CHAPTER 6

Image Compression

Introduction

- Image compression
  - To Solve the problem of reducing the amount of data required to represent a digital image
- Why do we need compression?
  - Data storage
  - Data transmission

Introduction

- Big data: 1000*1000*24bit*30fps*60*120=603 GB
- Image compression techniques fall into two broad categories:
  - ***Information preserving***: These methods allow an image to be compressed and decompressed without lossing information.
  - ***Information lossing***: These methods provide higher levels of data reduction but the result in a less than perfect reproduction of original image.

Introduction

- **How can we implement Compresion**
  - **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

Introduction

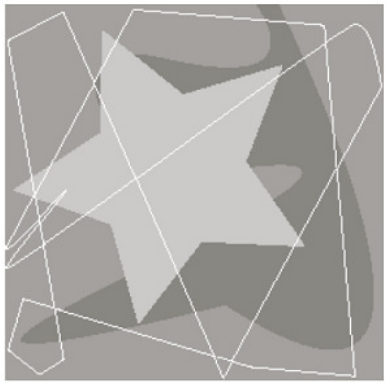
a b c

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

- Coding Redundancy

TABLE 8.1
Example of variable-length coding.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	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



The average number of bits required to represent each pixel is

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)\, p_r(r_k) = 0.25(2) + 0.47(1) + 0.24(3) + 0.03(3) = 1.81 bits$$

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

$$R = 1 - 1/4.42 = 0.774$$

Fundamentals

- Spatial and Temporal Redundancy

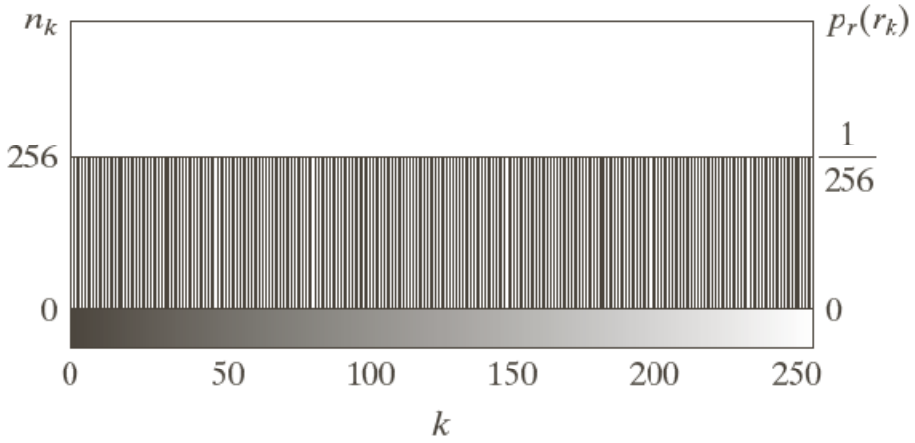


FIGURE 8.2 The intensity histogram of the image in Fig. 8.1(b).

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

Fundamentals

- Spatial and Temporal Redundancy

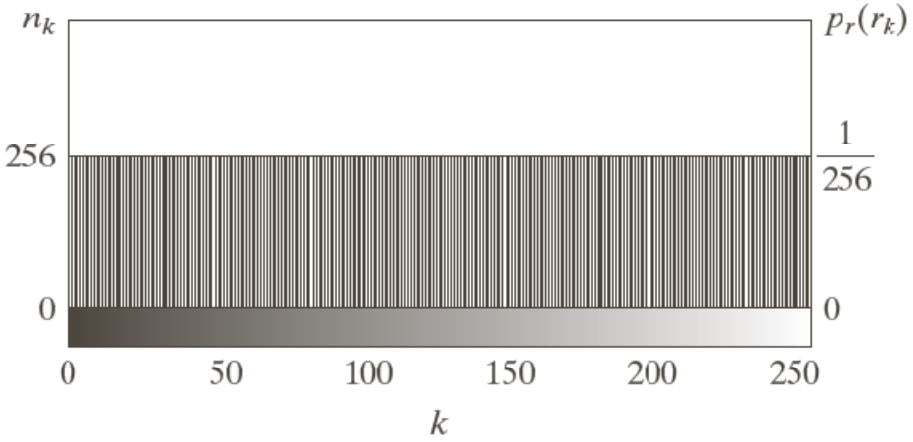


FIGURE 8.2 The intensity histogram of the image in Fig. 8.1(b).

Run-length pair specifies the start of a new intensity and the number of consecutive pixels that have that intensity.
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.

The compression ratio is

$$\frac{256\times256\times8}{(256+256)\times8}=128:1$$

Fundamentals

- Irrelevant information

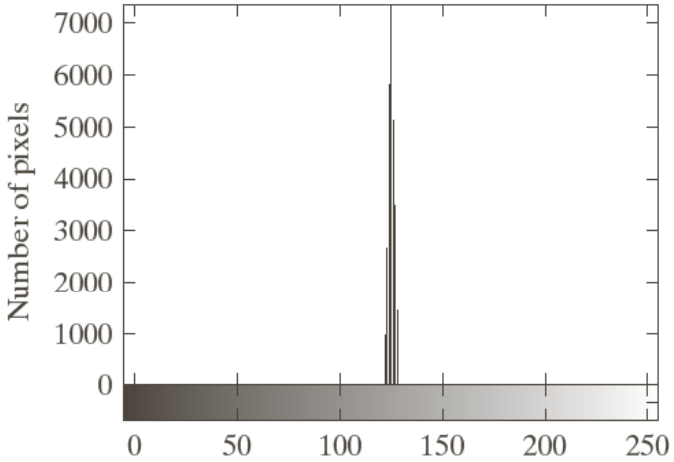


FIGURE 8.3
(a) Histogram of the image in Fig. 8.1(c)

$$256 \times 256 \times 8 / 8$$
$$= 65536 : 1$$

The Flow of Image Compression (1/2)

- What is the so-called image compression coding?
  - To store the image into bit-stream as compact as possible and to display the decoded image in the monitor as exact as possible
- Flow of compression
  - The image file is converted into a series of binary data, which is called the bit-stream
  - The decoder receives the encoded bit-stream and decodes it to reconstruct the image
  - The total data quantity of the bit-stream is less than the total data quantity of the original image

The Flow of Image Compression (2/2)

- Measure to evaluate the performance of image compression
  - Root Mean square error: $$RMSE = \sqrt{\frac{\sum_{x=0}^{W-1}\sum_{y=0}^{H-1}[f(x,y) - f'(x,y)]^2}{WH}}$$
  - Peak signal to noise ratio: $$PSNR = 20\log_{10}\frac{255}{MSE}$$
  - Compression Ratio: $$Cr = \frac{n1}{n2}$$

    Where *n1* is the data rate of original image and *n2* is that of the encoded bit-stream
- The flow of encoding
  - Reduce the correlation between pixels
  - Quantization
  - Source Coding

