Chapter 3:
Image Compression

3. Image Compression (IC)

- **Motivation:**

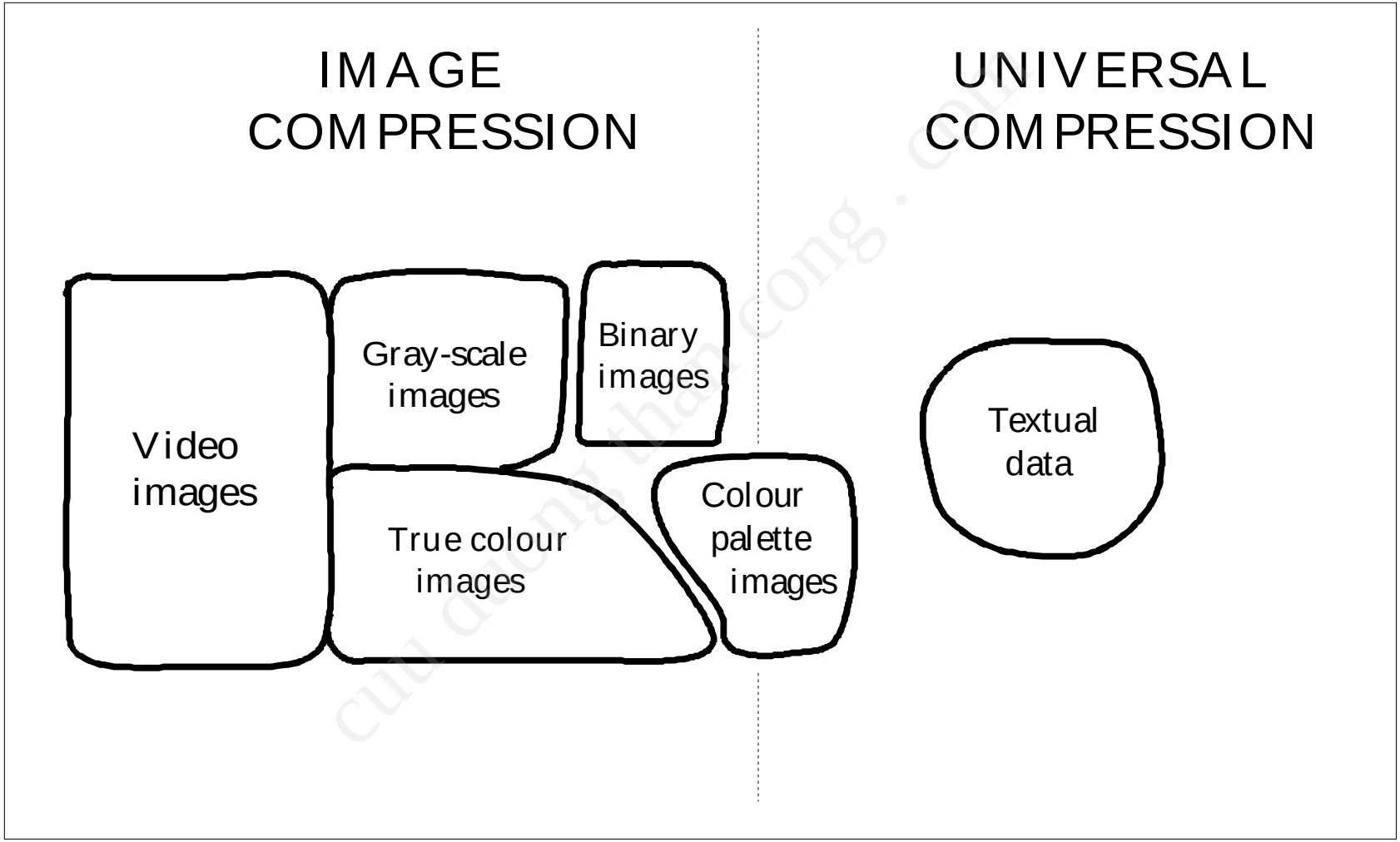
Still Image:

– One page of A4 format at 600 dpi is > 100 MB.
– One color image in digital camera generates 10-30 MB.
– Scanned 3” × 7” photograph at 300 dpi is 30 MB.

Digital Cinema:

– 4K × 2K × 3 × 12 bits/pel = 48 MB/frame or 1 GB/sec
or 70 GB/min.

3. IC: Image Types



3. IC: Lossless vs. Lossy (1)

- **Lossless compression**: reversible, information preserving, text compression algorithms, binary images, palette images.
- **Lossy compression**: irreversible, grayscale, color, video.
- **Near-lossless compression**: medical imaging, remote sensing.

    1) Why do we need lossy compression?
    2) When we can use lossy compession?

