جامعة اللاذقية

كلية الهندسة المعلوماتية

قسم البرمجيات ونظم المعلومات

Multimedia Systems

Graphics and Image Data Representation

Lecture 2

Image Data Representation

Basic Data Types

1-Bit Images

8-Bit Grey-Level Images

Halftoning and Dithering

24-Bit Color Images

8-Bit Color Images

Color Lookup Table (LUTs)

How to Devise a Color Lookup Table

Image Data Representation

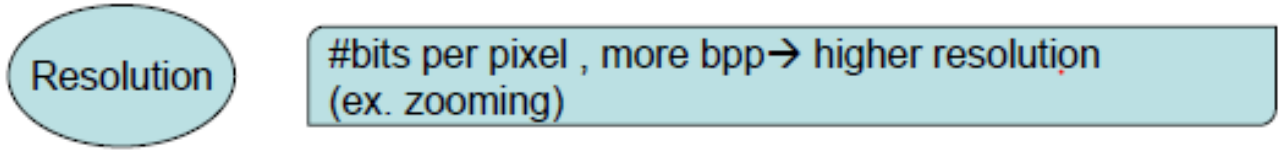
Pixel:

picture element in digital image

Image resolution:

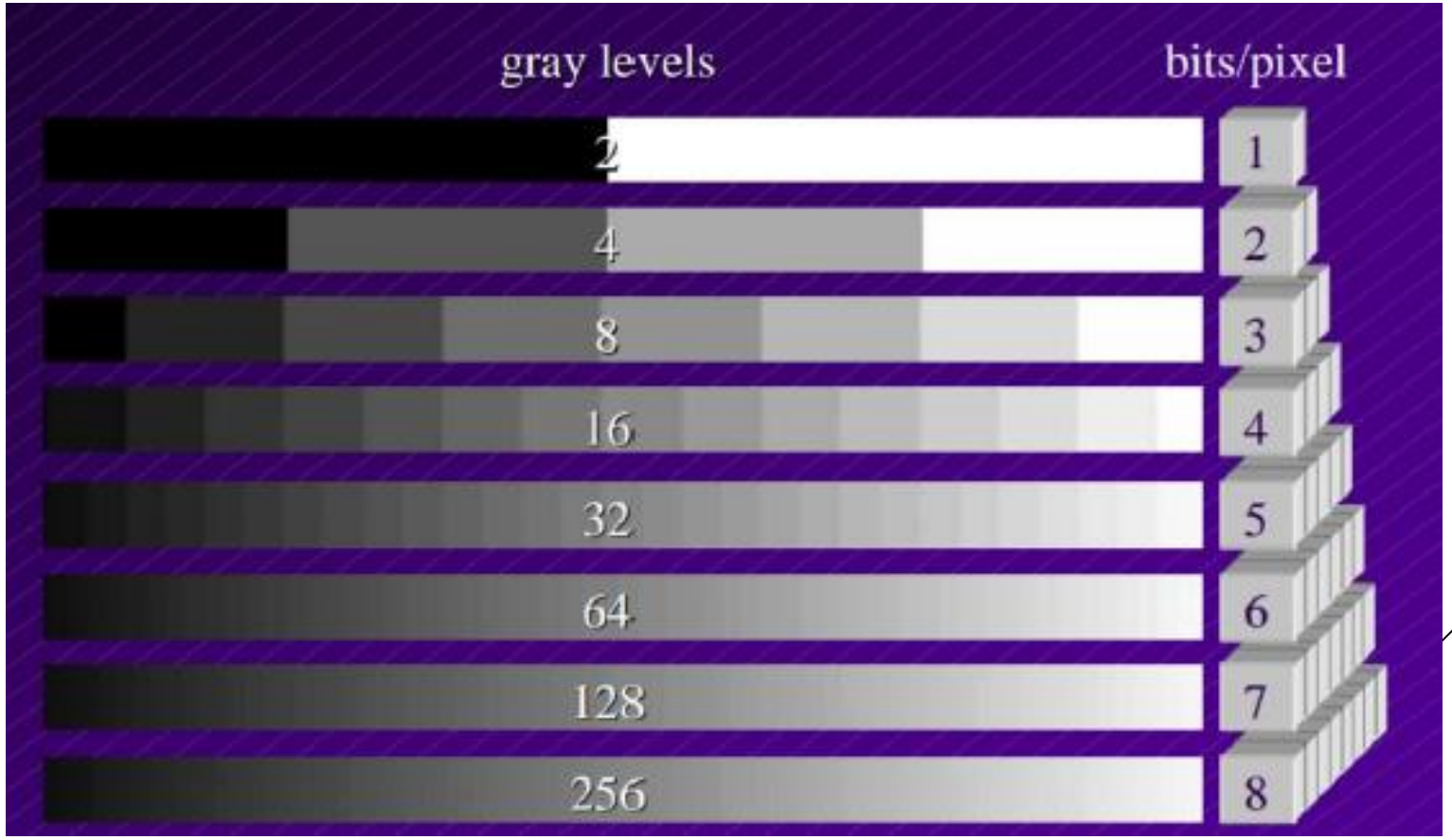
refers to the number of pixels in a digital image (higher resolution always yields better quality)ex. 1600*1200

R=width * height (spatial resolution)



If we want an image that has more than two gray levels, we have to increase bpp (intensity resolution)

Image Data Representation



1-bit Image

Also called Binary Image or Monochrome Image

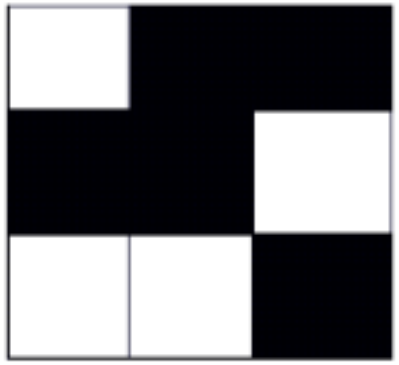
1-Bit Image Examples

1-bit Image: Features

Consist of on and off pixels (pixels—picture elements in digital images)

Each pixel is stored as a single bit (0 or 1),

0--black, 1--white



$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

1-bit Image: Size & Usage

Storage

Resolution

is the number of points (pixels) that can be plotted horizontally and vertically.

