Image Compression

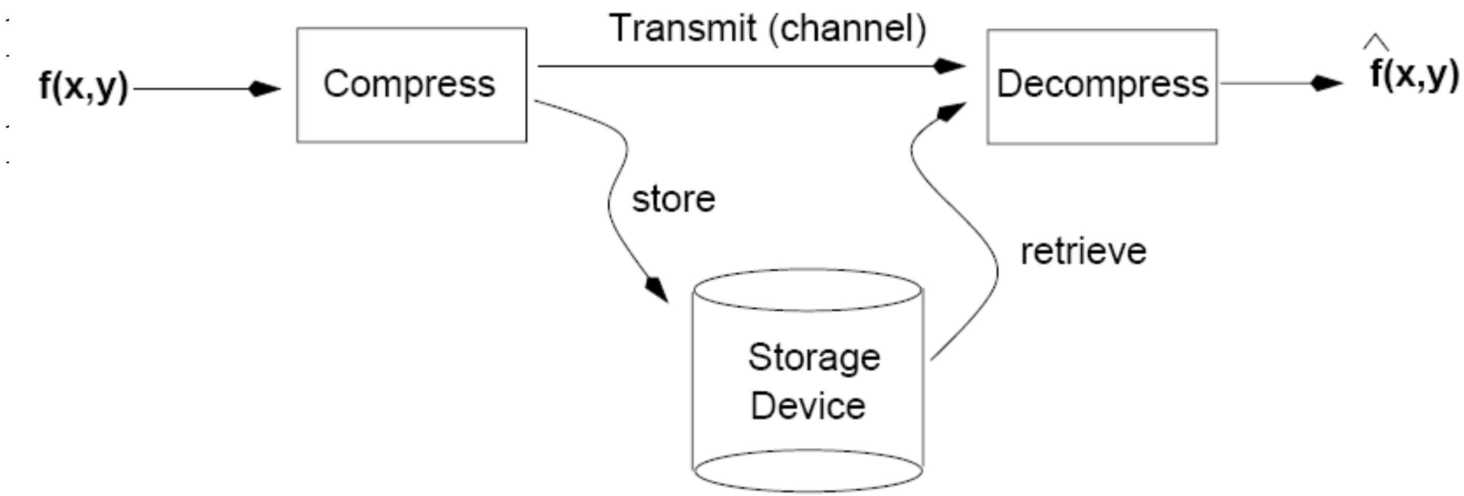
Instructor
LE Thanh Sach, Ph.D.
Ltsach@cse.hcmut.edu.vn
http://www.cse.hcmut.edu.vn/~ltsach/

Outline

- Introduction
- Lossless Compression
- Lossy Compression

Introduction

- The goal of image compression is to reduce the amount of data required to represent a digital image.
- Important for reducing **storage**



n

Approaches

- Lossless
  - **Information preserving**
  - **Low compression ratios**
  - **e.g., Huffman**

- Lossy
  - **Does not preserve information**
  - **High compression ratios**
  - **e.g., JPEG**

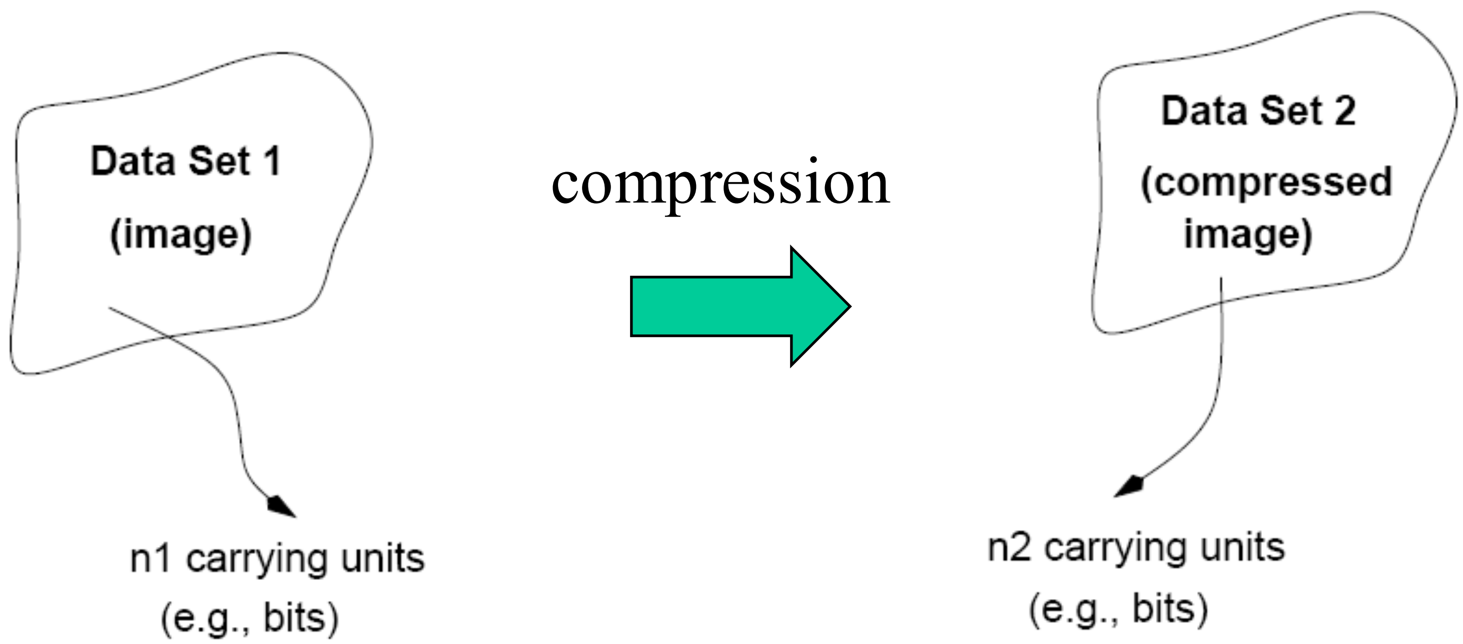
- Tradeoff: image quality **vs** compression ratio

Data vs Information

- Data and information are not synonymous terms!
- **Data** is the means by which **information** is conveyed.
- Data compression aims to reduce the amount of data required to represent a given quantity of information while preserving as much information as possible.

Data Redundancy

❖ Data redundancy is a mathematically quantifiable entity!



Data Redundancy (cont'd)

- Compression ratio: $C_R = \frac{n_1}{n_2}$

- Relative data redundancy: $R_D = 1 - \frac{1}{C_R}$

Example:

If $C_R = \frac{10}{1}$, then $R_D = 1 - \frac{1}{10} = 0.9$

(90% of the data in dataset 1 is redundant)

if $n_2 = n_1$, then C_R=1, R_D=0

if $n_2 \ll n_1$, then $C_R \to \infty$, $R_D \to 1$

if $n_2 \gg n_1$, then $C_R \to 0$, $R_D \to -\infty$

Types of Data Redundancy

(1) Coding

(2) Interpixel

(3) Psychovisual

- The role of compression is to reduce one or more of these redundancy types.

Coding Redundancy

- Data compression can be achieved using an appropriate <u>encoding scheme</u>.

<u>Example:</u> binary encoding

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

Encoding Schemes

❖Elements of an encoding scheme:

- **<u>Code:</u> a list of symbols (letters, numbers, bits etc.)**
- **<u>Code word:</u> a sequence of symbols used to represent a piece of information or an event (e.g., gray levels)**
- **<u>Code word length:</u> number of symbols in each code word**

<u>Example:</u> (binary code, symbols: 0,1, length: 3)

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

Definitions

N x M image

r_k: k-th gray level

$P(r_k)$: probability of r_k

Expected value:

$$E(X) = \sum_x xP(X = x)$$

$l(r_k)$: # of bits for r_k

Average # of bits: $L_{avg} = E(l(r_k)) = \sum_{k=0}^{L-1} l(r_k)P(r_k)$

Total # of bits: NML_{avg}

Constant Length Coding

❖ $l(r_k) = c$ which implies that $L_{avg}=c$

Example:

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$
$r_0 = 0$	0.19	000	3
$r_1 = 1/7$	0.25	001	3
$r_2 = 2/7$	0.21	010	3
$r_3 = 3/7$	0.16	011	3
$r_4 = 4/7$	0.08	100	3
$r_5 = 5/7$	0.06	101	3
$r_6 = 6/7$	0.03	110	3
$r_7 = 1$	0.02	111	3

Assume an image with $L = 8$

Assume $l(r_k) = 3,\ L_{avg} = \sum_{k=0}^{7} 3P(r_k) = 3\sum_{k=0}^{7} P(r_k) = 3$ bits

Total number of bits: $3NM$

Avoiding Coding Redundancy

❖ To avoid coding redundancy, codes should be selected according to the probabilities of the events.

- **Variable Length Coding**

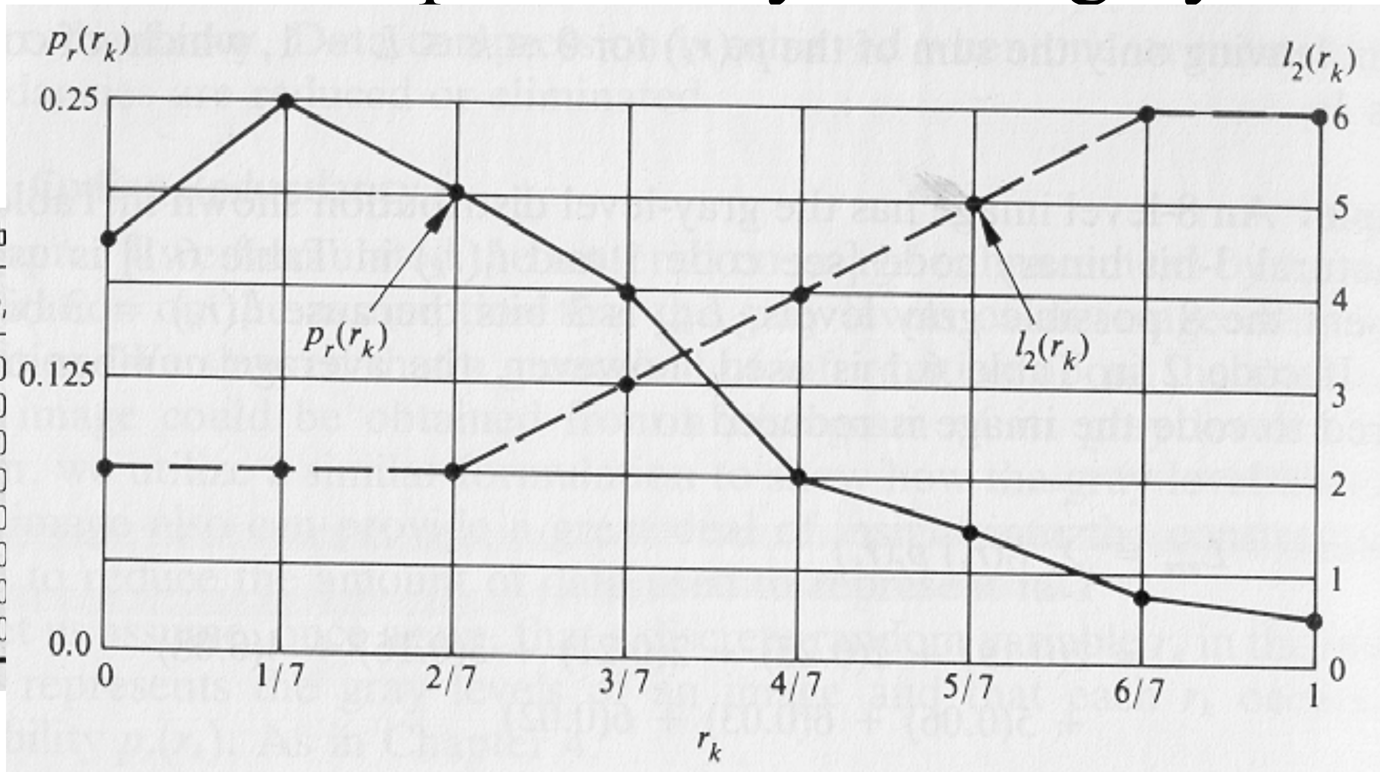
✍ **Assign fewer symbols (bits) to the more probable events (e.g., gray levels for images)**

Variable Length Coding

❖Consider the probability of the gray levels:

Table 6.1 **Variab**

r_k	p_r
$r_0 = 0$	0
$r_1 = 1/7$	0
$r_2 = 2/7$	0
$r_3 = 3/7$	0
$r_4 = 4/7$	0
$r_5 = 5/7$	0
$r_6 = 6/7$	0
$r_7 = 1$	0



