UNIVERSITÀ DEGLI STUDI DI TRIESTE

Image Compression

Alberto Carini

Preview

Image compression, *the art and science of reducing the amount of data required to represent an image*, is one of the most useful and commercially successful technologies in the field of digital image processing.

Everyone who owns a digital camera, surfs the web, or streams the latest Hollywood movies over the Internet benefits from the algorithms and standards that will be discussed here.

The material we will see is applicable to both still-image and video applications.

Fundamentals

The term **data compression** refers to the process of reducing the amount of data required to represent a given quantity of information.

In this definition, **data** and **information** are not the same; data are the means by which information is conveyed.

Because various amounts of data can be used to represent the same amount of information, representations that contain irrelevant or repeated information are said to contain **redundant data**.

If we let b and b′ denote the number of bits in two representations of the same information, the **relative data redundancy**, R, of the representation with b bits is

$$R = 1 - \frac{1}{C}$$

where C, commonly called the **compression ratio**, is defined as

$$C = \frac{b}{b'}$$

In the context of digital image compression, b usually is the number of bits needed to represent an image as a 2-D array of intensity values.

Fundamentals

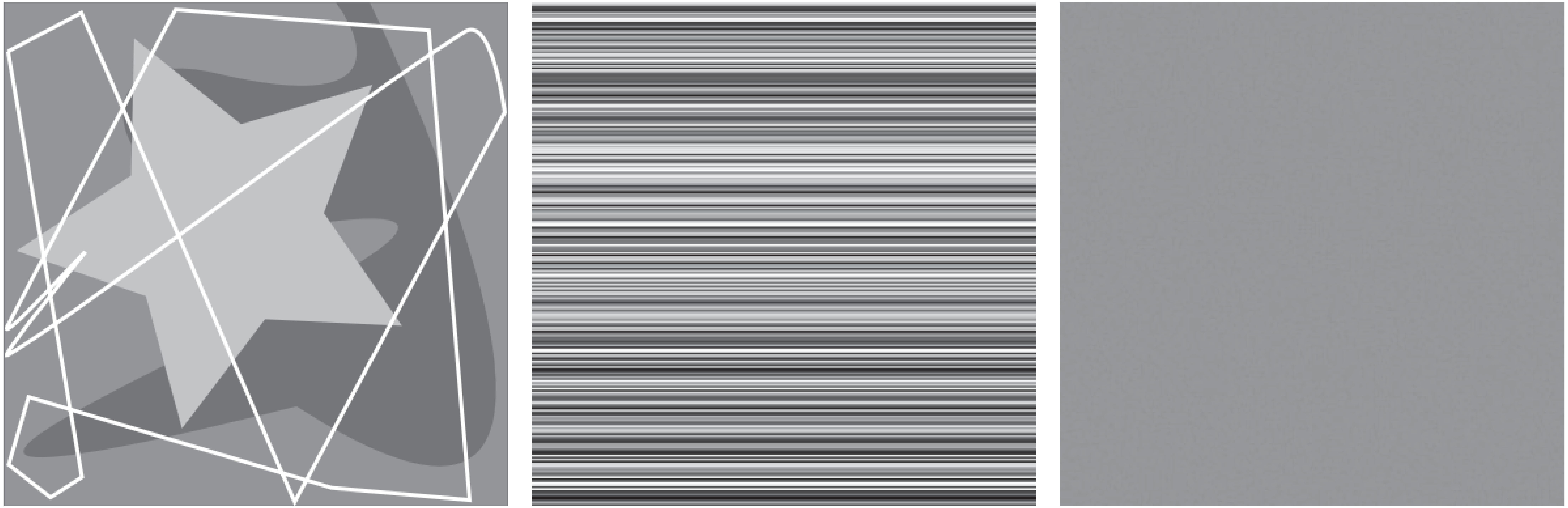
Two-dimensional intensity arrays suffer from three principal types of data redundancies:

1. **Coding redundancy**. A code is a system of symbols (letters, numbers, bits) used to represent a body of information or set of events. Each piece of information or event is assigned a sequence of code symbols, called a **code word**. The number of symbols in each code word is its **length**. The 8-bit codes that are used to represent the intensities in most 2-D intensity arrays contain more bits than are needed to represent the intensities.

2. **Spatial and temporal redundancy.** Because the pixels of most 2-D intensity arrays are *correlated spatially*, information is unnecessarily replicated in the representations of the correlated pixels. In a video sequence, *temporally correlated* pixels also duplicate information.

3. **Irrelevant information.** Most 2-D intensity arrays contain information that is ignored by the human visual system and/or extraneous to the intended use of the image. It is redundant in the sense that it is not used.

Fundamentals



a b c

FIGURE 8.1 Computer generated $256 \times 256 \times 8$ bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy, but may exhibit others as well.)

Coding redundancy

Assume that a discrete random variable r_k in the interval $[0, L - 1]$ is used to represent the intensities of an M × N image, and that each r_k occurs with probability $p_r(r_k)$.

$$p_r(r_k) = \frac{n_k}{MN} \qquad k = 0, 1, 2, \ldots, L-1$$

where L is the number of intensity values, and n_k is the number of times that the k-th intensity appears in the image.

If the number of bits used to represent each value of r_k is $l(r_k)$, then the average number of bits required to represent each pixel is

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

The total number of bits required to represent an M × N image is MNL_{avg}.

If the intensities are represented using a natural m-bit fixed-length code, $L_{avg} = m$.

Coding redundancy

In the first figure *m=8.* Consider the following variable length code:

TABLE 8.1
Example of variable-length coding.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	01010111	8	1	1
$r_{186} = 186$	0.25	01010111	8	000	3
$r_{255} = 255$	0.03	01010111	8	001	3
r_k for $k = 87, 128, 186, 255$	0	—	8	—	0

$$L_{\text{avg}} = 0.25(2) + 0.47(1) + 0.03(3) = 1.81 \text{ bits}$$

$$C = \frac{256 \times 256 \times 8}{118{,}621} = \frac{8}{1.81} \approx 4.42 \qquad R = 1 - \frac{1}{4.42} = 0.774$$

Coding redundancy

As the preceding example shows, coding redundancy is present when the codes assigned to a set of events (such as intensity values) do not take full advantage of the probabilities of the events.

Coding redundancy is almost always present when the intensities of an image are represented using a natural binary code.

The reason is that most images are composed of objects that have a regular and somewhat predictable morphology (shape) and reflectance, and are sampled so the objects being depicted are much larger than the picture elements.

The natural consequence is that, for most images, certain intensities are more probable than others.

A natural binary encoding assigns the same number of bits to both the most and least probable values, failing to minimize L_{avg}, and resulting in coding redundancy.

Spatial and temporal redundancy

Consider the computer-generated collection of constant intensity lines in the second figure.

In the corresponding 2-D intensity array:

1. All 256 intensities are equally probable: the histogram of the image is uniform.
2. Because the intensity of each line was selected randomly, its pixels are independent of one another in the vertical direction.
3. Because the pixels along each line are identical, they are maximally correlated in the horizontal direction.

Observations 2 and 3 reveal a significant spatial redundancy that can be eliminated by representing the image as a sequence of run-length pairs, where each run-length pair specifies the start of a new intensity and the number of consecutive pixels that have that intensity.

Spatial and temporal redundancy

In most images, pixels are correlated spatially (in both x and y) and in time (in videos).

Because most pixel intensities can be predicted reasonably well from neighboring intensities, the information carried by a single pixel is small.

Much of its visual contribution is redundant in the sense that it can be inferred from its neighbors.

To reduce the redundancy associated with spatially and temporally correlated pixels, a 2-D intensity array must be transformed into a more efficient but usually “non-visual” representation. For example, run-lengths or the differences between adjacent pixels.

Transformations of this type are called **mappings**.

A mapping is said to be **reversible** if the pixels of the original 2-D intensity array can be reconstructed without error from the transformed data set; otherwise, the mapping is said to be **irreversible**.

Irrelevant information

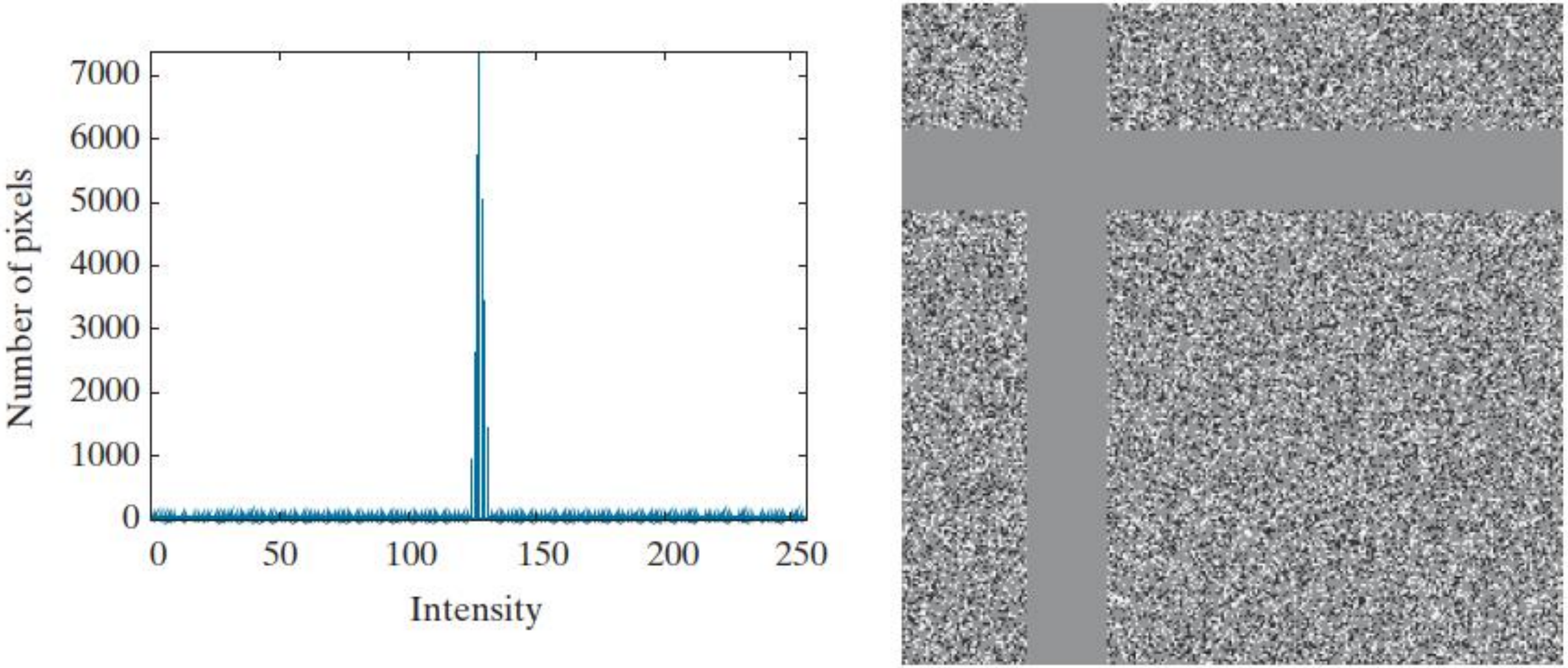
One of the simplest ways to compress a set of data is to remove superfluous data from the set.

In the context of digital image compression, information that is ignored by the human visual system, or is extraneous to the intended use of an image, are obvious candidates for omission.

Irrelevant information

a b

FIGURE 8.3
(a) Histogram of the image in Fig. 8.1(c) and (b) a histogram equalized version of the image.



Whether or not this information should be preserved is application dependent. If the information is important, as it might be in a medical application like digital X-ray archival, it should not be omitted; otherwise, the information is redundant and can be excluded for the sake of compression performance.

Irrelevant information

The redundancy examined here is fundamentally different from the redundancies discussed in the previous two sections.

Its elimination is possible because the information itself is not essential for normal visual processing and/or the intended use of the image.

Because its omission results in a loss of quantitative information, its removal is commonly referred to as **quantization**.

Because information is lost, quantization is an **irreversible operation**.

Measuring image information

How few bits are actually needed to represent the information in an image?

That is, is there a minimum amount of data that is sufficient to describe an image without losing information?

Information theory provides the mathematical framework to answer this questions.

Its fundamental premise is that the generation of information can be modeled as a probabilistic process which can be measured in a manner that agrees with intuition.

A random event *E* with probability *P(E)* is said to contain

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

units of information. If $P(E) = 1$, $I(E) = 0$ and no information is attributed to it.

The base of the logarithm determines the unit used to measure information.

If the base 2 is selected, the unit of information is the bit.

Note that if P(E) = ½, I(E) = - $\log_2$ ½ or 1 bit.

Measuring image information

Given a source of statistically independent random events from a discrete set of possible events $\{a_1, a_2, \ldots, a_J\}$ with associated probabilities $\{P(a_1), P(a_2), \ldots, P(a_J)\}$, the *average information per source output*, called the **entropy** of the source, is

$$H = -\sum_{j=1}^{J} P(a_j) \log P(a_j)$$

The a_j in this equation are called **source symbols**. Because they are statistically independent, the source itself is called a **zero-memory source**.

If an image is considered to be the output of an imaginary zero-memory "intensity source," we can use the histogram of the observed image to estimate the symbol probabilities of the source. Then, the intensity source's entropy becomes

$$\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_r(r_k)$$

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Measuring image information

For the first image :

$$
\begin{aligned}
\tilde{H} &= -[0.25\log_2 0.25 + 0.47\log_2 0.47 + 0.25\log_2 0.25 + 0.03\log_2 0.03] \\
&= -[0.25(-2) + 0.47(-1.09) + 0.25(-2) + 0.03(-5.06)] \\
&\approx 1.6614 \text{ bits/pixel}
\end{aligned}
$$

The variable length code gave 1.81 bit/pixel.

Although this is higher than the 1.6614 bits/pixel entropy estimate, **Shannon's first theorem**, also called the **noiseless coding theorem**, assures us that the image can be represented with as few as 1.6614 bits/pixel.

To prove it in a general way, Shannon looked at representing groups of consecutive source symbols with a single code word, and showed that

$$\lim_{n\to\infty}\left[\frac{L_{\text{avg},n}}{n}\right] = H$$

with $L_{avg,n}$ the average number of code symbols required to represent all n-symbol groups.

Measuring image information

Finally, we note that although the entropy provides a lower bound on the compression that can be achieved when directly coding statistically independent pixels, *it breaks down when* the pixels of an image are *correlated*.

Blocks of correlated pixels can be coded with fewer average bits per pixel than the equation predicts.

Less correlated descriptors (such as intensity run-lengths) are normally selected and coded.

When the output of a source of information depends on a finite number of preceding outputs, the source is called a **Markov source** or **finite memory source**.

Fidelity criteria

Because information is lost, a means of quantifying the nature of the loss is needed.
Two types of criteria can be used for such an assessment:
(1) objective fidelity criteria, and (2) subjective fidelity criteria.

The **root-mean-squared error**, e_{rms}, is the square root of the squared error averaged over the M×N array, or

$$e_{\mathrm{rms}} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x,y) - f(x,y) \right]^2 \right]^{1/2}$$

the **mean-squared signal-to-noise ratio** of the output image, denoted SNR_{ms}, can be defined as

$$\mathrm{SNR}_{\mathrm{ms}} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x,y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x,y) - f(x,y) \right]^2}$$

Fidelity criteria

TABLE 8.2
Rating scale of the Television Allocations Study Organization. (Frendendall and Behrend.)

Value	Rating	Description
1	Excellent	An image of extremely high quality, as good as you could desire.
2	Fine	An image of high quality, providing enjoyable viewing. Interference is not objectionable.
3	Passable	An image of acceptable quality. Interference is not objectionable.
4	Marginal	An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.
5	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
6	Unusable	An image so bad that you could not watch it.

Image compression model

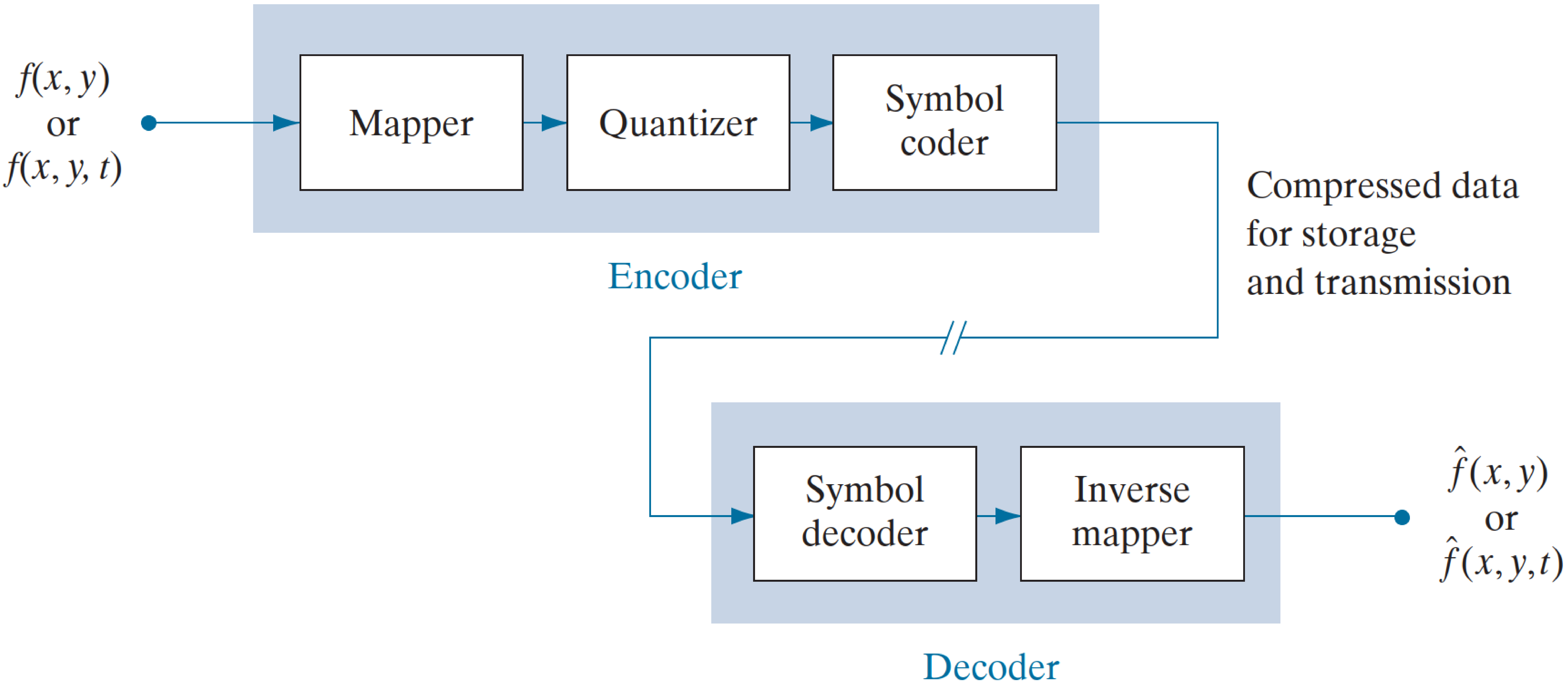


Image compression model

In general, $\hat{f}(x, \ldots)$ may or may not be an exact replica of $f(x, \ldots)$. If it is, the compression system is called **error free**, **lossless**, or **information preserving**. If not, the reconstructed output image is distorted, and the compression system is referred to as **lossy**.

The **encoder** is designed to remove the redundancies described in the previous sections through a series of three independent operations. In the first stage of the encoding process, a mapper transforms $f(x, \ldots)$ into a (usually nonvisual) format designed to reduce spatial and temporal redundancy. This operation generally is reversible and may or may not directly reduce the amount of data required to represent the image.

The **quantizer** reduces the accuracy of the mapper's output in accordance with a pre-established fidelity criterion. The goal is to keep irrelevant information out of the compressed representation. *This operation is irreversible.* It must be omitted when error-free compression is desired.

Image compression model

The **symbol coder** generates a fixed-length or variable-length code to represent the quantizer output, and maps the output in accordance with the code. In many cases, a variable-length code is used. The shortest code words are assigned to the most frequently occurring quantizer output values, thus minimizing coding redundancy. This operation is reversible.

Upon its completion, the input image has been processed for the removal of each of the three redundancies described in the previous sections.

The **decoder** contains only two components: a symbol decoder and an inverse mapper.

They perform, in reverse order, the inverse operations of the encoder’s symbol encoder and mapper.

Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general decoder model.

Image formats, containers, and compression standards

An **image file format** is a standard way to organize and store image data. It defines how the data is arranged and the type of compression (if any) that is used.

An **image container** is similar to a file format, but handles multiple types of image data.

Image compression standards, on the other hand, define procedures for compressing and decompressing images—that is, for reducing the amount of data needed to represent an image.

Image formats, containers, and compression standards

FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in blue; all others are in black.

Image Compression Standards, Formats, and Containers

- **Still Image**
  - **Binary**
    - CCITT Group 3
    - CCITT Group 4
    - JBIG (or JBIG1)
    - JBIG2
    - TIFF
  - **Continuous Tone**
    - JPEG
    - JPEG-LS
    - JPEG-2000
    - BMP
    - GIF
    - PDF
    - PNG
    - TIFF
    - WebP
- **Video**
  - DV
  - H.261
  - H.262
  - H.263
  - H.264
  - HVEC/H.265
  - MPEG-1
  - MPEG-2
  - MPEG-4
  - MPEG-4 AVC
  - AVS
  - HDV
  - M-JPEG
  - QuickTime
  - VC-1 (or WMV9)
  - WebP VP8

Image formats, containers, and compression standards

TABLE 8.3
Internationally sanctioned image compression standards. The numbers in brackets refer to sections in this chapter.

Name	Organization	Description
Bi-Level Still Images		
CCITT Group 3	ITU-T	Designed as a facsimile (FAX) method for transmitting binary documents over telephone lines. Supports 1-D and 2-D run-length [8.6] and Huffman [8.2] coding.
CCITT Group 4	ITU-T	A simplified and streamlined version of the CCITT Group 3 standard supporting 2-D run-length coding only.
JBIG or JBIG1	ISO/IEC/ ITU-T	A *Joint Bi-level Image Experts Group* standard for progressive, lossless compression of bi-level images. Continuous-tone images of up to 6 bits/pixel can be coded on a bit-plane basis [8.8]. Context-sensitive arithmetic coding [8.4] is used and an initial low-resolution version of the image can be gradually enhanced with additional compressed data.
JBIG2	ISO/IEC/ ITU-T	A follow-on to JBIG1 for bi-level images in desktop, Internet, and FAX applications. The compression method used is content based, with dictionary-based methods [8.7] for text and halftone regions, and Huffman [8.2] or arithmetic coding [8.4] for other image content. It can be lossy or lossless.