it is often simply stated as the total number of points in each direction

– Monochrome image with resolution: 640×480

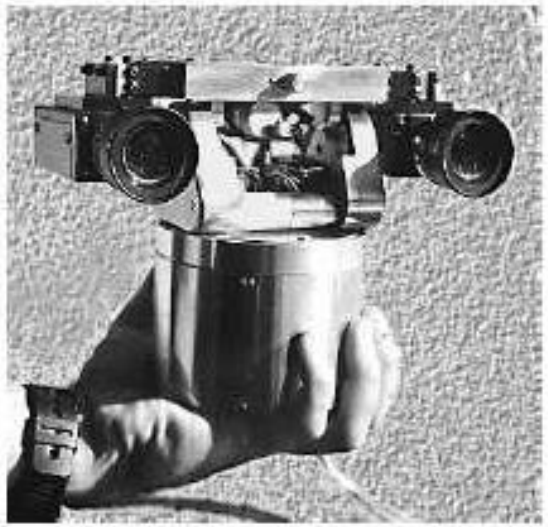
– 640×480/8 bytes

– Storing space needed: 38.4KB

Usage

– Pictures containing only simple graphics and text

8-Bit Gray-level Image



8-Bit Gray-Level Image Examples

8-Bit Gray-level Image

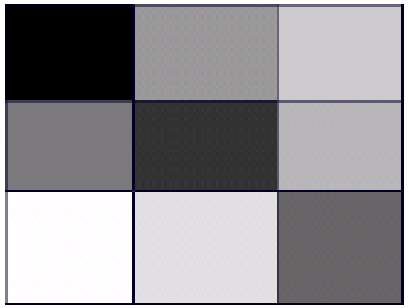
8-Bit Gray-Level Image VS 1-Bit Image

8-Bit Gray-level Image: Features

Each pixel is represented by a single byte

– A gray-level value between 0 and 255

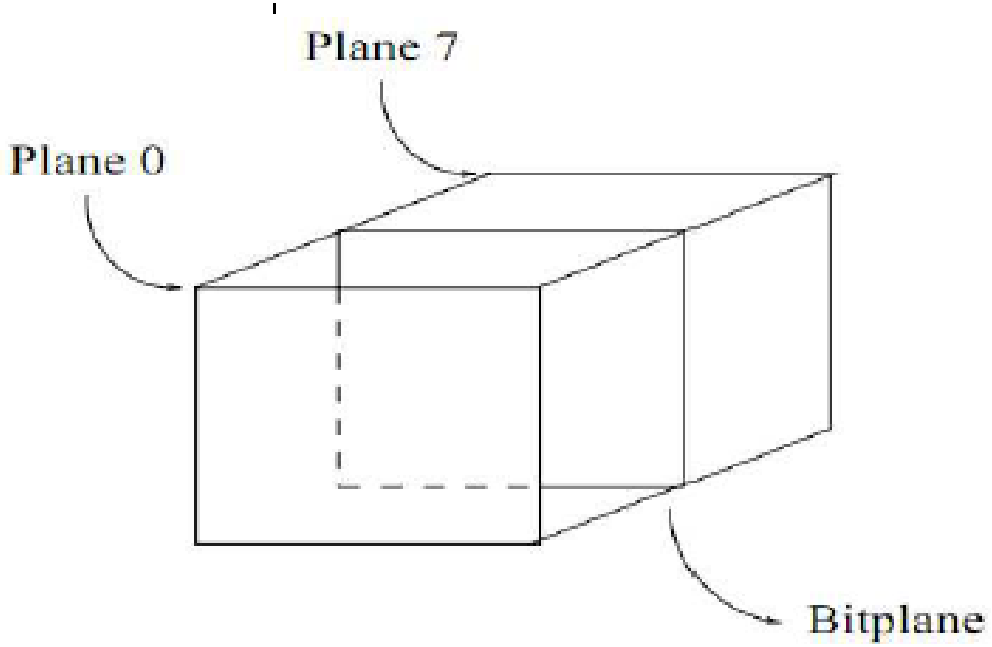
- The entire image can be thought of as a two dimensional array of pixel values



$$I = \begin{bmatrix} 0 & 150 & 200 \\ 120 & 50 & 180 \\ 250 & 220 & 100 \end{bmatrix}$$

8-Bit Gray-level Image: Features

- 8-Bit image as a set of 1-bit bitplanes
- Each plane consists of a 1 bit representation of the image at one level
- All the bitplanes make up a single byte that stores the value between 0 ~ 255



8-Bit Gray-level Image: Size

- Resolution

– High : 1600×1200

– Low : 640×480

– Aspect Ratio : ??

4:3

Aspect ratio: the ratio of horizontal to vertical pixels.

- The space needed by a 640×480 grey image

– 640×480=307,200 bytes

- Hardware storing Image Array

– frame buffer / "Video" card

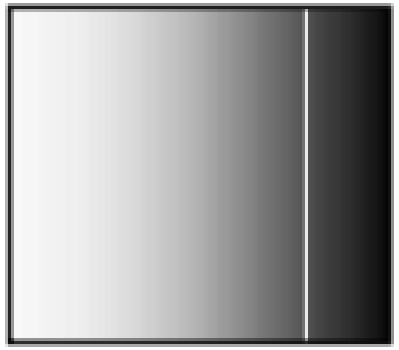
8-Bit Gray-level Image: Print

How to print an 8-bit gray-level image on 2-level (1-bit) printer?

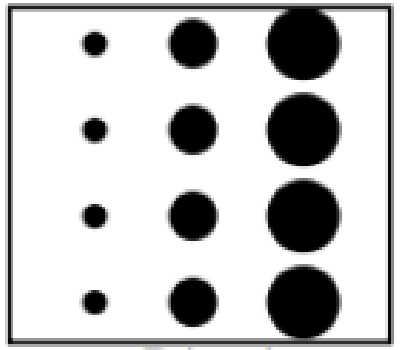
Such a device can usually only print a dot or not print it. Two approaches are presented. These are halftoning and dithering.

1) Halftoning

- Using dots of varying size to represent intensities
- area of dots is proportional to intensity in image
- Used for printing newspapers and photos.



I(x,y)



P(x,y)

8-Bit Gray-level Image: Print

Newspaper Image

From New York Times, 9/21/99

8-Bit Gray-level Image: Print

How to print an 8-bit gray-level image on 2-level (1-bit) printer?

2) Dithering

– The main strategy is to replace a pixel value by a larger pattern, say 2 x 2 or 4 x 4, such that the number of printed dots approximates the varying-sized disks of ink used in analogue, in halftone printing.

– Convert the color resolution into the spatial resolution.

- An N×N matrix represents levels N^2+1 of intensity

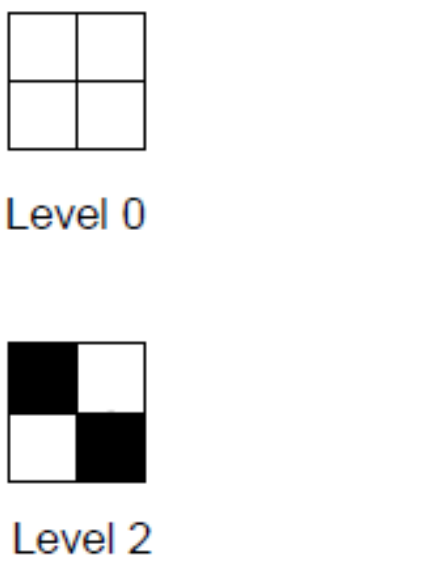
8-Bit Gray-level Image: Print

How to print an 8-bit gray-level image on 2-level (1-bit) printer?

