Chapter 8
Lossy Compression Algorithms

8.1 Introduction

- Lossless compression algorithms do not deliver *compression ratios* that are high enough. Hence, most multimedia compression algorithms are *lossy*.

- What is *lossy compression*?
  - The compressed data is not the same as the original data, but a close approximation of it.
  - Yields a much higher compression ratio than that of lossless compression.

8.2 Distortion Measures

- The three most commonly used distortion measures in image compression are:
  - *Mean Squared Error* (MSE) σ_d^2 ,

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

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

  - *Signal to Noise Ratio* (SNR), in decibel units (dB),

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

where σ_x^2 is the average square value of the original data sequence and σ_d^2 is the MSE.

  - *Peak Signal to Noise Ratio* (PSNR),

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

8.3 The Rate-Distortion Theory

- Provides a framework for the study of tradeoffs between Rate and Distortion.

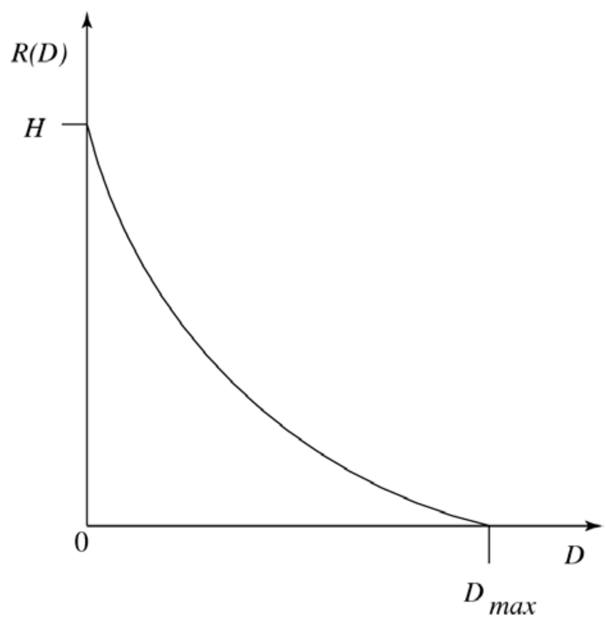


Fig. 8.1: Typical Rate Distortion Function.

8.4 Quantization

- Reduce the number of distinct output values to a much smaller set.

- Main source of the “loss” in lossy compression.

- Three different forms of quantization.
  - Uniform: midrise and midtread quantizers.
  - Nonuniform: companded quantizer.
  - Vector Quantization.

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 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.

- Two types of uniform scalar quantizers:
  - Midrise quantizers have even number of output levels.
  - Midtread quantizers have odd number of output levels, including zero as one of them (see Fig. 8.2).

- For the special case where Δ = 1, we can simply compute the output values for these quantizers as:

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

$$Q_{midtread}(x) = \lfloor x + 0.5 \rfloor \qquad (8.5)$$

- 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:

$$R = \lceil \log_2 M \rceil \qquad (8.6)$$

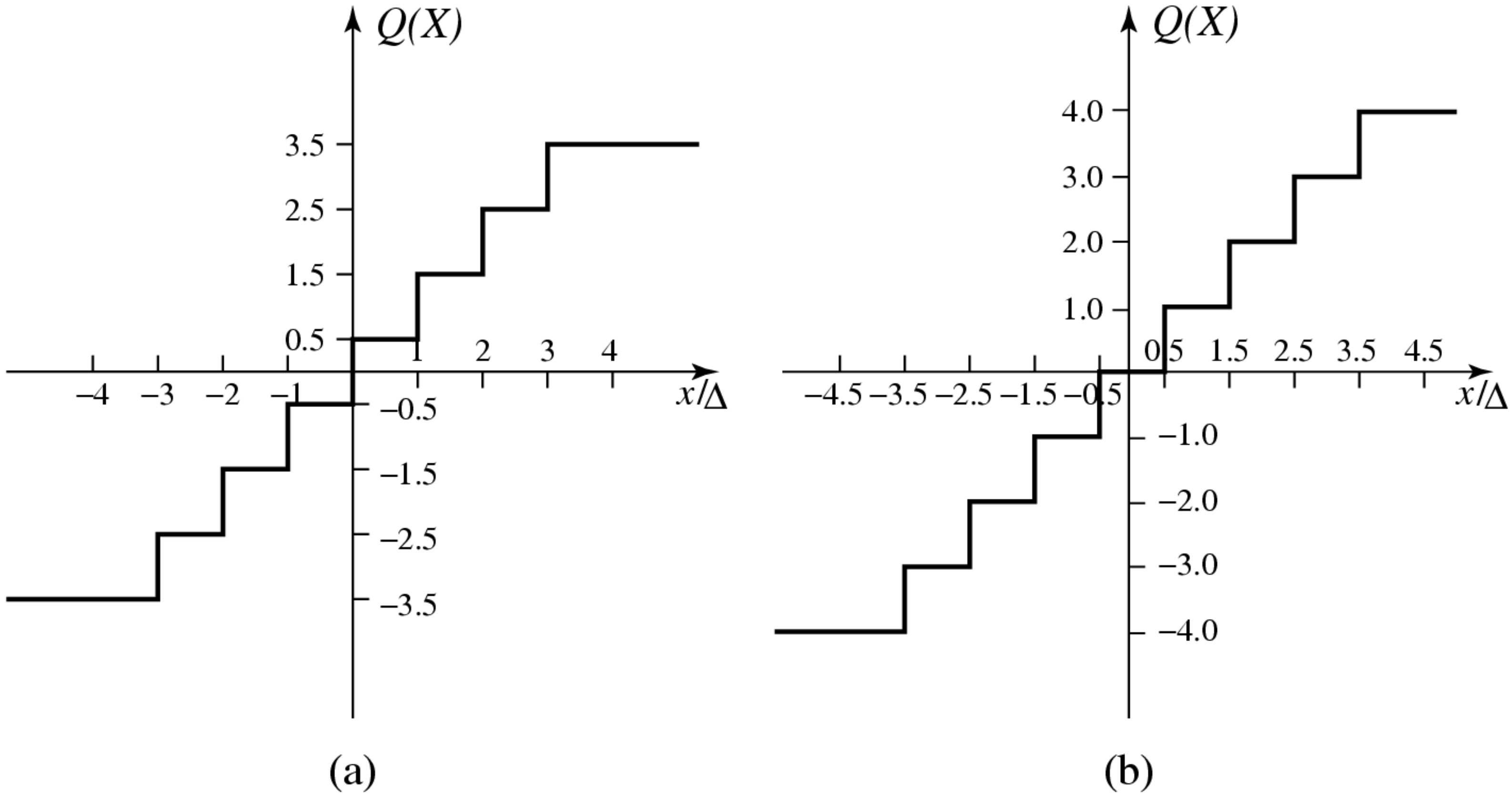


Fig. 8.2: Uniform Scalar Quantizers: (a) Midrise, (b) Midtread.

Quantization Error of Uniformly Distributed Source

- **Granular distortion**: quantization error caused by the quantize for bounded input.

  - 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 the positive data. The total distortion is twice the sum over the positive data, or

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

- Since the reconstruction values y_i are the midpoints of each interval, the quantization error must lie within the values $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$. For a uniformly distributed source, the graph of the quantization error is shown in Fig. 8.3.

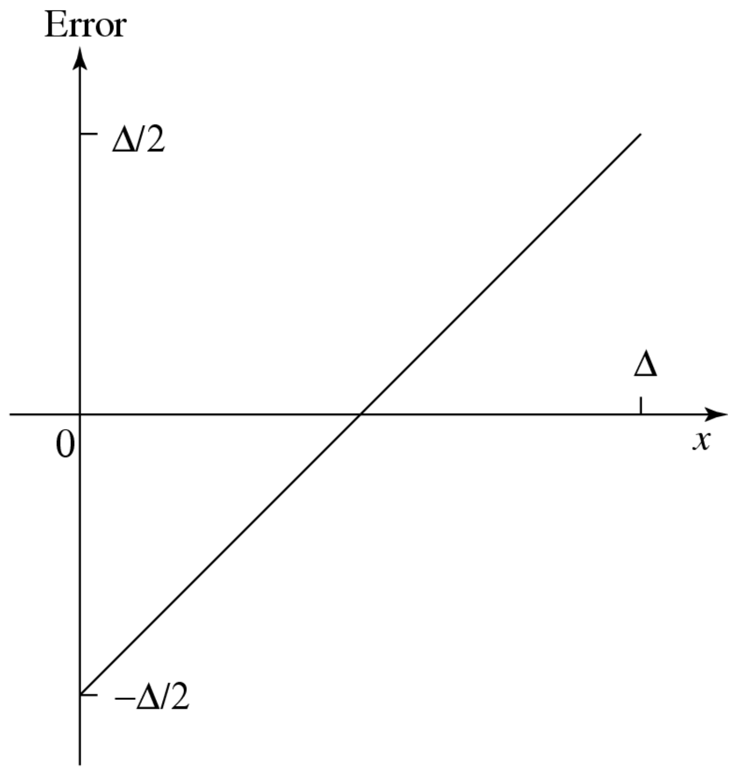


Fig. 8.3: Quantization error of a uniformly distributed source.

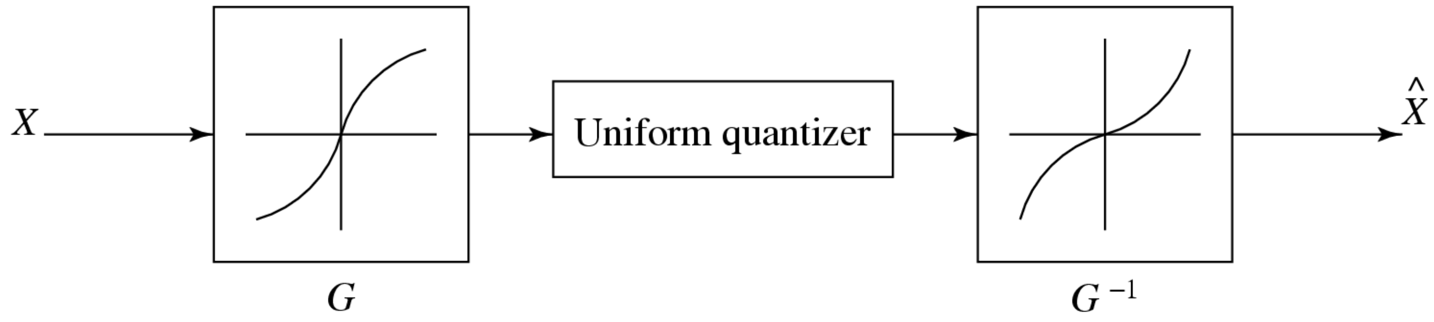


Fig. 8.4: Companded quantization.