Image formats, containers, and compression standards

Continuous-Tone Still Images

JPEG	ISO/IEC/ ITU-T	A *Joint Photographic Experts Group* standard for images of photographic quality. Its lossy baseline coding system (most commonly implemented) uses quantized discrete cosine transforms (DCT) on image blocks [8.9], Huffman [8.2], and run-length [8.6] coding. It is one of the most popular methods for compressing images on the Internet.
JPEG-LS	ISO/IEC/ ITU-T	A lossless to near-lossless standard for continuous-tone images based on adaptive prediction [8.10], context modeling [8.4], and Golomb coding [8.3].
JPEG-2000	ISO/IEC/ ITU-T	A follow-on to JPEG for increased compression of photographic quality images. Arithmetic coding [8.4] and quantized discrete wavelet transforms (DWT) [8.11] are used. The compression can be lossy or lossless.

Image formats, containers, and compression standards

Internationally sanctioned video compresssion standards. The numbers in brackets refer to sections in this chapter.

Name	Organization	Description
DV	IEC	*Digital Video*. A video standard tailored to home and semiprofessional video production applications and equipment, such as electronic news gathering and camcorders. Frames are compressed independently for uncomplicated editing using a DCT-based approach [8.9] similar to JPEG.
H.261	ITU-T	A two-way videoconferencing standard for ISDN (*integrated services digital network*) lines. It supports non-interlaced 352×288 and 176×144 resolution images, called CIF (*Common Intermediate Format*) and QCIF (*Quarter CIF*), respectively. A DCT-based compression approach [8.9] similar to JPEG is used, with frame-to-frame prediction differencing [8.10] to reduce temporal redundancy. A block-based technique is used to compensate for motion between frames.
H.262	ITU-T	See MPEG-2 below.
H.263	ITU-T	An enhanced version of H.261 designed for ordinary telephone modems (i.e., 28.8 Kb/s) with additional resolutions: SQCIF (*Sub-Quarter* CIF 128×96), 4CIF (704×576) and 16CIF (1408×512).
H.264	ITU-T	An extension of H.261–H.263 for videoconferencing, streaming, and television. It supports prediction differences within frames [8.10], variable block size integer transforms (rather than the DCT), and context adaptive arithmetic coding [8.4].
H.265 MPEG-H HEVC	ISO/IEC ITU-T	*High Efficiency Video Coding* (HVEC). An extension of H.264 that includes support for macroblock sizes up to 64×64 and additional intraframe prediction modes, both useful in 4K video applications.

Image formats, containers, and compression standards

Internationally sanctioned video compresssion standards. The numbers in brackets refer to sections in this chapter.

Name	Organization	Description
DV	IEC	*Digital Video*. A video standard tailored to home and semiprofessional video production applications and equipment, such as electronic news gathering and camcorders. Frames are compressed independently for uncomplicated editing using a DCT-based approach [8.9] similar to JPEG.
H.261	ITU-T	A two-way videoconferencing standard for ISDN (*integrated services digital network*) lines. It supports non-interlaced 352 × 288 and 176 × 144 resolution images, called CIF (*Common Intermediate Format*) and QCIF (*Quarter CIF*), respectively. A DCT-based compression approach [8.9] similar to JPEG is used, with frame-to-frame prediction differencing [8.10] to reduce temporal redundancy. A block-based technique is used to compensate for motion between frames.
H.262	ITU-T	See MPEG-2 below.
H.263	ITU-T	An enhanced version of H.261 designed for ordinary telephone modems (i.e., 28.8 Kb/s) with additional resolutions: SQCIF (*Sub-Quarter* CIF 128 × 96), 4CIF (704 × 576) and 16CIF (1408 × 512).

Image formats, containers, and compression standards

H.264	ITU-T	An extension of H.261–H.263 for videoconferencing, streaming, and television. It supports prediction differences within frames [8.10], variable block size integer transforms (rather than the DCT), and context adaptive arithmetic coding [8.4].
H.265 MPEG-H HEVC	ISO/IEC ITU-T	*High Efficiency Video Coding* (HVEC). An extension of H.264 that includes support for macroblock sizes up to 64×64 and additional intraframe prediction modes, both useful in 4K video applications.
MPEG-1	ISO/IEC	A *Motion Pictures Expert Group* standard for CD-ROM applications with non-interlaced video at up to 1.5 Mb/s. It is similar to H.261 but frame predictions can be based on the previous frame, next frame, or an interpolation of both. It is supported by almost all computers and DVD players.
MPEG-2	ISO/IEC	An extension of MPEG-1 designed for DVDs with transfer rates at up to 15 Mb/s. Supports interlaced video and HDTV. It is the most successful video standard to date.
MPEG-4	ISO/IEC	An extension of MPEG-2 that supports variable block sizes and prediction differencing [8.10] within frames.
MPEG-4 AVC	ISO/IEC	MPEG-4 Part 10 *Advanced Video Coding* (AVC). Identical to H.264.

Image formats, containers, and compression standards

Name	Organization	Description
Continuous-Tone Still Images		
BMP	Microsoft	*Windows Bitmap*. A file format used mainly for simple uncompressed images.
GIF	CompuServe	*Graphic Interchange Format*. A file format that uses lossless LZW coding [8.5] for 1- through 8-bit images. It is frequently used to make small animations and short low-resolution films for the Internet.
PDF	Adobe Systems	*Portable Document Format*. A format for representing 2-D documents in a device and resolution independent way. It can function as a container for JPEG, JPEG-2000, CCITT, and other compressed images. Some PDF versions have become ISO standards.
PNG	*World Wide Web Consortium* (W3C)	*Portable Network Graphics*. A file format that losslessly compresses full color images with transparency (up to 48 bits/pixel) by coding the difference between each pixel's value and a predicted value based on past pixels [8.10].
TIFF	Aldus	*Tagged Image File Format*. A flexible file format supporting a variety of image compression standards, including JPEG, JPEG-LS, JPEG-2000, JBIG2, and others.
WebP	Google	*WebP* supports lossy compression via WebP VP8 intraframe video compression (see below) and lossless compression using spatial prediction [8.10] and a variant of LZW backward referencing [8.5] and Huffman entropy coding [8.2]. Transparency is also supported.

Image formats, containers, and compression standards

Video

AVS	MII	*Audio-Video Standard*. Similar to H.264 but uses exponential Golomb coding [8.3]. Developed in China.
HDV	Company consortium	*High Definition Video*. An extension of DV for HD television that uses compression similar to MPEG-2, including temporal redundancy removal by prediction differencing [8.10].
M-JPEG	Various companies	*Motion JPEG*. A compression format in which each frame is compressed independently using JPEG.
Quick-Time	Apple Computer	A media container supporting DV, H.261, H.262, H.264, MPEG-1, MPEG-2, MPEG-4, and other video compression formats.
VC-1 WMV9	SMPTE Microsoft	The most used video format on the Internet. Adopted for HD and *Blu-ray* high-definition DVDs. It is similar to H.264/AVC, using an integer DCT with varying block sizes [8.9 and 8.10] and context-dependent variable-length code tables [8.2], but no predictions within frames.
WebP VP8	Google	A file format based on block transform coding [8.9] prediction differences within frames and between frames [8.10]. The differences are entropy encoded using an adaptive arithmetic coder [8.4].

Huffman coding

One of the most popular techniques for removing coding redundancy is due to Huffman.

When coding the symbols of an information source individually, Huffman coding yields the smallest possible number of code symbols per source symbol.

In terms of Shannon's first theorem, the resulting code is optimal for a fixed value of n, subject to the constraint that the source symbols be coded one at a time.

In practice, the source symbols may be either the intensities of an image or the output of an intensity mapping operation (pixel differences, run lengths, and so on).

Huffman coding

The first step in Huffman’s approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration, then combining the lowest probability symbols into a single symbol that replaces them in the next source reduction.

FIGURE 8.7
Huffman source reductions.

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Huffman coding

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source.

FIGURE 8.8
Huffman code assignment procedure.

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$$
$$= 2.2 \text{ bits/pixel}$$

Huffman coding

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time.

After the code has been created, coding and/or error-free decoding is accomplished in a simple lookup table manner.

The code itself is an **instantaneous uniquely decodable block code**.

It is called a **block code** because each source symbol is mapped into a fixed sequence of code symbols.

It is **instantaneous** because each code word in a string of code symbols can be decoded without referencing succeeding symbols.

It is **uniquely decodable** because any string of code symbols can be decoded in only one way.

Huffman coding

When a large number of symbols is to be coded, the construction of an optimal Huffman code is a nontrivial task. For the general case of J source symbols, J symbol probabilities, $J - 2$ source reductions, and $J - 2$ code assignments are required.

When source symbol probabilities can be estimated in advance, “near optimal” coding can be achieved with *pre-computed Huffman codes*.

Several popular image compression standards, including the JPEG and MPEG standards specify default Huffman coding tables that have been pre-computed based on experimental data.

Arithmetic coding

Unlike the variable-length codes as Huffman code, **arithmetic coding** generates **nonblock codes**.

In arithmetic coding, a one-to-one correspondence between source symbols and code words does not exist.

Instead, an entire sequence of source symbols is assigned a single arithmetic code word.

The code word itself defines an interval of real numbers between 0 and 1.

As the number of symbols in the message increases, the interval used to represent it becomes smaller, and the number of bits required to represent the interval becomes larger.

Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence.

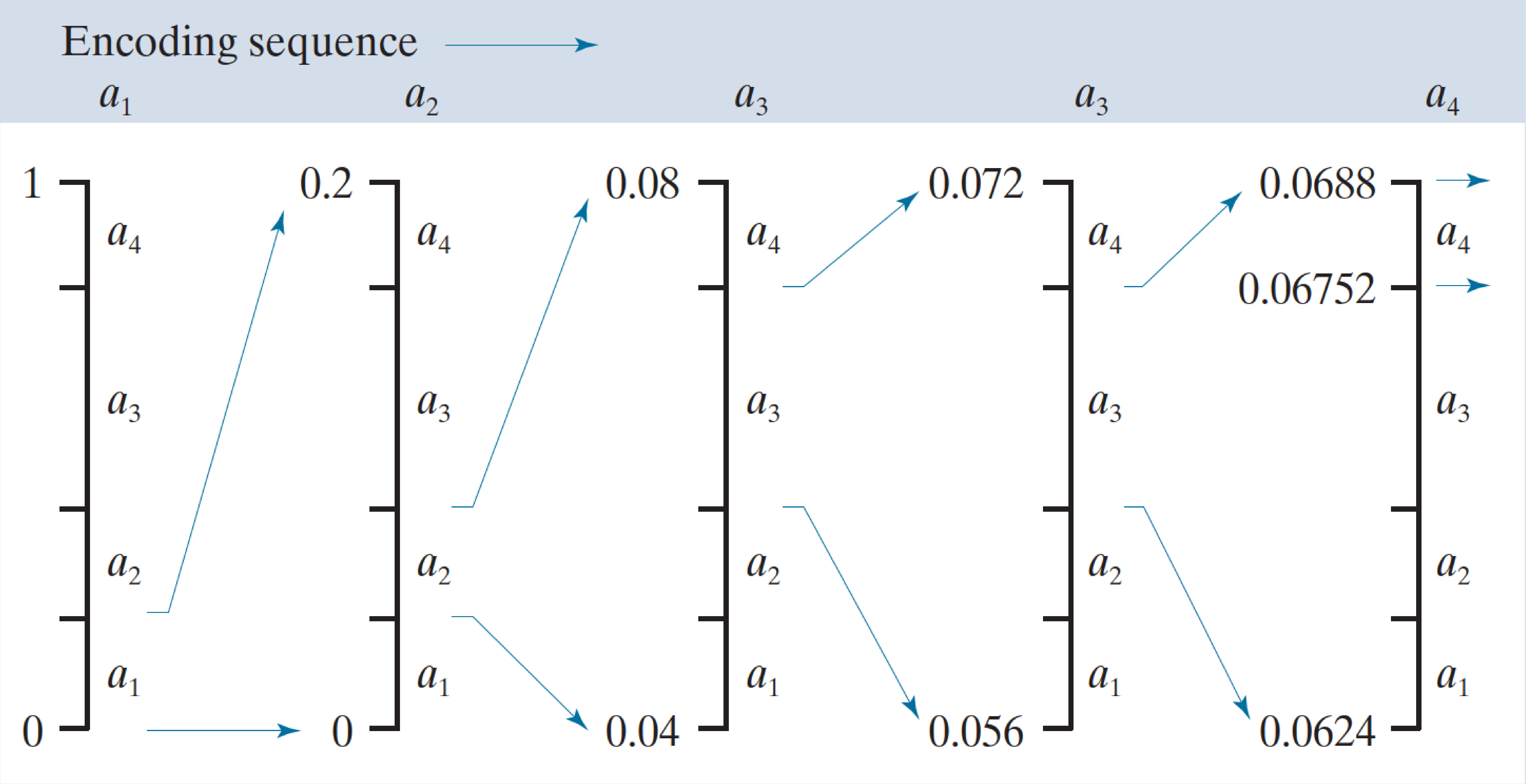
Because the technique does not require that the symbols be coded one at a time it achieves (but only in theory) the bound established by Shannon's first theorem

Arithmetic coding

TABLE 8.7
Arithmetic coding example.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

FIGURE 8.12
Arithmetic coding procedure.



Arithmetic coding

In the arithmetically coded message of the last example, three decimal digits are used to represent the five-symbol message.

This translates into 0.6 decimal digits per source symbol and compares favorably with the entropy of the source, which is 0.58 decimal digits per source symbol.

As the length of the sequence being coded increases, the resulting arithmetic code approaches the bound established by Shannon's first theorem.

In practice, two factors cause coding performance to fall short of the bound:

(1) the addition of the end-of-message indicator that is needed to separate one message from another, and

(2) the use of finite precision arithmetic.

Practical implementations of arithmetic coding address the latter problem by introducing a **scaling strategy** and a **rounding strategy.**

Arithmetic coding

With accurate input symbol probability models, i.e., models that provide the true probabilities of the symbols being coded, arithmetic coders are near optimal.

However, inaccurate probability models can lead to non-optimal results.

A simple way to improve the accuracy of the probabilities employed is to use an *adaptive, context dependent probability model*.

Adaptive probability models update symbol probabilities as symbols are coded or become known.

Thus, the probabilities adapt to the local statistics of the symbols being coded.

Context-dependent models provide probabilities that are based on a predefined neighborhood of pixels, called the **context**, around the symbols being coded.

Normally, a causal context (one limited to symbols that have already been coded) is used.

Arithmetic coding

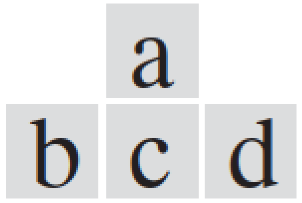
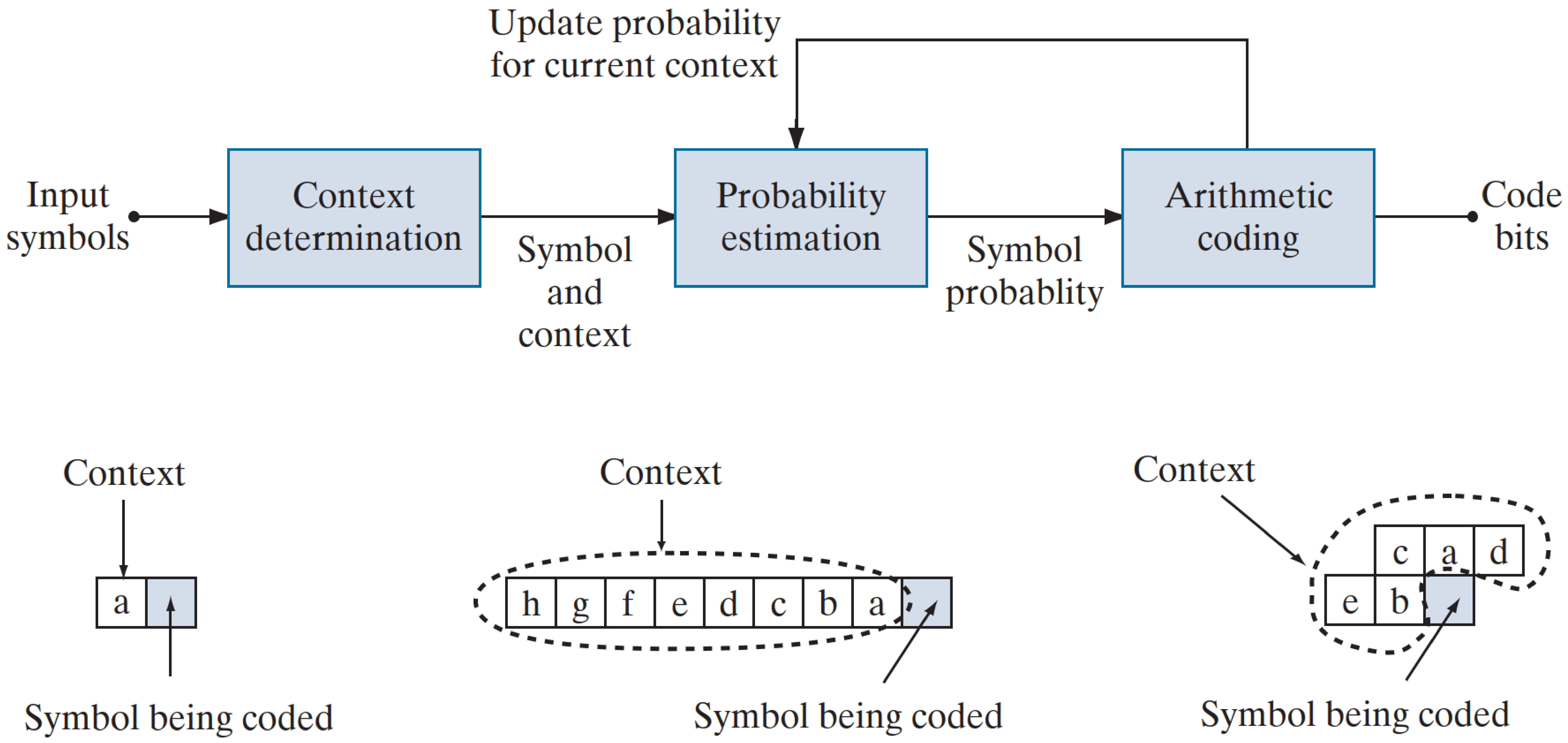


FIGURE 8.13
(a) An adaptive, context-based arithmetic coding approach (often used for binary source symbols). (b)–(d) Three possible context models.



Arithmetic coding

As each symbol (or bit) begins the coding process, its context is formed in the **Context determination** block.

Three possible contexts that can be used: (1) the immediately preceding symbol, (2) a group of preceding symbols, and (3) some number of preceding symbols plus symbols on the previous scan line.

For the three cases, the **Probability estimation** block must manage 2^1 (or 2), 2^8 (or 256), and 2^5 (or 32) contexts and their associated probabilities.

For instance, if the first context is used, conditional probabilities $P(0|a = 0)$, $P(1|a = 0)$, $P(0|a = 1)$, and $P(1|a = 1)$ must be tracked.

The appropriate probabilities are then passed to the **Arithmetic coding** block as a function of the current context, and drive the generation of the arithmetically coded output sequence.

The probabilities associated with the context involved in the current coding step are then updated to reflect the fact that another symbol within that context has been processed.

Run-length coding

Images with repeating intensities along their rows can often be compressed by representing runs of identical intensities as run-length pairs, where each run-length pair specifies the start of a new intensity and the number of consecutive pixels that have that intensity.

The technique, referred to as run-length encoding (RLE), was developed in the 1950s and became, along with its 2-D extensions, the standard compression approach in facsimile (FAX) coding.

Compression is achieved by eliminating a simple form of spatial redundancy—groups of identical intensities.

When there are few (or no) runs of identical pixels, run-length encoding results in data expansion.

RLE in the BMP file format

The BMP file format uses a form of run-length encoding in which image data is represented in two different modes: **encoded** and **absolute**.

In **encoded mode**, a two byte RLE representation is used. The first byte specifies the number of consecutive pixels that have the color index contained in the second byte. The 8-bit color index selects the run's intensity (color or gray value) from a table of 256 possible intensities.

In **absolute mode**, the first byte is 0, and the second byte signals one of four possible conditions:

Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3-255	Specify pixels individually

Run-length coding

Run-length encoding is particularly effective when compressing binary images.

Because there are only two possible intensities (black and white), adjacent pixels are more likely to be identical.

In addition, each image row can be represented by a sequence of lengths only, rather than length-intensity pairs.

The basic idea is to code each contiguous group (i.e., run) of 0's or 1's encountered in a left-to-right scan of a row by its length and to establish a convention for determining the value of the run. The most common conventions are

(1) to specify the value of the first run of each row, or

(2) to assume that each row begins with a white run, whose run length may in fact be zero.

Additional compression can be achieved by variable-length coding the run lengths themselves.

Run-length coding

Two of the oldest and most widely used image compression standards are the **CCITT Group 3 and 4** standards for binary image compression.

They were originally designed as facsimile (FAX) coding methods for transmitting documents over telephone networks.

The Group 3 standard uses a 1-D run-length coding technique in which the last K - 1 lines of each group of K lines (for K = 2 or 4) can be optionally coded in a 2-D manner.

The Group 4 standard is a simplified or streamlined version of the Group 3 standard in which only 2-D coding is allowed.

Both standards use the same **2-D** coding approach, which is two-dimensional in the sense that *information from the previous line is used to encode the current line.*

Symbol-based coding

In **symbol-** or **token-based coding**, an image is represented as a collection of frequently occurring subimages, called **symbols**.

Each such symbol is stored in a **symbol dictionary** and the image is coded as a set of **triplets** $\{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots\}$, where each (x_i, y_i) pair specifies the location of a symbol in the image and token t_i is the address of the symbol or subimage in the dictionary.

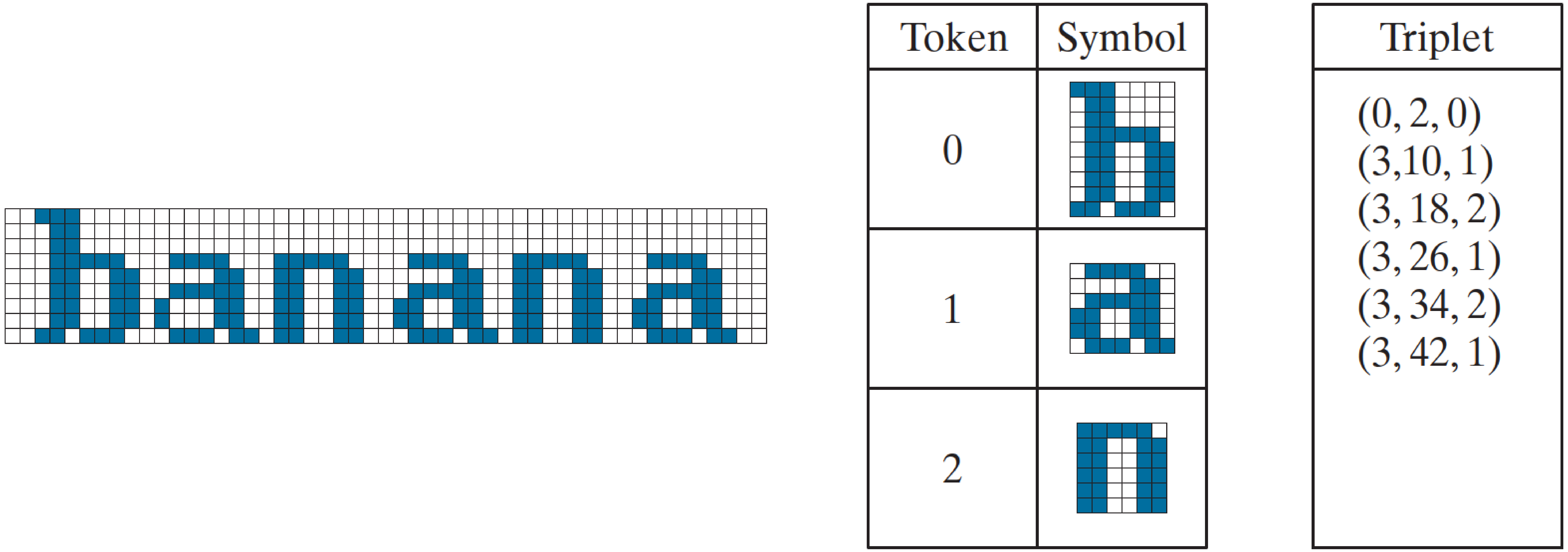
That is, each triplet represents an instance of a dictionary symbol in the image.