2) Dithering

2×2 pattern can represent 2^2 +1 levels

$$\begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix}$$



8-Bit Gray-level Image: Print

2) Dithering

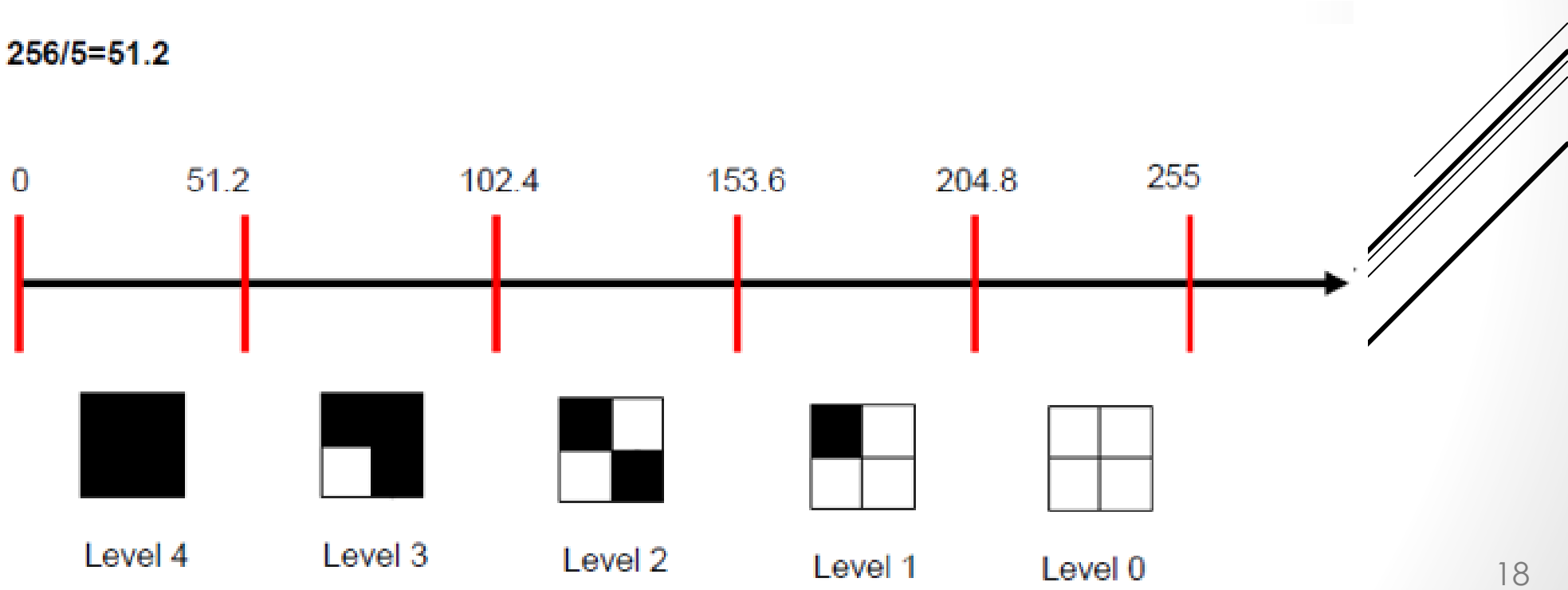
- we can first re-map image values in 0..255 into the new range [0-4] by (integer) dividing by 256/5.

- if the pixel value is **4** we print nothing, in a 2×2 area of printer output. But if the pixel value is **0** we print all four dots.

- If the intensity is > the dither matrix entry then print an on dot at that entry location: replace each pixel by an n x n matrix of dots.

8-Bit Gray-level Image: Print

2) Dithering

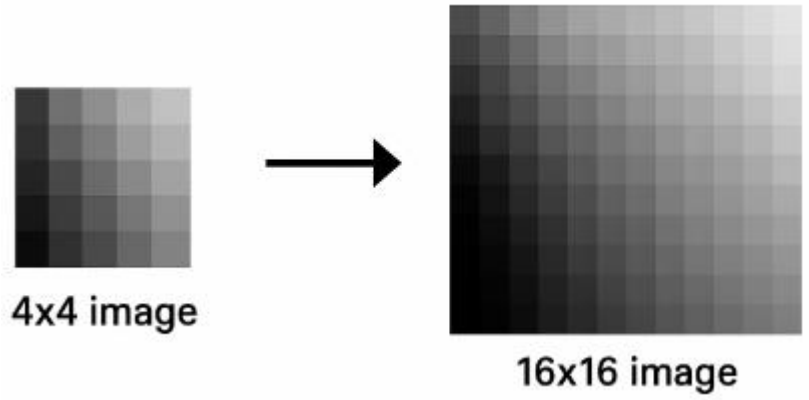


8-Bit Gray-level Image: Print

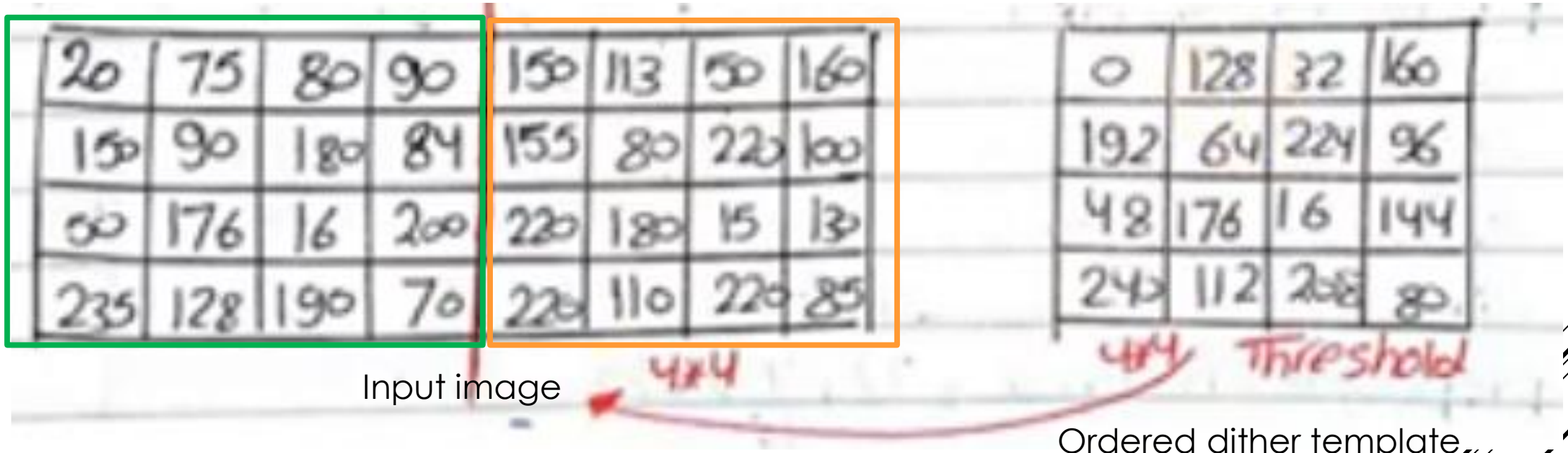
2) Ordered Dither

- If we increase the number of effective intensity levels by increasing the dither matrix size, we also increase the size of the output image.
- This reduces the amount of detail in any small part of the image, effectively reducing the spatial resolution.
- If one pixel uses 4×4 pattern, the size of an N×N image becomes **4N×4N**, makes an image **16** times as large!.

