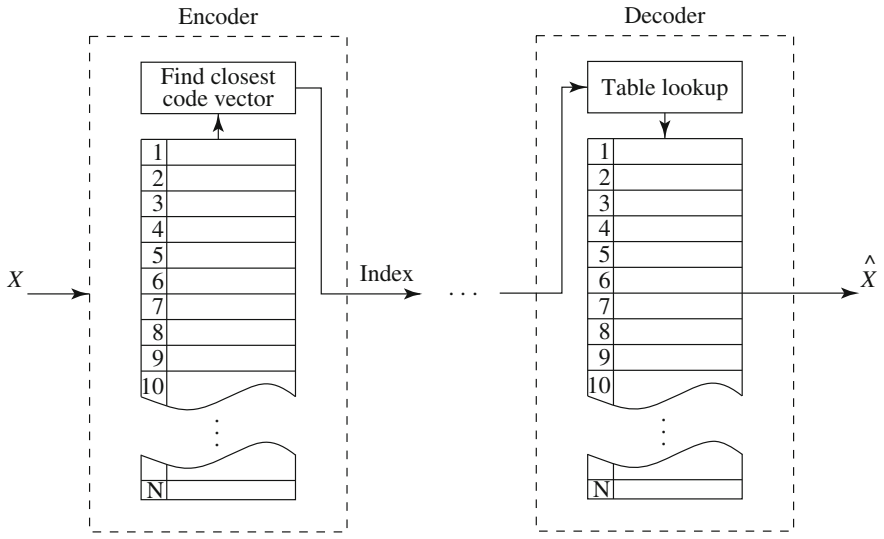

Fig. 8.5 Basic vector quantization procedure

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.