Storing repeated symbols only once can compress images significantly, particularly in document storage and retrieval applications where the symbols are often *character bitmaps* that are repeated many times.

Symbol-based coding

a b c

FIGURE 8.17
(a) A bi-level document, (b) symbol dictionary, and (c) the triplets used to locate the symbols in the document.



The starting image has 9 × 51 × 1 or 459 bits and, assuming that each triplet is composed of three bytes, the compressed representation has (6 × 3 × 8) + [(9 × 7) + (6 × 7) + (6 × 6)] or 285 bits; the resulting compression ratio C = 1.61.

Bit-plane coding

The run-length and symbol-based techniques can be applied to images with more than two intensities by individually processing their bit planes.

The technique, called bit-plane coding, is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images and compressing each binary image via one of several well-known binary compression methods.

The intensities of an *m*-bit monochrome image can be represented in the form of the base-2 polynomial

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \ldots + a_1 2^1 + a_0 2^0$$

A simple method of decomposing the image into a collection of binary images is to separate the *m* coefficients of the polynomial into *m* 1-bit bit planes.

The inherent disadvantage of this decomposition approach is that small changes in intensity can have a significant impact on the complexity of the bit planes.

Bit-plane coding

An alternative decomposition approach is to first represent the image by an *m*-bit **Gray code**.

The *m*-bit Gray code $g_{m-1} \ldots g_2\, g_1 g_0$ that corresponds to the polynomial can be computed from

$$g_i = a_i \oplus a_{i+1} \qquad 0 \le i \le m-2$$
$$g_{m-1} = a_{m-1}$$

Here, $\oplus$ denotes the exclusive OR operation.

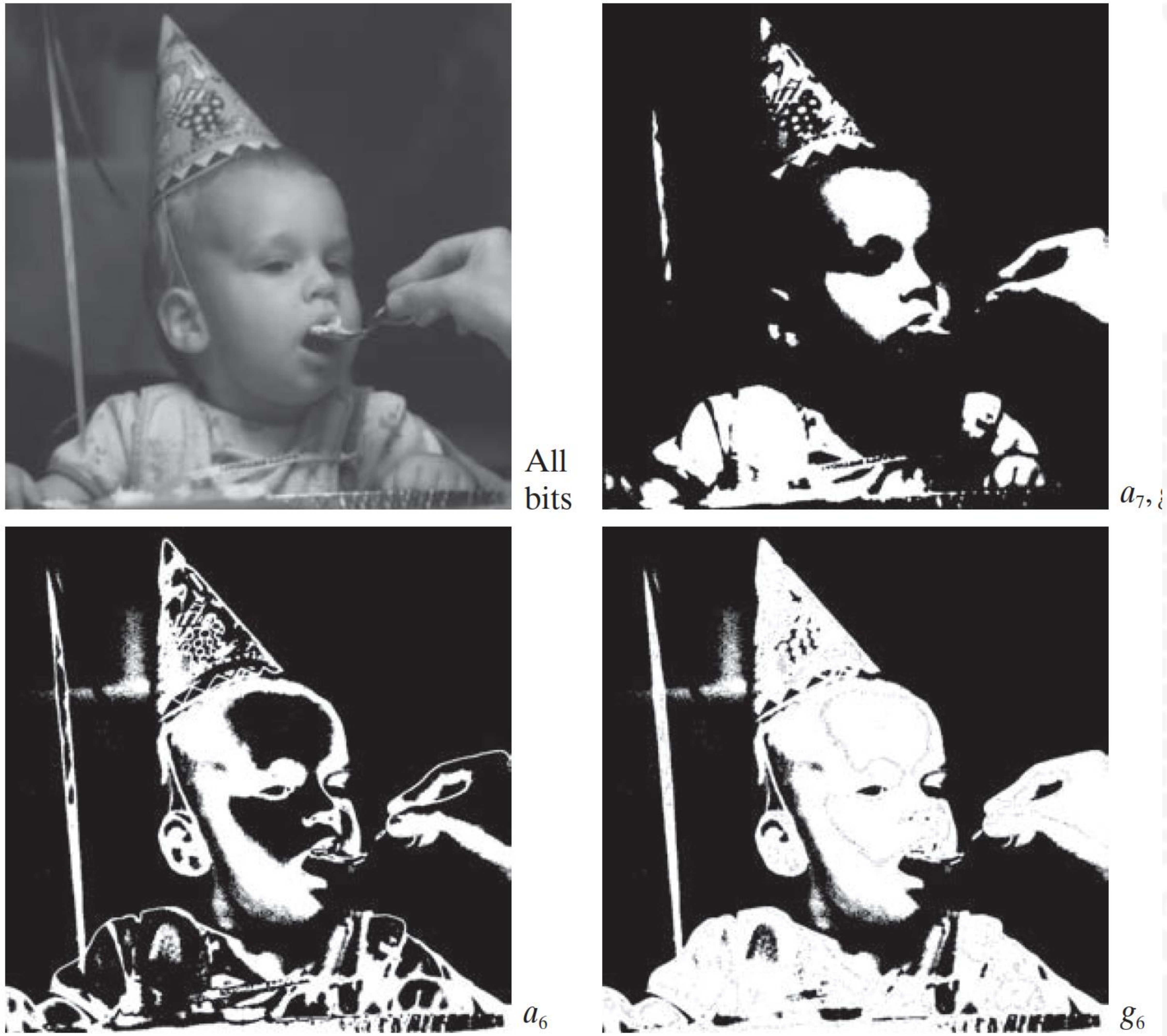
This code has the unique property that successive code words differ in only one bit position.

Small changes in intensity are less likely to affect all *m* bit planes.

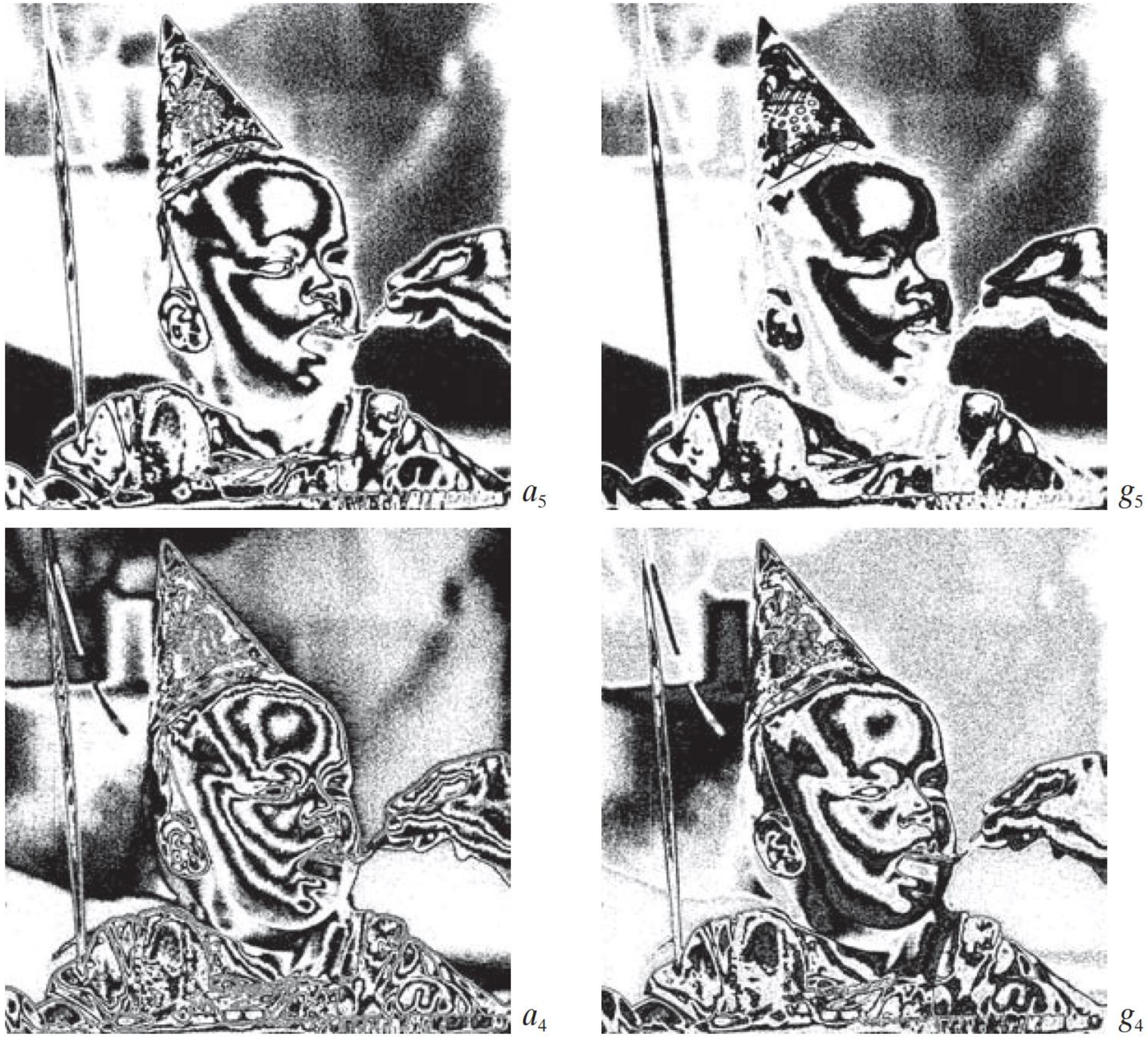
Bit-plane coding

a	b
c	d
e	f
g	h

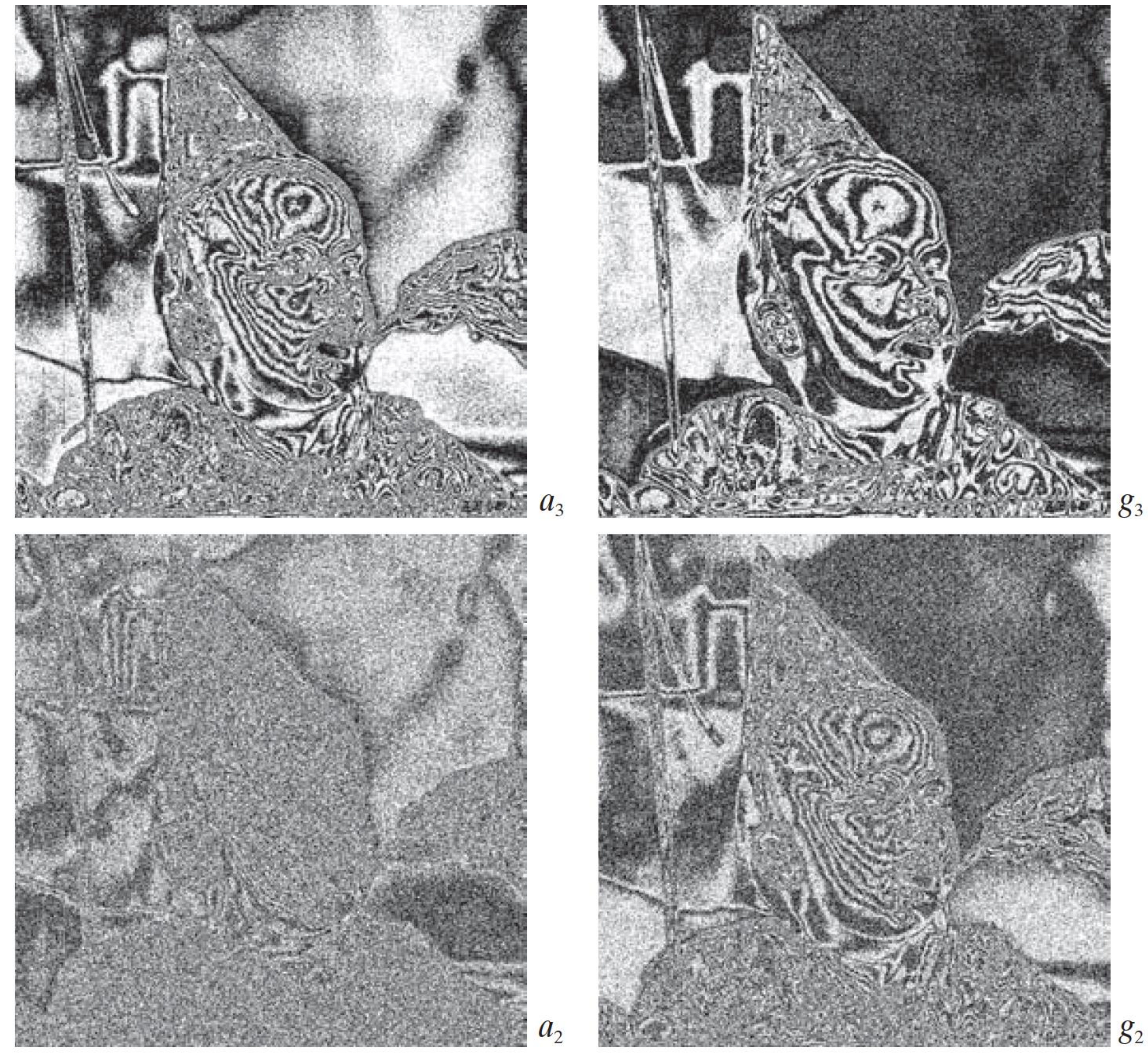
FIGURE 8.19
(a) A 256-bit monochrome image.
(b)–(h) The four most significant binary and Gray-coded bit planes of the image in (a).



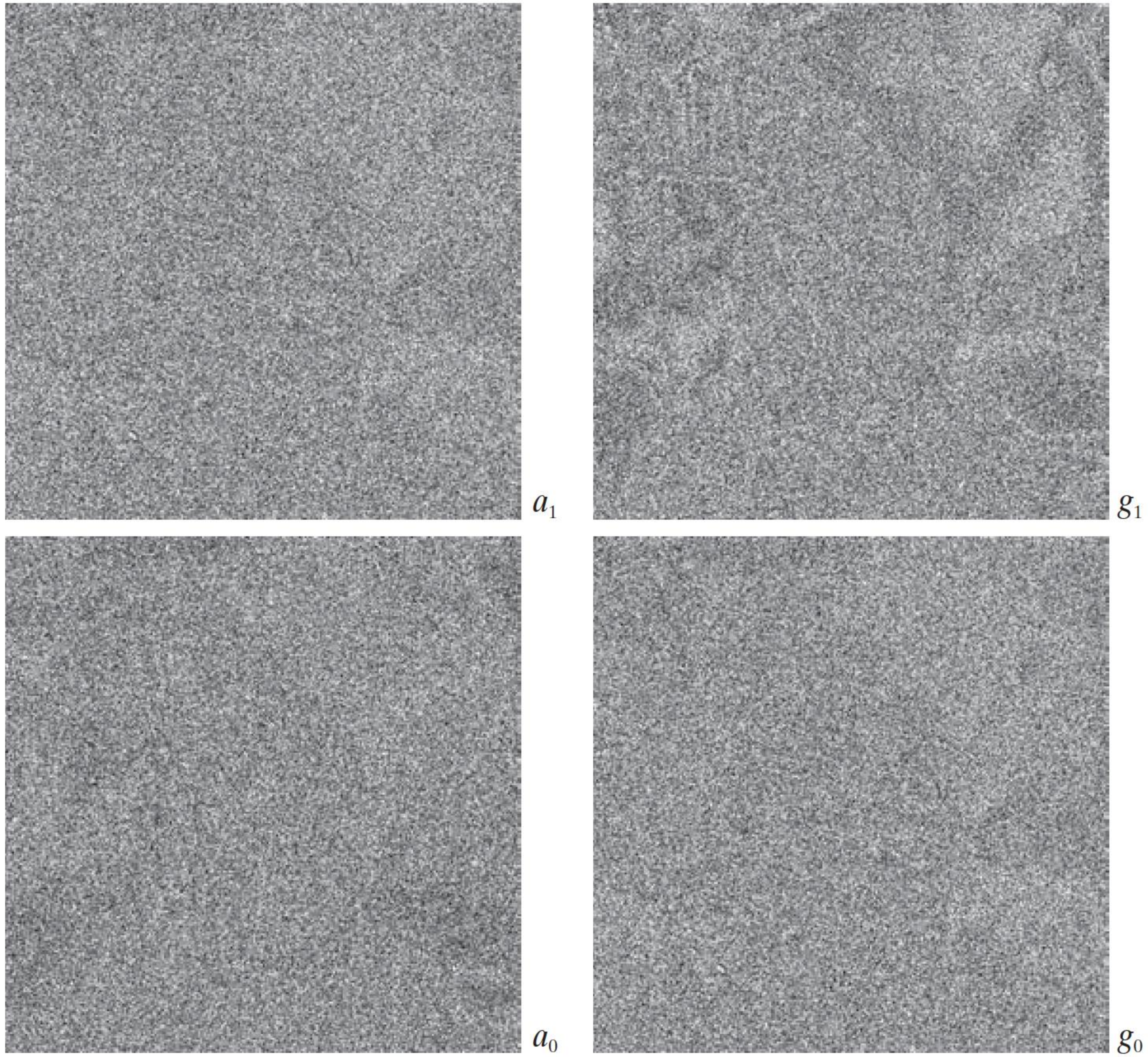
Bit-plane coding



Bit-plane coding



Bit-plane coding



Bit-plane coding

TABLE 8.11
JBIG2 lossless coding results for the binary and Gray-coded bit planes of Fig. 8.19(a). These results include the overhead of each bit plane's PDF representation.

Coefficient *m*	Binary Code (PDF bits)	Gray Code (PDF bits)	Compression Ratio
7	6,999	6,999	1.00
6	12,791	11,024	1.16
5	40,104	36,914	1.09
4	55,911	47,415	1.18
3	78,915	67,787	1.16
2	101,535	92,630	1.10
1	107,909	105,286	1.03
0	99,753	107,909	0.92

Block transform coding

Is a compression technique that divides an image into small non-overlapping blocks of equal size (e.g., 8 * 8) and processes the blocks independently using a 2-D transform.

In **block transform coding**, a reversible, linear transform (such as the Fourier transform) is used to map each block or subimage into a set of transform coefficients, which are then quantized and coded.

For most images, a significant number of the coefficients have small magnitudes and can be coarsely quantized (or discarded entirely) with little image distortion.

A variety of transformations, including the DFT, can be used to transform the image data.

Block transform coding

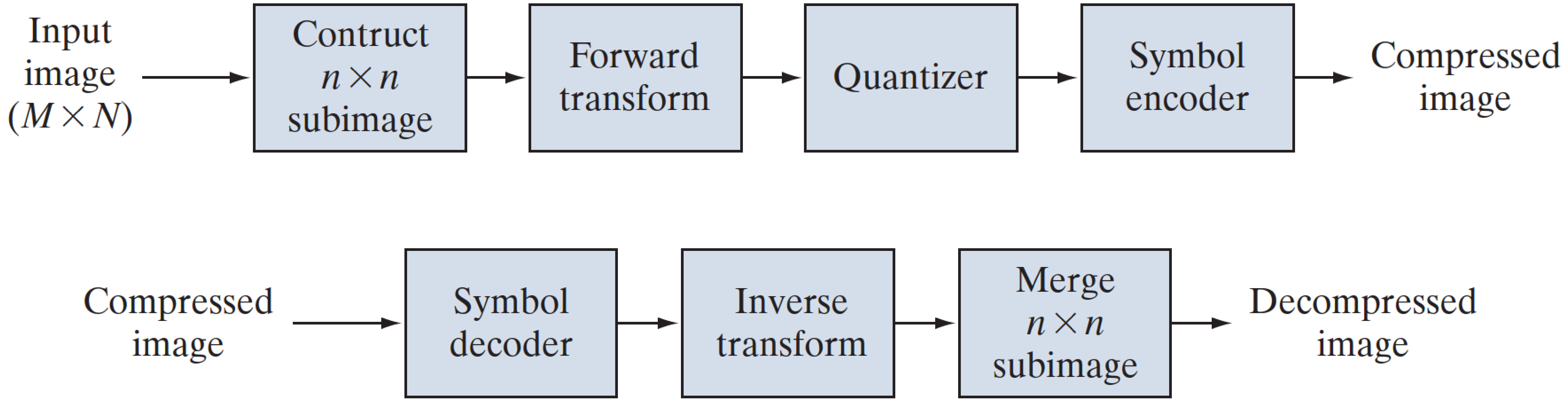


FIGURE 8.21
A block transform coding system: (a) encoder; (b) decoder.

Block transform coding

An $M \times N$ input image is subdivided first into subimages of size $n \times n$, which are then transformed to generate MN/n^2 subimage transform arrays, each of size $n \times n$.

The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients.

The quantization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least amount of information in a predefined sense.

These coefficients have the smallest impact on reconstructed subimage quality.

The encoding process terminates by coding (normally using a variable-length code) the quantized coefficients.

Any or all of the transform encoding steps can be adapted to local image content, called **adaptive transform coding**, or fixed for all subimages, called **nonadaptive transform coding**.

Transform selection

Block transform coding systems based on a variety of discrete 2-D transforms have been constructed and/or studied extensively.

The choice of a particular transform in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available. **Compression is achieved during the quantization** of the transformed coefficients (not during the transformation step).

Transform selection

Removing 50% of the transformed coefficients….
The actual rms errors were 2.32, 1.78, and 1.13 intensities, respectively.

