# Chapter 3
# Graphics and Image Data Representations

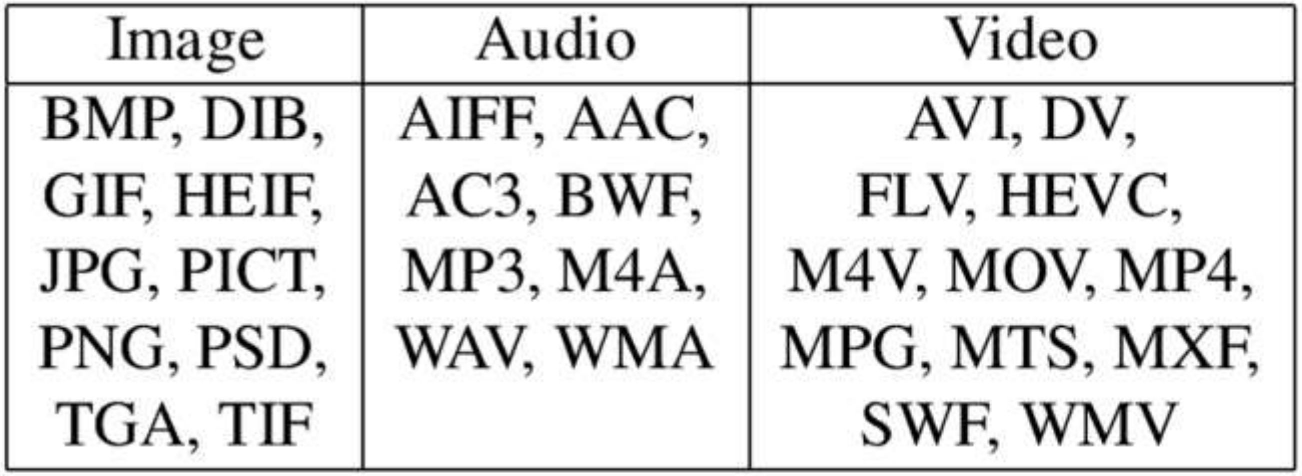
3.1 Graphics and Image Data Types

3.2 Popular File Formats

# 3.1 Graphics and Image Data Types

- The number of file formats used in multimedia continues to proliferate. For example, Table 3.1 shows a list of some file formats used in the popular product Adobe Premiere.

**Table 3.1:** Some Popular Adobe Premiere file formats



| Image | Audio | Video |
|---|---|---|
| BMP, DIB, GIF, HEIF, JPG, PICT, PNG, PSD, TGA, TIF | AIFF, AAC, AC3, BWF, MP3, M4A, WAV, WMA | AVI, DV, FLV, HEVC, M4V, MOV, MP4, MPG, MTS, MXF, SWF, WMV |

# 3.1.1 1-Bit Images

- Each pixel is stored as a single bit (0 or 1), so also referred to as **binary image**.

- Such an image is also called a 1-bit **monochrome** image since it contains no color.

- Fig. 3.1 shows a 1-bit monochrome image (called "Lena" by multimedia scientists — this is a standard image used to illustrate many algorithms).

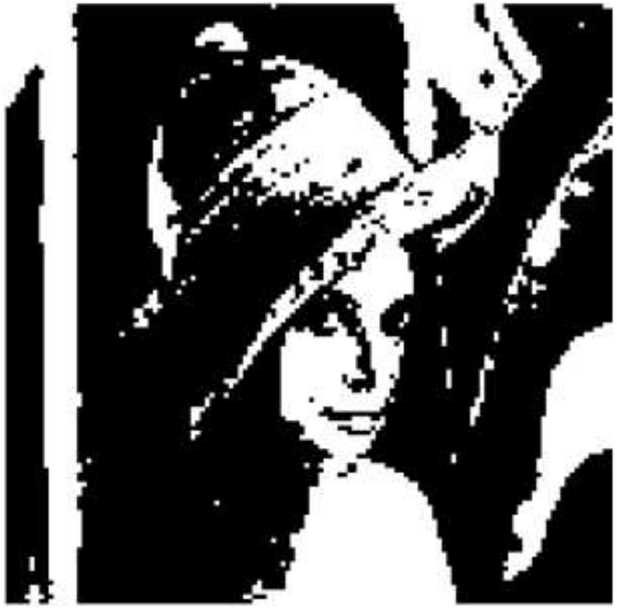


**Fig. 3.1**: Monochrome 1-bit Lena image.

# 3.1.2 8-bit Gray-level Images

- Each pixel has a gray-value between 0 and 255. Each pixel is represented by a single byte; e.g., a dark pixel might have a value of 10, and a bright one might be 230.

- **Bitmap**: The two-dimensional array of pixel values that represents the graphics/image data.

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

  - Fairly high resolution for such an image might be 1600 x 1200, whereas lower resolution might be 640 x 480.

- **Frame buffer**: Hardware used to store bitmap.

  - **Video card** (actually a *graphics card*) is used for this purpose.

  - The resolution of the video card does not have to match the desired resolution of the image, but if not enough video card memory is available then the data has to be shifted around in RAM for display.