3. IC: Lossless vs. Lossy (2)

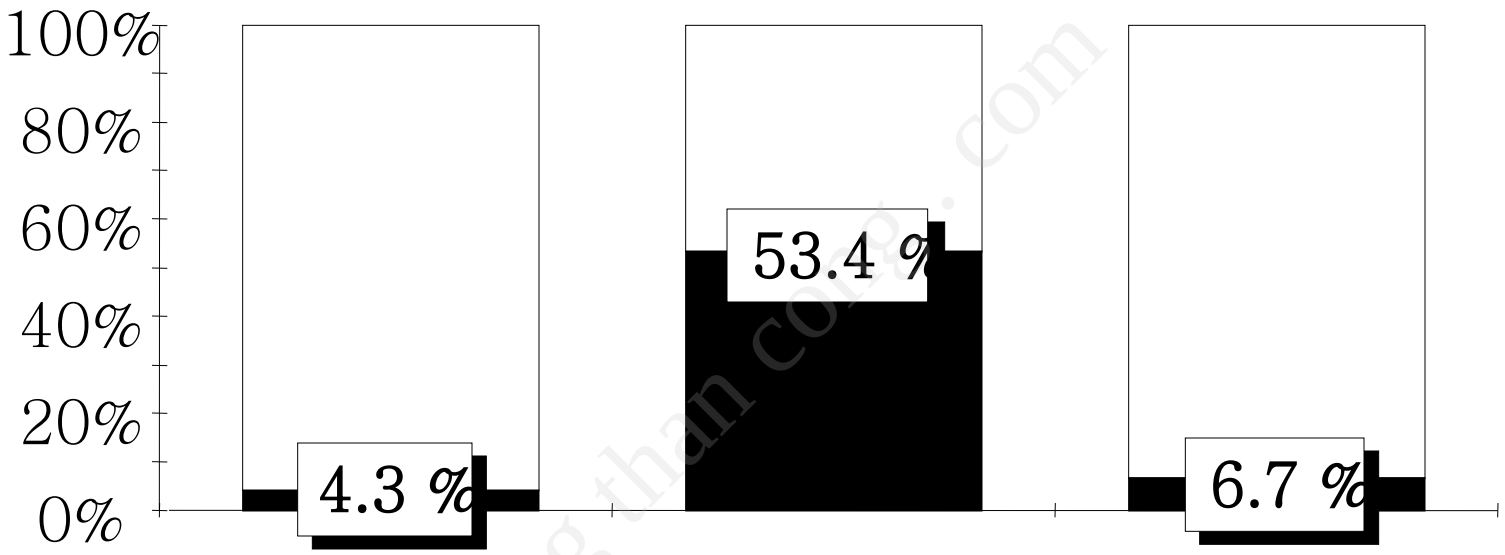


IMAGE:	CCITT-3	LENA	LENA
TYPE:	binary	gray-scale	gray-scale
METHOD:	JBIG (lossless)	JPEG (lossless)	JPEG (lossy)

3. IC: Redundancy (1)

- **Coding redundancy**

  Most 2-D intensity arrays contain more bits than are needed to represent the intensities.

- **Spatial and temporal redundancy**

  Pixels of most 2-D intensity arrays are correlated spatially and video sequences are temporally correlated.

- **Irrelevant information**

  Most 2-D intensity arrays contain information that is ignored by the human visual system.

3. IC: Redundancy (2)

Example of redundancy:

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.)

3. IC: Redundancy (3)

Example of Spatial and temporal redundancy:

1. All 256 intensities are equally probable.
2. The pixels along each line are identical.
3. The intensity of each line was selected randomly.

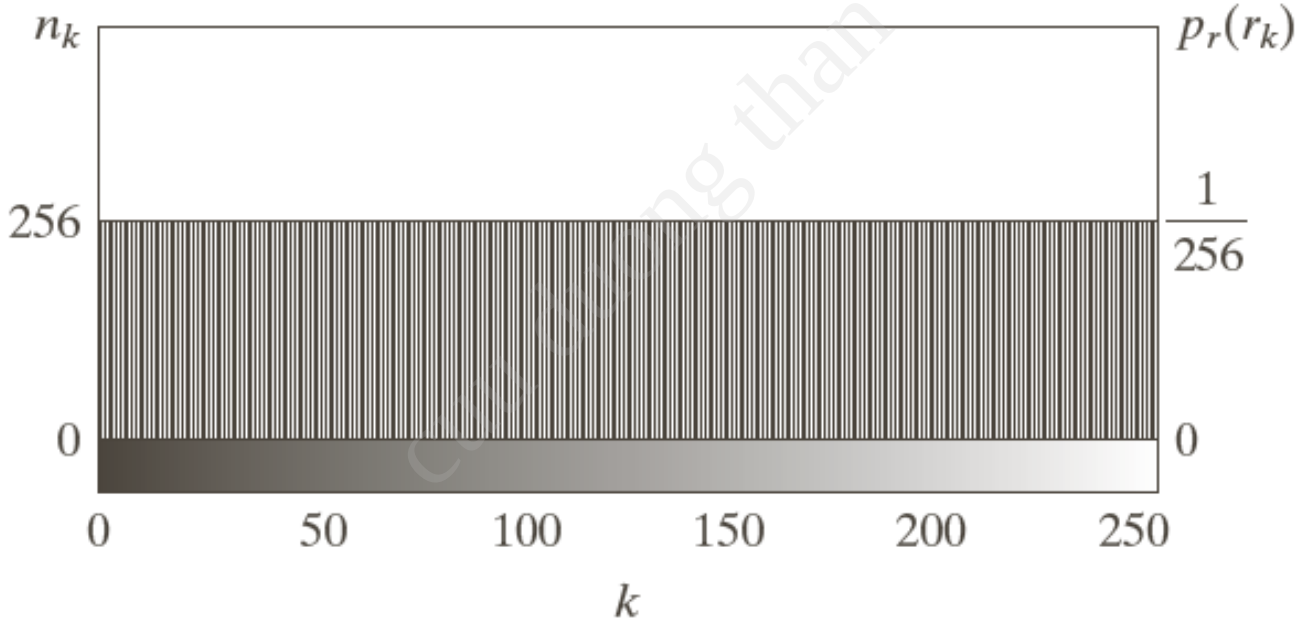


FIGURE 8.2 The intensity histogram of the image in Fig. 8.1(b).

3. IC: Measuring Image Information (1)

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.

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

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

a_j is called source symbols. Because they are statistically independent, the source called *zero – memory source.*

3. IC: Measuring Image Information (2)

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

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

$p_r(r_k)$ is the normalized histogram.

3. IC: Rate Measures

$$\text{Bitrate:} \quad \frac{\text{size of the compressed file}}{\text{pixels in the image}} = \frac{C}{N} \quad \text{bits/pixel}$$

$$\text{Compression ratio:} \quad \frac{\text{size of the original file}}{\text{size of the compressed file}} = \frac{N \cdot k}{C}$$

3. IC: Fidelity Criteria (1)

Let $f(x, y)$ be an input image and $\hat{f}(x, y)$ be an approximation of $f(x, y)$. The images are of size $M \times N$.