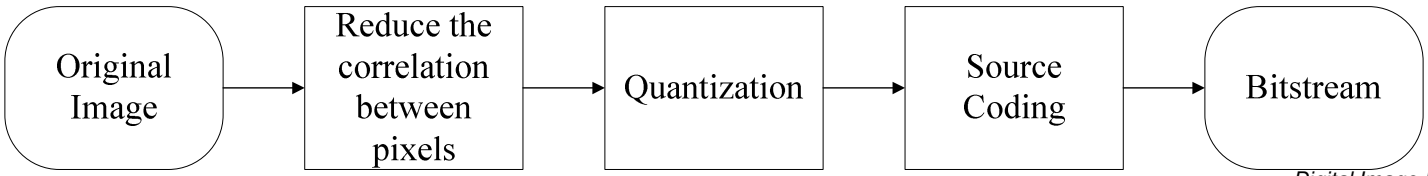


Image Compression models

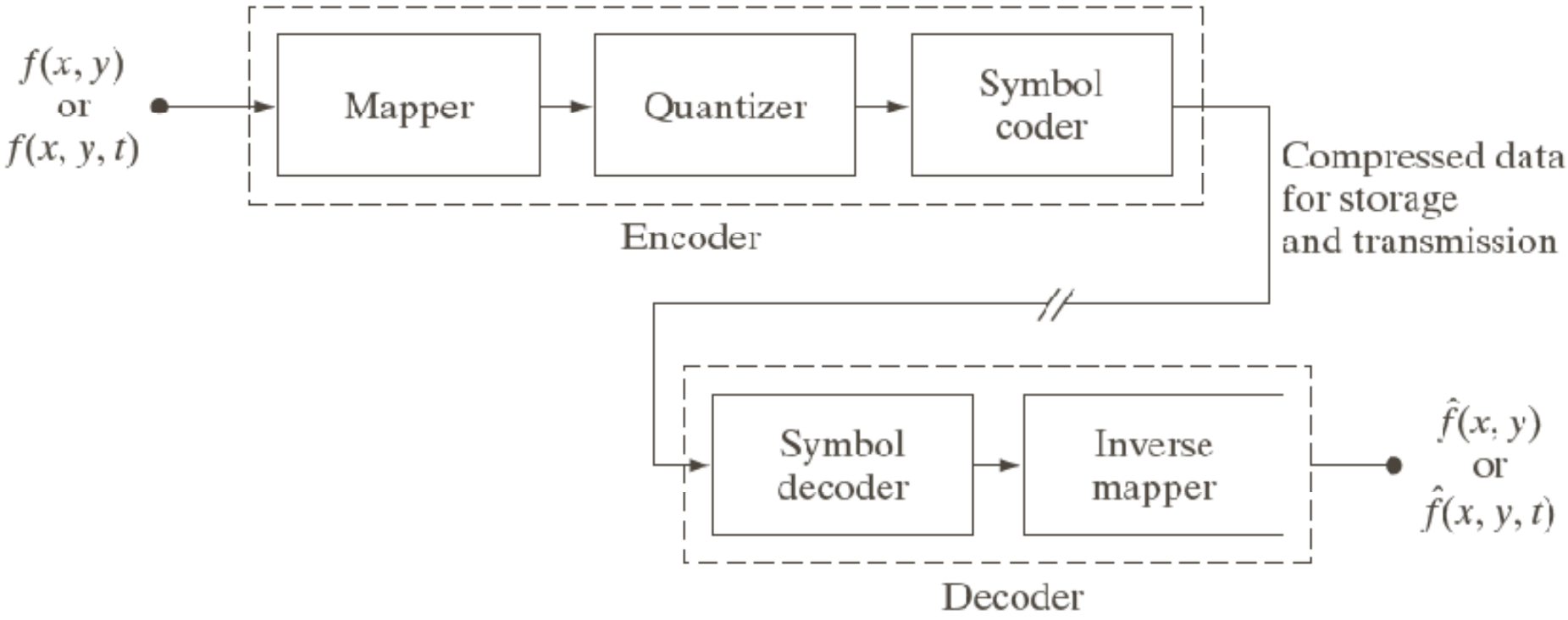


Image Compression models

- Some standards

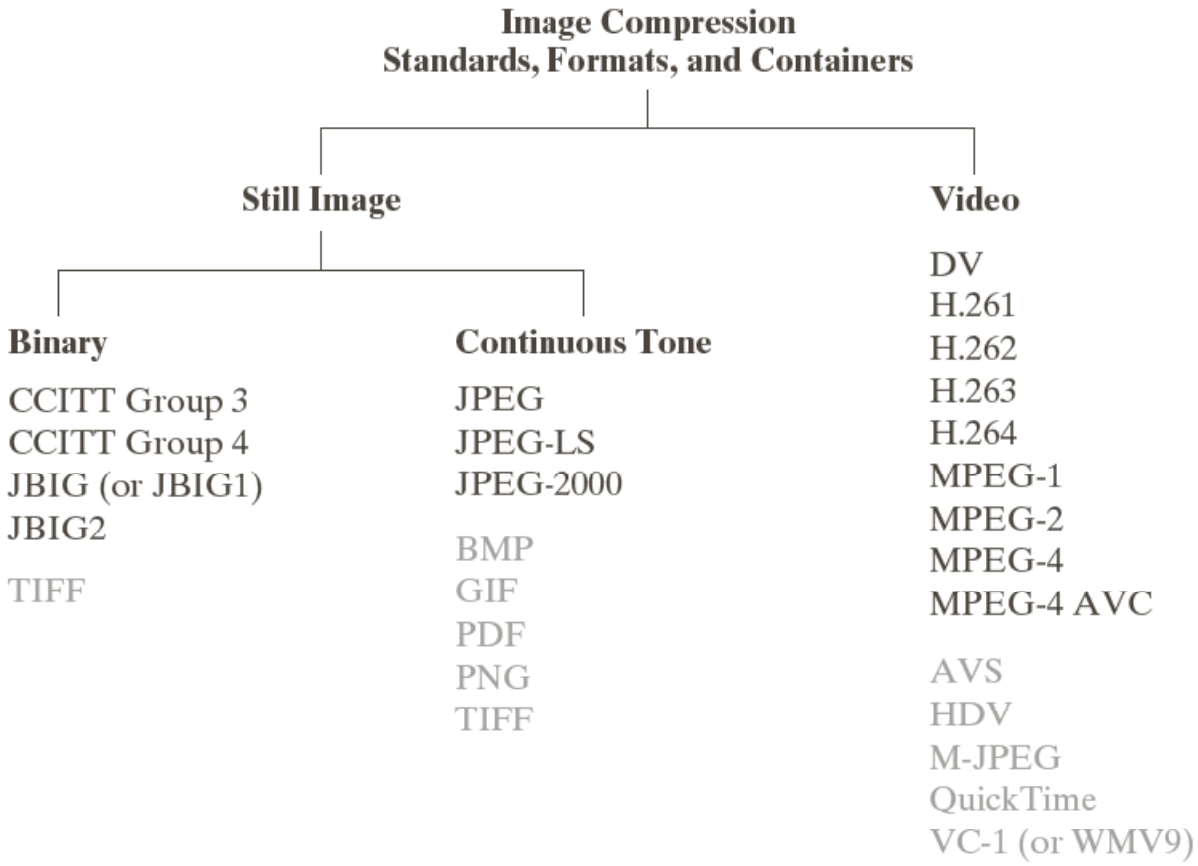
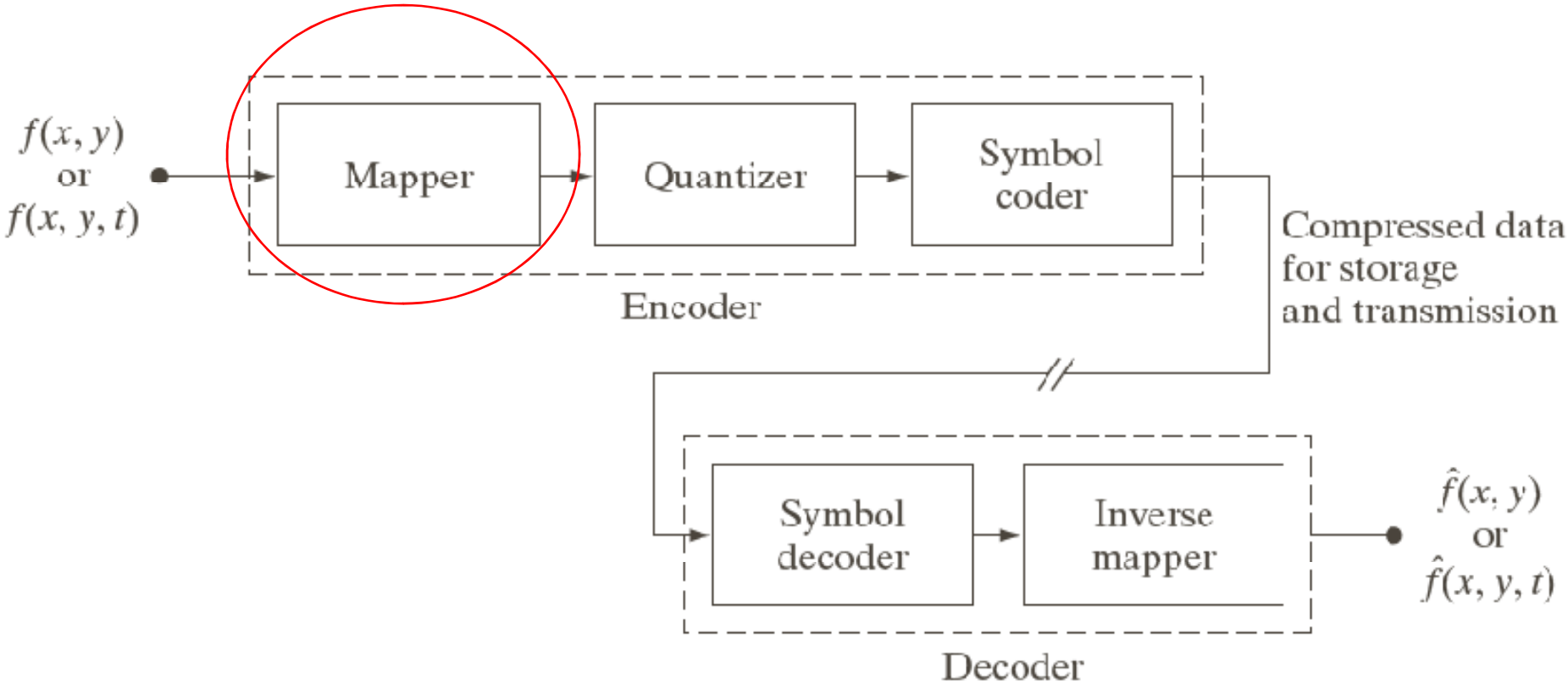


FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

Reducing Correlations between pixels



Reduce the Correlation between Pixels

- **Orthogonal Transform Coding**
  - KLT (Karhunen-Loeve Transform)
    - Maximal Decorrelation Process
  - DCT (Discrete Cosine Transform)
    - JPEG is a DCT-based image compression standard, which is a lossy coding method and may result in some loss of details and unrecoverable distortion.
- **Subband Coding**
  - DWT (Discrete Wavelet Transform)
    - To divide the spectrum of an image into the lowpass and the highpass components, DWT is a famous example.
    - JPEG 2000 is a 2-dimension DWT based image compression standard.
- **Predictive Coding**
  - DPCM
    - To remove mutual redundancy between seccessive pixels and encode only the new information

Covariance

- The covariance between two random variables X and Y, with expected value E[X]= μ_X and E[Y]= μ_Y is defined as

$$\begin{aligned}\operatorname{cov}(X,Y) &= E[(X-\mu_X)(Y-\mu_Y)] = E[XY]-\mu_X\mu_Y \\ &= \sum_X \sum_Y (X-\mu_X)(Y-\mu_Y) f(X,Y) \text{ (discrete)}\end{aligned}$$

- If entries in the column vector $X = [x_1, x_2, \ldots, x_N]^T$ are random variables, each with finite variance, then the covariance matrix C is the matrix whose (i, j) entry is the covariance

$$C = E[(X-\mu_x)(X-\mu_x)^T] = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nn} \end{bmatrix}$$

$$\mu_x = [\mu_1 \quad \mu_2 \quad \ldots \quad \mu_n]^T = [E(x_1) \quad E(x_2) \quad \ldots \quad E(x_n)]^T$$

The Orthogonal Transform

- **The Linear Transform**
  - The forward transform $y = Ax$
  - The inverse transform $x = Vy$
  - If we want to obtain the inverse transform, we need to compute the inverse of the transform matrix since

$$V = A^{-1}$$

$$VA = A^{-1}A = I$$

$$x = Vy = V(Ax) = (A^{-1}A)x = x$$

- **The Orthogonal Transform**
  - The forward transform $y = V^T x$
  - The inverse transform $x = Vy$
  - If we want to obtain the inverse transform, we need not to compute the inverse of the transform matrix since

$$VV^T = V^T V = I$$

$$x = Vy = V(V^T x) = (V^T V)x = x$$

Karhunen-Loeve Transform

- **KLT is** the optimum transform coder that is defined as the one that minimizes the mean square distortion of the reproduced data for a given number of total bits

The KLT

X: The input vector with size N-by-1

A: The transform matrix with size N-by-N

Y: The transformed vector with size N-by-1, and each components v(k) are mutually uncorrelated

C*xixj*: The covariance matrix of *xi* and *xj*

C*yiyj*: The covariance matrix of *yi* and *yj*

The transform matrix **A** is composed of the eigenvectors of the autocorrelation matrix **C***xixj*, which makes the output autocorrelation matrix **C***yiyj* be composed of the eigenvalues $\lambda_0, \lambda_1, \ldots \lambda_{N-1}$ in the diagonal direction. That is

$$
\begin{aligned}
C_{yy} &= E[(Y - E(Y))(Y - E(Y))^T] \\
&= E[YY^T] : zero-mean\ assumption \\
&= E[(Ax)(Ax)^T] = E[(Axx^T A^T)] \\
&= AE[xx^T]A^T = AC_{xx}A^T
\end{aligned}
$$

$$
C_{yy} = \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_{N-1} \end{bmatrix} = AC_{xx}A^T
$$

Discrete Cosine Transform (1/2)

- Why DCT is more appropriate for image compression than DFT?
  - The DCT can concentrate the energy of the transformed signal in low frequency, whereas the DFT can not
  - For image compression, the DCT can reduce the blocking effect than the DFT

Forward DCT

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

for $u = 0,...,N-1$ and $v = 0,...,N-1$

where $N = 8$ and $C(k) = \begin{cases} 1/\sqrt{2} & \text{for } k = 0 \\ 1 & \text{otherwise} \end{cases}$

