Computer Vision

Course 11

Code Redundancy

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

$$p(r_k) = \frac{n_k}{M \cdot N} \ , \ k = 0,1,2,...,L-1$$

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:

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

The total number of bits required to represent an $M \times N$ image is MNL_{avg}. If the intensities are represented using a natural m-bit fixed-length code, $L_{avg} = m$.

Consider image in Figure 8.1 (a) ($M=N=256$) and the coding Table 8.1.

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.

For Code 1 $L_{avg} = 8$. For Code 2 we have:

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

The total number of bits needed to represent the entire image is $MNL_{avg} = 256 \times 256 \times 1.81 = 118.621$.

$$C = \frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} = \frac{8}{1.81} \approx 4.42$$

$$R = 1 - \frac{1}{C} = 1 - \frac{1}{4.42} = 0.774$$

Thus, ***77.4%*** of the data in the original 8-bit 2-D intensity array is redundant.

The compression achieved by code 2 results from assigning fewer bits to the more probable intensity values than to the less probable ones, thus resulting a *variable-length code*. The best *fixed-length code* that can be assigned to the intensities

of the image in Figure 8.1(a) is the natural ***2***-bit counting sequence ***{00, 01, 10, 11}*** but the resulting compression is only ***C=8/2=4:1*** which is about ***10%*** less than the ***4.42:1*** compression of the variable-length code.

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. Most

images are composed of objects that have a regular and somewhat predictable morphology (shape) and reflectance, and are sampled so that the objects being depicted are much larger than the picture elements. For most images, certain intensities are more probable than others (that is, the histograms of most images are not uniform). A natural binary encoding assigns the same number of bits to both the most and least probable values, failing to minimize the value of L_{avg} and resulting in coding redundancy.

Course 11

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.)

Spatial and Temporal Redundancy

Consider the computer-generated image in Figure 8.1(b). In the corresponding 2-D intensity array:

- All 256 intensities are equally probable
- Because the intensity of each line was selected randomly, its pixels are independent of one another in the vertical direction

- Because the pixels along each line are identical, they are maximally correlated (completely dependent on one another) in the horizontal direction.

The first observation tells us that this image cannot be compressed by variable-length coding alone. Observation 2 and 3 reveal a significant spatial redundancy that can be eliminated, for instance, by representing this 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. A run-length based representation compresses the original 2-D, 8-bit intensity array by

$$C = \frac{256 \times 256 \times 8}{256 \times 2 \times 8} = 128:1$$

Each 256-pixel line of the original representation is replaced by a single 8-bit intensity value and length 256 in the run-length representation.

In most images, pixels are correlated spatially (in both x and y) and in time (in case of video sequences). Because most

pixels' 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 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. Transformations of this type are called *mappings*. A mapping is said to be *reversible* if the pixels of the original 2-D

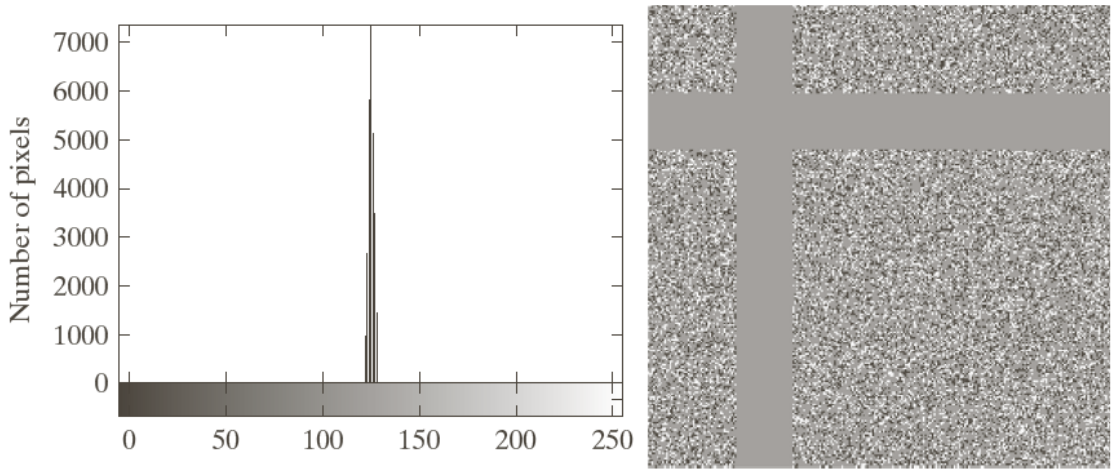
intensity array can be reconstructed without error from the transformed data set; otherwise the mapping is said to be *irreversible*.

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 DIP, information ignored by the system which uses the image (human eye, computer programs) are obvious candidates for omission. Thus, the computer-generated image in Figure

8.1(c), because it appears to be a homogeneous field of grey, can be represented by its average intensity alone – a single 8-bit value. The original ***256×256×8*** bit intensity array is reduced to a single byte; the resulting compression is ***65.536:1***.

Figure 8.3(a) shows the histogram of the image in Figure 8.1(c) – there are some intensity values (125 through 131) actually present. The human visual system averages these intensities, perceives only the average value, and ignores the small changes in intensity that are present in this case.



a b

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

Figure 8.3(b), a histogram equalized version of the image in Figure 8.1 (c), makes the intensity changes visible and reveals two previously undetected regions of constant intensity.

If the image in Figure 8.1 (c) is represented by its average value alone, this 'invisible' structure is lost.

The kind of redundancy can be eliminated 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* (mapping of a broad range of input values to a limited number of output values). Because information is lost, quantization is an irreversible operation.

Measuring Image Information

Problem: is there a minimum amount of data that is sufficient to describe an image without losing information? The answer is given by *information theory*.

The generation of information can be modelled as a probabilistic process. 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*** (the events always occurs), ***I(E)=0*** and no information is attributed to it (because no uncertainty is associated to the event, no information would be transferred by communicating that the event has occurred).

If the base ***m*** logarithm is used, the measurement is said to be in ***m***-ary units. For ***m=2*** the unit of information is the *bit*.

$P(E)=\frac{1}{2} \rightarrow I(E)=-\log\frac{1}{2}=1$ (when only one of two possible equally likely events occur, the amount of information is 1 bit)

Given a source of statistically independent random events from a discrete set of possible events $\{a_1, a_2, ..., a_J\}$ with associated probabilities $\{P(a_1), P(a_2), ..., P(a_J)\}$, 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 are called *source symbols*. Because they are statistically independent, the source itself is called *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. The intensity source's entropy becomes:

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

Formula (H) is the average information per intensity output of the imaginary intensity source in bits. It is not possible to code the *intensity values* of the imaginary source (and thus the sample image) with fewer than $\tilde{H}$ bits/pixel.

The entropy of the image in Figure 8.1(a) can be estimated as follows:

$$\tilde{H} = -\left[0.25\log_2 0.25 + 0.47\log_2 0.47 + 0.25\log_2 0.25 + 0.03\log_2 0.03\right]$$

$$\tilde{H} \approx 1.6614 \text{ bits/pixel}$$

In a similar manner, the entropy of the images in Figure 8.1(b) and (c) are estimated to be ***8*** bits/pixel and ***1.566*** bits/pixel, respectively. Note that the image in Figure 8.1 (a) appears to have the most visual information, but has almost the lowest computed entropy. The obvious conclusion is that the amount of entropy and thus the information in an image is not an intuitive notion.

Fidelity Criteria

When "irrelevant information" is removed, information is lost. In this case, we need a means of quantifying the nature of the lost. Two types of criteria can be used:

(1) objective fidelity criteria

(2) subjective fidelity criteria

When information loss can be expressed as a mathematical function of the input and output of a compression process, it is said to be based on an *objective fidelity criterion*. An

example is the root-mean-square (*rms*) error between two images. Let ***f(x,y)*** be an input image and $\bar{f}(x,y)$, be an approximation of ***f(x,y)*** obtained from compressing-decompressing the input. The error is:

$$e(x,y) = \bar{f}(x,y) - f(x,y)$$

$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} e(x,y)$ total error between the images

The *root-mean-square*, e_{rms} , between $f(x,y)$ and $\bar{f}(x,y)$ is

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

If $\bar{f}(x,y)$ is considered to be the sum of the original image $f(x,y)$ and an error or "noise" signal $e(x,y)$, the *mean-square signal-to-noise* of the output image can be defined by:

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

Decompressed images are ultimately viewed by humans. So, measuring image quality by the subjective evaluations of people is often more appropriate. This can be done by presenting a decompressed image to a cross section of viewers and averaging their evaluations.

Image Compression Models

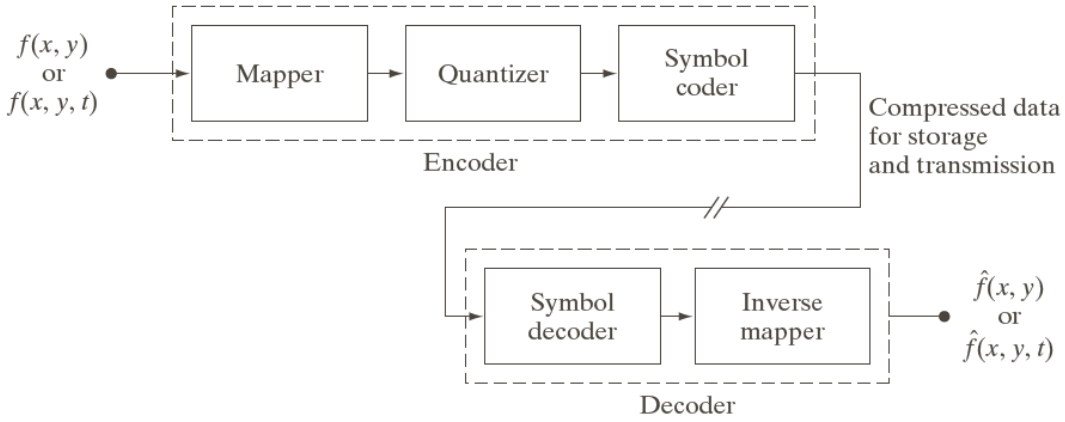


FIGURE 8.5 Functional block diagram of a general image compression system.