The *root - mean - square error* is

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

3. IC: Fidelity Criteria (2)

The *mean - square signal - to - noise ratio* of the output image, denoted SNR_{ms}

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

3. IC: Fidelity Criteria (3)

RMSE = 5.17 RMSE = 15.67 RMSE = 14.17

a b c

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

3. IC: Image Compression Models

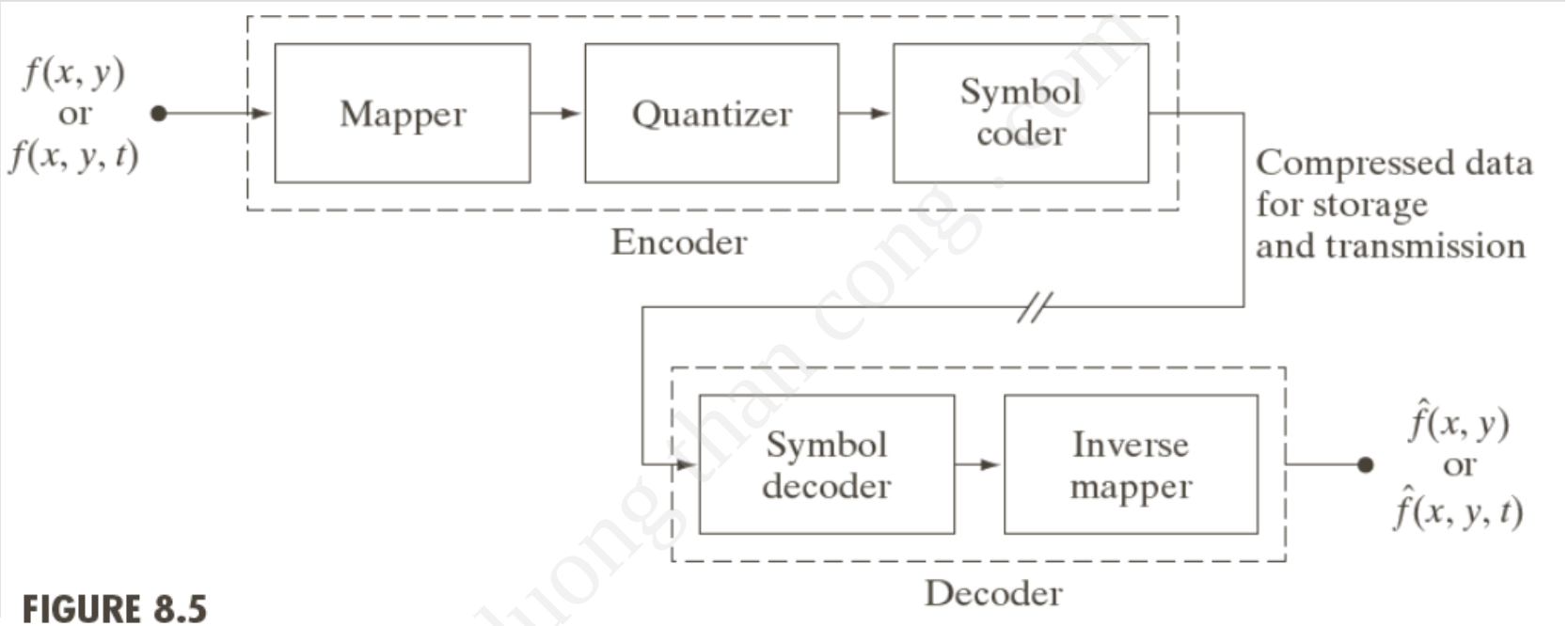


FIGURE 8.5
Functional block diagram of a general image compression system.

3. IC: Image Compression Standards

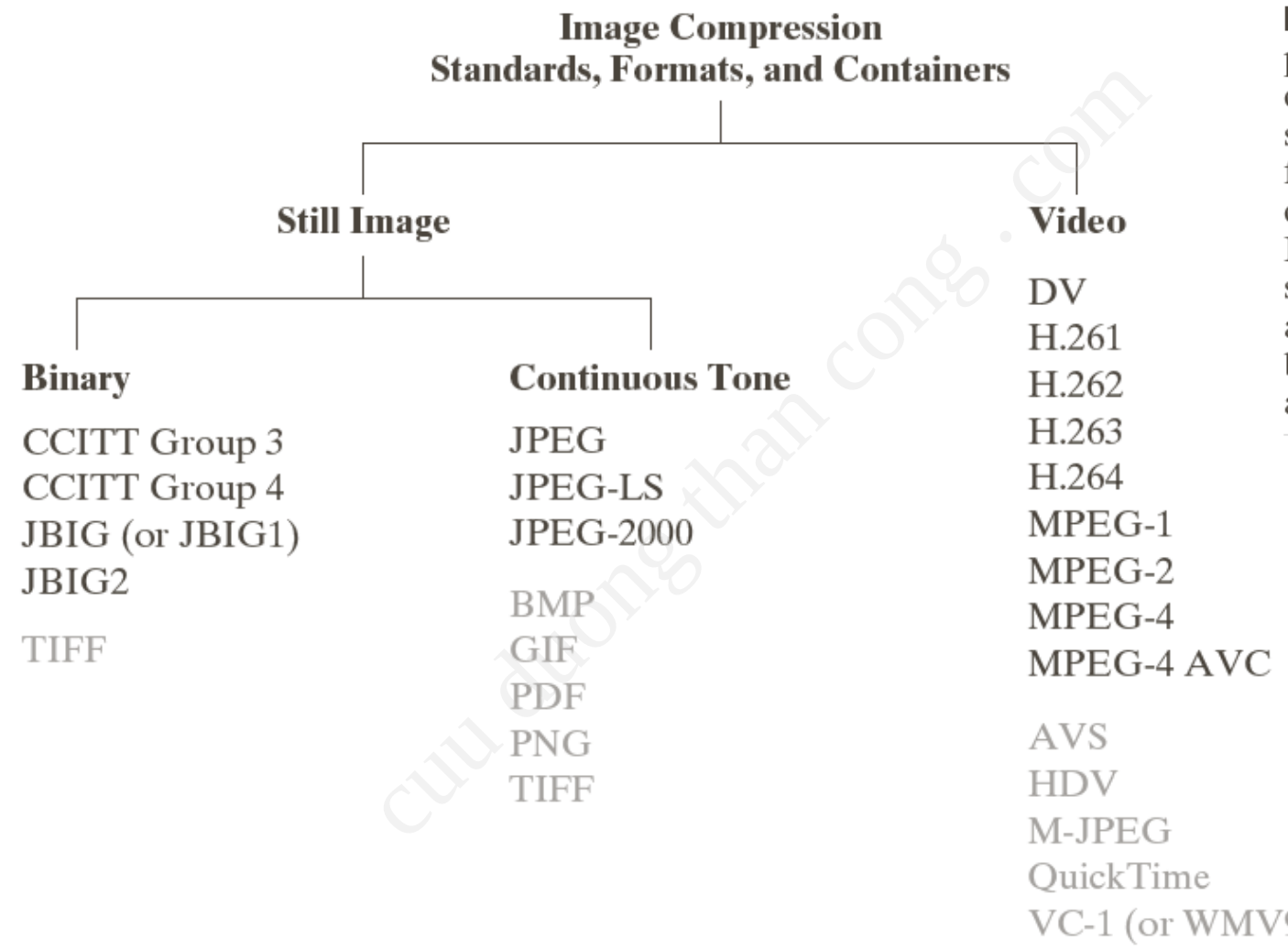


FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

3. IC: Huffman Coding (1)

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.

