Fundamentals

- Image Compression is reducing the amount of data required to represent an image.
- It is one of the most useful and commercially successful technology in the field of DIP.
- To better understand , consider the amount of data required to represent a two-hour SD(standard definition) television movie using 720x480x24 bit pixel arrays.
- A digital video is a sequence of video frames in which each frame is a full-color still image.
- Video players must display the frames sequentially at 30 fps(frames per second).
- So the SD digital data must be accessed at

30 frames/sec x (720x480) pixles/frame x 3 bytes/pixel = 31,104,000 bytes/sec

And a 2-hour movie consists of

31,104,000 bytes/sec x (60 x 60) sec/hr x 2 hr = 2.24 x 10^{11} bytes

- Web page images and high resolution digital camera photos also are compressed to save storage space and reduce transmission time.
- Image compression is also used in
  - Televideo conferencing
  - Remote sensing
  - Document and medical imaging,
  - Facsimile transmission (FAX)

Redundancy

- Let b and b´ denote the number of bits in two representations of the same information.
- The compression ration is defined as

$$C = b / b'$$

- If C = 10, the larger representation has 10 bits of data for every 1 bit of data in smaller representation.
- The relative data redundancy R of the representation with b bits is

$$R = 1 - (1/C)$$

- For C=10, R value is 0.9.
- Indicating that 90% of its data is redundant.

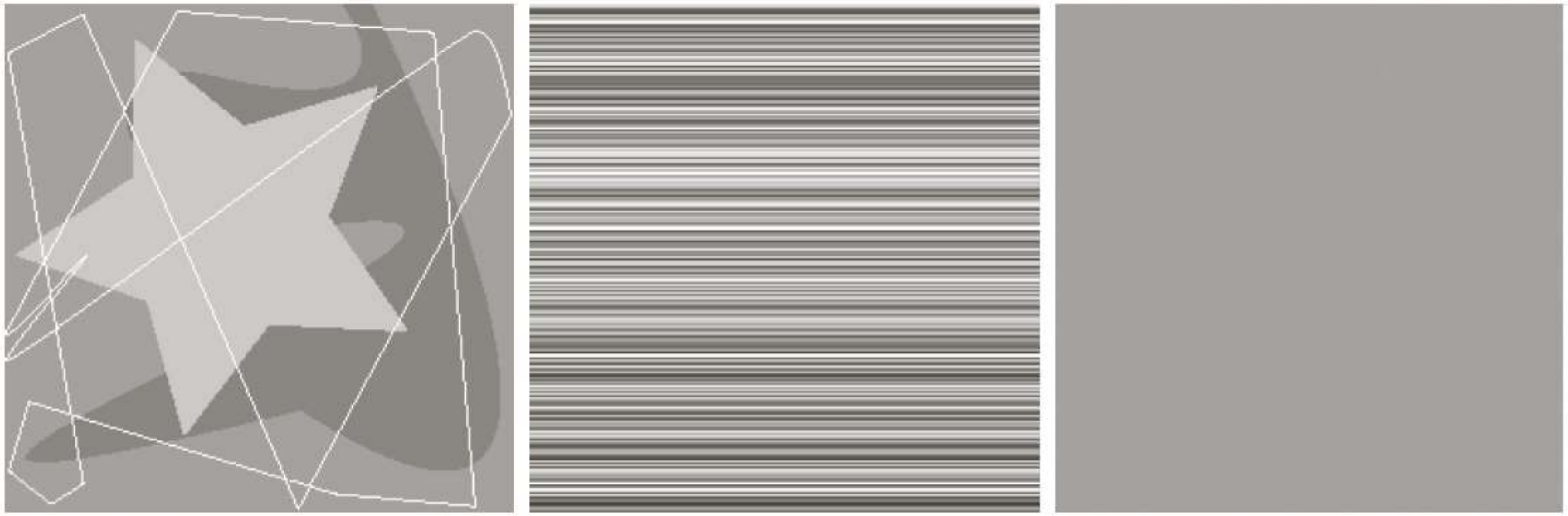
Redundancy

- Two dimensional intensity arrays suffer from three principal types of data redundancies that can be identified and exploited:
  - Coding redundancy
  - Spatial and temporal redundancy
  - Irrelevant information
- Coding redundancy:
  - A code is a system of symbols used to represent a body of information or set of events.
  - Each piece of information is assigned a *code word.*
  - The number of symbols in code word is its length.
  - The 8-bit codes that are used to represent the intensities in 2-D intensity arrays contain more bits than are needed to represent the intensities.

Redundancy

- Spatial and temporal redundancy:
  - Pixels of most 2-D intensity arrays are correlated spatially (i.e each pixel is similar to or dependent on neighboring pixels)
  - In a video sequence , pixels are correlated temporally (i.e each pixel is similar to or dependent on pixels in nearby frames)
- Irrelevant information:
  - Most 2-D intensity arrays contain information that is ignored by the human visual system.
  - Redundant in the sense that it is not used.

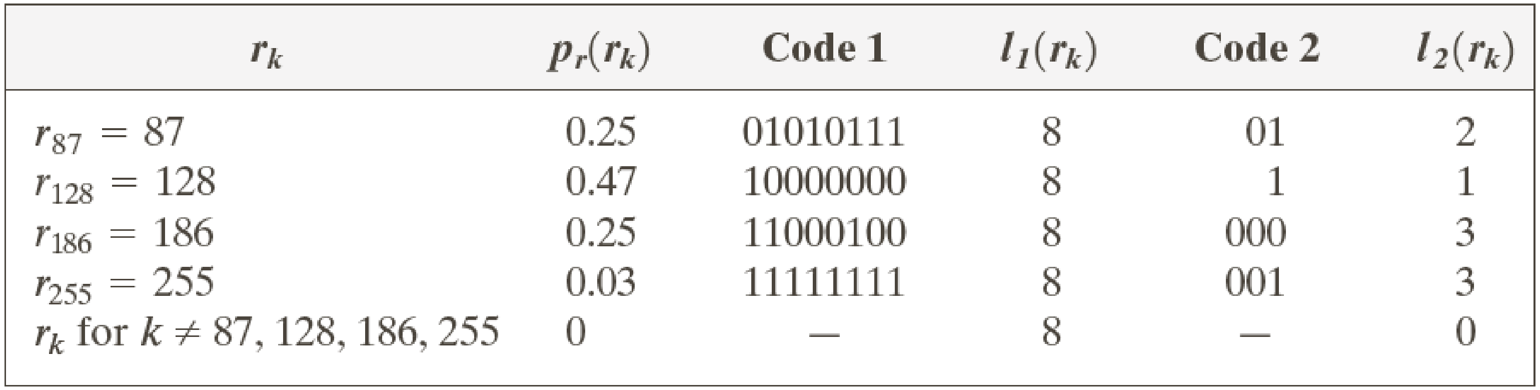
Image Compression



a b c

FIGURE 8.1 Computer generated 256 × 256 × 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

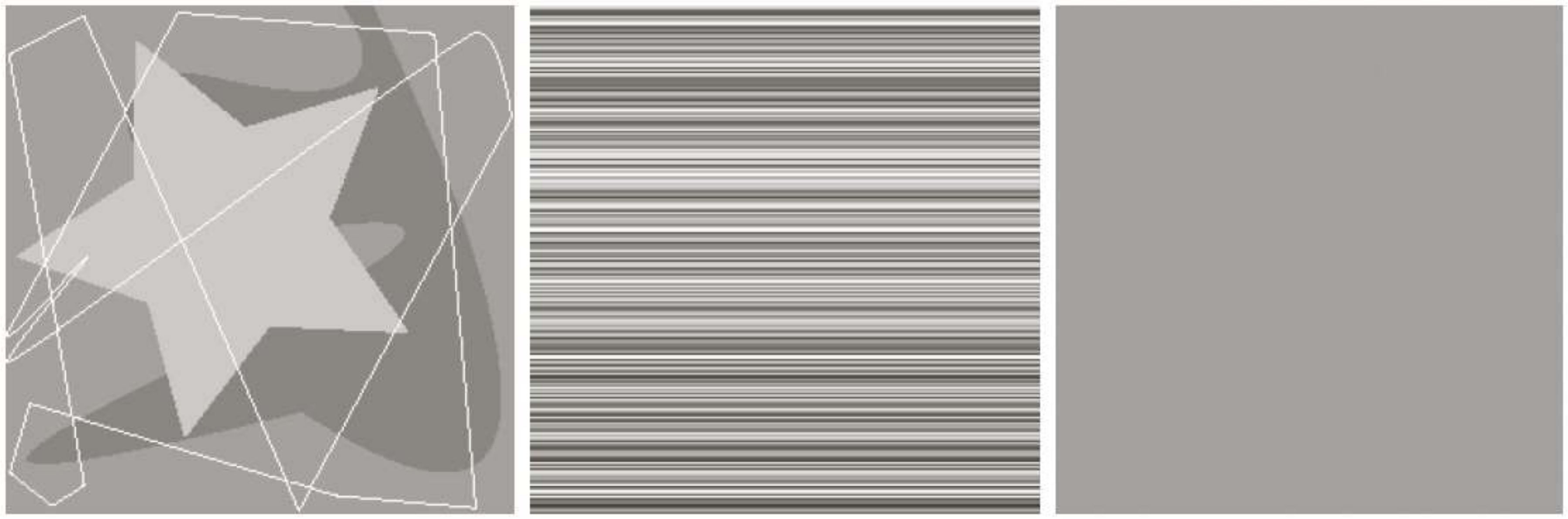


r_k	$p_r(r_k)$	**Code 1**	$l_1(r_k)$	**Code 2**	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	–	8	–	0

TABLE 8.1 Example of variable-length coding.

Spatial and temporal redundancy

- Observations in the corresponding 2-D intensity array
  - All 256 intensities are equally probable(Observe the histogram is uniform)
  - Because the intensity of each line was selected randomly, its pixels are independent of one another in vertical direction.
  - Because the pixels along each line are identical, they are maximally correlated in the horizontal direction.