- An 8-bit image can be thought of as a set of 1-bit **bit-planes**, where each plane consists of a 1-bit contribution to the image : a bit is turned on if the image pixel value has a nonzero value for that bit. E.g., if a pixel stores 1 byte, then all the pixels with byte value > 127 have the most significant bit turned on.

- Fig. 3.2 displays the concept of bit-planes graphically.

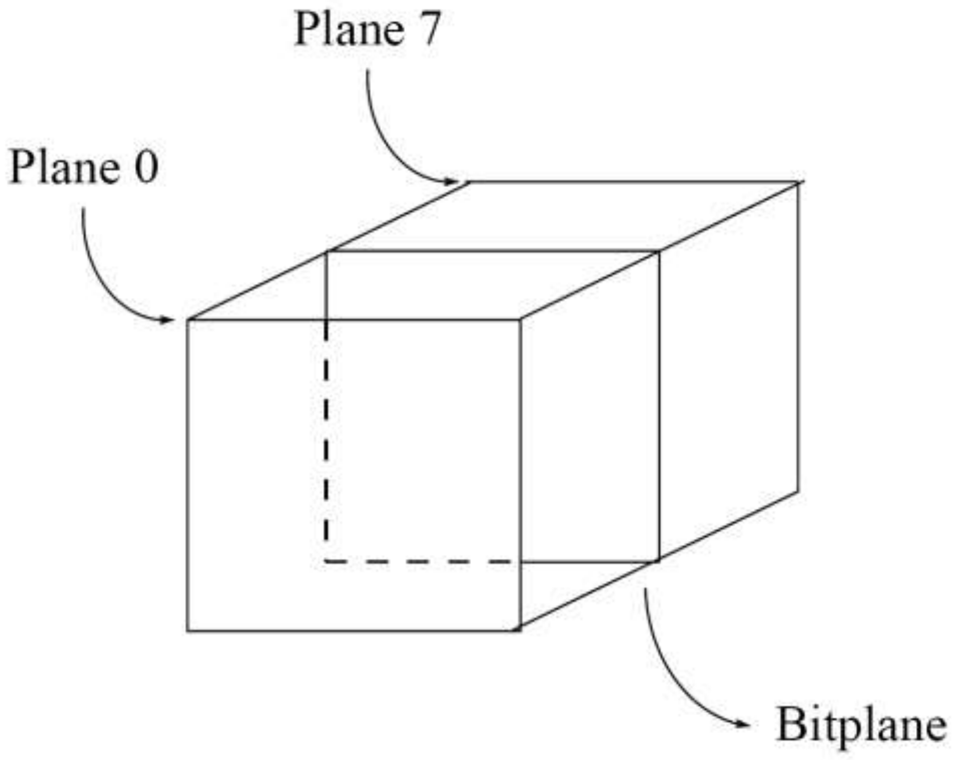


**Fig. 3.2**: Bit-planes for 8-bit grayscale image.

- Each pixel is usually stored as a byte (a value between 0 to 255), so a 640 x 480 grayscale image requires 300 kB of storage (640 x 480 = 307,200).

- Fig. 3.3 shows the Lena image again, but this time in grayscale.

- When an image is printed, the basic strategy of **dithering** is used, which trades intensity resolution for spatial resolution to provide ability to print multi-level images on 2-level (1-bit) printers.

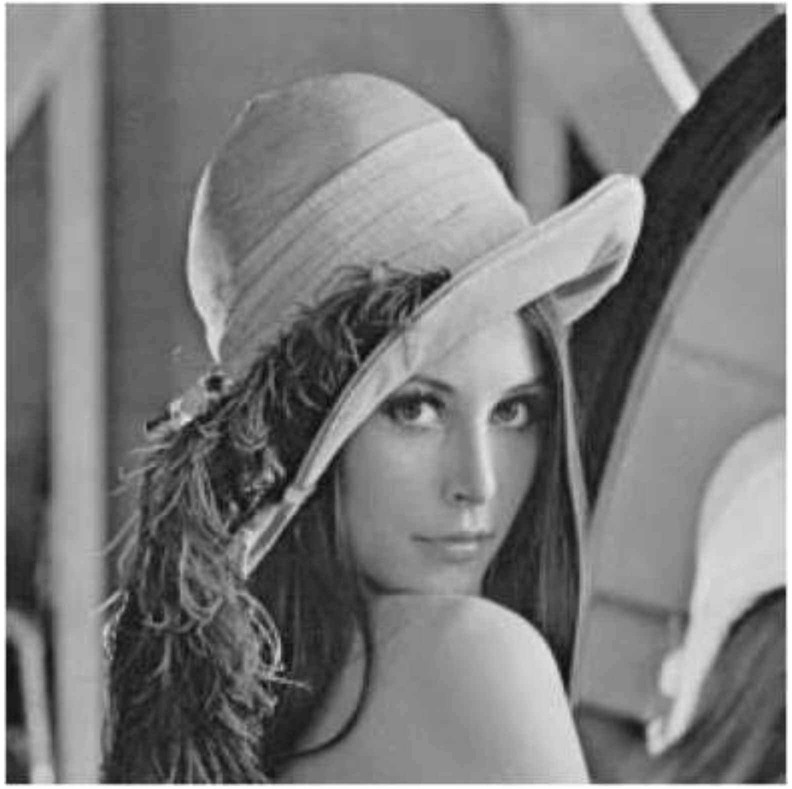


**Fig. 3.3**: Grayscale image of Lena.

# Dithering

- **Dithering** is used to calculate patterns of dots such that values from 0 to 255 correspond to patterns that are more and more filled at darker pixel values, for printing on a 1-bit printer.