3. IC: Huffman Coding (2)

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 is

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

$$= 2.2 \text{ bits/pixel}$$

3. IC: LZW Coding (Dictionary Coding) (1)

- **LZW** (**Lempel-Ziv-Welch**) coding, assigns fixed-length code words to variable length sequences of source symbols, but requires no **a priori** knowledge of the probability of the source symbols.
- LZW is used in:
  - Tagged Image file format (TIFF)
  - Graphic interchange format (GIF)
  - Portable document format (PDF)
- LZW was formulated in 1984.

3. IC: LZW Coding (2)

- **Algorithm:**
  - A codebook or "dictionary" containing the source symbols is constructed.
  - For 8-bit monochrome images, the first 256 words of the dictionary are assigned to the gray levels 0 - 255.
  - Remaining part of the dictionary is filled with sequences of the gray levels.

3. IC: LZW Coding (3)

- **Important features of LZW:**
  - The dictionary is created while the data are being encoded. So encoding can be done “on the fly”.
  - The dictionary is not required to be transmitted. The dictionary will be built up in the decoding.
  - If the dictionary “overflows” then we have to reinitialize the dictionary and add a bit to each one of the code words.
  - Choosing a large dictionary size avoids overflow, but spoils compressions.

3. IC: LZW Coding (4)

Example:

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	—

3. IC: LZW Coding (5)

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		

3. IC: LZW Coding (6)

- **Decoding LZW:**

Let the bit stream received be:

39 39 126 126 256 258 260 259 257 126

In LZW, the dictionary which was used for encoding need not be sent with the image. A separate dictionary is built by the decoder, on the “fly”, as it reads the received code words.

3. IC: LZW Coding (7)

Recognized	Encoded value	pixels	Dictionary address	Dictionary entry
	39	39		
39	39	39	256	39-39
39	126	126	257	39-126
126	126	126	258	126-126
126	256	39-39	259	126-39
256	258	126-126	260	39-39-126
258	260	39-39-126	261	126-126-39
260	259	126-39	262	39-39-126-126
259	257	39-126	263	126-39-39
257	126	126	264	39-126-126

3. IC: Run-Length Coding (1)

- **Run-length encoding**, or **RLE** is a technique used to **reduce the size of a repeating string of characters**.
- This repeating string is called a **run**, typically RLE encodes a run of symbols into two bytes, a **count** and a **symbol**.
- RLE can compress any type of data.
- RLE can not achieve high compression ratios compared to other compression methods.
- It is easy to implement and is quick to execute.
- Run-length encoding is supported by most bitmap file formats such as TIFF, BMP and PCX.

3. IC: Run-Length Coding (2)

Example:

WWWWWWWWWWWWBWWWWWWWWWWWWWW
BBBWWWWWWWWWWWWWWWWWWWWWWWW
WBWWWWWWWWWWWWWW

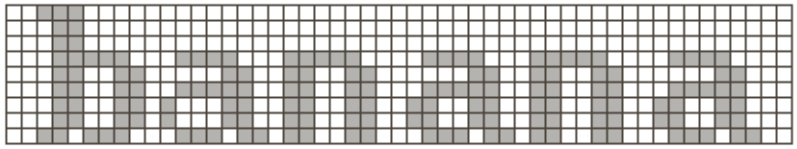
RLE coding: 12W1B12W3B24W1B14W

3. IC: Symbol-Based Coding (1)

- In **symbol-** or **token-based coding**, an image is represented as a collection of frequently occurring sub-images, called symbols.
- Each symbol is stored in a symbol dictionary.
- Image is coded as a set of triplets:

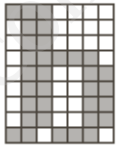
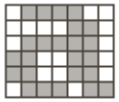
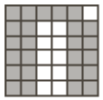
$$\{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots\}$$

3. IC: Symbol-Based Coding (2)



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.

Token	Symbol
0	
1	
2	

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

3. IC: Bit-Plane Coding (1)

- An *m*-bit gray scale image can be converted into *m* binary images by bit-plane slicing. These individual images are then encoded using run-length coding.
- Code the bit planes separately, using RLE (flatten each plane row-wise into a 1D array), or any other lossless compression technique.
- Let *I* be an image where every pixel value is *n*-bit long.
- Express every pixel in binary using *n* bits.
- Form *n* binary matrices (called bit planes), where the *i*-th matrix consists of the *i*-th bits of the pixels of *I*.

3. IC: Bit-Plane Coding (2)

Example:

Let I be the following 2×2 image where the pixels are 3 bits long:

101 110

111 011

The corresponding 3 bit planes are:

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

3. IC: Bit-Plane Coding (4)

- However, a small difference in the gray level of adjacent pixels can cause a disruption of the run of zeroes or ones.

  Example: Let us say one pixel has a gray level of 127 and the next pixel has a gray level of 128.

  In binary: 127 = 01111111 & 128 = 10000000

  Therefore a small change in gray level has decreased the run-lengths in all the bit-planes.

3. IC: Bit-Plane Coding (5)

- **Gray coded** images are free of this problem which affects binary images (see more in the text book [1]).

  In gray code, the **representation of adjacent gray levels will differ only in one bit** (unlike binary format where all the bits can change.

  Let $g_{m\text{-}1}\ldots g_1 g_0$ represent the gray code representation of a binary number, then:

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

$$g_{m-1} = a_{m-1}$$

  In gray code:

  127 = 01000000

  128 = 11000000

3. IC: Bit-Plane Coding (6)

- To convert a binary number $b_1\, b_2\, b_3 \ldots\, b_{n-1}\, b_n$ to its corresponding binary reflected Gray code:
  - Start at the right with the digit b_n. If the b_{n-1} is 1, replace b_n by 1-b_n; otherwise, leave it unchanged. Then proceed to b_{n-1}.
  - Continue up to the first digit b_1, which is kept the same since it is assumed to be a $b_0 = 0$.
  - The resulting number is the reflected binary Gray code.