Spatial and temporal redundancy

- First observation tells us that the image cannot be compressed by variable length coding alone.
- Observations 2 and 3 reveal a significant spatial redundancy.
- This can be eliminated by representing image as a sequence of run-length pairs.
- Run-length pair contains the start of new intensity and the number of consecutive intensity pixels that have that intensity.
- Run-length based representation compresses the 2-D, 8-bit intensity array by (256x256x8) / [(256+256)x8] or 128:1

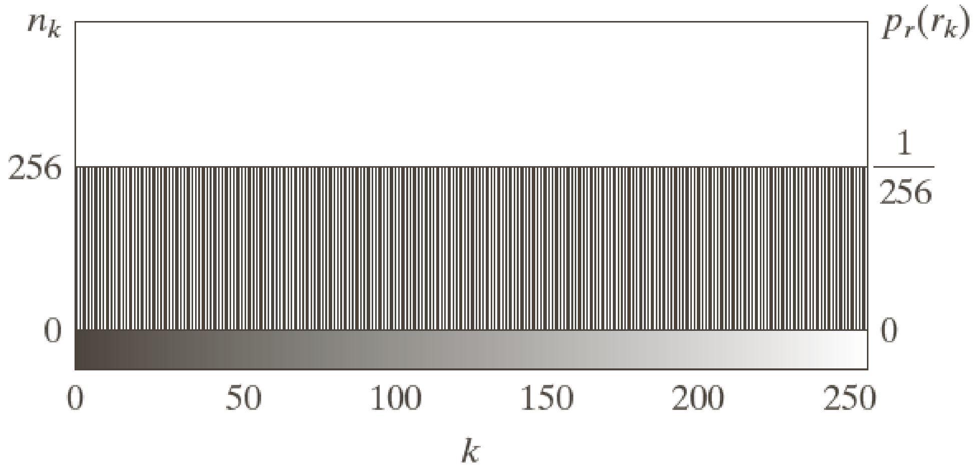
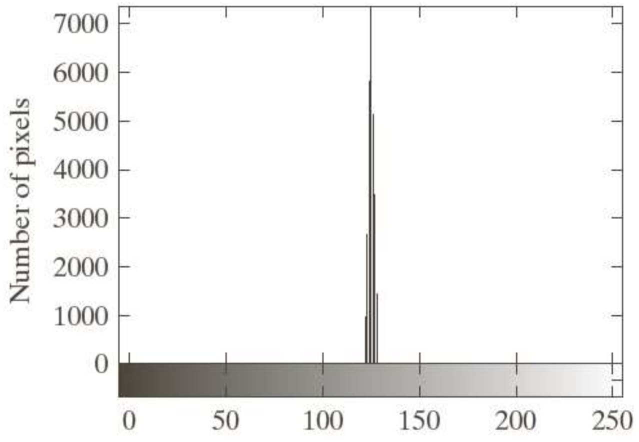
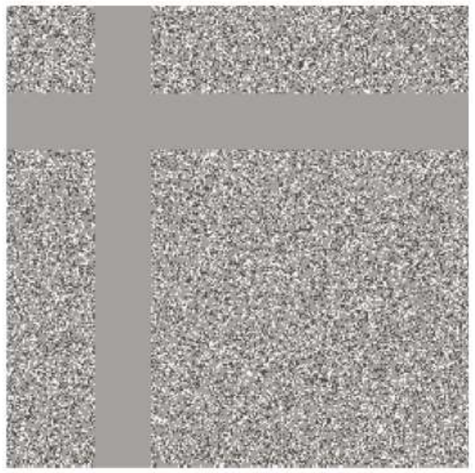


FIGURE 8.2 The intensity histogram of the image in Fig. 8.1(b).

Irrelevant information

- Most 2-D intensity arrays contain information that is ignored by the human visual system.
- Consider the below image and its histogram.
- Because it appears to be homogeneous field of gray, it can represented by its average intensity alone- a single 8-bit value.
- The resulting compression is (256x256x8) / 8 or 65,536:1





Measuring image information

- According to information theory, a random event E with probability P(E) is said to contain I(E) = log (1/P(E)) = -log(P(E)) units of information.
- The base of the logarithm determines the unit used to measure information.
- If base 2 is selected, the unit of information is the bit.
- Given a source of statistically independent random events from a discrete set of possible events { a1,a2, ... , aJ} with associated probabilities { P(a1), P(a2), ..., P(aJ) }.
- The average information per source output, called the entrpoy of the source, is

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

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

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

- Compute the entropy of an image having the following histogram

r_k	$P_r(r_k)$
87	0.25
128	0.47
186	0.25
255	0.03

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

Consider the simple 3X7, 8-bit image:

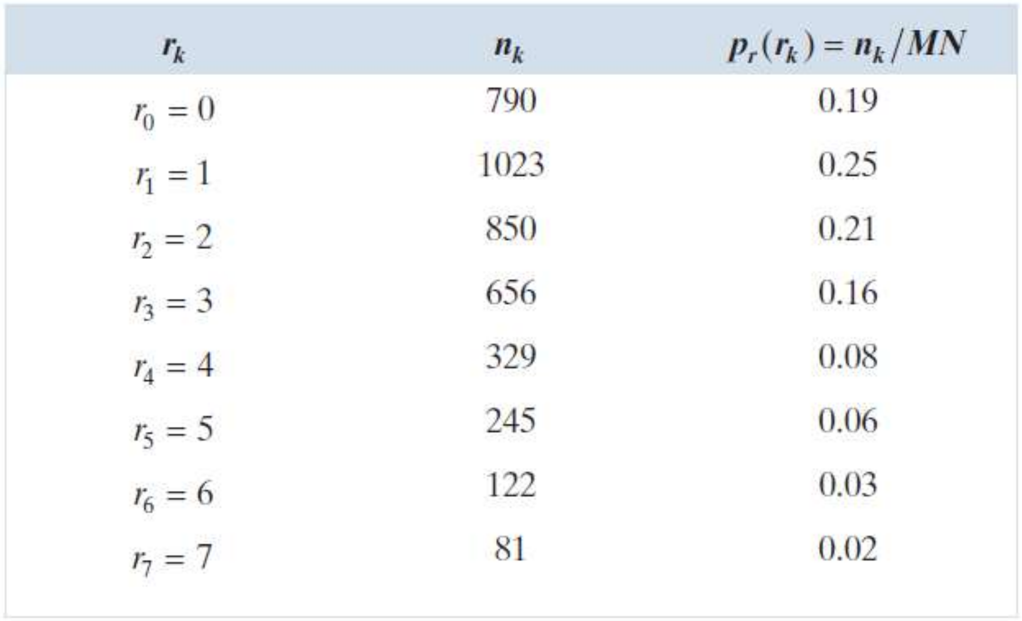
12 12 12 95 69 43 43

12 12 12 95 69 43 43

12 12 12 95 69 43 43

What is the entropy of the image?

- Compute the entropy of an image having the following histogram



r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

Fidelity Criteria

- The removal of "irrelevant information" from an image involves a loss of real or quantitative information.
- Two types of criteria can be used to quantify the loss:
  - Objective fidelity criteria
  - Subjective fidelity criteria.
- Objective fidelity criterion uses a mathematical function to express information loss.
- Example - root-mean-square (rms) error.
- Let $f(x,y)$ be an input image and $\hat{f}(x,y)$ is a decompressed image of f(x,y) then the rms error is given as

- If $\hat{f}(x,y)$ is considered to be the sum of the original image f(x,y) and an error or noise signal e(x,y), then

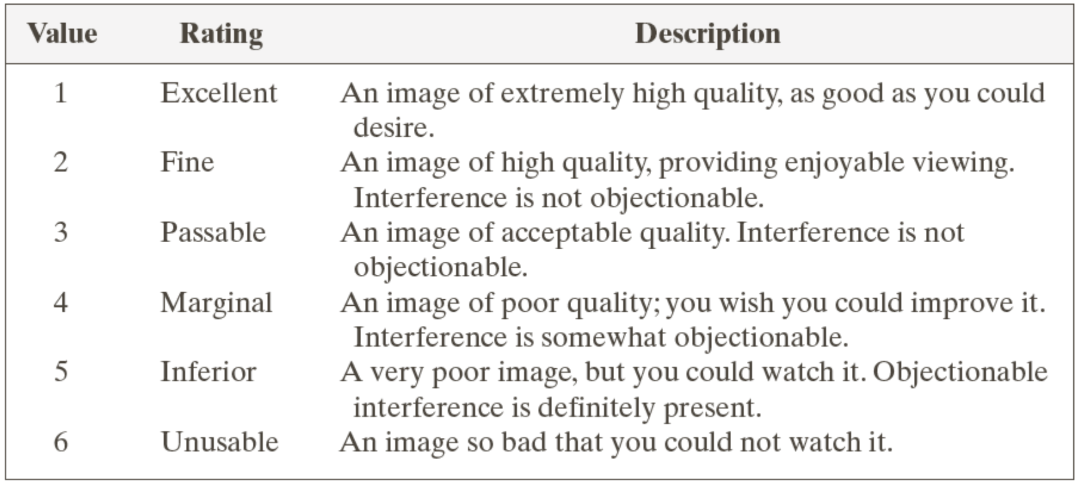
$$e_{\text{rms}} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x,y) - f(x,y) \right]^2 \right]^{1/2}$$

- The mean-square signal-to-noise ratio of the output image can be defined as

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

- The rms value of the signal-to-noise ratio of the output image can be obtained by taking square root of the above equation.

- Decompressed images are ultimately viewed by humans.
- So measuring image quality by the subjective evaluations of people is often more appropriate.
- This is called subjective fidelity criteria.



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.

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

- A subjective evaluation of the images using Table 8.2, however, might yield an
  - *excellent rating for (a),*
  - *a marginal rating for (b),*
  - *and an inferior or unusable rating for (c).*
- *Thus, using a subjective fidelity criteria, (b) is ranked ahead of (c).*

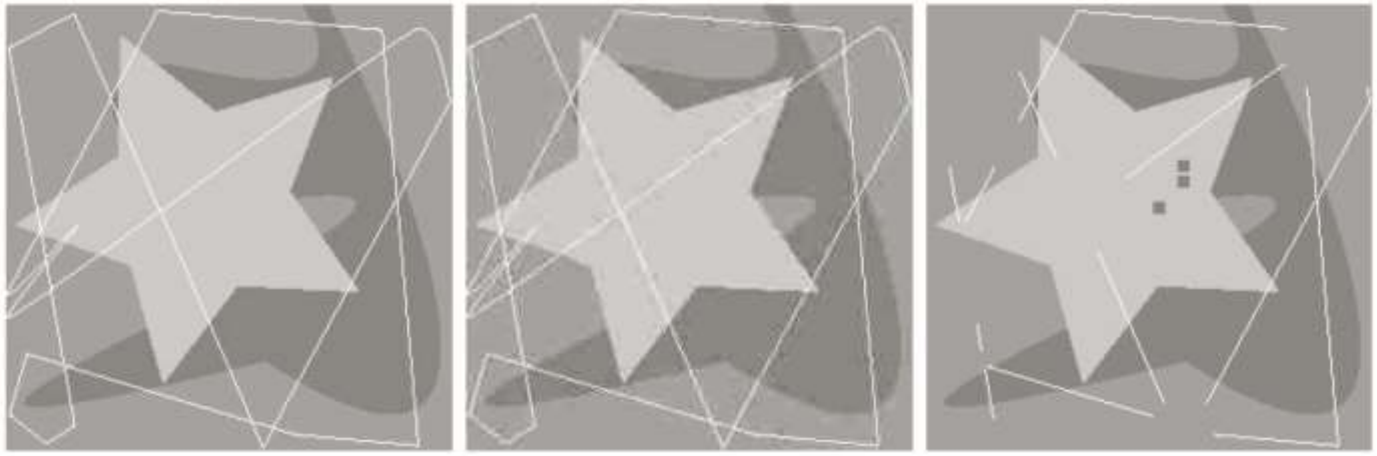
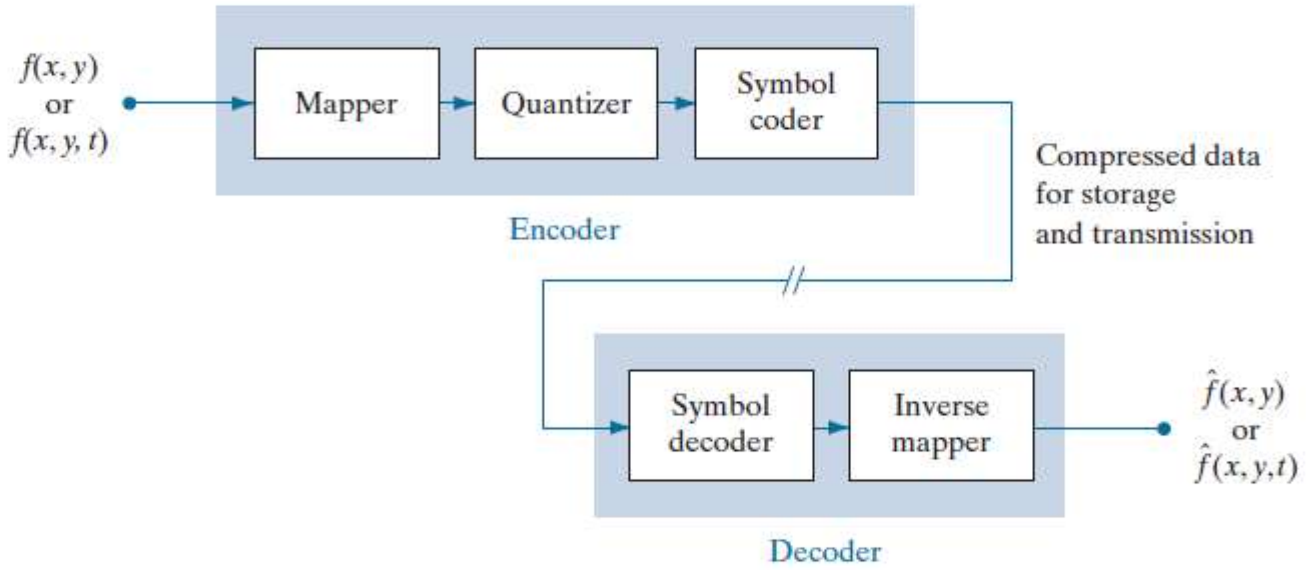