An image compression system is composed by two distinct functional components: an *encoder* and a *decoder*. Both operations can be performed in software, as is the case in

Web browsers and many commercial image editing programs, or in a combination of hardware and firmware, as in commercial DVD players. A *codec* is a device or program that is capable of both encoding and decoding.

Input image ***f(x,…)*** is fed into the encoder, which creates a compressed representation of the input. When the compressed representation is presented to its complementary decoder, a reconstructed output image $\bar{f}(x,...)$ is generated. In still-image applications, the encoded input and decoded

output are $f(x,y)$ and $\bar{f}(x,y)$, respectively; in video applications, they are $f(x,y,t)$ and $\bar{f}(x,y,t)$. In general, $\bar{f}(x,...)$ may or may not be an exact replica of $f(x,\ldots)$. If the two images are identical, the compression system is called *error free*, *lossless*, or *information preserving*. In the two images are different, the compression system is referred to as *lossy*.

The encoding and compression process

The encoder is designed to remove the three redundancies described before. 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. Thise operation generally is reversible and may or may not reduce directly 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. In video applications, the *bit rate* of the encoded output is often measured (in bits/second) and used to adjust the operation of the quantizer so that 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.

The *symbol coder* generates a fixed- 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.

The decoding or decompression process

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.

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 that is used. An *image container* is similar to a

file format but handles multiple types of image data. Image *compression standards* define procedures for compressing and decompressing images (for reducing the amount of data needed to represent an image).

ISO – *International Standards Organization*

IEC – *International Electrotechnical Commission*

ITU-T – *International Telecommunications Union*

CCITT – *Consultative Committee of the International Telephone and Telegraph*

Course 11

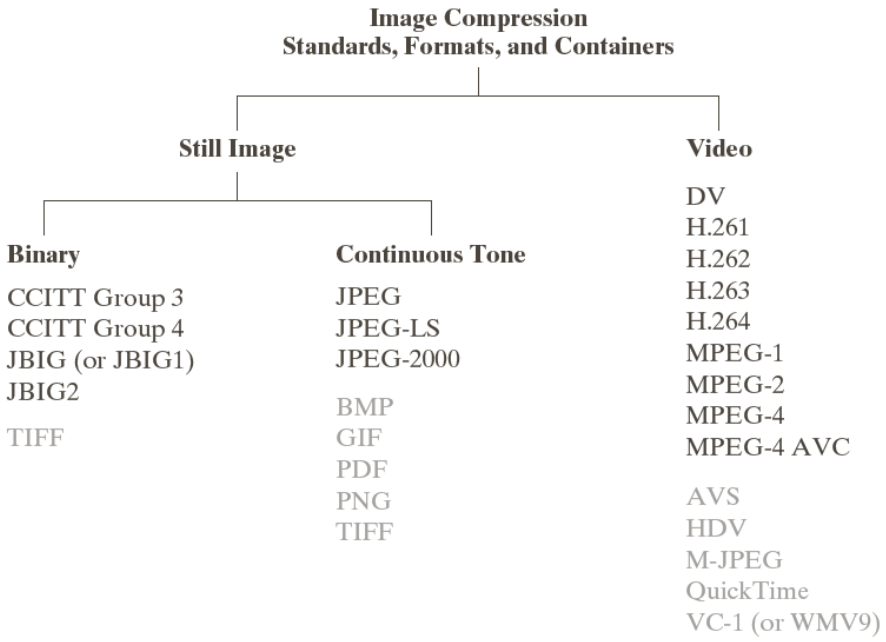


FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

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 [8.2.5] 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.

(*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.

Course 11

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.

Some Basic Compression Methods

Huffman Coding (1952)

When coding the symbols of an information source individually, *Huffman coding* yields the smallest possible number of code symbols per source symbol. In the 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 length, …).

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Fig. 8.7 illustrates this process for binary coding.

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				

FIGURE 8.7
Huffman source reductions.

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.

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				

FIGURE 8.8 Huffman code assignment procedure.

The average length of this code:

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

$$\mathbf{H(p) = 2.14353}$$

When a large number of symbols are 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 probability can be estimated in advance, "near optimal" coding can be achieved with pre-computed Huffman codes. JPEG and MPEG standards specify default Huffman coding tables that have been pre-computed based on experimental data.

Arithmetic Coding

Arithmetic coding generates non-block codes, there is no one-to-one correspondence between source symbols and code words. 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 in ***[0,1]***. 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.

Figure 8.12 and Table 8.6 illustrate the basic arithmetic coding process, for the sequence $a_1\ a_2\ a_3\ a_3\ a_4$ from a four-symbol source. At the end of the coding process we get the interval ***[0.06752, 0.0688]***.

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)

TABLE 8.6
Arithmetic coding example.

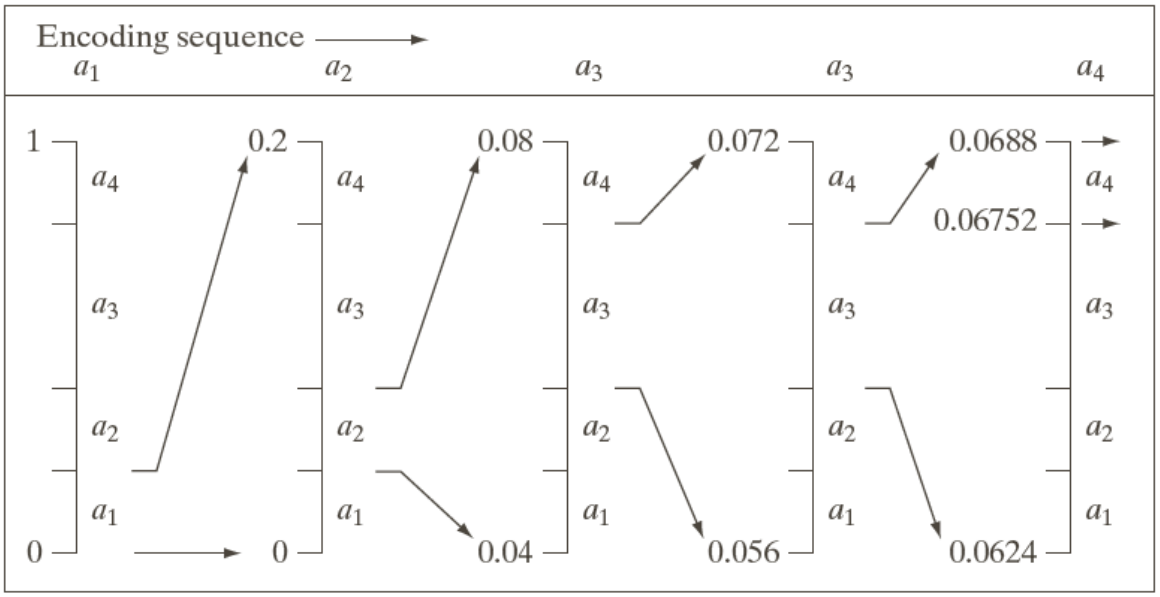


FIGURE 8.12
Arithmetic coding procedure.

Any number within the interval found can be used to represent the message.

Practical implementations of arithmetic coding take into account a scaling strategy and a rounding strategy. The scaling strategy renormalizes each subinterval to the ***[0,1)*** range before subdividing it in accordance with the symbol probabilities. The rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent the coding subintervals from being represented accurately.

Adaptive context dependent probability estimates

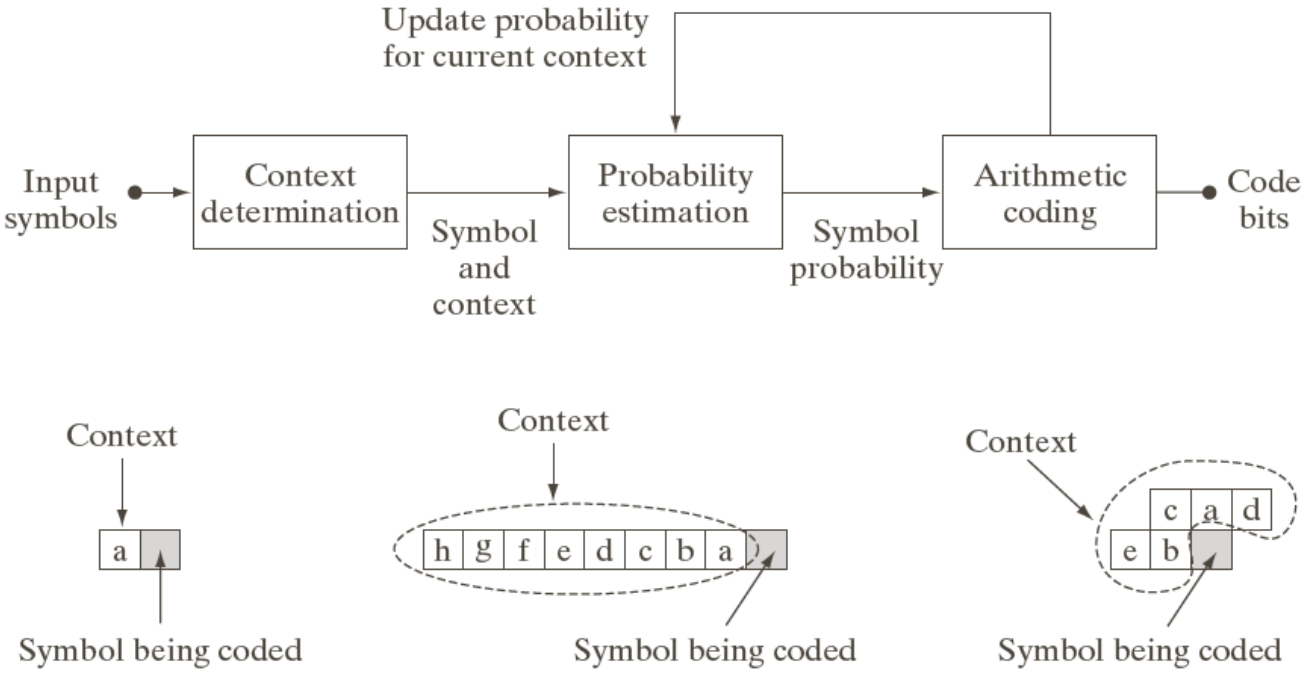
With accurate input symbol probability models, that is, models that provide the true probabilities of the symbols being coded, arithmetic coders are near optimal in the sense of minimizing the average number of code symbols required to represent the symbols being coded.

