Lecture 15

Color Processing, Compression

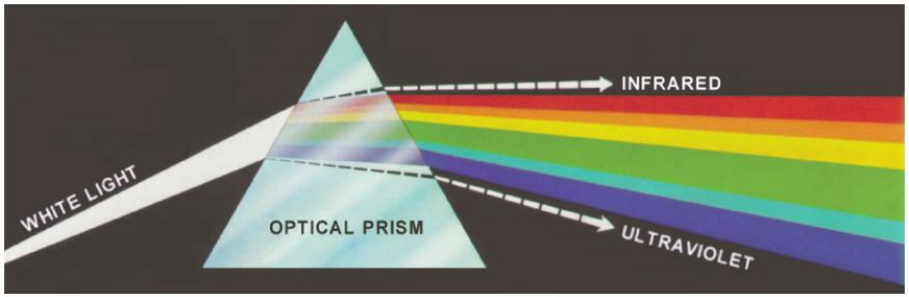
COLOR IMAGE PROCESSING

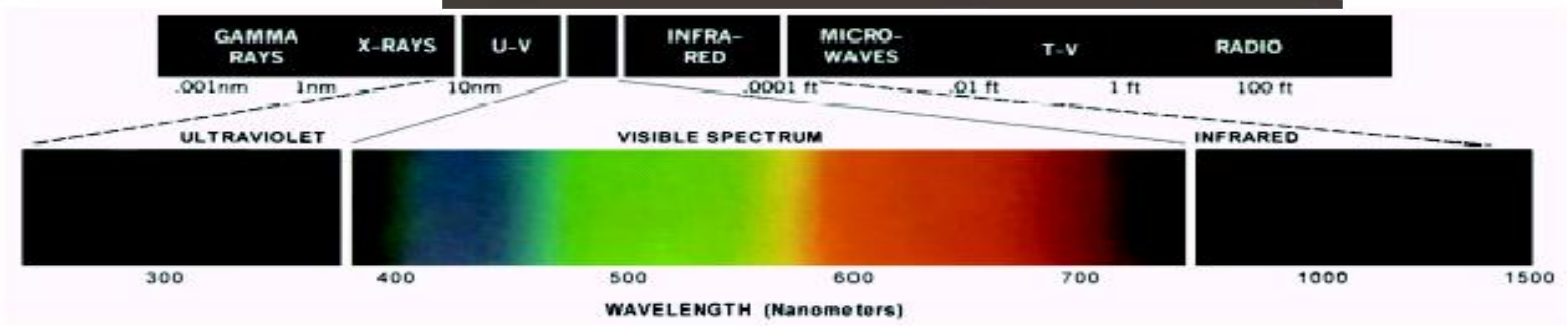
COLOR IMAGE PROCESSING

- Color Importance
  - Color is an excellent descriptor
    - Suitable for object Identification and Extraction
  - Discrimination
    - Humans can distinguish thousands of color shades and intensities but few shades of gray levels
- Color Image Processing
  - Full-Color Processing
    - Color is acquired with a full-color sensor
  - Pseudo-Color Processing
    - Assigning colors to monochrome images

COLOR FUNDAMENTALS

- Colors that humans perceive in an object are determined by the nature of the light reflected from the object
- Visible light is composed of a relatively narrow band of frequencies in the electromagnetic spectrum
- A body that reflects light that is balanced in all visible wavelengths appears white to the observer
- A body that favours reflectance in a limited range of the visible spectrum exhibits some shades of color
- Green objects reflect light with wavelengths primarily in the 500 to 570 nm range while absorbing most of the energy at other wavelengths





HUMAN PERSCEPTION OF COLOR

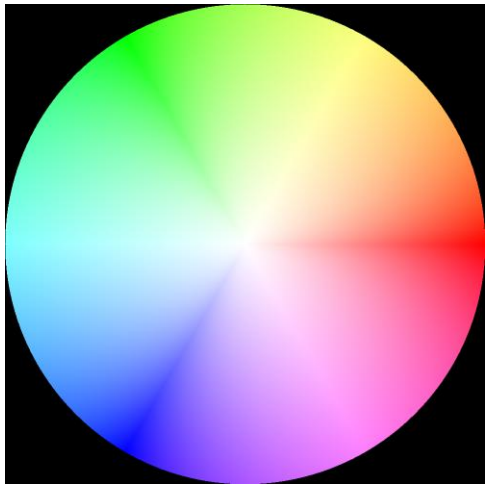
- Retina contains receptors
  - Cones
    - Day vision, can perceive color tone
    - Red, green, and blue cones
  - Rods
    - Night vision, perceive brightness only
- Color sensation
  - Luminance (brightness)
  - Chrominance
    - Hue (color tone)
    - Saturation (color purity)

Monochromatic images

- Image processing - static images -
- Monochromatic static image - continuous image function f(x,y)
  - arguments - two co-ordinates (x,y)
- Digital image functions - represented by matrices
  - co-ordinates = integer numbers
  - Cartesian (horizontal x axis, vertical y axis)
  - OR (row, column) matrices
- Monochromatic image function range
  - lowest value - black
  - highest value - white
- Limited brightness values = gray levels

Chromatic images

- **Colour**
  - **Represented by vector not scalar**
    - **Red, Green, Blue (RGB)**
    - **Hue, Saturation, Value (HSV)**
    - **luminance, chrominance (Yuv , Luv)**



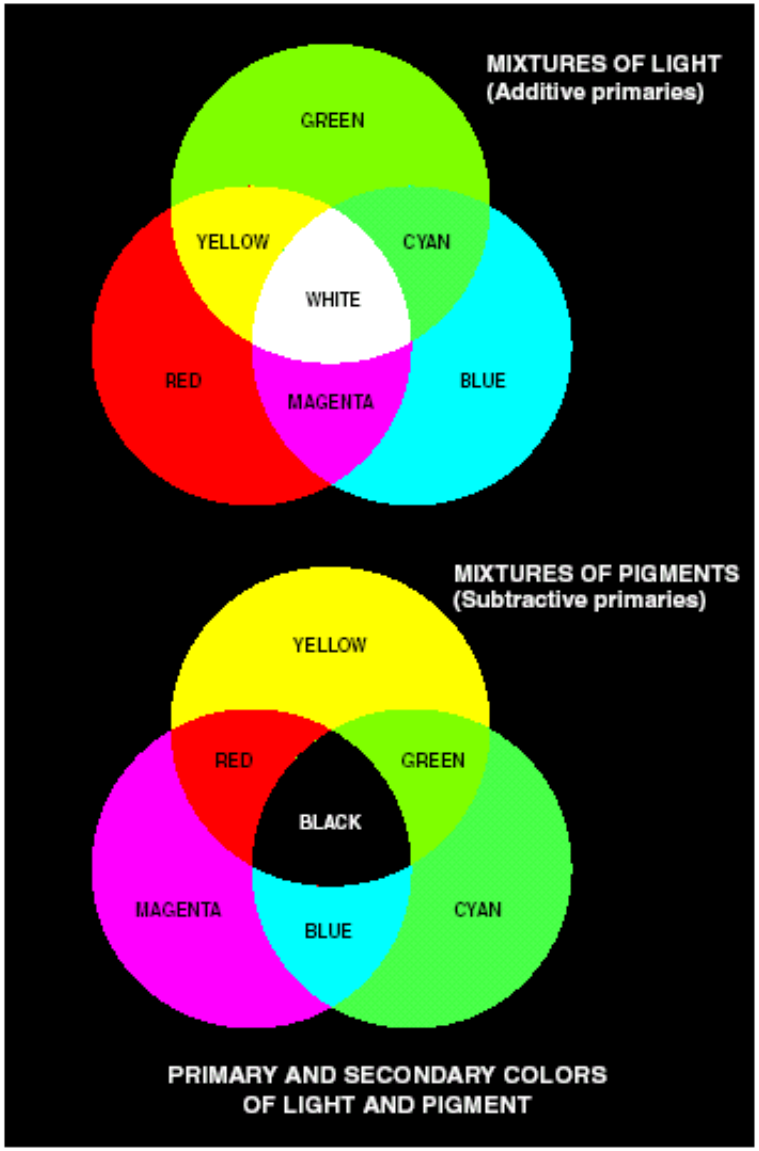
PRIMARY AND SECONDARY COLORS OF LIGHT AND PIGMENTS

- **The primary colors can be added to produce the secondary colors of light**
- **The primary colors of light and primary colors of pigments are different**

Magenta = Red + Blue
Cyan = Blue + Green
Yellow = Green + Red

- **For pigments, a primary color is defined as one that absorbs a <u>primary color of light</u> and reflects the other two**
- **Therefore, the <u>primary colors of pigments</u> are magenta, cyan, and yellow**

Magenta = White - Green
Cyan = White - Red
Yellow = White - Blue



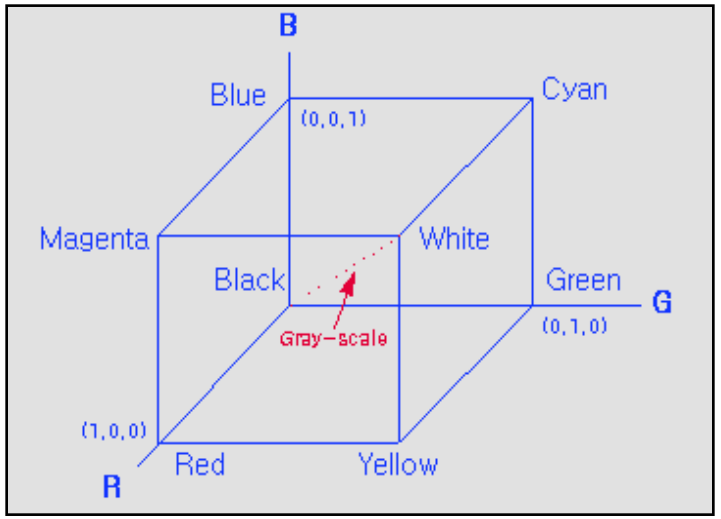
COLOR MODELS

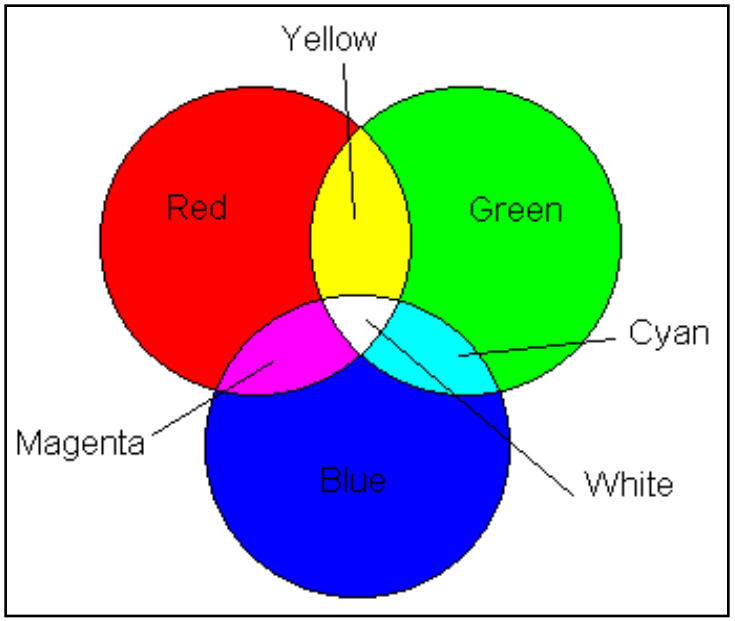
- Color Model
  - Specify colors in a standard way
  - A coordinate system that each color is represented by a single point.
- Most used models:
  - RGB model (Monitor/TV)
  - CMY model (3-color Printers)
  - HSI model (Color Image Processing and Description)