- *Companded quantization* is **nonlinear**.
- As shown above, a *compander* consists of a *compressor function* G, a uniform quantizer, and an *expander function* G^{-1}.
- The two commonly used companders are the μ-law and A-law companders.

8.4.3 Vector Quantization (VQ)

- According to Shannon's original work on information theory, any compression system performs better if it operates on vectors or groups of samples rather than individual symbols or samples.
- Form vectors of input samples by simply concatenating a number of consecutive samples into a single vector.
- Instead of single reconstruction values as in scalar quantization, in VQ code *vectors* with n components are used. A collection of these code vectors form the *codebook*.

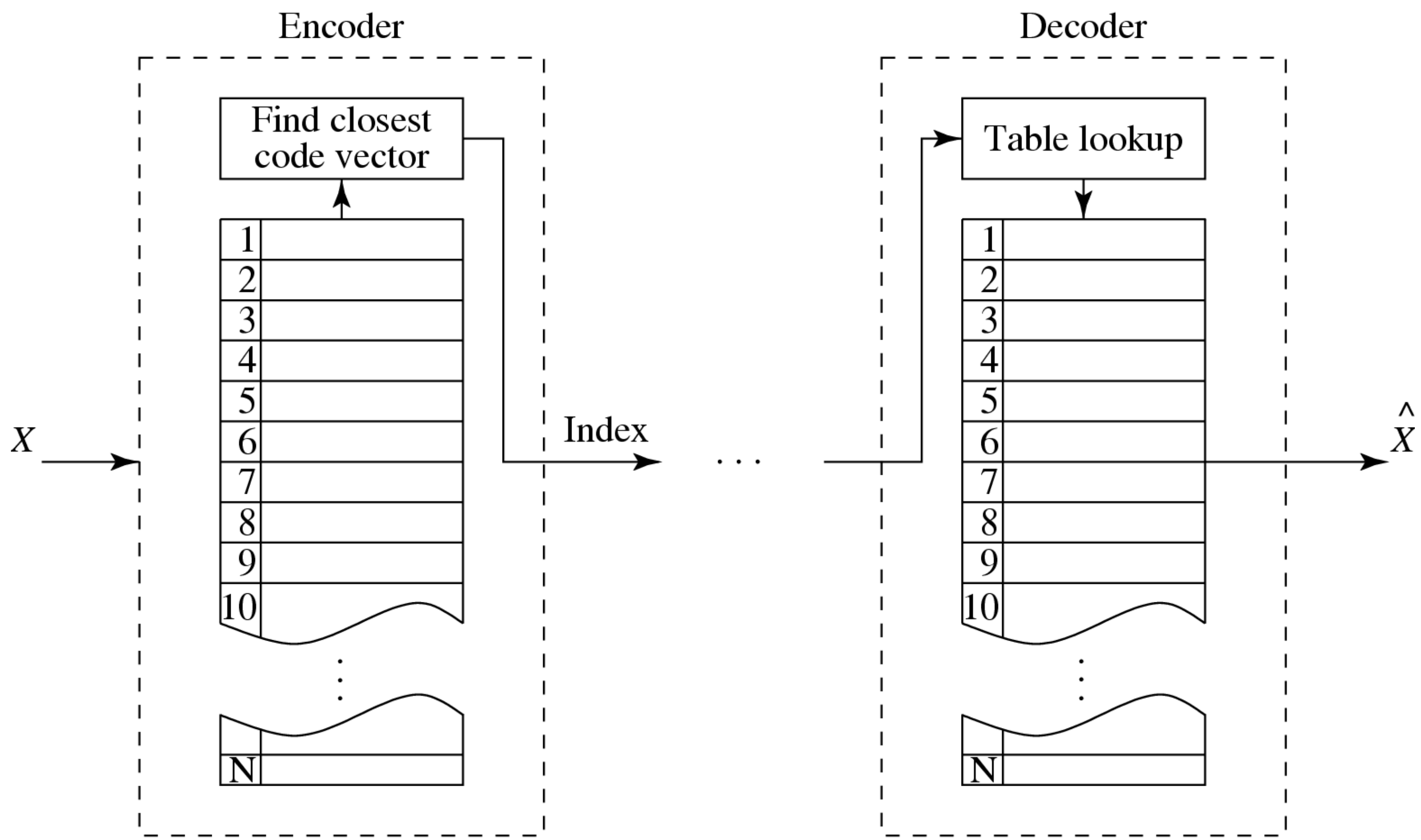


Fig. 8.5: Basic vector quantization procedure.

8.5 Transform Coding

- **The rationale behind transform coding**:

- If $\mathbf{Y}$ is the result of a linear transform $\mathbf{T}$ of the input vector $\mathbf{X}$ in such a way that the components of $\mathbf{Y}$ are much less correlated, then $\mathbf{Y}$ can be coded more efficiently than $\mathbf{X}$.

- If most information is accurately described by the first few components of a transformed vector, then the remaining components can be coarsely quantized, or even set to zero, with little signal distortion.

- Discrete Cosine Transform (DCT) will be studied first. In addition, we will examine the Karhunen-Loève Transform (KLT) which *optimally* decorrelates the components of the input $\mathbf{X}$.

8.5.1 Spatial Frequency and DCT

- *Spatial frequency* indicates how many times pixel values change across an image block.
- The DCT formalizes this notion with a measure of how much the image contents change in correspondence to the number of cycles of a cosine wave per block.
- 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* (re-compose) the signal.

Definition of DCT:

- Given an input 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:

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

where i, u = 0, 1, … , M – 1; j, v = 0, 1, … , N – 1; and the constants $C(u)$ and $C(v)$ are determined by

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

2D Discrete Cosine Transform (2D DCT):

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

where i, j, u, v = 0, 1, . . . , 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:

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

where i, j, u, v = 0, 1, . . . , 7.

1D Discrete Cosine Transform (1D DCT):

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

where i = 0, 1, . . . , 7, u = 0, 1, . . . , 7.

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

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

where i = 0, 1, . . . , 7, u = 0, 1, . . . , 7.

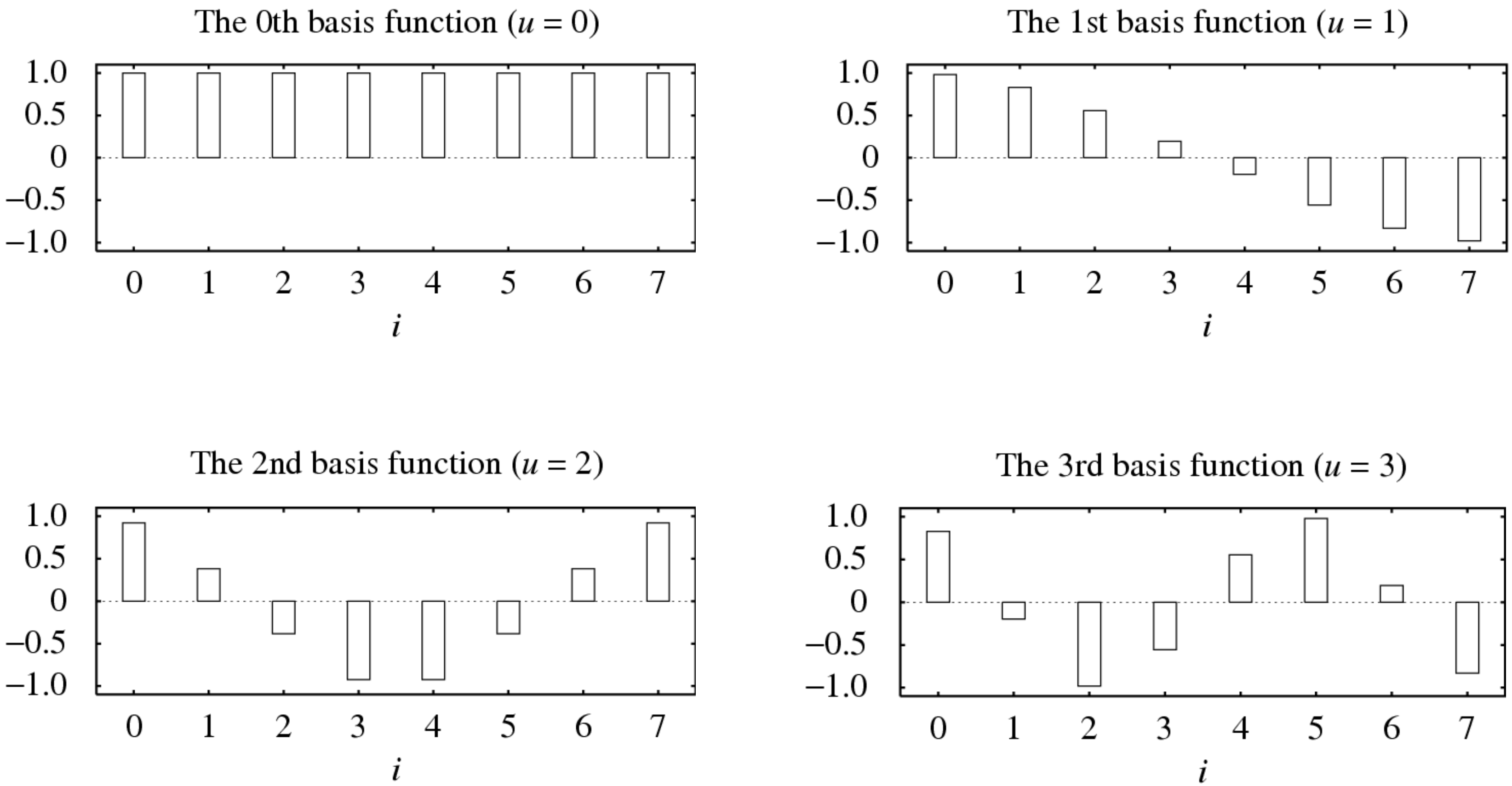


Fig. 8.6: The 1D DCT basis functions.