Adaptive probability models update symbol probabilities as symbols are being 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 neighbourhood of pixels – called the *context* – around the symbols being coded. A *causal context*, one limited to symbols that have already been coded, is used. Both the Q-coder and MQ-coder, two arithmetic coding techniques that have been incorporated into the JBIG, JPEG-2000, use probability models that are both adaptive and context dependent. The Q-coder dynamically updates symbol

probabilities during the interval renormalizations that are part of the arithmetic coding process.

Arithmetic coding often is used when binary symbols are to be coded.



a
b c d

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

LZW Coding

Lempel-Ziv-Welch (LZW) coding assigns fixed-length code words to variable length sequences of source symbols. A key feature of LZW coding is that it requires no a priori knowledge of the probability of occurrence of the symbols to be encoded. LZW compression has been integrated into a variety of mainstream imaging formats, including GIF, TIFF, and PDF. The PNG format was created to get around LZW licensing requirements.

LZW coding is conceptually very simple: 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,…, 255. As the encoder sequentially examines image pixels, intensity sequences that are not in the dictionary are placed in algorithmically determined (e.g. the next unused) locations. If the first two pixels of the image are white, for instance, sequence “255-255” might be assign to location 256. The

next time that two consecutive white pixels are encountered, code word 256 is used to represent them. If a 9-bit, 512-word dictionary is employed in the coding process, the original (8+8) bits that were used to represent the two pixels are replaced by a single 9-bit code word.

Clearly, the size of the dictionary is an important system parameter. If it is too small, the detection of matching intensity-level sequences will be less likely; if it is too large,

the size of the code words will adversely affect compression performance.

Consider the image

39	**39**	**126**	**126**
39	**39**	**126**	**126**
39	**39**	**126**	**126**
39	**39**	**126**	**126**

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

EXAMPLE 8.7: LZW coding.

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.

Note that a unique feature of the LZW coding is that the coding dictionary is created while the data are being encoded. An LZW decoder builds an identical decompression dictionary as it decodes simultaneously the encoded data stream. Most practical applications require a strategy for handling dictionary overflow. A simple solution is to flush or reinitialize the dictionary when it becomes full and continue coding with a new initialized dictionary. A more complex option is to monitor compression performance and flush the

dictionary when it becomes poor or unacceptable. Alternatively, the least used dictionary entries can be tracked and replaced when necessary.

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*, where each run-length pair specifies the start of a new intensity and the number of pixels that have that intensity. *Run-length*

encoding (RLE) was developed in the ‘50s and became the standard compression approach in facsimile (FAX) coding. Compression is achieved by eliminating a simple form of spatial redundancy – groups of identical intensities.

The BMP file format uses a form of run-length encoding in which image data is represented in two different modes: encoded and absolute – and 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 colour index contained in the second byte. The 8-bit colour index selects the run's intensity (colour or grey 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 8.8.

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

TABLE 8.8 BMP absolute coding mode options. In this mode, the first byte of the BMP pair is 0.

If the second byte 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 colour index of one pixel. The total number of bytes must be aligned on a 16-bit word boundary.

Run-length encoding is particularly effective when compressing binary images. There are only two possible

intensities (black and white), adjacent pixels are more likely to be identical. Each image row can be represented by a sequence of length only – rather than length-intensity pairs. The basic idea is to code each contiguous group (i.e. run) of 0s and 1s encountered in a left to right scan of a row by its length and to established a convention for determining the value of the run. The most common conventions are: (1) to specify the value of the first run on each row or (2) to assume that each row begin with a white run, whose length may be 0.

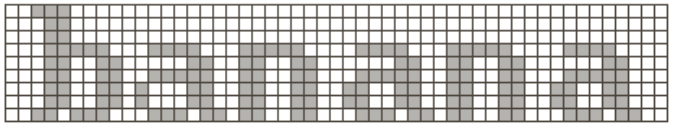
Additional compression can be achieved by variable-length coding the run length themselves. The black and white run length can be coded separately using variable length codes that are specifically made to their own statistics.

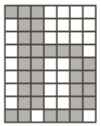
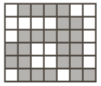
Symbol-Based Coding

In *symbol-* or *token-based* coding, an image is represented as a collection of frequently occurring sub-images, called *symbols*. Each such symbol is stored in a *symbol dictionary* and the image is coded as a set of triplets $(x_i\ y_i, t_i)$ where each

(x_i, y_i) pair specifies the location of a symbol in the image and the token t_i is the address of the symbol or sub-image in the dictionary. 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.

$$\{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots\}$$



Token	Symbol
0	
1	
2	

Triplet
(0, 2, 0) (3, 10, 1) (3, 18, 2) (3, 26, 1) (3, 34, 2) (3, 42, 1)

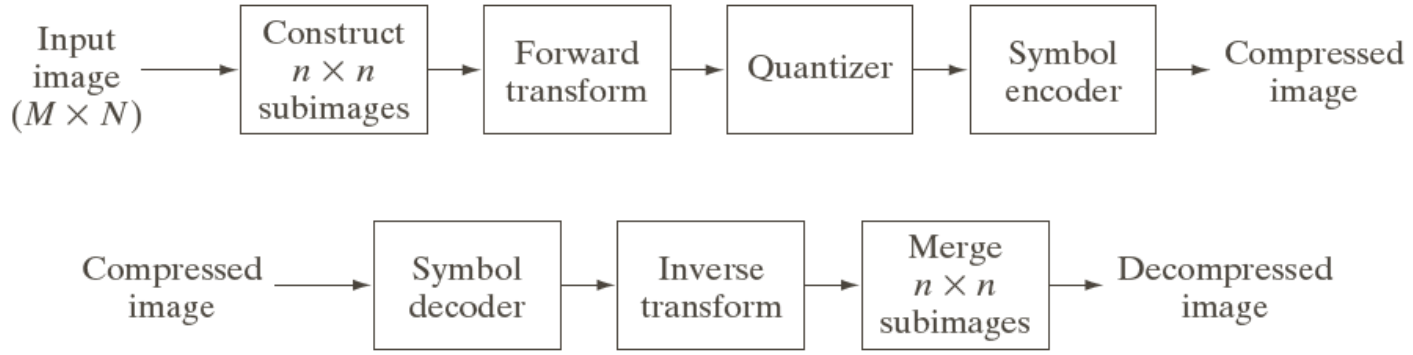
a b c

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

Block Transform Coding

Consider 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

coefficients have small magnitudes and can be quantized (or omitted entirely) with little image distortion.



a
b

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

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 *non-adaptive transform coding*.

Transform selection

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 steps).

Consider a subimage $g(x,y)$ of size $n \times n$ whose forward, discrete transform, $T(u,v)$, can be expressed in terms of the general relation:

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

Given $T(u,v)$, $g(x,y)$ similarly can be obtained using the generalized inverse discrete transform:

$$g(x,y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v) s(x,y,u,v) \quad , \quad x,y = 0,1,...,n-1 \quad (2)$$

r(x,y,u,v) and ***s(x,y,u,v)*** are called the *forward* and *inverse transformation kernels*, respectively (also called *basis functions* or *basis images*). The ***T(u,v)*** for ***u,v=0,1,…,n-1*** are called *transform coefficients*.

The kernel in (1) is *separable* if:

$$r(x,y,u,v) = r_1(x,u)r_2(y,v)$$

The kernel is *symmetric* if $r_1 = r_2$

$$r(x,y,u,v) = r_1(x,u)r_1(y,v)$$

For the discrete Fourier transform:

$$r(x,y,u,v) = e^{-i2\pi\frac{(ux+vy)}{n}}$$

$$s(x,y,u,v) = \frac{1}{n^2} e^{i2\pi\frac{(ux+vy)}{n}}$$

Walsh-Hadamard transform (WHT) – $n = 2^m$

$$r(x,y,u,v) = s(x,y,u,v) = \frac{1}{n}(-1)^{\sum_{i=0}^{m-1}\left\lfloor b_i(x)p_i(u)+b_i(y)p_i(v)\right\rfloor}$$

The summation in the exponent of ***(-1)*** in the above expression is performed in modulo 2 arithmetic and $b_k(z)$ is

the ***k***th (from right to left) in the binary representation of ***z***. The $p_i(u)$ are computed using:

$$p_0(u) = b_{m-1}(u)$$
$$p_1(u) = b_{m-1}(u) + b_{m-2}(u)$$
$$p_2(u) = b_{m-2}(u) + b_{m-3}(u)$$
$$\vdots$$
$$p_{m-1}(u) = b_1(u) + b_0(u)$$

where the sums are performed in modulo 2 arithmetic. The Walsh-Hadamard kernels consist of alternating ***+1*** and ***-1***

arranged in a checkboard pattern. Figure 8.22 shows the kernel for ***n=4***. Each block consists of ***4×4=16*** elements.

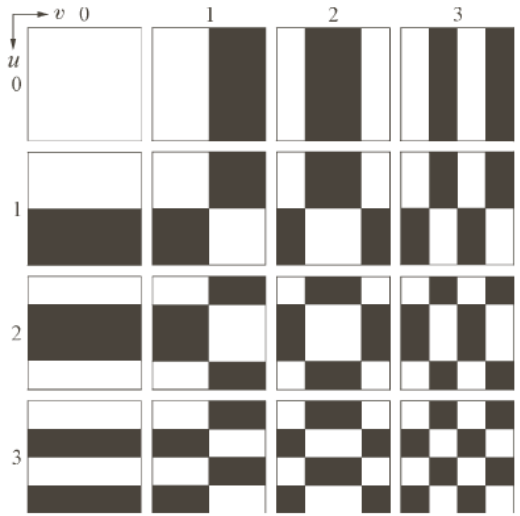


FIGURE 8.22 Walsh-Hadamard basis functions for $n = 4$. The origin of each block is at its top left.