RGB COLOR MODEL

- **Pixel Depth:** The number of bits used to represent each pixel in RGB space.
- **Full-color** image: 24-bit RGB color image.
  - (R, G, B) = (8 bits, 8 bits, 8 bits)
  - Number of colors:

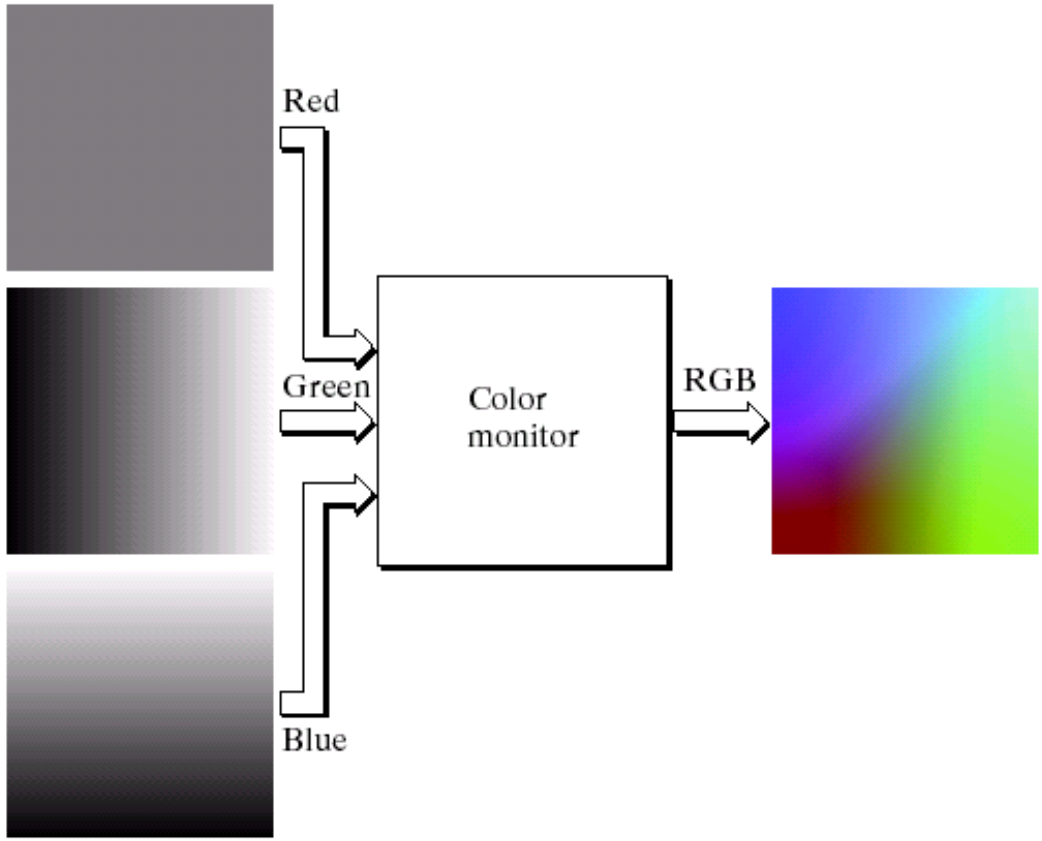
$$\left(2^8\right)^3 = 16{,}777{,}216$$





a
b

FIGURE 6.9
(a) Generating the RGB image of the cross-sectional color plane (127, G, B). (b) The three hidden surface planes in the color cube of Fig. 6.8.



COLOR IMAGE - RGB

Color Image

a b
c d

FIGURE 6.38
(a) RGB image.
(b) Red component image.
(c) Green component.
(d) Blue component.

R-Channel

G-Channel

B-Channel

CMY Model

- Color Printer, Color Copier
- RGB data to CMY

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

HSI COLOR MODEL

- Human description of color is Hue, Saturation and Brightness:
- Hue
  - represents dominant color as perceived by an observer. It is an attribute associated with the dominant wavelength.
- Saturation
  - refers to the relative purity or the amount of white light mixed with a hue. The pure spectrum colors are fully saturated.
    - Pure colors are fully saturated.
    - Pink is less saturated.
- Intensity
  - reflects the brightness.

HSI Color Model

- **The HSI model uses three measures to describe colors:**
  - **Hue: A color attribute that describes a pure color (pure yellow, orange or red)**
  - **Saturation: Gives a measure of how much a pure color is diluted with white light**
  - **Intensity: Intensity is the same achromatic notion that we have seen in grey level images**

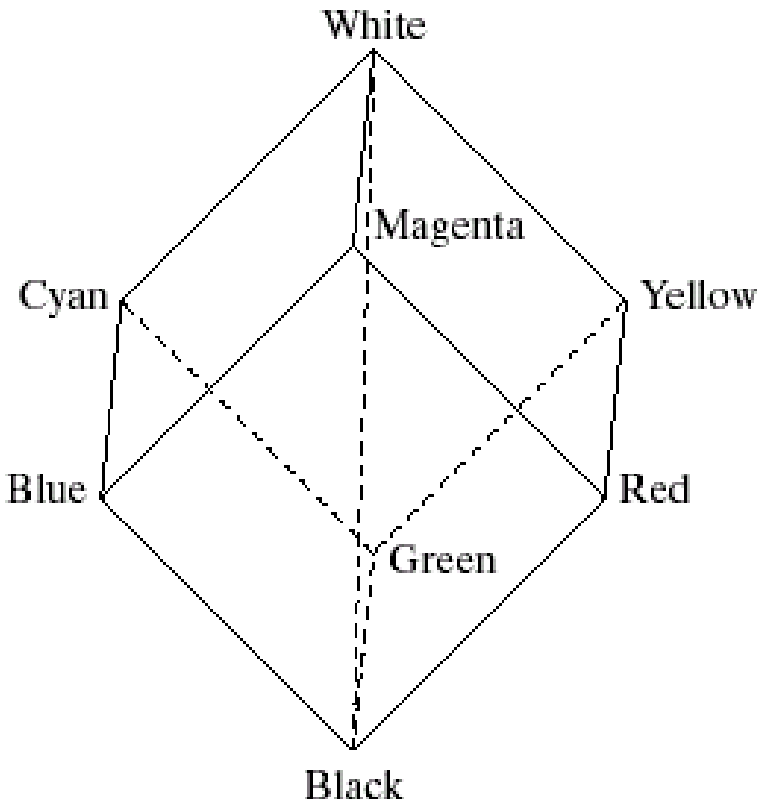
HSI Color Model

- Intensity can be extracted from RGB images
- Remember the diagonal on the RGB color cube that we saw previously ran from black to white
- Now consider if we stand this cube on the black vertex and position the white vertex directly above it

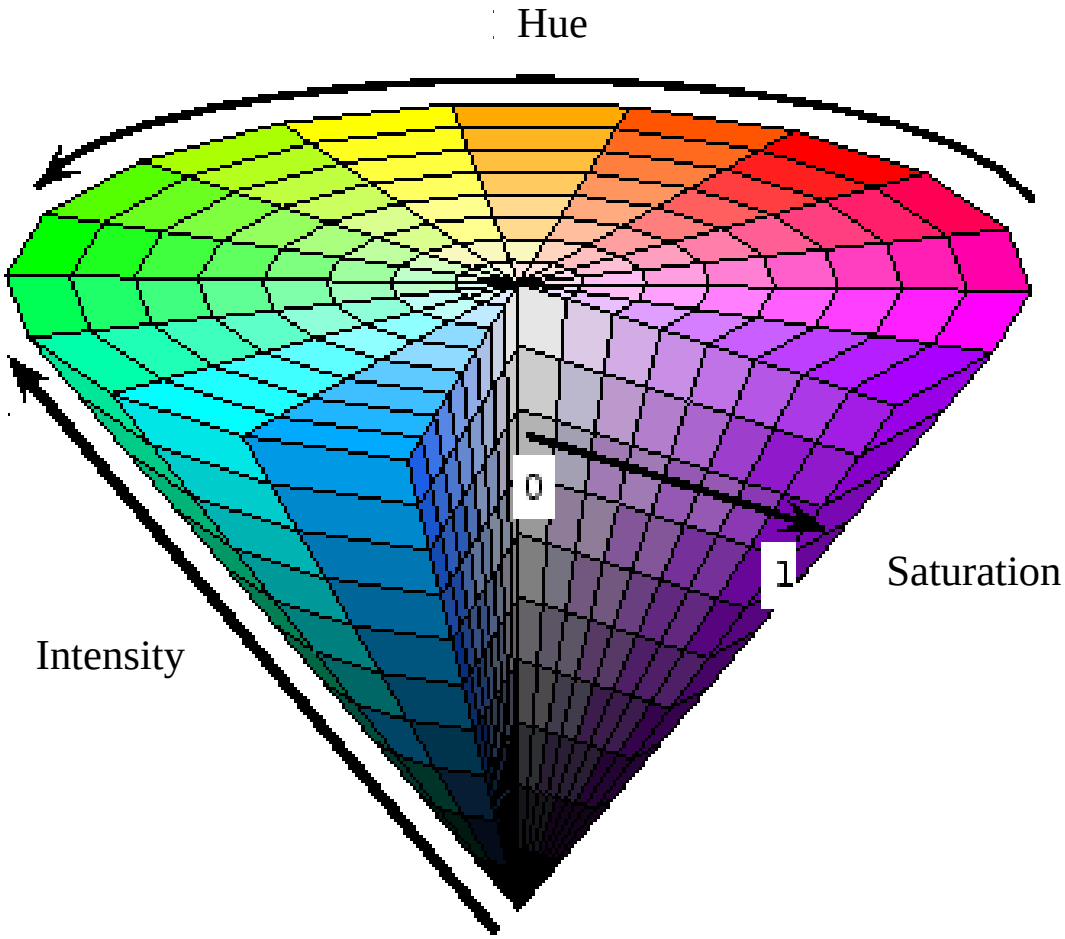
HSI Color Model

- **The intensity component of any color can be determined by passing a plane *perpendicular* to the intensity axis and containing the color point**

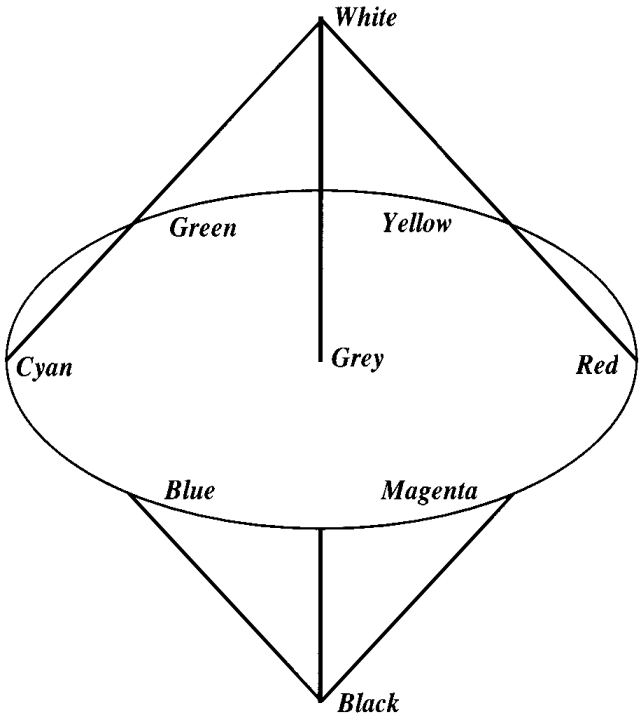
- **The intersection of the plane with the intensity axis gives us the intensity component of the color**