- 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 analog, in **halftone printing** (e.g., for newspaper photos).

1. Half-tone printing is an analog process that uses smaller or larger filled circles of black ink to represent shading, for newspaper printing.
2. For example, if we use a 2 x 2 **dither matrix:**

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

we can first re-map image values in 0..255 into the new range 0..4 by (integer) dividing by 256/5. Then, e.g., if the pixel value is 0 we print nothing, in a 2 x 2 area of printer output. But if the pixel value is 4 we print all four dots.

- The rule is:

  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.

- Note that the image size may be much larger, for a dithered image, since replacing each pixel by a 4 x 4 array of dots, makes an image 16 times as large.

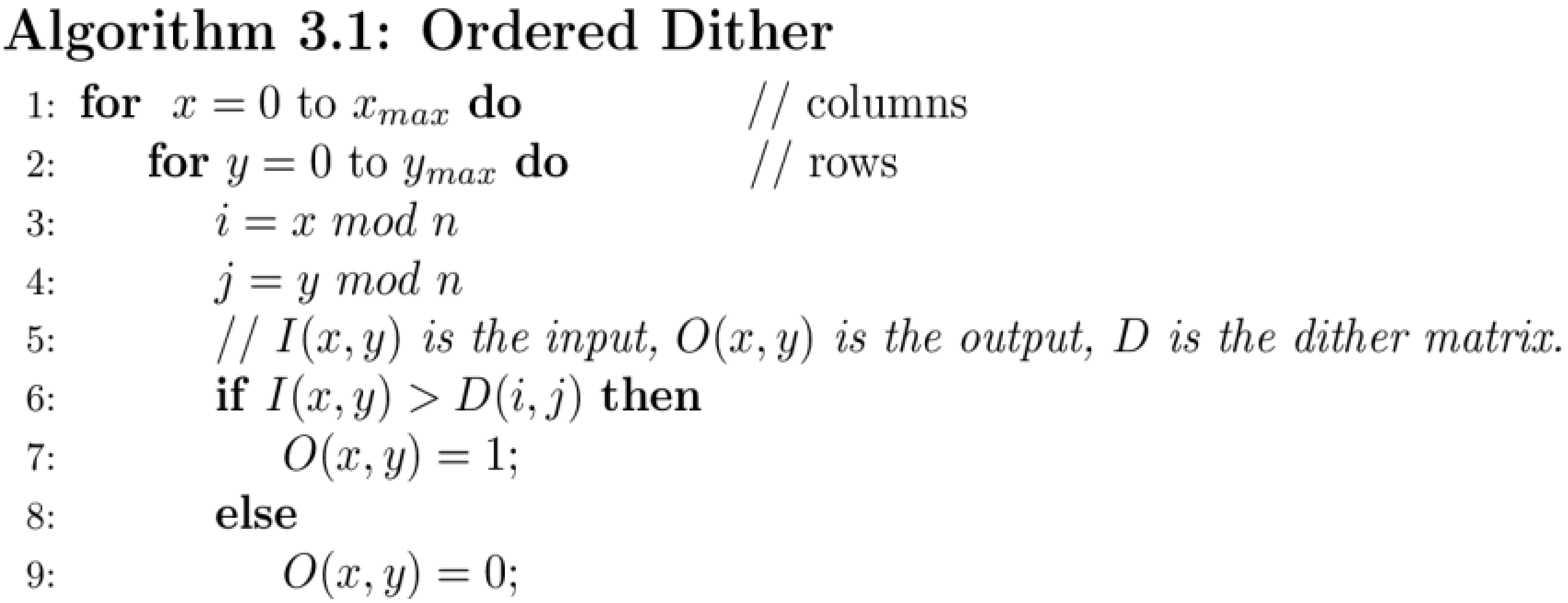
- A clever trick can get around this problem. Suppose we wish to use a larger, 4 x 4 dither matrix, such as

$$\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

- An **ordered dither** consists of turning on the printer out-put bit for a pixel if the intensity level is greater than the particular matrix element just at that pixel position.

- Fig. 3.4 (a) shows a grayscale image of “Lena”. The ordered-dither version is shown as Fig. 3.4 (b), with a detail of Lena's right eye in Fig. 3.4 (c).