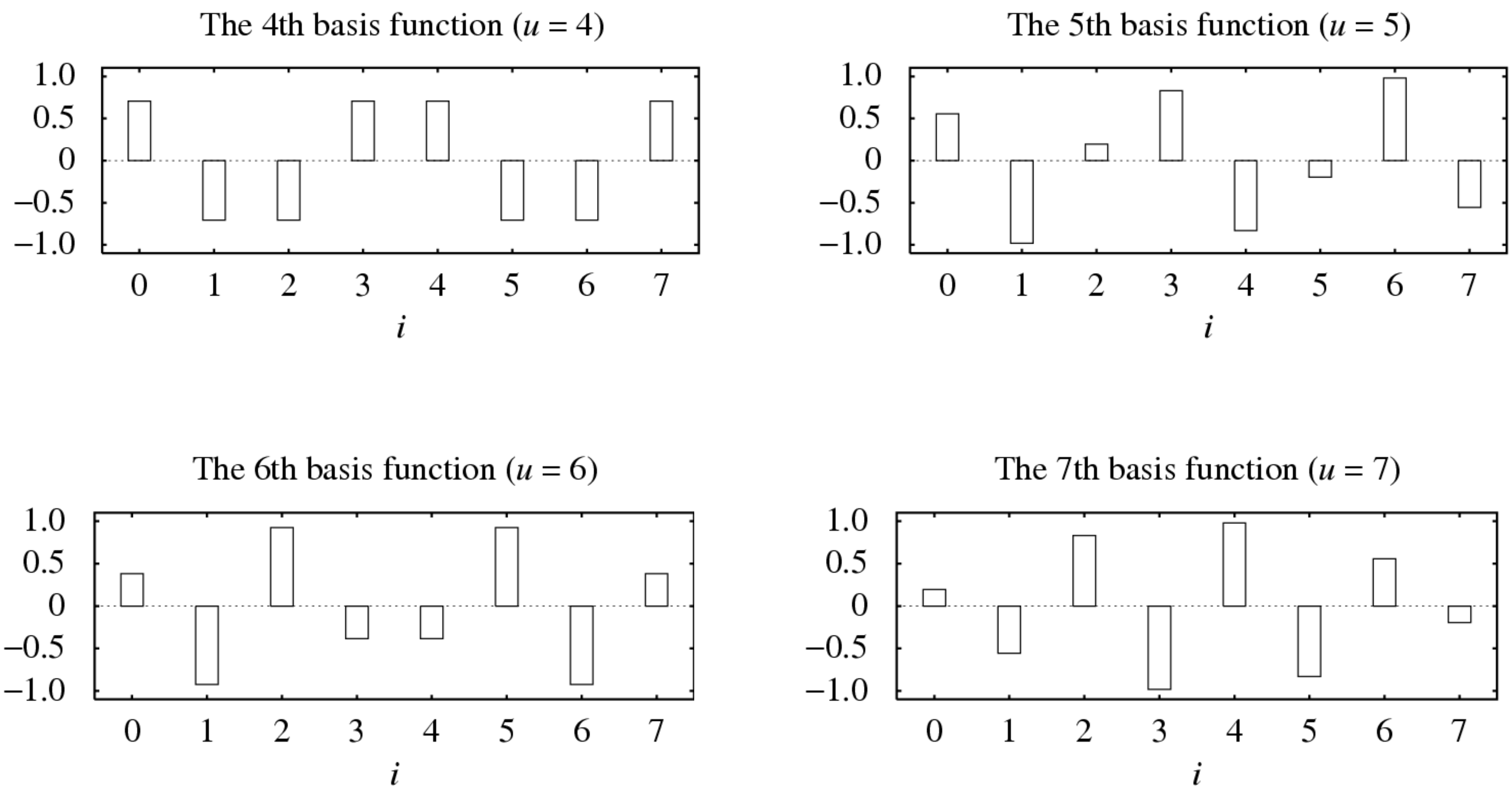


Fig. 8.6 (Cont'd): The 1D DCT basis functions.

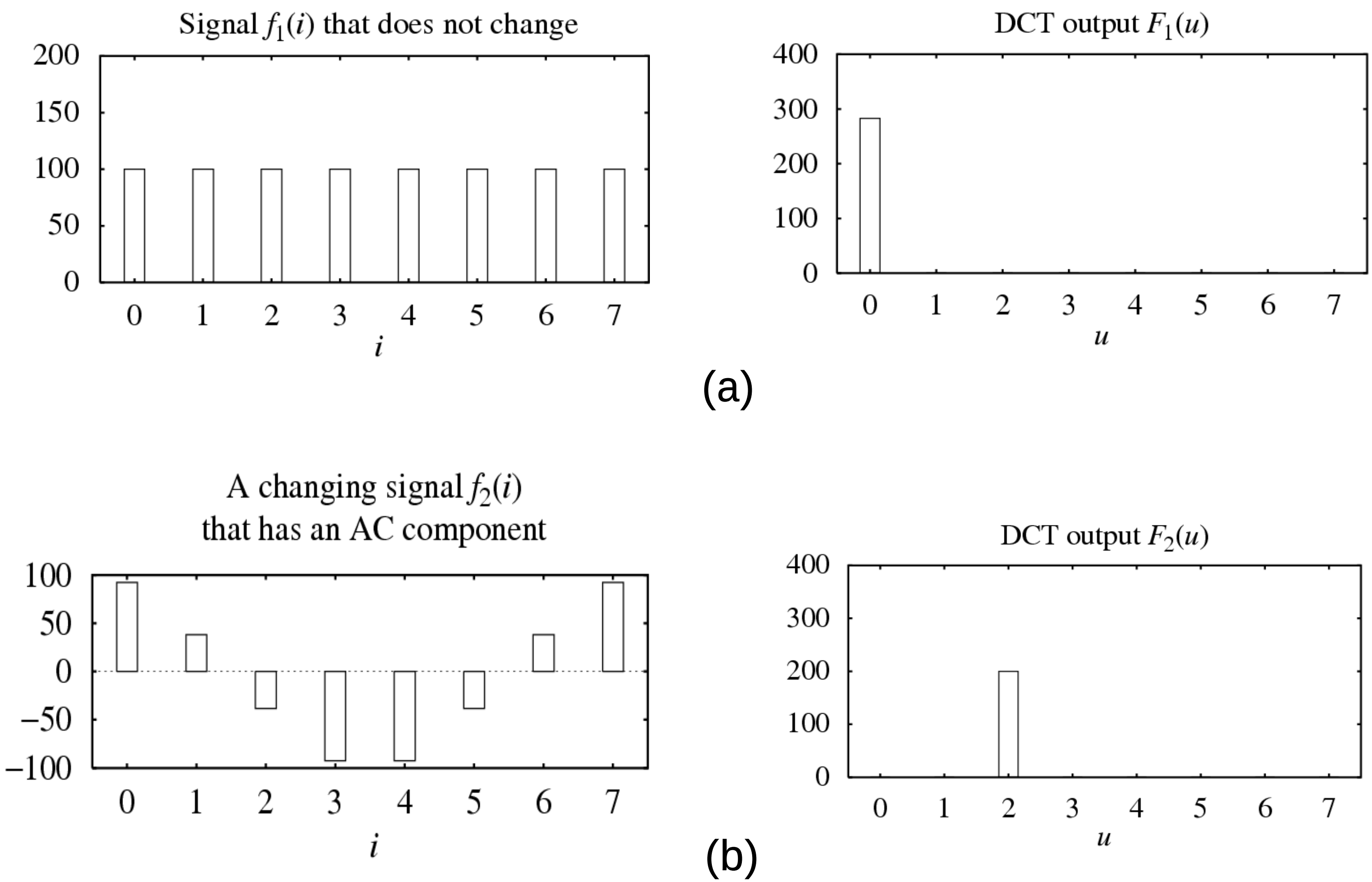
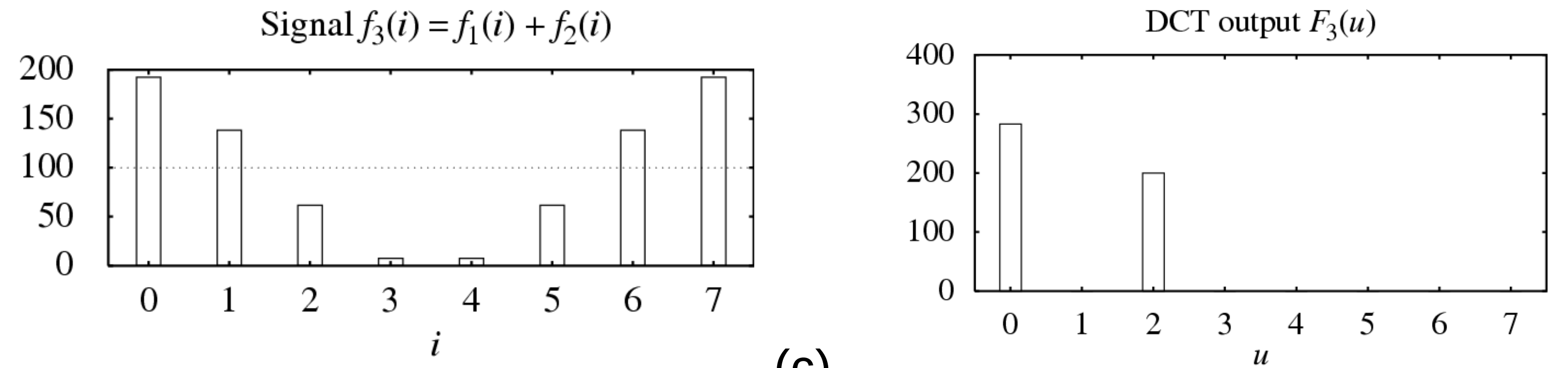
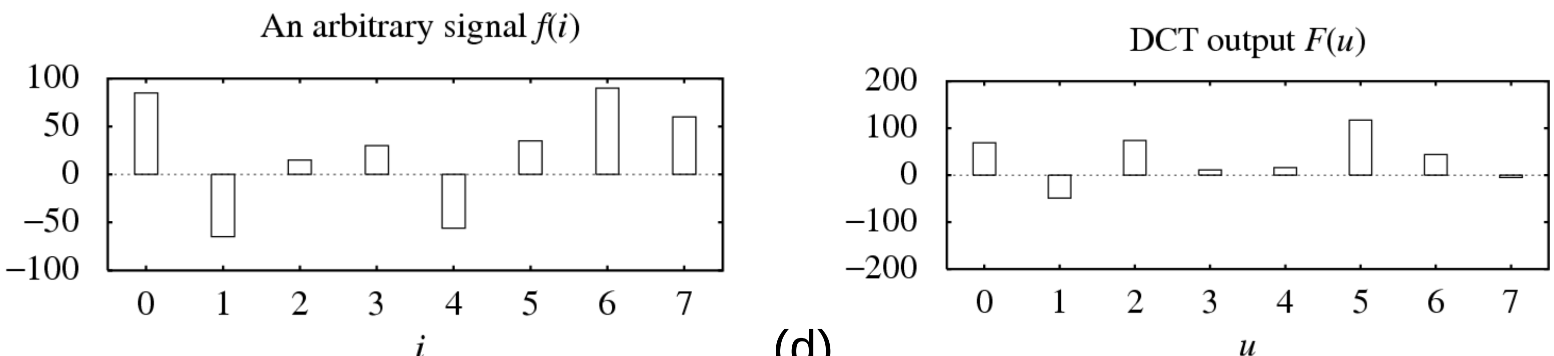


Fig. 8.7: Examples of 1D Discrete Cosine Transform:
(a) A DC signal $f_1(i)$, **(b)** An AC signal $f_2(i)$.



(c)



(d)

Fig. 8.7 (Cont'd): Examples of 1D Discrete Cosine Transform: **(c)** $f_3(i) = f_1(i) + f_2(i)$, and **(d)** an arbitrary signal $f(i)$.