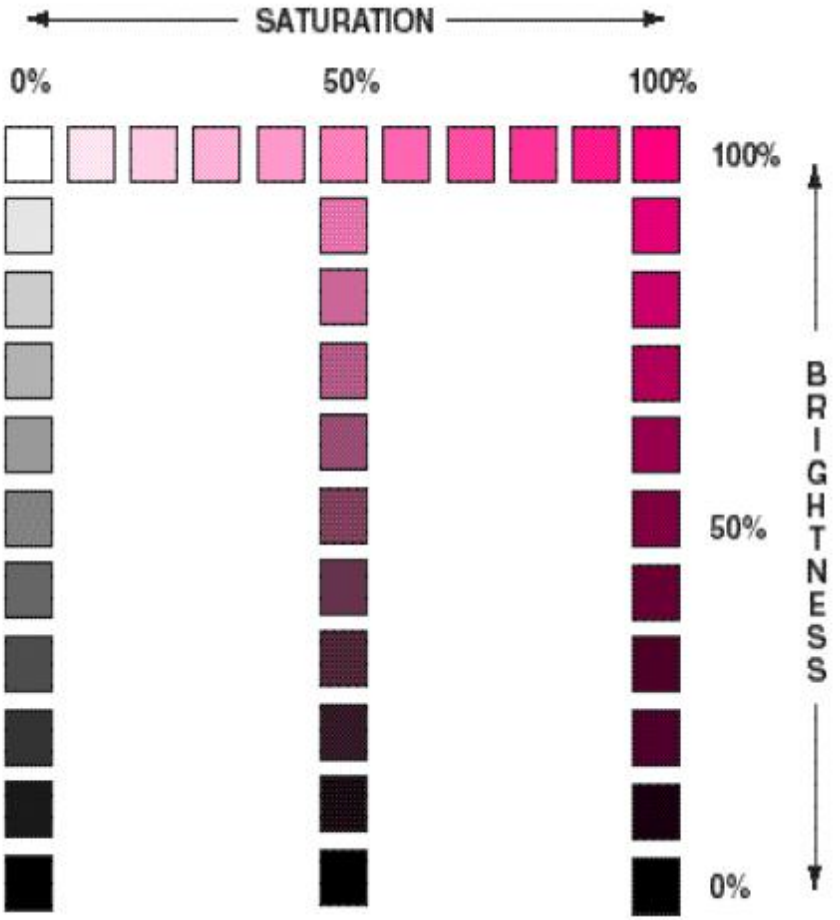


HSI Color Model



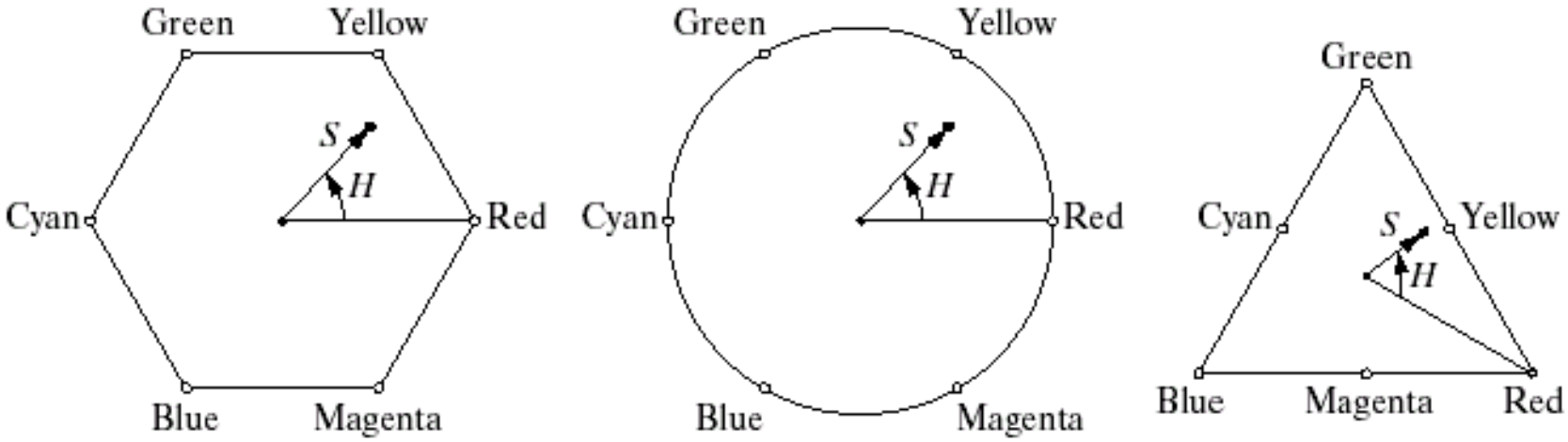
HSI COLOR MODEL- SINGLE HUE



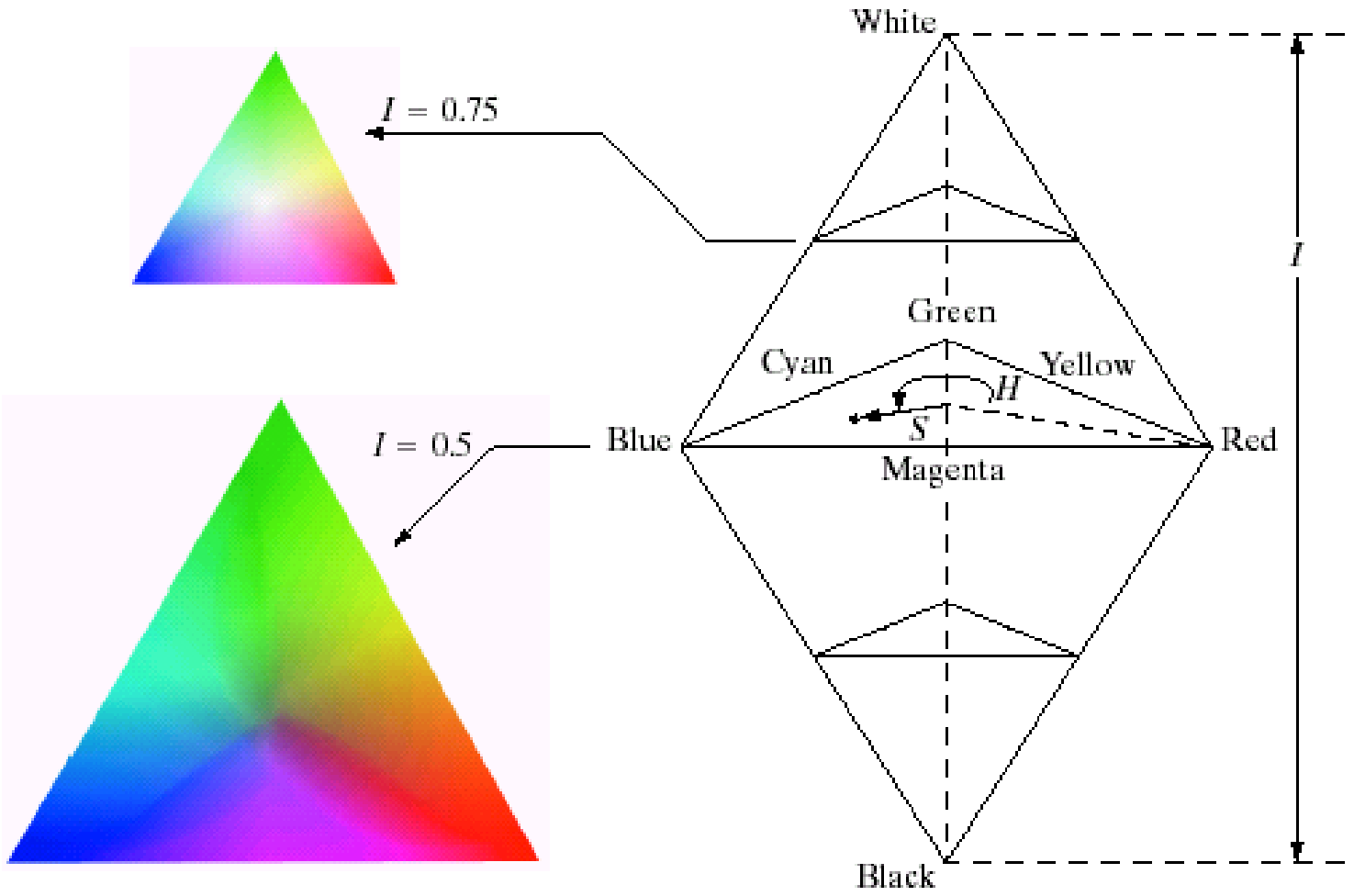


HSI Color Model

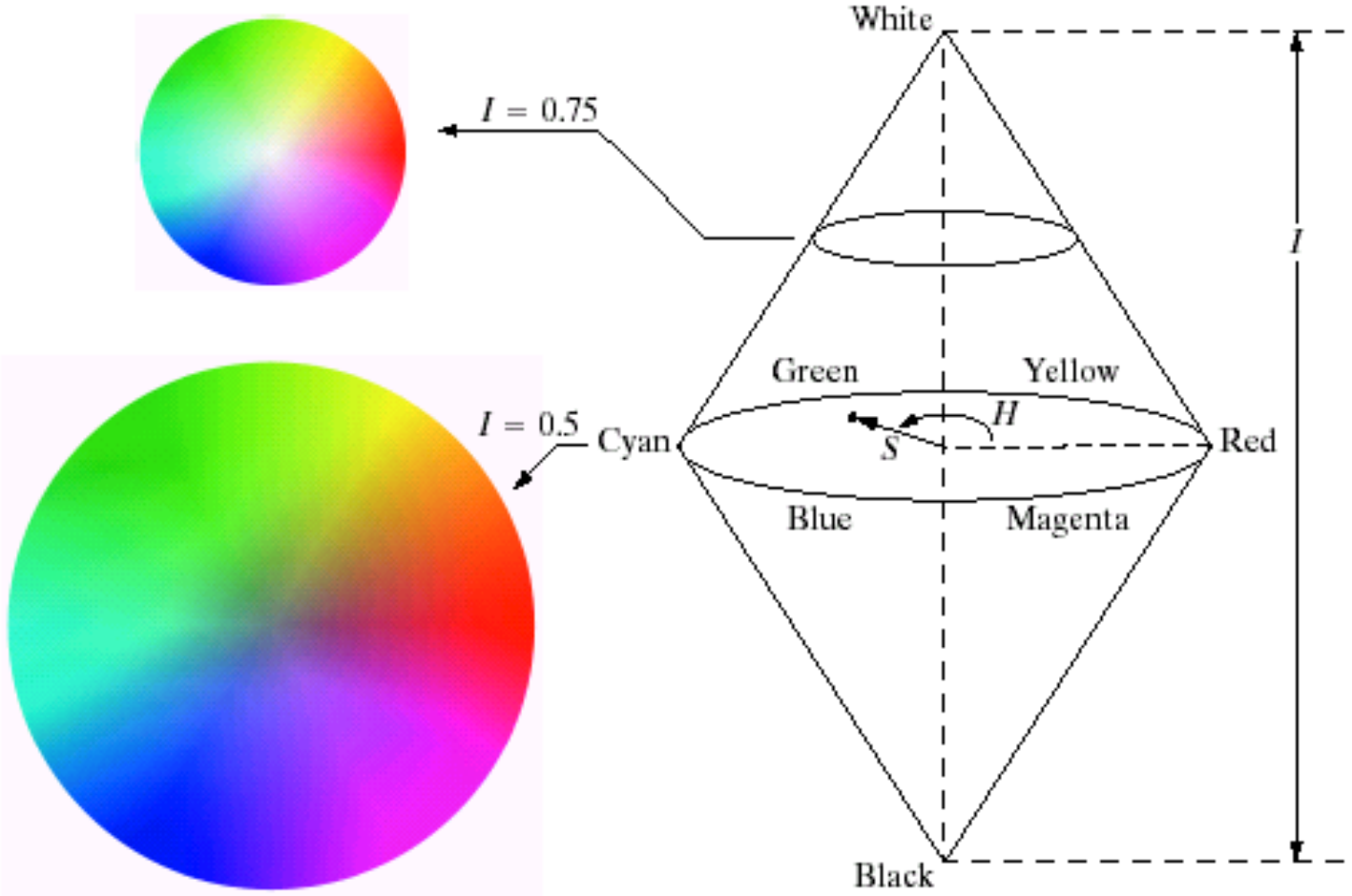
Because the only important things are the angle and the length of the saturation vector this plane is also often represented as a circle or a triangle