- An algorithm for Ordered Dither with *n x n* dither matrix:



**Algorithm 3.1: Ordered Dither**

```
for $x = 0$ to $x_{max}$ do                    // columns
    for $y = 0$ to $y_{max}$ do                // 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 $I(x,y) > D(i,j)$ then
            $O(x,y) = 1;$
        else
            $O(x,y) = 0;$
```

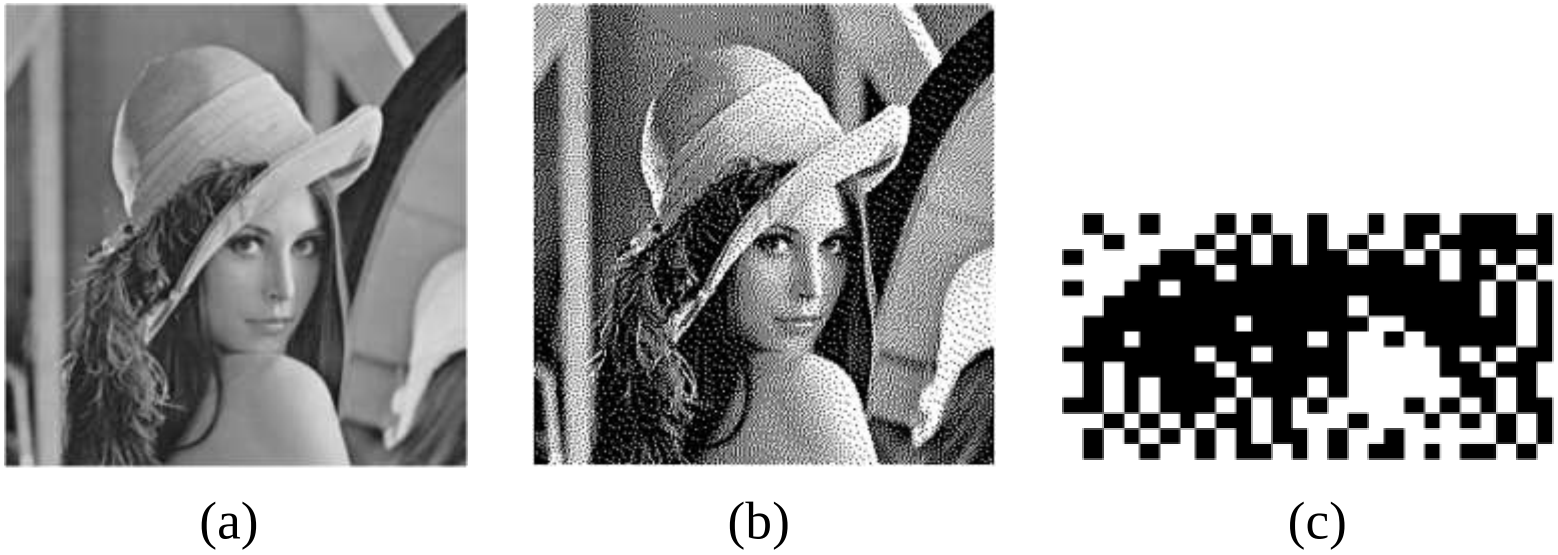


**Fig. 3.4**: Dithering of grayscale images.

(a): 8-bit grey image “lenagray.bmp”.

(b): Dithered version of the image.

(c): Detail of dithered version.

# 3.1.3 Image Data Types

- The most common data types for graphics and image file formats — 24-bit color and 8-bit color.

- Some formats are restricted to particular hardware / operating system platforms, while others are "cross-platform" formats.

- Even if some formats are not cross-platform, there are conversion applications that will recognize and translate formats from one system to another.

- Most image formats incorporate some variation of a **compression** technique due to the large storage size of image files. Compression techniques can be classified into either **lossless** or **lossy**.

# 3.1.4 24-bit Color Images

- In a color 24-bit image, each pixel is represented by three bytes, usually representing RGB.
  - This format supports 256 x 256 x 256 possible combined colors, or a total of 16,777,216 possible colors.
  - However such flexibility does result in a storage penalty: A 640 x 480 24-bit color image would require 921.6 kB of storage without any compression.
- **An important point**: many 24-bit color images are actually stored as 32-bit images, with the extra byte of data for each pixel used to store an *alpha* value representing special effect information (e.g., transparency).
- Fig. 3.5 shows the image **forestfire.bmp**: a 24-bit image in Microsoft Windows BMP format, and the grayscale images for the Red, Green, and Blue channels for this image.