3. IC: Bit-Plane Coding (7)

Example: Gray coding

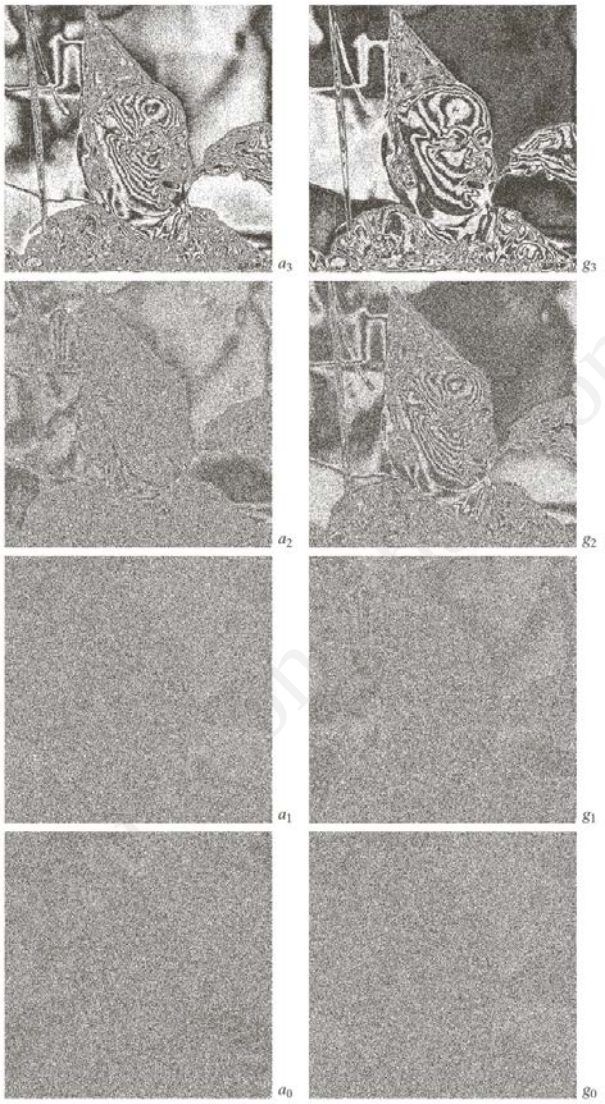
Dec	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

3. IC: Bit-Plane Coding (8)

a b
c d
e f
g h

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

3. IC: Bit-Plane Coding (9)



a b
c d
e f
g h

FIGURE 8.20
(a)–(h) The four least significant binary (left column) and Gray-coded (right column) bit planes of the image in Fig. 8.19(a).

3. IC: Bit-Plane Coding (10)

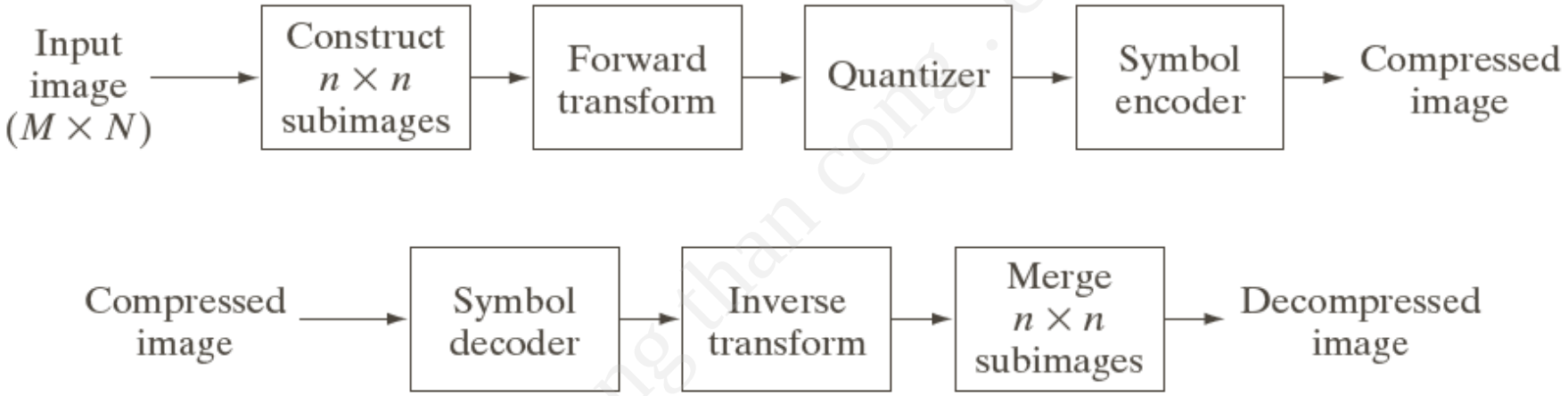
- **Decoding a gray coded image:**

The MSB is retained as such, i.e.,

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

$$a_{m-1} = g_{m-1}$$

3. IC: Block Transform Coding (1)



3. IC: Block Transform Coding (2)

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

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

for $u, v = 0, 1, 2, \ldots, n\text{-}1$.

3. IC: Block Transform Coding (3)

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

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

for $x, y = 0,1,2,...,n\text{-}1$.