HSI Color Model



HSI Color Model



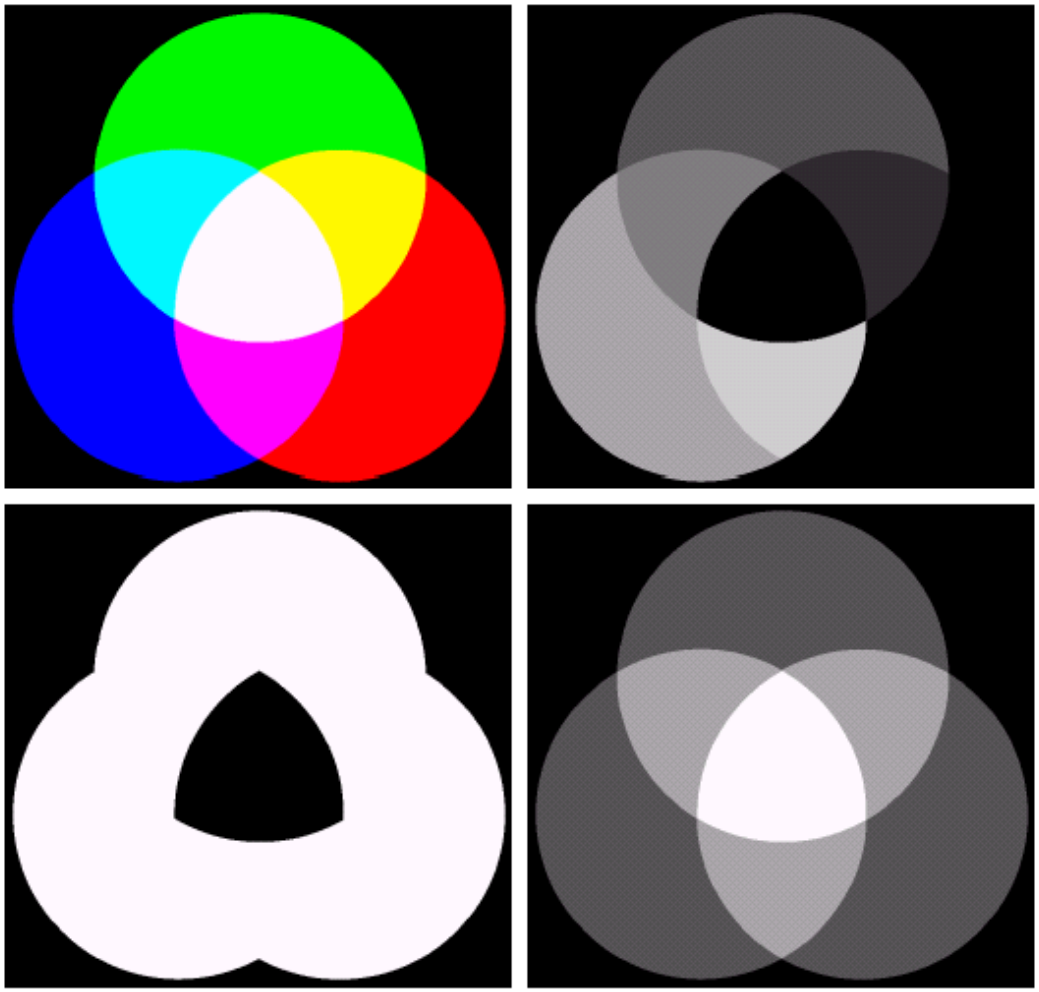
Converting from RGB to HSI

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{\frac{1}{2}}}\right\}$$

$$H = \begin{cases} \theta & \text{if } B \le G \\ 360-\theta & \text{if } B > G \end{cases}$$

$$S = 1 - \frac{3}{(R+G+B)}\left[\min(R,G,B)\right]$$

$$I = \frac{1}{3}(R+G+B)$$



a b
c d

FIGURE 6.16 (a) RGB image and the components of its corresponding HSI image: (b) hue, (c) saturation, and (d) intensity.

COLOR IMAGE - HSI

H-Channel S-Channel I-Channel

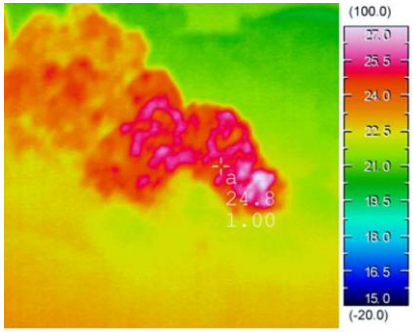
a b c

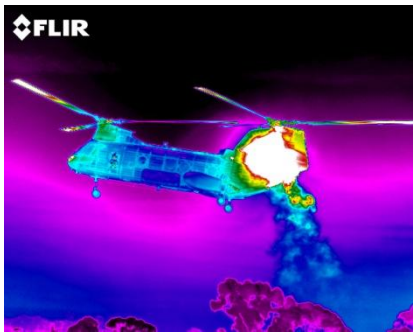
FIGURE 6.39 HSI components of the RGB color image in Fig. 6.38(a). (a) Hue. (b) Saturation. (c) Intensity.

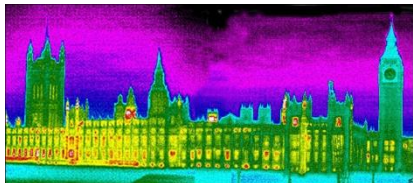
Pseudocolor Image Processing

- Pseudocolor (also called false color) image processing consists of assigning colors to grey values based on a specific criterion

- The principle use of pseudocolor image processing is for human visualization



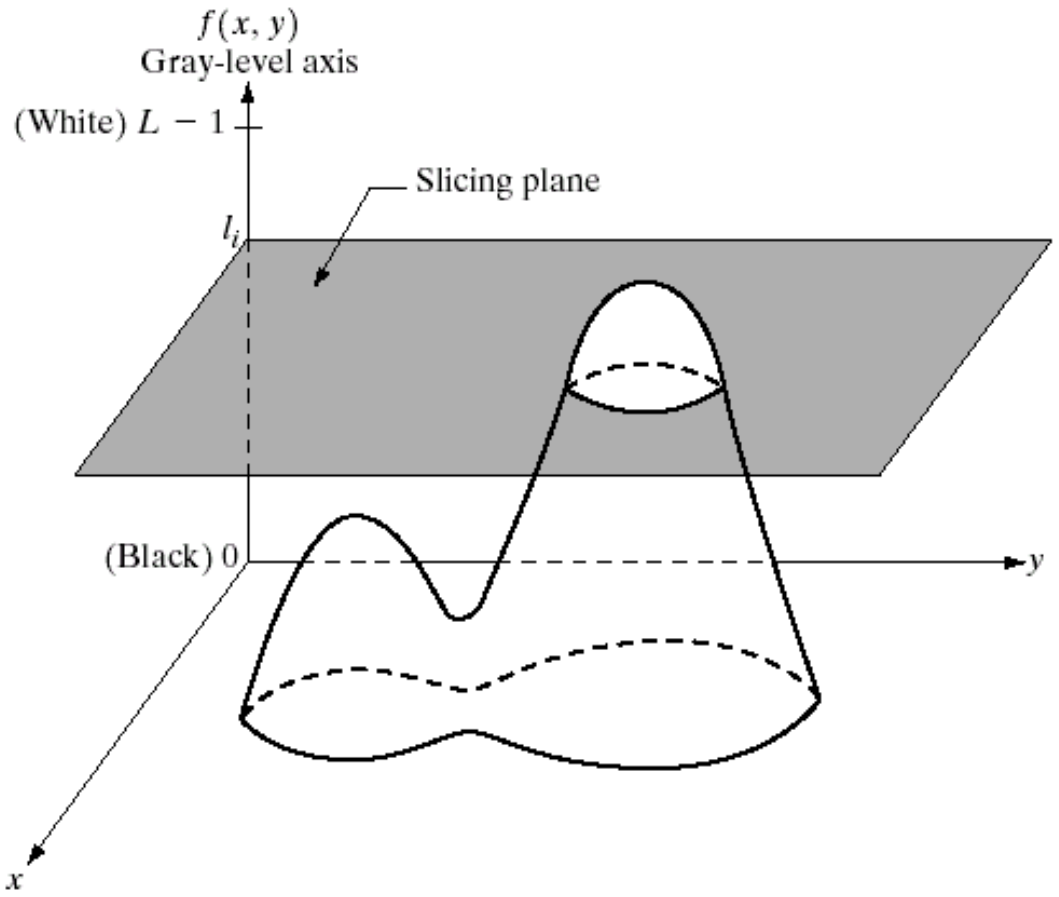




Pseudo Color Image Processing – Intensity Slicing

- Intensity slicing and color coding is one of the simplest kinds of pseudocolor image processing
- First we consider an image as a 3D function mapping spatial coordinates to intensities (that we can consider heights)
- Now consider placing planes at certain levels parallel to the coordinate plane
- If a value is one side of such a plane it is rendered in one color, and a different color if on the other side

Pseudo Color Image Processing – Intensity Slicing



Pseudo Color Image Processing – Intensity Slicing

- **In general intensity slicing can be summarized as:**
  - **Let [*0, L-1*] represent the grey scale**
  - **Let l_0 represent black [$f(x, y) = 0$] and let l_{L-1} represent white [$f(x, y) = L-1$]**
  - **Suppose *P* planes perpendicular to the intensity axis are defined at levels $l_1, l_2, \ldots, l_p$**
  - **Assuming that $0 < P < L-1$ then the *P* planes partition the grey scale into $P + 1$ intervals $V_1, V_2, \ldots, V_{P+1}$**

Pseudo Color Image Processing – Intensity Slicing

- Grey level color assignments can then be made according to the relation:

$$f(x,y) = c_k \qquad \text{if } f(x,y) \in V_k$$

- where c_k is the color associated with the k^{th} intensity level V_k defined by the partitioning planes at $l = k - 1$ and $l = k$