Discrete cosine transform (DCT)

$$r(x,y,u,v) = s(x,y,u,v) =$$

$$= \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2n}\right]\cos\left[\frac{(2y+1)v\pi}{2n}\right]$$

$$\alpha(u) = \begin{cases} \sqrt{\dfrac{1}{n}} & \text{for } u = 0 \\ \sqrt{\dfrac{2}{n}} & \text{for } u = 1,2,...,n-1 \end{cases}$$

Course 11

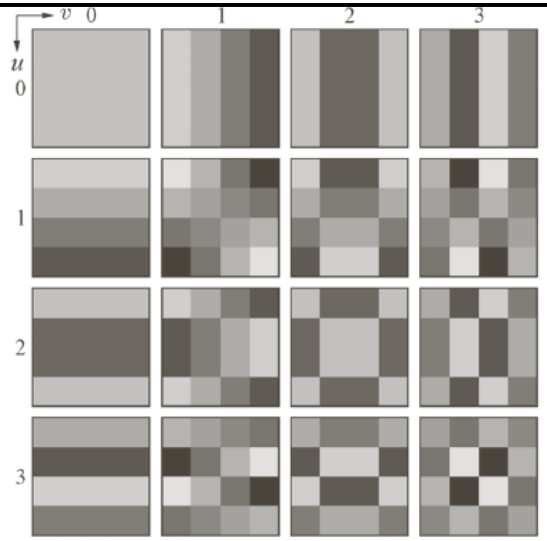


FIGURE 8.23
Discrete-cosine basis functions for $n = 4$. The origin of each block is at its top left.

FIGURE 8.9(a)

a b c
d e f

FIGURE 8.24 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).

Figure 8.24(a) show three approximations of the 512×512 monochrome image in Figure 8.9(a). These pictures were obtained by dividing the original image into subimages of size 8×8, representing each subimage using the transforms DFT, WHT and DCT, truncating 50% of the resulting coefficients, and taking the inverse transform of the truncated coefficient arrays. In each case, the 32 retained coefficients were selected on the basis of maximum magnitude.

This example shows that the information packing ability of the DCT is superior to that of DFT and WHT - this observation usually holds for most images. The Karhunen-Loève transform is the optimal transform in an information packing sense – it is a data dependent transform and its computational task is significant.

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

Subimage size selection

In most applications, images are subdivided so that the correlation (redundancy) between adjacent subimages is reduced to some acceptable level and so that n is an integer power of 2. The most popular subimage sizes are 8×8 and 16×16.

Representation and Description

After segmentation, the resulting sets of pixels are represented in a form suitable for further processing. There are two ways of representing a region:

(1) using the boundary (external characteristics)

(2) using the internal characteristics (the pixels inside the region)

The next task is to describe the region based on the chosen representation.

External representation is chosen when the primary focus is on shape characteristics, internal representation is used when the focus is on regional properties, such as color and texture.

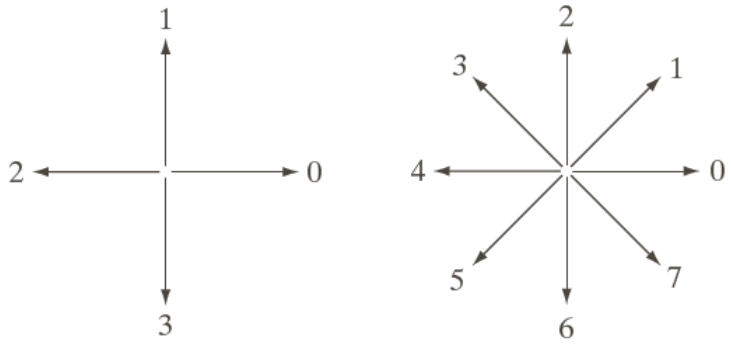
Representation

- boundary following
- chain codes
- polygonal approximations
- signatures
- skeletons

Chain Codes

Chain codes are used to represent a boundary by a connected sequence of straight line segments of specified length and direction.

The direction of each segment is coded by using a numbering scheme such as the ones shown below. A boundary code formed using a sequence of such directional numbers is referred to as a *Freeman chain code*.



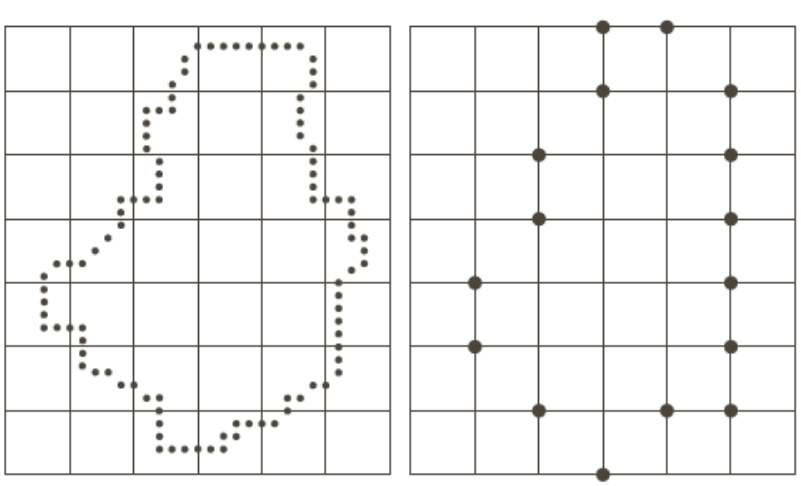
This method generally is unacceptable to be applied for each pair of consecutive pixels:

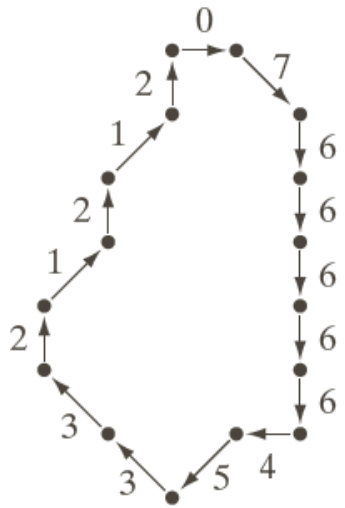
(a) The resulting chain of codes usually is quite long;

(b) Sensitive to noise: any small disturbances along the boundary due to noise or imperfect segmentation cause changes in the code that may not necessarily be related to the shape of the boundary.

A frequently used method to solve the problem is to resample the boundary by selecting larger grid spacing. A boundary point is assigned to each node of the large grid, depending on

the proximity of the original boundary to that node. The accuracy of the resulting code representation depends on the spacing of the sampling grid.





a b c

FIGURE 11.4 (a) Digital boundary with resampling grid superimposed. (b) Result of resampling. (c) 8-directional chain-coded boundary.

The chain code of a boundary depends on the starting point. The problem is solved by normalization.

Normalization for starting point:

Treat the code as a circular sequence and redefine the starting point so that the resulting sequence of numbers forms an integer of minimum magnitude.

Normalization for rotation:

Use the first difference of the chain code instead of the code itself. The difference is simply by counting counterclockwise the number of directions that separate two adjacent elements of the code.

Example: The first difference of the 4-direction chain code

10103322 is ***33133030***.

Merging technique

The idea is to merge points along a boundary until the least square error line fit of the points merged so far exceeds a preset threshold. When this condition occurs, the parameters of the line are stored, the error is set to 0, and the procedure is repeated, merging new points along the boundary until the error again exceds the threshold. One of the main problem

with this technique is that vertices do not corespond with corners in the boundary.

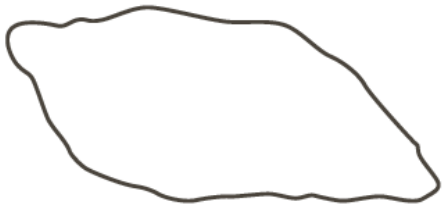
Splitting technique

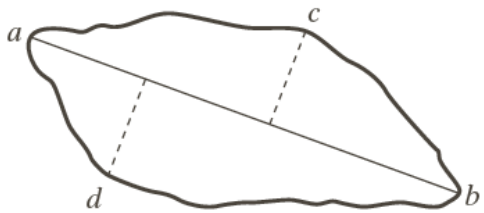
One approach to boundary segment *splitting* is to subdivide a segment successively into two parts until a specified criterion is satisfied. For instance, a requirement might be that the maximum perpendicular distance from a boundary segment to the line joining its two end points not exceed a preset

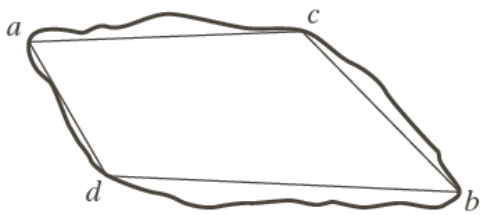
threshold. If it does, the point having the greatest distance from the line becomes a vertex, thus subdividing the initial segment into two subsegments.

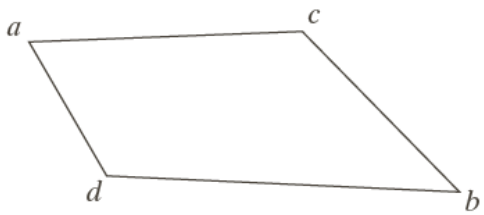
This approach has the advantage of seeking prominent inflection points. For a closed boundary, the best starting points usually are the two farthest points in the boundary.

Course 11









Boundary Descriptors

The *length* of a boundary is one of its simplest descriptors. The number of pixels along a boundary gives a rough approximation of its length.