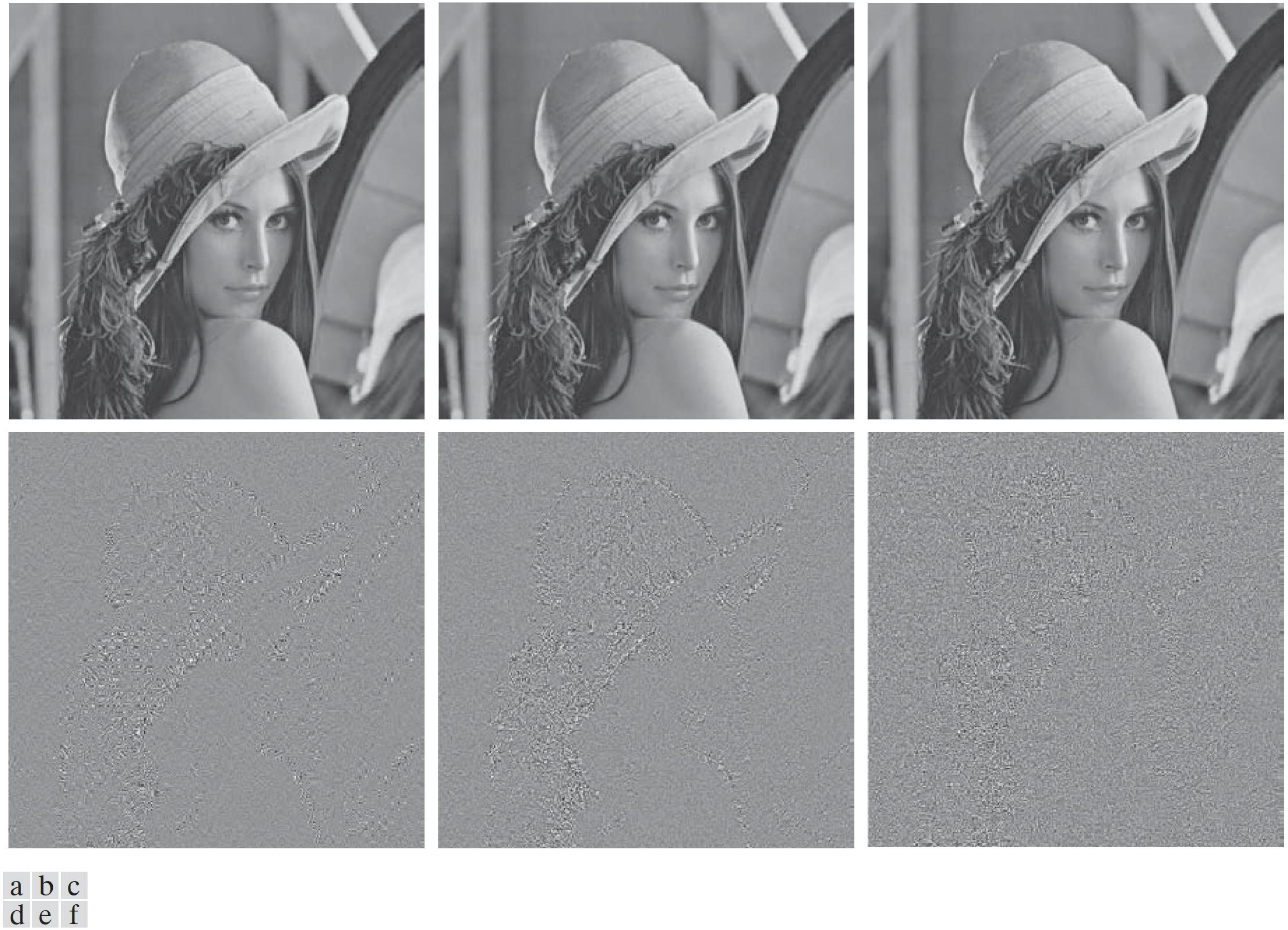


FIGURE 8.22 Approximations of Fig. 8.9(a) using the (a) Fourier, (b) Walsh-Hadamard, and (c) cosine transforms, together with the corresponding scaled error images in (d)–(f).

Transform selection

The small differences in mean-squared reconstruction error are related directly to the energy or information packing properties of the transforms employed.

An $n \times n$ subimage $g(x, y)$ can be expressed as a function of its 2-D transform $T(u,v)$:

$$\mathbf{G} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v)\mathbf{S}_{uv}$$

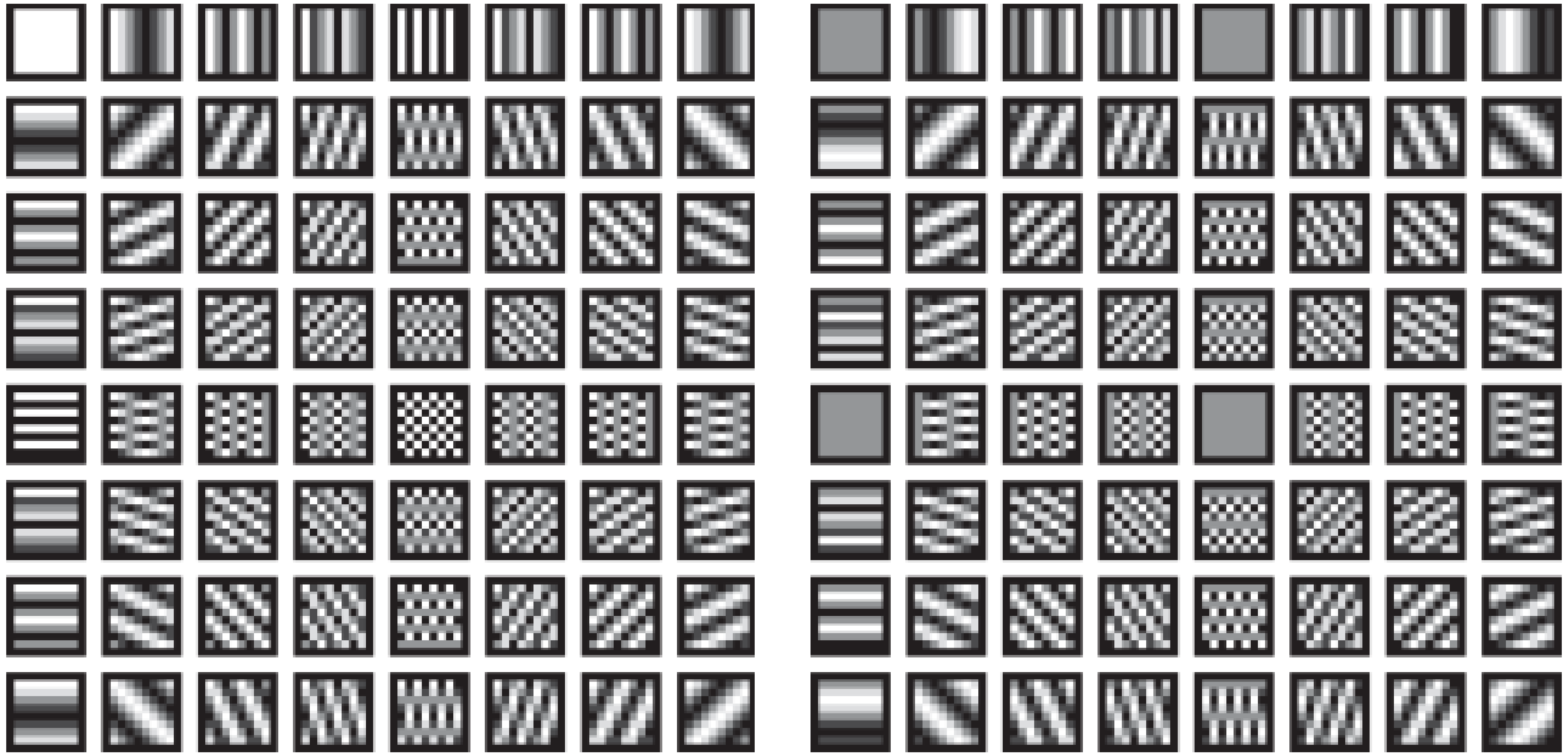
G, the matrix containing the pixels of the input subimage, is explicitly defined as a linear combination of n^2 basis images of size $n \times n$. In fact,

$$g(x,y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v)s(u,v,x,y)$$

Where $s(u,v,x,y)$ are the inverse transform coefficients. For any u, v pair, varying x,y we obtain a basis image.

Transform selection

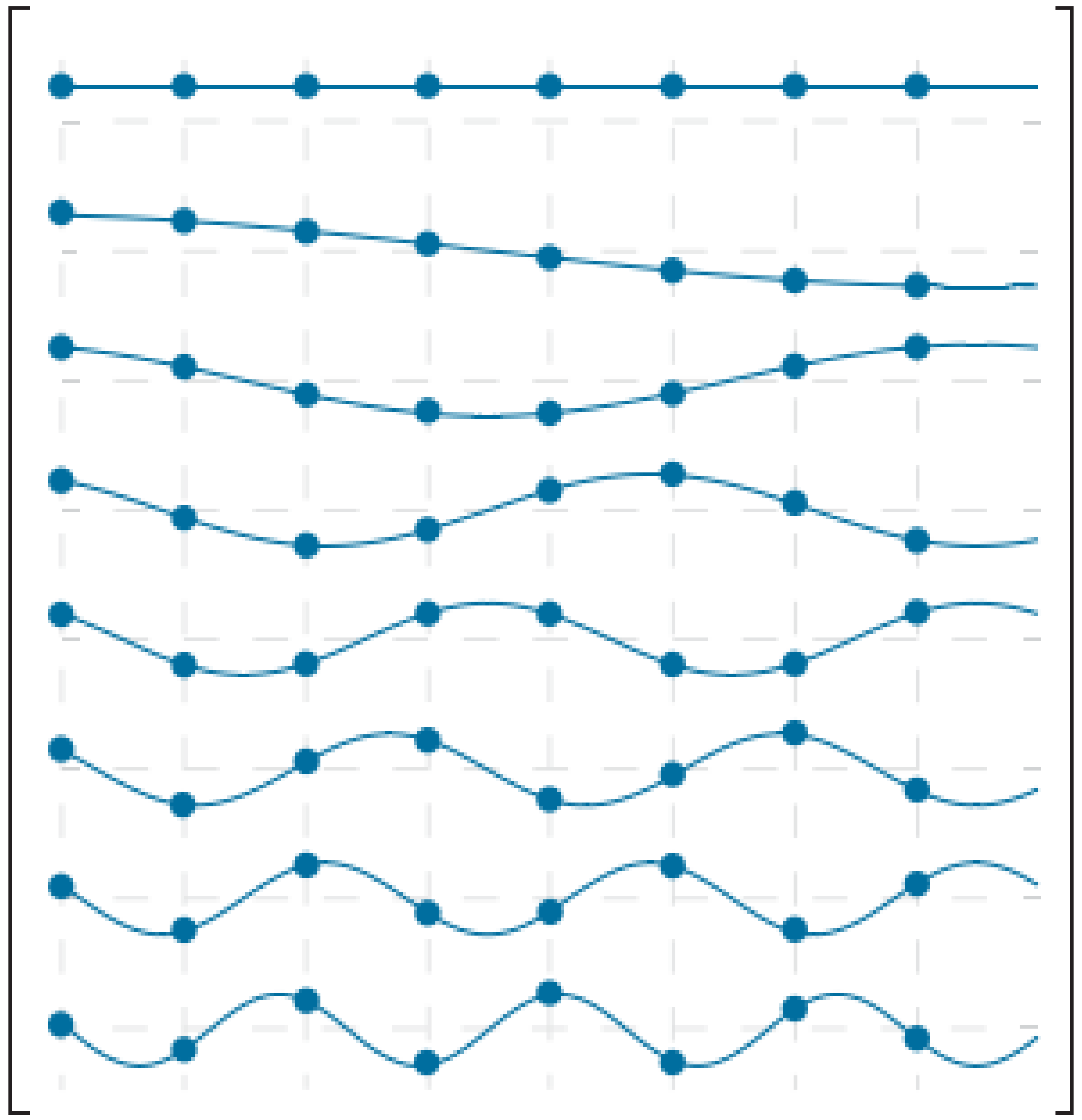
$$
\frac{1}{\sqrt{8}}\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & \omega & -j & -j\omega & -1 & -\omega & j & j\omega \\
1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & -j\omega & j & \omega & -1 & j\omega & -j & -\omega \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -\omega & -j & j\omega & -1 & \omega & j & -j\omega \\
1 & j & -1 & -j & 1 & j & -1 & -j \\
1 & j\omega & j & -\omega & -1 & -j\omega & -j & \omega
\end{bmatrix}
$$



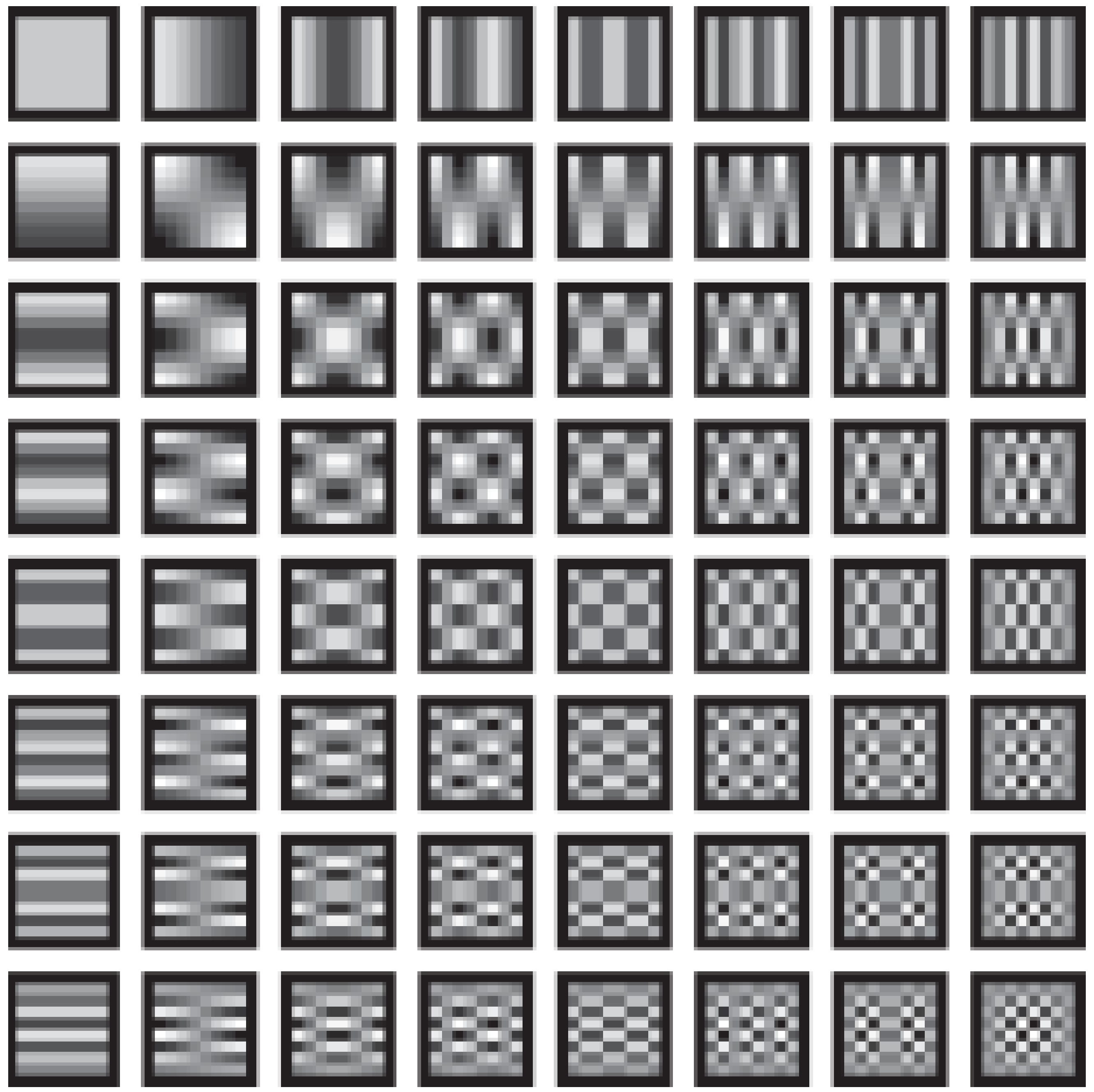
a b c

FIGURE 7.7 (a) Tranformation matrix $\mathbf{A}_F$ of the discrete Fourier transform for $N = 8$, where $\omega = e^{-j2\pi/8}$ or $(1 - j)/\sqrt{2}$. (b) and (c) The real and imaginary parts of the DFT basis images of size 8×8. For clarity, a black border has been added around each basis image. For 1-D transforms, matrix $\mathbf{A}_F$ is used in conjunction with Eqs. (7-43) and (7-44); for 2-D transforms, it is used with Eqs. (7-41) and (7-42).

Transform selection



0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35
0.49	0.42	0.28	0.10	−0.10	−0.28	−0.42	−0.49
0.46	0.19	−0.19	−0.46	−0.46	−0.19	0.19	0.46
0.42	−0.10	−0.49	−0.28	0.28	0.49	0.10	−0.42
0.35	−0.35	−0.35	0.35	0.35	−0.35	−0.35	0.35
0.28	−0.49	0.10	0.42	−0.42	−0.10	0.49	−0.28
0.19	−0.46	0.46	−0.19	−0.19	0.46	−0.46	0.19
0.10	−0.28	0.42	−0.49	0.49	−0.42	0.28	−0.10



a b c

FIGURE 7.10 The transformation matrix and basis images of the discrete cosine transform for $N = 8$. (a) Graphical representation of orthogonal transformation matrix $\mathbf{A}_C$, (b) $\mathbf{A}_C$ rounded to two decimal places, and (c) basis images. For 1-D transforms, matrix $\mathbf{A}_C$ is used in conjunction with Eqs. (7-28) and (7-29); for 2-D transforms, it is used with Eqs. (7-35) and (7-36).

Walsh Hadamard transform

Walsh-Hadamard transforms (WHTs) are non-sinusoidal transformations that decompose a function into a linear combination of rectangular basis functions, called **Walsh functions**, of value +1 and −1.

The ordering of the basis functions within a Walsh-Hadamard transformation matrix determines the variant of the transform that is being computed.

For **Hadamard ordering** (also called **natural ordering**), for *N=2*

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

where the matrix on the right (without the scalar multiplier)

is called a **Hadamard matrix** of order 2. Letting $\mathbf{H}_N$ denote the Hadamard matrix of order N, a simple recursive relationship for generating Hadamard-ordered transformation matrices is

$$\mathbf{A}_W = \frac{1}{\sqrt{N}}\mathbf{H}_N \qquad \mathbf{H}_{2N} = \begin{bmatrix} \mathbf{H}_N & \mathbf{H}_N \\ \mathbf{H}_N & -\mathbf{H}_N \end{bmatrix}$$

Walsh Hadamard transform

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\mathbf{H}_4 = \begin{bmatrix} \mathbf{H}_2 & \mathbf{H}_2 \\ \mathbf{H}_2 & -\mathbf{H}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\mathbf{H}_8 = \begin{bmatrix} \mathbf{H}_4 & \mathbf{H}_4 \\ \mathbf{H}_4 & -\mathbf{H}_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

The number of sign changes along a row of a Hadamard matrix is known as the **sequency** of the row.
Like frequency, sequency measures the rate of change of a function, and like the sinusoidal basis functions of the Fourier transform, every Walsh function has a unique sequency.

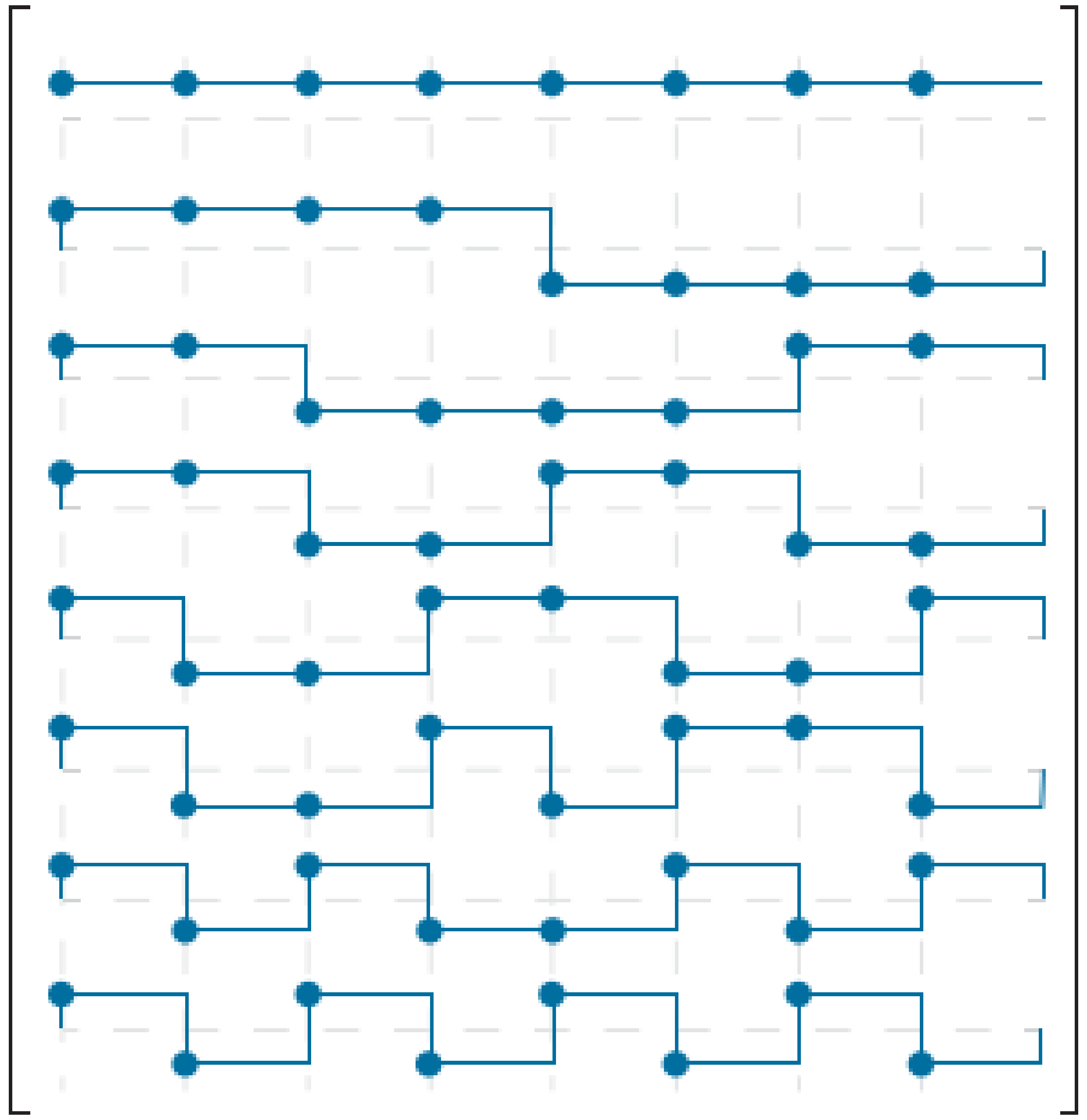
Walsh Hadamard transform

Arranging the basis vectors of a Hadamard matrix so the sequency increases as a function of *u* is both desirable and common in signal and image processing applications.