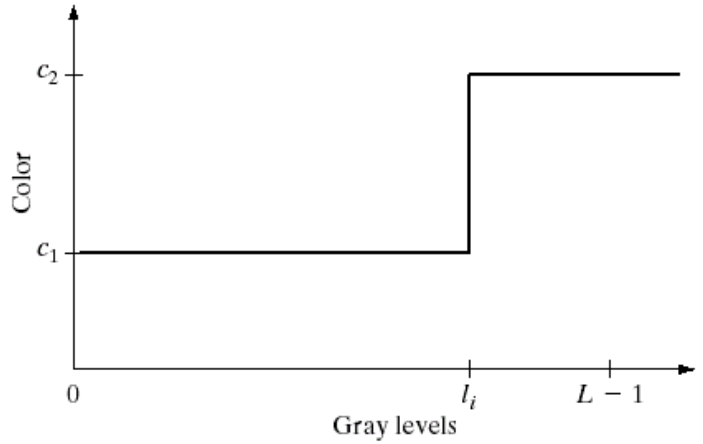
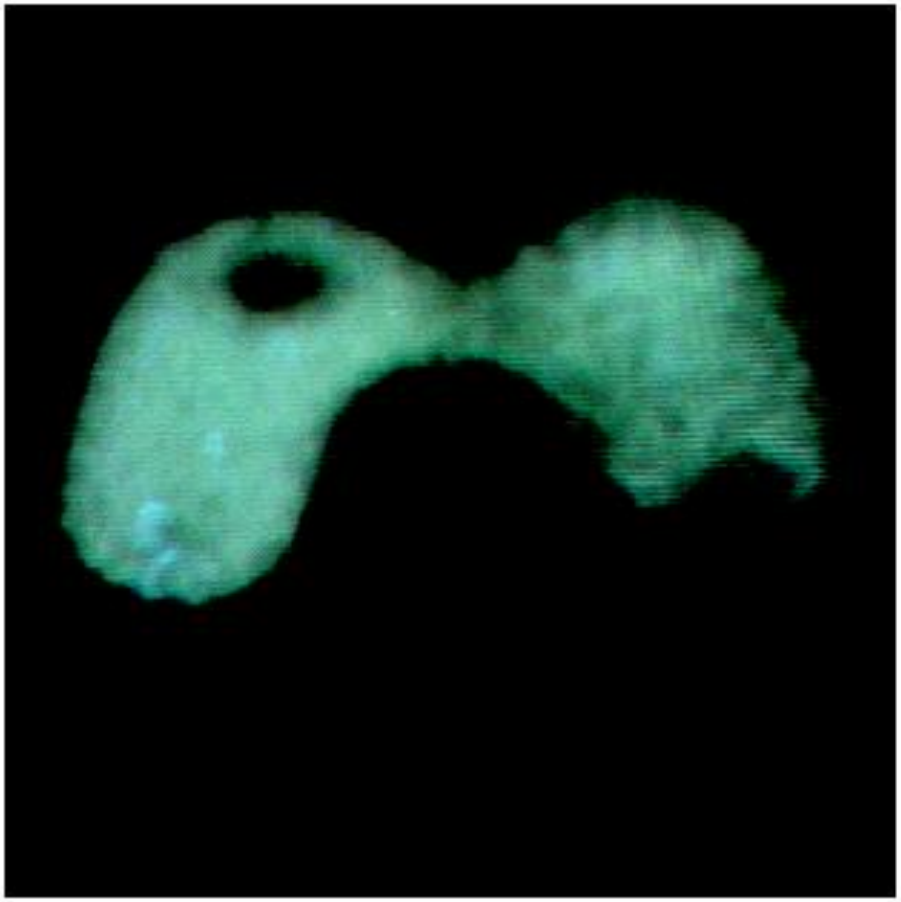
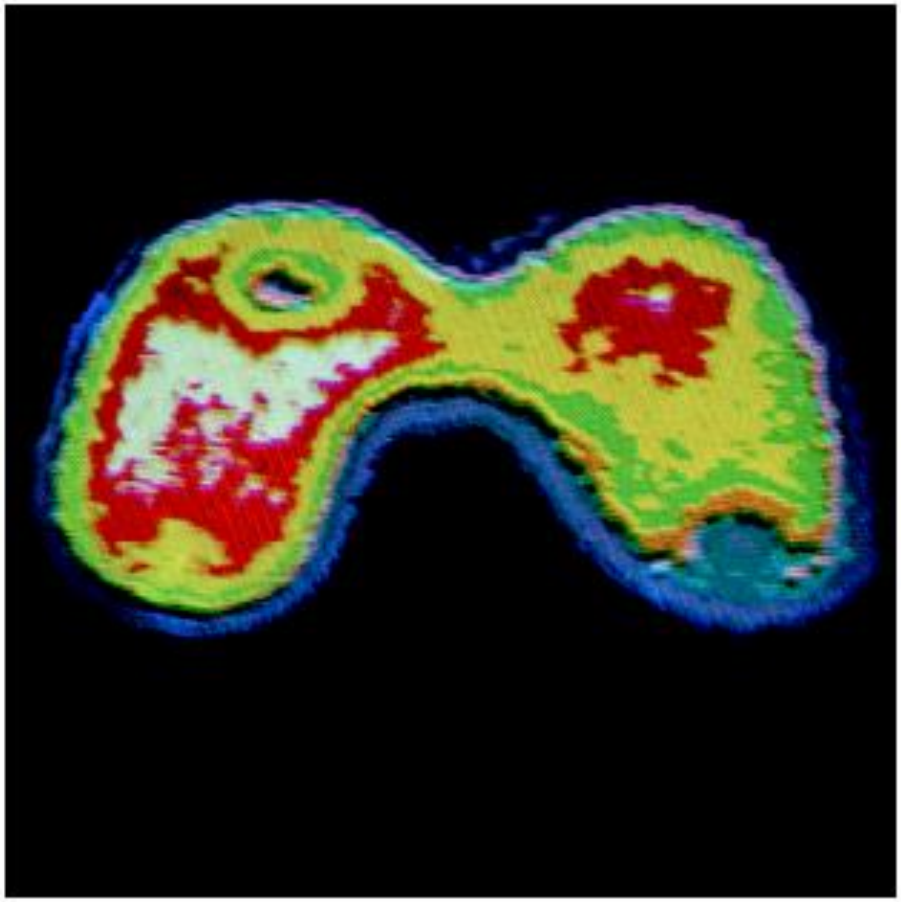


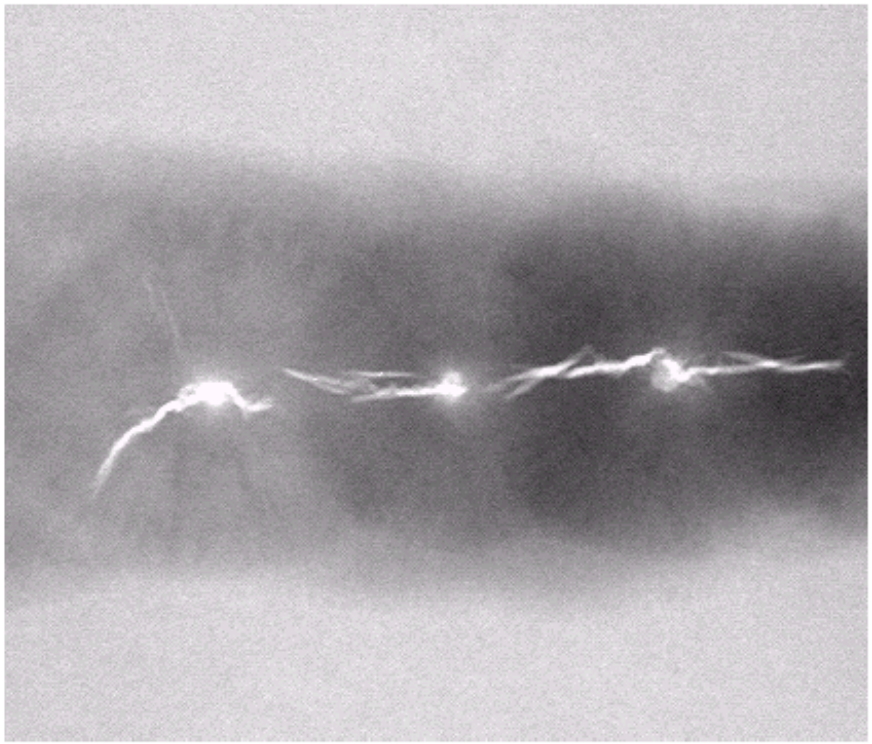
FIGURE 6.19 An alternative representation of the intensity-slicing technique.

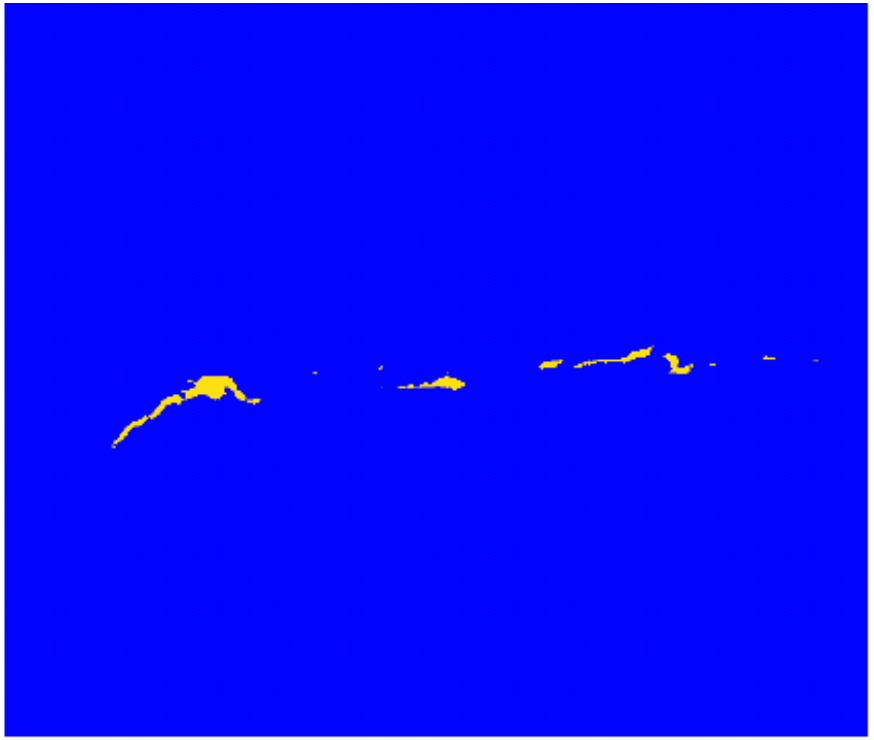
Pseudo Color Image Processing – Intensity Slicing