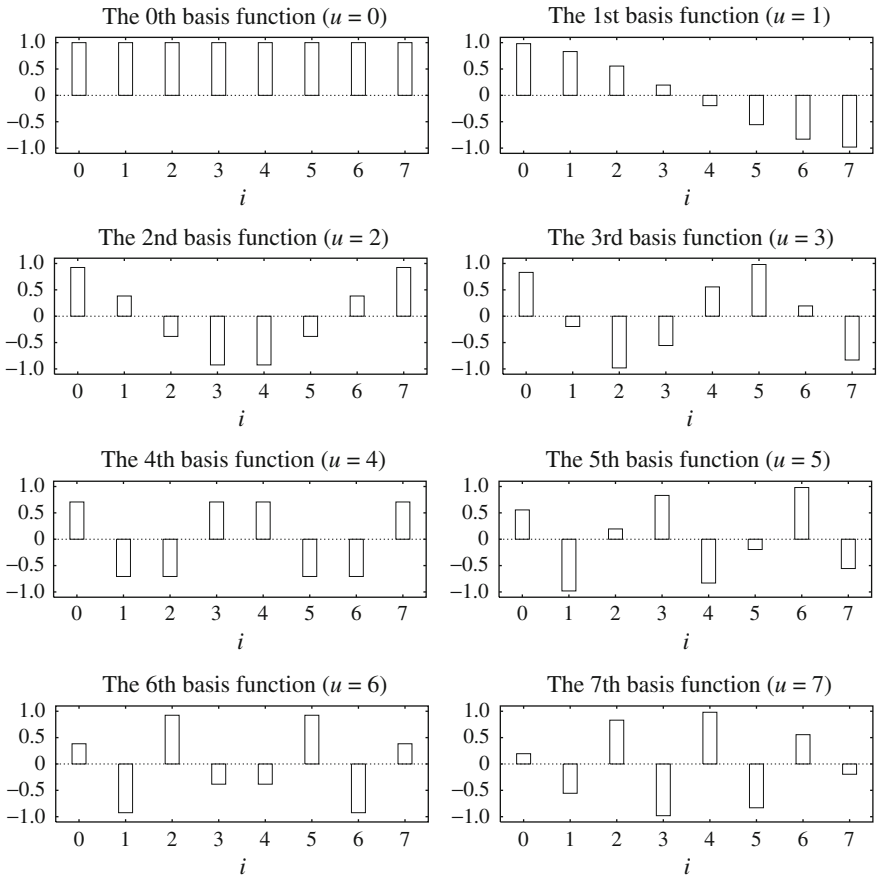

Fig. 8.6 The 1D DCT basis functions

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

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

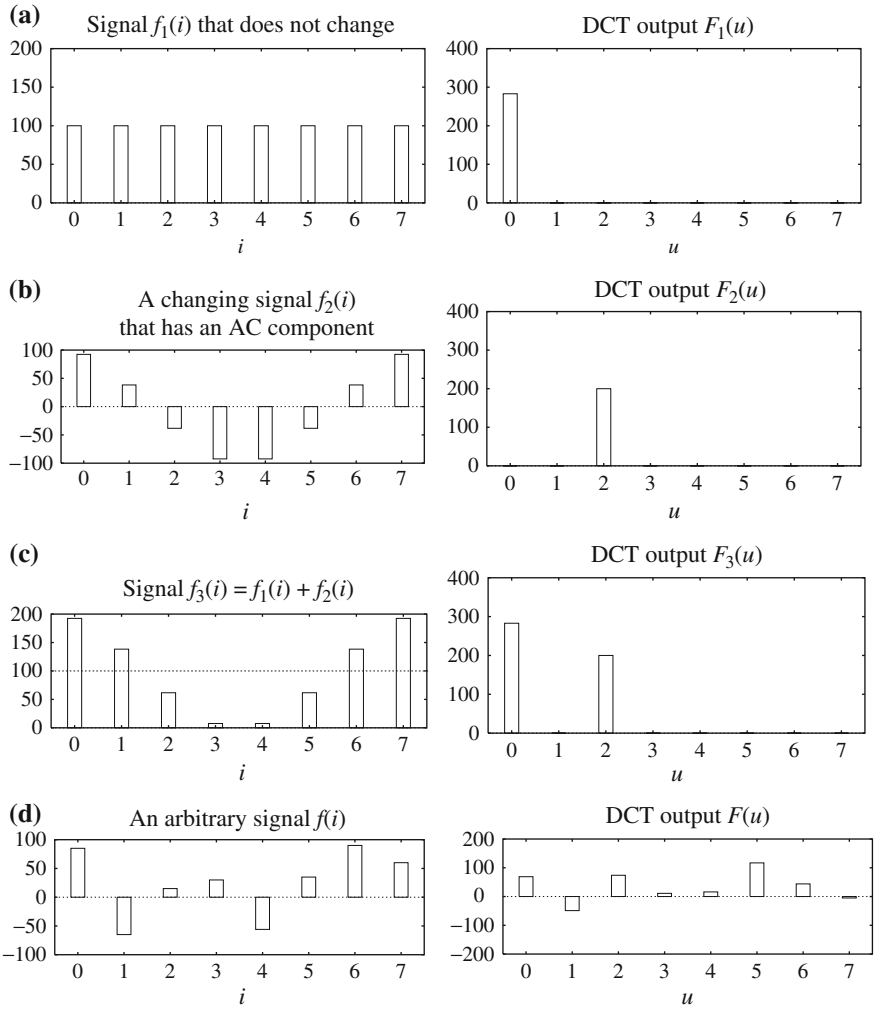

Fig. 8.7 Examples of 1D Discrete Cosine Transform: **a** a DC signal $f_1(i)$; **b** an AC signal $f_2(i)$; **c** $f_3(i) = f_1(i) + f_2(i)$; and **d** an arbitrary signal $f(i)$