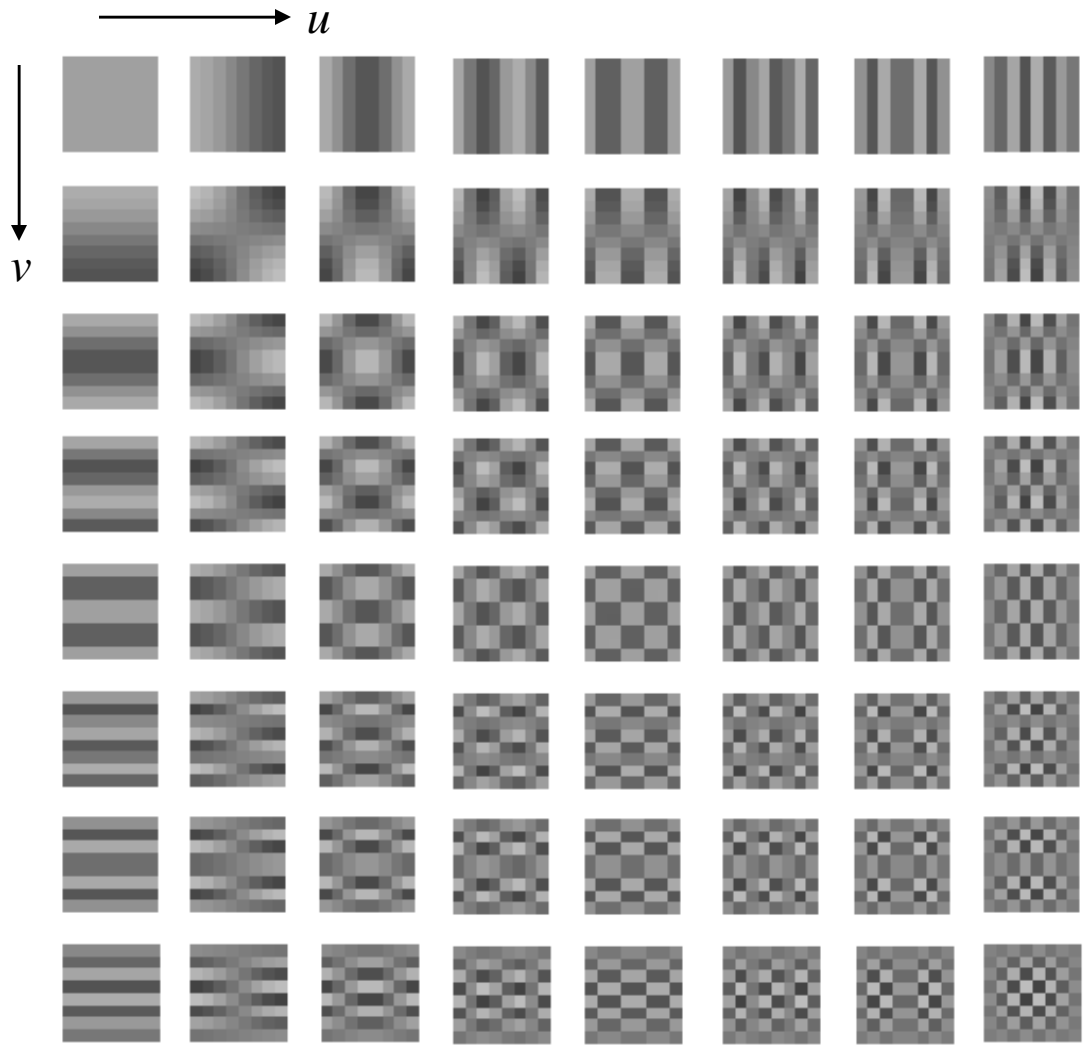
Inverse DCT

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

for $x = 0,...,N-1$ and $y = 0,...,N-1$ where $N = 8$

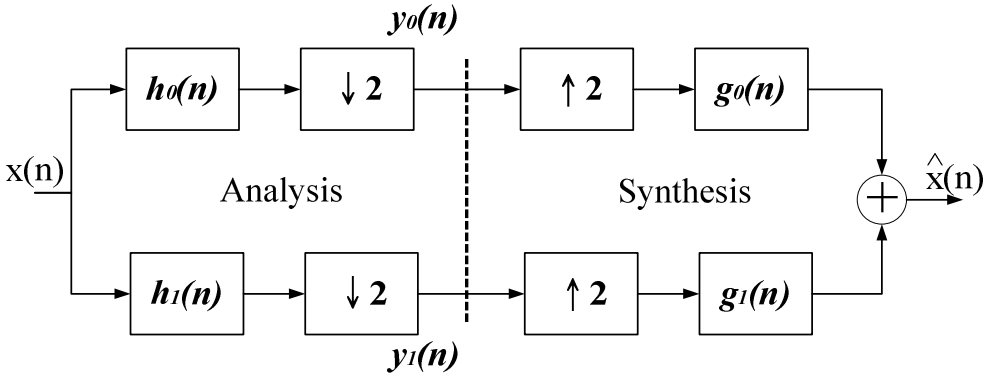
Discrete Cosine Transform (2/2)

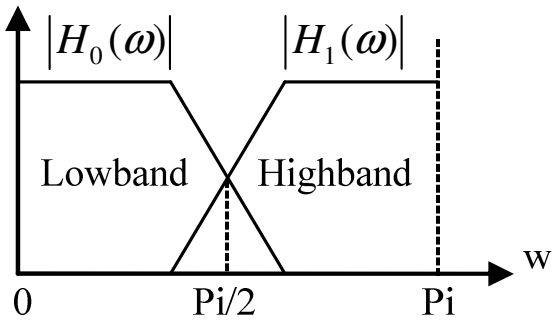
The 8-by-8 DCT basis



Discrete Wavelet Transform (1/2)

- Subband Coding
  - The spectrum of the input data is decomposed into a set of bandlimitted components, which is called subbands
  - Ideally, the subbands can be assembled back to reconstruct the original spectrum without any error
- The input signal will be filtered into lowpass and highpass components through analysis filters
- The human perception system has different sensitivity to different frequency band
  - The human eyes are less sensitive to high frequency-band color components
  - The human ears is less sensitive to the low-frequency band less than 0.01 Hz and high-frequency band larger than 20 KHz





Discrete Wavelet Transform (2/2)

- 1D DWT applied alternatively to vertical and horizontal direction line by line
- The LL band is recursively decomposed, first vertically, and then horizontally