Solution ➔ ordered dither

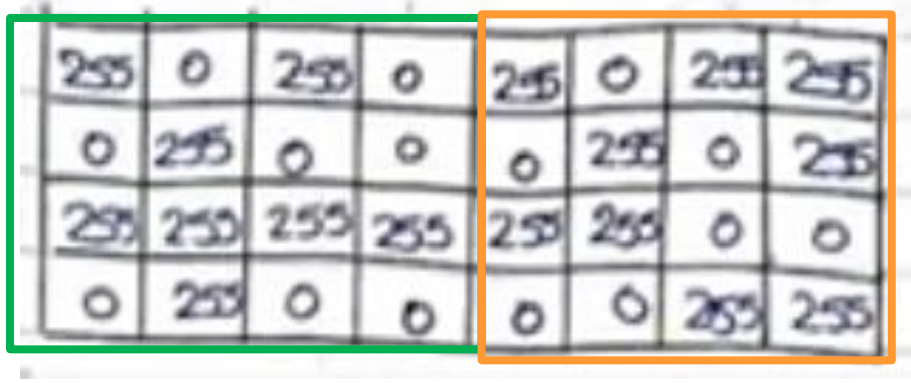


8-Bit Gray-level Image: Print



Input image

Ordered dither template



Output image

8-Bit Gray-level Image: Print

2) Ordered Dither

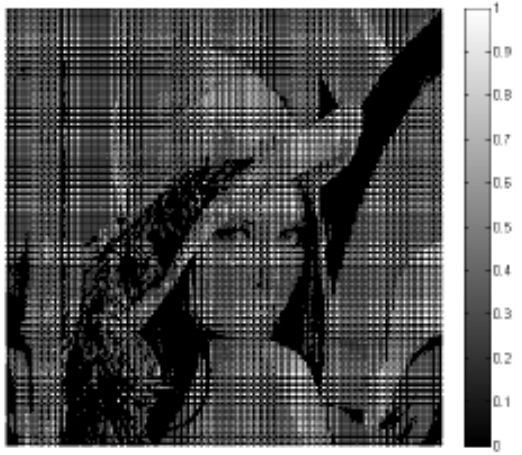
- An algorithm for ordered dither, with n x n dither matrix, is as follows:

```
begin
    for x = 0 to x_max                    // columns
        for y = 0 to y_max                // rows
            i = x mod n
            j = y mod n
            // I(x, y) is the input, O(x, y) is the output, D is the dither matrix.
            if I(x, y) >= D(i, j)
                O(x, y) = 255 ;
            else
                O(x, y) = 0;
end
```

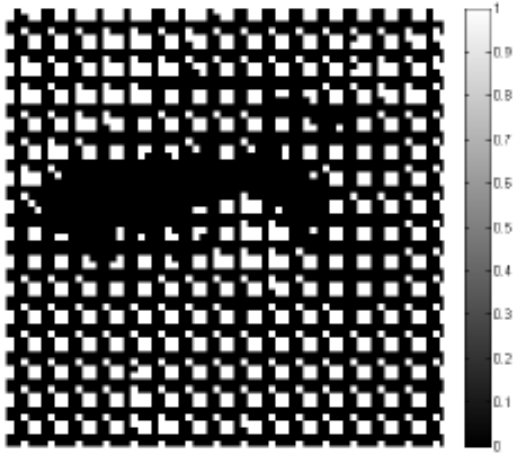
8-Bit Gray-level Image: Print

2) Ordered Dither

(a)



(b)



(c)

(a): 8-bit grey image “lena512.bmp”. (b): Dithered version of the image.

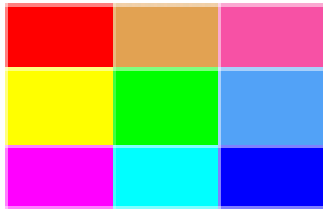
24-Bit Color Image: Feature

Each pixel using **three** bytes: representing RGB

– Value from 0 to 255 ;

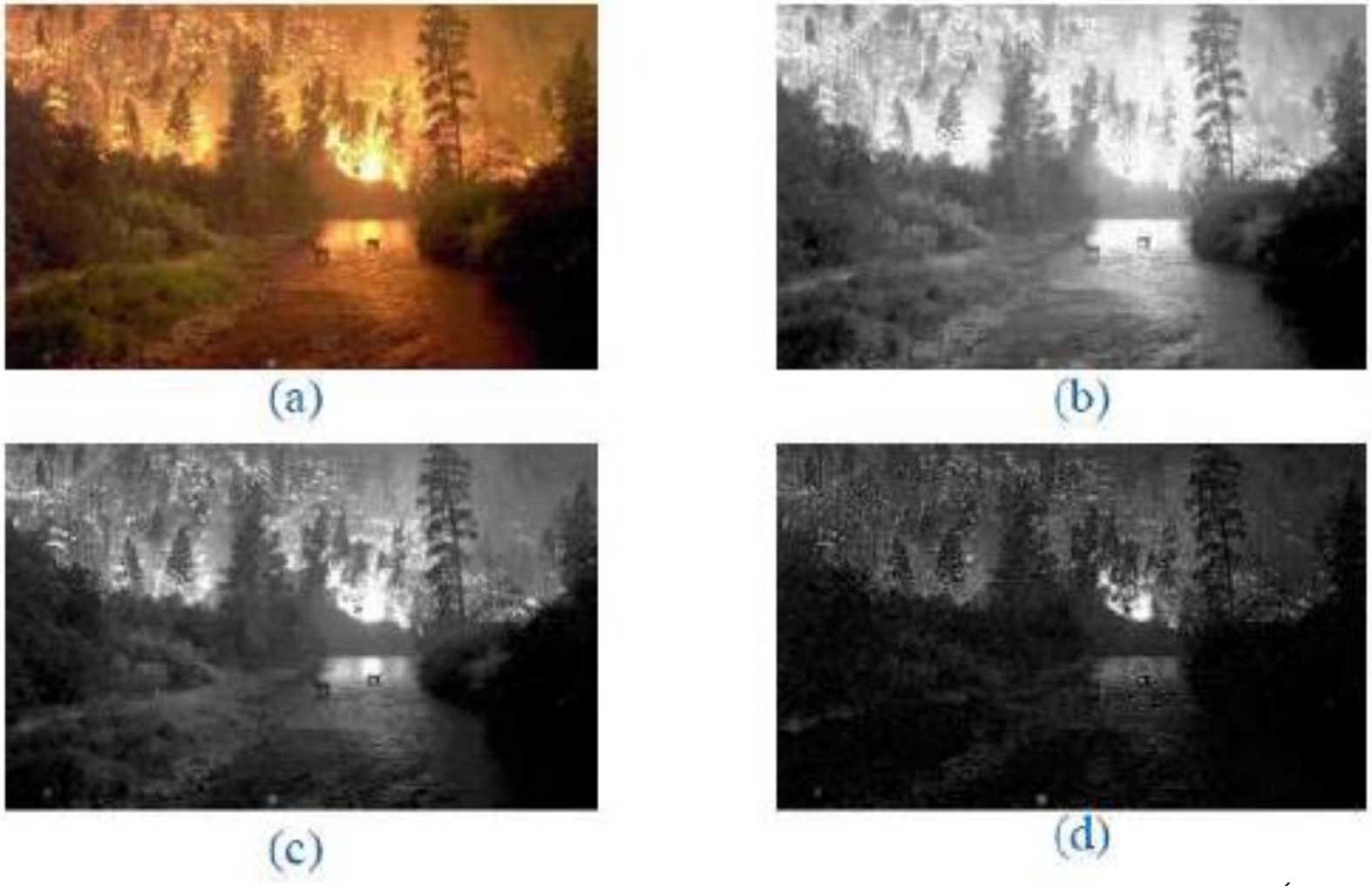
– Supports 256×256×256 colors, 16,777,216