FIGURE 8.4 Three approximations of the image in Fig. 8.1(a).

Image Compression Models

- Image compression system is composed of two distinct functional components: an encoder and a decoder.
- Encoder performs compression and decoder performs decompression.
- Both the operations can be performed in software (eg. Browsers, image editors)or in a combination of hardware and firmware (eg. DVD players).
- A *codec is a device or program that is capable* of both encoding and decoding.

FIGURE 8.5 Functional block diagram of a general image compression system.



Encoding or Compression process

- In the first stage of the encoding process, a *mapper transforms f (x,…) 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.
- Run-length coding is an example of a mapping that normally yields compression in the first step of the encoding process.
- The mapping of an image into a set of less correlated transform coefficients (see Section 8.9) is an example of the opposite case (the coefficients must be further processed to achieve compression).
- In video applications, the mapper uses previous (and, in some cases, future) video frames to facilitate the removal of temporal redundancy.

Encoding or Compression process

- The *quantizer in Fig. 8.5 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. As noted earlier, this operation is irreversible.
- It must be omitted when error-free compression is desired.
- In video applications, the *bit rate of the encoded output is often measured (in bits/second),* and is used to adjust the operation of the quantizer so a predetermined average output rate is maintained.
- Thus, the visual quality of the output can vary from frame to frame as a function of image content.

Encoding or Compression process

- In the third and final stage of the encoding process, 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, 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.

The Decoding or Decompression Process

- The decoder of Fig. 8.5 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.
- In video applications, decoded output frames are maintained in an internal frame store (not shown) and used to reinsert the temporal redundancy that was removed at the encoder.

IMAGE FORMATS, CONTAINERS, AND COMPRESSION STANDARDS

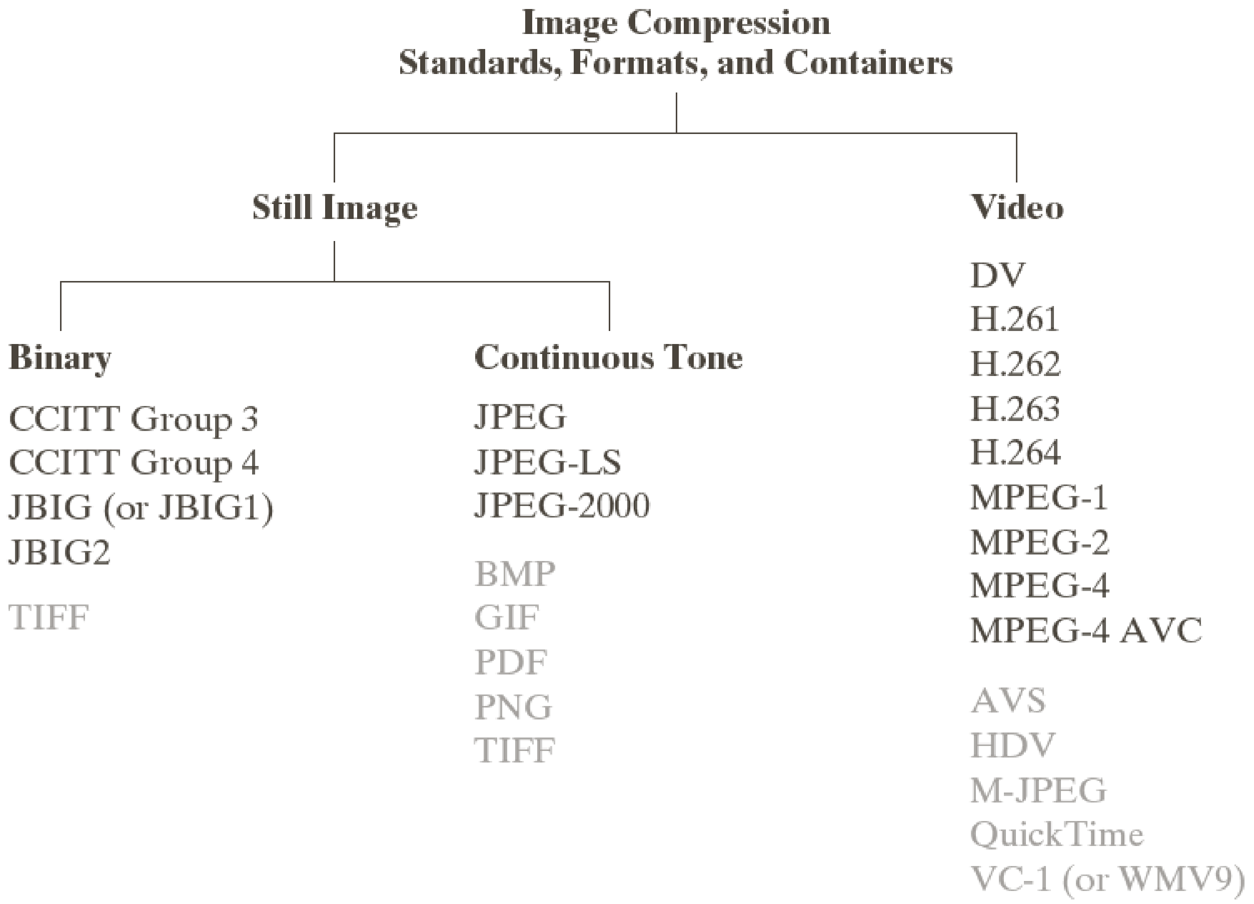
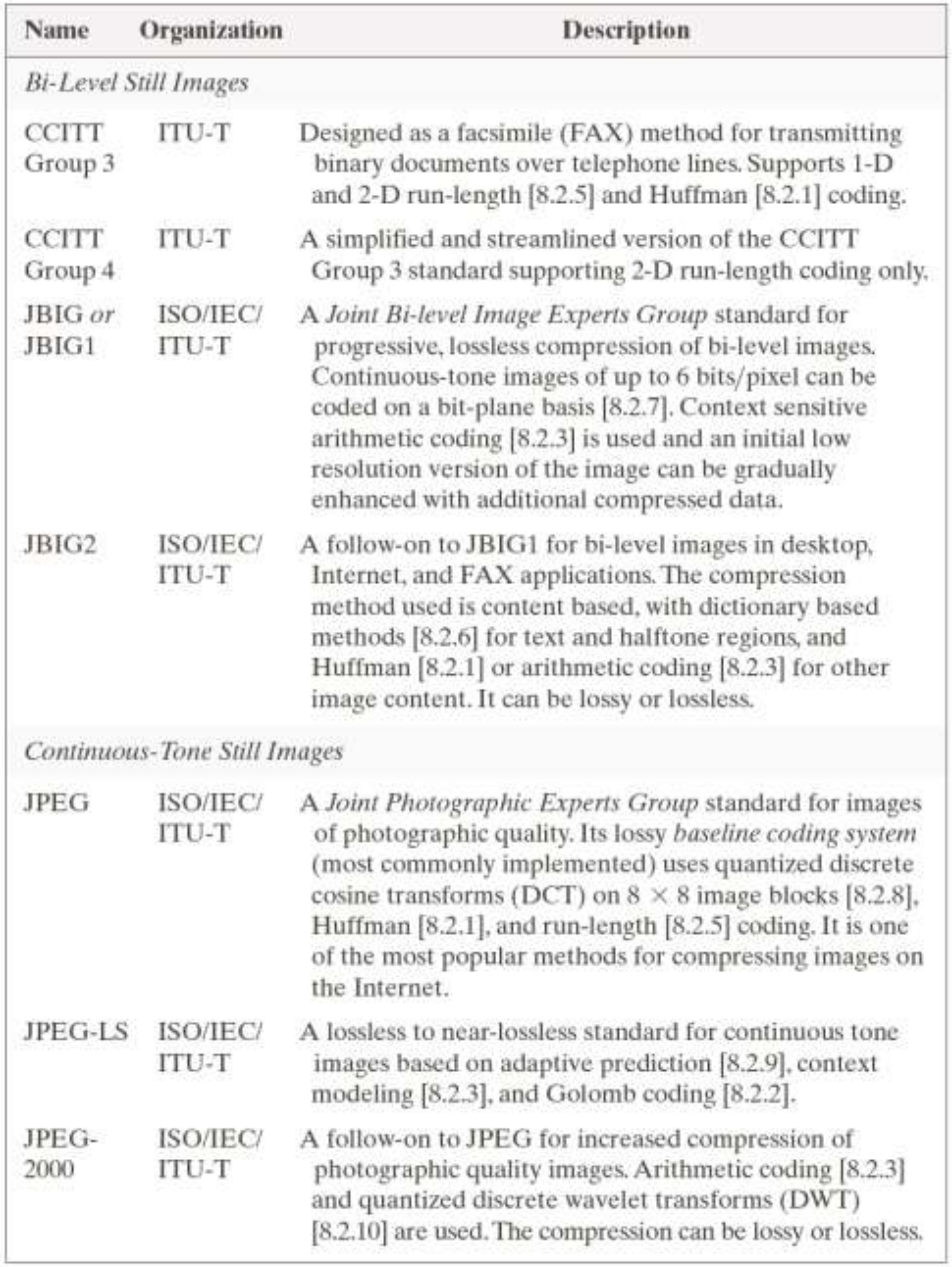