1D scaling function $\phi(x), \phi(y)$

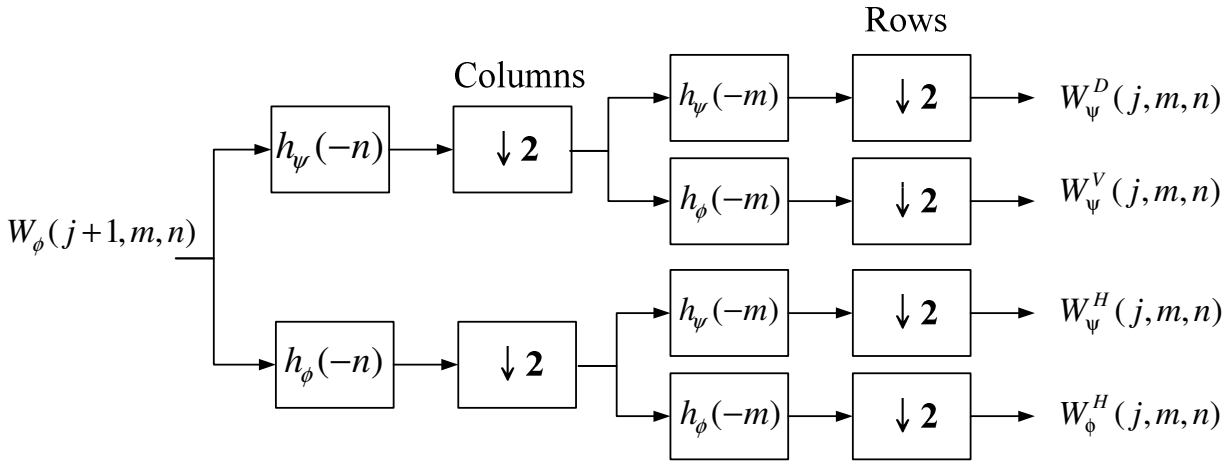
1D wavelet function $\psi(x), \psi(y)$

2D scaling function $\phi(x, y) = \phi(x)\phi(y)$

2D wavelet function $\psi^D(x, y) = \psi(x)\psi(y)$

$\psi^H(x, y) = \psi(x)\phi(y)$

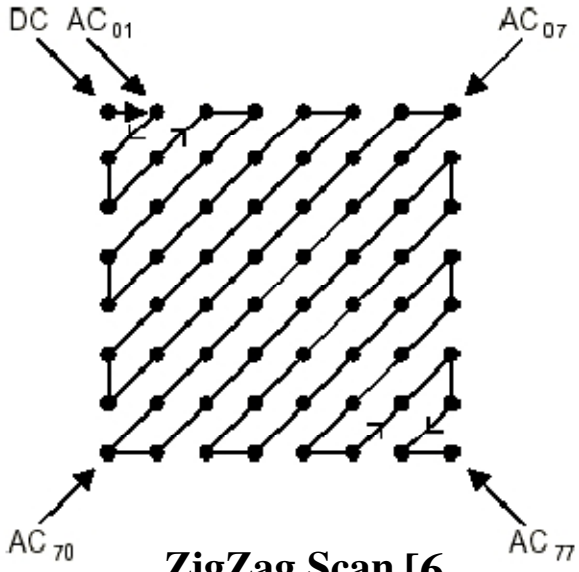
$\psi^V(x, y) = \phi(y)\psi(x)$



Differential Coding - JPEG (1/2)

- Transform Coefficients
  - DC coefficient
  - AC coefficients
- Because there is usually strong correlation between the DC coefficients of adjacent 8×8 blocks, the quantized DC coefficient is encoded as the difference from the DC term of the previous block
- The other 63 entries are the AC components. They are treated separately from the DC coefficients in the entropy coding process

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

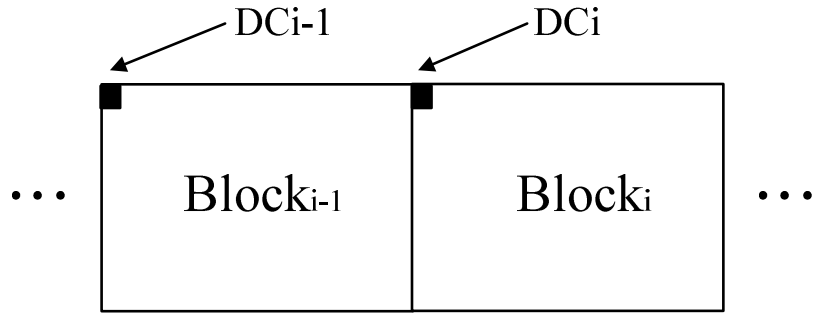


ZigZag Scan [6]

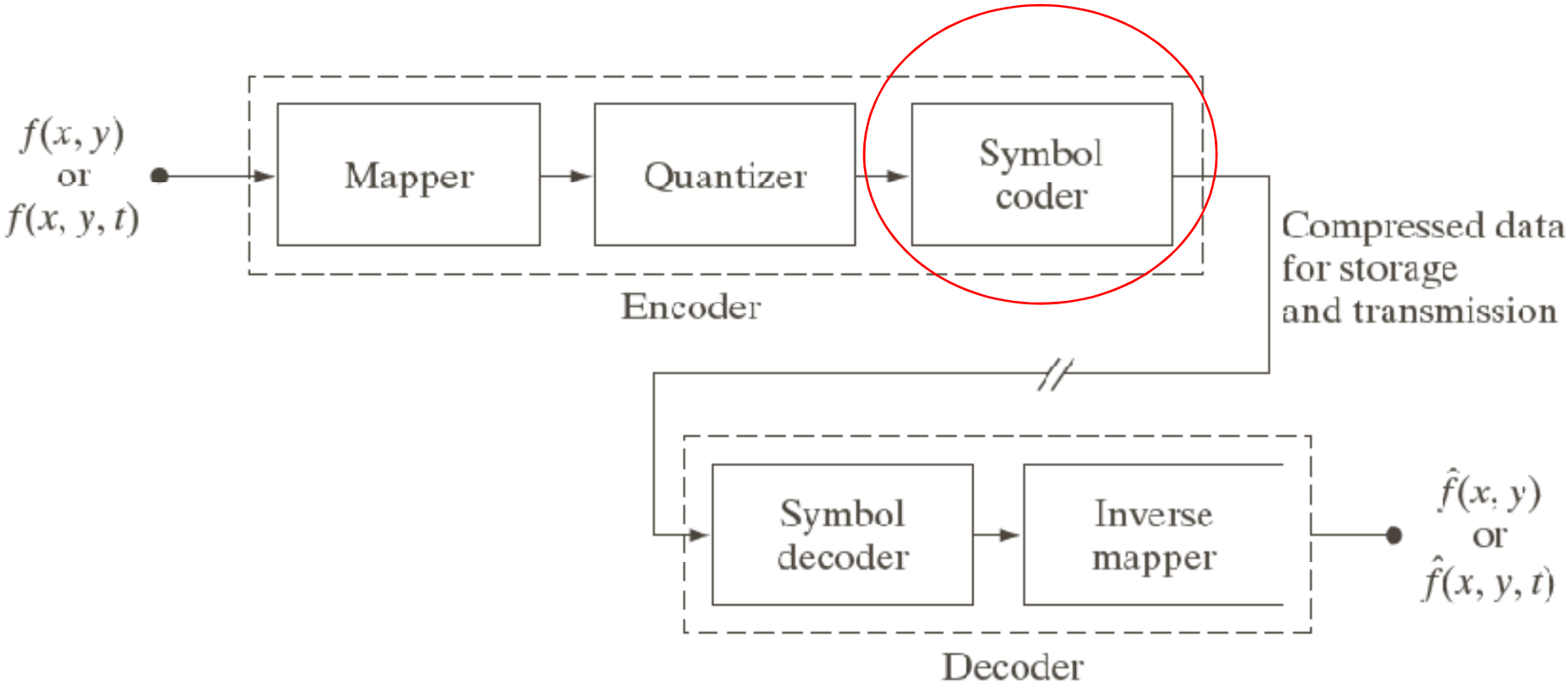
Differential Coding - JPEG (2/2)

- We set $DC_0 = 0$.
- DC of the current block DC_i will be equal to $DC_{i-1} + Diff_i$.
- Therefore, in the JPEG file, the first coefficient is actually the difference of DCs. Then the difference is encoded with Huffman coding algorithm together with the encoding of AC coefficients

Differential Coding : $Diff_i = DC_i - DC_{i-1}$



Source Coding or Symbol Coder



Symbol Coder (Source Coding)

- **Source Coding**
  - To achieve less average length of bits per pixel of the image.
  - Assigns short descriptions to the more frequent outcomes and long descriptions to the less frequent outcomes
  - Entropy Coding Methods
    - Huffman Coding
    - Arithmetic Coding
    - Run Length Coding
    - Dictionary Codes
      - Lempel-Ziv77
      - Lempel-Ziv 78

Huffman 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.