The *diameter* of a boundary $\boldsymbol{B}$ is defined as:

$$\mathbf{Diam}(\boldsymbol{B}) = \max\{ \boldsymbol{D}(\boldsymbol{p_i}, \boldsymbol{p_j}); \ \boldsymbol{p_i}, \boldsymbol{p_j} \in \boldsymbol{B}\}$$

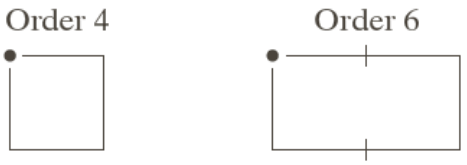
where $\boldsymbol{D}$ is a distance measure. The value of the diameter and the orientation of a line segment connecting the two extreme points that comprise the diameter (this line is called the *major*

axis of the boundary) are useful descriptors of a boundary. The *minor axis* of a boundary is defined as the line perpendicular to the major axis, and of such length that a box passing through the outer four points of intersection of the boundary with the two axes completely encloses the boundary. The box just described is called the *basic rectangle*, and the ratio of the major axis to the minor axis is called the *eccentricity* of the boundary. This also is a useful descriptor.*Curvature* is defined as the rate of change of slope.

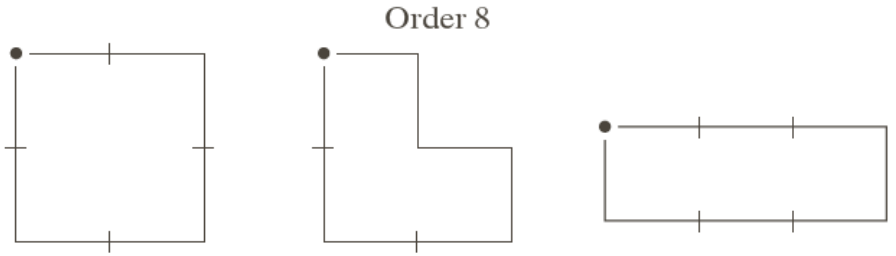
Shape numbers

Assume that the boundary is described by the first difference of a the associated chain-code. The *shape number* of such a boundary, based on the 4-directional code, is defined as the first difference of smallest magnitude. The *order n* of a shape number is defined as the number of digits in its representation. Moreover, *n* is even for a closed boundary, and its value limits the number of possible different shapes.

Course 11



Chain code:	0 3 2 1	0 0 3 2 2 1
Difference:	3 3 3 3	3 0 3 3 0 3
Shape no.:	3 3 3 3	0 3 3 0 3 3

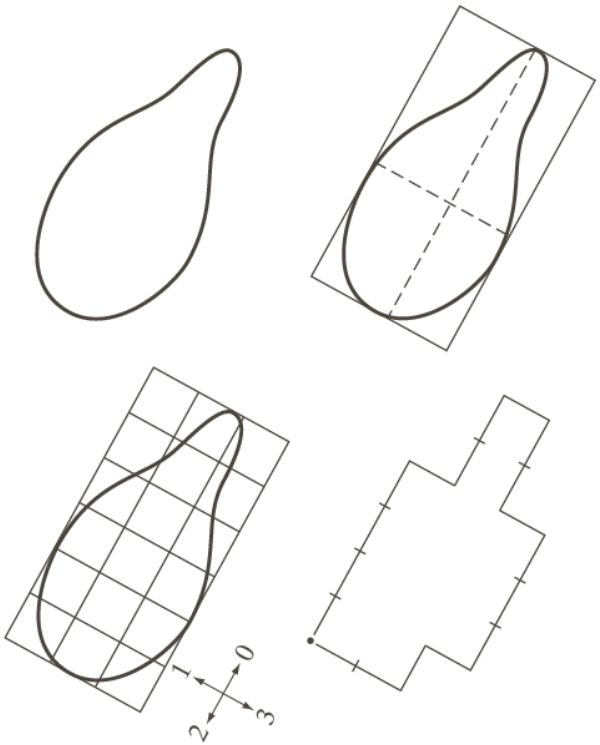


Chain code:	0 0 3 3 2 2 1 1	0 3 0 3 2 2 1 1	0 0 0 3 2 2 2 1
Difference:	3 0 3 0 3 0 3 0	3 3 1 3 3 0 3 0	3 0 0 3 3 0 0 3
Shape no.:	0 3 0 3 0 3 0 3	0 3 0 3 3 1 3 3	0 0 3 3 0 0 3 3

Although the first difference of a chain code is independent of rotation, in general the coded boundary depends on the orientation of the grid. One way to normalize the grid orientation is by aligning the chain-code grid with the sides of the basic rectangle.

In practice, for a desired shape order, we find the rectangle of order *n* whose eccentricity best approximates that of the basic rectangle of the region and use this new rectangle to establish the grid size.

Course 11



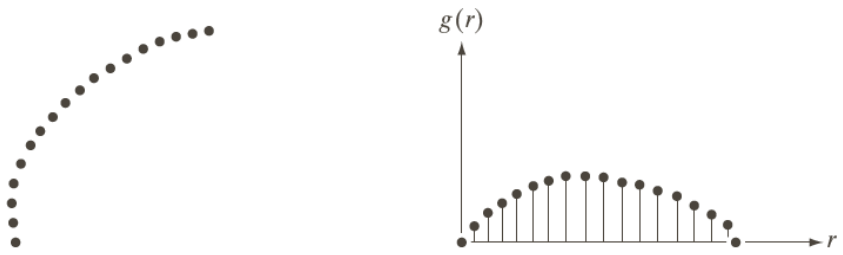
Chain code: 0 0 0 0 3 0 0 3 2 2 3 2 2 2 1 2 1 1

Difference: 3 0 0 0 3 1 0 3 3 0 1 3 0 0 3 1 3 0

Shape no.: 0 0 0 3 1 0 3 3 0 1 3 0 0 3 1 3 0 3

Statistical moments

The shape of boundary segments (and of signature waveforms) can be described quantitatively by using statistical moments, such as the mean, variance, and higher order moments.



We represent the segment of a boundary by a 1-D function ***g(r).*** This function is obtained by connecting the two end points of the segment and rotating the line segment until it is horizontal. The coordinates of the points are rotated by the same angle.

Let us treat the amplitude of ***g*** as a discrete random variable ***v*** and form an amplitude histogram $p(v_i)$, $i = 0, 1, 2, \ldots, A - 1$, where ***A*** is the number of discrete amplitude increments in which we divide the amplitude scale.

The ***n***th moment of ***v*** about its mean is:

$$\mu_n(v) = \sum_{i=0}^{A-1} (v_i - m)^n \, p(v_i) \;, \quad m = \sum_{i=0}^{A-1} v_i \, p(v_i).$$

The quantity ***m*** is recognized as the mean or average value of ***v*** and μ_2 as its variance. Generally, only the first few moments are required to differentiate between signatures of clearly distinct shapes.

Regional descriptors

The *area* of a region is defined as the number of pixels in the region. The *perimeter* of a region is the length of its boundary. These two descriptors apply primarily to situations in which the size of the regions of interest is invariant. A more frequent use of these two descriptors is in measuring *compactness* of a region:

$$\textbf{compactness} = \frac{\textbf{(perimeter)}^2}{\textbf{area}} = \frac{\boldsymbol{P}^2}{\boldsymbol{A}}$$

Another descriptor of compactness is the *circularity ratio:*

$$\textbf{circularity ratio} = \frac{\textbf{area of the region}}{\textbf{area of the circle having the same perimeter}}$$

The area of a circle with perimeter length P is $P^2/4\pi$.

$$R_c = \frac{4\pi A}{P^2}.$$

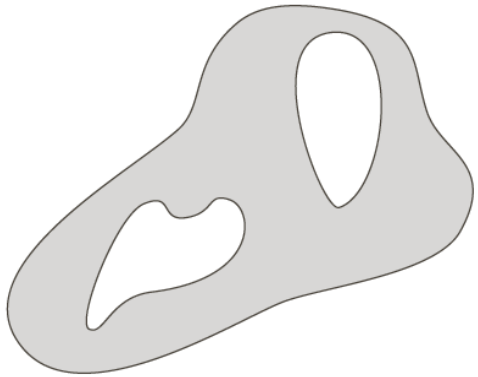
The value of this measure is 1 for a circular region and $\pi/4$ for a square. Compactness is a dimensionless measure and thus is insensitive to uniform scale changes; it is insensitive also to

orientation, ignoring computational errors that may be introduced in resizing and rotating a digital region.

Other simple measures used as region descriptors include the mean and median of the intensity levels, the minimum and maximum intensity values, and the number of pixels with values above and below the mean.

Topological Descriptors

Topology is the study of properties of a figure that are unaffected by any deformation, as long as there is no tearing or joining of the figure (sometimes these are called *rubber-sheet* distortions).



For example, the above figure shows a region with two holes. Thus if a topological descriptor is defined by the *number of holes* (***H***) in the region, this property obviously will not be affected by a stretching or rotation transformation. In general, however, the number of holes will change if the region is torn or folded. Note that, as stretching affects distance, topological properties do not depend on the notion of distance or any properties implicitly based on the concept of a distance measure.

Another topological property useful for region description is the number of connected components (***C***).

The number of holes ***H*** and connected components ***C*** in a figure can be used to define the *Euler number* ***E****:*

$$E = C - H.$$

Regions represented by straight-line segments (referred to as *polygonal networks)* have a particularly simple interpretation in terms of the Euler number.