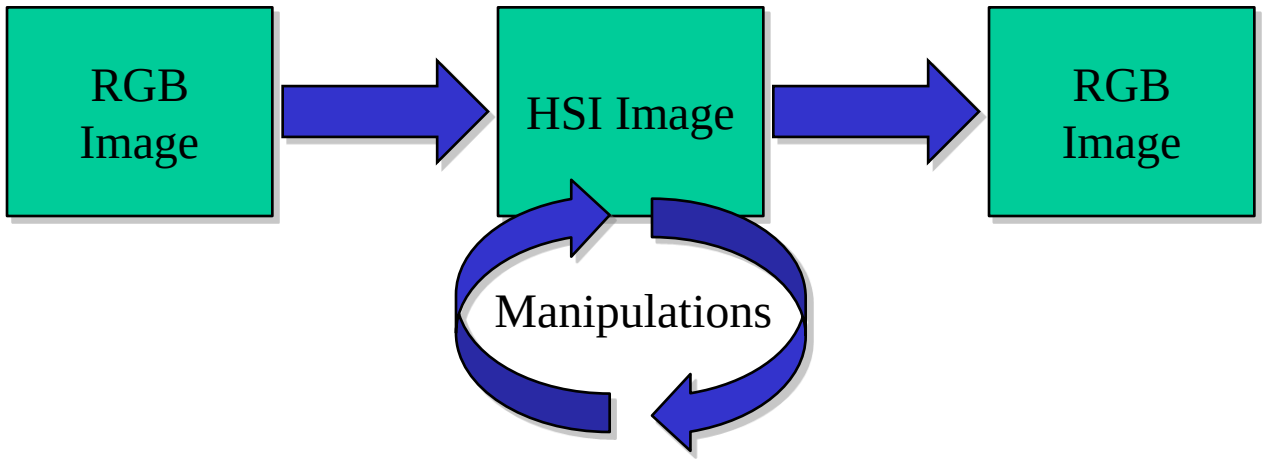
Pseudo Color Image Processing – Intensity Slicing





Manipulating Images In The HSI Model

- In order to manipulate an image under the HSI model we:
  - First convert it from RGB to HSI
  - Perform our manipulations under HSI
  - Finally convert the image back from HSI to RGB



COLOR IMAGE - SMOOTHING

- Smoothing can be viewed as a spatial filtering operation in which the coefficients of the filtering mask are all 1's
- This concept can be easily extended to the processing of full-color images
- Simply smooth each of the RGB color planes and then combine the processed planes to form a smoothed full-color result

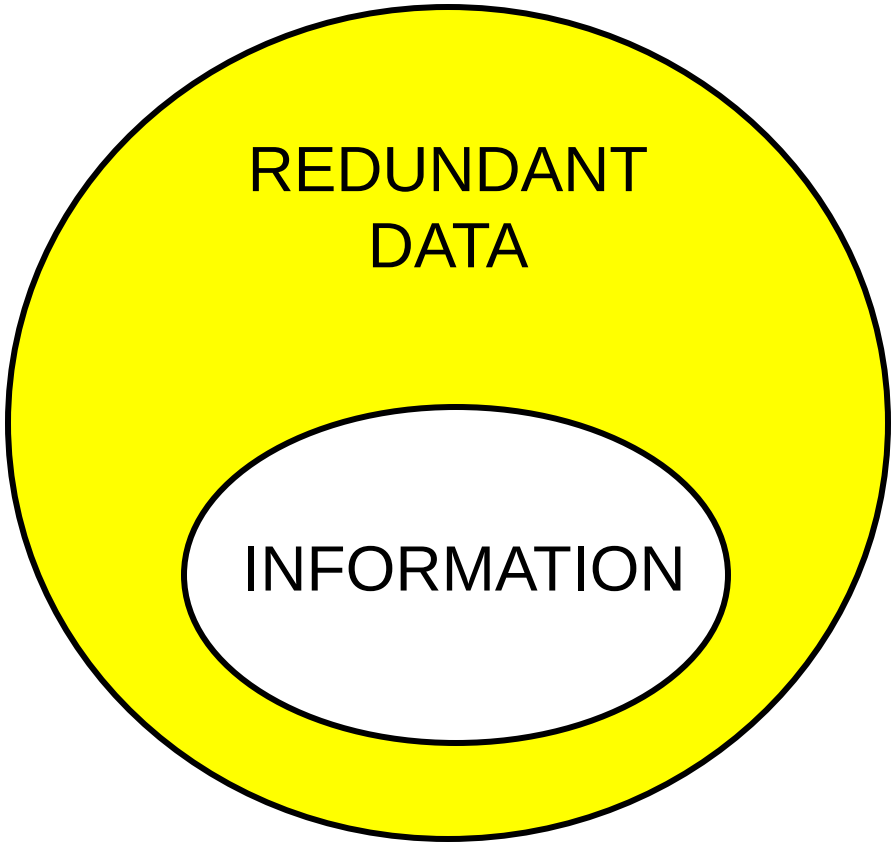
$$\hat{\mathbf{C}}(x,y) = \frac{1}{MN}\begin{bmatrix} \sum_{(x,y)\in S_{xy}} R(x,y) \\ \sum_{(x,y)\in S_{xy}} G(x,y) \\ \sum_{(x,y)\in S_{xy}} B(x,y) \end{bmatrix}$$

IMAGE COMPRESSION

IMAGE COMPRESSION

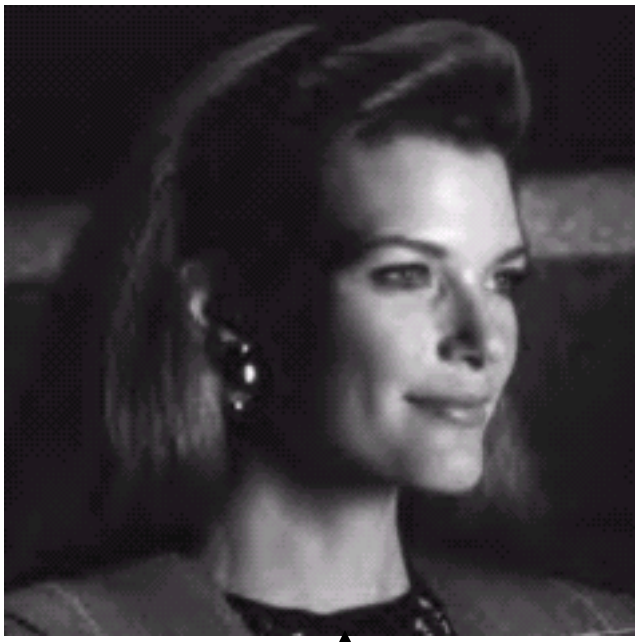
- Addresses the problem of reducing the amount of data required to represent a digital image
- The underlying basis of the reduction process is the removal of redundant data

Information vs Data



DATA = INFORMATION + REDUNDANT DATA

IMAGE COMPRESSION: CODING AND DECODING



original image
262144 Bytes

image encoder

compressed bit stream
0011100001001101…
(2428 Bytes)

image decoder

compression ratio (CR) = 108:1

LOSSY VS LOSSLESS COMPRESSION

- **Lossless** (Information preserving)
  - Images can be compressed and restored without any loss of information.
  - Application: Medical images
- **Lossy**
  - Perfect recovery is not possible but provides a large data compression.
  - Example: TV signals, teleconferencing

FUNDAMENTALS

- Raw image: A set of **n1** bits
- Compressed image: A set of **n2** bits.
- Compression ratio: $C_R = \frac{n_1}{n_2}$
- Relative Data Redundancy of first set: $R_D = 1 - \frac{1}{C_R}$
- Example: *n1*= 100KB and *n2* = 10Kb, then *CR* = 10 , and *RD* = 90%
- Special cases:
  - n1>>n2 → CR ≈ ∞, RD ≈ 1
  - n1 ≈ n2 → CR ≈ 1, RD ≈ 0

DATA REDUNDANCY

- Three basic data redundancies:
  - Coding redundancy
  - Spatial and Temporal redundancy
  - Irrelevant Information

CODING REDUNDANCY

- Type of coding (# of bits for each gray level)
- Image histogram:
  - ***rk***: Represents the gray levels of an image
  - ***pr*(*rk*)**: Probability of occurrence of ***rk***

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0,1,2,\ldots,L-1$$

  - ***l(rk)***: Number of bits used to represent each ***rk*** (after compression)
  - **Lavg**: Average # of bits required to represent each pixel:

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

CODING REDUNDANCY

- It makes sense to assign fewer bits to those rk for which pr(rk) are large in order to reduce the sum.
- This achieves data compression and results in a variable length code.
- More probable gray levels will have fewer # of bits.
- Basic type is variable length coding

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

VARIABLE LENGTH CODING

r_k	$p_r(r_k)$	code1	$l_1(r_k)$	code2	$l_2(r_k)$
$r_0=0$	0.19	000	3	11	2
$r_1=1/7$	0.25	001	3	01	2
$r_2=2/7$	0.21	010	3	10	2
$r_3=3/7$	0.16	011	3	001	3
$r_4=4/7$	0.08	100	3	0001	4
$r_5=5/7$	0.06	101	3	00001	5
$r_6=6/7$	0.03	110	3	000001	6
$r_7=1$	0.02	111	3	000000	6

VARIABLE LENGTH CODING

- **Computing L_{avg}**

$$L_{avg} = \sum_{k=0}^{7} l_2(r_k)\, p_r(r_k)$$

=2(0.19)+2(0.05)+2(0.21)+3(0.16)+4(0.08)+5(0.06)+6(0.03)+6(0.02)

=2.7 bits

- **C_R=3/2.7=1.11**
- **R_D=1-1/1.11=0.099=9.9%**

IMAGE COMPRESSION MODEL

- **Encoder:** Create a set of symbols from image data.
  - **Source Encoder:** Reduce input redundancy.
  - **Channel Encoder**: Increase noise resistance
- **Channel:** Transmission path.
- **Decoder:** Construct output image from transmitted symbols
  - **Source Decoder**: Reverse of Source encoder
  - **Channel Decoder**: Error detection/correction.

IMAGE COMPRESSION MODEL

- **Encoder:** Create a set of symbols from image date.
  - **Source Encoder:** Reduce input redundancy.
  - **Channel Encoder**: Increase noise immunity (noise robustness)
- **Channel:** Transmission path.
- **Decoder:** Construct output image from transmitted symbols
  - **Source Decoder**: Reverse of Source encoder
  - **Channel Decoder**: Error detection/correction.

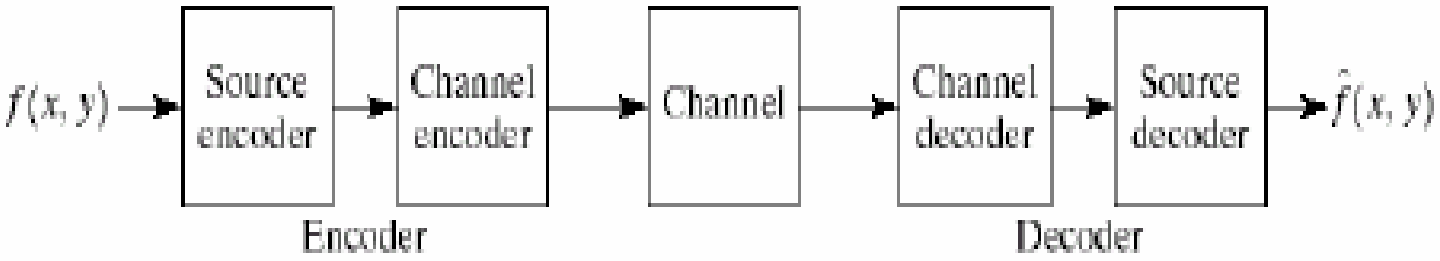
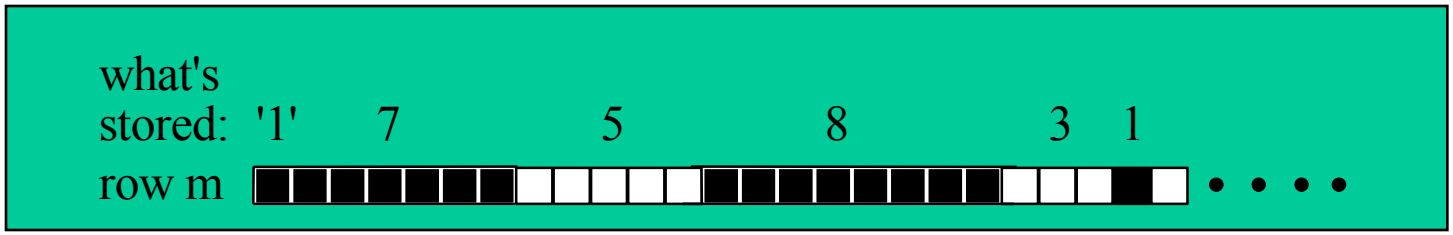


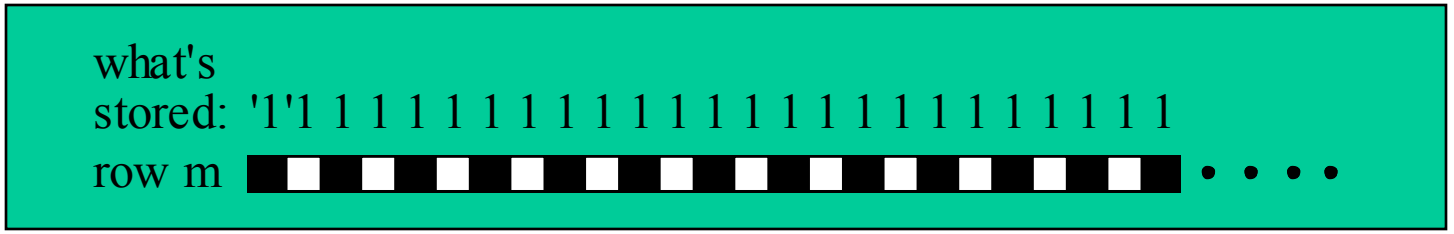
FIGURE 8.5 A general compression system model.