Huffman Coding

FIGURE 8.8 Huffman code assignment procedure.

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

The average length of this code is

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

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

Huffman Coding (1/2)

- The code construction process has a complexity of $O(N\log_2 N)$
- Huffman codes satisfy the prefix-condition
  - Uniquely decodable: no codeword is a prefix of another codeword

Huffman Coding Algorithm

(1) Order the symbols according to the probabilities

Alphabet set: $S_1, S_2, \ldots, S_N$

Probabilities: $P_1, P_2, \ldots, P_N$

The symbols are arranged so that $P_1 \geqq P_2 \geqq \ldots \geqq P_N$

(2) Apply a contraction process to the two symbols with the smallest probabilities. Replace the last two symbols S_N and S_{N-1} to form a new symbol H_{N-1} that has the probabilities $P_1 + P_2$.

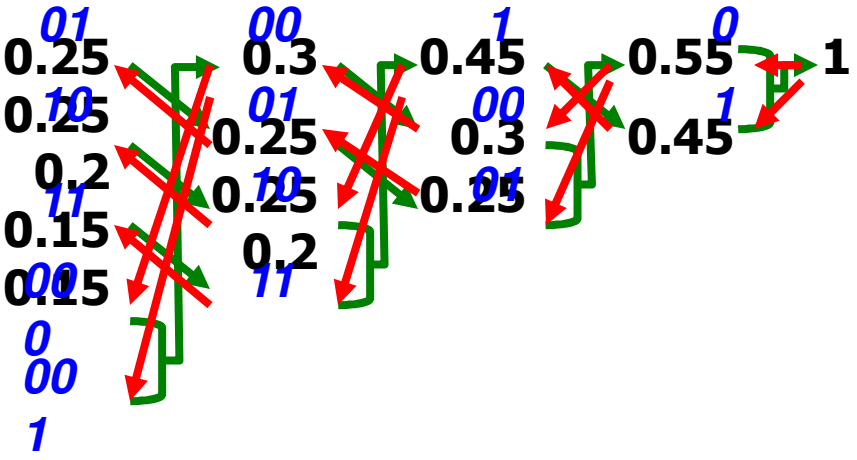
The new set of symbols has N-1 members: $S_1, S_2, \ldots, S_{N-2}, H_{N-1}$

(3) Repeat the step 2 until the final set has only one member.

(4) The codeword for each symbol S_i is obtained by traversing the binary tree from its root to the leaf node corresponding to S_i

Huffman Coding (2/2)

Codeword length	Codeword	X	Probability
2	01	1	0.25
2	10	2	0.25
3	11	3	0.2
3	000	4	0.15
3	001	5	0.15



Arithmetic Coding (1/4)

Shannon-Fano-Elias Coding

- We take **X**={1,2,…,m}, p(x)>0 for all x.
- Modified cumulative distribution function $\overline{F} = \sum_{a<x} P(a) + \frac{1}{2} P(x)$
- Assume we round off $\overline{F(x)}$ to $l(x)$, which is denoted by $\lfloor \overline{F(x)} \rfloor_{l(x)}$
- The codeword of symbol x has $l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$ bits
- Codeword is the binary value of $\overline{F(x)}$ with $l(x)$ bits

x	*P(x)*	*F(x)*	$\overline{F(x)}$	$\overline{F(x)}$ **in binary**	*l(x)*	**codeword**
1	**0.25**	**0.25**	**0.125**	**0.001**	**3**	**001**
2	**0.25**	**0.50**	**0.375**	**0.011**	**3**	**011**
3	**0.20**	**0.70**	**0.600**	**0.$\overline{1001}$**	**4**	**1001**
4	**0.15**	**0.85**	**0.775**	**0.1100$\overline{011}$**	**4**	**1100**
5	**0.15**	**1.00**	**0.925**	**0.1110$\overline{110}$**	**4**	**1110**

Arithmetic Coding (2/4)

- **Arithmetic Coding**: a direct extension of Shannon-Fano-Elias coding calculate the probability mass function $p(x^n)$ and the cumulative distribution function $F(x^n)$ for the souece sequence x^n
  - Lossless compression technique
  - Treate multiple symbols as a single data unit

Arithmetic Coding Algorithm

Input symbol is l

$Previous_{low}$ is the lower bound for the old interval

$Previous_{high}$ is the upper bound for the old interval

Range is $Previous_{high}$ - $Previous_{low}$

Let $Previous_{low}$= 0, $Previous_{high}$ = 1, Range = $Previous_{high}$ – $Previous_{low}$ =1

WHILE (input symbol != EOF)

get input symbol l

Range = $Previous_{high}$ - $Previous_{low}$

New $Previous_{low}$ = $Previous_{low}$ + Range* $interval_{low}$ of l

New $Previous_{high}$ = $Previous_{low}$ + Range* $interval_{high}$ of l

END

Run-Length Coding

1. Run-length Encoding, or **RLE** is a technique used to **reduce the size of a repeating string of characters**
2. This repeating string is called a *run*, typically RLE encodes a run of symbols into two bytes , a **count** and a **symbol**.
3. RLE can compress any type of data
4. RLE cannot achieve high compression ratios compared to other compression methods
5. It is easy to implement and is quick to execute

Run-Length Coding

- **Example**

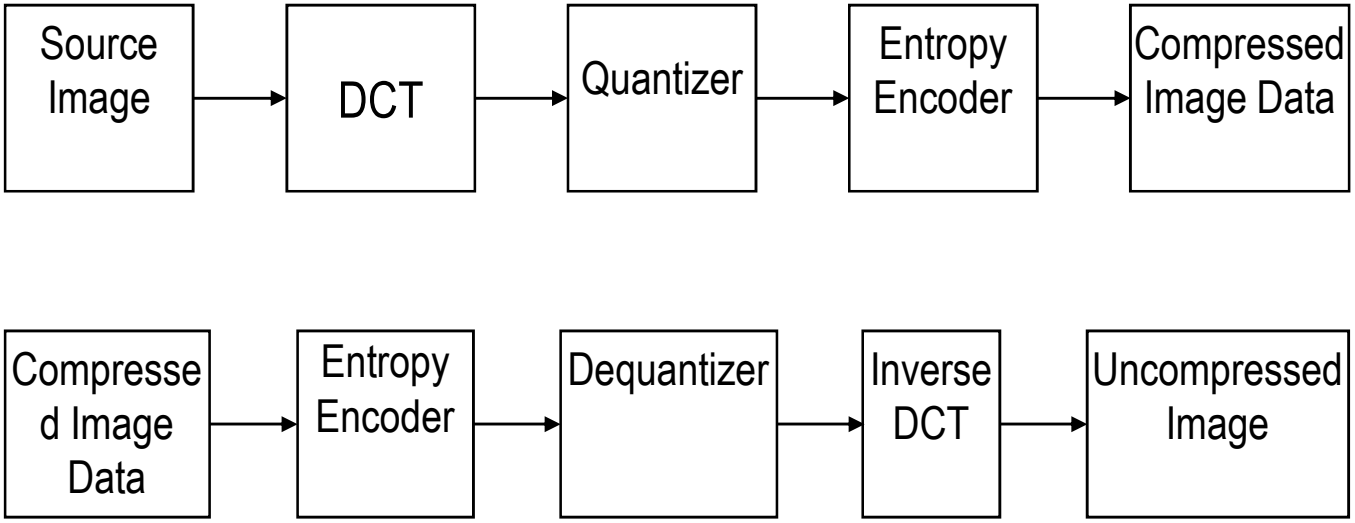
  WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWW
  WWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWW
  WWWW