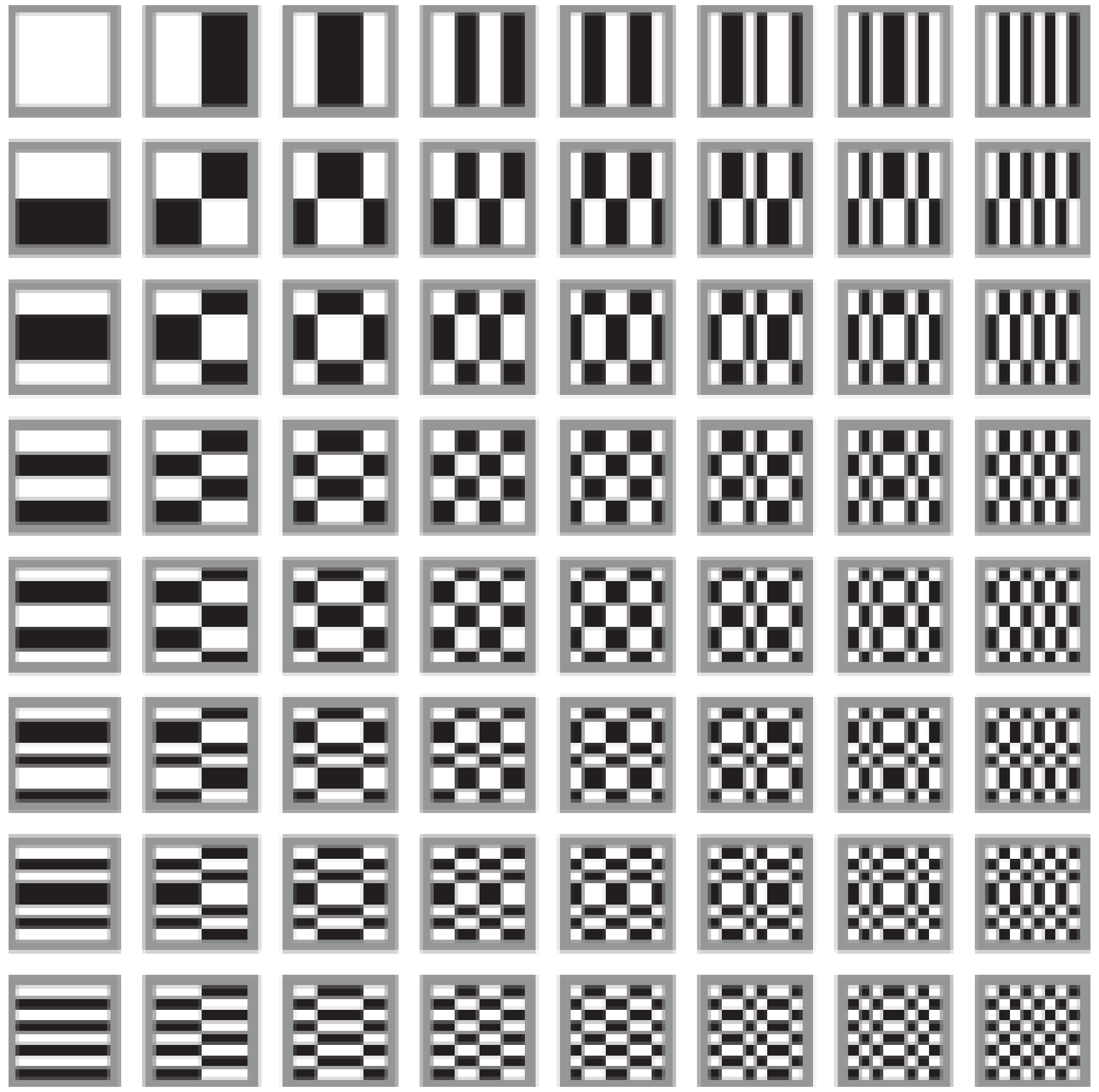
$$\mathbf{H}_8' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{A}_{W'} = \frac{1}{\sqrt{N}}\mathbf{H}_8'$$

Walsh Hadamard transform



$$\begin{bmatrix} 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 \\ 0.35 & 0.35 & 0.35 & 0.35 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.35 & 0.35 & -0.35 & -0.35 & -0.35 & -0.35 & 0.35 & 0.35 \\ 0.35 & 0.35 & -0.35 & -0.35 & 0.35 & 0.35 & -0.35 & -0.35 \\ 0.35 & -0.35 & -0.35 & 0.35 & 0.35 & -0.35 & -0.35 & 0.35 \\ 0.35 & -0.35 & -0.35 & 0.35 & -0.35 & 0.35 & 0.35 & -0.35 \\ 0.35 & -0.35 & 0.35 & -0.35 & -0.35 & 0.35 & -0.35 & 0.35 \\ 0.35 & -0.35 & 0.35 & -0.35 & 0.35 & -0.35 & 0.35 & -0.35 \end{bmatrix}$$



a b c

FIGURE 7.16 The transformation matrix and basis images of the sequency-ordered Walsh-Hadamard transform for $N = 8$. (a) Graphical representation of orthogonal transformation matrix $\mathbf{A}_{W'}$, (b) $\mathbf{A}_{W'}$ rounded to two decimal places, and (c) basis images. For 1-D transforms, matrix $\mathbf{A}_{W'}$ is used in conjunction with Eqs. (7-28) and (7-29); for 2-D transforms, it is used with Eqs. (7-35) and (7-36).

Transform selection

$$\mathbf{G} = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u,v)\mathbf{S}_{uv}$$

If we now define a transform coefficient **masking function**

$$\chi(u,v) = \begin{cases} 0 & \text{if } T(u,v) \text{ satisfies a specified truncation criterion} \\ 1 & \text{otherwise} \end{cases}$$

an approximation of **G** can be obtained from the truncated expansion

$$\hat{\mathbf{G}} = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \chi(u,v)T(u,v)\mathbf{S}_{uv}$$

where *χ(u,v)* is constructed to eliminate the basis images that make the smallest contribution to the total sum.

Transform selection

The mean-squared error between subimage G and approximation Gˆ then is

$$
\begin{aligned}
e_{ms} &= E\left\{\left\|\mathbf{G}-\hat{\mathbf{G}}\right\|^2\right\} \\
&= E\left\{\left\|\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u,v)\mathbf{S}_{uv} - \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \chi(u,v)T(u,v)\mathbf{S}_{uv}\right\|^2\right\} \\
&= E\left\{\left\|\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u,v)\mathbf{S}_{uv}\left[1-\chi(u,v)\right]\right\|^2\right\} \\
&= \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \sigma^2_{T(u,v)}\left[1-\chi(u,v)\right]
\end{aligned}
$$

where $\left\|\mathbf{G}-\hat{\mathbf{G}}\right\|$ is the norm of matrix $\left(\mathbf{G}-\hat{\mathbf{G}}\right)$ and $\sigma^2_{T(u,v)}$ is the variance of the coefficient at transform location (u,v). The final simplification is based on the orthonormal nature of the basis images and the assumption that the pixels of $\mathbf{G}$ are generated by a random process with zero mean and known covariance. The total

Transform selection

The information packing ability of the DCT is superior to that of the DFT and WHT.

Although this condition usually holds for most images, the Karhunen-Loève transform, not the DCT, is the optimal transform in an information packing sense.

In fact, the KLT minimizes the mean-squared error for any input image and any number of retained coefficients.

However, because the KLT is data dependent, obtaining the KLT basis images for each subimage, in general, is a nontrivial computational task.

For this reason, the KLT is used infrequently for image compression.

Most transform coding systems are based on the DCT, which provides a good compromise between information packing ability and computational complexity.

It has the advantages of packing the most information into the fewest coefficients (for most images), and minimizing the block-like appearance, called **blocking artifact**, that results when the boundaries between subimages become visible.

Subimage size selection

Another significant factor affecting transform coding error and computational complexity is subimage size.

In most applications, images are subdivided so the correlation (redundancy) between adjacent subimages is reduced to some acceptable level and so n is an integer power of 2 where, as before, n is the subimage dimension.

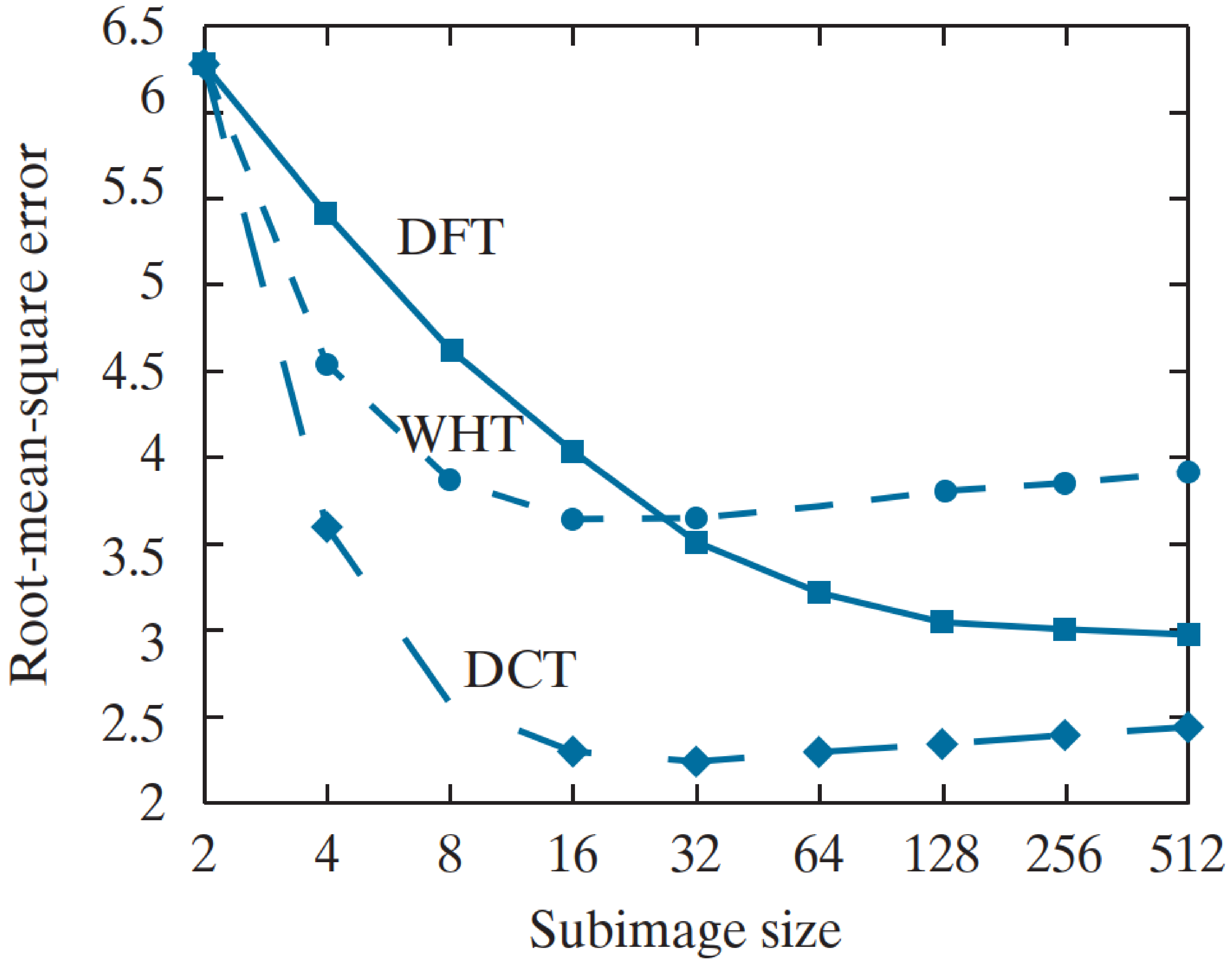
The latter condition simplifies the computation of the subimage transforms.

In general, both the level of compression and computational complexity increase as the subimage size increases.

The most popular subimage sizes are 8×8 and 16×16.

Subimage size selection

FIGURE 8.23
Reconstruction error versus subimage size.



The data plotted were obtained by dividing the monochrome image of Lena into subimages of size n × n, for n = 2,4,8,16,…,256,512, computing the transform of each subimage, truncating 75% of the resulting coefficients, and taking the inverse transform of the truncated arrays.

Bit allocation

The reconstruction error is a function of the number and relative importance of the transform coefficients that are discarded, as well as the precision that is used to represent the retained coefficients.

In most transform coding systems, the retained coefficients are selected on the basis of maximum variance, called **zonal coding**, or on the basis of maximum magnitude, called **threshold coding**.

The overall process of truncating, quantizing, and coding the coefficients of a transformed subimage is commonly called **bit allocation**.

Bit allocation

FIGURE 8.25 Approximations of Fig. 8.9(a) using 12.5% of the DCT coefficients: (a)–(b) threshold coding results; (c)–(d) zonal coding results. The difference images are scaled by 4.

Zonal Coding Implementation

Zonal coding is based on the information theory concept of viewing information as uncertainty: the transform coefficients of maximum variance carry the most image information, and should be retained in the coding process.

The variances themselves can be calculated directly from the ensemble of MN/n^2 transformed subimage arrays or based on an assumed image model.

In either case, the zonal sampling process can be viewed as multiplying each $T(u,v)$ by the corresponding element in a **zonal mask**, which is constructed by placing a 1 in the locations of maximum variance and a 0 in all other locations.

Coefficients of maximum variance usually are located around the origin of an image transform.

The coefficients retained during the zonal sampling process must be quantized and coded, so zonal masks are sometimes depicted showing the number of bits used to code each coefficient .

Zonal Coding Implementation

a b
c d

FIGURE 8.26
A typical
(a) zonal mask,
(b) zonal bit allocation,

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	3	3	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Threshold Coding Implementation

Zonal coding usually is implemented by using a single fixed mask for all subimages.

Threshold coding, however, is inherently **adaptive** in the sense that the location of the transform coefficients retained for each subimage vary from one subimage to another.

Threshold coding is the adaptive transform coding approach most often used in practice because of its computational simplicity.

The underlying concept is that, for any subimage, the transform coefficients of largest magnitude make the most significant contribution to reconstructed subimage quality.

Because the locations of the maximum coefficients vary from one subimage to another, the elements of $\chi(u,v)T(u,v)$ normally are reordered (in a predefined manner) to form a 1-D, run-length coded sequence.

Threshold Coding Implementation

(c) threshold mask, and
(d) thresholded coefficient ordering sequence. Shading highlights the coefficients that are retained.

1	1	0	1	1	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
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	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Threshold Coding Implementation

There are three basic ways to threshold a transformed subimage:

(1) A single global threshold can be applied to all subimages;

(2) a different threshold can be used for each subimage, or;

(3) the threshold can be varied as a function of the location of each coefficient within the subimage.

In the first approach, the level of compression differs from image to image, depending on the number of coefficients that exceed the global threshold.

In the second, called **N-largest coding**, the same number of coefficients is discarded for each subimage. As a result, the code rate is constant and known in advance.

The third technique, like the first, results in a variable code rate, but offers the advantage that thresholding and quantization can be combined by replacing $\chi(u,v)T(u,v)$ with

$$\hat{T}(u,v) = \text{round}\left[\frac{T(u,v)}{Z(u,v)}\right]$$

Threshold Coding Implementation

Z(u,v) is an element of the following transform normalization array:

$$\mathbf{Z} = \begin{bmatrix} Z(0,0) & Z(0,1) & \dots & Z(0,n-1) \\ Z(1,0) & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ Z(n-1,0) & Z(n-1,1) & \dots & Z(n-1,n-1) \end{bmatrix}$$

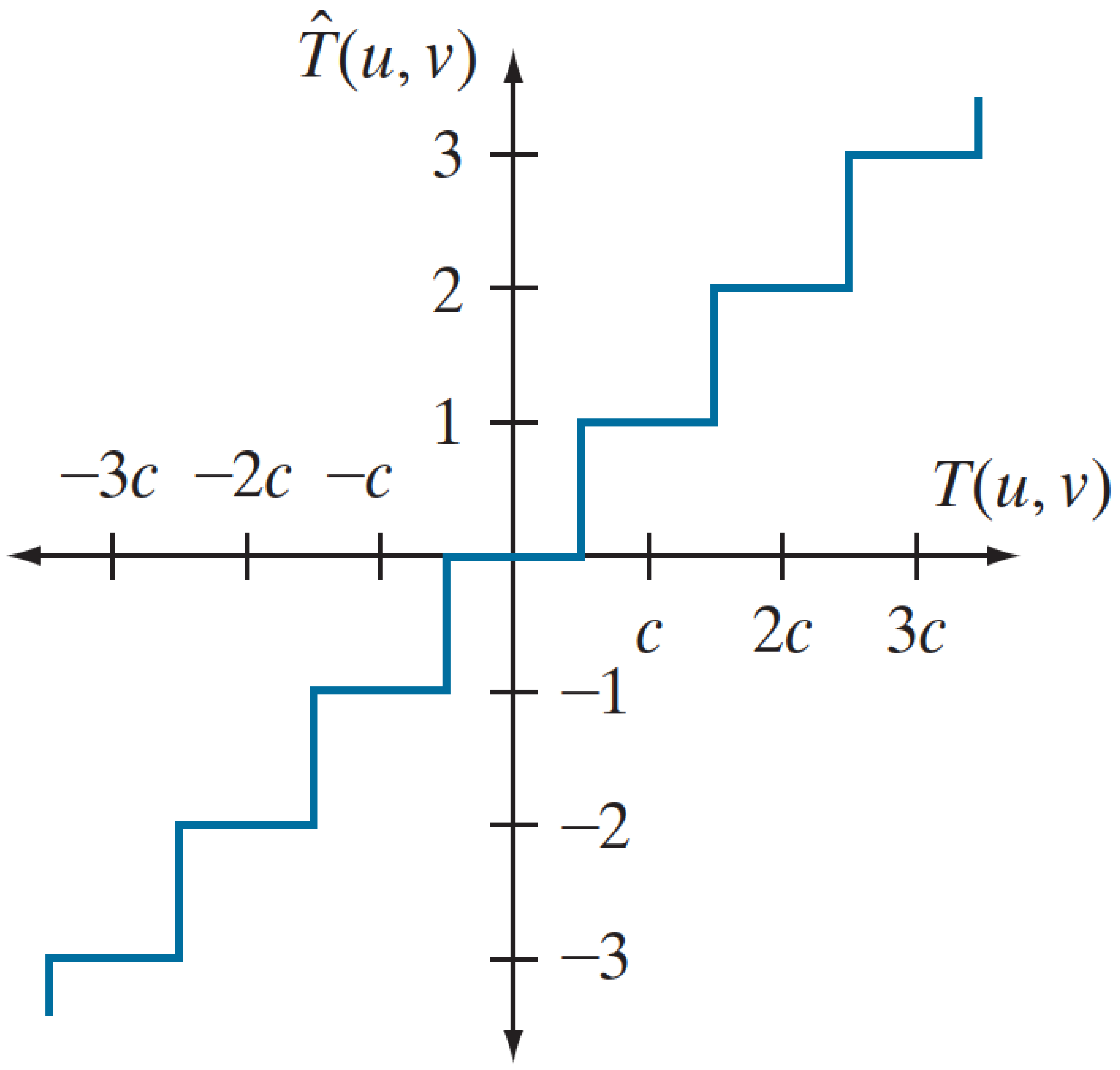
Before $\hat{T}(u,v)$ can be inverse transformed to obtain an approximation of subimage g(x, y), it must be multiplied by Z(u,v).

$$\dot{T}(u,v) = \hat{T}(u,v)Z(u,v)$$

Threshold Coding Implementation

a b

FIGURE 8.27
(a) A threshold coding quantization curve [see Eq. (8-28)]. (b) A typical normalization matrix.



16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Threshold Coding Implementation

FIGURE 8.28 Approximations of Fig. 8.9(a) using the DCT and normalization array of Fig. 8.27(b): (a) **Z**, (b) 2**Z**, (c) 4**Z**, (d) 8**Z**, (e) 16**Z**, and (f) 32**Z**.

JPEG

One of the most popular continuous tone, still-frame compression standards is the JPEG standard.

It defines three different coding systems:

(1) a lossy baseline coding system, which is based on the DCT and is adequate for most compression applications;

(2) an extended coding system for greater compression, higher precision, or progressive reconstruction applications; and

(3) a lossless independent coding system for reversible compression.

To be JPEG compatible, a product or system must include support for the baseline system.

No particular file format, spatial resolution, or color space model is specified.

JPEG

In the baseline system, often called the **sequential baseline system**, the input and output data precision is limited to 8 bits, whereas the quantized DCT values are restricted to 11 bits. The compression itself is performed in three sequential steps:

DCT computation, quantization, and variable-length code assignment.

The image is first subdivided into pixel blocks of size 8 × 8, which are processed left-to-right, top-to-bottom.

For each 8 × 8 block, its 64 pixels are level-shifted by subtracting the quantity 2^{k-1}, where 2^k is the maximum number of intensity levels.

The 2-D discrete cosine transform of the block is then computed, quantized and reordered, using the zigzag pattern, to form a 1-D sequence of quantized coefficients.

Because the one-dimensionally reordered array generated under the zigzag pattern is arranged qualitatively according to increasing spatial frequency, the JPEG coding procedure is designed to take advantage of the long runs of zeros that normally result from the reordering.