RUN LENGTH CODING

- Run Length
  - **The length of consecutively identical symbols**
- Run length Coding Example



- When Does it Work?
  - **Images containing many runs of 1's and 0's**
- When Does it Not Work?



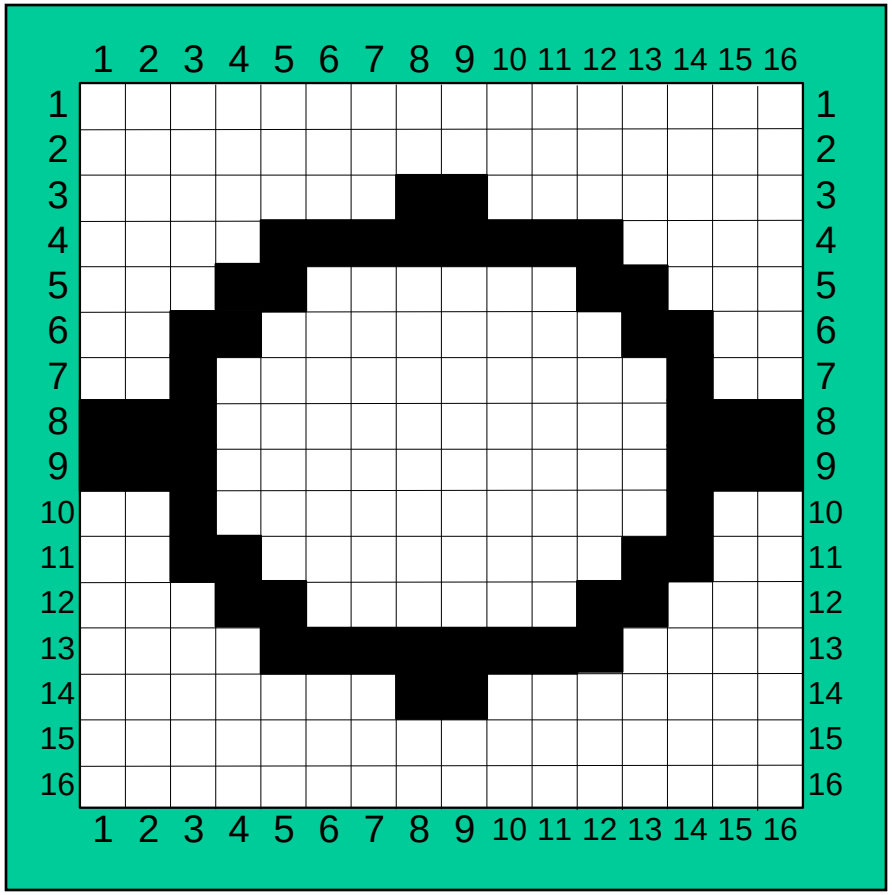
RUN LENGTH CODING

- Decoding Example

A binary image is encoded using run length code row by row, where "0" represents white, and "1" represents black. The code is given by

Row 1: "0", 16
Row 2: "0", 16
Row 3: "0", 7, 2, 7
Row 4: "0", 4, 8, 4
Row 5: "0", 3, 2, 6, 3, 2
Row 6: "0", 2, 2, 8, 2, 2
Row 7: "0", 2, 1, 10, 1, 2
Row 8: "1", 3, 10, 3
Row 9: "1", 3, 10, 3
Row 10: "0", 2, 1, 10, 1, 2
Row 11: "0", 2, 2, 8, 2, 2
Row 12: "0", 3, 2, 6, 3, 2
Row 13: "0", 4, 8, 4
Row 14: "0", 7, 2, 7
Row 15: "0", 16
Row 16: "0", 16

decode



LOSSLESS GRAY-SCALE IMAGE COMPRESSION

VARIABLE WORD LENGTH CODING: EXAMPLE

- **A 4x4 4bits/pixel original image is given by**

Default Code Book

0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101
6: 0110
7: 0111
8: 1000
9: 1001
10: 1010
11: 1011
12: 1100
13: 1101
14: 1110
15: 1111

2	8	6	6
6	8	8	8
8	8	10	10
9	10	10	14

encode

0010	1000	0110	0110
0110	1000	1000	1000
1000	1000	1010	1010
1001	1010	1010	1110

Bit rate = 4bits/pixel

Total # of bits used to represent the image:

4x16 = 64 bits

VARIABLE WORD LENGTH CODING: EXAMPLE

- **Encode the original image with a CODE BOOK given left**

Huffman Code Book

0: 0000000
1: 0000001
2: 0001
3: 0000010
4: 0000011
5: 0000100
6: 01
7: 0000101
8: 10
9: 00100
10: 11
11: 0000110
12: 0000111
13: 001010
14: 0011
15: 001011

2	8	6	6
6	8	8	8
8	8	10	10
9	10	10	14

encode

0001	10	01	01
01	10	10	10
10	10	11	11
00100	11	11	0011

Total # of bits used to represent the image:

4+2+2+2+2+2+2+2+2+
2+2+2+5+2+2+4
= 39 bits

Bit rate = 39/16
= 2.4375 bits/pixel

CR = 64/39 = 1.6410

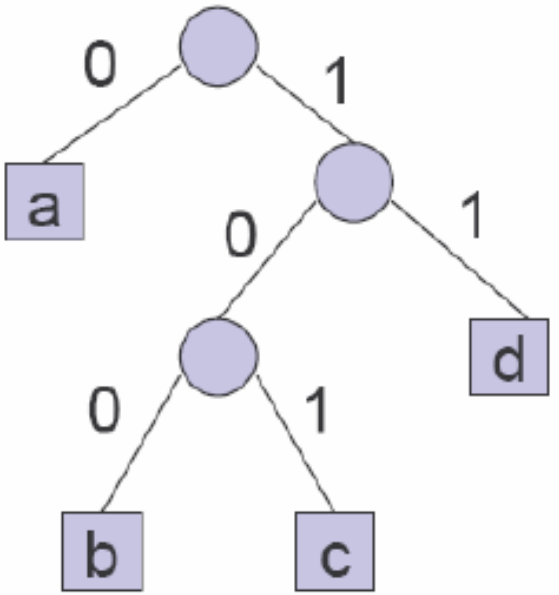
HUFFMAN CODING

- Uses frequencies (Probability) of symbols in a string to build a variable rate prefix code.
- Each symbol is mapped to a binary string.
- More frequent symbols have shorter codes.
- No code is a prefix of another. (Uniquely decodable)

HUFFMAN CODING

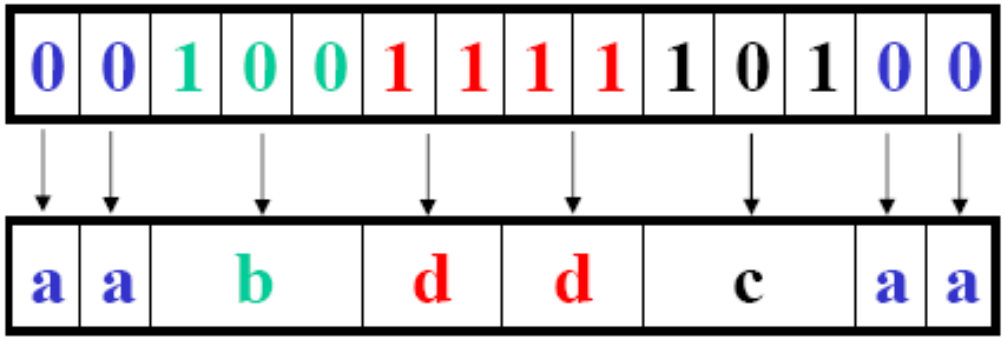
- Example:
  - We have four symbols a, b, c, and d.

a 0
b 100
c 101
d 11



HUFFMAN CODING

- A source string: **aabddcaa**
  - Fixed Length Coding: 16 bits (ordinary coding)
    - **00 00 01 11 11 10 00 00**
- Variable length coding: 14 bits (Huffman coding)
  - **0 0 100 11 11 101 0 0**
- Uniquely Decodable:



HUFFMAN CODING

- Step-1
  - Arrange probability in decreasing order and consider them as tree leaves
- Step-2
  - Merge two nodes with smallest prob. to a new node and sum up prob.
  - Arbitrarily assign 1 and 0 to each pair of merging branch
- Step-3
  - Repeat until no more than one node left.
  - Read out codeword sequentially from root to leaf