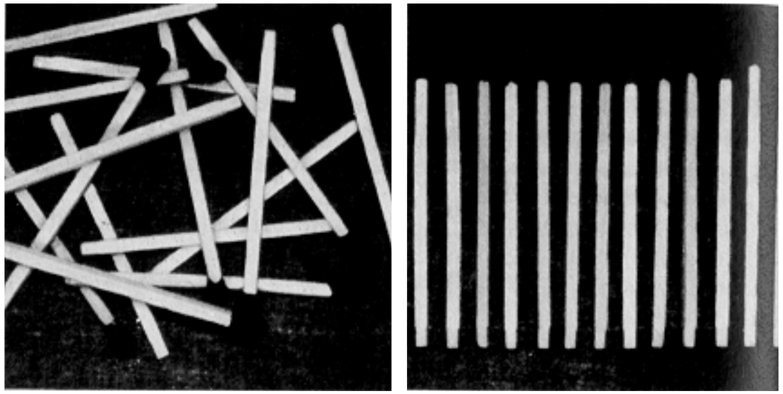
) = 2.7 bits

about 10%)

099

Interpixel redundancy

❖ Interpixel redundancy implies that any pixel value can be reasonably predicted by its neighbors (i.e., correlated).



$$\text{correlation: } f(x)\ o\ g(x) = \int_{-\infty}^{\infty} f^{*}(a)g(x+a)da$$

$$\text{autocorrelation: } g(x) = f(x)$$

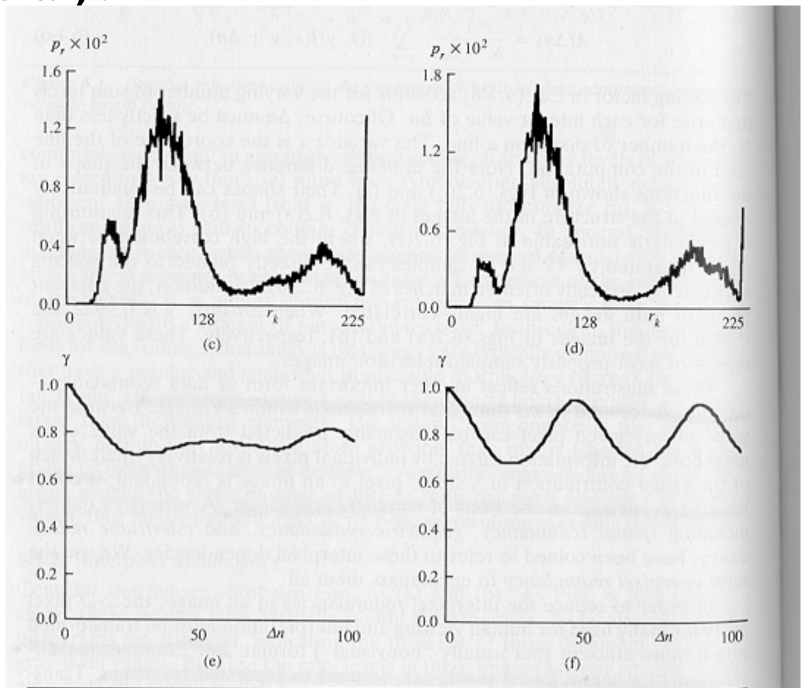
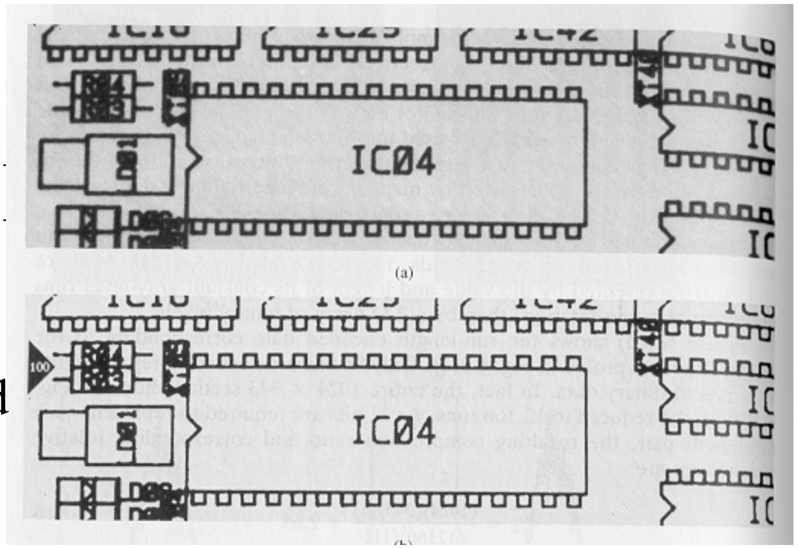


Figure 6.2 Two images and their gray-level histograms and normalized autocorrelation

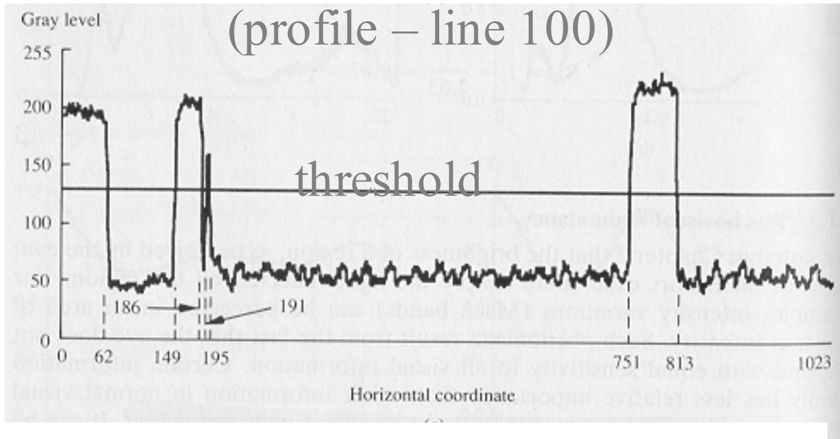
Interpixel redundancy (cont'd)

- To reduce interpixel redundnacy, the data must be transformed in another format (i.e., through a transformation)
  - **e.g., thresholding, or differences between adjacent pixels, or DFT**

original



thresholded

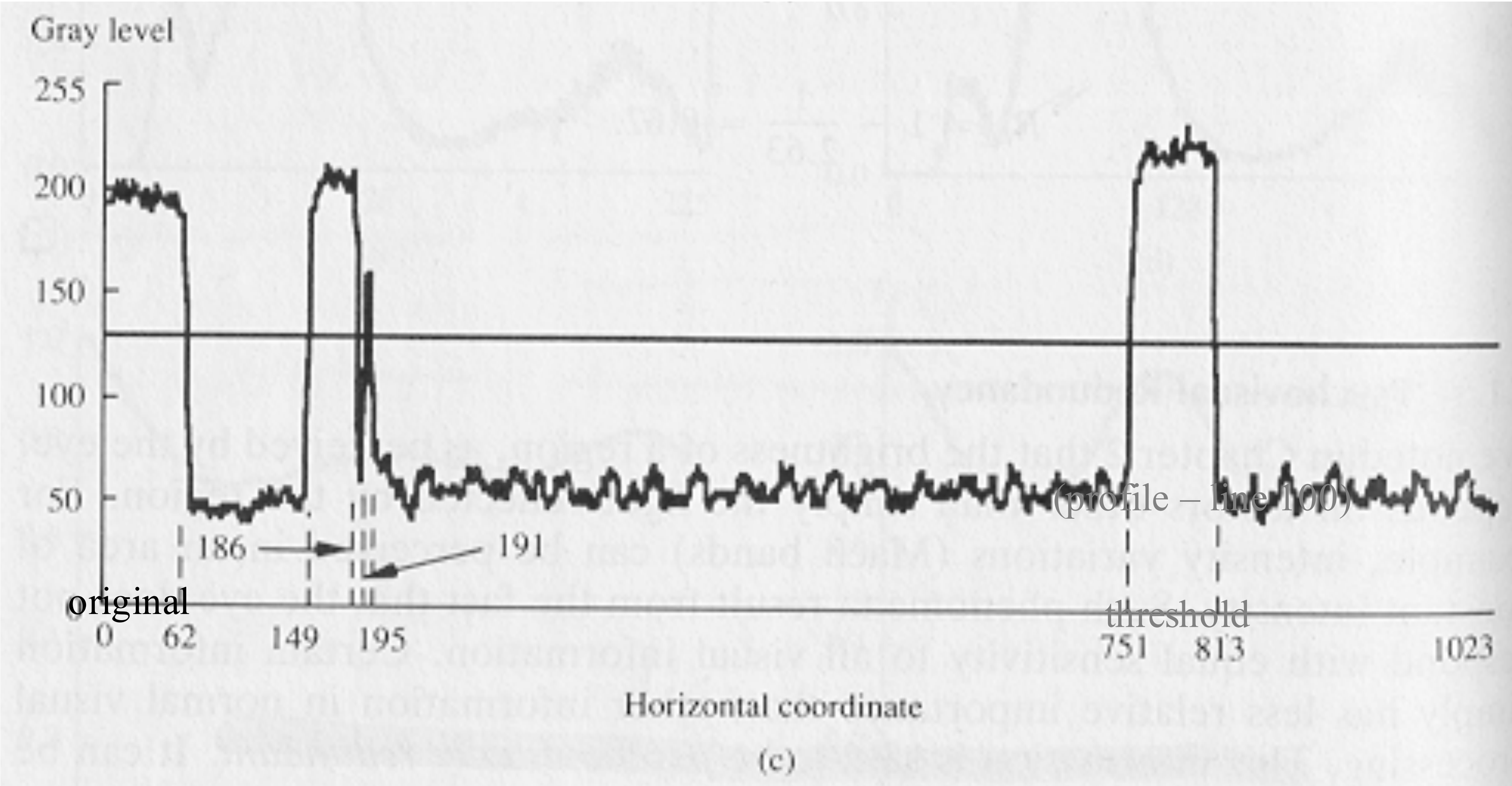


Run-length encoding:

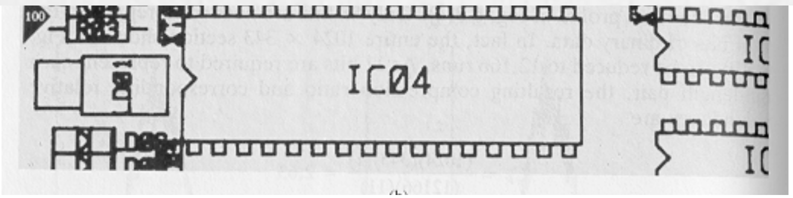
(1,63) (0,87) (1,37) (0,5) (1,4) (0, 556) (1,62) (0,210)

Using 11 bits/pair: (1+10)

88 bits are required (compared to 1024 !!)



thresholded



Run-length encoding:

(1,63) (0,87) (1,37) (0,5) (1,4) (0, 556) (1,62) (0,210)

Using 11 bits/pair: (1+10)

88 bits are required (compared to 1024 !!)

Psychovisual redundancy

- Takes into advantage the peculiarities of the human visual system.
- The eye does not respond with equal sensitivity to all visual information.
- Humans search for important features (e.g., edges, texture, etc.) and do not perform quantitative analysis of every pixel in the image.

Psychovisual redundancy (cont'd) Example: Quantization

256 gray levels

8/4
=
2:1

16 gray levels

16 gray levels
improved gray-scale quantization

i.e., add to each pixel a pseudo-random number prior to quantization

How do we measure information?

- What is the information content of a message/image?

- What is the minimum amount of data that is sufficient to describe completely an image without loss of information?

Modeling the Information Generation Process

- Assume that information generation process is a <u>probabilistic process</u>.

- A random event E which occurs with probability $P(E)$ contains:

$$I(E) = log(\frac{1}{P(E)}) = -log(P(E)) \text{ units of information}$$

(note that when $P(E) = 1$, then $I(E) = 0$: no information !)

How much information does a pixel contain?