3. IC: Block Transform Coding (4)

- **Discrete Cosine Transform (DCT):**

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

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

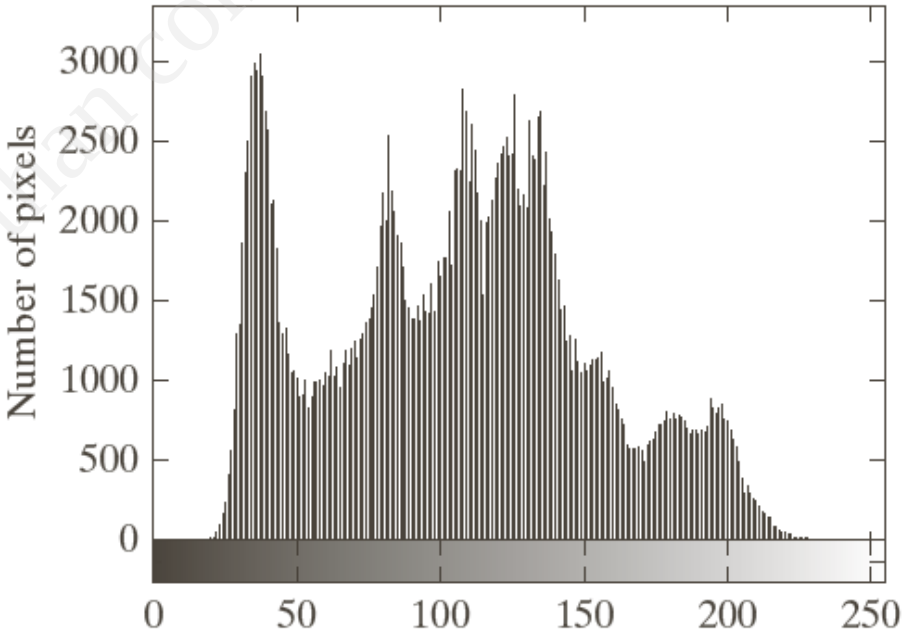
$$\text{where } \alpha(u/v) = \begin{cases} \sqrt{\frac{1}{n}} & \text{for } u/v = 0 \\ \sqrt{\frac{2}{n}} & \text{for } u/v = 1,2,\ldots,n-1 \end{cases}$$

3. IC: Block Transform Coding (5)

Example:

a b

FIGURE 8.9 (a) A 512 × 512 8-bit image, and (b) its histogram.



3. IC: Block Transform Coding (6)

In each case, 50% of the resulting coefficients were truncated and taking the inverse transform of the truncated coefficients arrays.

a b c
d e f

RMSE = 2.32 RMSE = 1.78 RMSE = 1.13

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).

3. IC: Block Transform Coding (7)

- **Subimage size selection:**

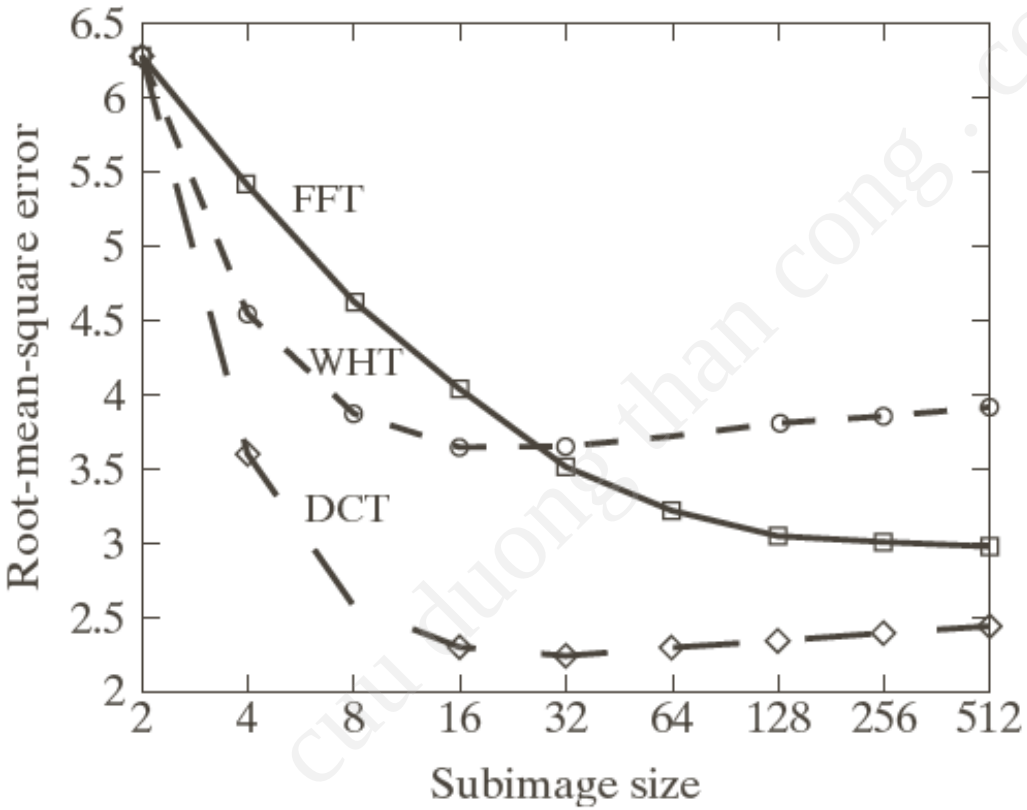
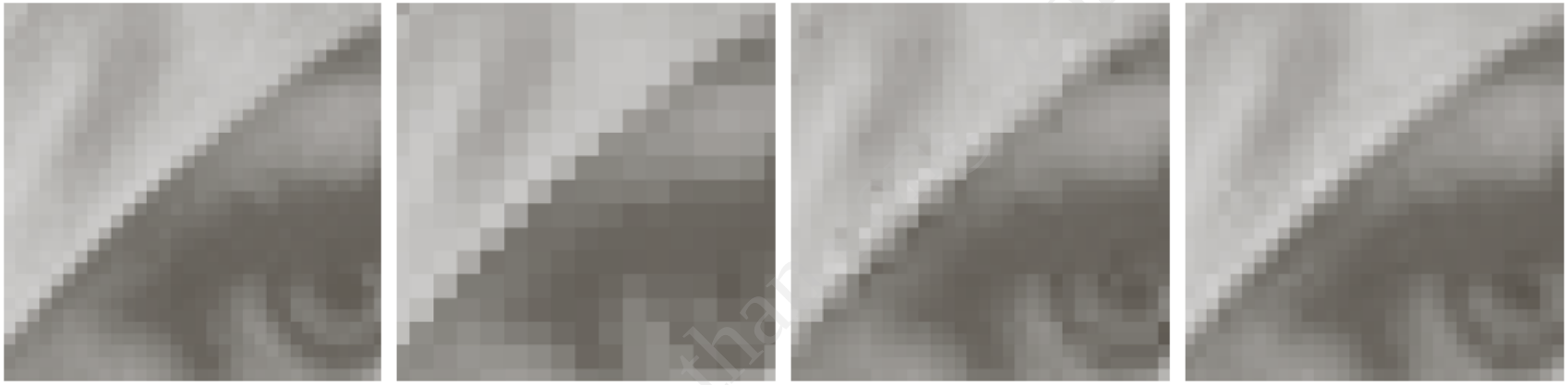


FIGURE 8.26 Reconstruction error versus subimage size.