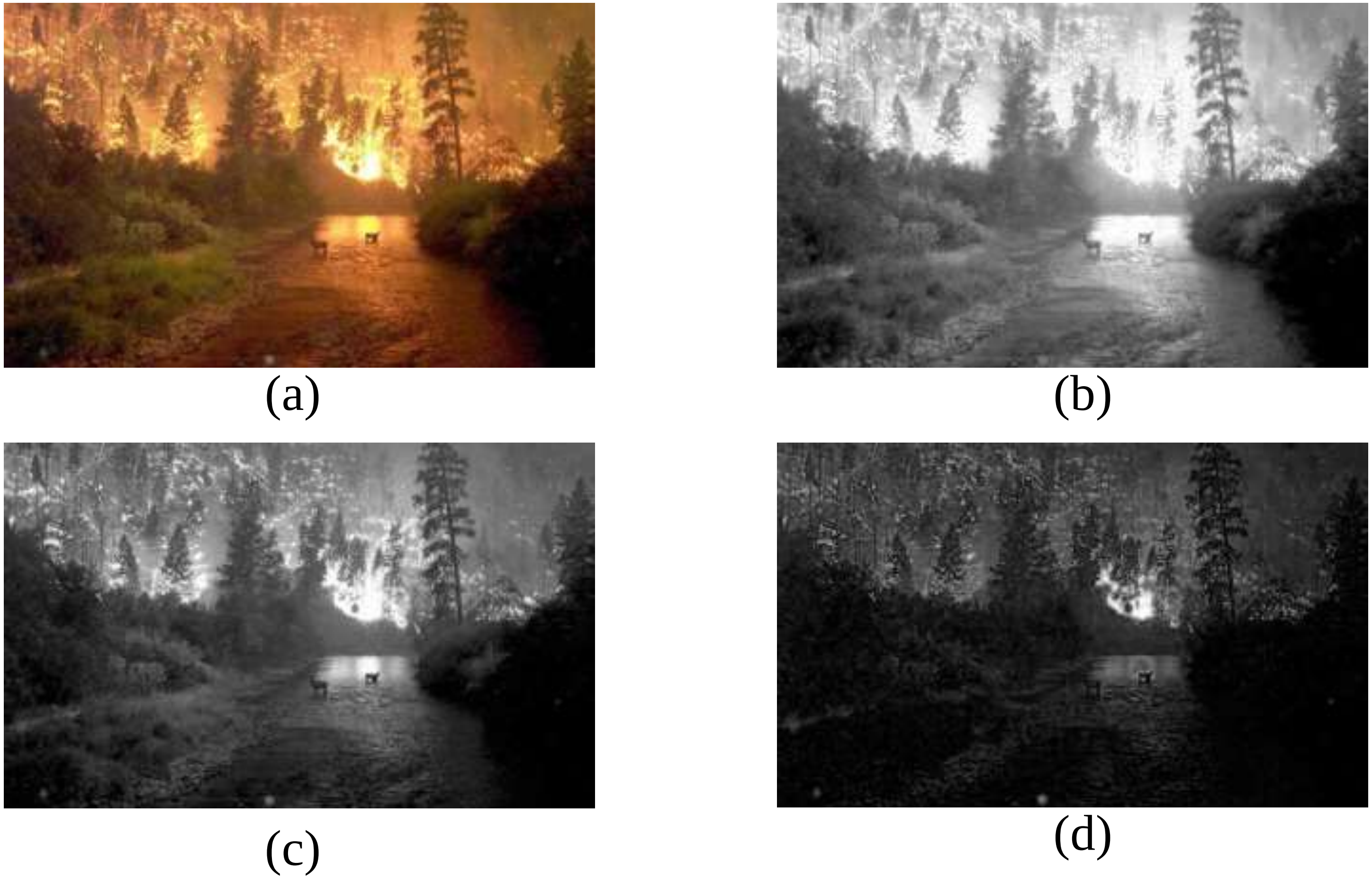


**Fig. 3.5**: High-resolution color and separate R, G, B color channel images. (a): Example of 24-bit color image “forestfire.bmp”. (b, c, d): R, G, and B color channels for this image.

# 3.1.5 Higher Bit-depth Images

More information about the scene being imaged can be gained by using more accuracy for pixel depth (64 bits, say); or by using special cameras that view more than just three colors (i.e., RGB).

- could use invisible light (e.g., infra-red, ultraviolet) for security cameras: “dark flash”.
- use higher-dimensional medical images of skin (> 3-D) to diagnose skin carcinoma.
- in satellite imaging, use high-D to obtain types of crop growth, etc.

Such images are called *multispectral* (more than 3 colors) or hyperspectral (a great many image planes, say 224 colors for satellite imaging).

# 3.1.6 8-Bit Color Images

- Many systems can make use of 8 bits of color information (the so-called “256 colors”) in producing a screen image.

- Such image files use the concept of a **lookup table** to store color information.
  - Basically, the image stores not color, but instead just a set of bytes, each of which is actually an index into a table with 3-byte values that specify the color for a pixel with that lookup table index.

- Fig. 3.7 shows the resulting 8-bit image, in GIF format.