- Suppose that the gray level value of pixels is generated by a random variable, then r_k contains

$$I(r_k) = -log(P(r_k)) \quad \text{units of information}$$

Average information of an image

❖ **<u>Entropy</u>**: the average information content of an image

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

using $$I(r_k) = -\,log(P(r_k))$$

we have: $$H = -\sum_{k=0}^{L-1} P(r_k) log(P(r_k))$$ units/pixel

<u>Assumption:</u> statistically independent random events

Modeling the Information Generation Process (cont'd)

❖ **Redundancy** $R = L_{avg} - H$

where: $L_{avg} = E(l(r_k)) = \sum_{k=0}^{L-1} l(r_k)P(r_k)$

note that if $L_{avg} = H$, then $R = 0$ - no redundancy

Entropy Estimation

❖Not easy!

image

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

Gray Level	*Count*	*Probability*
21	12	3/8
95	4	1/8
169	4	1/8
243	12	3/8

Entropy Estimation

- First order estimate of H:

$$H = -\sum_{k=0}^{3} P(r_k) log(P(r_k)) = 1.81 \text{ bits/pixel}$$

Total bits: 4 x 8 x 1.81 = 58 bits

Estimating Entropy (cont'd)

- Second order estimate of H:
  - **Use relative frequencies of pixel blocks :**

image

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

Gray Level Pair	*Count*	*Probability*
(21, 21)	8	1/4
(21, 95)	4	1/8
(95, 169)	4	1/8
(169, 243)	4	1/8
(243, 243)	8	1/4
(243, 21)	4	1/8

$$H = 2.5/2 = 1.25 \text{ bits/pixel}$$

Estimating Entropy (cont'd)

- Comments on first and second order entropy estimates:

  - **The first-order estimate gives only a <u>lower-bound</u> on the compression that can be achieved.**

  - **Differences between higher-order estimates of entropy and the first-order estimate indicate the presence of <u>interpixel redundancies</u>.**

Question

- How do we deal with interpixel redundancy?

Apply a transformation!

Estimating Entropy (cont'd)

- E.g., consider <u>difference</u> image:

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

→

21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0

Gray Level or Difference	*Count*	*Probability*
0	12	1/2
21	4	1/8
74	12	3/8

Estimating Entropy (cont'd)

❖Entropy of difference image:

$$H = -\sum_{k=0}^{2} P(r_k) log(P(r_k)) = 1.41 \text{ bits/pixel}$$

- Better than before (i.e., H=1.81 for original image), however, a better transformation could be found:

1.41 bits/pixel > 1.25 bits/pixel (from 2nd order estimate of H)

Image Compression Model

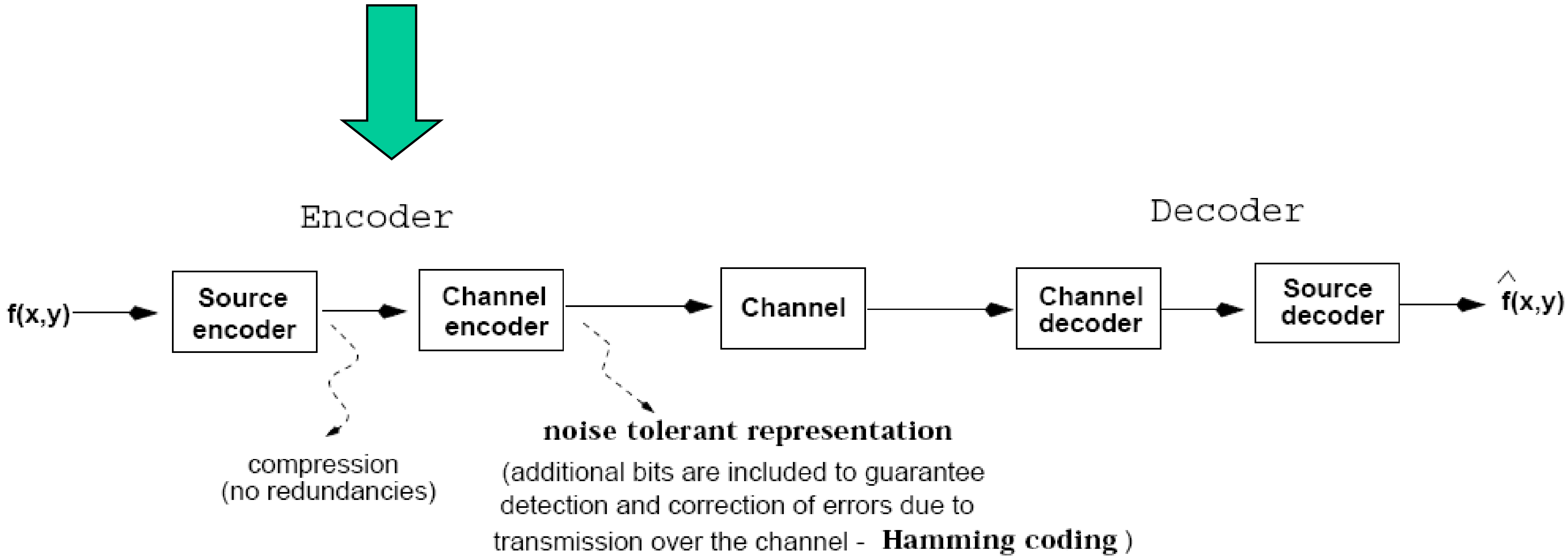
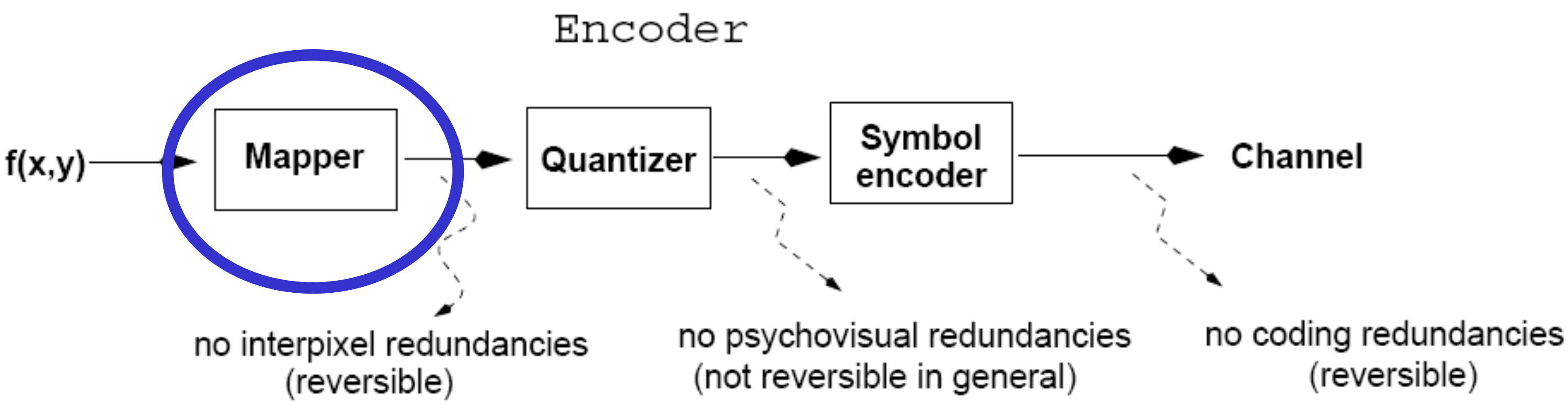
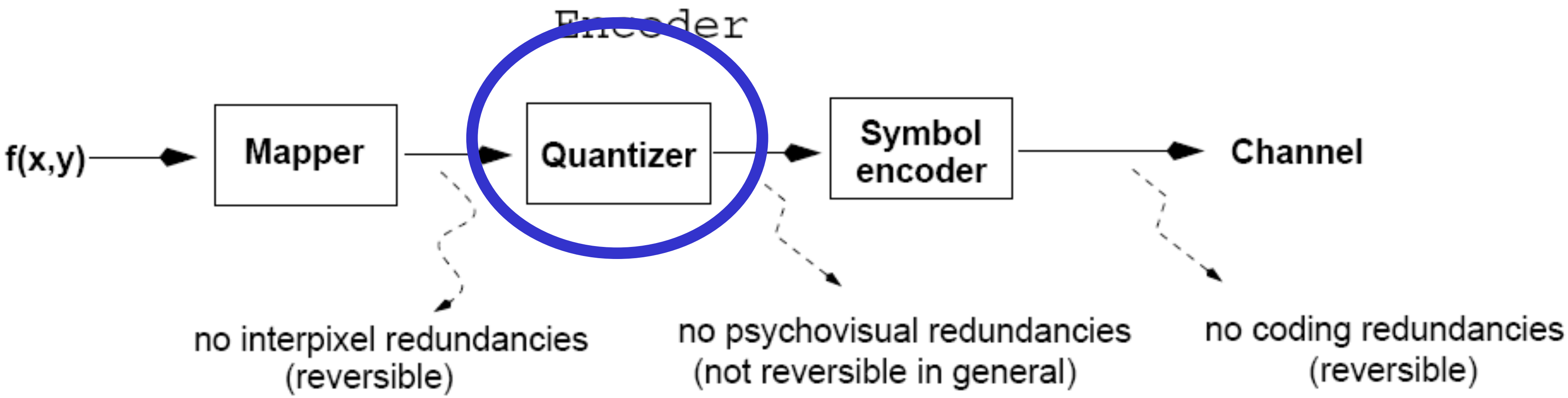


Image Compression Model (cont'd)



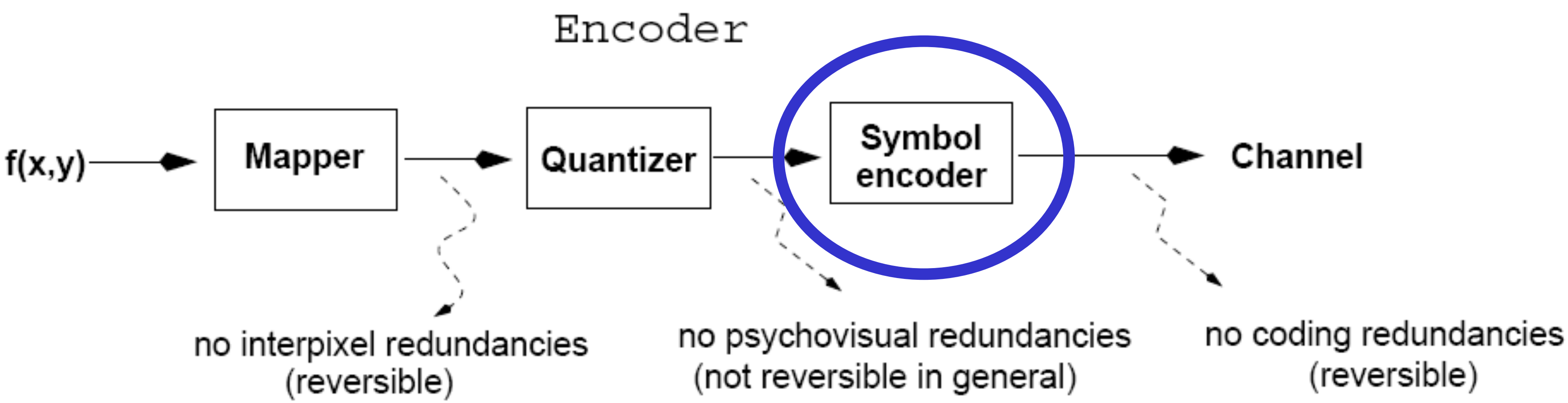
❖ **Mapper:** transforms the input data into a format that facilitates reduction of interpixel redundancies.

Image Compression Model (cont'd)



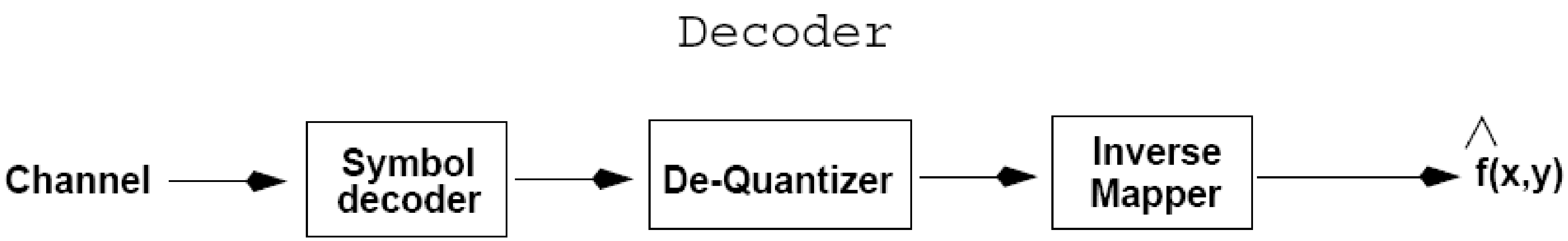
- **Quantizer:** reduces the accuracy of the mapper's output in accordance with some pre-established fidelity criteria.

Image Compression Model (cont'd)



❖ **Symbol encoder**: assigns the shortest code to the most frequently occurring output values.

Image Compression Models (cont'd)



❖ The inverse operations are performed.

❖ But … quantization is irreversible in general.

Fidelity Criteria

$$\hat{f}(x,y) = f(x,y) + e(x,y)$$

❖How close is $f(x,y)$ to $\hat{f}(x,y)$?