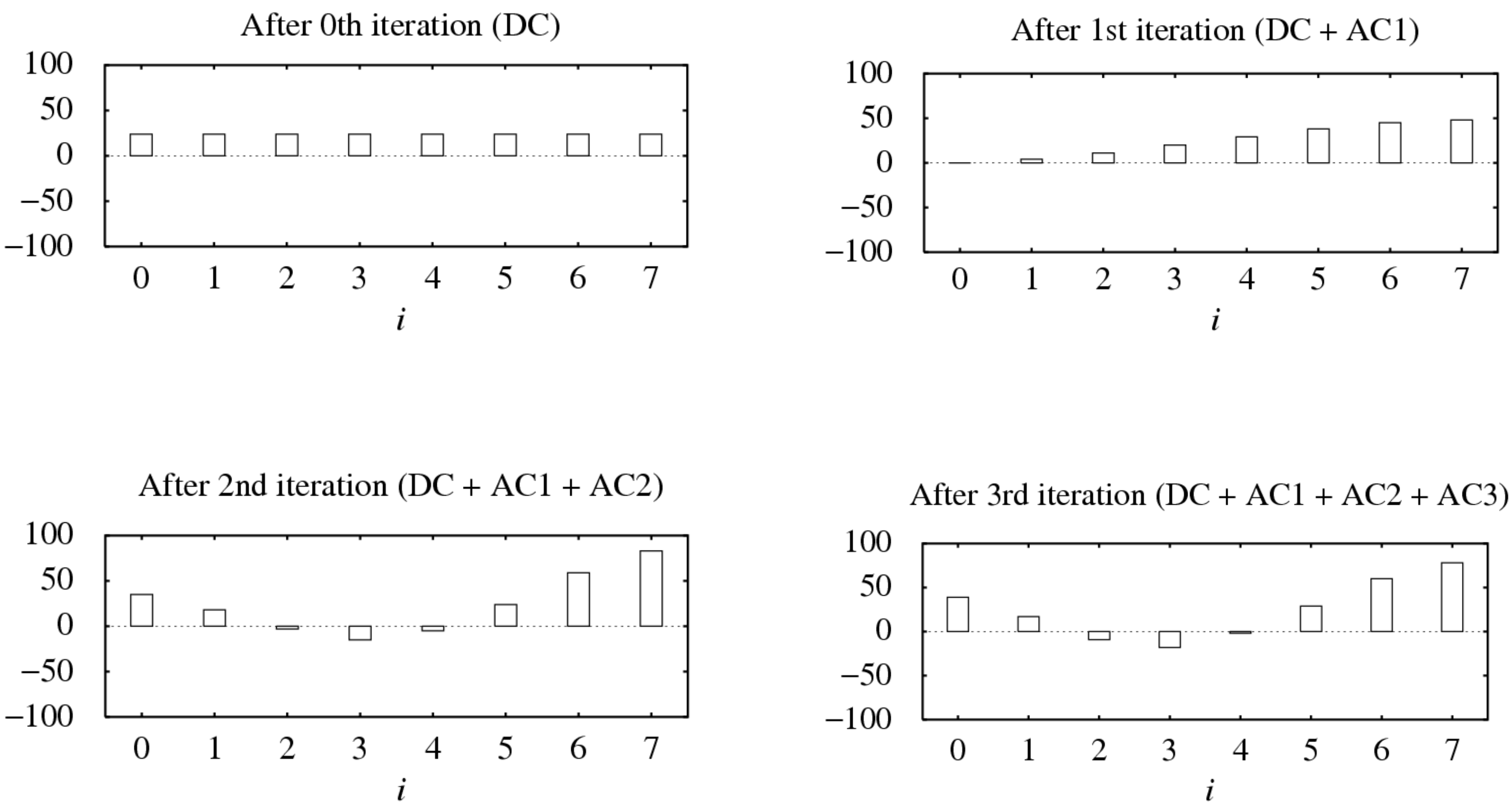


Fig. 8.8: An example of 1D IDCT.

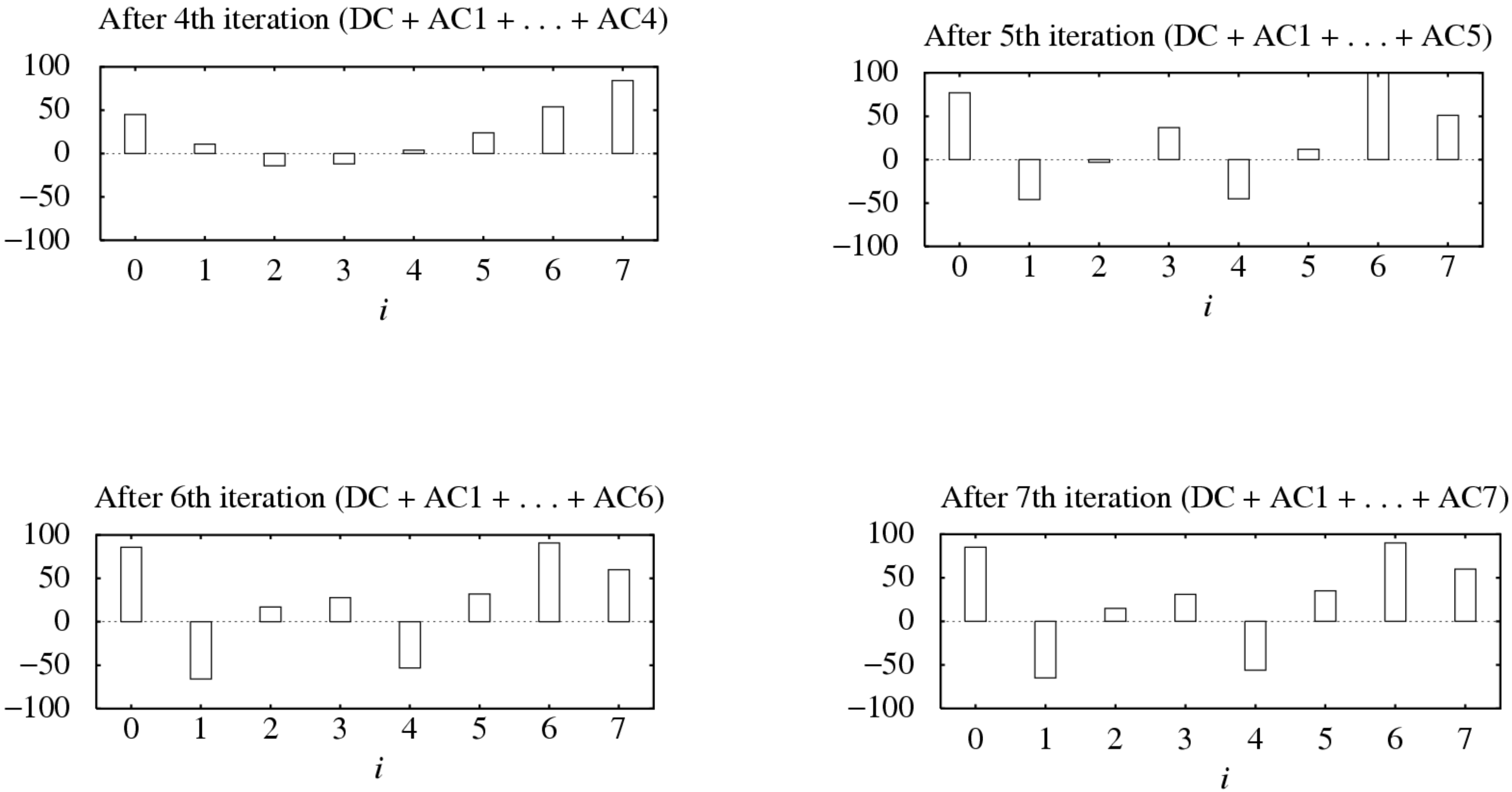


Fig. 8.8 (Cont'd): An example of 1D IDCT.

The DCT is a linear transform:

- In general, a transform T (or function) is linear, iff

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

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

- From the definition in Eq. 8.17 or 8.19, this property can readily be proven for the DCT because it uses only simple arithmetic operations.

The Cosine Basis Functions

- Function $B_p(i)$ and $B_q(i)$ are *orthogonal*, if

$$\sum_i \left[B_p(i) \cdot B_q(i)\right] = 0 \qquad if \quad p \neq q \qquad (8.22)$$

- Function $B_p(i)$ and $B_q(i)$ are *orthonormal*, if they are orthogonal and

$$\sum_i \left[B_p(i) \cdot B_q(i)\right] = 1 \qquad if \quad p = q \qquad (8.23)$$

- It can be shown that:

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

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

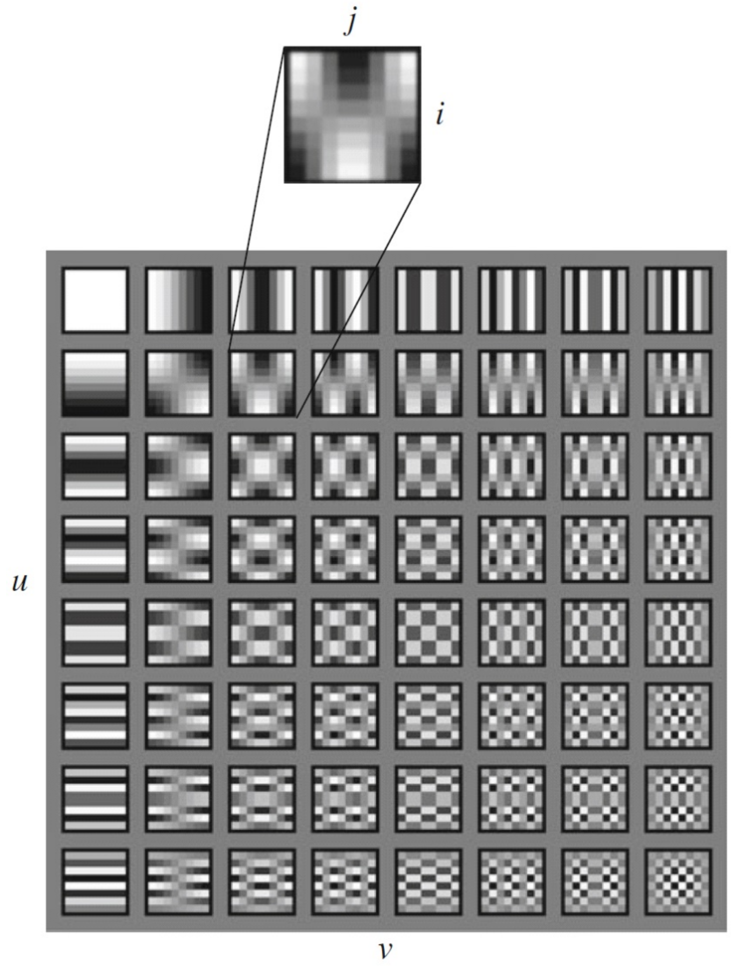


Fig. 8.9: Graphical Illustration of 8 × 8 2D DCT basis.