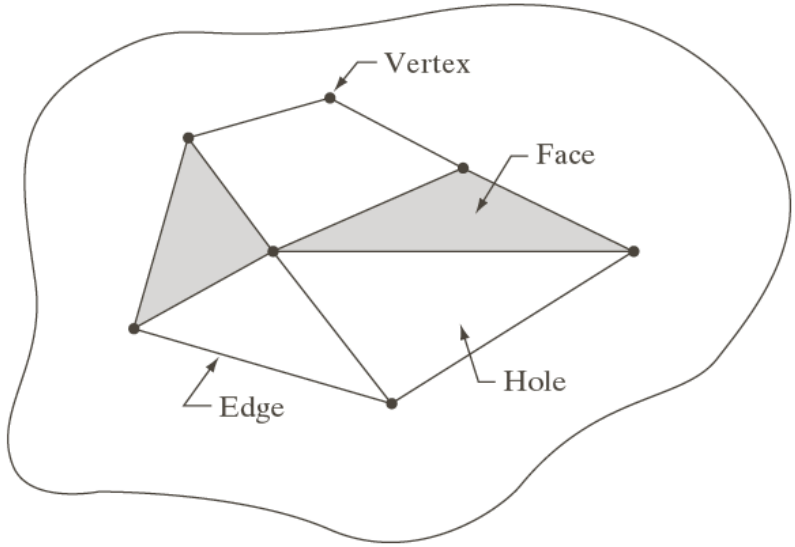
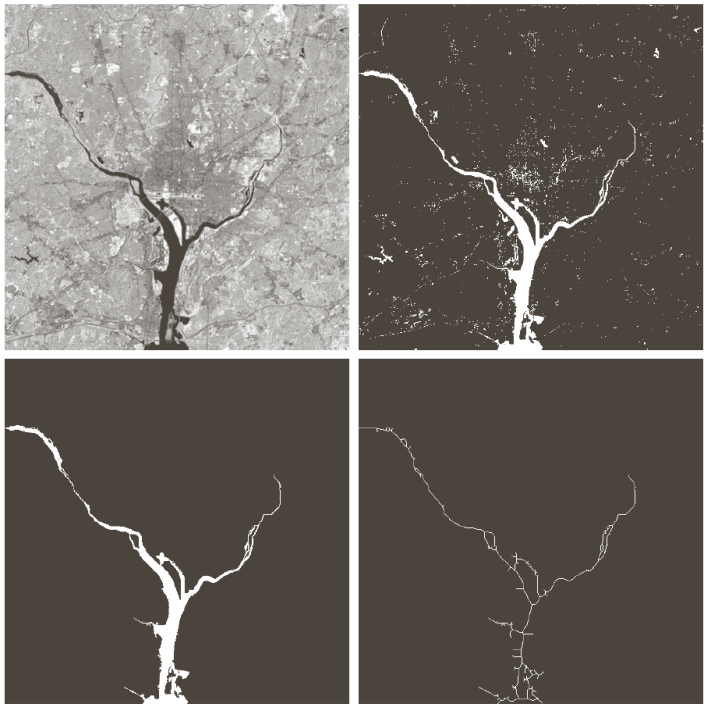


FIGURE 11.26 A region containing a polygonal network.

Figure 11.26 shows a polygonal network. Classifying interior regions of such a network into faces and holes is often important. Denoting the number of vertices by V, the number

of edges by ***Q***, and the number of faces by ***F*** gives the following relationship, called the *Euler formula:*

$$V-Q+F = C-H = E.$$



a b
c d

FIGURE 11.27 (a) Infrared image of the Washington, D.C. area. (b) Thresholded image. (c) The largest connected component of (b). Skeleton of (c).

Suppose we want to segment the river from image in Fig. 11.27 (a). The image in Fig. 11.27 (b) has 1591 connected components (obtained using 8-connectivity) and its Euler number is 1552, from which we deduce that the number of holes is 39. Figure 11.27(c) shows the connected component with the largest number of elements (8479). This is the desired result, which we already know cannot be segmented by itself from the image using a threshold.

Texture

An important approach to region description is to quantify its *texture* content. Although no formal definition of texture exists, this descriptor provides measures of properties such as smoothness, coarseness and regularity. The three principal approaches for describing the texture of a region are statistical, structural, and spectral. Statistical approaches yield characterizations of textures as smooth, coarse, grainy, … Structural techniques deal with the arrangement of image

primitives, such as the description of texture based on regularly spaced parallel lines. Spectral techniques are based on properties of the Fourier spectrum and are used primarily to detect global periodicity in an image by identifying high-energy, narrow peaks in the spectrum.

Statistical approaches

One of the simplest for describing texture is to use statistical moments of the intensity histogram of an image or region. Let ***z*** be a random variable denoting intensity and let $\boldsymbol{p(z_i)}$**,**

$i = 0, 1, 2, \ldots, L-1$ be the corresponding histogram, where L is the number of distinct intensity levels. The *n-th* moment of z about the mean is where m is the mean value of z is:

$$\mu_n(z) = \sum_{i=0}^{L-1} (z_i - m)^n \, p(z_i) \quad , \quad m = \sum_{i=0}^{L-1} z_i \, p(z_i) \; .$$

Note that $\mu_0 = 1$ and $\mu_1 = 0$. The second moment, the *variance* ($\sigma^2(z) = \mu_2(z)$) is of particular importance in texture description. It is a measure of intensity contrast that can be

used to establish descriptors of relative smoothness. For example, the measure:

$$R(z) = 1 - \frac{1}{1 + \sigma^2(z)}$$

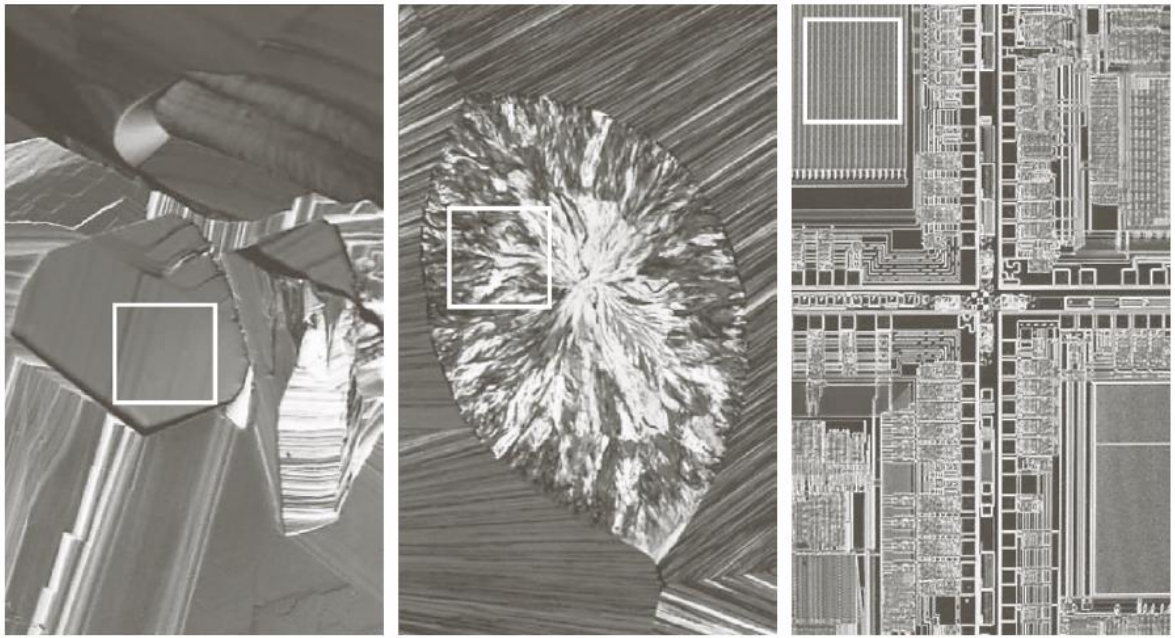
is 0 for areas of constant intensity and approaches 1 for large values of $\sigma^2(z)$.

The third moment $\mu_3(z)$ is a measure of the skewness of the histogram while the forth moment is a measure of its relative

flatness. Some other useful textures measure are “uniformity” and the average entropy:

$$U(z) = \sum_{i=0}^{L-1} p^2(z_i)$$

$$e(z) = -\sum_{i=0}^{L-1} p(z_i) \log_2 p(z_i)$$



a b c

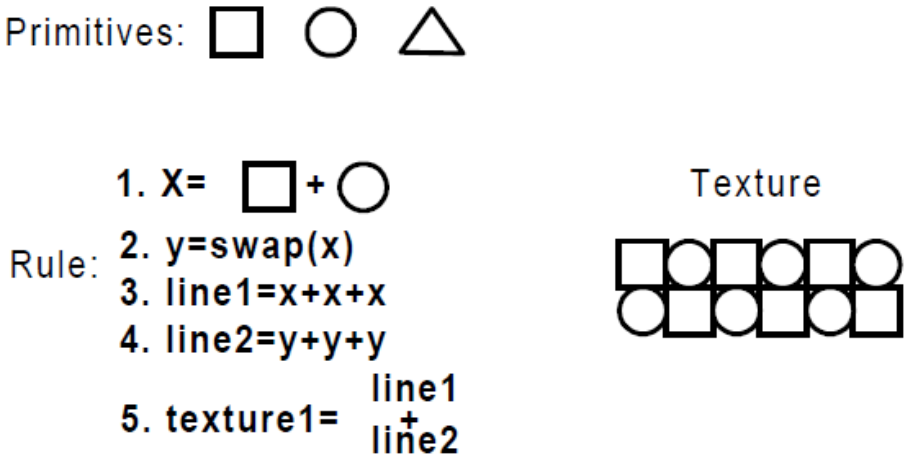
FIGURE 11.28 The white squares mark, from left to right, smooth, coarse, and regular textures. These are optical microscope images of a superconductor, human cholesterol, and a microprocessor. (Courtesy of Dr. Michael W. Davidson, Florida State University.)

Texture	Mean	Standard deviation	R (normalized)	Third moment	Uniformity	Entropy
Smooth	82.64	11.79	0.002	−0.105	0.026	5.434
Coarse	143.56	74.63	0.079	−0.151	0.005	7.783
Regular	99.72	33.73	0.017	0.750	0.013	6.674

TABLE 11.2 Texture measures for the subimages shown in Fig. 11.28.

Structural aproach

Structural techniques deal with the arrangement of image primitives. They use a set of predefined texture primitives and a set of construction rules to define how a texture region is constructed with the primitives and the rules.



Spectral approaches

Spectral techniques use the Fourier transform of the image and its properties in order to detect global periodicity in an image, by identifying high-energy, narrow peaks in the spectrum.

The Fourier spectrum is ideally suited for describing the directionality of periodic or almost periodic 2-D patterns in an image.