❖Criteria

- **Subjective:** **based on human observers**
- **Objective:** **mathematically defined criteria**

Subjective Fidelity Criteria

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

Objective Fidelity Criteria

❖Root mean square error (RMS)

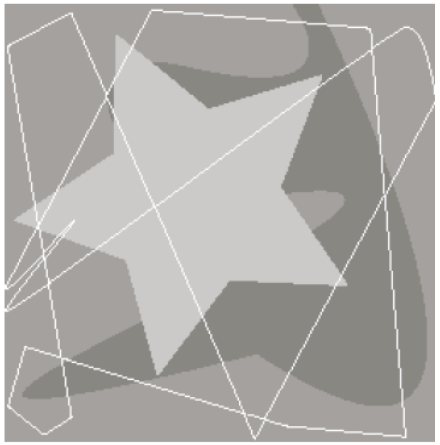
$$e_{rms} = \sqrt{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\hat{f}(x,y) - f(x,y))^2}$$

❖Mean-square signal-to-noise ratio (SNR)

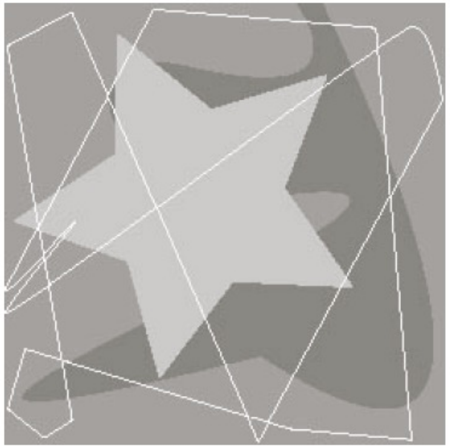
$$SNR_{ms} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\hat{f}(x,y))^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\hat{f}(x,y) - f(x,y))^2}$$

Example

original

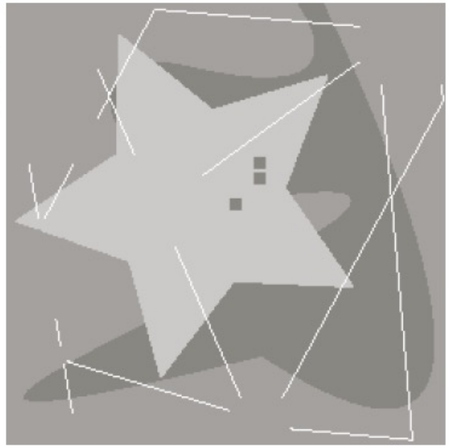


RMS=5.17



RMS=15.67

RMS=14.17

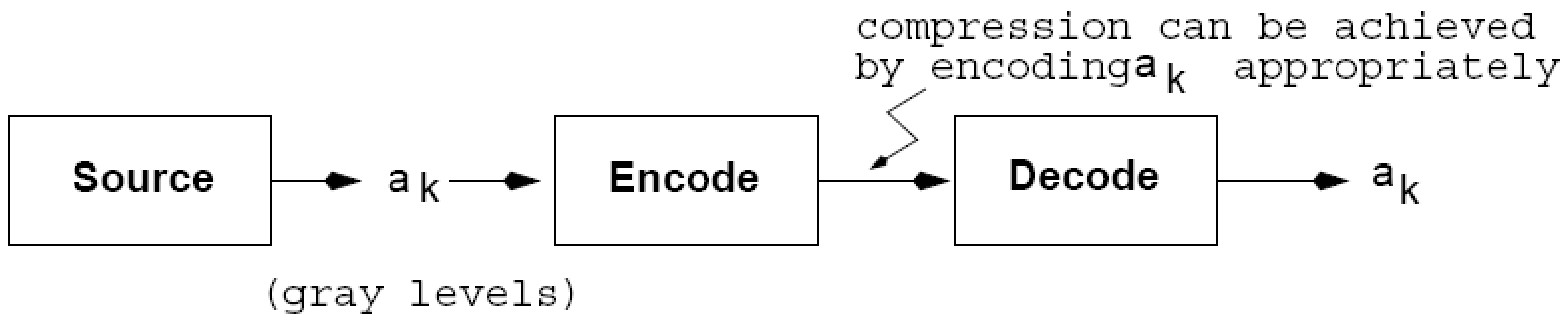


Lossless Compression

$$e(x,y) = \hat{f}(x,y) - f(x,y) = 0$$

- Huffman, Golomb, Arithmetic → coding redundancy
- LZW, Run-length, Symbol-based, Bit-plane → interpixel redundancy

Huffman Coding
(i.e., removes coding redundancy)



- It is a variable-length coding technique.
- It creates the *optimal code* for a set of source symbols.
- Assumption: symbols are encoded one at a time!

Huffman Coding (cont'd)

❖ **<u>Optimal code:</u>** minimizes the number of code symbols per source symbol.

- *<u>Forward Pass</u>*
  1. Sort probabilities per symbol
  2. Combine the lowest two probabilities
  3. Repeat *Step2* until only two probabilities remain.

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

Huffman Coding (cont'd)

❖ *Backward Pass*

Assign code symbols going backwards

Original source			Source reduction							
Sym.	Prob.	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								

Huffman Coding (cont'd)

❖ L_{avg} using Huffman coding:

$$L_{avg} = E(l(a_k)) = \sum_{k=1}^{6} l(a_k)P(a_k) =$$

$$3\text{x}0.1 + 1\text{x}0.4 + 5\text{x}0.06 + 4\text{x}0.1 + 5\text{x}0.04 + 2\text{x}0.3 = 2.2 \text{ bits/symbol}$$

❖ L_{avg} assuming binary codes:

6 symbols, we need a 3-bit code

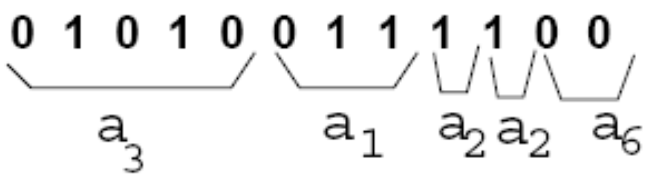
$(a_1\text{: }000,\ a_2\text{: }001,\ a_3\text{: }010,\ a_4\text{: }011,\ a_5\text{: }100,\ a_6\text{: }101)$

$$L_{avg} = \sum_{k=1}^{6} l(a_k)P(a_k) = \sum_{k=1}^{6} 3P(a_k) = 3\sum_{k=1}^{6} P(a_k) = 3 \text{ bits/symbol}$$

Huffman Coding (cont'd)

❖Comments

- **After the code has been created, *coding/decoding* can be implemented using a look-up table.**
- **Decoding can be done in an unambiguous way !!**



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

Arithmetic (or Range) Coding (i.e., removes coding redundancy)

- No assumption on encoding symbols one at a time.
  - **No one-to-one correspondence between source and code words.**

- Slower than Huffman coding but typically achieves better compression.

- A sequence of source symbols is assigned a single arithmetic code word which corresponds to a sub-interval in [0,1]

Arithmetic Coding (cont'd)

- As the number of symbols in the message increases, the interval used to represent it becomes smaller.
  - **Each symbol reduces the size of the interval according to its probability.**

- Smaller intervals require more information units (i.e., bits) to be represented.

Arithmetic Coding (cont'd)

Encode message: $a_1\ a_2\ a_3\ a_3\ a_4$

Source Symbol	Probability
a_1	0.2
a_2	0.2
a_3	0.4
a_4	0.2

1) Assume message occupies [0, 1)

0 1

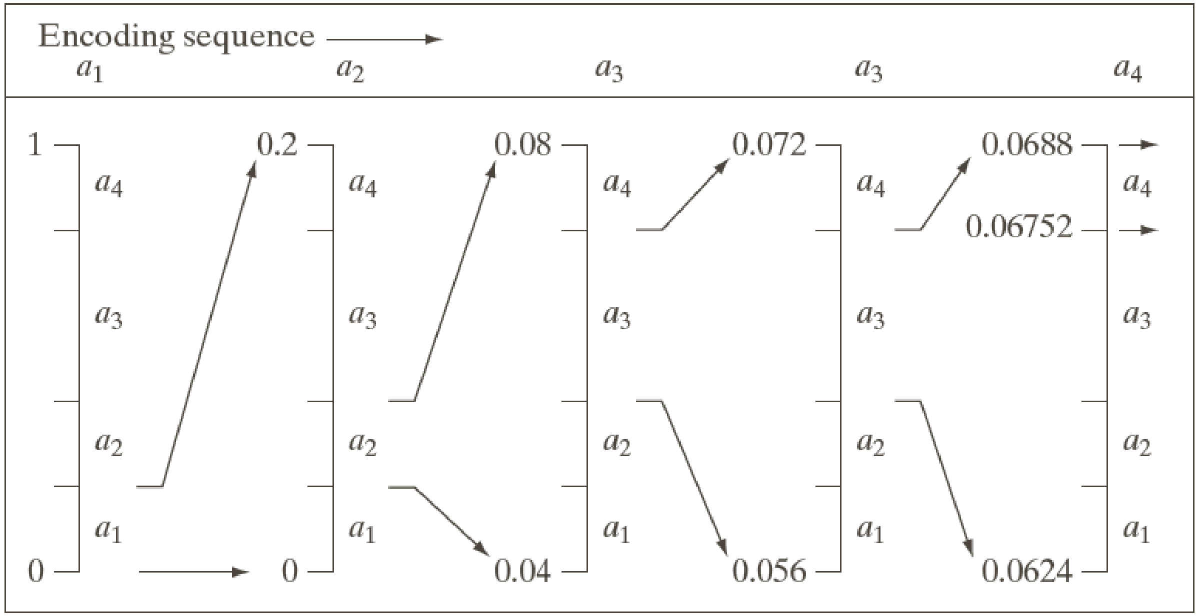
Initial Subinterval
[0.0, 0.2)
[0.2, 0.4)
[0.4, 0.8)
[0.8, 1.0)

2) Subdivide [0, 1) based on the probabilities of α_i

3) Update interval by processing source symbols

Example

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



$a_1\ a_2\ a_3\ a_3\ a_4$

[0.06752, 0.0688)

or,

0.068

Example

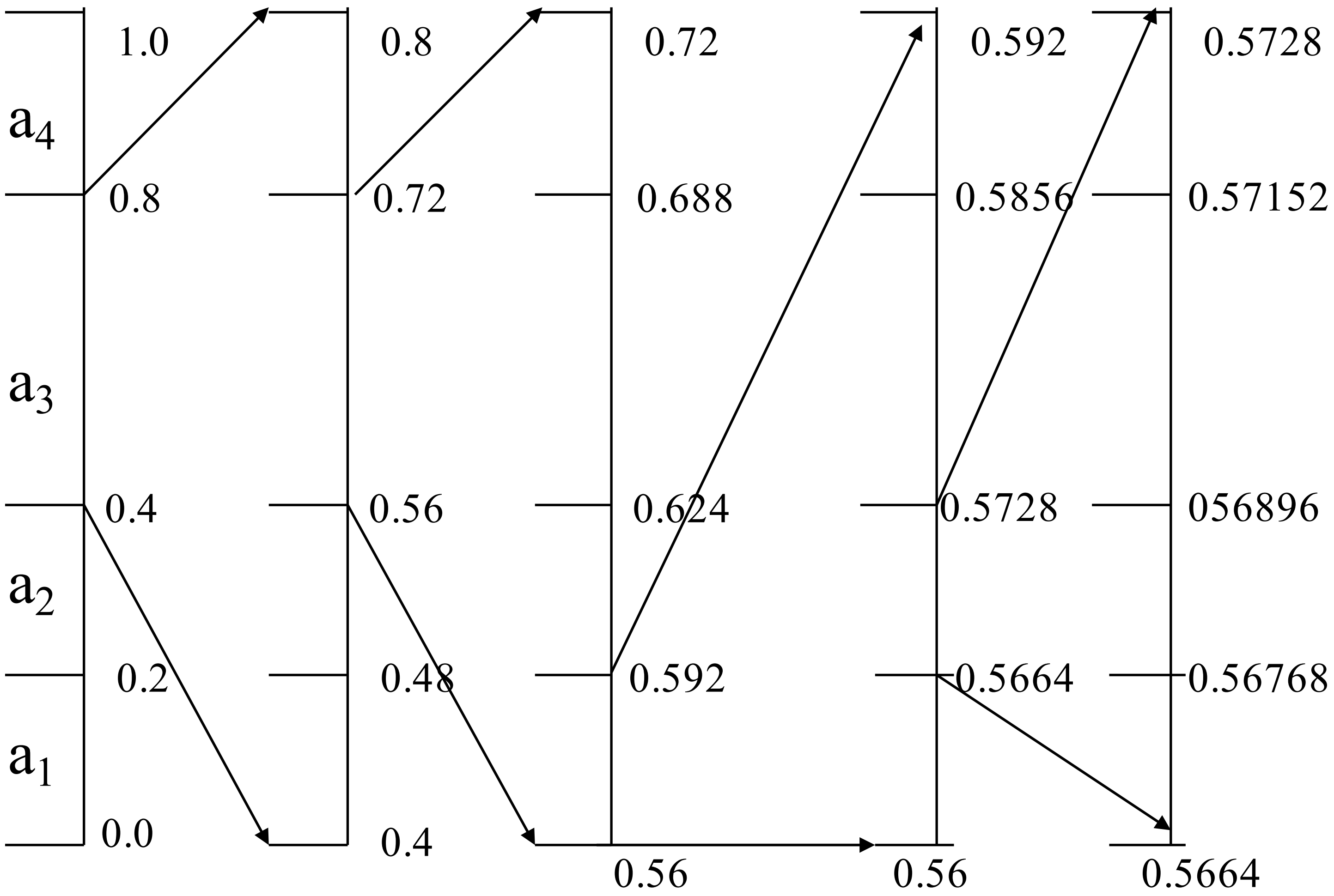
- The message $a_1\ a_2\ a_3\ a_3\ a_4$ is encoded using 3 decimal digits or 0.6 decimal digits per source symbol.