3. IC: Block Transform Coding (8)



a b c d

FIGURE 8.27 Approximations of Fig. 8.27(a) using 25% of the DCT coefficients and (b) 2 × 2 subimages, (c) 4 × 4 subimages, and (d) 8 × 8 subimages. The original image in (a) is a zoomed section of Fig. 8.9(a).

3. IC: Block Transform Coding (9)

- **Bit allocation:**

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

  Zonal coding

  The retained coefficients are selected on the basis of maximum variance.

  Threshold coding

  The retained coefficients are selected on the basis of maximum magnitude.

3. IC: Block Transform Coding (10)

RMSE = 4.5

RMSE = 6.5

a b
c d

FIGURE 8.28 Approximations of Fig. 8.9(a) using 12.5% of the 8×8 DCT coefficients: (a)—(b) threshold coding results; (c)—(d) zonal coding results. The difference images are scaled by 4.

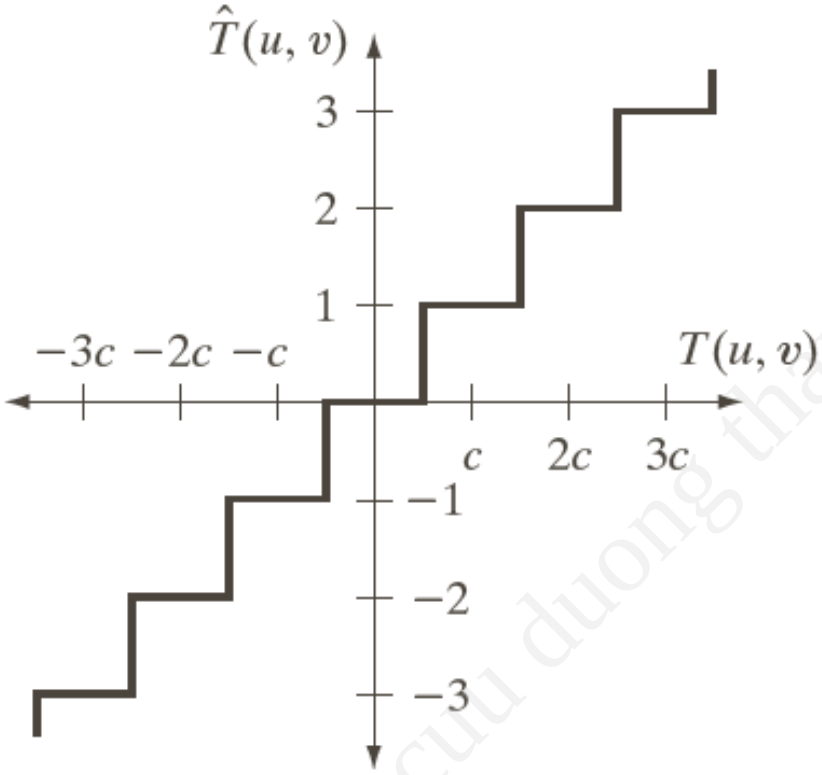
3. IC: Block Transform Coding (11)

- **Threshold coding:**

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

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

3. IC: Block Transform Coding (12)



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

3. IC: Block Transform Coding (13)

FIGURE 8.31 Approximations of Fig. 8.9(a) using the DCT and normalization array of Fig. 8.30(b): (a) **Z**, (b) 2**Z**, (c) 4**Z**, (d) 8**Z**, (e) 16**Z**, and (f) 32**Z**.

3. IC: JPEG Compression (1)

- JPEG stands for Joint Photographic Experts Group
  - Used on 24-bit color files. Works well on photographic images. Although it is a lossy compression technique, it yields an **excellent quality image with high compression rates.**

- It defines three different coding systems:
  - A lossy baseline coding system, adequate for most compression applications.
  - An extended coding system for greater compression, higher precision, or progressive reconstruction applications.
  - A lossless independent coding system for reversible compression.

3. IC: JPEG Compression (2)

- **Steps in JPEG compression:**
  - (Optionally) If the color is represented in RGB mode, translate it to YUV.
  - Divide the file into 8×8 blocks.
  - Transform the pixel information from the spatial domain to the frequency domain with the **Discrete Cosine Transform (DCT)**.
  - Quantize the resulting values by dividing each coefficient by an integer value and **rounding off to the nearest integer**.
  - Look at the resulting coefficients in a **zigzag order**. Do a **run-length encoding** of the coefficients ordered in this manner. Follow by **Huffman coding**.

3. IC: JPEG Compression (3)

Step 1a: Converting RGB to YUV

- YUV color mode stores color in terms of its luminance (brightness) and chrominance (hue).
- The human eye is less sensitive to chrominance than luminance.
- YUV is not required for JPEG compression, but it gives a **better compression rate**.
- It's simple arithmetic to convert RGB to YUV. The formula is based on the relative contributions that red, green, and blue make to the luminance and chrominance factors.
- There are several different formulas in use depending on the target monitor.

3. IC: JPEG Compression (4)

For example:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

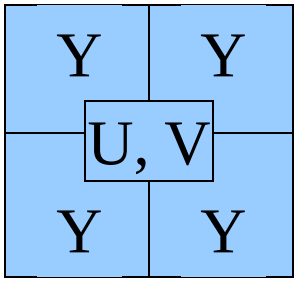
$$U = -0.1687 * R - 0.3313 * G + 0.5 * B + 128$$

$$V = 0.5 * R - 0.4187 * G - 0.813 * B + 128$$

3. IC: JPEG Compression (5)

Step 1b: Down-sampling

- The chrominance information can (optionally) be down-sampled.
- The notation 4:1:1 means that for each block of four pixels, you have 4 samples of luminance information (Y), and 1 each of the two chrominance components (U and V).