2D Basis Functions

- For a particular pair of *u* and *v*, the respective 2D basis function is:

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

- The enlarged block shown in Fig. 8.9 is for the basis function:

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

2D Separable Basis

- The 2D DCT can be *separated* into a sequence of two, 1D DCT steps:

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

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

- 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:

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

- 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 x N.

When N = 8, we have:

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

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

2D IDCT Matrix Implementation

The 2D IDCT matrix implementation is simply:

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

- See the textbook for step-by-step derivation of the above equation.
  - The key point is: the DCT-matrix is orthogonal, hence,

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

Comparison of DCT and DFT

- The discrete cosine transform is a close counterpart to the Discrete Fourier Transform (DFT). DCT is a transform that only involves the real part of the DFT.
- For a continuous signal, we define the continuous Fourier transform 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

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

- Because the use of digital computers requires us to discretize the input signal, we define a DFT that operates on 8 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

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

- 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.

- DCT of 8 input samples corresponds to DFT of the 16 samples made up of original 8 input samples and a symmetric copy of these, as shown in Fig. 8.10.

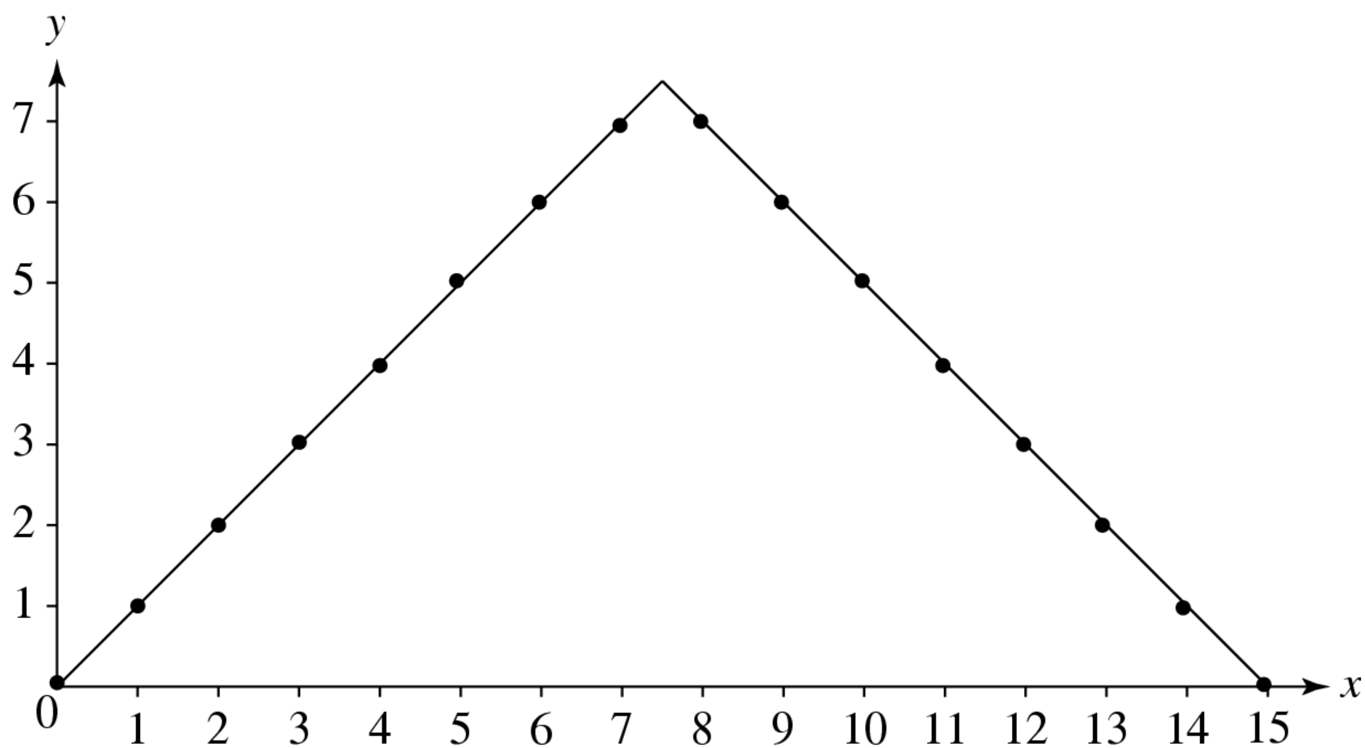


Fig. 8.10: Symmetric extension of the ramp function.

A Simple Comparison of DCT and DFT

Table 8.1 and Fig. 8.11 show the comparison of DCT and DFT on a ramp function, if only the first three terms are used.

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

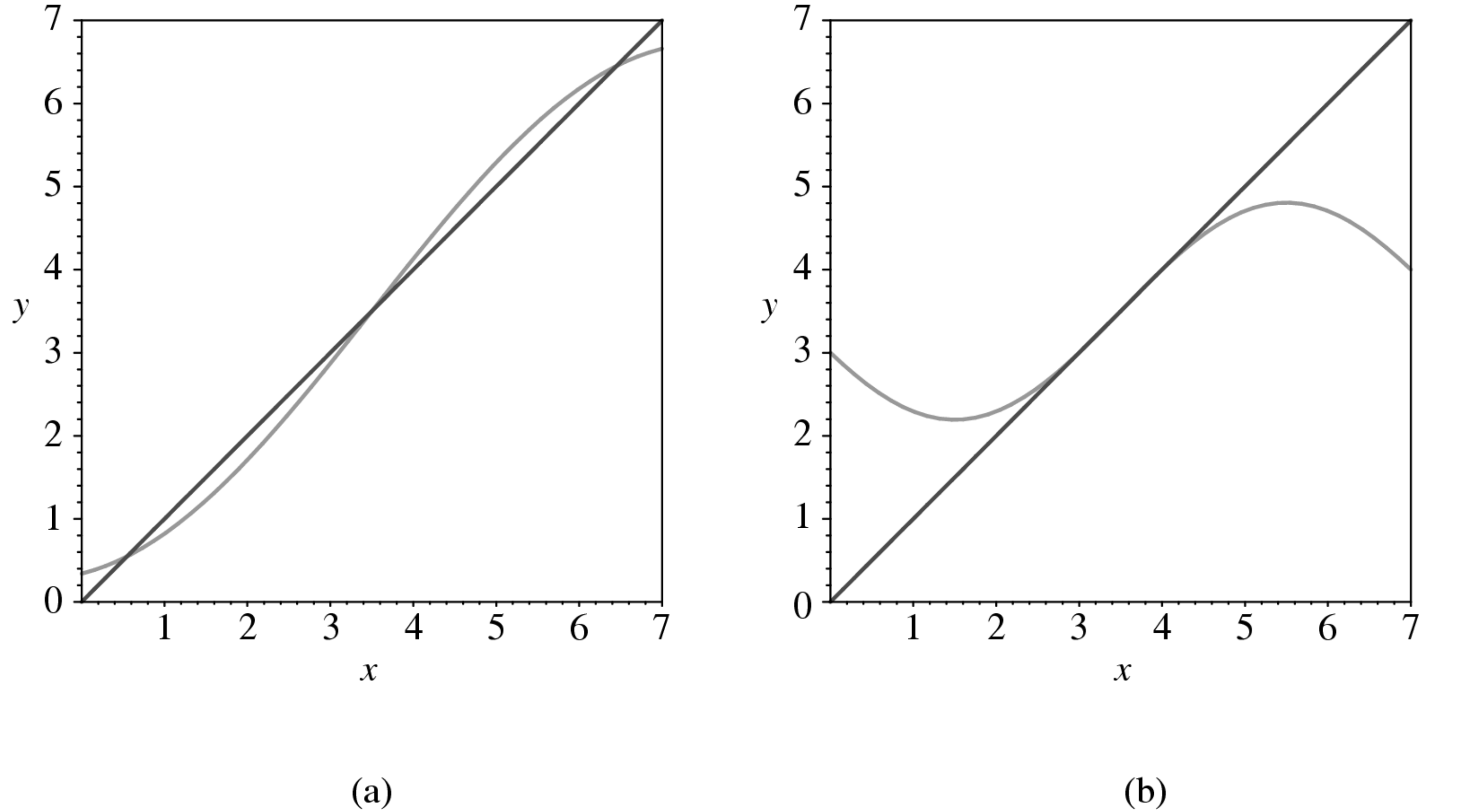


Fig. 8.11: Approximation of the ramp function: (a) 3 Term DCT Approximation, (b) 3 Term DFT Approximation.

8.5.2 Karhunen-Loève Transform (KLT)

- The Karhunen-Loève transform is a reversible linear transform that exploits the statistical properties of the vector representation.
- It optimally decorrelates the input signal.
- To understand the optimality of the KLT, consider the autocorrelation matrix $\mathbf{R}_\mathbf{X}$ of the input vector $\mathbf{X}$ defined as

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

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

- Our goal is to find a transform $\mathbf{T}$ such that the components of the output $\mathbf{Y}$ are uncorrelated, i.e $E[\mathrm{Y}_t \mathrm{Y}_s] = 0$, 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 non-negative definite, there are k orthogonal eigenvectors $u_1, u_2, \ldots, u_k$ and k corresponding real and nonnegative eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k \geq 0$.

- If 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

$$\mathbf{R_Y} = E[\mathbf{YY}^T] = E[\mathbf{TXX}^T\mathbf{T}] = \mathbf{TR_X T}^T \tag{8.39-8.41}$$

$$= \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ 0 & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \lambda_k \end{bmatrix} \tag{8.42}$$

KLT Example

To illustrate the mechanics of the KLT, consider the four 3D input vectors x_1 = (4, 4, 5), x_2 = (3, 2, 5), x_3 = (5, 7, 6), and x_4 = (6, 7, 7).

- Estimate the mean:

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

- Estimate the autocorrelation matrix of the input:

$$\mathbf{R}_{\mathrm{X}} = \frac{1}{M}\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T - \mathbf{m}_x \mathbf{m}_x^T \qquad (8.43)$$

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

- The eigenvalues of $\mathbf{R_X}$ are λ_1 = 6.1963, λ_2 = 0.2147, and λ_3 = 0.0264. The corresponding eigenvectors are

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

- 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 apply the KLT

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

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

- Since the rows of $\mathbf{T}$ are orthonormal vectors, the inverse transform is just the transpose: $\mathrm{T}^{-1} = \mathrm{T}^{T}$, and

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

- In general, after the KLT most of the "energy" of the transform coefficients are concentrated within the first few components. This is the "energy compaction" property of the KLT.