Three features of the spectrum are suited for texture description:

(1) prominent peaks give the principal direction of the patterns;

(2) the location of the peaks gives the fundamental spatial period of the patterns;

(3) eliminating any periodic components via filtering leaves nonperiodic image elements, which can be described by statistical techniques.

We express the spectrum in polar coordinates to yield a function $S(r, \theta)$. For each direction θ, $S(r, \theta)$ may be considered a 1-D function $S_\theta(r)$. Similarly, for each frequency r, $S_r(\theta)$ is a 1-D function. Analyzing $S_\theta(r)$ for a fixed value of θ yields the behavior of the spectrum (such as the presence of peaks) along a radial direction from the origin, whereas analyzing $S_r(\theta)$ for a fixed value of r yields the behavior along a circle centered on the origin.

A more global description is obtained by using the following functions:

$$S(r)=\sum_{\theta=0}^{\pi} S_{\theta}(r) \quad , \quad S(\theta)=\sum_{r=1}^{R_0} S_{r}(\theta)$$

where R_0 is the radius of a circle centered at the origin.

$S(r)$ and $S(\theta)$, that constitute a spectral-energy description of texture for an entire image or region under consideration.

Furthermore, descriptors of these functions themselves can be computed in order to characterize their behavior

quantitatively. Descriptors typically used for this purpose are the location of the highest value, the mean and variance of both the amplitude and axial variations, and the distance between the mean and the highest value of the function.

Gray-level co-occurrence matrices

Q – an operator that specifies the position of two pixels relative to each other

I – a grayscale image with ***L*** possible intesnities

$G = \left(g_{ij} \right)_{i,j=1}^{K}$ graylevel co-occurrence matrix

g_{ij} = number of pairs ***(i,j)*** specified by ***Q*** from ***I***

n= total number of pairs in ***I*** that satisfy ***Q***

$$p_{ij} = g_{ij} / n$$

$\boldsymbol{p_{ij}}$ = an estimate of the probability that the pair of point ***(i,j)*** satisfying ***Q*** apear in ***I***

$$\sum_{i=1}^{K}\sum_{j=1}^{K} p_{ij} = 1$$

$$m_r = \sum_{i=1}^{K} i \sum_{j=1}^{K} p_{ij} \quad , \quad m_c = \sum_{j=1}^{K} j \sum_{i=1}^{K} p_{ij}$$

$$\sigma_r^2 = \sum_{i=1}^{K} (i - m_r)^2 \sum_{j=1}^{K} p_{ij} \quad , \quad \sigma_c^2 = \sum_{j=1}^{K} (j - m_c)^2 \sum_{j=1}^{K} p_{ij}$$

Set of descriptors for characterizing the contents of G:

1. Maximum probability – measures the strongest response of G (the range of values is [0,1]):

$$\max\{p_{ij}; i, j = 1,...,K\}$$

2. Correlation – a measure of how correlated a pixel is to its neighbor over the entire image; range of values [-1,1]. This measure is not defined if either standard deviation is zero:

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\frac{(i-m_r)(j-m_c)}{\sigma_r\sigma_c}\ ,\ \ \sigma_r \neq 0\ ,\ \sigma_c \neq 0$$

3. Contrast -a measure of intensity contrast between a pixel and its neighbor over the entire image; range of values is 0 (***G*** is constant) to ***(K-1)²***

$$\sum_{i=1}^{K}\sum_{j=1}^{K}(i-j)^2 p_{ij}$$

4. Uniformity (Energy) – range [0,1]. Uniformity is 1 for constant images

$$\sum_{i=1}^{K}\sum_{j=1}^{K}p_{ij}^2$$

5. Homogeneity – measures the spatial closeness of the distribution of the elements in G to the diagonal. The range of values is [0,1] (the value 1 is achieved for diagonal matrices)

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\frac{p_{ij}}{1+|i-j|}$$

6. Entropy – measures the randomness of the elements of G. The maximum value is $2\log_2 K$

$$-\sum_{i=1}^{K}\sum_{j=1}^{K} p_{ij} \log_2 p_{ij}$$

$$P(i) = \sum_{j=1}^{K} p_{ij} \quad , \quad P(j) = \sum_{i=1}^{K} p_{ij}$$

$$m_r = \sum_{i=1}^{K} iP(i) \quad , \; m_c = \sum_{j=1}^{K} jP(j)$$

$$\sigma_r^2 = \sum_{i=1}^{K} (i - m_r)^2 P(i) \quad , \quad \sigma_c^2 = \sum_{j=1}^{K} (j - m_c)^2 P(j)$$

Local Binary Patterns[1,2]

- a texture descriptor
- for each pixel in the image a ‘binary pattern’ is associated by comparing the pixel with its 8 neighbors

[1] T. Ojala, M. Pietikainen, T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns." *IEEE Transactions on pattern analysis and machine intelligence* 24.7 (2002): 971-987.

[2] M.S.Nixon, A.S. Aguado – Feature Extraction & Image Processing for Computer Vision, 3rd ed, Academic Press, 2012

$$\begin{matrix} 118 & 120 & 95 \\ 90 & 106 & 110 \\ 45 & 3 & 2 \end{matrix} \rightarrow \begin{matrix} 1 & 1 & 0 \\ 0 & - & 1 \\ 0 & 0 & 0 \end{matrix}$$

(binary pattern 10100001 of 106)

$$\begin{matrix} 1 & 128 & 64 \\ 2 & - & 32 \\ 4 & 8 & 16 \end{matrix} \rightarrow \begin{matrix} 1 & 128 & 0 \\ 0 & 161 & 32 \\ 0 & 0 & 0 \end{matrix}$$

P – the center pixel, $P_x \in N_8(P)$

$$s(x) = \begin{cases} 1 & \text{if } P_x > P \\ 0 & \text{otherwise} \end{cases}$$

$$LBP = \sum_{x \in N_8} s(x) 2^{x-1}$$

The Local Binary Patterns are

1. invariant to global illumination changes
2. invariant to local contrast changes
3. translation invariant

LBP is not scale or rotation invariant

For obtaining scale invariance it is necessary to take into consideration points at a greater distance. We consider P

points placed on a circle with centre (x_0, y_0) and radius R. The coordinates of these points are:

$$z(i) = \begin{bmatrix} x_0 + R\cos\left(\frac{2\pi}{P} i\right) \\ y_0 + R\sin\left(\frac{2\pi}{P} i\right) \end{bmatrix}$$

$$LBP\_S(P,R) = \sum_{i=1}^{P} s(z(i))2^{i-1}$$

In order to find the intensity value for *z(i)* usually interpolation is employed rather than nearest neighbour technique.

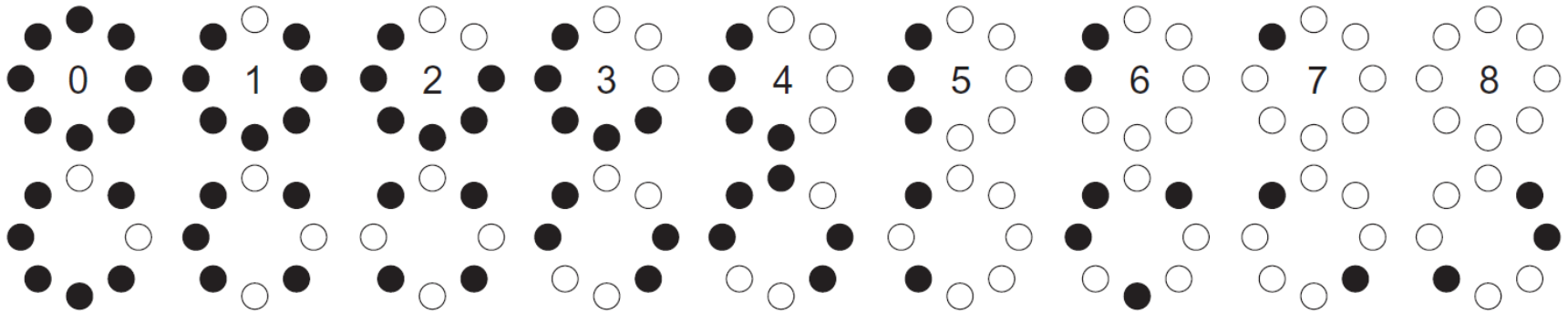
For rotation invariance, the derived binary pattern is shifted to achieve the minimum integer:

$$LBP\_R(P,R) = \min\left\{\text{Circular\_rotation}\left(\sum_{i=1}^{P} s(z(i))2^{i-1}\right)\right\}$$

Uniform binary patterns: the code has at most two transition "1-0" or "0-1". The other patterns are called nonuniform. For a binary pattern computed with the N_8 neighbours, there are 58 uniform pattern (out of 256).