  RLE coding:

  12W1B12W3B24W1B14W

JPEG Compression (Transform)

- JPEG stands for Joint Photographic Experts Group



JPEG coder and decoder

JPEG Compression

- **JPEG compression consists basic steps:**
  1. **Input the source dark - gray image I.**
  2. **Partition image into 8 x 8 pixel blocks and perform the DCT on each block.**
  3. **Quantize resulting DCT coefficients.**
  4. **Entropy code the reduced coefficients.**

JPEG Compression

- **The second step consists of separating image components are:**
  - **Broken into arrays or "tiles" of 8 x 8 pixels.**
  - **The elements within the tiles are converted to signed integers (for pixels in the range of 0 to 255, subtract 128).**
  - **These tiles are then transformed into the spatial frequency domain via the forward DCT.**
  - **Element (0,0) of the 8 x 8 block is referred to as DC, DC is the average value of the 8 x 8 original pixel values.**
  - **The 63 other elements are referred to as AC_{YX}, where x and y are the position of the element in the array.**

JPEG Compression

- **For example**

50	55	61	60	70	61	60	71
63	59	55	90	89	85	69	72
62	59	68	113	114	64	66	73
63	58	71	102	54	106	70	69
61	61	68	100	76	88	68	70
79	65	60	70	77	68	58	75
82	71	64	59	55	61	65	80
81	79	69	68	65	76	78	90

F - A 8x8 block extracted

-128

-78	-73	-67	-68	-58	-67	-68	-57
-65	-69	-73	-38	-39	-43	-59	-56
-66	-69	-60	-15	-14	-64	-62	-55
-65	-70	-57	-26	-74	-22	-58	-59
-67	-67	-60	-28	-52	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-46	-57	-64	-69	-73	-67	-63	-48
-47	-49	-59	-60	-63	-52	-50	-38

F - Signed block

JPEG Compression

- **DCT of image block F as matrix G:**

$$G(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} F(x,y)\cos\left((x+1/2)\pi u/8\right)\cos\left((y+1/2)\pi v/8\right)$$

where u = 0, 1, ..,7; v = 0, 1, ..,7

and

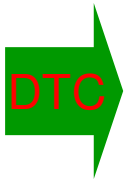
$$\alpha(0)=\sqrt{1/8},\ \alpha(\boldsymbol{u})=\sqrt{2/8},\ \boldsymbol{u}=1,2,\ldots,7$$

JPEG Compression

- **For example**

-78	-73	-67	-68	-58	-67	-68	-57
-65	-69	-73	-38	-39	-43	-59	-56
-66	-69	-60	-15	-14	-64	-62	-55
-65	-70	-57	-26	-74	-22	-58	-59
-67	-67	-60	-28	-52	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-46	-57	-64	-69	-73	-67	-63	-48
-47	-49	-59	-60	-63	-52	-50	-38

F - Signed block



-452.87	-21.19	-25.44	2.29	29.62	4.79	5.53	-23.80
-7.23	-15.53	-44.20	0.54	10.69	0.55	-6.63	0.76
-18.38	0.78	36.34	-2.75	-4.56	-17.35	-6.34	21.68
-30.61	4.11	15.13	2.30	-5.01	-8.56	5.49	7.27
-1.38	-8.87	2.77	-5.20	-24.37	9.18	12.25	-15.55
-16.93	6.49	13.74	-8.25	-8.03	3.95	7.50	0.72
2.34	4.00	-8.34	-5.50	16.10	-9.96	-9.09	17.00
1.19	2.57	-11.31	-10.66	12.22	-3.73	-12.53	13.28

G = DTC of F

JPEG Compression

- **Quantization**
  - The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This allows one to greatly reduce the amount of information in the high frequency components.
  - This is done by simply dividing each component in the frequency domain by a constant for that component, and then rounding to the nearest integer.
  - This is the main lossy operation in the whole process. As a result of this, it is typically the case that many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers, which take many fewer bits to store.

JPEG Compression

- **Quantization**
  - A typical quantization matrix, as specified in the original JPEG Standard Q is

$$Q =$$

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

$$B(x,y) = \text{round}\left(\frac{G(x,y)}{Q(x,y)}\right),$$

$$\text{for } x, y = 0,1,\ldots,7$$

JPEG Compression

- **For example**

-452.87	-21.19	-25.44	2.29	29.62	4.79	5.53	-23.80
-7.23	-15.53	-44.20	0.54	10.69	0.55	-6.63	0.76
-18.38	0.78	36.34	-2.75	-4.56	-17.35	-6.34	21.68
-30.61	4.11	15.13	2.30	-5.01	-8.56	5.49	7.27
-1.38	-8.87	2.77	-5.20	-24.37	9.18	12.25	-15.55
-16.93	6.49	13.74	-8.25	-8.03	3.95	7.50	0.72
2.34	4.00	-8.34	-5.50	16.10	-9.96	-9.09	17.00
1.19	2.57	-11.31	-10.66	12.22	-3.73	-12.53	13.28

G = DTC of F

Quantization

-28	-2	-3	0	1	0	0	0
-1	-1	-3	0	0	0	0	0
-1	0	2	0	0	0	0	0
-2	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

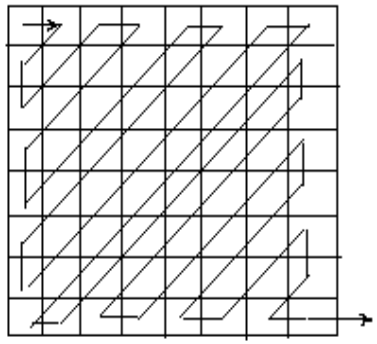
B - Quantization of G

JPEG Compression

- **For example**

-28	-2	-3	0	1	0	0	0
-1	-1	-3	0	0	0	0	0
-1	0	2	0	0	0	0	0
-2	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

B - Quantization of G



- The coefficients in B are reordered in accordance with the zigzag ordering: B(0,0), B(0,1), B(1,0), B(2,0), B(1,1), B(1,2),...

- {-28, -2, -1, -1,-1,-3, 0, -3, 0, -2, 0, 0, 2, 0, 1, 0, 0, 0, 1, 0, -1, EOB}

  Where the EOB symbol denotes the end-of-block condition.