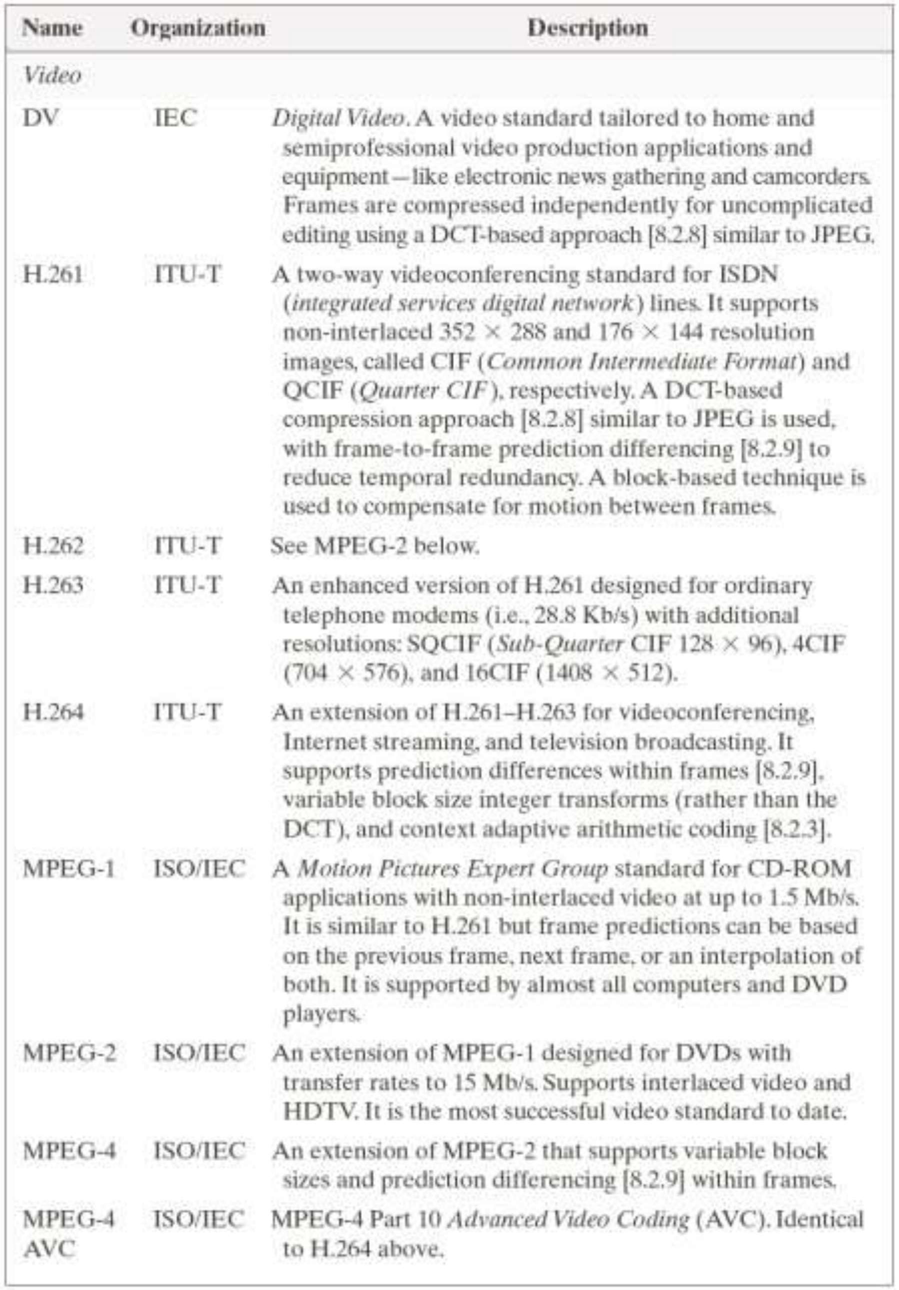
FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.



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 [8.2.5] and Huffman [8.2.1] 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 [8.2.7]. Context sensitive arithmetic coding [8.2.3] 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 [8.2.6] for text and halftone regions, and Huffman [8.2.1] or arithmetic coding [8.2.3] for other image content. It can be lossy or lossless.
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 8 × 8 image blocks [8.2.8], Huffman [8.2.1], and run-length [8.2.5] 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 [8.2.9], context modeling [8.2.3], and Golomb coding [8.2.2].
JPEG-2000	ISO/IEC/ ITU-T	A follow-on to JPEG for increased compression of photographic quality images. Arithmetic coding [8.2.3] and quantized discrete wavelet transforms (DWT) [8.2.10] are used. The compression can be lossy or lossless.

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

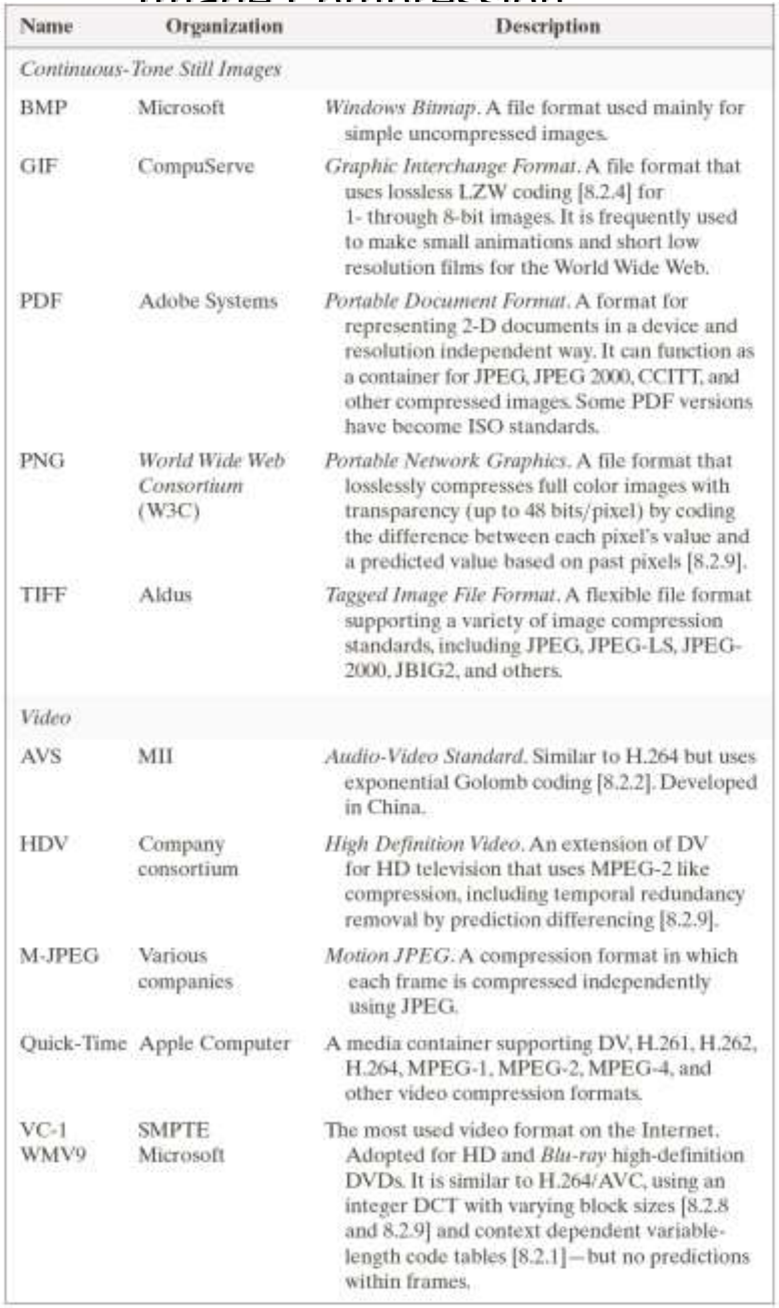
(*Continues*)



Name	Organization	Description
Video		
DV	IEC	*Digital Video*. A video standard tailored to home and semiprofessional video production applications and equipment—like electronic news gathering and camcorders. Frames are compressed independently for uncomplicated editing using a DCT-based approach [8.2.8] 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 [8.2.8] similar to JPEG is used, with frame-to-frame prediction differencing [8.2.9] 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, Internet streaming, and television broadcasting. It supports prediction differences within frames [8.2.9], variable block size integer transforms (rather than the DCT), and context adaptive arithmetic coding [8.2.3].
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 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 [8.2.9] within frames.
MPEG-4 AVC	ISO/IEC	MPEG-4 Part 10 *Advanced Video Coding* (AVC). Identical to H.264 above.

TABLE 8.3
(*Continued*)

Chapter 8
Image Compression



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 [8.2.4] for 1- through 8-bit images. It is frequently used to make small animations and short low resolution films for the World Wide Web.
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 [8.2.9].
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.
Video		
AVS	MII	*Audio-Video Standard*. Similar to H.264 but uses exponential Golomb coding [8.2.2]. Developed in China.
HDV	Company consortium	*High Definition Video*. An extension of DV for HD television that uses MPEG-2 like compression, including temporal redundancy removal by prediction differencing [8.2.9].
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 [8.2.8 and 8.2.9] and context dependent variable-length code tables [8.2.1]—but no predictions within frames.

TABLE 8.4 Popular image compression standards, file formats, and containers, not included in Table 8.3.

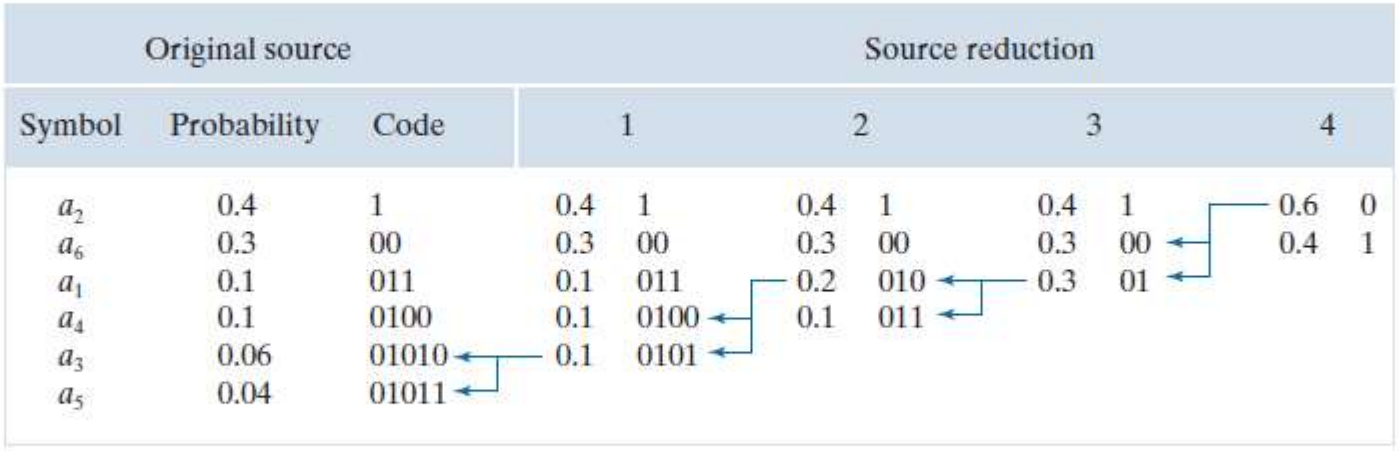
Huffman Coding

- One of the most popular technique for removing coding redundancy.
- It yields the smallest possible number of code symbols per source symbol.
- The source symbol may be either the intensities of an image or the output of an intensity mapping operation(pixel differences, run lengths, and so on).