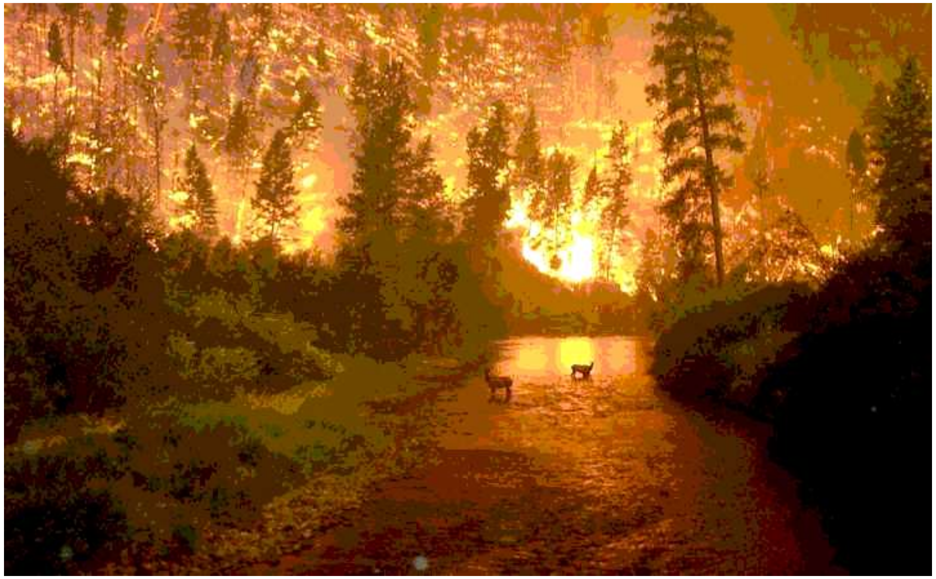


**Fig. 3.7:** Example of 8-bit color image.

- Note the great savings in space for 8-bit images, over 24-bit ones: a 640 x 480 8-bit color image only requires 300 kB of storage, compared to 921.6 kB for a color image (again, without any compression applied).

# 3.1.7 Color Look-up Tables (LUTs)

- 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).

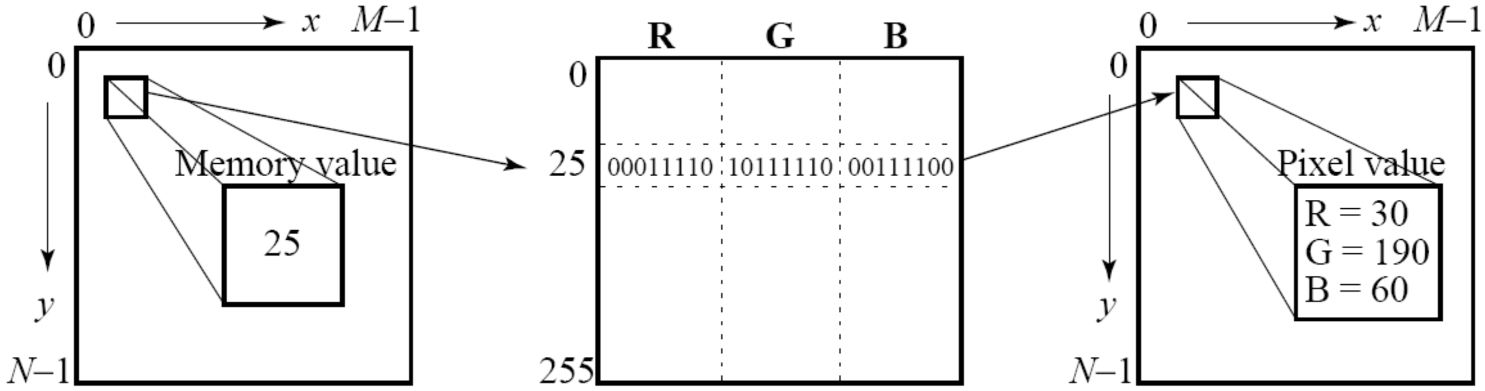
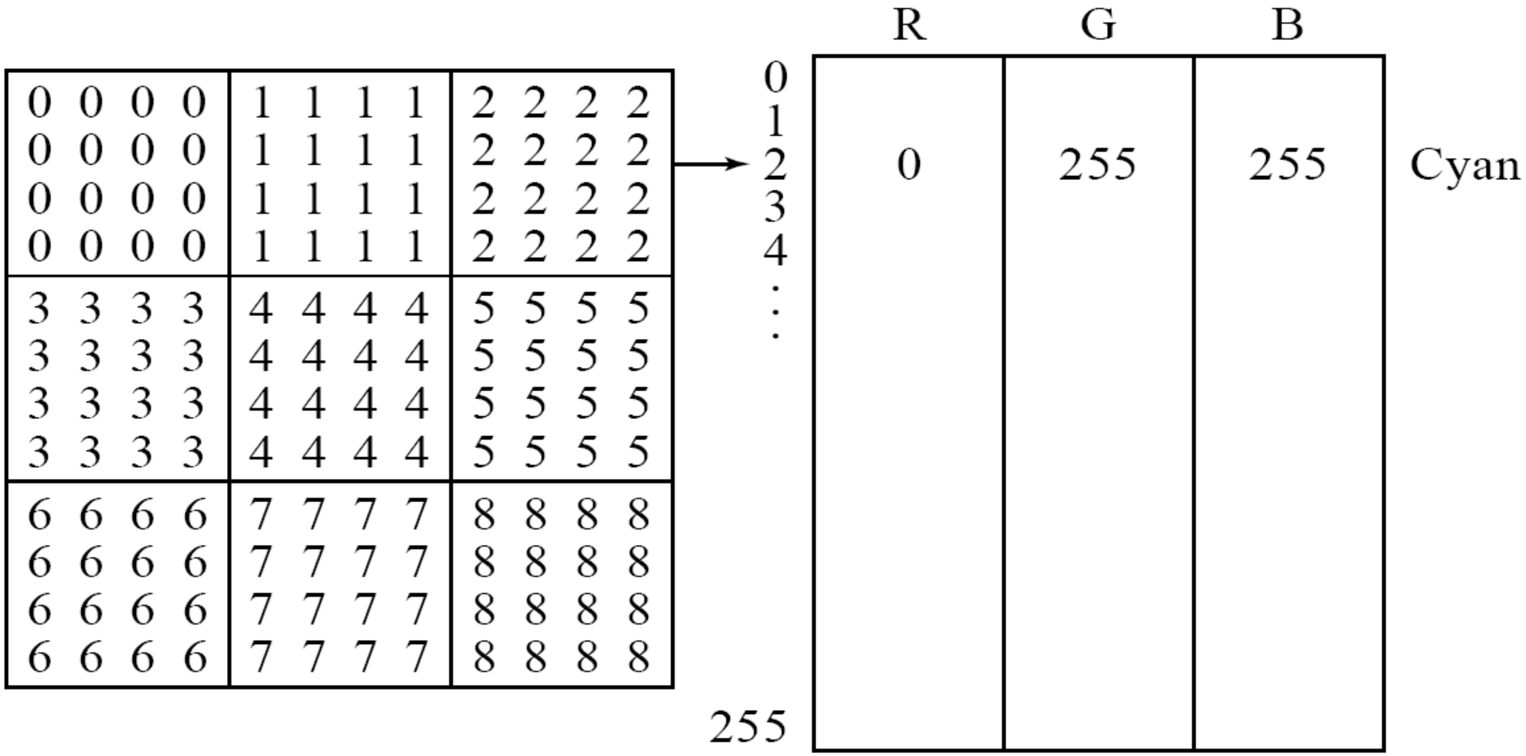