8.6 Wavelet-Based Coding

- The objective of the wavelet transform is to decompose the input signal into components that are easier to deal with, have special interpretations, or have some components that can be thresholded away, for compression purposes.
- We want to be able to at least approximately reconstruct the original signal given these components.
- The basis functions of the wavelet transform are localized in both time and frequency.
- There are two types of wavelet transforms: the continuous wavelet transform (CWT) and the discrete wavelet transform (DWT).

Wavelet Transform Example

Suppose we are given the following input sequence.

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

- 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:

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

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

- 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 of the number of elements in the original sequence.

- 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

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

- 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 contain 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.

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

$$x_{n,\,2i} = x_{n-1,\,i} + d_{n-1,\,i}$$
$$x_{n,\,2i+1} = x_{n-1,\,i} - d_{n-1,\,i} \tag{8.49}$$

- This transform is the discrete Haar wavelet transform.

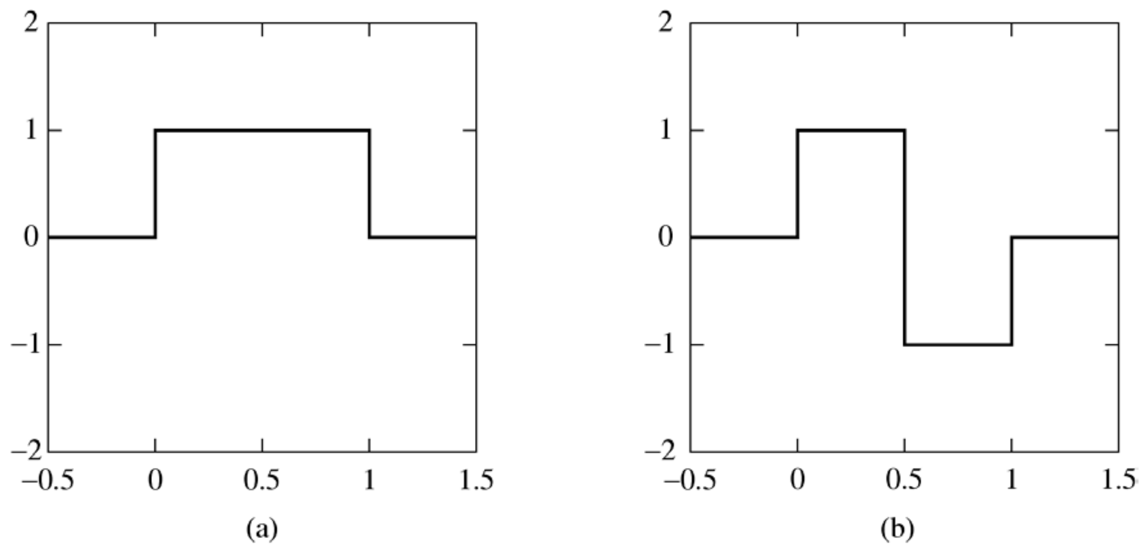
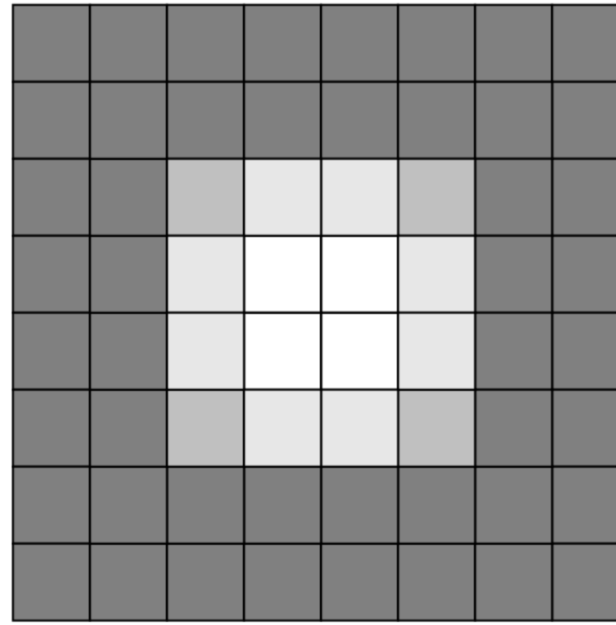


Fig. 8.12: Haar Transform: (a) scaling function, (b) wavelet function.

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

(a)



(b)

Fig. 8.13: Input image for the 2D Haar Wavelet Transform.
(a) The pixel values. (b) Shown as an 8 × 8 image.

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.

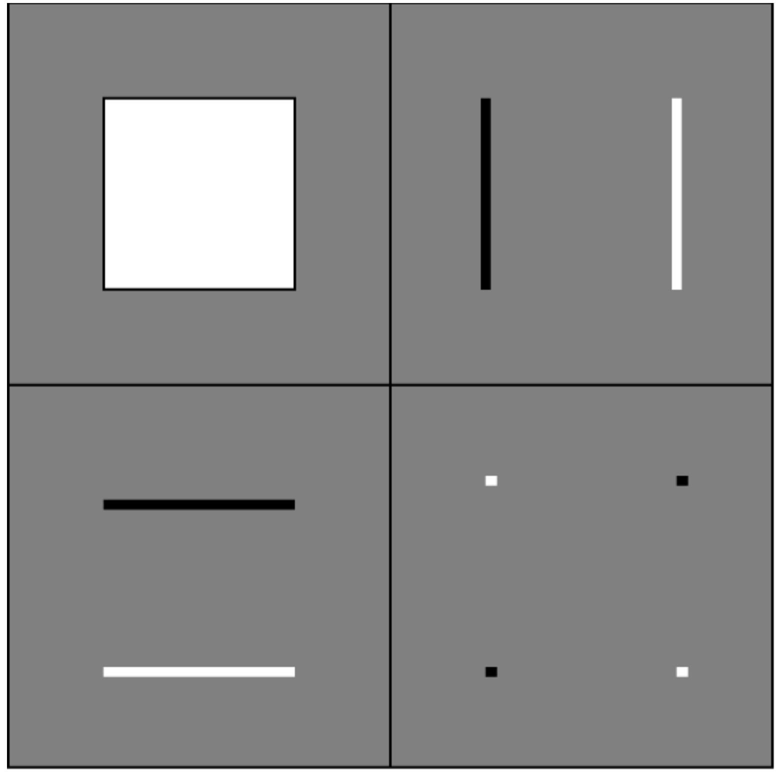


Fig. 8.16: A simple graphical illustration of Wavelet Transform.

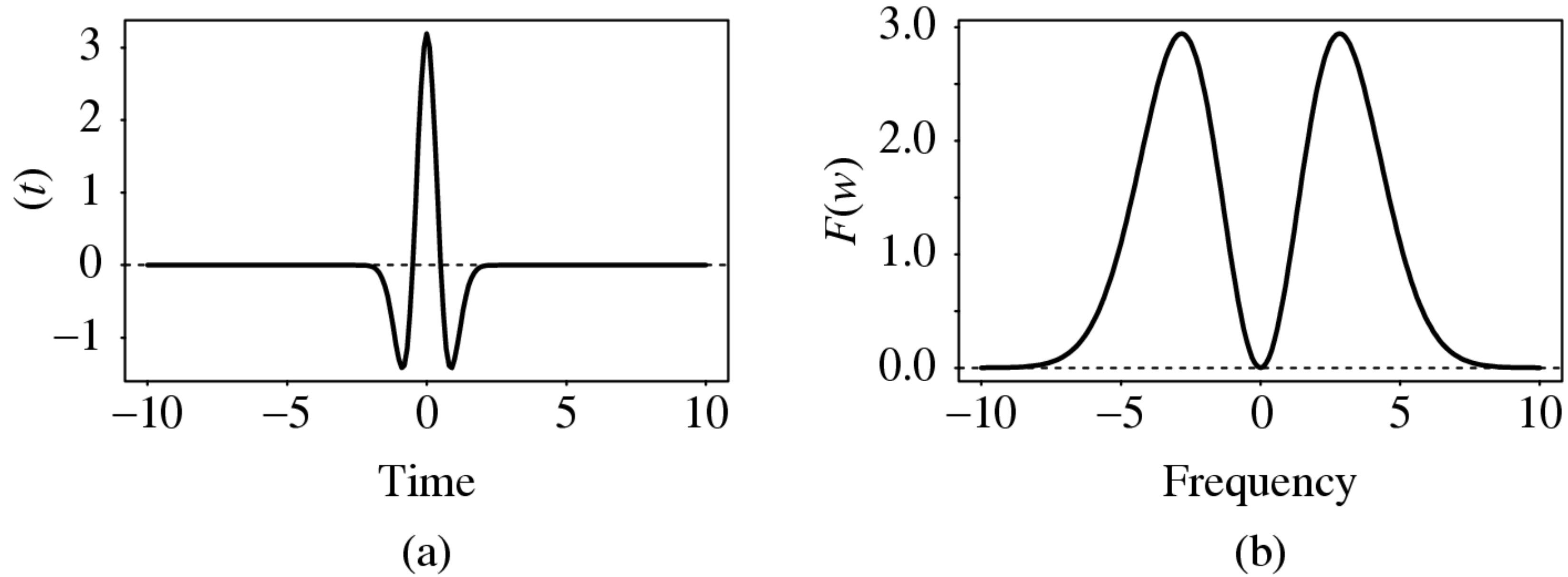


Fig. 8.17: A Mexican Hat Wavelet:
(a) $\sigma = 0.5$, (b) its Fourier transform.

8.6.2 Continuous Wavelet Transform

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

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

- Another way to state the admissibility condition is that the 0th moment M_0 of $\psi(t)$ is zero. The pth moment is defined as

$$M_p = \int_{-\infty}^{\infty} t^p \psi(t)dt \tag{8.56}$$

- The function ψ is normalized, i.e., $||\psi|| = 1$ and centered at $t = 0$. A family of wavelet functions is obtained by scaling and translating the "mother wavelet" ψ

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

- The *continuous wavelet transform* (CWT) of $f \in \mathbf{L}^2(R)$ at time u and scale s is defined as:

$$W(f, s, u) = \int_{-\infty}^{+\infty} f(t)\psi_{s,u}(t)\, dt \qquad (8.58)$$

- The inverse of the continuous wavelet transform is:

$$f(t) = \frac{1}{C_\psi} \int_0^{+\infty} \int_{-\infty}^{+\infty} W(f, s, u) \frac{1}{\sqrt{s}} \psi\left(\frac{t-u}{s}\right) \frac{1}{s^2}\; du\, ds \qquad (8.59)$$

where

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

and $\psi(w)$ is the Fourier transform of $\psi(t)$.

8.6.3 Discrete Wavelet Transform

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

- The DWT makes the connection between wavelets in the continuous time domain and "filter banks" in the discrete time domain in a multiresolution analysis framework.