The operator U counts the number of transitions "1-0", "0-1"

$$U(LBP\_S(P,R)) = | s(z(0)) - s(z(P)) | + \sum_{i=1}^{P-1} | s(z(i)) - s(z(i+1)) |$$



Rotation invariant uniform binary patterns

Patterns 0-8 correspond to basic features:

0/8 – bright/dark spot

1-7 – lines of varying degrees of curvature

1 – end of a line

2 – a sharp point

4 – an edge

Features with LBP:

- compute LBP image
- divide the LBP matrix in subblocks
- concatenate the histograms of the LBP subimages

LPB applications

Facial Analysis: recognition, detection, expression recognition, gender

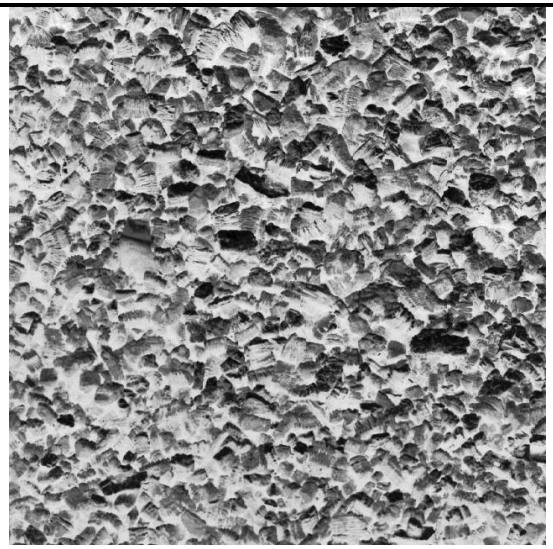
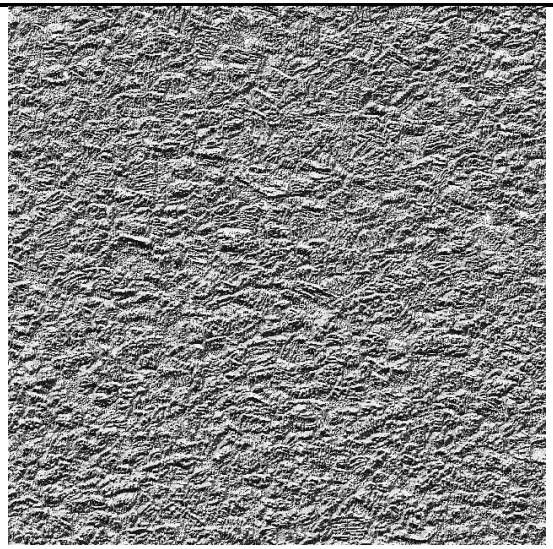
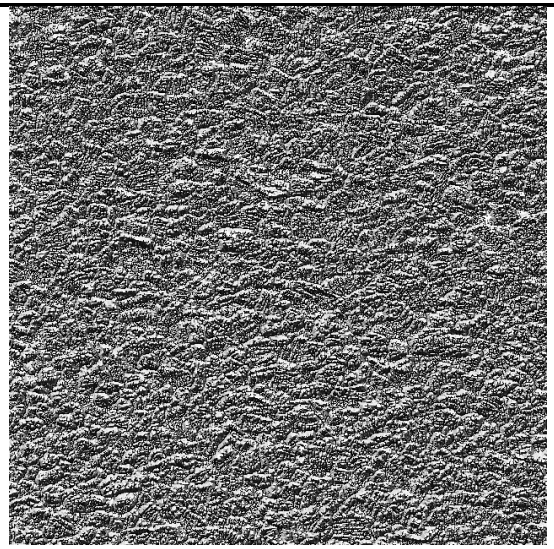
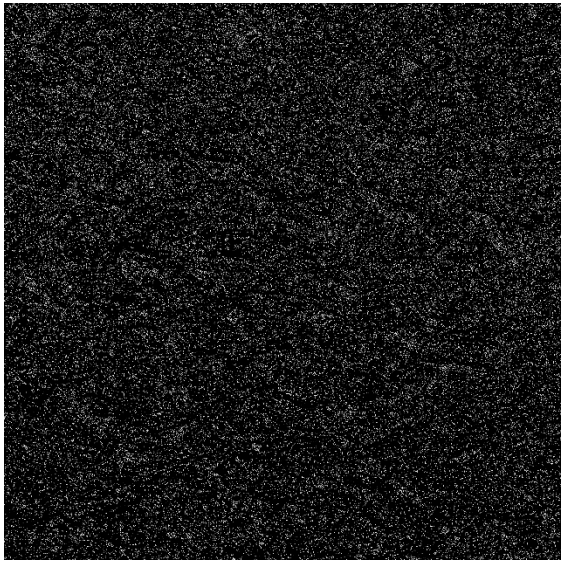
Texture Analysis: classification, segmentation, background subtraction, visual inspection

Image Analysis: interest region description, image forensics, image retrieval, biometrics

Motion Analysis: gesture recognition, lip reading, object detection, gaze tracking

Computer Vision

Course 11

Computer Vision

Course 11

1-original image
2-LBP image
3-Uniform LBP image
4-LBP-ULBP

Histogram of Oriented Gradients (HOG)[3, 4]

- pedestrian detection

Algorithm for computing HOG features:

1. **Gradient** – approximate the two components $\boldsymbol{g_x}$ and $\boldsymbol{g_y}$ of the image I using central differences:

$$g_x(x,y) = I(x,y+1) - I(x,y-1) \;,\; g_y(x,y) = I(x-1,y) - I(x+1,y)$$

The gradient is transformed in polar coordinates, with angles in *[0,180]*

$$r = \sqrt{g_x^2 + g_y^2} \quad , \quad \theta = \frac{180}{\pi}\left(\text{atan2}(g_y, g_x) \bmod \pi\right)$$

[3] N. Dalal, B. Triggs – Histogram of Oriented Gradients for Human Detection, IEEE Con on Computer Vision and Pattern Recognition, v. 1, p. 886-893, 2005

[4] Carlo Tomasi – Histogram of Oriented Gradients

atan2 – the inverse of the tangent function for argument g_y/g_x, which yields values between $[-\pi, \pi]$

2. **Cell Orientation Histograms** Divide the image into adjacent, non-overlapping cells of size $C \times C$ $(C=8)$. Using $B=9$ bins, compute histograms of the gradient angles for each cell. If the image is slightly changed, a pixel whose orientation is close to a bin boundary might contribute to a different bin than in the original image. To prevent this, each pixel in a cell, contributes to two adjacent bins (modulo B) a fraction of the pixel's gradient magnitude

that decreases linearly with the distance of that pixel's gradient orientation from the two bin centers.

The bins are numbered from 0 to *B-1*, and have width $w = \frac{180}{B}$. Bin i has boundaries *[wi, w(i+1)]* and center $c_i = w(i + \frac{1}{2})$. A pixel with magnitude of the gradient r and orientation θ contributes a vote

$$v_j = r\frac{c_{j+1} - \theta}{w} \quad \text{to bin number } j = \left\lfloor \frac{\theta}{w} - \frac{1}{2} \right\rfloor \bmod B$$

and a vote

$$v_{j+1} = r\frac{\theta - c_j}{w} \quad \text{to bin number } (j+1) \bmod B.$$

This scheme is called *voting by bilinear interpolation.* The resulting cell histograms is a vector with B nonnegative values.

3. **Block Normalization** – Group the cells into overlapping blocks of 2×2 cells each. Each block will have $2C\times2C$ size. Two horizontally or vertically consecutive block overlap by two cells. Thus, each internal cell will be o member of four blocks. Concatenate the four cell histograms in each block into a single block feature b, and normalize the vector b using the Euclidean norm

$$b \leftarrow \frac{1}{\sqrt{\| b \|^2 + \varepsilon}} b$$

The small positive constant $\varepsilon > 0$ is used in order to avoid division by zero.

Block normalization reduces the effect of changes in contrast between images of the same object.

4. **HOG features** – the normalized block features are concatenated into a single HOG feature vector *h* which is normalized in the following way:

$$h \leftarrow \frac{1}{\sqrt{\| h \|^2 + \varepsilon}} h$$

$$h_{\max} \leftarrow \min\{h_{\max}, \tau\}$$

$$h \leftarrow \frac{1}{\sqrt{\| h \|^2 + \varepsilon}} h$$

h_{max} are components of h of maximal magnitude, and τ is a positive threshold ($\tau=0.2$). This thresholding operation ensures that very large gradients do not have too much influence (they would wash out all other image detail). The final normalization makes the HOG feature independent of overall image contrast.

Harris Corner Detector[5]

Harris and Stephens started from the idea that a corner point is a point where the gradient of the image is strong in more than one direction at the same time. Location along edges where the gradient is strong in only direction, should not be considered as corners. The detector should be isotropic (independent of the orientation of the local gradients).

[5] W. Burger, M.J. Burge - Principles of Digital Image Processing - Core Algorithms, Springer, 2009

Local Structure Matrix

Let ***f(x,y)*** be an image. We denote by g_x and g_y the approximations for the components of the gradient of ***f***. Compute the following three values for each pixel ***(x,y)*** in ***f***:

$$A(x,y) = g_x^2(x,y),\ B(x,y) = g_y^2(x,y)\ ,\ C(x,y) = g_x(x,y) \cdot g_y(x,y)$$

and the *local structural matrix* ***M***:

$$M(x,y) = \begin{pmatrix} A(x,y) & C(x,y) \\ C(x,y) & B(x,y) \end{pmatrix}$$

Each of the matrices *A*, *B* and *C* are smoothed by convolving them with a linear Gaussian filter $\boldsymbol{G_\sigma}$ and we build the modified local structural matrix:

$$\bar{M}(x,y)=\begin{pmatrix} G_\sigma * A & G_\sigma * C \\ G_\sigma * C & G_\sigma * B \end{pmatrix}=\begin{pmatrix} \bar{A}(x,y) & \bar{C}(x,y) \\ \bar{C}(x,y) & \bar{B}(x,y) \end{pmatrix}.$$

Since the *2x2* matrix $\bar{M}(x,y)$ is symmetric, it can be diagonalized to:

$$\tilde{M}=\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

where λ_1 and λ_2 are the eigenvalues of $\tilde{M}$:

$$\lambda_{1,2} = \frac{1}{2}\left(\bar{A} + \bar{B} \pm \sqrt{\bar{A}^2 - 2\bar{A}\bar{B} + \bar{B}^2 + 4\bar{C}^2} \right).$$

These eigenvalues, which are nonnegative, contain essential information about local image structure. In a region that is uniform, $\bar{M} = 0$ and $\lambda_1 = \lambda_2 = 0$. On an ideal ramp, these eigenvalues are $\lambda_1 > 0$ and $\lambda_2 = 0$, independent of the orientation of the edge.

The *eigenvalues* thus encode an edge's *strength*, and their associated *eigenvectors* represent the edge's *orientation*.

A corner should have a strong edge in the main direction (corresponding to the larger of the two eigenvalues), another edge normal to the first (corresponding to the smaller eigenvalue), and both eigenvalues must be significant. Since $\bar{A}, \bar{B} \geq 0$ we can assume that $trace(\bar{M}) > 0$ and thus $|\lambda_1| \geq |\lambda_2|$. Therefore, only the smaller of the two eigenvalues, λ_2 is relevant for corner detection in an image.