- The entropy of this message is:

$$-(3 \times 0.2\log_{10}(0.2)+0.4\log_{10}(0.4))=0.5786 \text{ digits/symbol}$$

Note: Finite precision arithmetic might cause problems due to truncations!

Arithmetic Coding (cont'd)



Decode 0.572

$a_3\, a_3\, a_1\, a_2\, a_4$

Arithmetic Coding (cont'd)

- Encode
  - **Low = 0**
  - **High = 1**
  - **Loop. For all the symbols.**
    - ✓ Range = high - low
    - ✓ High = low + range * high_range of the symbol being coded
    - ✓ Low = low + range * low_range of the symbol being coded
- Where:
  - **Range, keeps track of where the next range should be.**
  - **High and low, specify the output number.**

Arithmetic Coding (cont'd)

Symbol	Range	Low value	High value
		0	1
b	1	0.5	0.75
a	0.25	0.5	0.625
c	0.125	0.59375	0.625
a	0.03125	0.59375	0.609375

Symbol	Probability	Range
a	2	[0.0 , 0.5)
b	1	[0.5 , 0.75)
c	1	[0.75 , 1.0)

Arithmetic Coding (cont'd)

- Decode
  - **Loop. For all the symbols.**
    - ✓Range = high_range of the symbol - low_range of the symbol
    - ✓Number = number - low_range of the symbol
    - ✓Number = number / range

Arithmetic Coding (cont'd)

Symbol	Range	Number
b	0.25	0.59375
a	0.5	0.375
c	0.25	0.75
a	0.5	0

Symbol	Probability	Range
a	2	[0.0 , 0.5)
b	1	[0.5 , 0.75)
c	1	[0.75 , 1.0)

LZW Coding
(i.e., removes inter-pixel redundancy)

- Requires no priori knowledge of probability distribution of pixels

- Assigns fixed length code words to variable length sequences

- Patented Algorithm US 4,558,302

- Included in GIF and TIFF and PDF file formats

LZW Coding

- A codebook or a dictionary has to be constructed.
  - **Single pixel values and blocks of pixel values**

- For an 8-bit image, the first 256 entries are assigned to the gray levels 0,1,2,..,255.

- As the encoder examines image pixels, gray level sequences (i.e., pixel combinations) that are not in the dictionary are assigned to a new entry.

Example

Consider the following 4 x 4 8 bit 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	-

Initial Dictionary

Example

39 39 126 126

39 39 126 126

39 39 126 126

39 39 126 126

- Is 39 in the dictionary……..Yes
- What about 39-39………….No
- Then add 39-39 in entry 256

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	- 39-39
511	-

Example

concatenated sequence (CS)

(CR) (P)

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

If CS is found:
(1) No Output
(2) CR=CS

If CS not found:
(1) Output D(CR)
(2) Add CS to D
(3) CR=P

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		

Decoding 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.

Run-length coding (RLC)
(i.e., removes interpixel redunancy)

- Used to reduce the size of a repeating string of characters (i.e., runs)

a a a b b b b b b c c → (a,3) (b, 6) (c, 2)

- Encodes a run of symbols into two bytes , a count and a symbol.
- Can compress any type of data but cannot achieve high compression ratios compared to other compression methods.

Run-length coding
(i.e., removes interpixel redunancy)

- Code each contiguous group of 0's and 1's, encountered in a left to right scan of a row, by its <u>length</u>.

1 1 1 1 1 0 0 0 0 0 0 1 → (1,5) (0, 6) (1, 1)

Bit-plane coding
(i.e., removes interpixel redundancy)

- An effective technique to reduce inter pixel redundancy is to process each bit plane individually

- The image is decomposed into a series of binary images.

- Each binary image is compressed using one of well known binary compression techniques.
  - **e.g., Huffman, Run-length, etc.**

Combining Huffman Coding with Run-length Coding

- Once a message has been encoded using Huffman coding, additional compression can be achieved by encoding the lengths of the runs using variable-length coding!

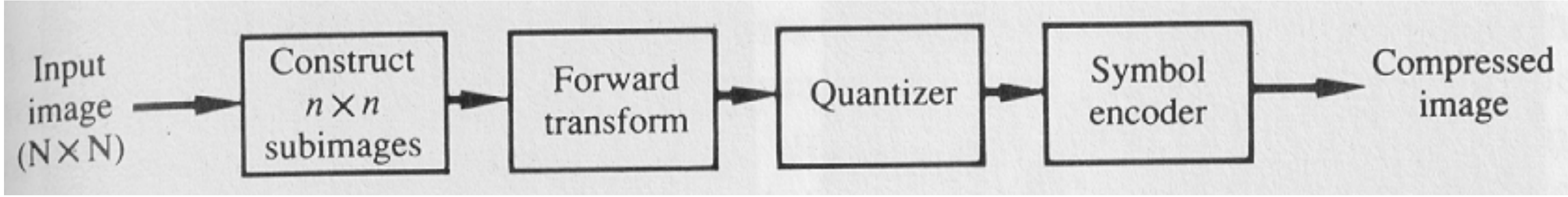
0 1 0 1 0 0 1 1 1 1 0 0

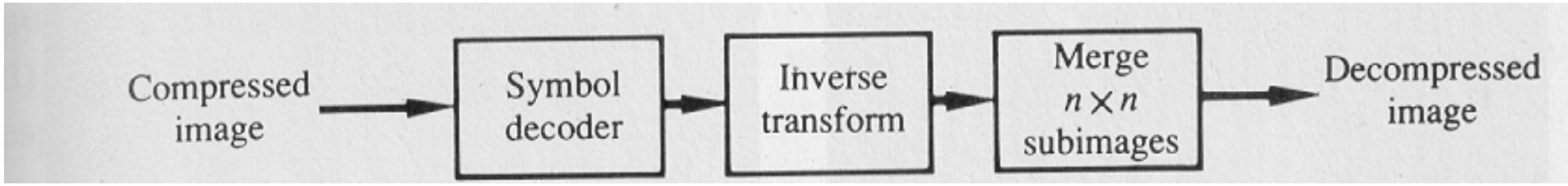
e.g., (0,1)(1,1)(0,1)(1,1)(0,2)(1,4)(0,2)

Lossy Compression

- ❖ Transform the image into a domain where compression can be performed more efficiently.
- ❖ Note that the transformation itself does not compress the image!

~ $(N/n)^2$ subimages

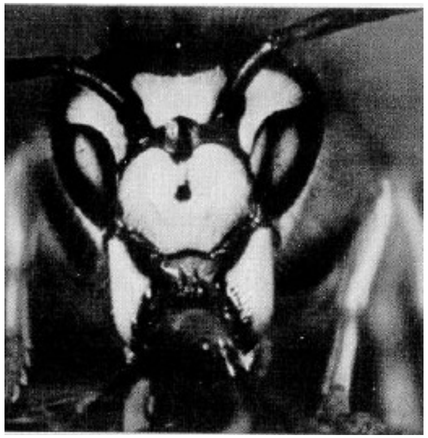


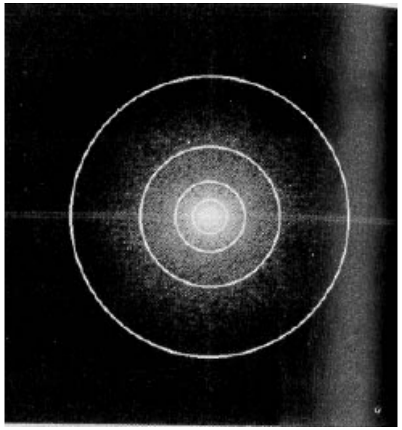


Lossy Compression (cont'd)

❖Example: Fourier Transform

$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)e^{\frac{j2\pi(ux+vy)}{N}}, \quad x,y=0,1,\ldots,N-1$$





The magnitude of the FT decreases, as u, v increase!

K << N

$$\hat{f}(x,y) = \frac{1}{N}\sum_{u=0}^{K-1}\sum_{v=0}^{K-1} F(u,v)e^{\frac{j2\pi(ux+vy)}{N}}, \quad x,y=0,1,\ldots,N-1$$

$$\sum_{x,y}(\hat{f}(x,y) - f(x,y))^2 \text{ is very small !!}$$

Transform Selection

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}T(u,v)h(x,y,u,v)$$

- T(u,v) can be computed using various transformations, for example:
  - **DFT**
  - **DCT (Discrete Cosine Transform)**
  - **KLT (Karhunen-Loeve Transformation)**

DCT

forward

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

u, v=0,1,...,N-1

inverse

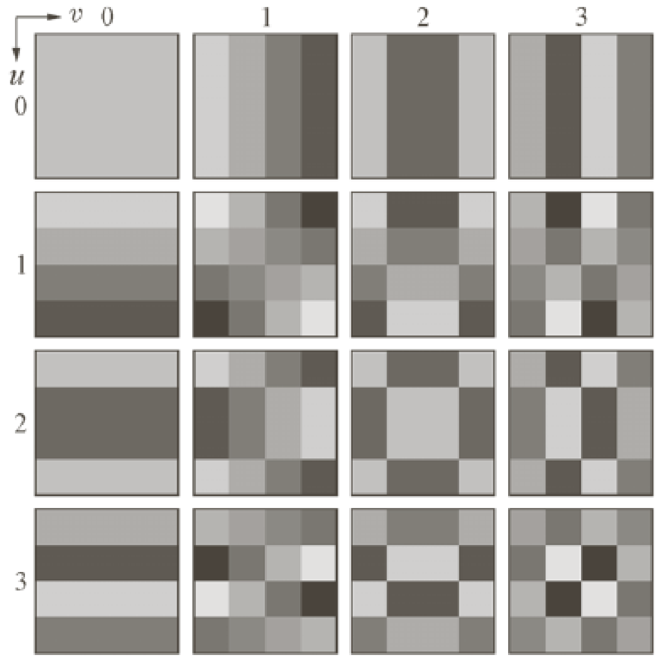
$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v)cos(\frac{(2x+1)u\pi}{2N})cos(\frac{(2y+1)v\pi}{2N}),$$

x, y=0,1,...,N-1

$$\alpha(u) = \begin{cases} \sqrt{1/N} & \text{if u=0} \\ \sqrt{2/N} & \text{if u>0} \end{cases} \qquad \alpha(v) = \begin{cases} \sqrt{1/N} & \text{if v=0} \\ \sqrt{2/N} & \text{if v>0} \end{cases}$$

DCT (cont'd)

❖Basis set of functions for a 4x4 image (i.e.,cosines of different frequencies).