- It is possible to show that the dilated and translated family of wavelets ψ

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

form an *orthonormal* basis of $\mathbf{L}^2(R)$.

Multiresolution Analysis in the Wavelet Domain

- Multiresolution analysis provides the tool to *adapt signal resolution* to *only relevant details* for a particular task.

  The approximation component is then recursively decomposed into approximation and detail at successively coarser scales.

- Wavelet functions $\psi(t)$ are used to characterize detail information. The averaging (approximation) information is formally determined by a kind of dual to the mother wavelet, called the "scaling function" $\varphi(t)$.

- Wavelets are set up such that the approximation at resolution 2^{-j} contains all the necessary information to compute an approximation at coarser resolution $2^{-(j+1)}$.

- The scaling function must satisfy the so-called **dilation equation**:

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

- The *wavelet* at the coarser level is also expressible as a sum of translated scaling functions:

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

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

- 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.

Block Diagram of 1D Dyadic Wavelet Transform

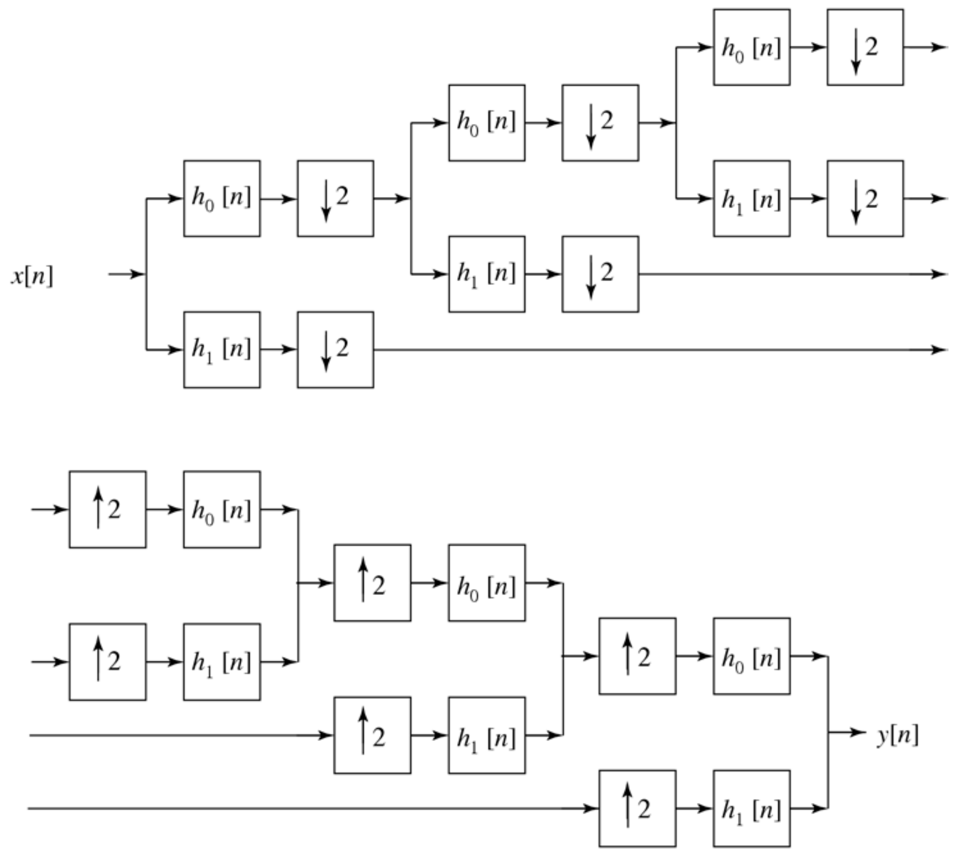


Fig. 8.18: The block diagram of the 1D dyadic wavelet transform.

Biorthogonal Wavelets

- 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". 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 $\tilde{h}_0[n]$.

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

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

Table 8.2: Orthogonal Wavelet Filters

Wavelet	Num. 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]

Table 8.3: Biorthogonal Wavelet Filters

Wavelet	Filter	Num. 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

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

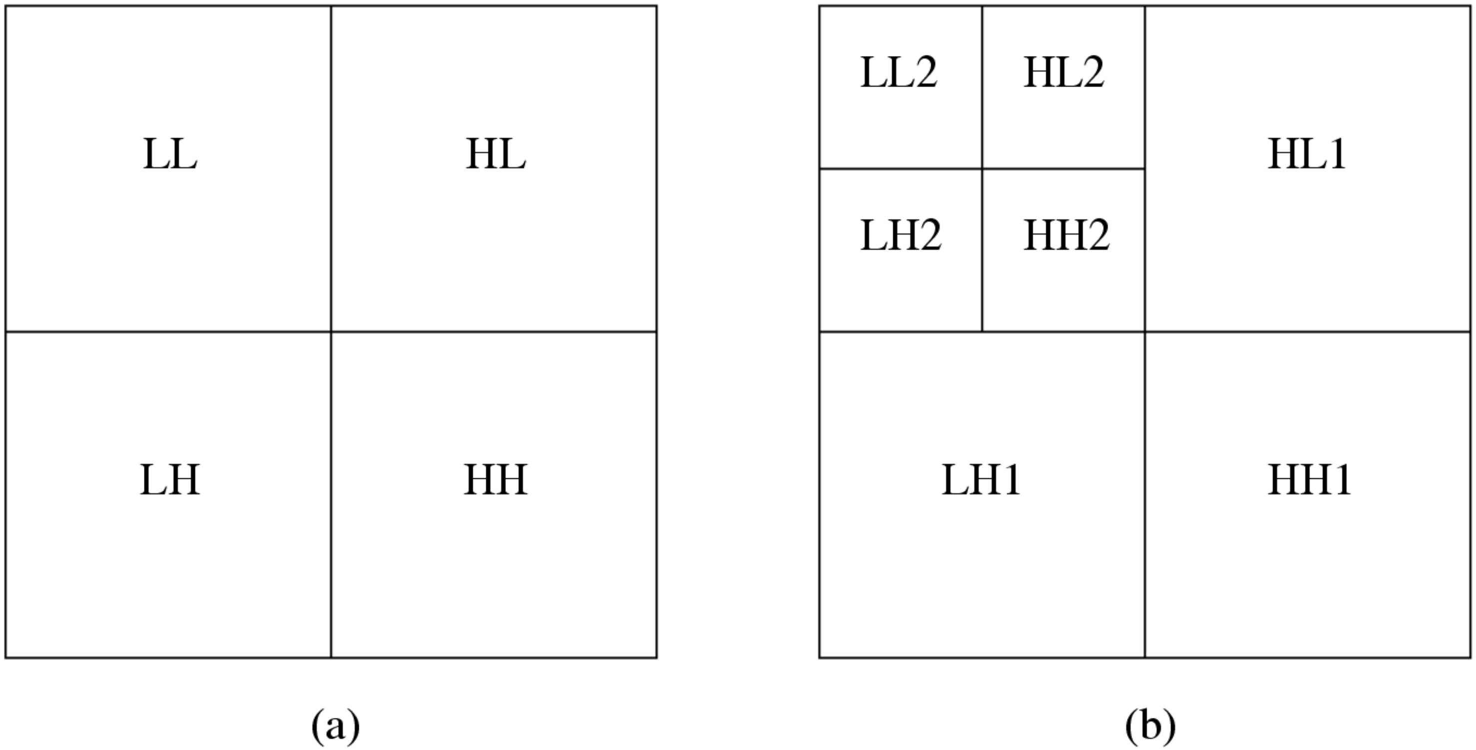
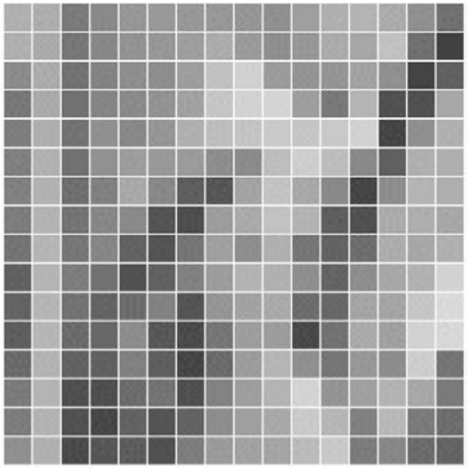


Fig. 8.19: The two-dimensional discrete wavelet transform (a) One level transform, (b) two level transform.

2D Wavelet Transform Example

- The input image is a sub-sampled version of the image Lena. The size of the input is 16×16. The filter used in the example is the Antonini 9/7 filter set

(a)



(b)

Fig. 8.20: The Lena image: (a) Original 128 × 128 image. (b) 16 × 16 sub-sampled image.

- The input image is shown in numerical form below.

$$I_{00}(x,y) =$$

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

- First, we need to compute the analysis and synthesis high-pass filters.

$$h_1[n] = [-0.065, 0.041, 0.418, -0.788, 0.418, 0.041, -0.065]$$

$$\tilde{h}_1[n] = [-0.038, -0.024, 0.111, 0.377, -0.853, 0.377, 0.111, -0.024, -0.038] \qquad (8.70)$$

- Convolve 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:

$$(I_{00}(:,0) * h_0[n])\downarrow 2 = [245, 156, 171, 183, 184, 173, 228, 160]$$

$$(I_{00}(:,0) * h_1[n])\downarrow 2 = [-30, 3, 0, 7, -5, -16, -3, 16]$$

- 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]

- Continue the same process for the remaining rows.

The result after all rows have been processed:

$$I_{00}(x,y) \; = $$

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

- Apply the filters to the columns of the resulting image. Apply both $h_0[n]$ and $h_1[n]$ to each column and discard the odd indexed results:

$$(I_{11}(0,:) * h_0[n]) \downarrow 2 = [353, 280, 269, 256, 240, 206, 160, 153]^T$$

$$(I_{11}(0,:) * h_1[n]) \downarrow 2 = [-12, 10, -7, -4, 2, -1, 43, 16]^T$$

- Concatenate the above results into a single column and apply the same procedure to each of the remaining columns.