Steps in Huffman coding

- The first step is to create a series of source reductions.
  - The source symbols are arranged in order according to their probabilities.
  - Combine the lowest probability symbols into a single symbol that replaces them in the next source reduction.
  - This process is repeated until a reduced source with two symbols is reached.

- The second step is to code each reduced source.
  - Starting with the smallest source and working back to the original source.
  - The minimum length binary code for a two symbol source are the symbols 0 and 1.
  - As the reduced source symbol was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0 and 1 are arbitrarily appended to each to distinguish from each other.
  - This operation is repeated for each reduced source until the original source is reached.



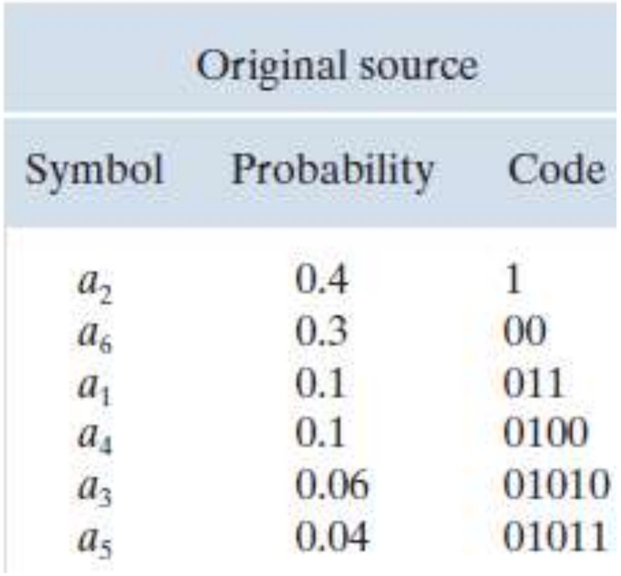
Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

- The average length of this code is

L_{avg} = (0.4)(1) +(0.3)(2)+(0.1)(3)+(0.1)(4)+(0.06)(5)+(0.04)(5)

= 2.2 bits/pixel.

- And the entropy of the source is 2.14 bits/symbol.
- 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 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.



Original source		
Symbol	Probability	Code
a_2	0.4	1
a_6	0.3	00
a_1	0.1	011
a_4	0.1	0100
a_3	0.06	01010
a_5	0.04	01011

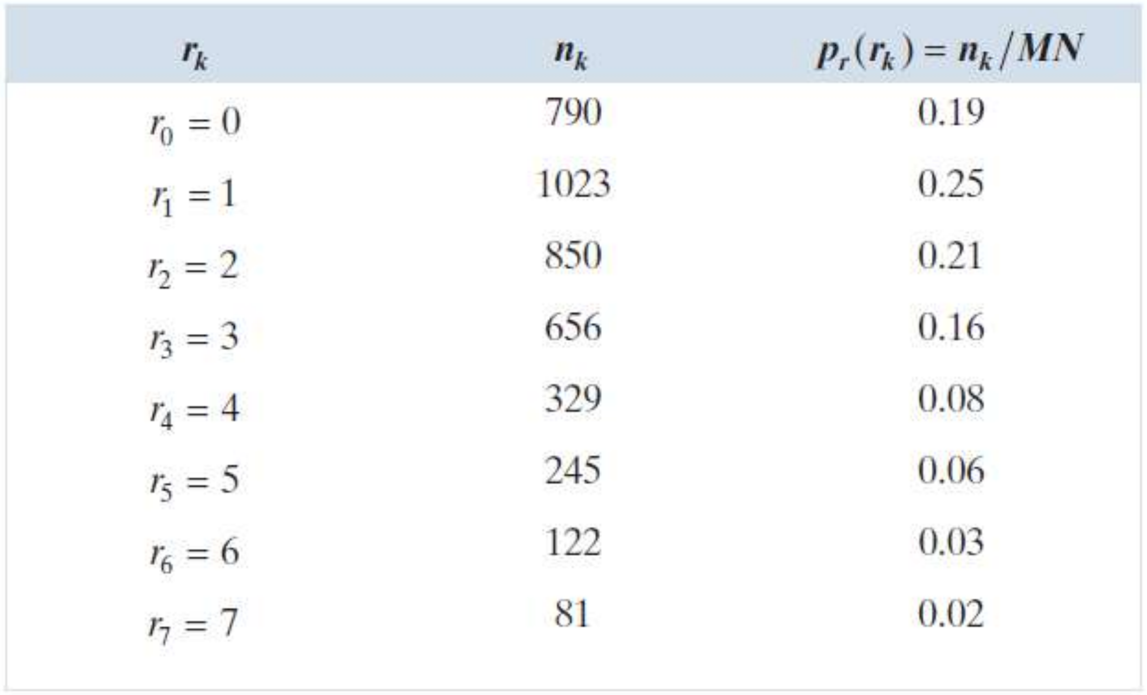
- Decode the string 010100111100 using the above Huffman coding table.

Ans: a3a1a2a2a6

- Encode the message $a_1a_2a_6a_3$ using above Huffman coding table.

Ans: 01110001010

- Compute Huffman codes for the source symbols r_k



r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

Consider the simple 3X7, 8-bit image:

12 12 12 95 69 43 43

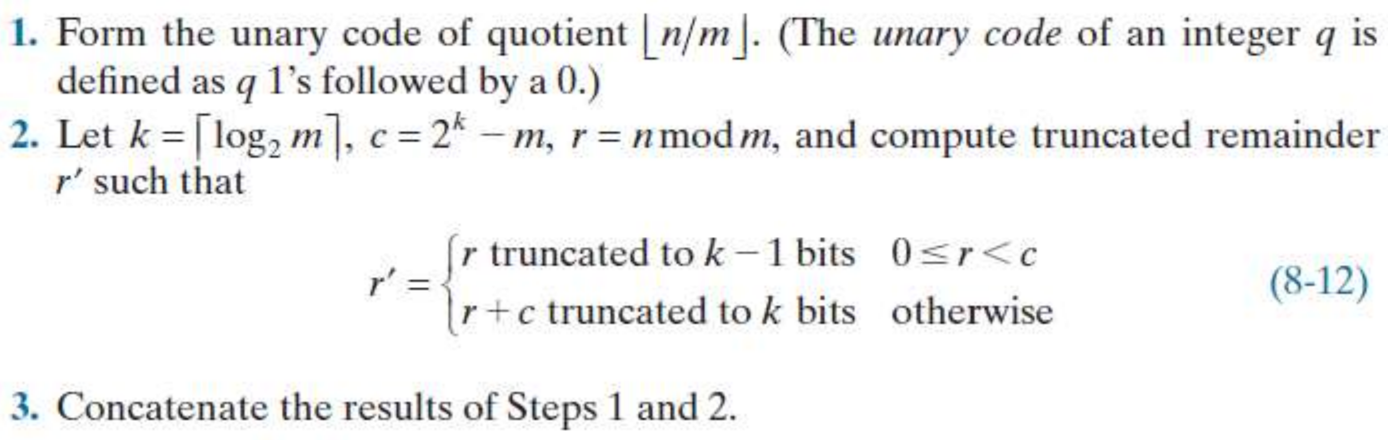
12 12 12 95 69 43 43

12 12 12 95 69 43 43

Compute Huffman codes for the source symbols in the image

Golomb Coding

- It is used to code nonnegative integer inputs with exponentially decaying probability distributions.
- Given a nonnegative integer *n* and a positive integer divisor *m* > 0, the Golomb code of *n* with respect to *m*, denoted $G_m(n)$, is constructed as follows:



1. Form the unary code of quotient $\lfloor n/m \rfloor$. (The *unary code* of an integer q is defined as q 1's followed by a 0.)
2. Let $k = \lceil \log_2 m \rceil$, $c = 2^k - m$, $r = n \bmod m$, and compute truncated remainder r' such that

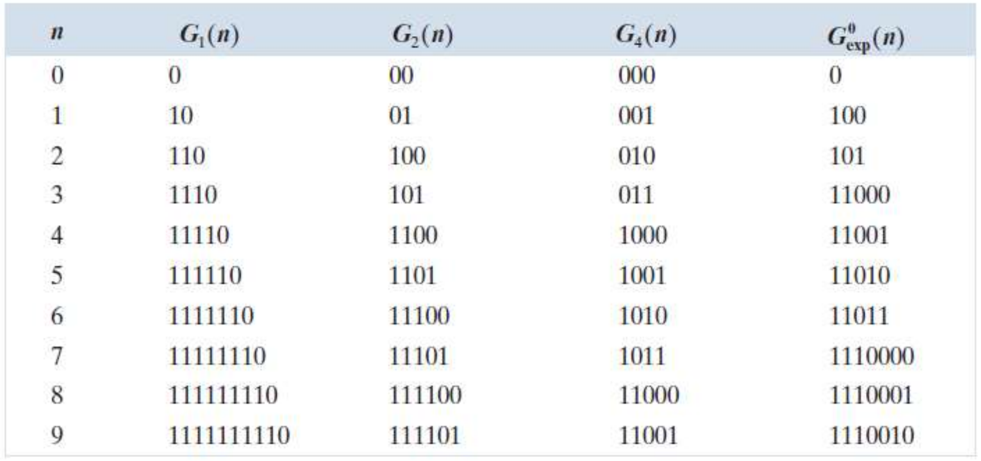
$$r' = \begin{cases} r \text{ truncated to } k-1 \text{ bits} & 0 \le r < c \\ r + c \text{ truncated to } k \text{ bits} & \text{otherwise} \end{cases} \qquad (8\text{-}12)$$

3. Concatenate the results of Steps 1 and 2.

Chapter 8
Image Compression

To compute $G_4(9)$, for example, begin by determining the unary code of the quotient $\lfloor 9/4 \rfloor = \lfloor 2.25 \rfloor = 2$, which is 110 (the result of Step 1). Then let $k = \lceil \log_2 4 \rceil = 2$, $c = 2^2 - 4 = 0$, and $r = 9 \bmod 4$, which in binary is $1001 \bmod 0100$ or 0001. In accordance with Eq. (8-12), r' is then r (i.e., 0001) truncated to 2 bits, which is 01 (the result of Step 2). Finally, concatenate 110 from Step 1 and 01 from Step 2 to get 11001, which is $G_4(9)$.

- For the special case of $m = 2^k$, $c = 0$ and $r' = r = n \bmod m$ truncated to k bits for all n.
- The divisions required to generate the resulting Golomb codes become binary shift operations.
- These computationally simpler codes are called Golomb-Rice codes or Rice codes.
- The first three columns in the below table are Rice codes.



n	$G_1(n)$	$G_2(n)$	$G_4(n)$	$G^0_{\exp}(n)$
0	0	00	000	0
1	10	01	001	100
2	110	100	010	101
3	1110	101	011	11000
4	11110	1100	1000	11001
5	111110	1101	1001	11010
6	1111110	11100	1010	11011
7	11111110	11101	1011	1110000
8	111111110	111100	11000	1110001
9	1111111110	111101	11001	1110010

TABLE 8.5 Several Golomb codes for the integers 0 – 9.