**Fig. 3.8**: Color LUT for 8-bit color images.

- A **Color-picker** consists of an array of fairly large blocks of color (or a semi-continuous range of colors) such that a mouse-click will select the color indicated.

  - In reality, a color-picker displays the palette colors associated with index values from 0 to 255.
  - Fig. 3.9 displays the concept of a color-picker: if the user selects the color block with index value 2, then the color meant is cyan, with RGB values (0, 255, 255).



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 |
| 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 |
| 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 |
| 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 8 | 8 | 8 | 8 |
| 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 8 | 8 | 8 | 8 |
| 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 8 | 8 | 8 | 8 |
| 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 8 | 8 | 8 | 8 |

| | R | G | B | |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | 0 | 255 | 255 | Cyan |
| 3 | | | | |
| 4 | | | | |
| ⋮ | | | | |
| 255 | | | | |

**Fig. 3.9:** Color-picker for 8-bit color: each block of the color-picker corresponds to one row of the color LUT.

A very simple animation process is possible via simply changing the color table: this is called **color cycling** or **palette animation**.

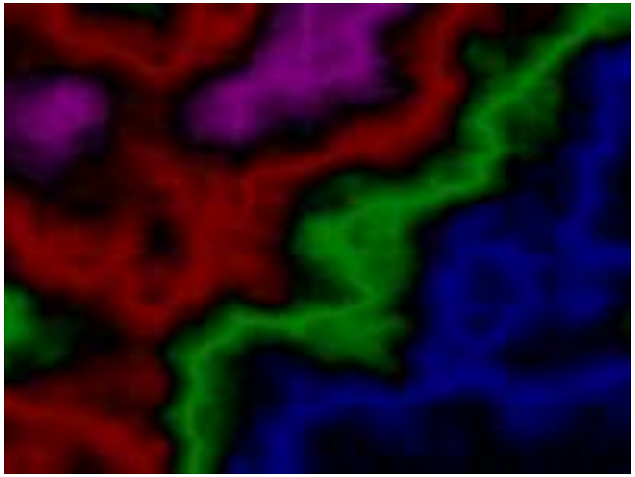
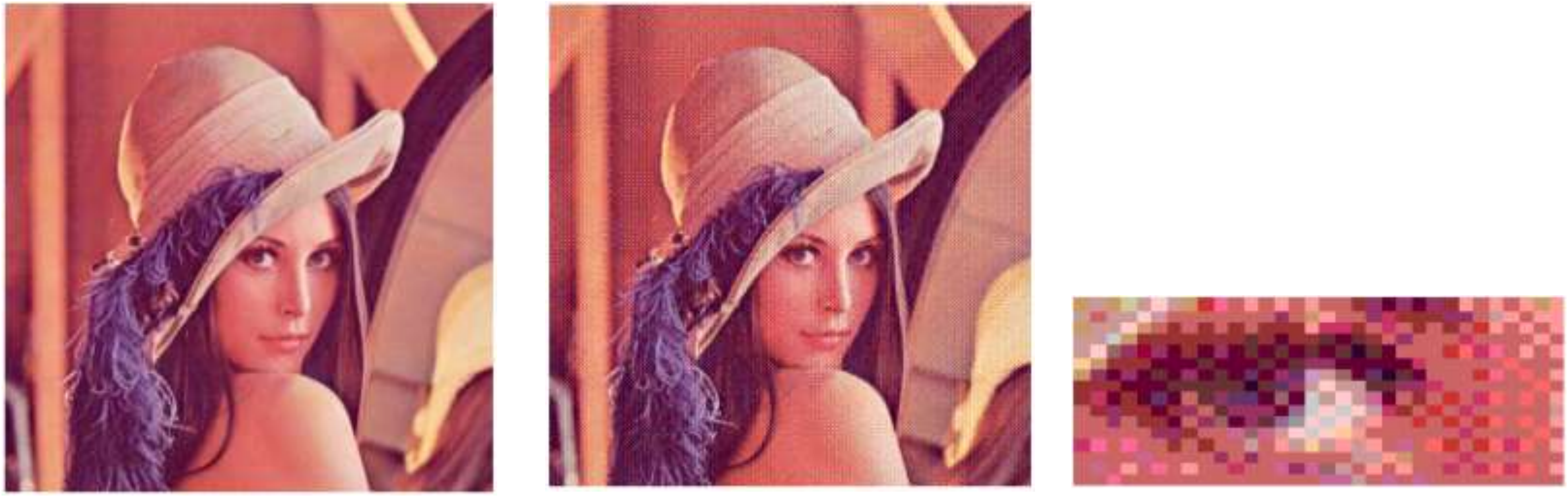