3. IC: JPEG Compression (6)

Step 2: Divide into 8 × 8 blocks

- Note that with YUV color, you have 16 pixels of information in each block for the Y component (though only 8 in each direction for the U and V components).
- If the file doesn't divide evenly into 8 × 8 blocks, extra pixels are added to the end and discarded after the compression.
- The values are shifted "left" by subtracting 128.

3. IC: JPEG Compression (7)

Step 3: Discrete cosine transform (DCT)

- The DCT transforms the data from the spatial domain to the frequency domain.
- The spatial domain shows the amplitude of the color as you move through space.
- The frequency domain shows how quickly the amplitude of the color is changing from one pixel to the next in an image file.
- The frequency domain is a better representation for the data because it makes it possible for you to separate out – and throw away – information that isn't very important to human perception.

3. IC: JPEG Compression (8)

- The human eye is not very sensitive to high frequency changes – especially in photographic images, so the high frequency data can, to some extent, be discarded.
- The color amplitude information can be thought of as a wave (in two dimensions).
- You're decomposing the wave into its component frequencies.
- For the 8×8 matrix of color data, you're getting an 8×8 matrix of coefficients for the frequency components.

3. IC: JPEG Compression (9)

Step 4: Quantize the coefficients computed by the DCT

- The DCT is lossless in that the reverse DCT will give back exactly your initial information (ignoring the rounding error that results from using floating point numbers).
- The values from the DCT are initially floating-point.
- They are changed to integers by quantization.
- **Quantization** involves dividing each coefficient by an integer between 1 and 255 and rounding off.
  - The quantization table is chosen to reduce the precision of each coefficient to no more than necessary.
  - The quantization table is carried along with the compressed file.

3. IC: JPEG Compression (10)

Step 5: Arrange in "zigzag" order

- This is done so that the coefficients are in order of increasing frequency.
- The higher frequency coefficients are more likely to be 0 after quantization.
- This improves the compression of run-length encoding.
- Do run-length encoding and Huffman coding.

3. IC: JPEG Compression (11)

Example:

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	63	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

EXAMPLE 8.17: JPEG baseline coding and decoding.

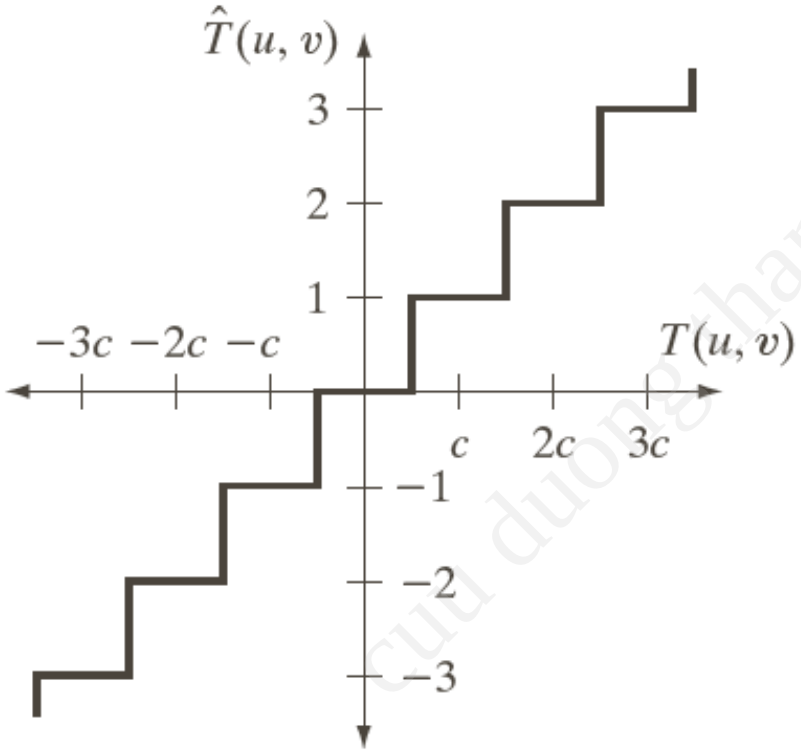
3. IC: JPEG Compression (12)

After forward DCT:

−415	−29	−62	25	55	−20	−1	3
7	−21	−62	9	11	−7	−6	6
−46	8	77	−25	−30	10	7	−5
−50	13	35	−15	−9	6	0	3
11	−8	−13	−2	−1	1	−4	1
−10	1	3	−3	−1	0	2	−1
−4	−1	2	−1	2	−3	1	−2
−1	−1	−1	−2	−1	−1	0	−1

3. IC: JPEG Compression (13)

Using threshold coding quantization with the normalization matrix $Z(u, v)$ as:



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

3. IC: JPEG Compression (14)

After quantization we obtain:

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3. IC: JPEG Compression (15)

Using zigzag ordering pattern as:

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

3. IC: JPEG Compression (16)

The resulting 1D coefficient sequence is

[−26 −3 1 −3 −2 −6 2 −4 1 −4 1 1 5 0 2 0 0 −1 2 0 0 0 0 0 −1 −1 EOB]

3. IC: JPEG Compression (16)

After denormalization, we obtain:

−416	−33	−60	32	48	0	0	0
12	−24	−56	0	0	0	0	0
−42	13	80	−24	−40	0	0	0
−56	17	44	−29	0	0	0	0
18	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

3. IC: JPEG Compression (18)

After inverse DCT, we obtain the reconstructed subimage:

58	64	67	64	59	62	70	78
56	55	67	89	98	88	74	69
60	50	70	119	141	116	80	64
69	51	71	128	149	115	77	68
74	53	64	105	115	84	65	72
76	57	56	74	75	57	57	74
83	69	59	60	61	61	67	78
93	81	67	62	69	80	84	84

3. IC: JPEG Compression (19)

The differences between the original and reconstructed subimage are:

−6	−9	−6	2	11	−1	−6	−5
7	4	−1	1	11	−3	−5	3
2	9	−2	−6	−3	−12	−14	9
−6	7	0	−4	−5	−9	−7	1
−7	8	4	−1	6	4	3	−2
3	8	4	−4	2	6	1	1
2	2	5	−1	−6	0	−2	5
−6	−2	2	6	−4	−4	−6	10

3. IC: JPEG Compression (20)

a b c
d e f

FIGURE 8.32 Two JPEG approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image.