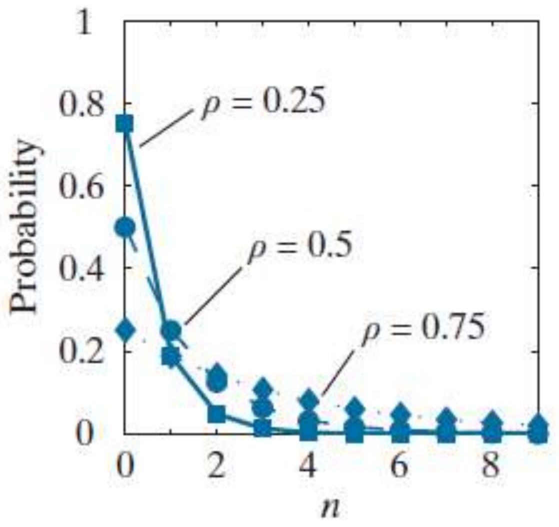
- Golomb codes are optimal when the integers to be represented are geometrically distrubuted with probability mass function(PMF)

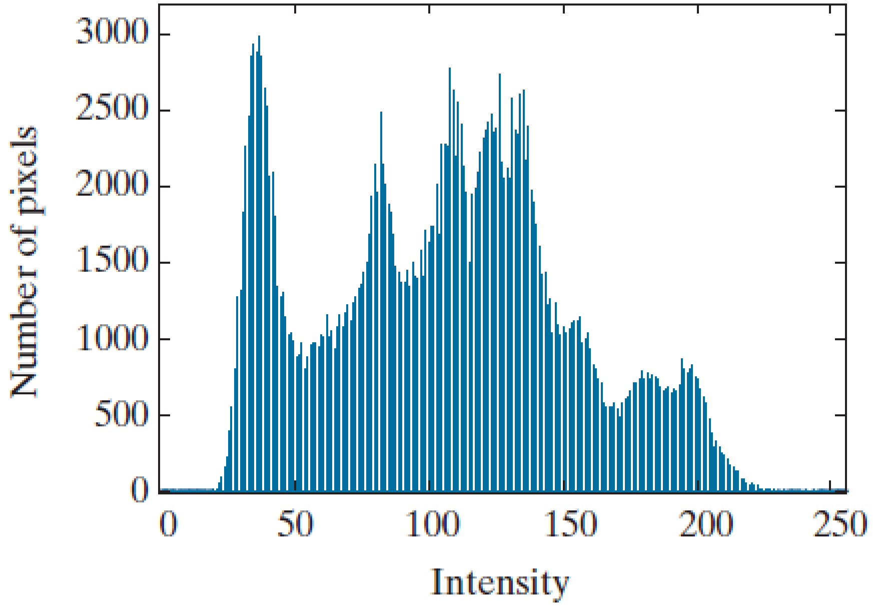
$$p(n) = (1-\rho)\rho^n$$

for some $0 < \rho < 1$.

- Golomb code provides the shortest average code length when

$$m = \left\lceil \frac{log_2(1+\rho)}{log_2(\frac{1}{\rho})} \right\rceil$$



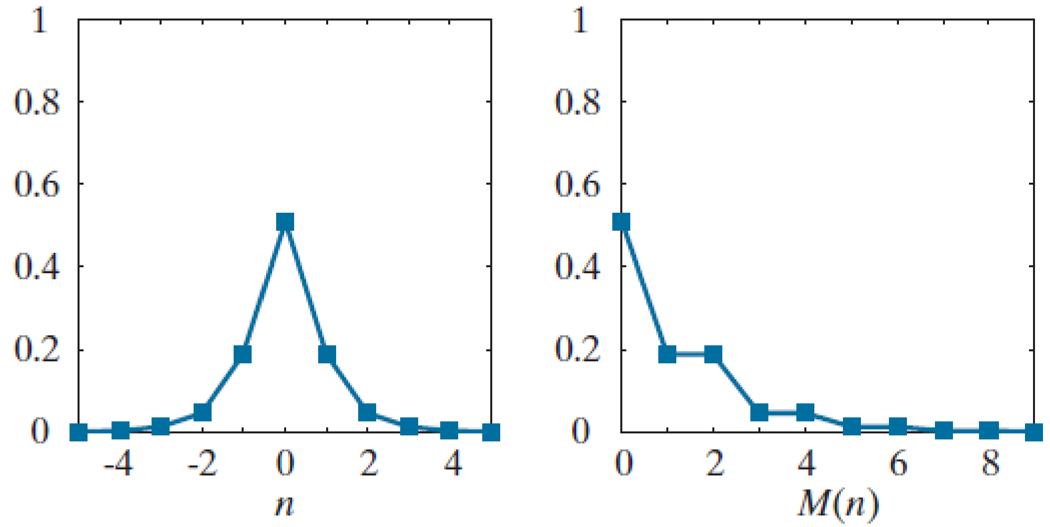


- Probabilities of intensities in an image are unlikely to match the above mentioned PMF.
- So Golomb codes are seldom used for coding intensities.

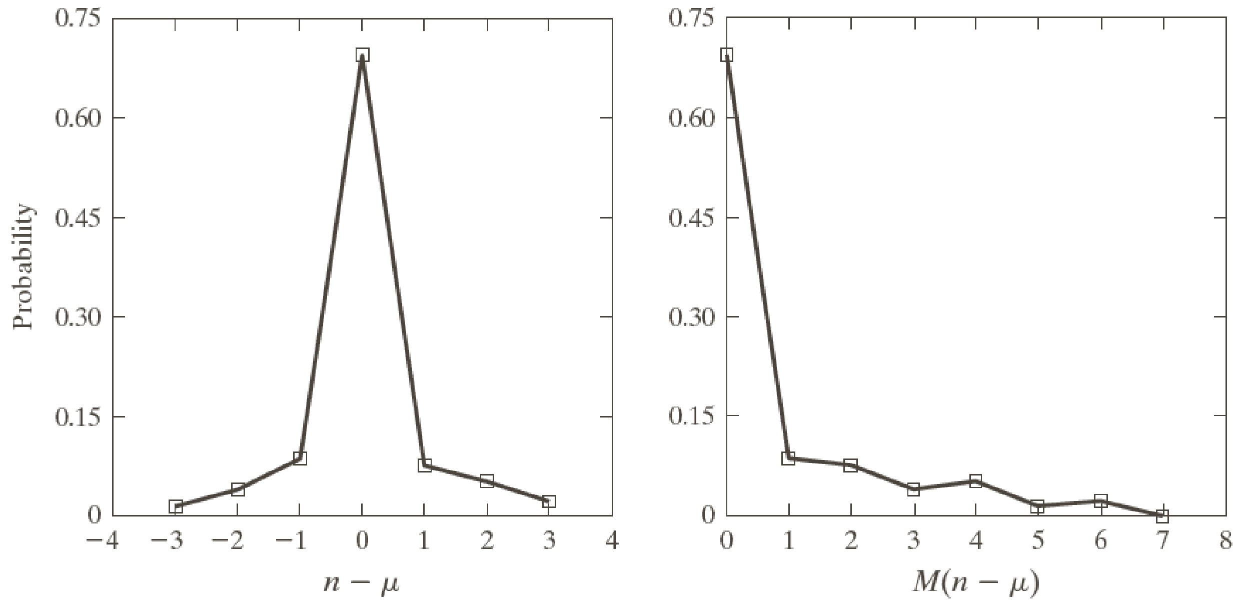
- Probabilities of intensity differences are likely to match the above PMF.
- So Golomb coding is useful for encoding intensity differences.
- To handle negative differences in Golomb coding , a mapping like

$$M(n) = \begin{cases} 2n & n \geq 0 \\ 2|n| - 1 & n < 0 \end{cases}$$

is used.



- Using the above mapping function, the two sided PMF can be transformed into the one-sided PMF as shown in above figure.
- The negative integers are mapped into the odd positive integers.
- Now M(n) can be encoded using an appropriate Golomb-Rice code

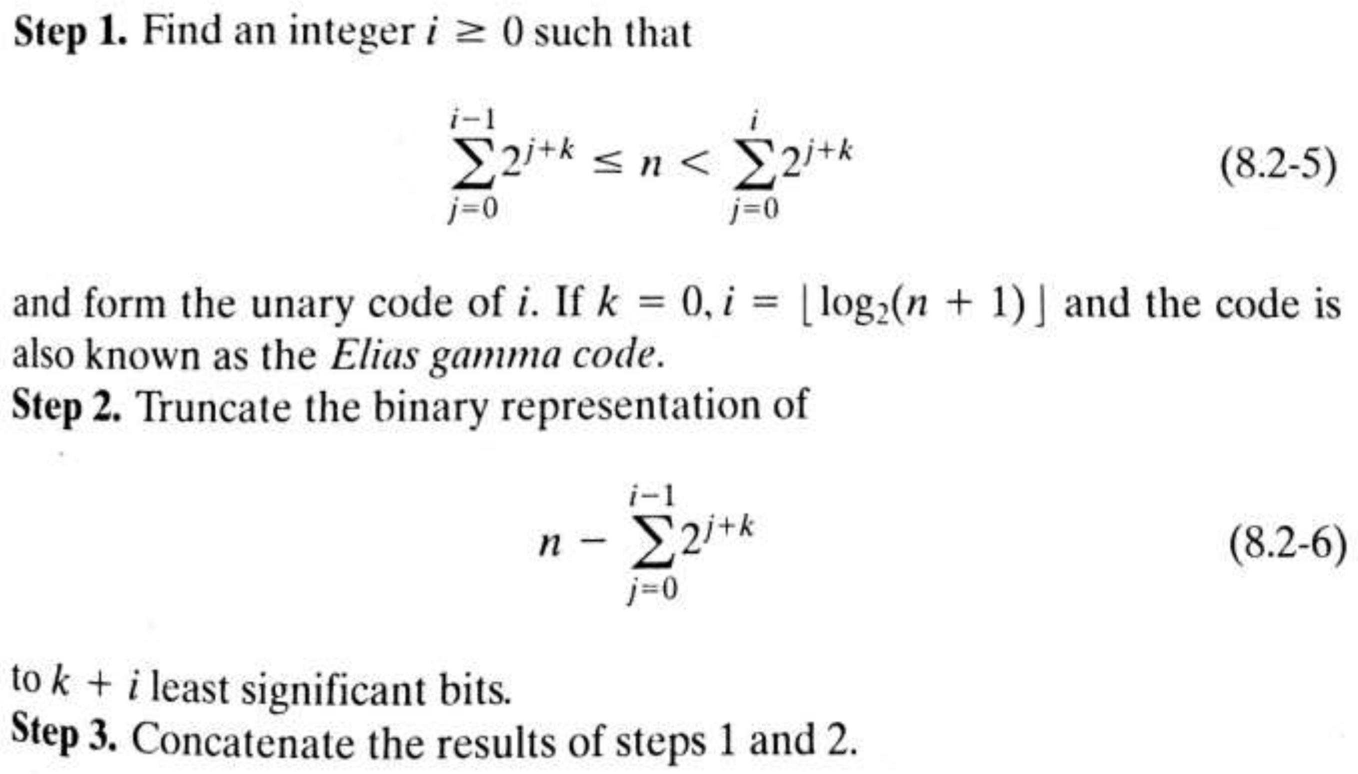


a b

FIGURE 8.11 (a) The probability distribution of the image in Fig. 8.1(c) after subtracting the mean intensity from each pixel, and (b) a mapped version of (a) using Eq. (8.2-4).