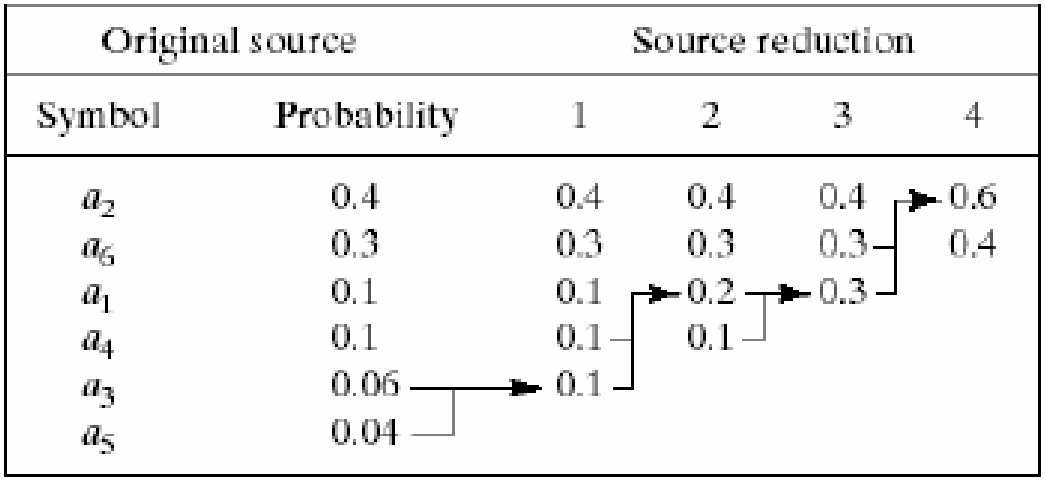
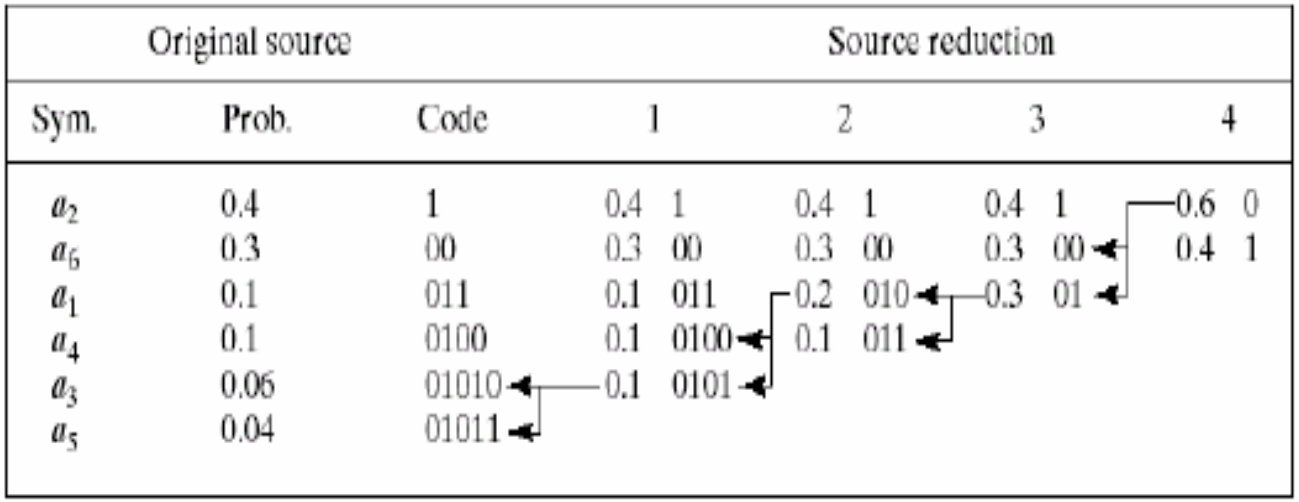


FIGURE 8.11 Huffman source reductions.

HUFFMAN CODING

FIGURE 8.12 Huffman code assignment procedure.



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										

$$L_{avg} = 2.2 \text{ bits/symbol}$$

Example Huffman encoding

- A = 0
  B = 100
  C = 1010
  D = 1011
  R = 11
- ABRACADABRA = 01001101010010110100110
- This is eleven letters in 23 bits
- A fixed-width encoding would require 3 bits for five different letters, or 33 bits for 11 letters
- Notice that the encoded bit string *can* be decoded!

Why it works

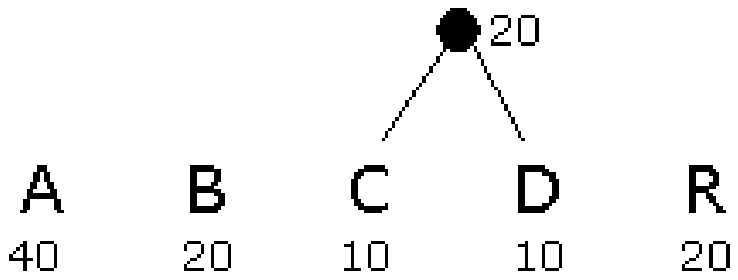
- In this example, A was the most common letter
- In ABRACADABRA:
  - 5 As code for **A** is 1 bit long
  - 2 Rs code for **R** is 2 bits long
  - 2 Bs code for **B** is 3 bits long
  - 1 C code for **C** is 4 bits long
  - 1 D code for **D** is 4 bits long

Creating a Huffman encoding

- For each encoding unit (letter, in this example), associate a frequency (number of times it occurs)
  - You can also use a percentage or a probability
- Create a binary tree whose children are the encoding units with the smallest frequencies
  - The frequency of the root is the sum of the frequencies of the leaves
- Repeat this procedure until all the encoding units are in the binary tree

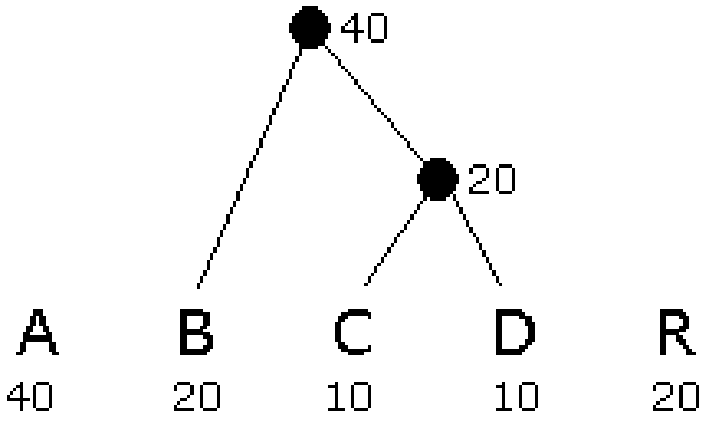
Example, step I

- Assume that relative frequencies are:
  - **A**: 40
  - **B**: 20
  - **C**: 10
  - **D**: 10
  - **R**: 20
- (I chose simpler numbers than the real frequencies)
- Smallest number are 10 and 10 (C and D), so connect those



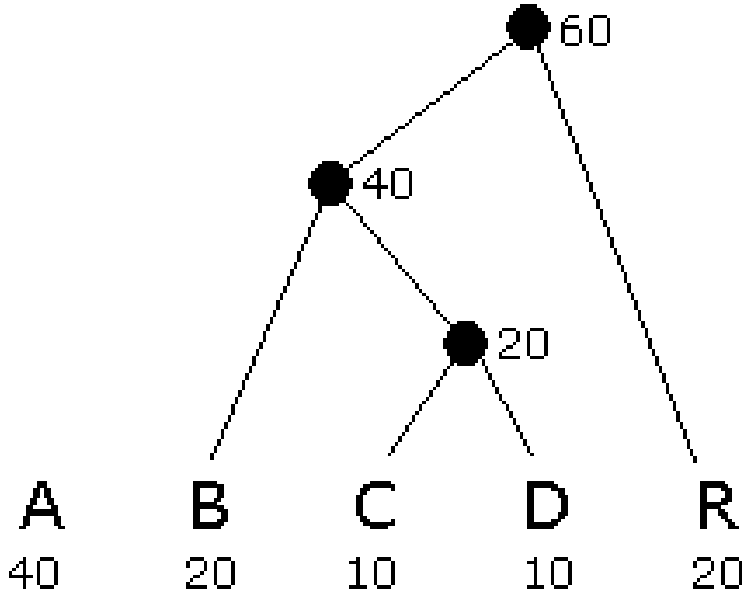
Example, step II

- C and D have already been used, and the new node above them (call it C+D) has value 20
- The smallest values are B, C+D, and R, all of which have value 20
  - Connect any two of these



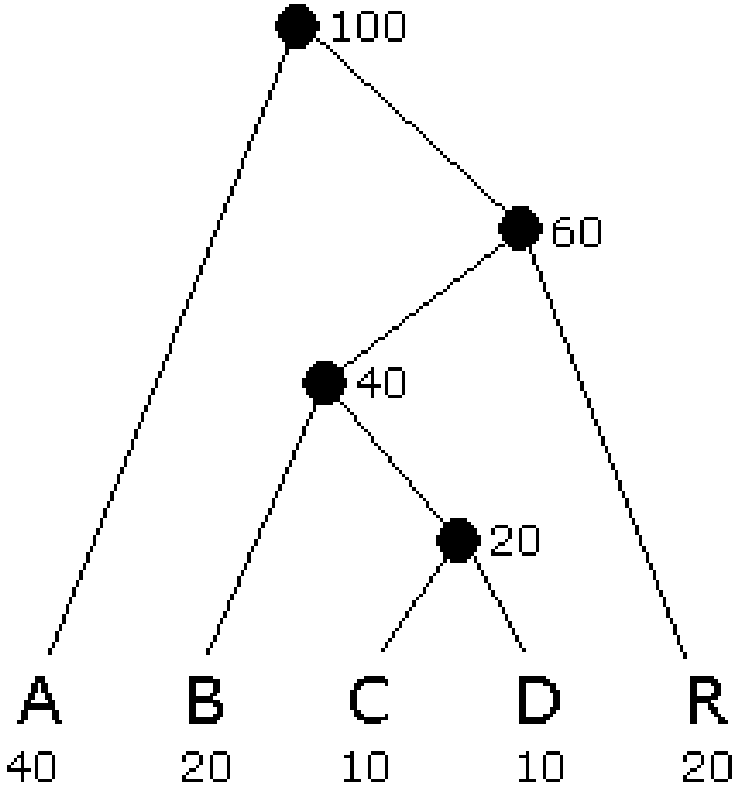
Example, step III

- The smallest values is R, while A and B+C+D all have value 40
- Connect R to either of the others



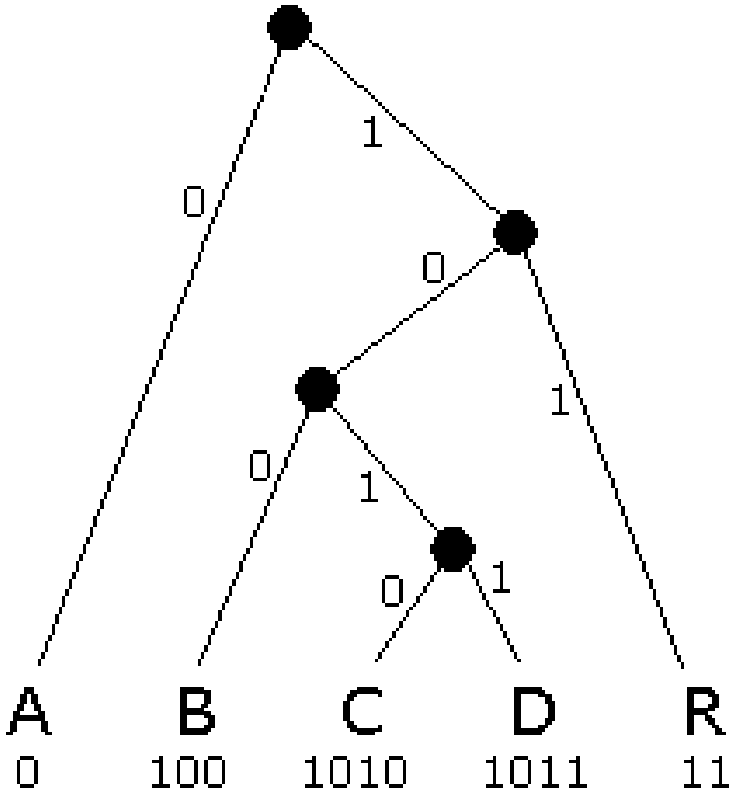
Example, step IV

- Connect the final two nodes



Example, step V

- Assign 0 to left branches, 1 to right branches
- Each encoding is a path from the root



- A = 0
  B = 100
  C = 1010
  D = 1011
  R = 11
- Each path terminates at a leaf
- Do you see why encoded strings are decodable?

OBJECTIVE CRITERIA

- **The error e(x,y) can be defined as**

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

- **Total error between the two images**

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]$$

- **Root-mean-square-error**

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

- **Mean-square signal-to-noise ratio of the output image**

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