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

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

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

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

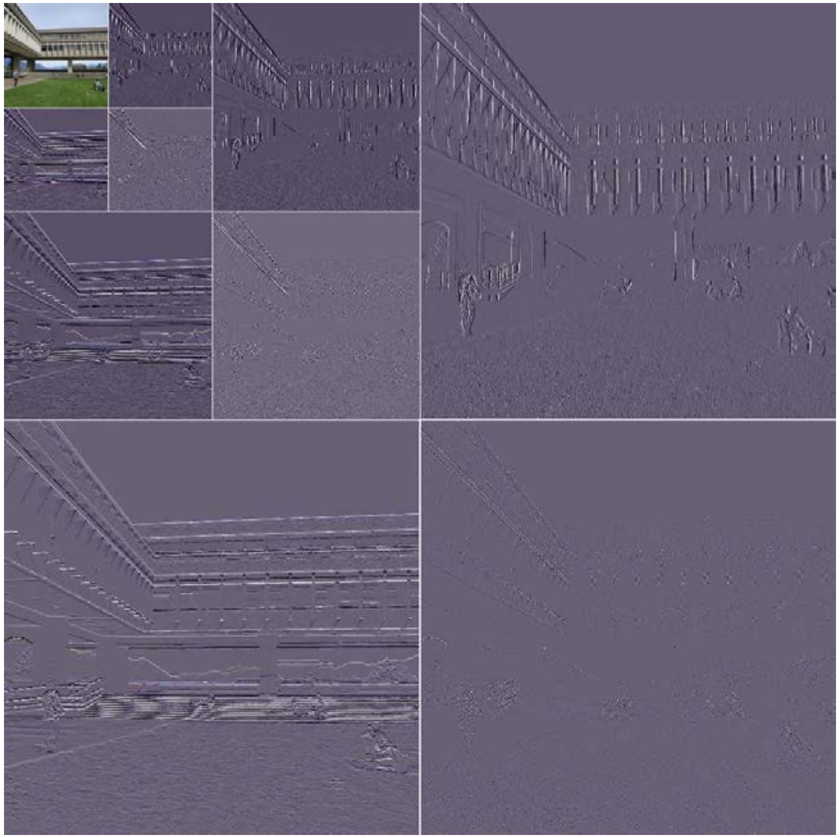


Fig. 8.21: Haar wavelet decomposition.

8.7 Wavelet Packets

- 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.

- A wavelet packet decomposition allows the decomposition to be represented by any pruned subtree of the full tree topology.

- The wavelet packet decomposition is very flexible since a best wavelet basis in the sense of some cost metric can be found within a large library of permissible bases.

- The computational requirement for wavelet packet decomposition is relatively low as each decomposition can be computed in the order of $N \log N$ using fast filter banks.

8.8 Embedded Zerotree of Wavelet Coefficients

- Effective and computationally efficient for image coding.

- The EZW algorithm addresses two problems:
  1. obtaining the best image quality for a given bit-rate, and
  2. accomplishing this task in an embedded fashion.

- Using an embedded code allows the encoder to terminate the encoding at any point. Hence, the encoder is able to meet any target bit-rate exactly.

- Similarly, a decoder can cease to decode at any point and can produce reconstructions corresponding to all lower-rate encodings.

8.8.1 The Zerotree Data Structure

- The EZW algorithm efficiently codes the "significance map" which indicates the locations of nonzero quantized wavelet coefficients.

  This is achieved using a new data structure called the *zerotree*.

- Using the hierarchical wavelet decomposition presented earlier, we can relate every coefficient at a given scale to a set of coefficients at the next finer scale of similar orientation.

- 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".

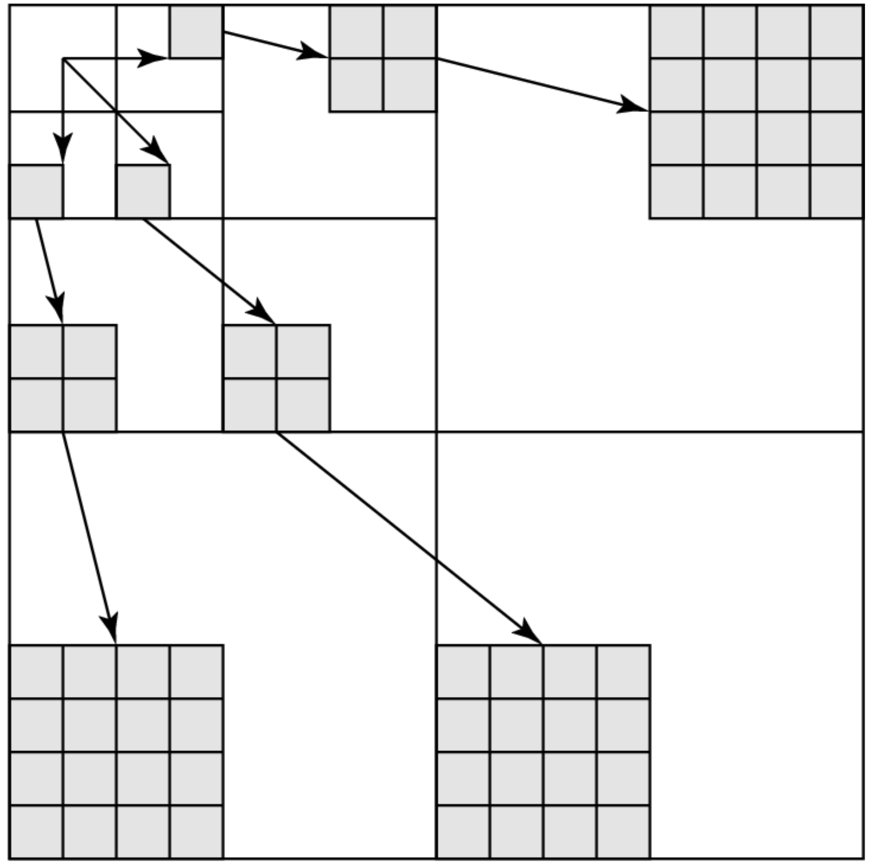


Fig. 8.22: Parent child relationship in a zerotree.

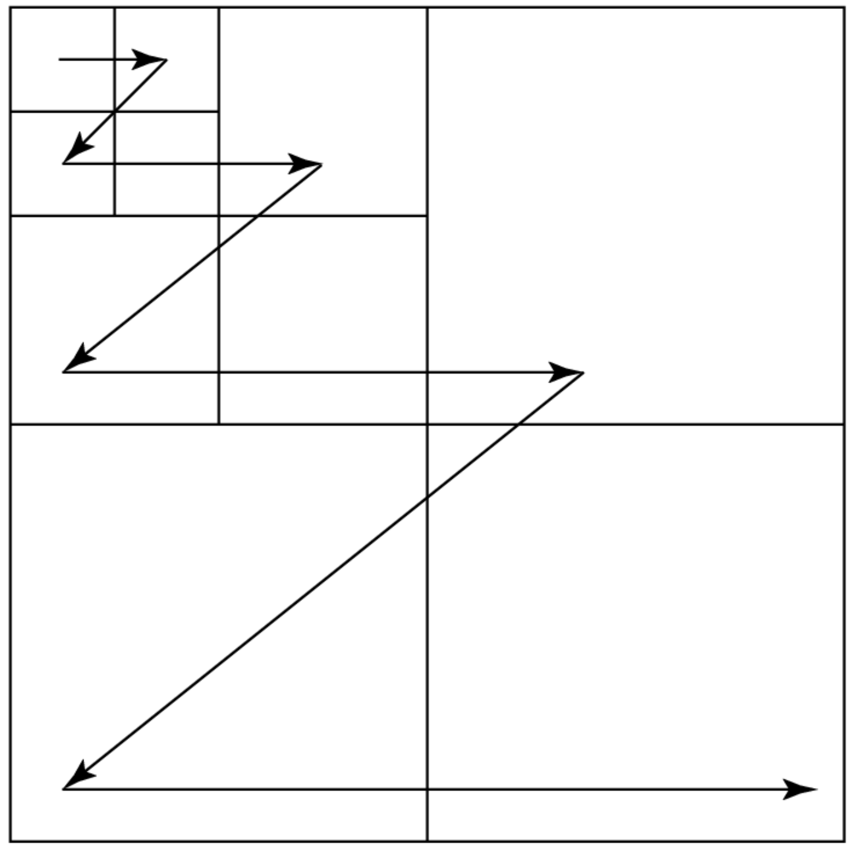


Fig. 8.23: EZW scanning order.

- Given a threshold T, a coefficient x is an element of the zerotree if it is insignificant and all of its descendants are insignificant as well.

- The significance map is coded using the zerotree with a four-symbol alphabet:
  - **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.

Successive Approximation Quantization

- **Motivation**:
  - Takes advantage of the efficient encoding of the significance map using the zerotree data structure by allowing it to encode more significance maps.
  - Produce an embedded code that provides a coarse-to-fine, multiprecision logarithmic representation of the scale space corresponding to the wavelet-transformed image.

- The SAQ method sequentially applies a sequence of thresholds $T_0, \ldots, T_{N-1}$ to determine the significance of each coefficient.

- A *dominant list* and a *subordinate list* are maintained during the encoding and decoding process.

Dominant Pass

- Coefficients having their coordinates on the dominant list implies that they are not yet significant.

- 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 0 to enable the possibility of the occurrence of a zerotree on future dominant passes at smaller thresholds.

- The resulting significance map is zerotree coded.

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.

- 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.

- After the completion of 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.

EZW Example

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.

Encoding

- Since the largest coefficient is 57, the initial threshold T_0 is 32.
- At the beginning, the dominant list contains the coordinates of all 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 a n to represent them.

- The coefficient −29 is insignificant, but contains a significant descendant 33 in LH1. Therefore, it is coded as z.

- Continuing in this manner, the dominant pass outputs the following symbols:

$$D_0 : pnztpttptztttttttttttpttt$$

- There are five coefficients 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).

- 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.

$$S_0 : 10000$$

- Now the dominant list contains the coordinates of all the coefficients except those found to be significant and the subordinate list contains the values:

$$\{57, 37, 39, 33, 33\}.$$

- Now, 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.

- 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.

- 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 second dominant pass is $T_1 = 16$. Using the same procedure as above, the dominant pass outputs the following symbols

$$D_1\text{: } zznptnpttztptttttttttttttptttttt \tag{8.71}$$

- 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:

$$S_1 : 10000110000$$

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

D_2 : *zzzzzzzzptpzpptnttptppttpttpttpnpptttttptttttttttttttt*

S_2 : 01100111001101100000110110

D_3 : *zzzzzzztzpztztnttptttttptnnttttptttpptppttpttttt*

S_3 : 00100010001110100110001001111101100010

D_4 : *zzzzzttztztzztzzpttpppttttpttpttnpttptptttpt*

S_4 : 1111101001101011000001011101101100010010010101010

D_5 : *zzzztzttttztzzzzttpttptttttnptppttttppttp*

Decoding

- Suppose we only received information from the first dominant and subordinate pass. 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 reconstruct the value of these coefficients using the center of the uncertainty interval.

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 pass.

- If the decoder received only D_0, S_0, D_1, S_1, D_2, and only the first 10 bits of S_2, then the reconstruction is

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.

8.9 Set Partitioning in Hierarchical Trees (SPIHT)

- The SPIHT algorithm is an extension of the EZW algorithm.

- The SPIHT algorithm significantly improved the performance of its predecessor by changing the way subsets of coefficients are partitioned and how 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 only produce very marginal gain in compression.

- No ordering information is explicitly transmitted to the decoder. Instead, the decoder reproduces the execution path of the encoder and recovers the ordering information.