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

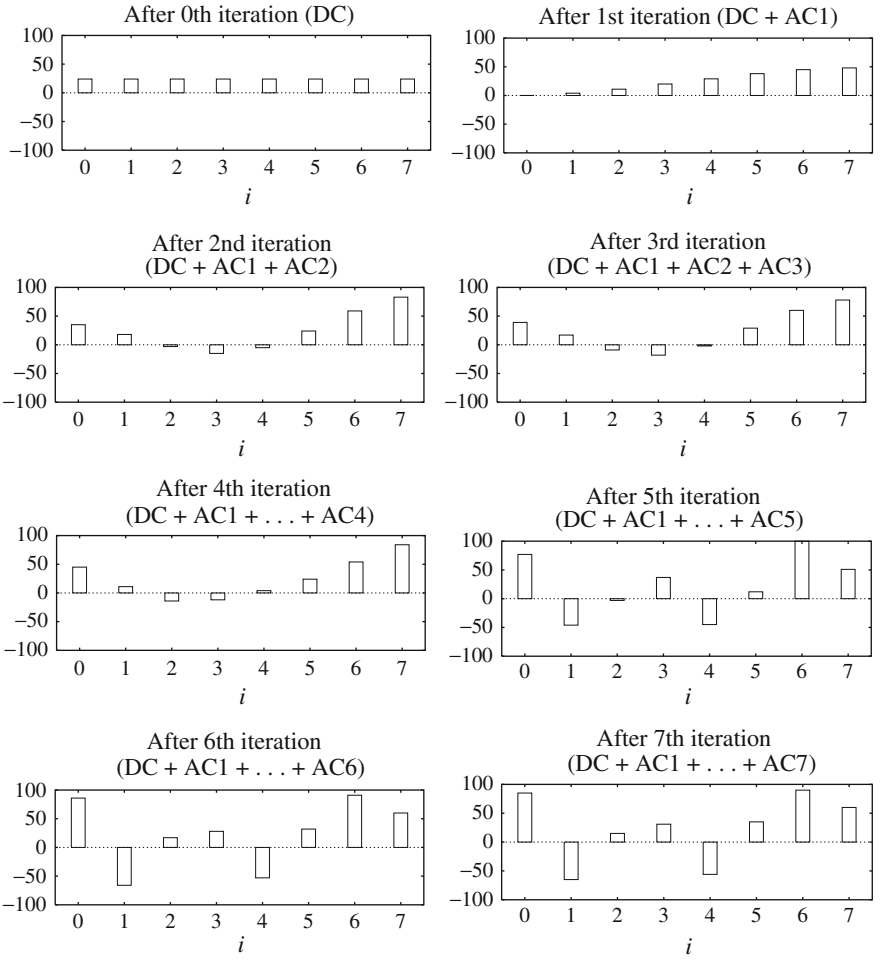

Fig. 8.8 An example of 1D IDCT

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

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:

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

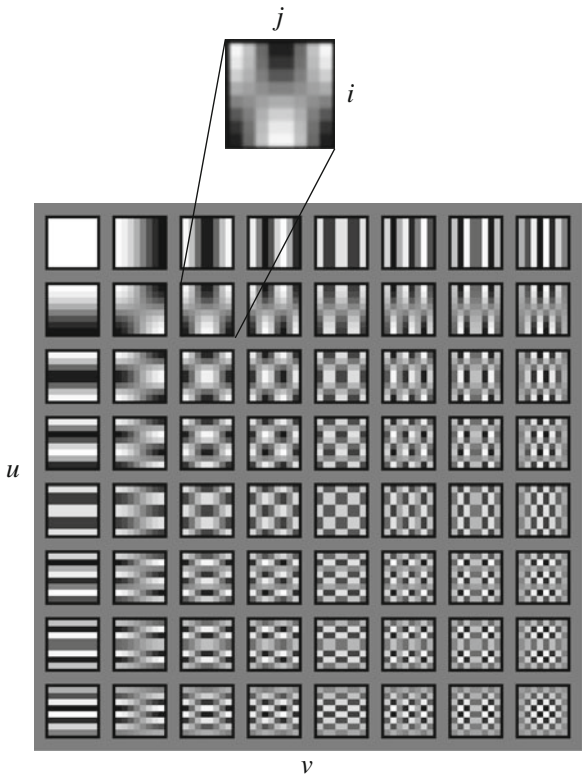

Fig. 8.9 Graphical illustration of 8×8 2D DCT basis

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

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

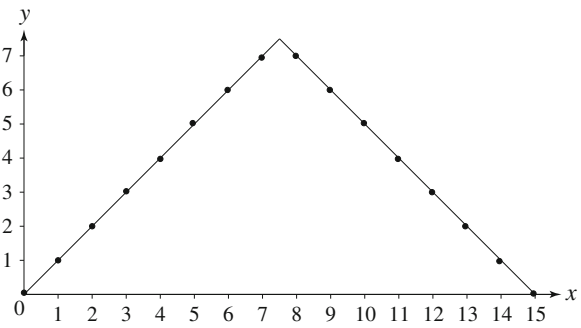

Fig. 8.10 Symmetric extension of the ramp function

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.

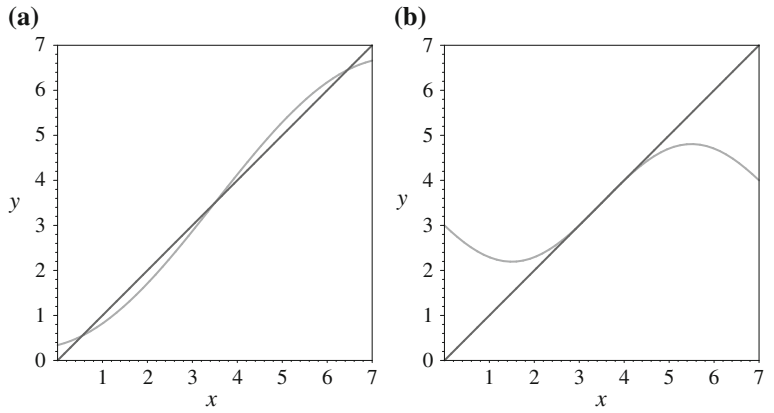

Fig. 8.11 Approximation of the ramp function: **a** three-term DCT approximation; **b** three-term DFT approximation

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.