DCT (cont'd)

	DFT	WHT	DCT

8 x 8 subimages

64 coefficients per subimage

50% of the coefficients truncated

RMS error: 2.32 1.78 1.13

DCT (cont'd)

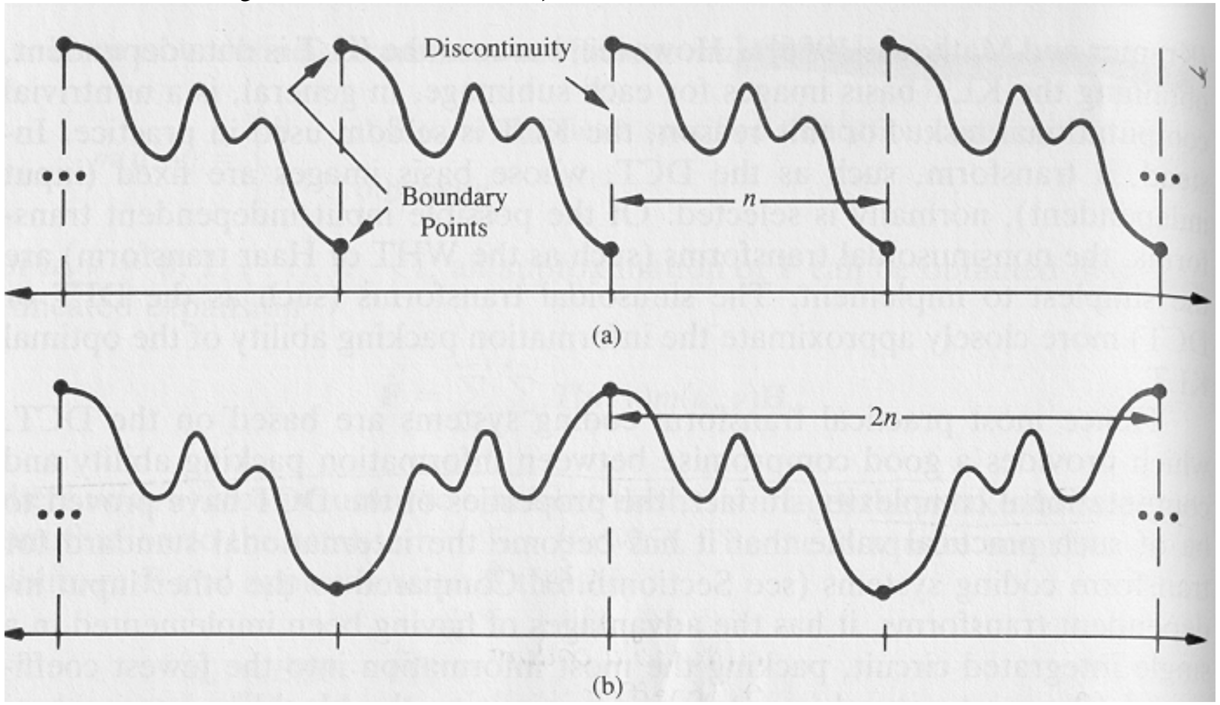
❖DCT minimizes "blocking artifacts" (i.e., boundaries between subimages do not become very visible).

DFT

i.e., n-point periodicity gives rise to discontinuities!

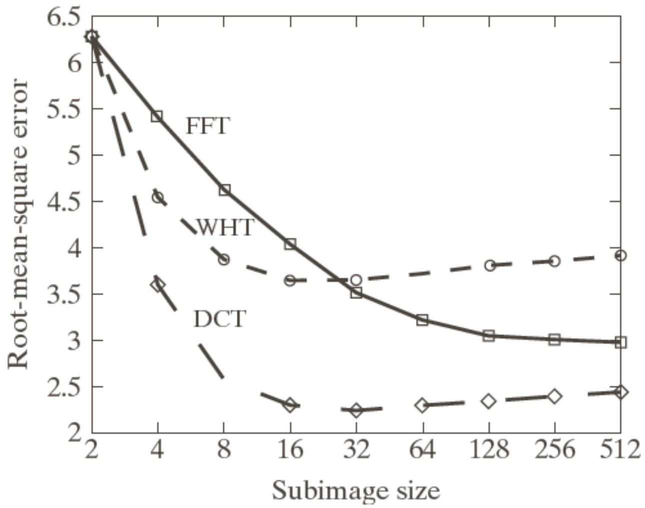
DCT

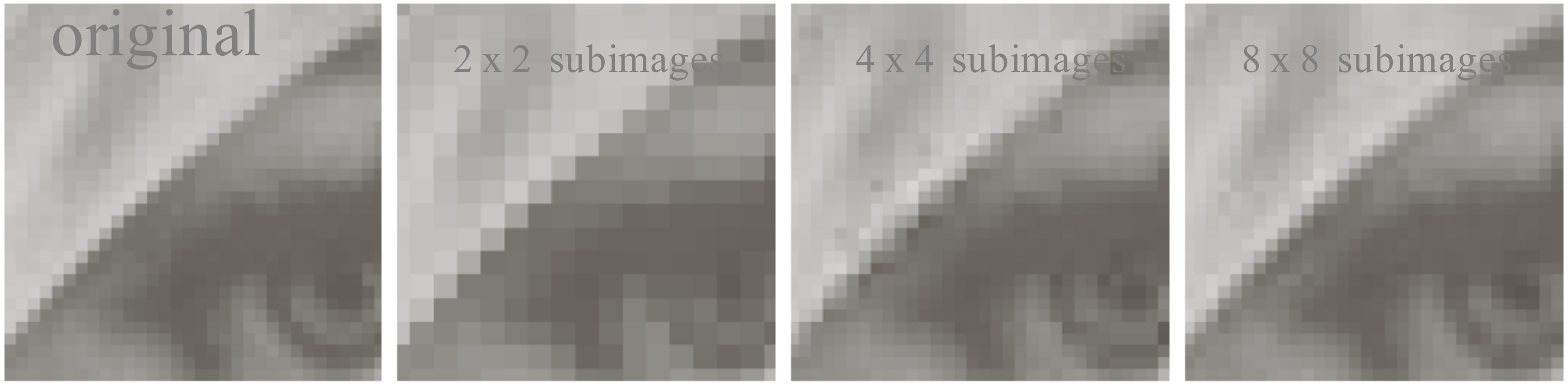
i.e., 2n-point periodicity prevents discontinuities!



DCT (cont'd)

- Subimage size selection





JPEG Compression

- ❖JPEG uses DCT for handling interpixel redundancy.

- ❖Modes of operation:

  (1) Sequential DCT-based encoding

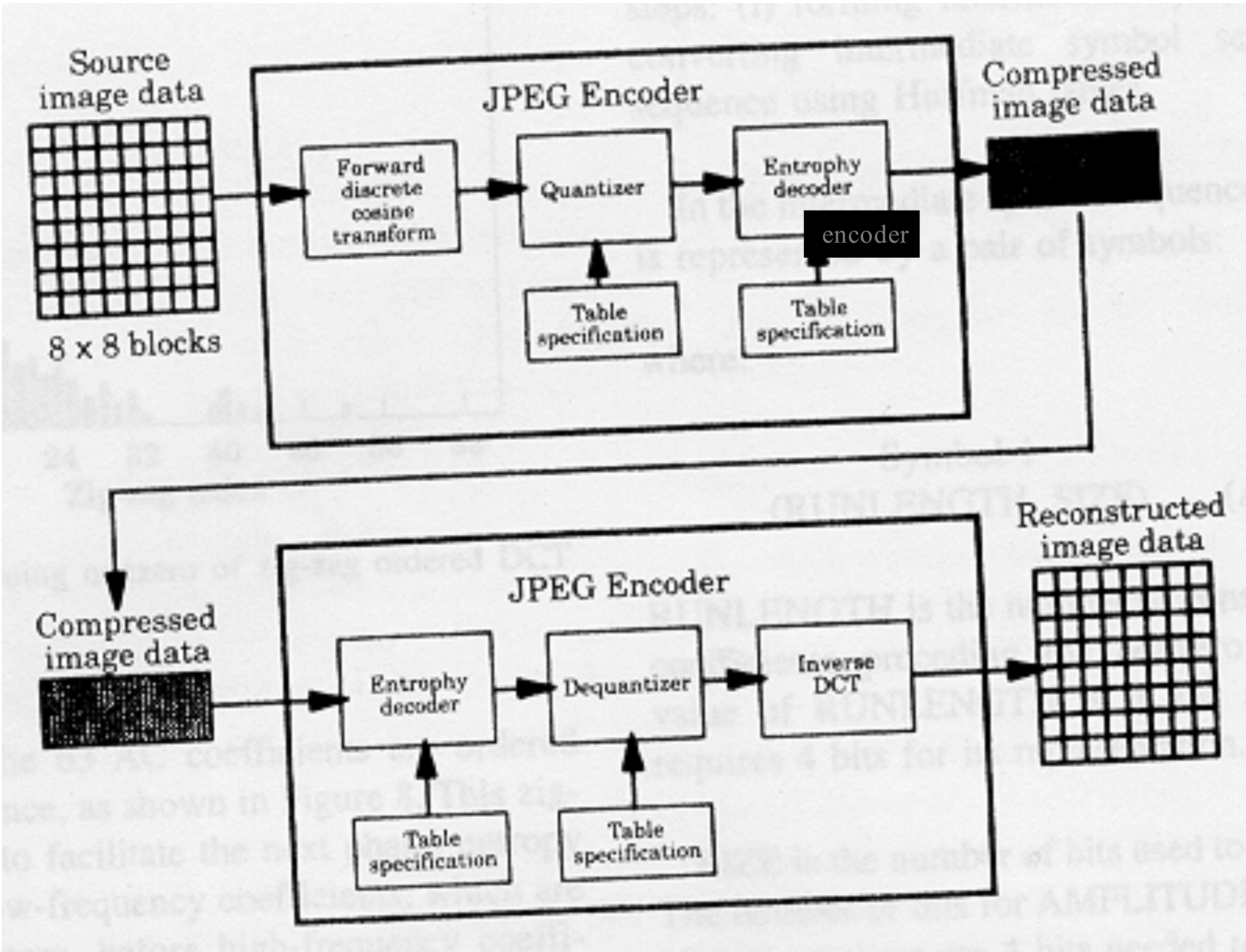
  (2) Progressive DCT-based encoding

  (3) Lossless encoding

  (4) Hierarchical encoding

JPEG Compression

(Sequential DCT-based encoding)



JPEG Steps

1. Divide the image into 8x8 subimages;

   For each subimage do:

2. Shift the gray-levels in the range [-128, 127]
3. Apply DCT (64 coefficients will be obtained: 1 DC coefficient F(0,0), 63 AC coefficients F(u,v)).
4. Quantize the coefficients (i.e., reduce the amplitude of coefficients that do not contribute a lot).

$$C_q(u,v) = Round\left[\frac{C(u,v)}{Q(u,v)}\right]$$

← Quantization Table

JPEG Steps (cont'd)

5. Order the coefficients using zig-zag ordering

   - Place non-zero coefficients first
   - Create long runs of zeros (i.e., good for run-length encoding)
   - See next slide

6. Encode coefficients.

   DC coefficients are encoded using predictive encoding

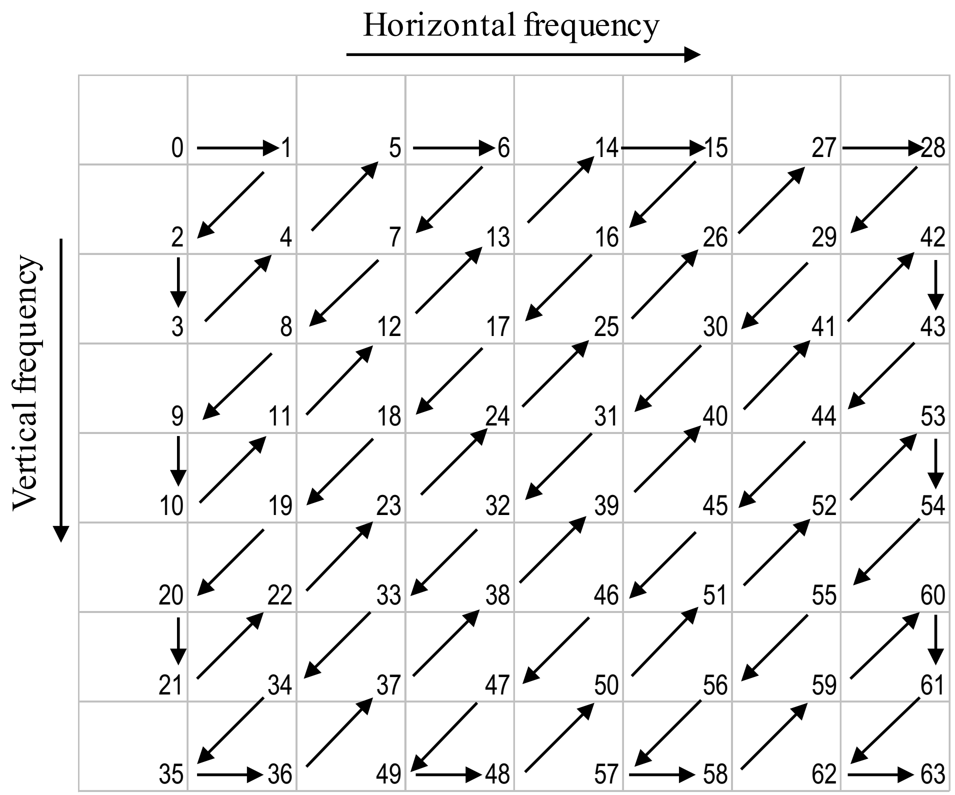
   All coefficients are converted to a binary sequence:

   6.1 Form intermediate symbol sequence

   6.2 Apply Huffman (or arithmetic) coding (i.e., entropy coding)

JPEG Steps (cont'd)

❖ AC coefficients are arranged into a zig-zag sequence:



Shifting and DCT

(a) Original 8   8 block

140	144	147	140	140	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

(b) Shifted block

12	16	19	12	11	27	51	47
16	24	12	19	12	20	39	51
24	27	8	39	35	34	24	44
40	17	28	32	24	27	8	32
34	20	28	20	12	8	19	34
19	39	12	27	27	12	8	34
8	28	-5	39	34	16	12	19
20	27	8	27	24	19	19	8

(c) Block after FDCT Eqn. (5)

185	-17	14	-8	23	-9	-13	-18
20	-34	26	-9	-10	10	13	6
-10	-23	-1	6	-18	3	-20	0
-8	-5	14	-14	-8	-2	-3	8
-3	9	7	1	-11	17	18	15
3	-2	-18	8	8	-3	0	-6
8	0	-2	3	-1	-7	-1	-1
0	-7	-2	1	1	4	-6	0

(non-centered spectrum)

Quantization

❖Quantization Table Example

(d) Quantization table (quality = 2)

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

```
for i=0 to n;
  for j=0 to n;
    Q[i,j]= 1 + (1+i+j)*quality;
  end j;
end i;
```

$$1 \leq quality \leq 25$$

(best - low compression) (worst - high compression)

Quantization (cont'd)

(c) Block after FDCT Eqn. (5)

185	-17	14	-8	23	-9	-13	-18
20	-34	26	-9	-10	10	13	6
-10	-23	-1	6	-18	3	-20	0
-8	-5	14	-14	-8	-2	-3	8
-3	9	7	1	-11	17	18	15
3	-2	-18	8	8	-3	0	-6
8	0	-2	3	-1	-7	-1	-1
0	-7	-2	1	1	4	-6	0

(d) Quantization table (quality = 2)

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

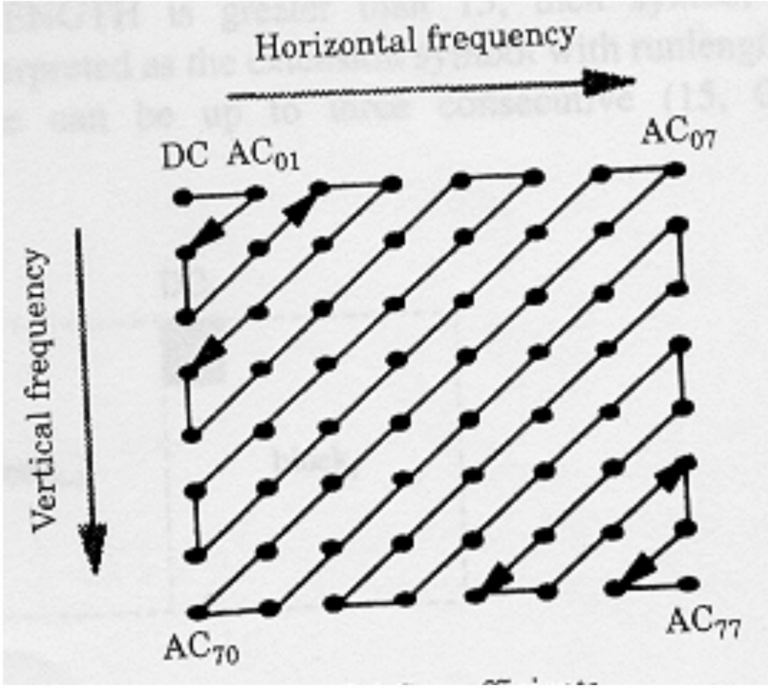
(e) Block after quantization Eqn. (6)

61	-3	2	0	2	0	0	-1
4	-4	2	0	0	0	0	0
-1	-2	0	0	-1	0	-1	0
0	0	1	0	0	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

Zig-Zag Ordering (cont'd)

(e) Block after quantization Eqn. (6)

61	-3	2	0	2	0	0	-1
4	-4	2	0	0	0	0	0
-1	-2	0	0	-1	0	-1	0
0	0	1	0	0	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



(f) Zig-zag sequence

61,-3,4,-1,-4,2,0,2,-2,0,0,0,0,0,2,0,0,0,1,0,0,0,0,0,0,-1,0,0,-1,0,0,
0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Intermediate Coding (cont'd)

(f) Zig-zag sequence

61,-3,4,-1,-4,2,0,2,-2,0,0,0,0,0,2,0,0,0,1,0,0,0,0,0,0,-1,0,0,-1,0,0,
0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

(g) Intermediate synbol sequence

DC

(6)(61),(0,2)(-3),(0,3)(4),(0,1)(-1),(0,3)(-4),(0,2)(2),(1,2)(2),(0,2)(-2),
[redacted](5,2)(2),(3,1)(1),(6,1)(-1),(2,1)(-1),(4,1)(-1),(7,1)(-1),(0,0)

End of Block

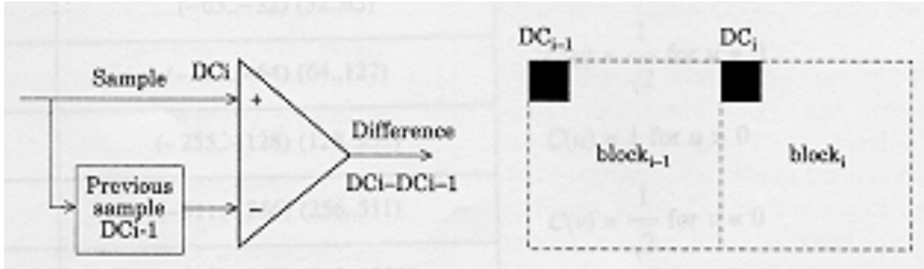
	symbol_1,	symbol_2
DC	(6)	(61)
AC	(0,2)	(-3)

DC/AC Symbol Encoding

❖DC encoding

symbol_1 (size) — symbol_2 (amplitude)

predictive coding:



[-2048, 2047]
11 bits

❖AC encoding

symbol1 (RUN-LENGTH, SIZE) — symbol2 (AMPLITUDE)

0 0 0 0 0 0 476
(6,9)(476)

RUN-LENGTH: #zeros preceding this coefficient
0 <= RUN-LENGTH <= 15
4 bits

SIZE: #bits for amplitude

AMPLITUDE: [-1023, 1024]
10 bits

If RUN-LENGTH > 15, then symbol (15,0) means RUN-LENGTH=16

Entropy Encoding (cont'd)

DC

(d) Intermediate synbol sequence

(6)(61),(0,2)(-3),(0,3)(4),(0,1)(-1),(0,3)(-4),(0,2)(2),(1,2)(2),(0,2)(-2),
[redacted] (5,2)(2),(3,1)(1),(6,1)(-1),(2,1)(-1),(4,1)(-1),(7,1)(-1),(0,0)

(e) Encoded bit sequence (total 98 bits)

11101111010010010000001000110110110111001011111111011
11011101010111110110111000111011011111101001010

End of Block

See Table 8.17-8.19, page 500, 501, 501

Entropy Encoding

Symbol_1
(Variable Length Code (VLC))

(Runlength, size)	Code word
(0,,0) EOB	1010
(0,1)	00
(0,2)	01
(0,3)	100
(1,2)	11011
(2,1)	11100
(3,1)	111010
(4,1)	111011
(5,2)	11111110111
(6,1)	1111011
(7,1)	11111010

Symbol_2
(Variable Length Integer (VLI))

Size	Amplitude range
1	(−1, 1)
2	(−3, −2) (2,3)
3	(−7..−4) (4..7)
4	(−15..−8) (8..15)
5	(−31..−16) (16..31)
6	(−63..−32) (32..63)
7	(−127..−64) (64..127)
8	(−255..−128) (128..255)
9	(−511..−256) (256..511)
10	(−1023..−512) (512..1023)

(1,2)(12) → (11011 1100)
VLC VLI

See Table 8.17-8.19, page 500, 501, 501

JPEG Examples

10 (8k bytes)

50 (21k bytes)

90 (58k bytes)

worst quality,

highest compression

best quality,

lowest compression

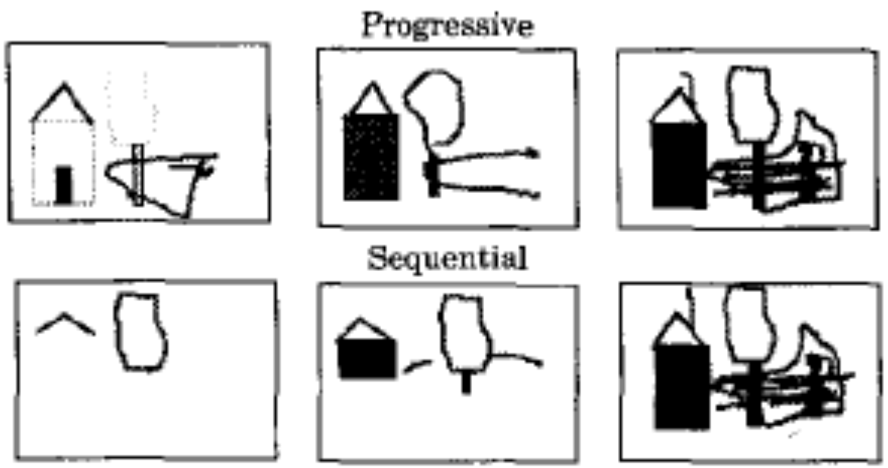
Results

Table 6. Results of JPEG Compression for Grayscale Image 'Lisa' (320 ×240 pixels)

Quality factors	Original number of bits	Compressed number of bits	Compression ratio (Cr)	Bits/pixel (Nb)	RMS error
1	512,000	48,021	10·66	0·75	2·25
2	512,000	30,490	16·79	0·48	2·75
4	512,000	20,264	25·27	0·32	3·43
8	512,000	14,162	36·14	0·22	4·24
15	512,000	10,479	48·85	0·16	5.36
25	512,000	9,034	56·64	0·14	6·40

Progressive JPEG

❖The image is encoded in multiple scans, in order to produce a quick, rough decoded image when transmission time is long.



Progressive JPEG (cont'd)

- Each scan, codes a subset of DCT coefficients.

- Let's look at three methods:

  (1) Progressive spectral selection algorithm
  (2) Progressive successive approximation algorithm
  (3) Combined progressive algorithm

Progressive JPEG (cont'd)

(1) Progressive spectral selection algorithm

- **Group DCT coefficients into several spectral bands**
- **Send low-frequency DCT coefficients first**
- **Send higher-frequency DCT coefficients next**

Band 1: DC coefficient only
Band 2: AC_1 and AC_2 coefficients
Band 3: AC_3, AC_4, AC_5, AC_6, coefficients
Band 4: AC_7...AC_{63}, coefficients

Progressive JPEG (cont'd)

(2) Progressive successive approximation algorithm

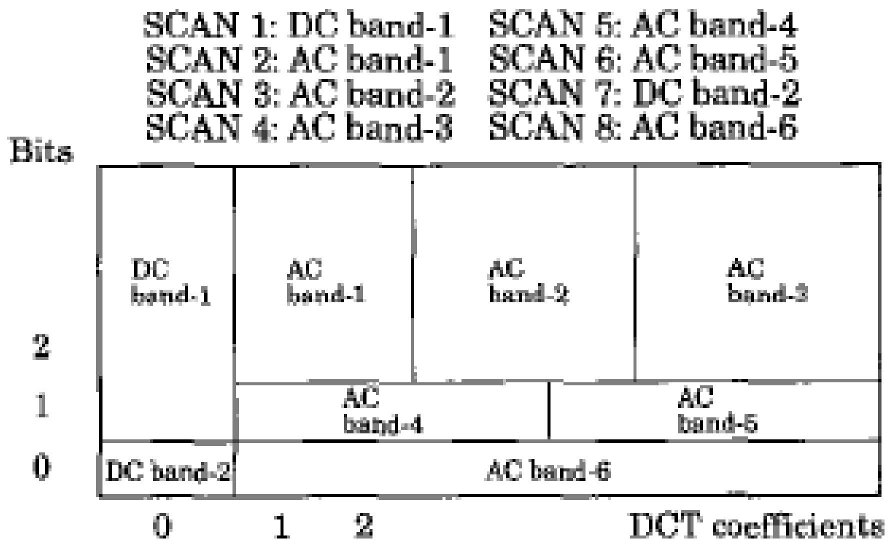
- **All DCT coefficients are sent first with lower precision**
- **Refine them in later scans**

Band 1: All DCT coefficients (divided by four)
Band 2: All DCT coefficients (divided by two)
Band 3: All DCT coefficients (full resolution)

Progressive JPEG (cont'd)

(3) Combined progressive algorithm

- ✍ **Combines spectral selection and successive approximation.**



Results using spectral selection

	Spectral selection
Scan 1	DC, AC1, AC2
Scan 2	AC3–AC9
Scan 3	AC10–AC35
Scan 4	AC 36–AC 63

Table 8. Progressive spectral selection JPEG. (Image 'Cheetah': 320 × 240 pixels –> 512,000 bits)