Exponential Golomb code

- Exponential Golomb codes are useful for the encoding of run lengths.
- Because both short and long runs are encoded efficiently.
- An order-k exponential Golomb code is computed as follows.



Step 1. Find an integer $i \geq 0$ such that

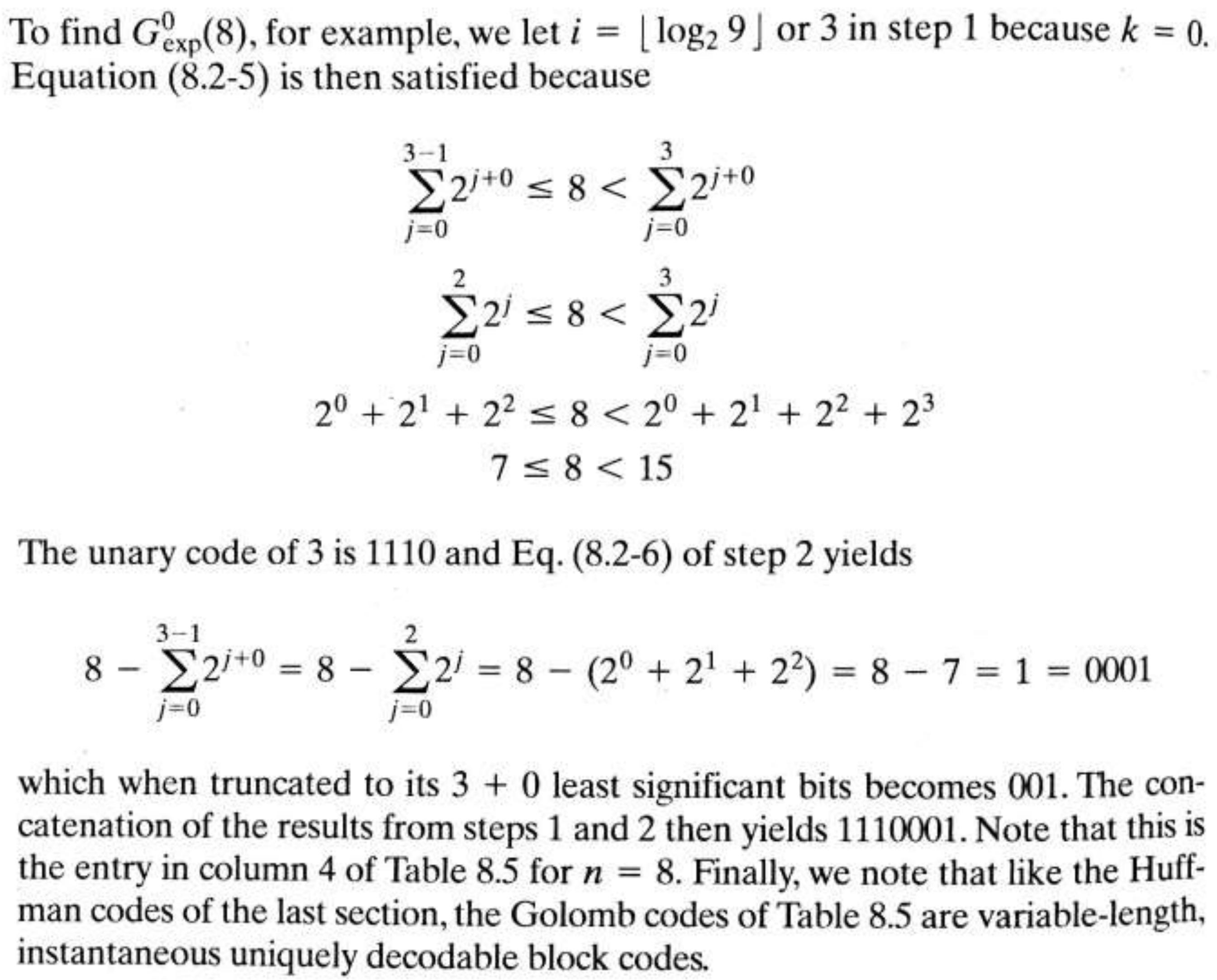
$$\sum_{j=0}^{i-1} 2^{j+k} \leq n < \sum_{j=0}^{i} 2^{j+k} \qquad (8.2\text{-}5)$$

and form the unary code of i. If $k = 0$, $i = \lfloor \log_2(n+1) \rfloor$ and the code is also known as the *Elias gamma code*.
Step 2. Truncate the binary representation of

$$n - \sum_{j=0}^{i-1} 2^{j+k} \qquad (8.2\text{-}6)$$

to $k + i$ least significant bits.
Step 3. Concatenate the results of steps 1 and 2.

Example for Exponential Golomb Coding



To find $G_{\text{exp}}^{0}(8)$, for example, we let $i = \lfloor \log_2 9 \rfloor$ or 3 in step 1 because $k = 0$. Equation (8.2-5) is then satisfied because

$$\sum_{j=0}^{3-1} 2^{j+0} \le 8 < \sum_{j=0}^{3} 2^{j+0}$$

$$\sum_{j=0}^{2} 2^{j} \le 8 < \sum_{j=0}^{3} 2^{j}$$

$$2^0 + 2^1 + 2^2 \le 8 < 2^0 + 2^1 + 2^2 + 2^3$$

$$7 \le 8 < 15$$

The unary code of 3 is 1110 and Eq. (8.2-6) of step 2 yields

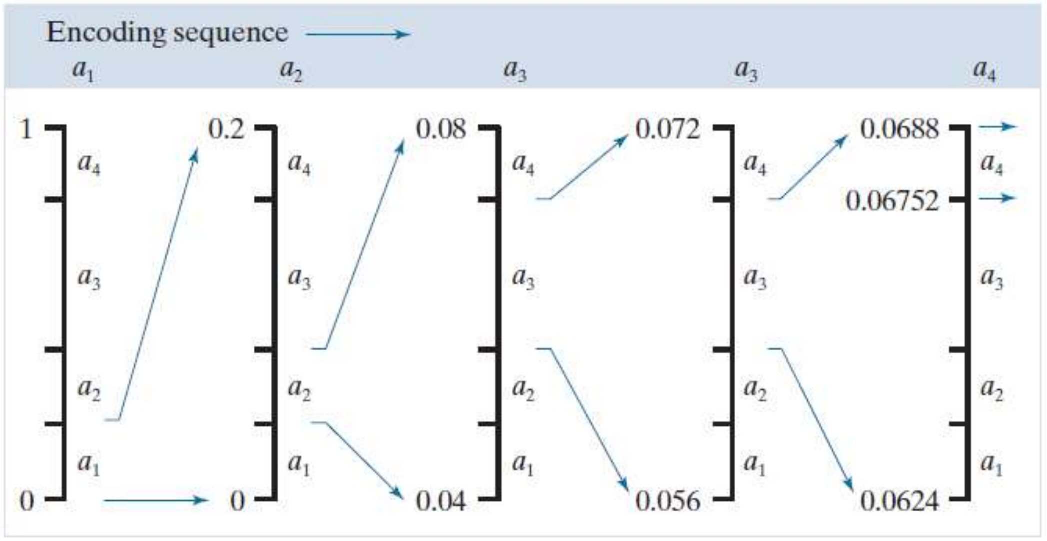
$$8 - \sum_{j=0}^{3-1} 2^{j+0} = 8 - \sum_{j=0}^{2} 2^{j} = 8 - (2^0 + 2^1 + 2^2) = 8 - 7 = 1 = 0001$$

which when truncated to its $3 + 0$ least significant bits becomes 001. The concatenation of the results from steps 1 and 2 then yields 1110001. Note that this is the entry in column 4 of Table 8.5 for $n = 8$. Finally, we note that like the Huffman codes of the last section, the Golomb codes of Table 8.5 are variable-length, instantaneous uniquely decodable block codes.

Arithmetic Coding

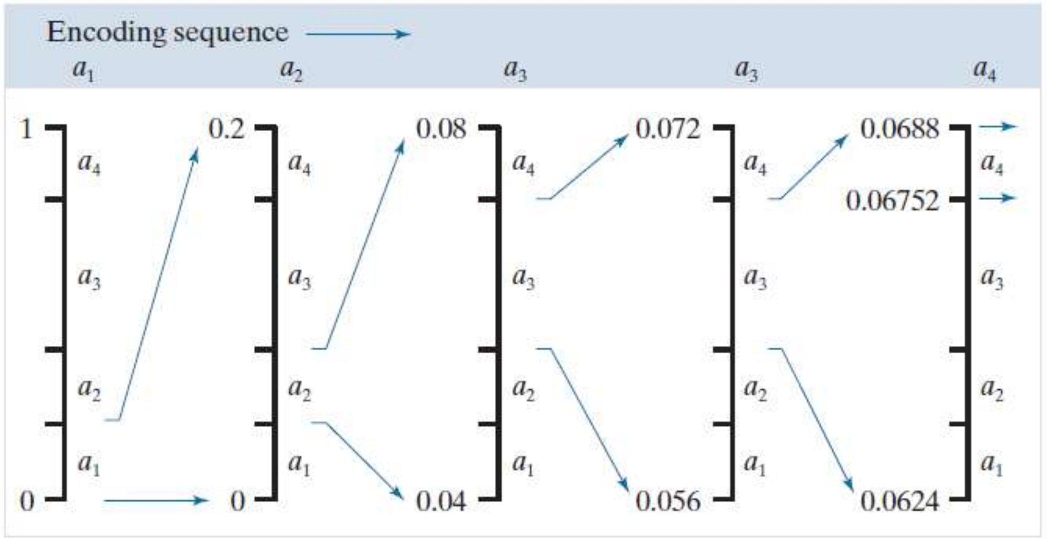
- Arithmetic coding generates nonblock codes.
- An entire sequence of source symbols (or message) is assigned a single arithmetic code word.
- The code word itself defines an interval of real numbers between 0 and 1.
- As the number of symbols in the message increases, the interval used to represent it becomes smaller and the number of information units(say, 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 occurence.

- The below figure illustrates the basic arithmetic coding process
- A five symbol sequence or message , a1a2a3a3a4, from a four-symbol source is coded.
- The table shows the probabilities of each source symbol.
- As per the table the interval [0,1) is subdivided initially into four regions based on the probabilities of each source symbol.
- Because a1 is the first symbol of the message its interval [0,0.2) is further expanded accordance with their probability.



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)

- The next symbol in the message is a2 so its interval is expanded according to the probabilities and so on until the last symbol in the message.
- The final interval is [0.06752,0.0688).
- Any number with in this subinterval- for example, 0.068 – can be used to encode the message.
- The decoding is just the reverse process.