JPEG

Quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

102	-33	-58	35	58	-51	15	-12
5	-34	49	18	27	1	-5	3
-46	14	80	-35	-50	19	7	-18
-53	21	34	-20	2	34	36	12
9	-2	9	-5	-32	-15	45	37
-8	15	-16	7	-8	11	4	7
19	-28	-2	-26	-2	7	-44	-21
18	25	-12	-44	35	48	-37	-3

DCT coefficients

Quantization
[÷]

6	-3	-6	2	2	-1	0	0
0	-3	4	1	1	0	0	0
-3	1	5	-1	-1	0	0	0
-4	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantized DCT coefficients

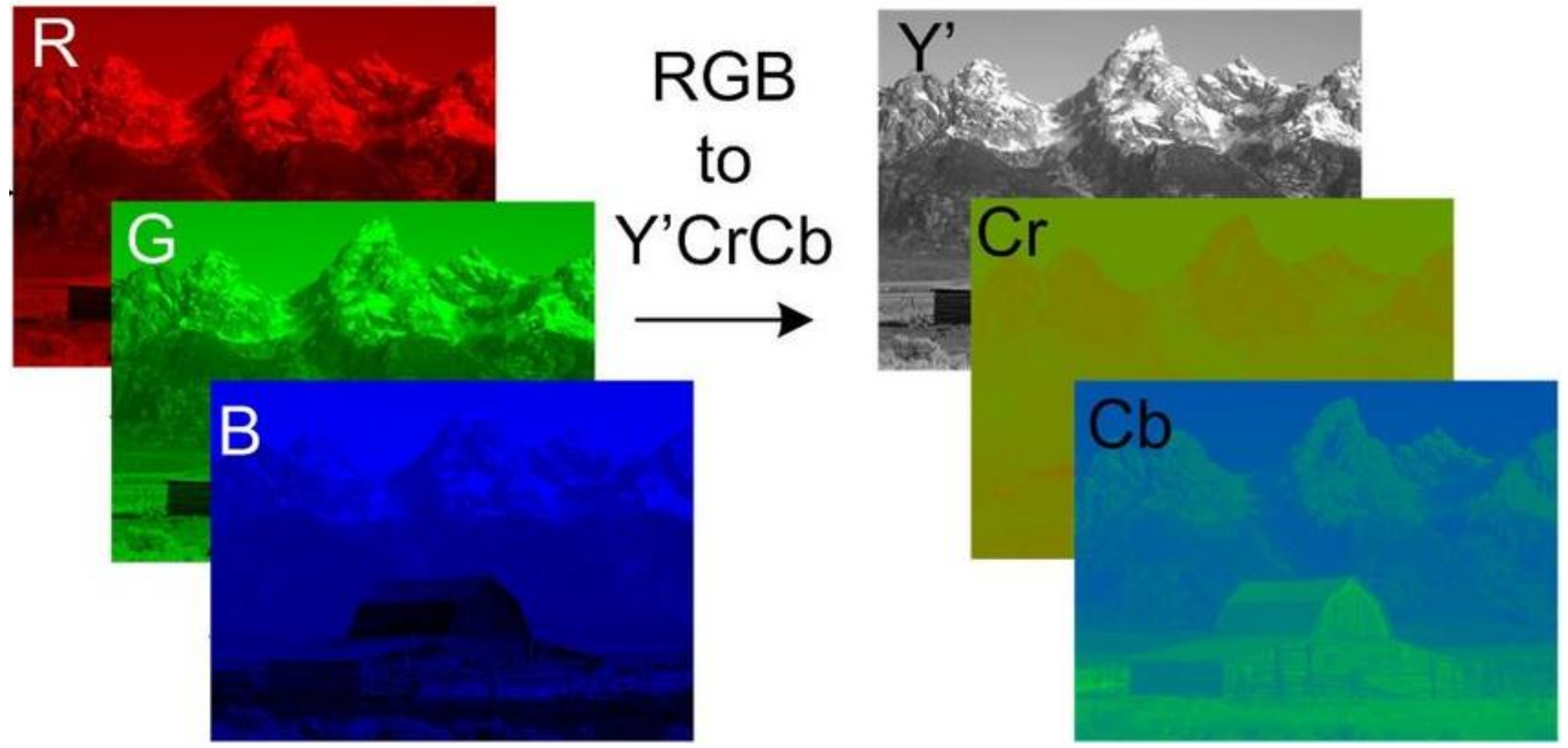
JPEG

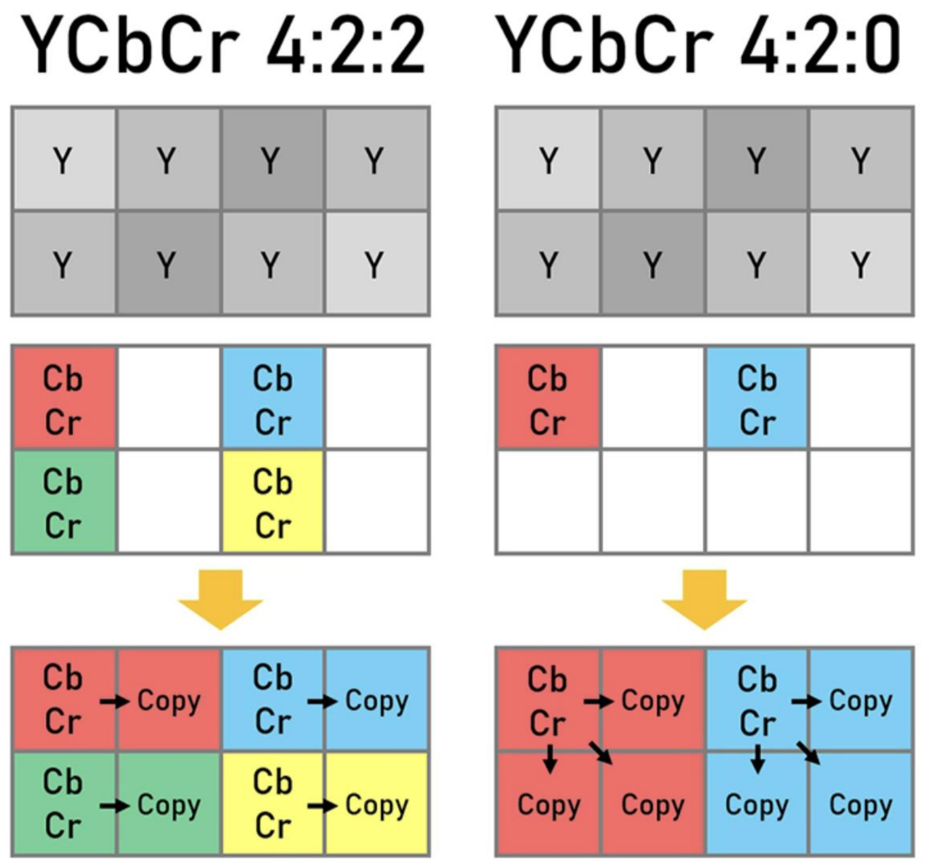
The nonzero AC coefficients are coded using a variable-length code (Huffman) that defines the coefficient values and number of preceding zeros.

The DC coefficient is difference coded relative to the DC coefficient of the previous subimage.

The JPEG recommended luminance quantization array is the one we have seen that can be scaled to provide a variety of compression levels.

The scaling of this array allows users to select the “quality” of JPEG compressions.





JPEG

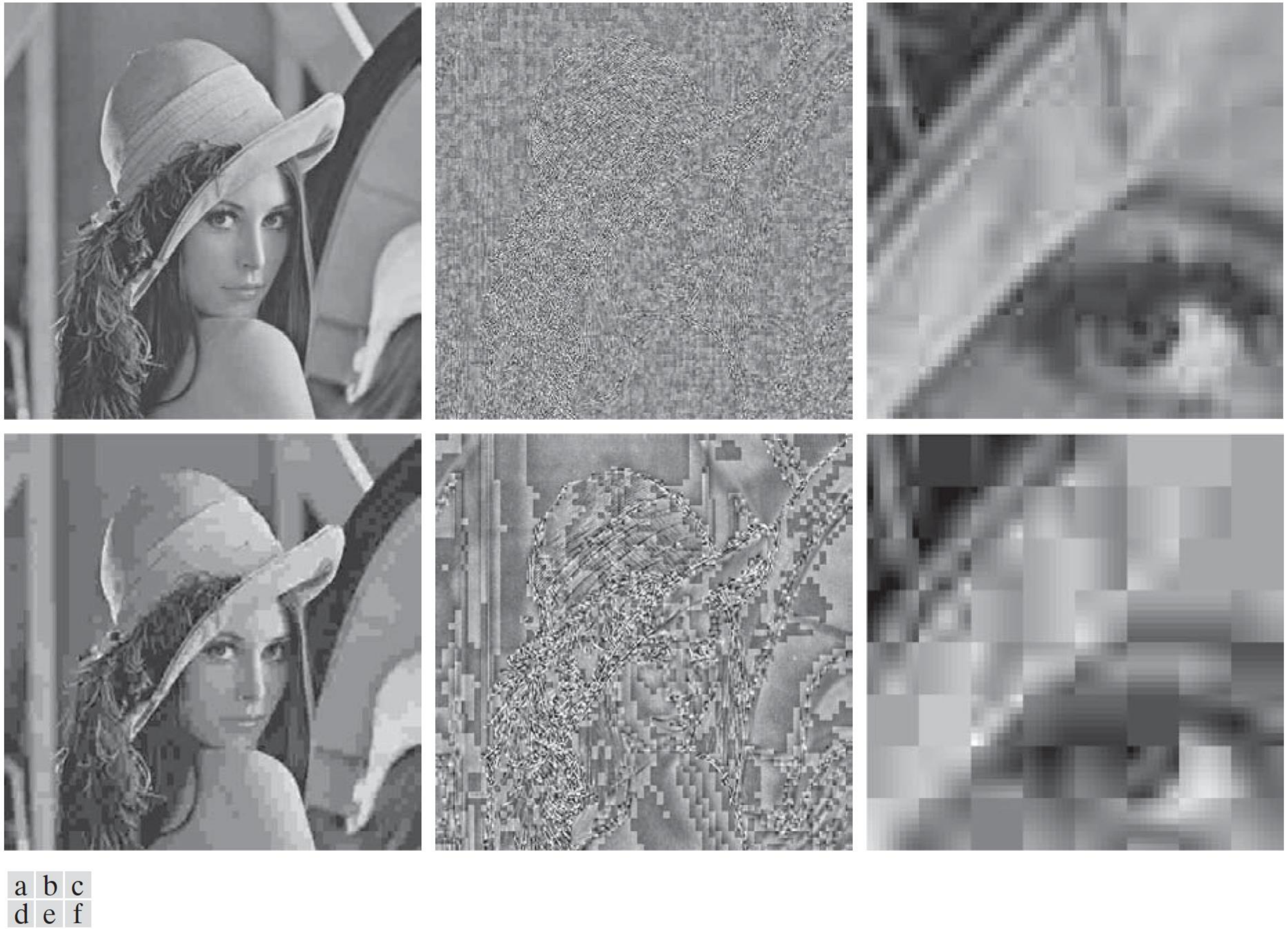


FIGURE 8.29 Two JPEG approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image.

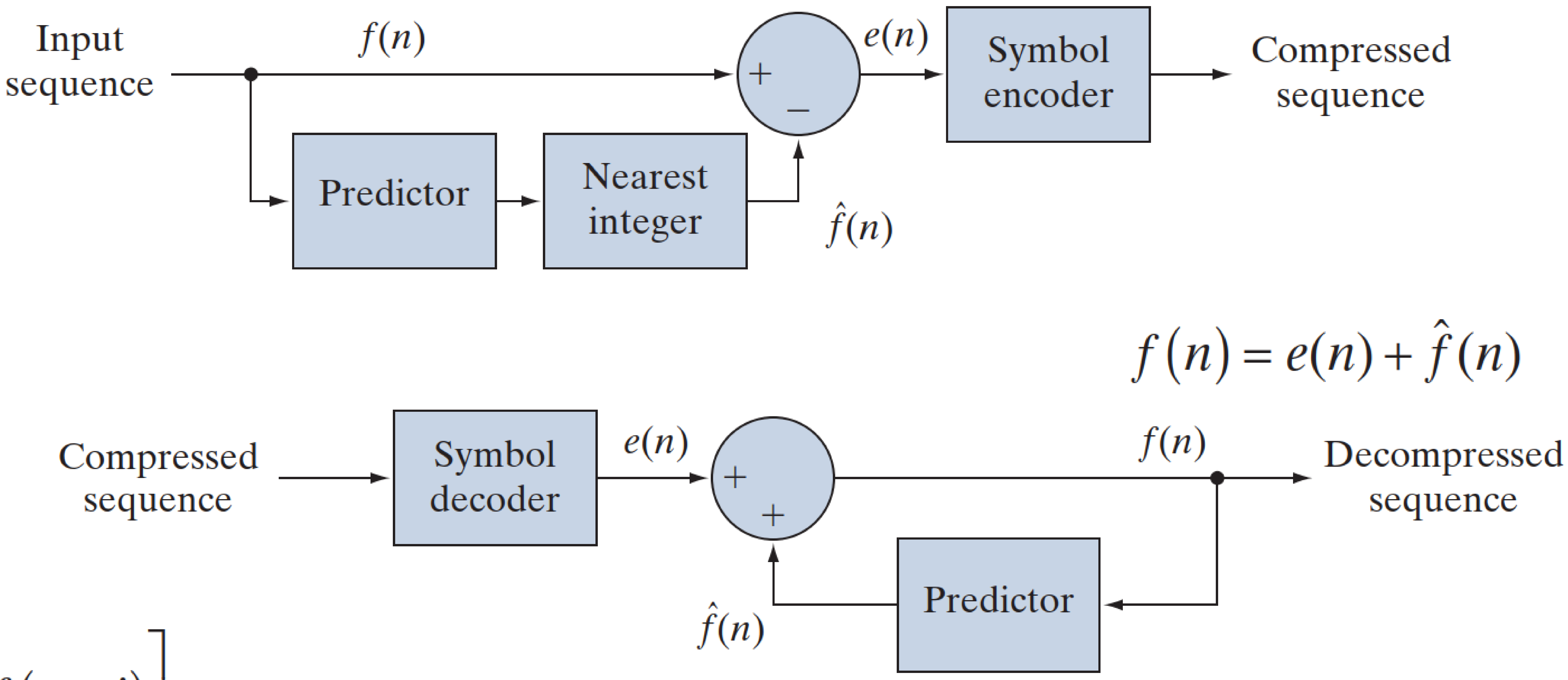
Predictive coding

The **predictive coding** approach is based on eliminating the redundancies of closely spaced pixels—in space and/or time—by extracting and coding only the **new information** in each pixel.

The new information of a pixel is defined as the difference between the actual and predicted value of the pixel.

Lossless predictive coding

$$e(n) = f(n) - \hat{f}(n)$$



a
b

FIGURE 8.30
A lossless predictive coding model: (a) encoder; (b) decoder.

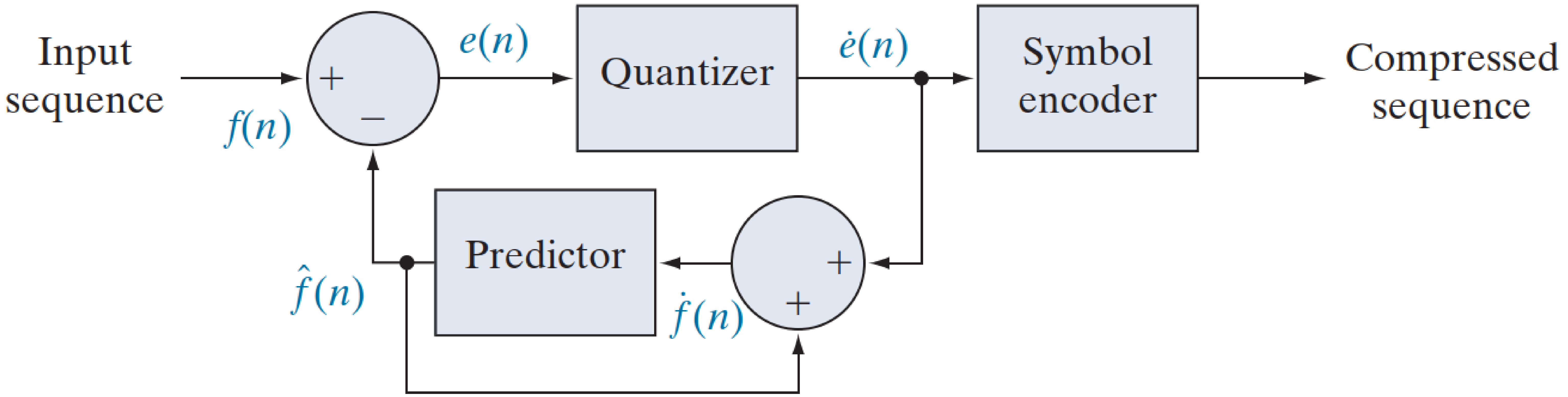
$$f(n) = e(n) + \hat{f}(n)$$

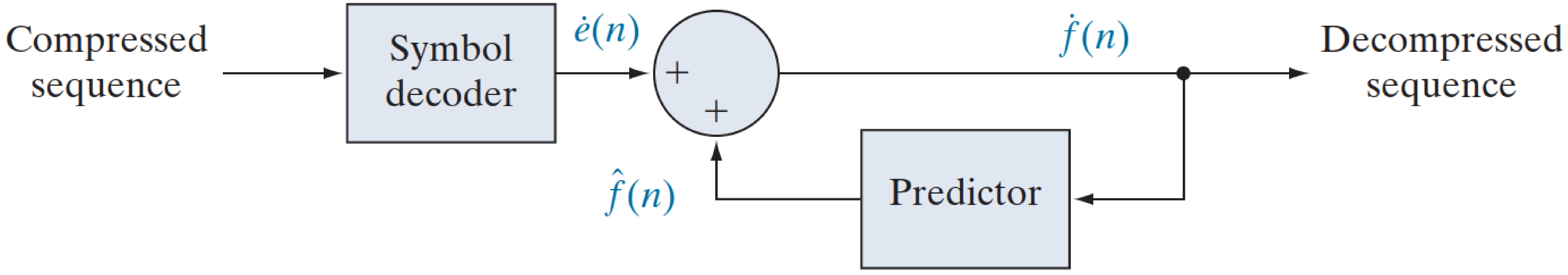
$$\hat{f}(n) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(n-i)\right]$$

Lossy predictive coding

a
b

FIGURE 8.38
A lossy predictive coding model:
(a) encoder;
(b) decoder.





Lossless predictive coding

Various local, global, and adaptive methods can be used to generate fˆ(n).

In many cases, the prediction is formed as a linear combination of m previous samples.

That is,

$$\hat{f}(n) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(n-i)\right]$$

If the input sequence is considered to be samples of an image, the f(n) are pixels and the *m* samples used to predict the value of each pixel come from the current scan line (called **1-D linear predictive coding**), from the current and previous scan lines (called **2-D linear predictive coding**), or from the current image and previous images in a sequence of images (called **3-D linear predictive coding**). Thus, for 1-D linear predictive image coding

$$\hat{f}(x,y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x,y-i)\right]$$

Lossless predictive coding

In 2-D predictive coding, the prediction is a function of the previous pixels in a left-to-right, top-to-bottom scan of an image.

In the 3-D case, it is based on these pixels and the previous pixels of preceding frames.

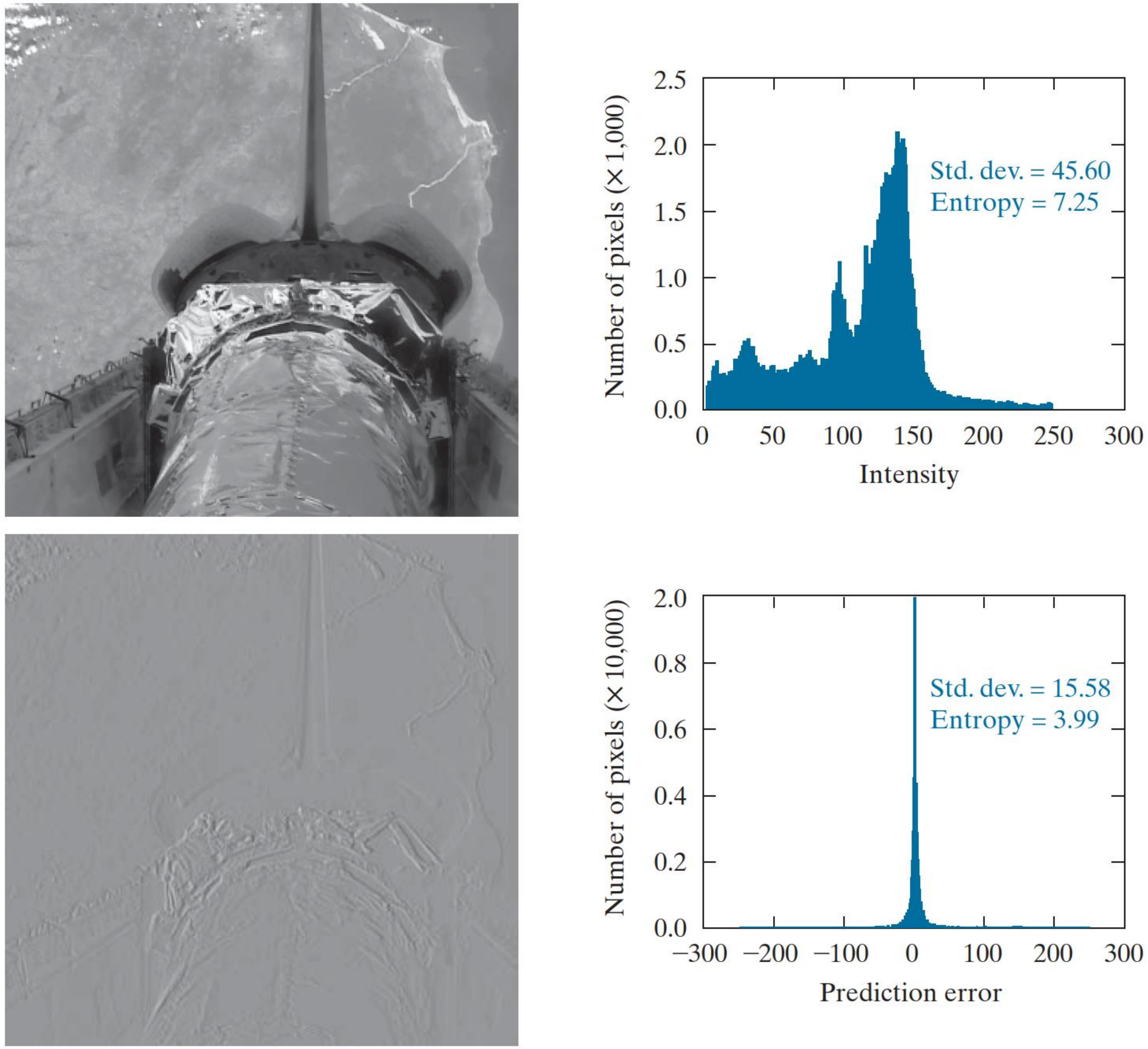
The prediction cannot be evaluated for the first *m* pixels of each line, so those pixels must be coded by using other means (such as a Huffman code) and considered as an overhead of the predictive coding process.

Lossless predictive coding

$$\hat{f}(x,y) = \text{round}\left[\alpha f(x,y-1)\right]$$

The entropy of the prediction error is significantly less than the estimated entropy of the original image (3.99 bits pixel as opposed to 7.25 bits pixel).

This decrease in entropy reflects removal of a great deal of spatial redundancy, despite the fact that for k-bit images, (k + 1)-bit numbers are needed to represent accurately the prediction error sequence e(x, y).



a b
c d

FIGURE 8.31 (a) A view of the Earth from an orbiting space shuttle. (b) The intensity histogram of (a). (c) The prediction error image resulting from Eq. (8-33). (d) A histogram of the prediction error. (Original image courtesy of NASA.)

Lossless predictive coding

$$\hat{f}(x,y,t) = \text{round}\left[\alpha f(x,y,t-1)\right]$$

The standard deviation of the error is much smaller than in the previous example: 3.76 bits pixel as opposed to 15.58 bits pixel.