JPEG Compression

- **The Zero Run Length Coding (ZRLC)**
  - Let's consider the 63 vector (it's the 64 vector without the first coefficient). Say that we have -2, -1, -1,-1,-3, 0, -3, 0, -2, 0, 0, 2, 0, 1, 0, 0, 0, 1, 0, -1, 0 , 0 ,0 , only 0,..,0. Here it is how the RLC JPEG compression is done for this example :

    (0,-2), (0,-1), (0,-1), (0,-1), (0,-3), (1,-3), (1,-2), (2, 2), (1,1), (3,1), (1,-1), EOB

    ACTUALLY, EOB has as an equivalent (0,0) and it will be (later) Huffman coded like (0,0). So we'll encode :

    (0,-2), (0,-1), (0,-1), (0,-1), (0,-3), (1,-3), (1,-2), (2, 2), (1,1), (3,1), (1,-1), (0,0)

JPEG Compression

- **Huffman coding**
  - JPEG standard stores the minimum size in bits in which keep that value (it's called the category of that value) and then a bit-coded representation of that value like this:

Values	Category	Bits for the value
0	0	-
-1, 1	1	0, 1
-3,-2, 2,3	2	00,01, 10,11
-7,-6,-5,-4, 4,5,6,7	3	000,001,010,011, 100,101,110,111
-15,..,-8, 8,..,15	4	0000,....,0111, 1000,....,1111
-31,..,-16, 16,..,31	5	00000,....,01111, 10000,....,11111
-63,..,-32, 32,..,63	6	
-127,..,-64, 64,..,12	7	
-255,..,-128, 128,..,255	8	
-511,..,-256, 256,..,511	9	
-1023,..,-512, 512,..,1023	10	
-2047,..,-1024, 1024,..,2047	11	
-4095,..,-2048, 2048,..,4095	12	
-8191,..,-4096, 4096,..,8191	13	
-16383,..,-8192, 8192,..,16383	14	
-32767,..,-16384, 16384,..,32767	15	

JPEG Compression

- **Huffman coding**
  - In consequence for the previous example:
    (0,-2), (0,-1), (0,-1), (0,-1), (0,-3), (1,-3), (1,-2), (2, 2), (1,1), (3,1), (1,-1), **(0,0)**
  - let's encode ONLY the right value of these pairs, except the pairs that are special markers like (0,0)

Value	Category	bit-coded	codes as
-2	2	01	2, 01
-1	1	0	1, 0
-3	2	00	2, 00
2	2	10	2, 10
1	1	1	1, 1

  - (0,2)01, (0,1)0, (0,1)0, (0,1)0, (0,2)00, (1,2)00, (1,2)01, (2,2)10, (1,1)1, (3,1)1, (1,1)0, (0,0)

JPEG Compression

- **Huffman coding**
  - (0,2) 01, (0,1)0, (0,1)0, (0,1)0, (0,2)00, (1,2)00, (1,2)01, (2,2)10, (1,1)1, (3,1)1, (1,1)0, (0,0)
  - The pairs of 2 values enclosed in bracket parenthesis, can be represented on a byte. In this byte, the high nibble represents the number of previous 0s, and the lower nibble is the category of the new value different by 0
  - The FINAL step of the encoding consists in Huffman encoding this byte, and then writing in the JPG file, as a stream of bits, the Huffman code of this byte, followed by the remaining bit-representation of that number. The final stream of bits written in the JPG file on disk for the previous example

    (01)01 (00)0 (00)0 (00)0 (01)00 (111001)00 (111001)01 (11111000)10 (1100)1 (111010)1 (1100)0 (1010)

Pairs	AC Huffman code
0, 2	01
0, 1	00
1, 2	111001
2, 2	11111000
1, 1	1100
3, 1	111010
0, 0	1010

JPEG Compression

- **Huffman coding**
  - The encoding of the DC coefficient
    - DC is the coefficient in the quantized vector corresponding to the lowest frequency in the image (it's the 0 frequency) , and (before quantization) is mathematically = (the sum of 8x8 image samples) / 8 .
    - The authors of the JPEG standard noticed that there's a very close connection between the DC coefficient of consecutive blocks, so they've decided to encode in the JPG file the difference between the DCs of consecutive 8x8 blocks:

      $$Diff = DC(i) - DC(i-1)$$

      And in JPG decoding you will start from 0 -- you consider that the first DC(0)=0

JPEG Compression

- **Huffman coding**
  - The encoding of the DC coefficient
    - Diff = (category, bit-coded representation). For example, if Diff is equal to -511 , then Diff corresponds to (9, 000000000). Say that 9 has a Huffman code = 1111110. (In the JPG file, there are 2 Huffman tables for an image component: one for DC (and one for AC). In the JPG file, the bits corresponding to the DC coefficient will be: 1111110 000000000
    - And, applied to this example of DC and to the previous example of ACs, for this vector with 64 coefficients, THE FINAL STREAM OF BITS written in the JPG file will be:

      1111110 000000000 (01)01 (00)0 (00)0 (00)0 (01)00 (111001)00 (111001)01 (11111000)10 (1100)1 (111010)1 (1100)0 (1010)
    - Note: (In the JPG file , first it's encoded DC then ACs)

JPEG Compression

- **JPEG Decoder process**
  - Decoding of Huffman, Decoding of ZRLC of bit stream from JPEG file (for a block 8x8) -> {-28, -2, -1, -1,-1,-3, 0, -3, 0, -2, 0, 0, 2, 0, 1, 0, 0, 0, 1, 0, -1, **EOB**}
  - Dequantize the 64 vector : "for (i=0;i<=63;i++) vector[i]*=quant[i]“
  - Re-order from zig-zag the 64 vector into an 8x8 block
  - Apply the Inverse DCT transform to the 8x8 block
    => F’

JPEG Compression

- **Example**
  - Dequantize and Re-order from zig-zag

-448	-22	-30	0	24	0	0	0
-12	-12	-42	0	0	0	0	0
-14	0	32	0	0	0	0	0
-28	0	22	0	0	0	0	0
0	0	0	0	0	0	0	0
-24	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

JPEG Compression

- **Example**
  - Inverse DTC

-74	-77	-72	-62	-59	-65	-66	-61
-68	-67	-56	-41	-38	-48	-56	-55
-73	-67	-51	-32	-30	-45	-58	-62
-73	-68	-54	-36	-35	-49	-61	-64
-60	-61	-54	-43	-42	-51	-56	-53
-54	-62	-63	-58	-57	-61	-58	-50
-52	-62	-67	-65	-64	-65	-59	-49
-40	-51	-56	-54	-54	-55	-49	-39

JPEG Compression

- **Example**
  - Unsigned

54	51	56	66	69	63	62	67
60	61	72	87	90	80	72	73
55	61	77	96	98	83	70	66
55	60	74	92	93	79	67	64
68	67	74	85	86	77	72	75
74	66	65	70	71	67	70	78
76	66	61	63	64	63	69	79
88	77	72	74	74	73	79	89

Uncompressed Image Block

50	55	61	60	70	61	60	71
63	59	55	90	89	85	69	72
62	59	68	113	114	64	66	73
63	58	71	102	54	106	70	69
61	61	68	100	76	88	68	70
79	65	60	70	77	68	58	75
82	71	64	59	55	61	65	80
81	79	69	68	65	76	78	90

Origin Image Block