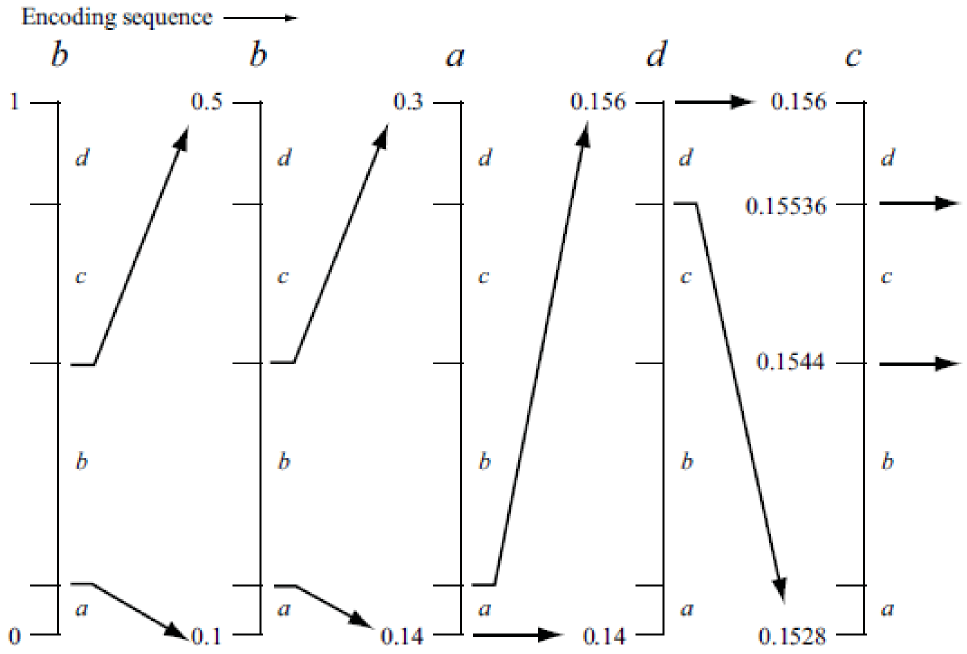


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

Example 2

- Encode the sequence “ bbadc” using arithmetic coding.

Source Symbol	Probability	Initial Sub interval
a	0.1	[0 , 0.1)
b	0.4	[0.1, 0.5)
c	0.3	[0.5, 0.8)
d	0.2	[0.8, 1.0)



Arithmetic Decoding

- Decode the message 0.23355 using arithmetic decoding for the given probability distribution

Symbol	Probability	Range
a	0.2	[0.0, 0.2)
e	0.3	[0.2, 0.5)
i	0.1	[0.5, 0.6)
o	0.2	[0.6, 0.8)
u	0.1	[0.8, 0.9)
!	0.1	[0.9, 1.0)

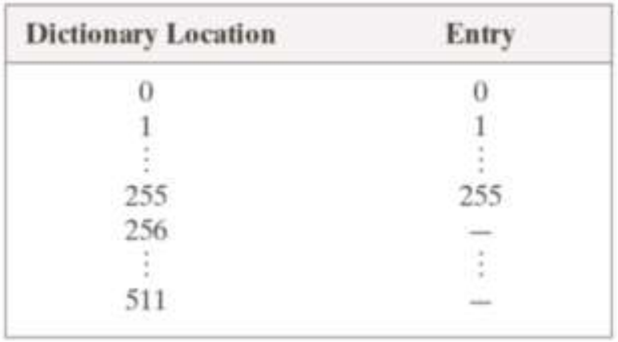
The arithmetic decoding process is the reverse of the encoding procedure. Start by dividing the [0, 1) interval according to the symbol probabilities. This is shown in Table P8.18. The decoder immediately knows the message 0.23355 begins with an "e", since the coded message lies in the interval [0.2, 0.5). This makes it clear that the second symbol is an "a", which narrows the interval to [0.2, 0.26). To further see this, divide the interval [0.2, 0.5) according to the symbol probabilities. Proceeding like this, which is the same procedure used to code the message, we get "eaii!".

LZW (Lempel-Ziv-Welch) Coding

- This coding method **removes both coding redundancy and spatial redundancy.**
- Assigns fixed length code words to variable length sequences of source symbols.
- The key feature of LZW coding is that it requires no a priori knowledge of the probability of occurrence of the symbols to be encoded.

EXAMPLE 8.7:
LZW coding.

- A codebook or dictionary containing the source symbols to be coded is constructed.
- For 8-bit monochrome images, the first 256 words of the dictionary are assigned to intensities 0,1,2,....,255.
- As the encoder sequentially examines image pixels, intensity sequences that are not in the dictionary are placed in the next unused locations.
- For example, if first two pixels of the image are white, the sequence “255-255” is assigned to location 256.
- The next time that two consecutive white pixels are encountered, code word 256 is used to represent them.



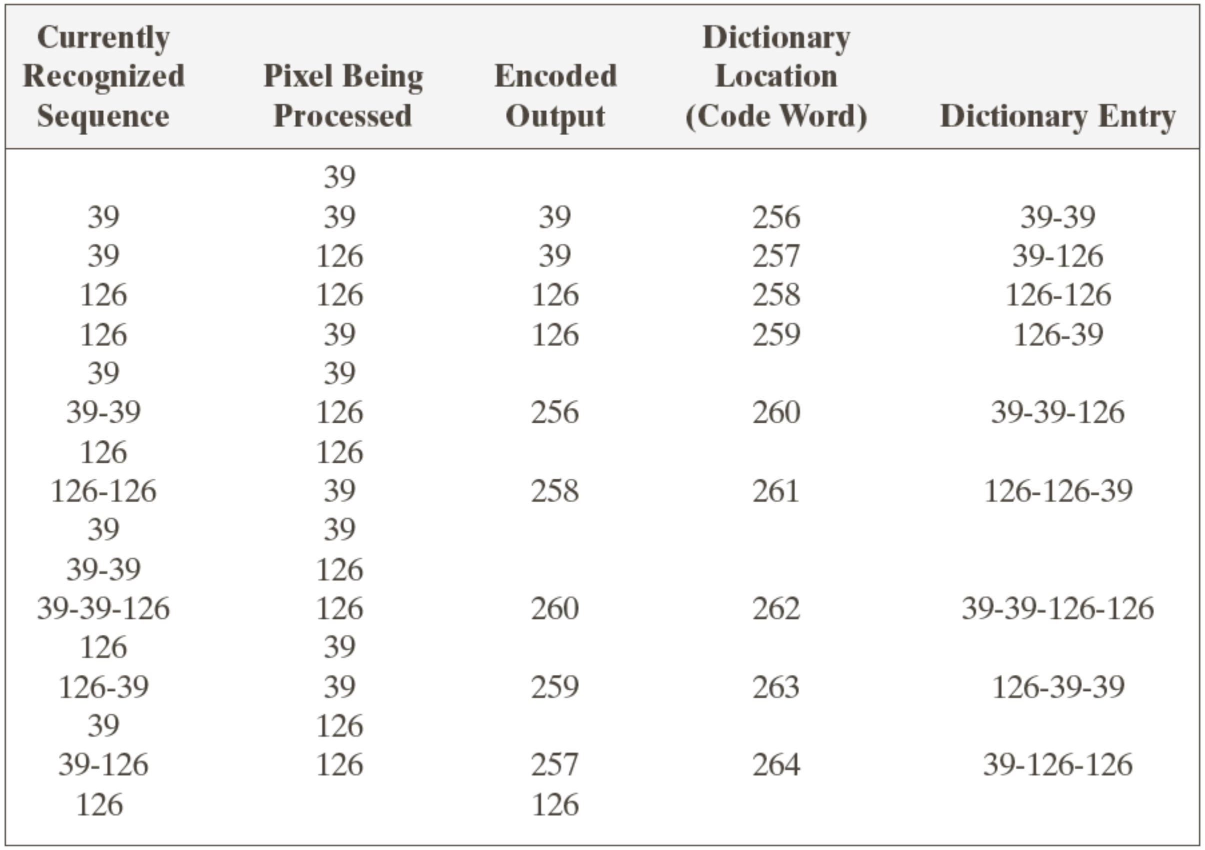
Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

LZW encoding process

- The image is encoded by processing its pixels in a left-to-right, top-to-bottom manner.
- Each successive intensity value is concatenated with a variable called "currently recognized sequence" which is initially null or empty.
- **The dictionary is searched for each concatenated sequence and**
- **case-1 : if found**
  - **the currently recognized sequence is replaced by the newly concatenated and recognized sequence. No output codes are generated nor the dictionary is altered.**
- **case-2: if not found**
  - **the address of the currently recognized sequence is output as the next encoded value.**
  - **The concatenated but unrecognized sequence is added to the dictionary, and the currently recognized sequence is initialized to the current pixel value**

- Compute the dictionary and encoding for the following 4X4, 8-bit image:

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126



Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

TABLE 8.7 LZW coding example.

Run-Length Coding

- Images with repeating intensities along their rows(or columns) can often be compressed by representing runs of identical intensities as run-length pairs.
- Each run-length pair specifies start of a new intensity and the number of consecutive pixels that have that intensity .
- **Thus compression is achieved by eliminating a simple form of spatial redundancy**.
- When there are few(or no) runs of identical pixels, run-length encoding results in data expansion.
- **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.

- For binary images, each image row can be represented by a sequence of lengths only-rather than length-intensity pairs.
- The basic idea is to code each continuous group(i.e run) of 0s and 1s 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
  2. To assume that each row begins with a white run, whose run length may in fact be zero.

- For example, the run-length coding of the image

  0 0 0 1 1

  1 1 0 0 1

  0 0 0 0 1

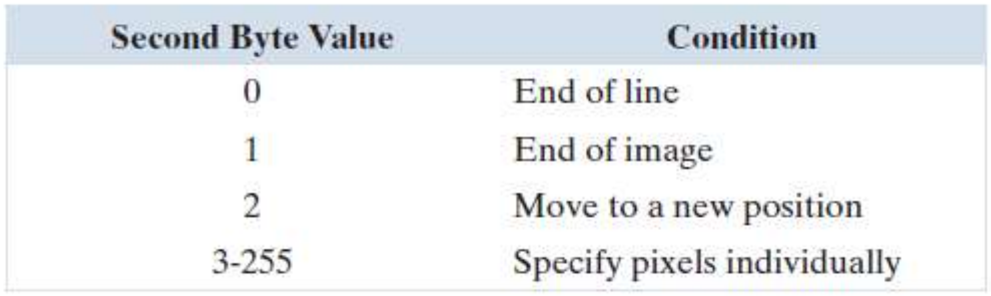
  using first convention is { {(3,0) , 2 }, { (2,1), 2,1}, { (4,0), 1 } }.

  using second convention is { {0,3,2}, {2,2,1}, { 0,4,1 } }
- Additional compression can be achieved by variable length coding the run lengths themselves.

Run length encoding 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. Either mode can occur anywhere in the image.
- 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, as* shown in Table 8.9.

TABLE 8.9 BMP absolute coding mode options. In this mode, the first byte of the BMP pair is 0.



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 encoding in the BMP File Format

- When the second byte is 0 or 1, the end of a line or the end of the image has been reached.
- If it is 2, the next two bytes contain unsigned horizontal and vertical offsets to a new spatial position (and pixel) in the image.
- If the second byte is between 3 and 255, it specifies the number of uncompressed pixels that follow with each subsequent byte containing the color index of one pixel.
- The total number of bytes must be aligned on a 16-bit word boundary.