In addition, the entropy of the prediction error has decreased from 3.99 to 2.59 bits pixel.

a b
c d

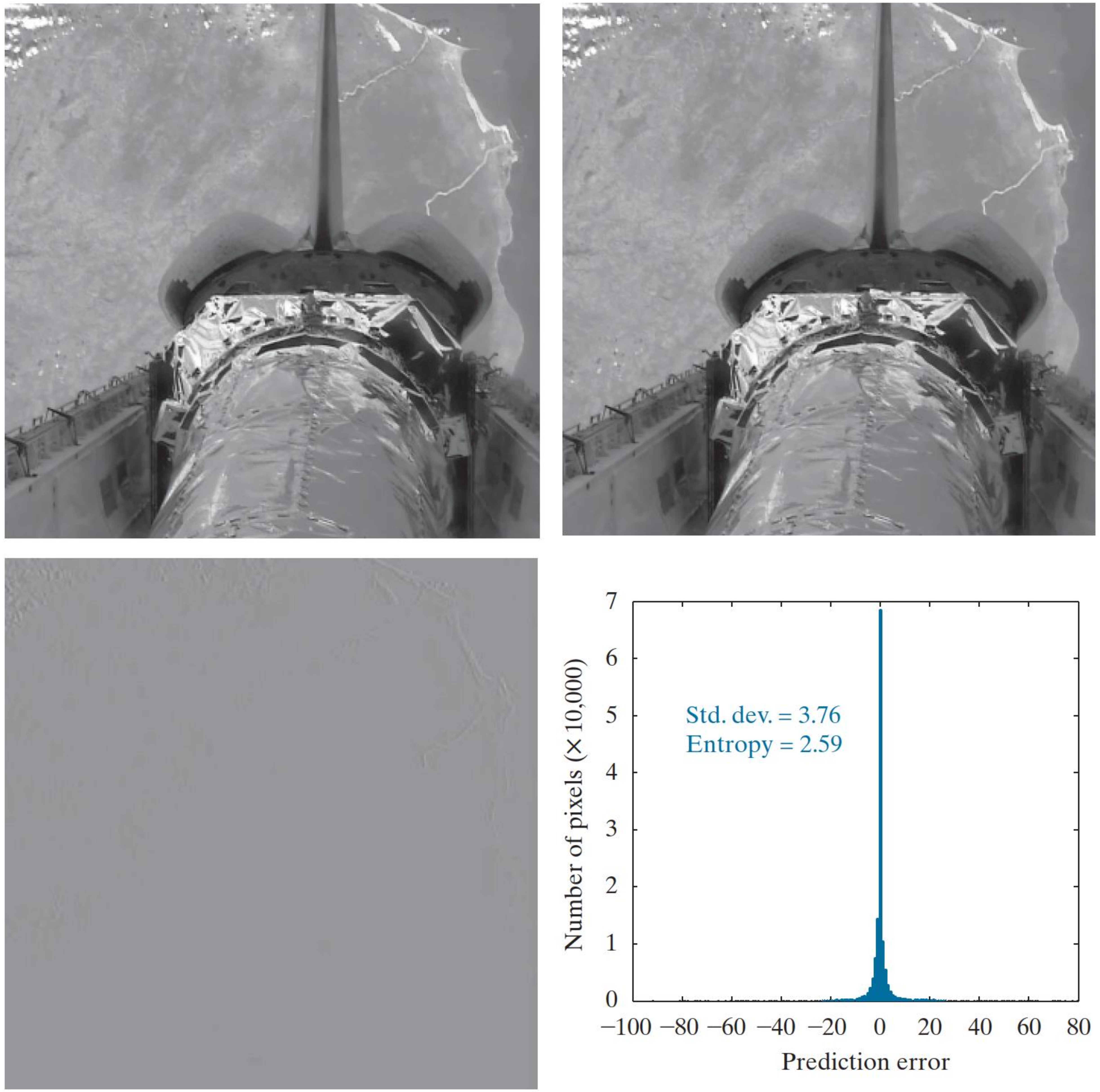


FIGURE 8.32 (a) and (b) Two views of Earth from an orbiting space shuttle video. (c) The prediction error image resulting from Eq. (8-35). (d) A histogram of the prediction error. (Original images courtesy of NASA.)

Motion compensated prediction residuals

Successive frames in a video sequence often are very similar.

Coding their differences can reduce temporal redundancy and provide significant compression.

However, when a sequence of frames contains rapidly moving objects—or involves camera zoom and pan, sudden scene changes, or fade-ins and fade-outs—the similarity between neighboring frames is reduced, and compression is affected negatively.

Video compression systems avoid the problem of data expansion in two ways:

1. By tracking object movement and compensating for it during the prediction and differencing process.

2. By switching to an alternate coding method when there is insufficient interframe correlation (similarity between frames) to make predictive coding advantageous.

The first of these, called **motion compensation.**

Motion compensated prediction residuals

When there is insufficient interframe correlation to make predictive coding effective, the second problem is typically addressed using a block-oriented 2-D transform, like JPEG's DCT-based coding.

Frames compressed in this way (i.e., without a prediction residual) are called **intraframes** or **Independent frames** (**I-frames**).

They can be decoded without access to other frames in the video to which they belong.

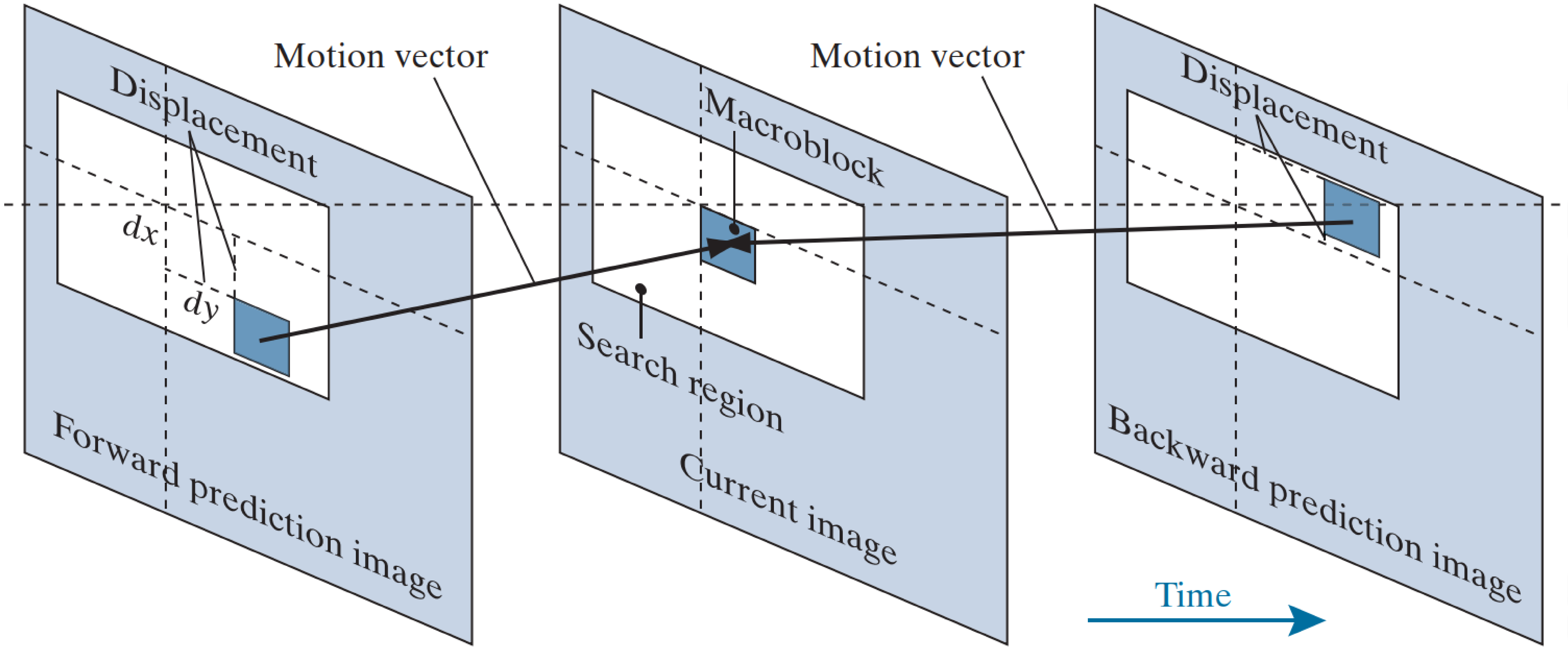
I-frames usually resemble JPEG encoded images, and are ideal starting points for the generation of prediction residuals.

Moreover, they provide a high degree of random access, ease of editing, and resistance to the propagation of transmission error.

As a result, all standards require the periodic insertion of I-frames into the compressed video codestream.

Motion compensated prediction residuals

FIGURE 8.33
Macroblock motion specification.



Motion compensated prediction residuals

Each video frame is divided into non-overlapping rectangular regions (typically of size 4×4 to 16×16) called **macroblocks**.

The “movement” of each macroblock with respect to its “most likely” position in the previous (or subsequent) video frame, called the **reference frame**, is encoded in a **motion vector**.

The vector describes the motion by defining the horizontal and vertical displacement from the “most likely” position.

The displacements typically are specified to the nearest pixel, ½ pixel, or ¼ pixel precision.

If subpixel precision is used, the predictions must be interpolated from a combination of pixels in the reference frame.

An encoded frame that is based on the previous frame is called a **Predictive frame** (**P-frame**); one that is based on the subsequent frame is called a **Bidirectional frame** (**B-frame**). B-frames require the compressed codestream to be reordered so that frames are presented to the decoder in the proper decoding sequence, rather than the natural display order.

Motion compensated prediction residuals

Motion estimation is the key component of motion compensation.

During motion estimation, the motion of objects is measured and encoded into motion vectors.

The search for the “best” motion vector requires that a criterion of optimality be defined.

For example, motion vectors may be selected on the basis **of maximum correlation** or **minimum error** between macroblock pixels and the predicted from the chosen reference frame.

One of the most commonly used error measures is **mean absolute distortion (MAD)**

$$MAD(x,y) = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left|f(x+i,\ y+j) - p(x+i+dx,\ y+j+dy)\right|$$

Typically, dx and dy must fall within a limited search region around each macroblock.

Values from ±8 to ±64 pixels are common, and the horizontal search area is often slightly larger than the vertical area.

Motion compensated prediction residuals

Motion estimation is performed by searching for the dx and dy that minimize MAD(x, y) over the allowed range of motion vector displacements, including subpixel displacements.

This process often is called **block matching**.

An exhaustive search guarantees the best possible result, but is computationally expensive because every possible motion must be tested over the entire displacement range.

For 16 ×16 macroblocks and a ±32 pixel displacement, 4225 16 × 16 MAD calculations must be performed for each macroblock in a frame when integer displacement precision is used. If ½ or ¼ pixel precision is desired, the number of calculations is multiplied by a factor of 4 or 16, respectively.

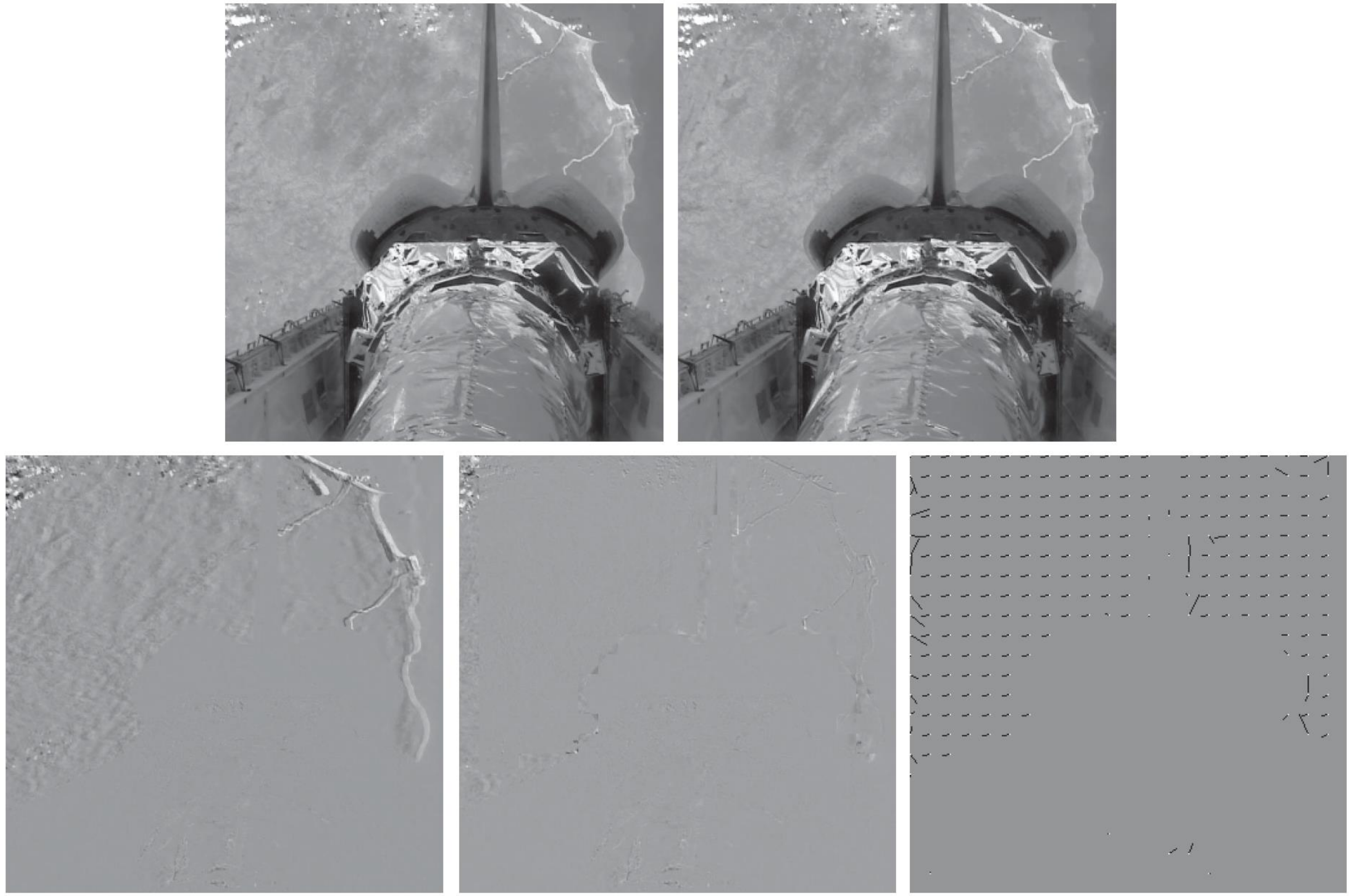
Fast search algorithms can reduce the computational burden, but may or may not yield optimal motion vectors.

Motion compensated prediction residuals

The standard deviation of the prediction residual in Fig. 8.34(c) is 12.73 intensity levels; its entropy is 4.17 bits pixel.

Figure 8.34(d) shows a motion compensated prediction residual with a much lower standard deviation (5.62 as opposed to 12.73 intensity levels) and slightly lower entropy (3.04 vs. 4.17 bits pixel).

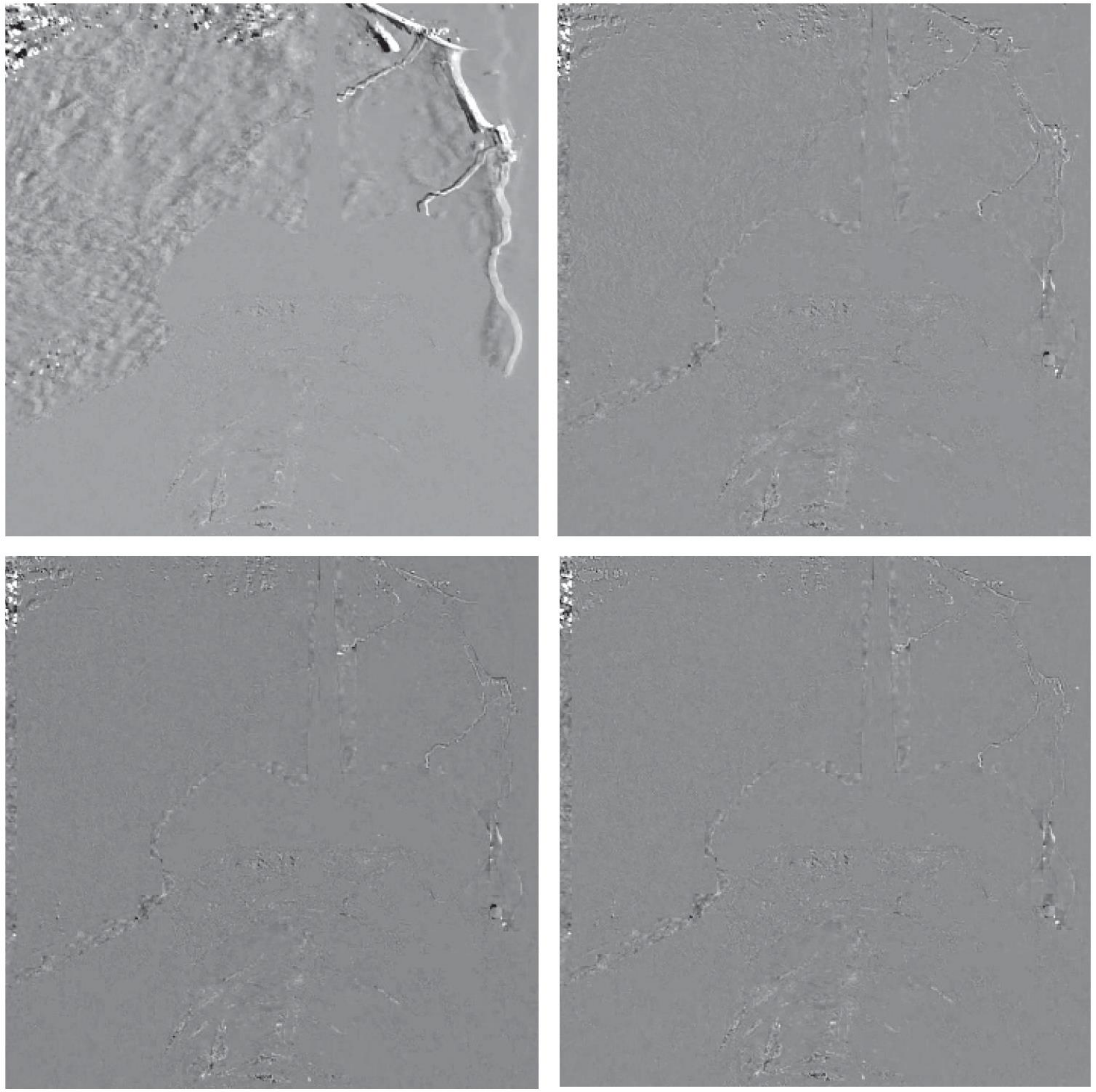
The motion prediction used 16 × 16 macroblocks and compared each macroblock against every 16 × 16 region in Fig. 8.34(a) that fell within ±16 pixels of the macroblock's position in (b).



a b
c d e

FIGURE 8.34 (a) and (b) Two views of Earth that are thirteen frames apart in an orbiting space shuttle video. (c) A prediction error image without motion compensation. (d) The prediction residual with motion compensation. (e) The motion vectors associated with (d). The white dots in (e) represent the arrow heads of the motion vectors that are depicted. (Original images courtesy of NASA.)

Motion compensated prediction residuals



a b
c d

FIGURE 8.35 Sub-pixel motion compensated prediction residuals: (a) without motion compensation; (b) single pixel precision; (c) ½ pixel precision; and (d) ¼ pixel precision. (All prediction errors have been scaled to the full intensity range and then multiplied by 2 to increase their visibility.)

Motion compensated prediction residuals

Motion estimation is a computationally demanding task.

Fortunately, only the encoder must estimate macroblock motion.

Given the motion vectors of the macroblocks, the decoder simply accesses the areas of the reference frames that were used in the encoder to form the prediction residuals.

Because of this, motion estimation is not included in most video compression standards.

Compression standards focus on the decoder, placing constraints on macroblock dimensions, motion vector precision, horizontal and vertical displacement ranges, and the like.

Most of the standards use an 8×8 DCT for I-frame encoding, but specify a larger area (i.e., 16×16 macroblock) for motion compensation.

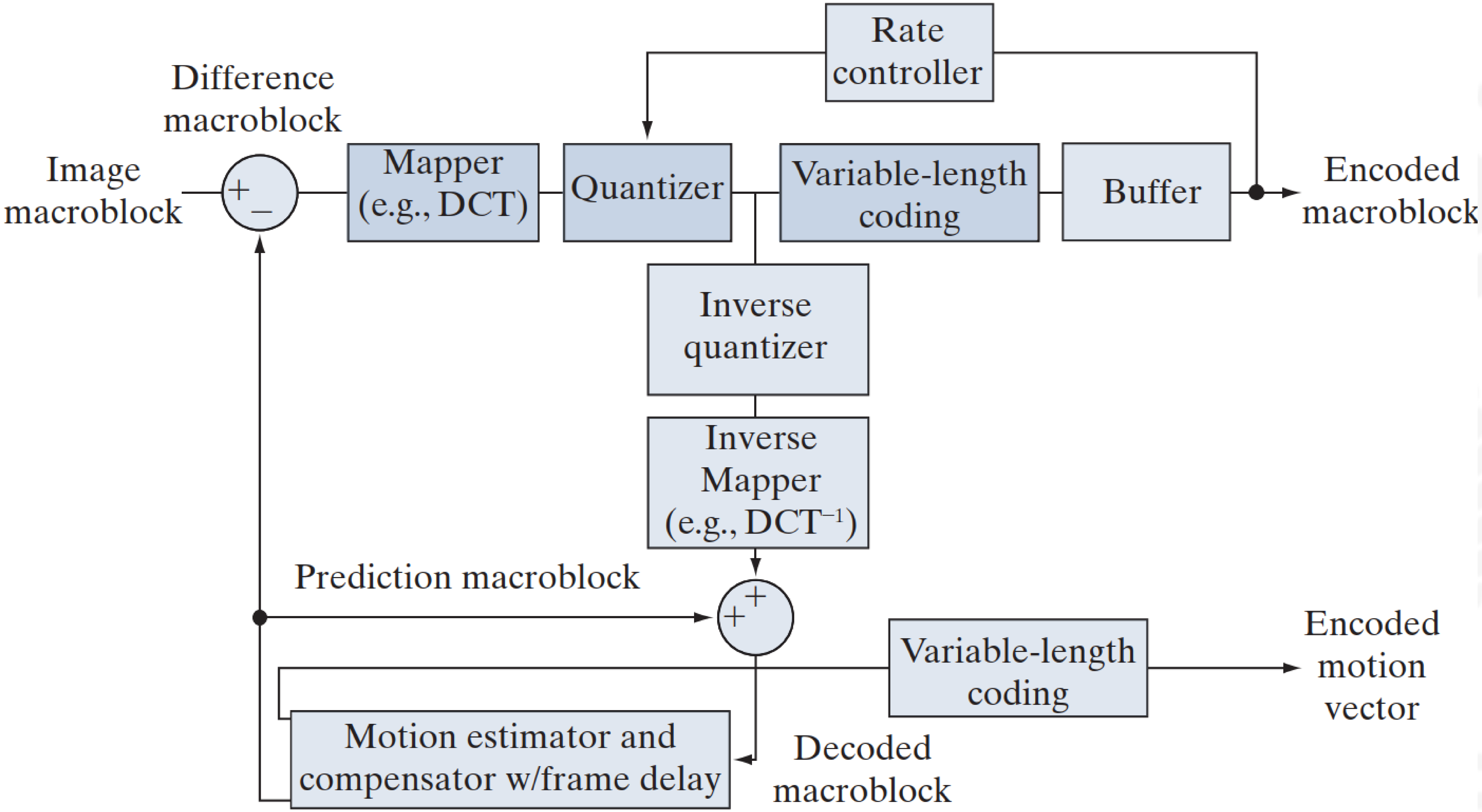
In addition, even the P- and B-frame prediction residuals are transform coded due to the effectiveness of DCT coefficient quantization.