- Each pixel described by different grey values of RGB
- 24-Bit Color image, 640×480×3 bytes ➔ 921.6KB



$$R = \begin{bmatrix} 255 & 240 & 240 \\ 255 & 0 & 80 \\ 255 & 0 & 0 \end{bmatrix} \quad G = \begin{bmatrix} 0 & 160 & 80 \\ 255 & 255 & 160 \\ 0 & 255 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 80 & 160 \\ 0 & 0 & 240 \\ 255 & 255 & 255 \end{bmatrix}$$

24-Bit Color Image



High-resolution color and separate R, G, B color channel images. (a): Example of 24-bit color image. (b, c, d) R, G, and B color channels for this image

8-Bit Color Image: Case

256-colors image

Doubling the resolution (2*640*2*480) requires four times the image size

- The idea of using Lookup table(*color palette*)

– An image store a set of bytes, not the real color

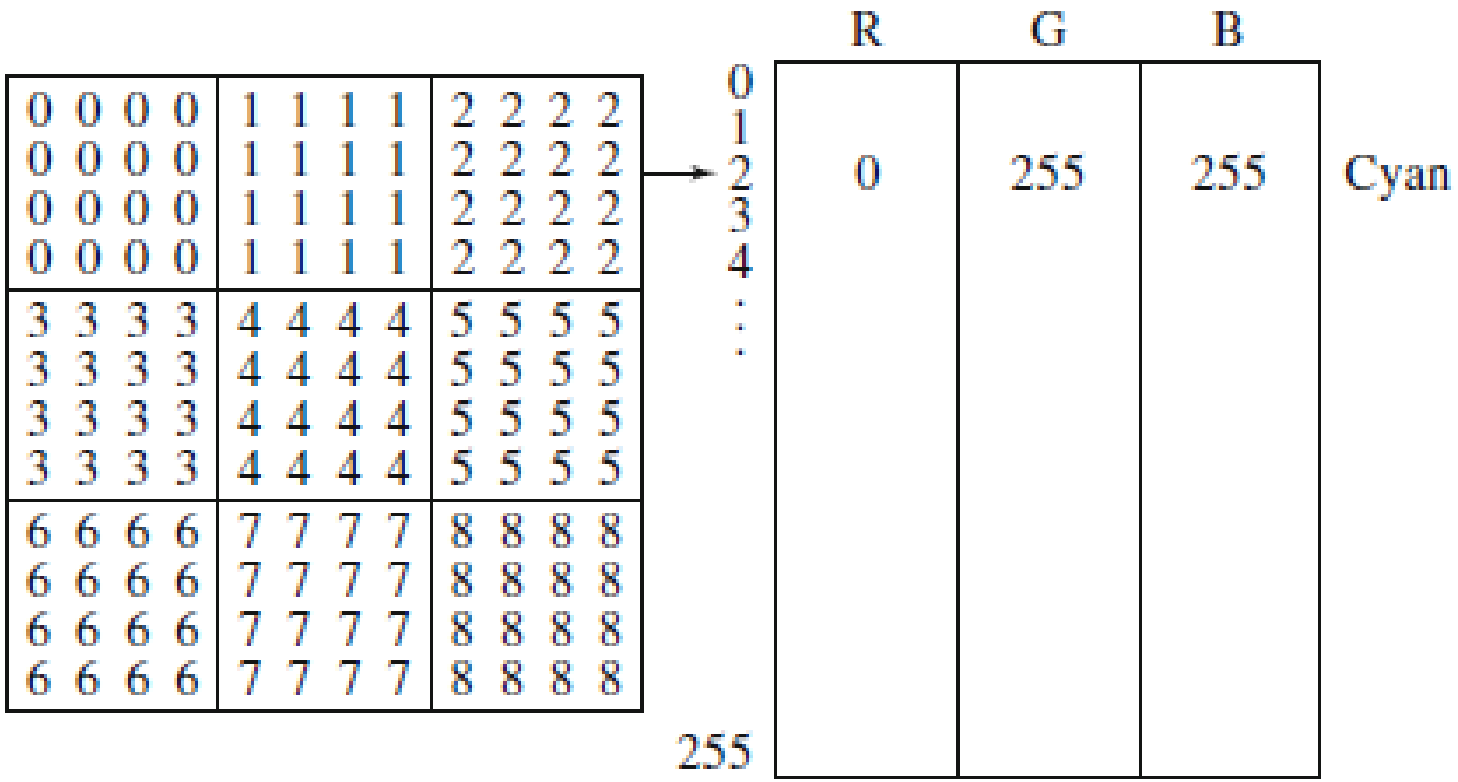
– Bytes value is the index to a 3-bytes color table

– Choosing what colors to put in table is important

Great savings in space for 8-bit images, over 24-bit ones

8-Bit Color Image: Case

Also called 256-colors image



image

Color palette

8-Bit Color Image: Case

Great savings in space for 8-bit images, over 24-bit ones

24-bit Color Image

A 640 x 480 24-bit color image only requires 921.6 kB of storage

8-bit Color Image

A 640 x 480 8-bit color image only requires 307.2 kB of storage

without any compression applied

8-Bit Color Image: Case

Example:

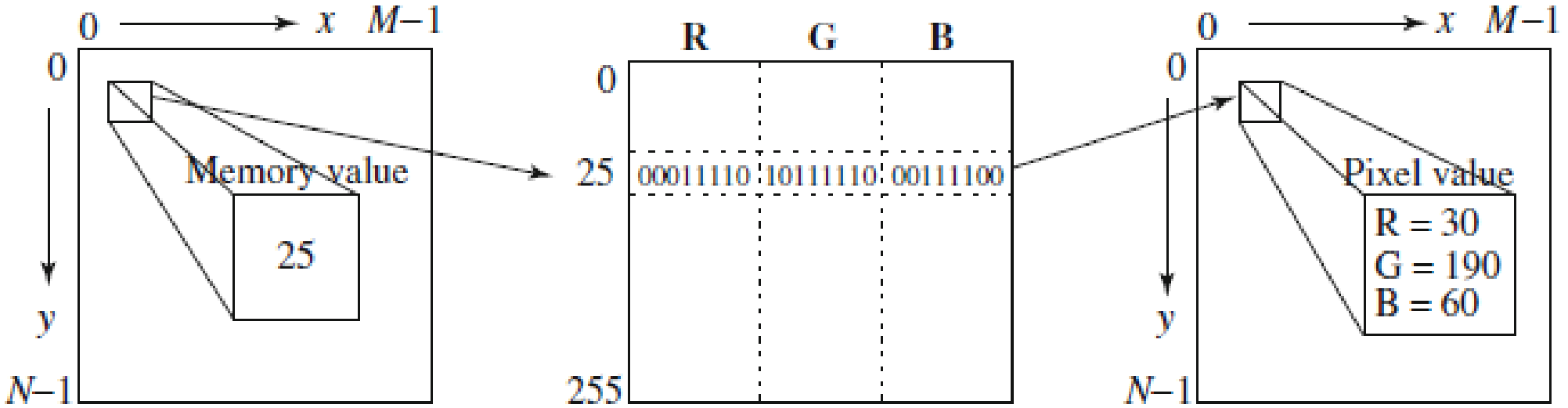
let's calculate the size of 100*100 image,

- If we store it using 24-bit: 100*100*3=30.000 byte.
- If we store it using 8-bit: 100*100*1=10.000 byte + 768 byte for color palette

  ➔ 10.768 byte

Color Lookup Tables : Case

The idea used in 8-bit color images is to store only the index, or code value, for each pixel. Then, e.g., if a pixel stores the value 25, the meaning is to go to row 25 in a color look-up table (LUT)



Value as the Index to Table

Get the color values by Searching

The RGB value of the pixel

Color Lookup Tables : Case

Change color by adjusting the LUT

Example : change LUT

Index	R	G	B
1	255	0	0

into

Index	R	G	B
1	0	255	0

For the color index by 1, that is to convert red to green

An important application: Medical Image

Convert the grey image into color image

How to Devise a Color Lookup Table

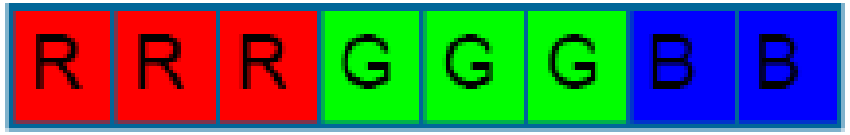
The most straight forward way to make 8-bit lookup color out of 24-bit color would be to divide the RGB cube into equal slices in each dimension.

Then the centers of each of the resulting cubes would serve as the entries in the color LUT, and simply scaling the RGB ranges 0 .. 255 into the appropriate ranges would generate the 8-bit codes.

Since humans are more sensitive to R and G than to B, we could shrink the R range and G range 0 .. 255 into the 3-bit range 0 .. 7 and shrink the B range down to the 2-bit range 0 .. 3, making a total of 8 bits.

Example: 8-Bit Color Image

- Humans are more sensitive to R and G than to B
- So R=3, G=3 and B=2



How to Devise a Color Lookup Table

To shrink R and G, we could simply divide the R or G byte value by (256/8 =) 32 and then truncate. Then each pixel in the image gets replaced by its 8-bit index, and the color LUT serves to generate 24-bit color

Then each pixel in the image gets replaced by its 8-bit index.

Example:

– R: 16, 48, 80, 112, 144, 176, 208, 240

– G: 16, 48, 80, 112, 144, 176, 208, 240

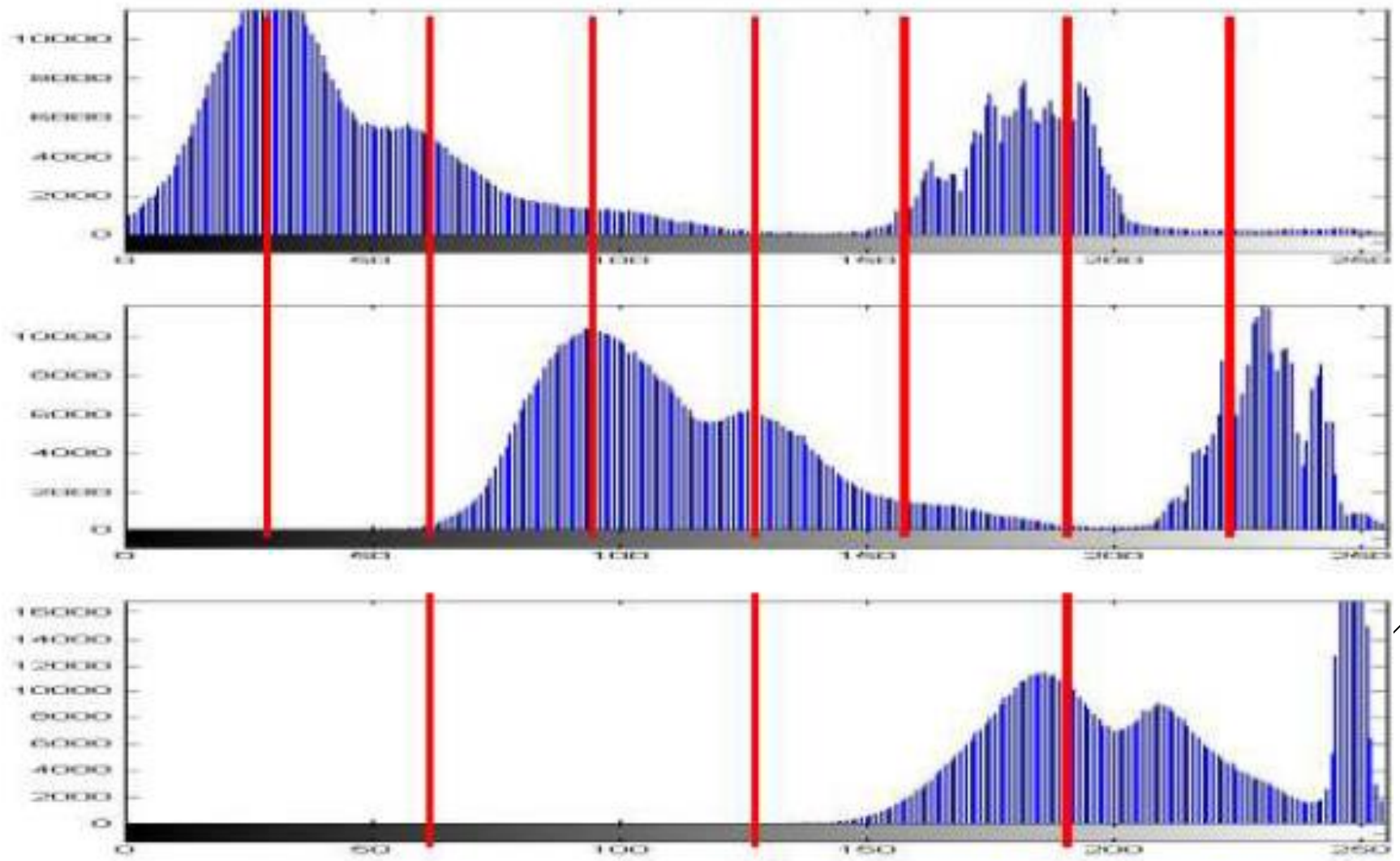
– B: 32, 96, 160, 224

A pixel with color **[30, 129, 80]** should be converted into:

[16, 144, 96]

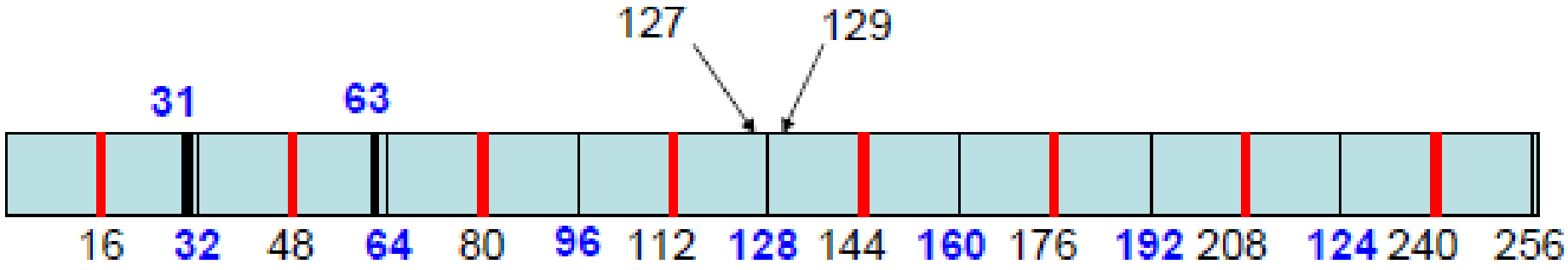
Color Lookup Tables : How to apply

T



Color Lookup Tables : How to apply

If a slight change in RGB results in shifting to a new code, an edge appears



129 -> 144

127 -> 112

2 level difference -> 32 level difference-> an edge

Color Lookup Tables : How to apply

Approach 2: Median-Cut Algorithm:

A simple alternate solution that does a better job for this color reduction problem.

- The idea is to sort the R byte values and find their median; then values smaller than the median are labeled with a "0" bit and values larger than the median are labeled with a "1" bit.

- This type of scheme will indeed concentrate bits where they most need to differentiate between high populations of close colors.

- One can most easily visualize finding the median by using a histogram showing counts at position 0..255.

Color Lookup Tables : How to apply

Approach 2: Median-Cut Algorithm:

Example 1: let's use a simplified set of colors:

Red (255,0,0)

Green (0,255,0)

Blue (0,0,255)

Yellow (255,255,0)

Cyan (0,255,255)

Magenta (255,0,255)

Color Lookup Tables : How to apply

Approach 2: Median-Cut Algorithm:

Example 1:

1. Sort the colors by red channel:

(0, 0, 255), (0, 255, 0), (0, 255, 255), (255, 0, 0), (255, 0, 255), (255, 255, 0)

2. Divide the colors into 2 groups:

Group 1	Group 2
(0, 0, 255), (0, 255, 0), (0, 255, 255)	(255, 0, 0), (255, 0, 255), (255, 255, 0)

Color Lookup Tables : How to apply

Approach 2: Median-Cut Algorithm:

Example 1:

Group 1	Group 2
(0, 0, 255), (0, 255, 0), (0, 255, 255)	(255, 0, 0), (255, 0, 255), (255, 255, 0)

3. Repeat the Division: Divide each Group by the Green channel:

Group 1-1	Group 1-2
(0, 0, 255)	(0, 255, 0),(0, 255, 255)

Group 2-1	Group 2-2
(255, 0, 0), (255, 0, 255)	(255, 255, 0)

Color Lookup Tables : How to apply

Approach 2: Median-Cut Algorithm:

Example 1:

3. Choose the colors:

Group 1-1	Group 1-2
(0, 0, 255)	(0, 255, 0),(0, 255, 255)

Group 2-1	Group 2-2
(255, 0, 0), (255, 0, 255)	(255, 255, 0)

- From Group 1-1, the median is (0, 0, 255)
- From Group 1-2, the median is (0, 255, 128)
- From Group 2-1, the median is (255, 0, 128)
- From Group 2-2, the median is (255, 255, 0)

Color Lookup Tables : How to apply

Approach 2: Median-Cut Algorithm:

Example 2:

Suppose one group has these 4 colors:

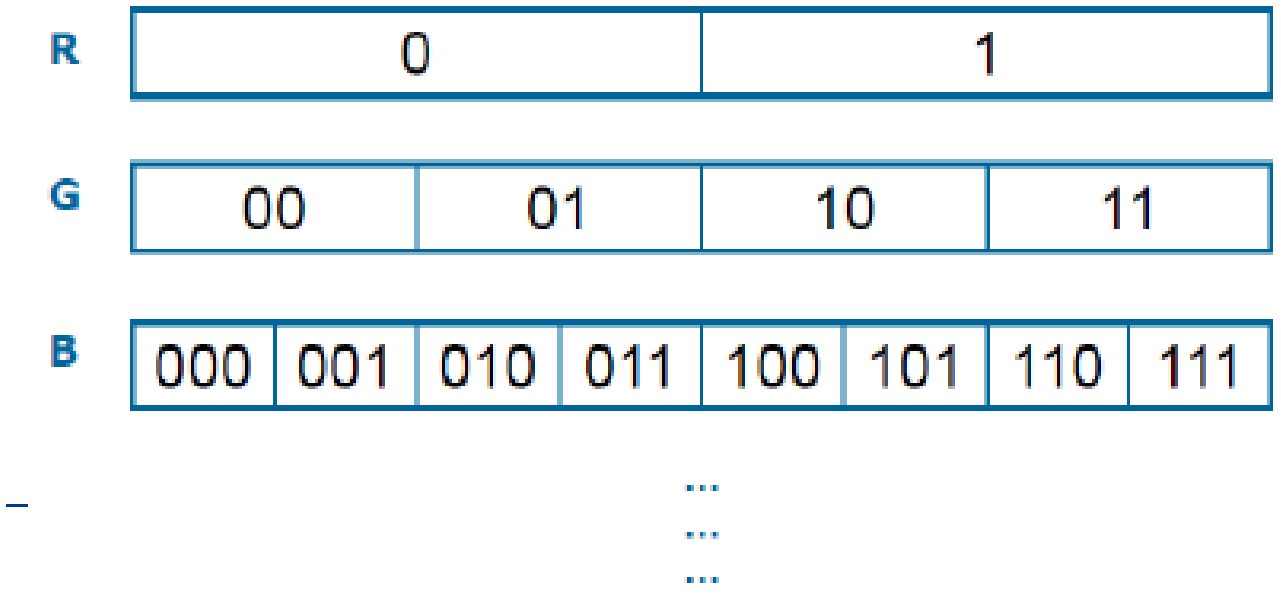
Group XX
(100, 80, 50), (105, 95, 52) , (110, 90, 60), (120, 85, 55)

- R_{avg}= (100+105+110+120)/4=108.75
- G_{avg}= 87.5
- B_{avg}= 54.25

So, (109 , 88, 53) will be in LUT.

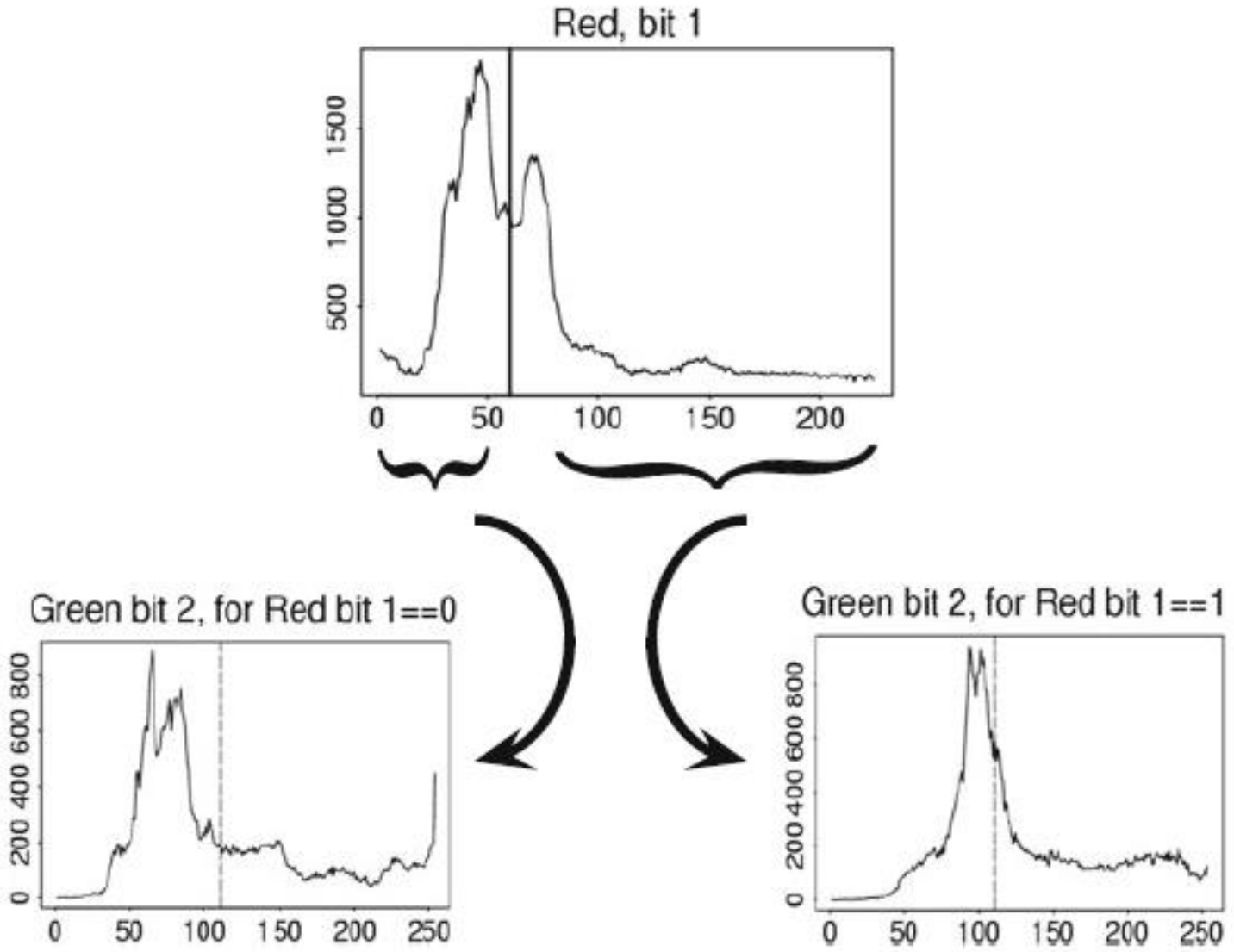
Color Lookup Tables : How to apply

Approach 2: Median-Cut Algorithm:



Color Lookup Tables : How to apply

Median-



Color Lookup Tables : How to apply

The Median: The Median is the 'middle value' in your list.

When the totals of the list are odd, the median is the middle entry in the list **after sorting the list into increasing order.**

When the totals of the list are even, the median is equal to the sum of the two middle (after sorting the list into increasing order) numbers divided by two.

Thus, remember to line up your values.

Examples:

Find the Median of: 9, 3, 44, 17, 15 (Odd amount of numbers)

Line up your numbers: 3, 9, 15, 17, 44 (smallest to largest)

The Median is: 15 (The number in the middle)

Find the Median of: 8, 3, 44, 17, 12, 6 (Even amount of numbers)

Line up your numbers: 3, 6, 8, 12, 17, 44

Add the 2 middles numbers and divide by 2: 8 12 = 20 ÷ 2 = 10

The Median is 10.

Popular image formats

Format Name	Description	Extensions
TIFF	Tagged Image File Format	.tif .tiff
JPEG	Joint Phtographic Experts Group	.jpg .jpeg
GIF	Graphics Interchange Format	.gif
PNG	Portable Network Graphics	.png
XWD	X Window Dump	.xwd

End of lecture