Scan number	Bits transmitted	Compression ratio	Bits/pixel	RMS error
1	29,005	17·65	0·45	19·97
2	37,237	7·73	1·04	13·67
3	71,259	3·72	2·15	7·90
4	32,489	3·01	2·66	4·59
Sequential JPEG	172,117	2·97	2·69	4·59

Results using successive approximation

Successive

Table 9. Progressive successive approximation JPEG. (Image 'Cheetah': 320 × 240 pixels –> 512,000 bits)

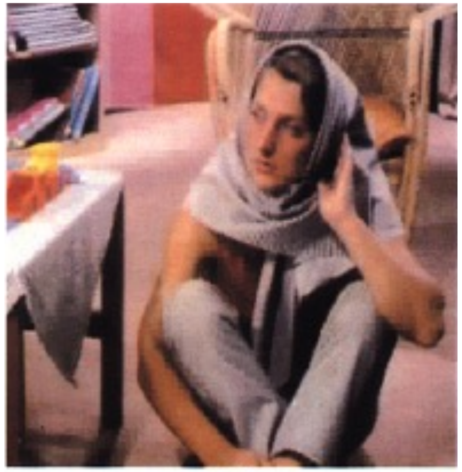
Scan number	Bits transmitted	Compression ratio	Bits/pixel	RMS error
1	26,215	19·53	0·41	22·48
2	34,506	8·43	0·95	12·75
3	63,792	4·11	1.95	7·56
4	95,267	2·33	2·43	4·59
Sequential JPEG	172,117	2·97	2·69	4·59

full resolution

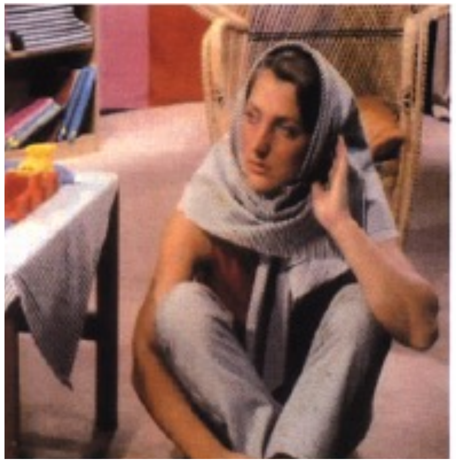
Example using successive approximation

after 0.9s

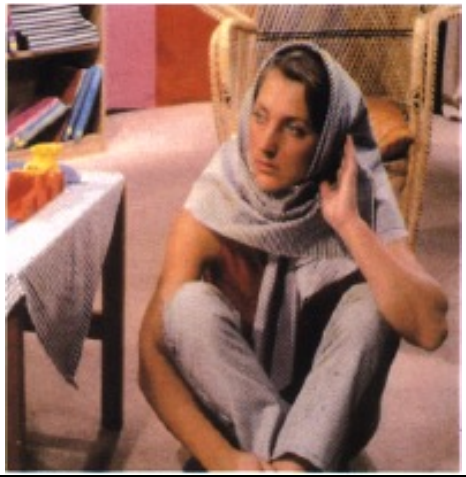
after 1.6s



after 3.6s

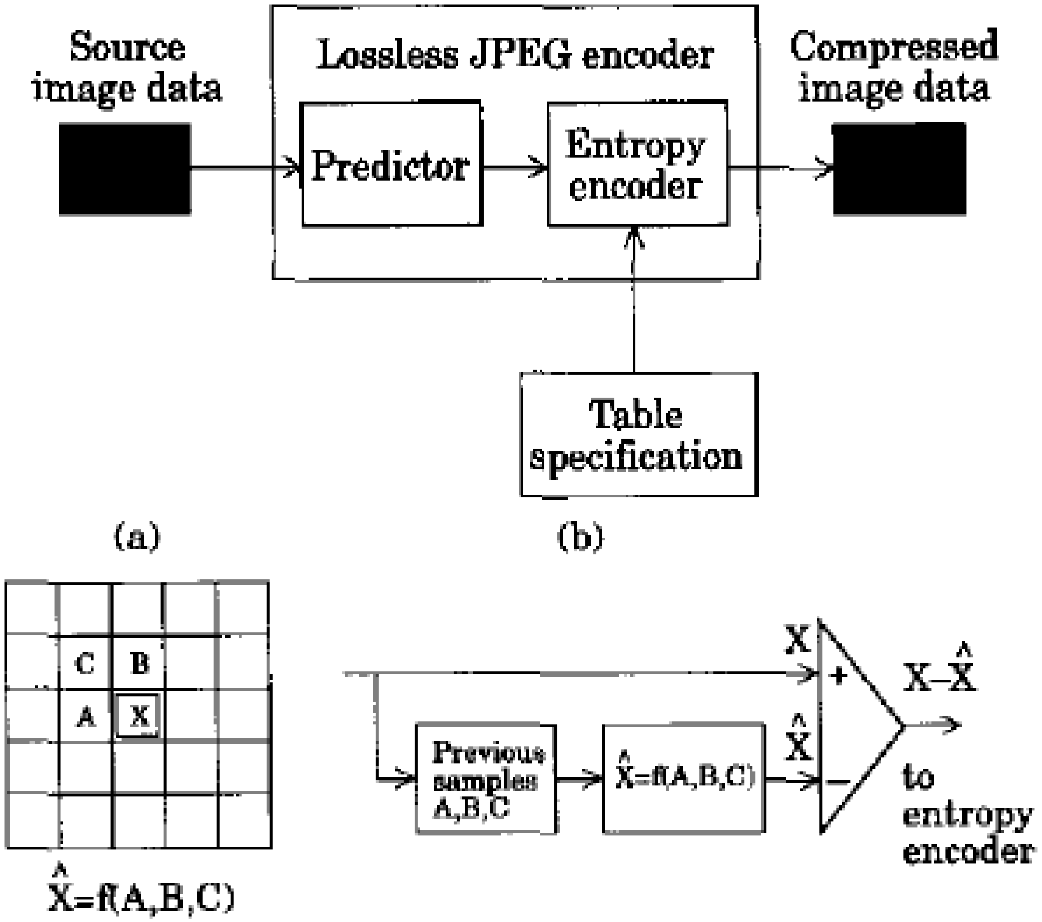


after 7.0s



Lossless JPEG

Use a predictive algorithm instead of DCT-based



Fingerprint Compression

- An image coding standard for digitized fingerprints, developed and maintained by:
  - **FBI**
  - **Los Alamos National Lab (LANL)**
  - **National Institute for Standards and Technology (NIST).**

- The standard employs a discrete wavelet transform-based algorithm (*Wavelet/Scalar Quantization or WSQ*).

Memory Requirements

- FBI is digitizing fingerprints at 500 dots per inch with 8 bits of grayscale resolution.
- A single fingerprint card turns into about 10 MB of data!

A sample fingerprint image

768 x 768 pixels =589,824 bytes

Preserving Fingerprint Details

The "white" spots in the middle of the black ridges are *sweat pores*

They're admissible points of identification in court, as are the little black flesh "islands" in the grooves between the ridges

These details are just a couple pixels wide!

What compression scheme should be used?

❖Better use a lossless method to preserve every pixel perfectly.

❖Unfortunately, in practice lossless methods haven't done better than 2:1 on fingerprints!

❖Does JPEG work well for fingerprint compression?

Results using JPEG compression

file size 45853 bytes
compression ratio: 12.9

The fine details are pretty much history, and the whole image has this artificial "blocky" pattern superimposed on it.

The blocking artifacts affect the performance of manual or automated
systems!

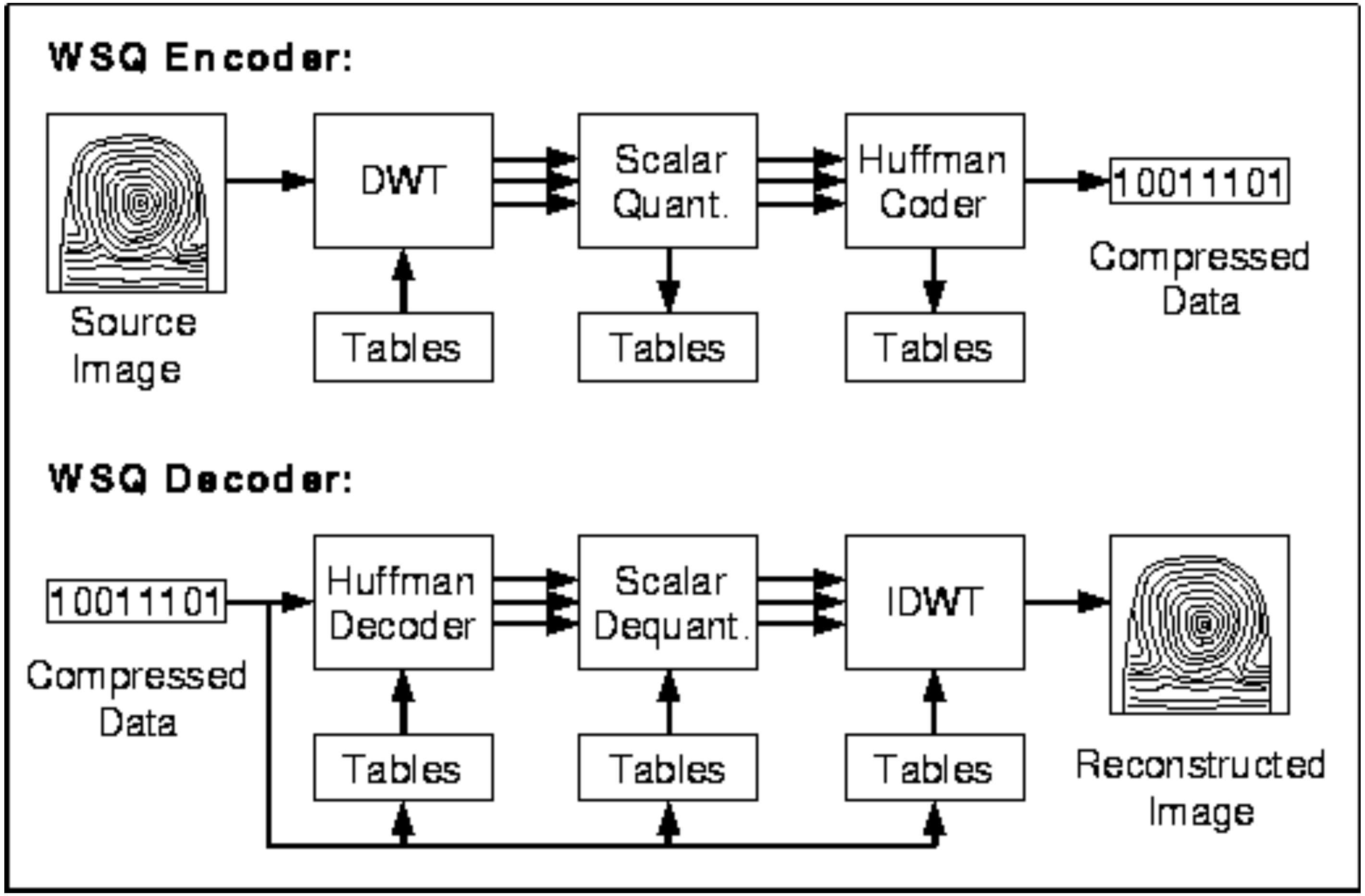
Results using WSQ compression

file size 45621 bytes
compression ratio: 12.9

The fine details are preserved better
than they are with JPEG.

NO blocking artifacts!

WSQ Algorithm



Varying compression ratio

- FBI's target bit rate is around 0.75 bits per pixel (bpp)
  - **i.e., corresponds to a target compression ratio of 10.7 (assuming 8-bit images)**

- This target bit rate is set via a "knob" on the WSQ algorithm.
  - **i.e., similar to the "quality" parameter in many JPEG implementations.**

Varying compression ratio (cont'd)

- In practice, the WSQ algorithm yields a higher compression ratio than the target because of unpredictable amounts of lossless entropy coding gain.
  - **i.e., mostly due to variable amounts of blank space in the images.**

- Fingerprints coded with WSQ at a target of 0.75 bpp will actually come in around 15:1

Varying compression ratio (cont'd)

Original image 768 x 768 pixels (589824 bytes)

Varying compression ratio (cont'd) 0.9 bpp compression

WSQ image, file size 47619 bytes, compression ratio 12.4

JPEG image, file size 49658 bytes, compression ratio 11.9

Varying compression ratio (cont'd) 0.75 bpp compression

WSQ image, file size 39270 bytes
compression ratio 15.0

JPEG image, file size 40780 bytes,
compression ratio 14.5

Varying compression ratio (cont'd)
0.6 bpp compression

WSQ image, file size 30987 bytes,
compression ratio 19.0

JPEG image, file size 30081 bytes,
compression ratio 19.6