Motion compensated prediction residuals

Parameter	H.261	MPEG-1	H.262 MPEG-2	H.263	MPEG-4	VC-1 WMV-9	H.264 MPEG-4 AVC
Motion vector precision	1	½	½	½	¼	¼	¼
Macroblock sizes	16 × 16	16 × 16	16 × 16 16 × 8	16 × 16 8 × 8	16 × 16 8 × 8	16 × 16 8 × 8	16 × 16 16 × 8 8 × 8 8 × 4 4 × 8 4 × 4
Transform	8 × 8 DCT	8 × 8 DCT	8 × 8 DCT	8 × 8 DCT	8 × 8 DCT	8 × 8 8 × 4 4 × 8 4 × 4 Integer DCT	4 × 4 8 × 8 Integer
Interframe predictions	P	P, B	P, B	P, B	P, B	P, B	P, B
I-frame intra-predictions	No	No	No	No	No	No	Yes

Motion compensated prediction residuals

FIGURE 8.36
A typical motion compensated video encoder.



High Efficiency Video Coding (HEVC) Standard

1) Coding tree units and coding tree block (CTB) structure:

The core of the coding layer in previous standards was the macroblock, containing a 16×16 block of luma samples and, in the usual case of 4:2:0 color sampling, two corresponding 8×8 blocks of chroma samples; whereas the analogous structure in HEVC is the coding tree unit (CTU), which has a size selected by the encoder and can be larger than a traditional macroblock.

The CTU consists of a luma CTB and the corresponding chroma CTBs and syntax elements. The size L×L of a luma CTB can be chosen as L = 16, 32, or 64 samples, with the larger sizes typically enabling better compression.

HEVC then supports a partitioning of the CTBs into smaller blocks using a tree structure and quadtree-like signaling.

High Efficiency Video Coding (HEVC) Standard

2) Coding units (CUs) and coding blocks (CBs):

The quadtree syntax of the CTU specifies the size and positions of its luma and chroma CBs. The splitting of a CTU into luma and chroma CBs is signaled jointly.

One luma CB and ordinarily two chroma CBs, together with associated syntax, form a coding unit (CU).

A CTB may contain only one CU or may be split to form multiple CUs, and each CU has an associated partitioning into prediction units (PUs) and a tree of transform units (TUs).

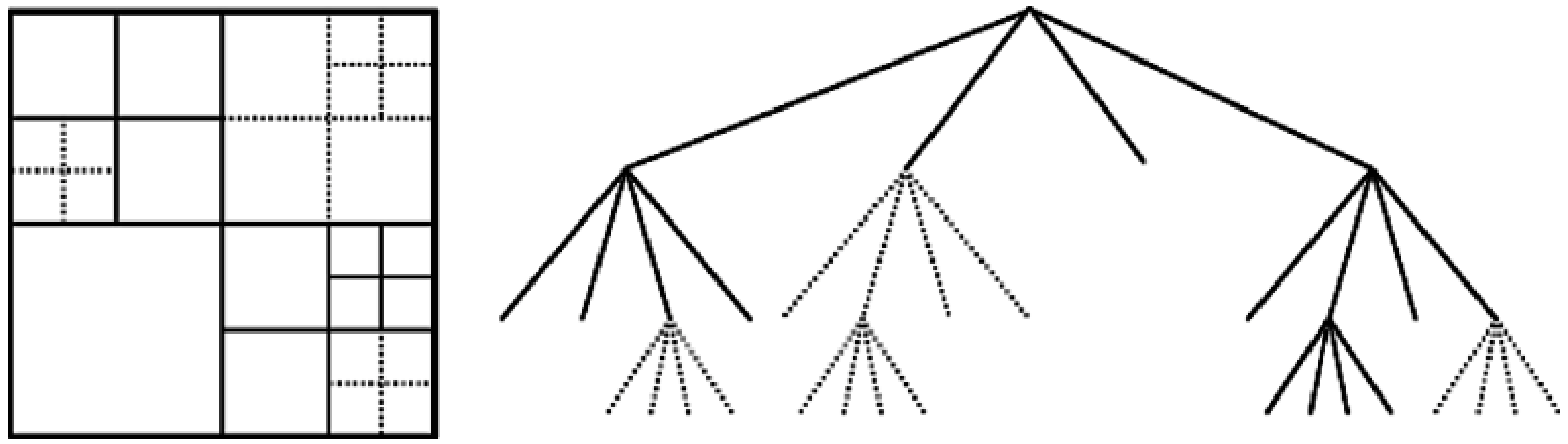
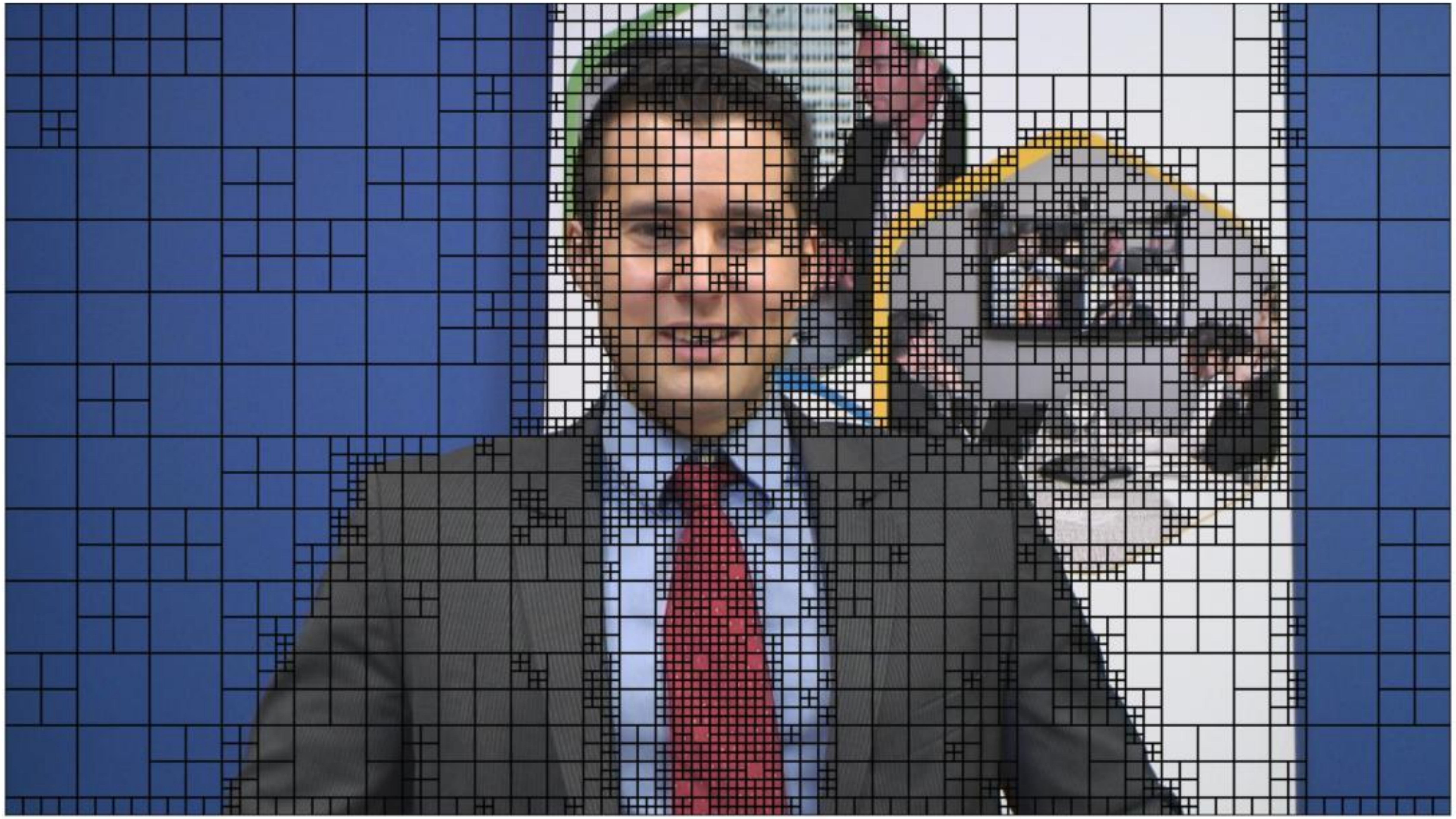


Fig. 4. Subdivision of a CTB into CBs [and transform block (TBs)]. Solid lines indicate CB boundaries and dotted lines indicate TB boundaries. (a) CTB with its partitioning. (b) Corresponding quadtree.

High Efficiency Video Coding (HEVC) Standard



Taken from https://sonnati.wordpress.com/2014/06/20/h265-part-i-technical-overview/

High Efficiency Video Coding (HEVC) Standard

3) Prediction units and prediction blocks (PBs):

The decision whether to code a picture area using interpicture or intrapicture prediction is made at the CU level.

Depending on the basic prediction-type decision, the luma and chroma CBs can then be further split in size and predicted from luma and chroma prediction blocks (PBs).

HEVC supports variable PB sizes from 64×64 down to 4×4 samples.

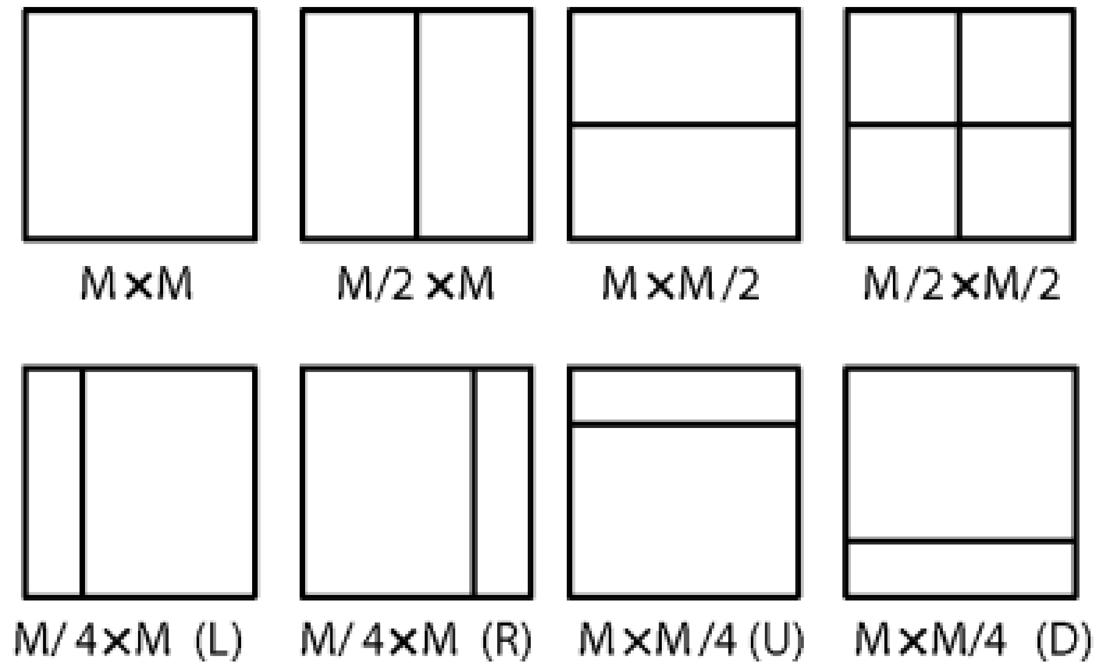


Fig. 3. Modes for splitting a CB into PBs, subject to certain size constraints. For intrapicture-predicted CBs, only $M \times M$ and $M/2 \times M/2$ are supported.

High Efficiency Video Coding (HEVC) Standard

4) TUs and transform blocks:

The prediction residual is coded using block transforms.

The luma CB residual may be identical to the luma transform block (TB) or may be further split into smaller luma TBs. The same applies to the chroma TBs.

Integer basis functions similar to those of a discrete cosine transform (DCT) are defined for the square TB sizes 4×4, 8×8, 16×16, and 32×32.

For the *4×4* transform of luma *intrapicture* prediction residuals, an integer transform derived from a form of *discrete sine transform* (DST) is alternatively specified.

High Efficiency Video Coding (HEVC) Standard

5) Motion vector signaling:

Advanced motion vector prediction (AMVP) is used, including derivation of several most probable candidates based on data from adjacent PBs and the reference picture.

A merge mode for MV coding can also be used, allowing the inheritance of MVs from temporally or spatially neighboring PBs.

High Efficiency Video Coding (HEVC) Standard

6) Motion compensation:

Quarter-sample precision is used for the MVs, and 7-tap or 8-tap filters are used for interpolation of fractional-sample positions.

Similar to H.264/MPEG-4 AVC, multiple reference pictures are used.

For each PB, either one or two motion vectors can be transmitted, resulting either in unipredictive or bipredictive coding, respectively.

As in H.264/MPEG-4 AVC, a scaling and offset operation may be applied to the prediction signal(s) in a manner known as weighted prediction.

High Efficiency Video Coding (HEVC) Standard

7) Intrapicture prediction:

The decoded boundary samples of adjacent blocks are used as reference data for spatial prediction in regions where interpicture prediction is not performed.

Intrapicture prediction supports 33 directional modes, plus planar (surface fitting) and DC (flat) prediction modes.

The selected intrapicture prediction modes are encoded by deriving most probable modes (e.g., prediction directions) based on those of previously decoded neighboring PBs.

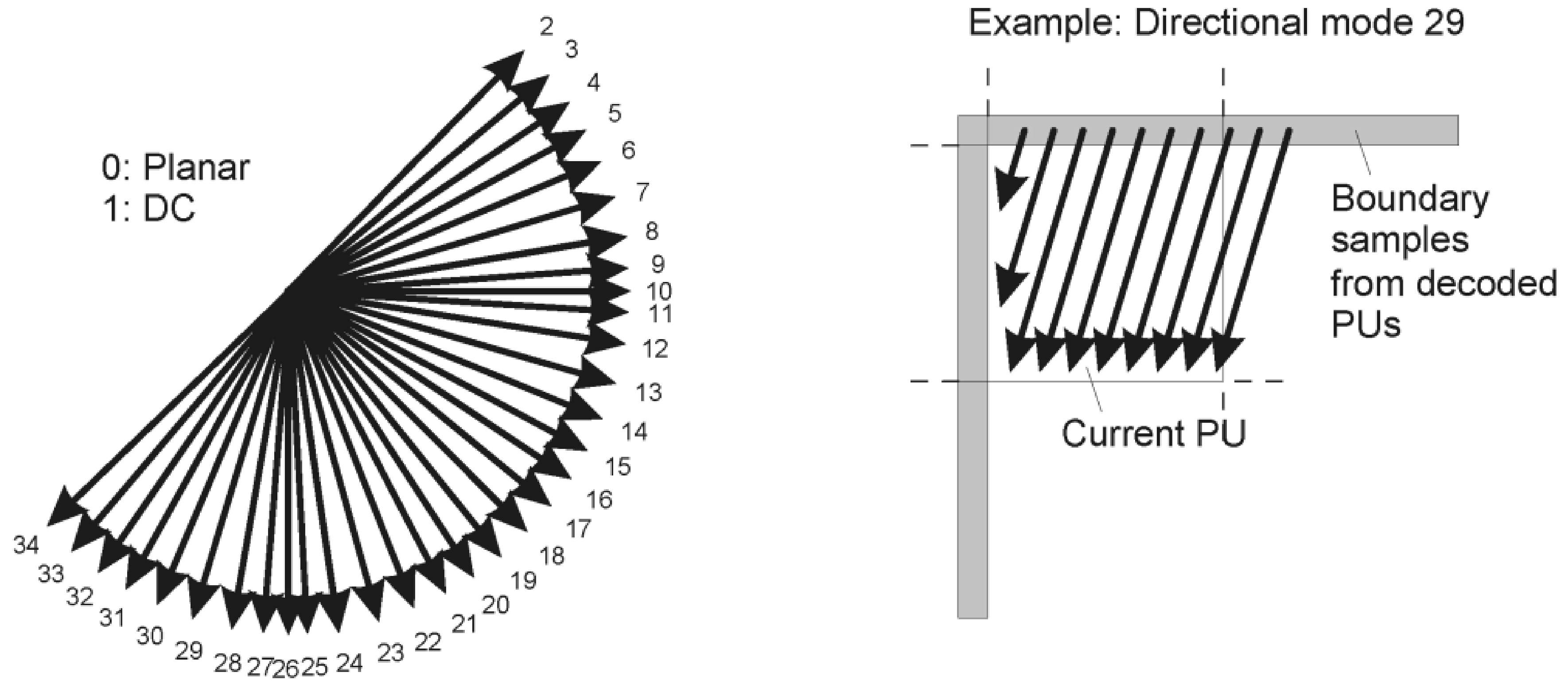


Fig. 6. Modes and directional orientations for intrapicture prediction.

High Efficiency Video Coding (HEVC) Standard

8) Quantization control:

As in H.264/MPEG-4 AVC, uniform reconstruction quantization (URQ) is used in HEVC, with quantization scaling matrices supported for the various transform block sizes.

9) Entropy coding:

Context adaptive binary arithmetic coding (CABAC) is used for entropy coding.

This is similar to the CABAC scheme in H.264/MPEG-4 AVC, but has undergone several improvements to improve its throughput speed (especially for parallel-processing architectures) and its compression performance, and to reduce its context memory requirements.

High Efficiency Video Coding (HEVC) Standard

10) In-loop deblocking filtering:

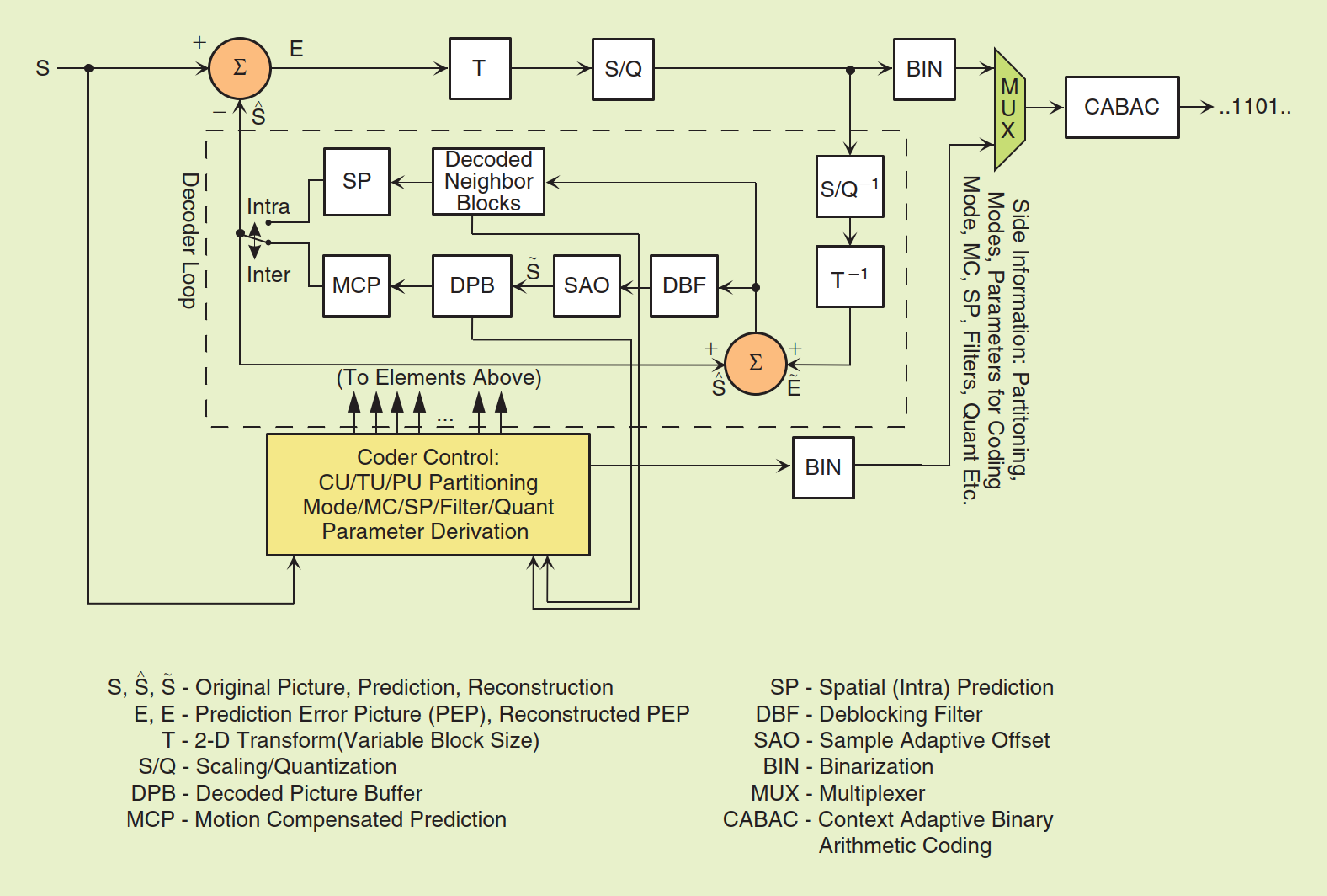
A deblocking filter similar to the one used in H.264/MPEG-4 AVC is operated within the interpicture prediction loop.

However, the design is simplified in regard to its decision-making and filtering processes, and is made more friendly to parallel processing.

11) Sample adaptive offset (SAO):

A nonlinear amplitude mapping is introduced within the interpicture prediction loop after the deblocking filter. Its goal is to better reconstruct the original signal amplitudes by using a look-up table that is described by a few additional parameters that can be determined by histogram analysis at the encoder side.

High Efficiency Video Coding (HEVC) Standard



[FIG1] **Hybrid video encoder for HEVC.**

Study:

- Rafael Gonzalez, Richard Woods, “Digital Image Processing”, 4th edition, Pearson, 2018
  - Chapter 8.1, 8.2, 8.4, 8.6, 8.7, 8.8, 8.9, 8.10
- Sullivan, G. J., Ohm, J. R., Han, W. J., & Wiegand, T. (2012). Overview of the high efficiency video coding (HEVC) standard. IEEE Transactions on circuits and systems for video technology, 22(12), 1649-1668.
- Ohm, J. R., & Sullivan, G. J. (2012). High efficiency video coding: the next frontier in video compression [standards in a nutshell]. IEEE Signal Processing Magazine, 30(1), 152-158.