Fig. 3.10 (a) shows a 24-bit color image of “Lena”, and Fig. 3.10 (b) shows the same image reduced to only 5 bits via dithering. A detail of the left eye is shown in Fig. 3.10 (c).



(a) (b) (c)

**Fig. 3.10:** (a): 24-bit color image “lena.bmp”. (b): Version with color dithering. (c): Detail of dithered version.

# How to devise a color look-up table

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

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

    b) 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, thus making up a total of 8 bits.

    c) 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.

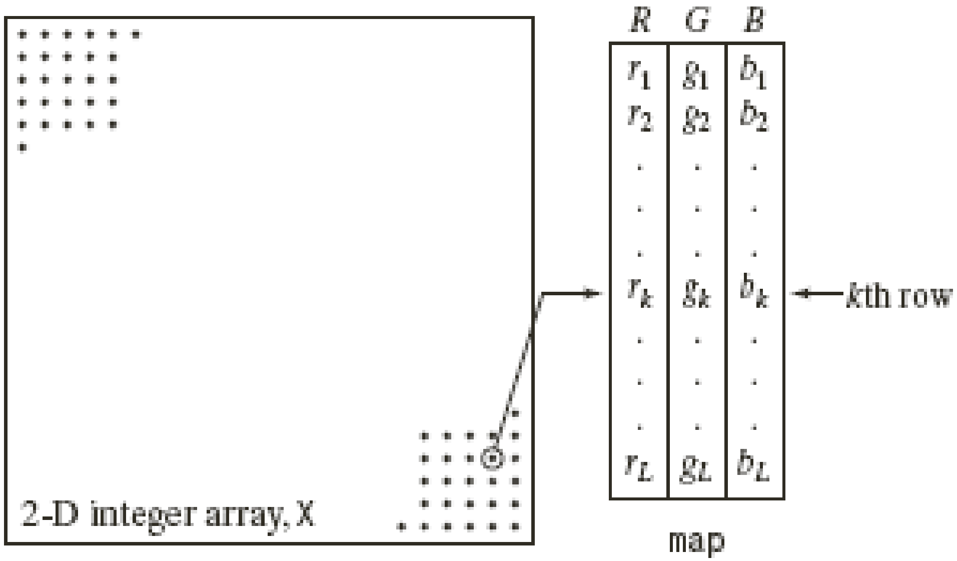
# Indexed Images

- An *indexed* image has two components:
  - *data matrix* of integers, X,
  - *color map matrix*, map.
- Matrix map is an *m* X 3 array of class double containing floating-point values in the range [0,1].
- The length of the map is equal to the number of colors it defines.
- Each row of map specifies the red, green, and blue components of a single color (if the three columns of map are equal, the color map becomes a *gray-scale map).*

# Indexed Images

- The color of each pixel is determined by using the corresponding value of integer matrix X as an index (hence the name *indexed* image) into map.

If X is of class double, then value 1 points to the first row in map, value 2 points to the second row, and so on. If X is of class uint8 or uint16, then 0 points to the first row in map.



Value of circled element = $k$

# Indexed Images

- To display an indexed image we write

  imshow(X, map)

- or, alternatively,

  image(X)

  colormap(map)

- A color map is stored with an indexed image and is automatically loaded with the image when the imread function is used to load the image.

# imapprox

- Sometimes it is necessary to approximate an indexed image by one with fewer colors. For this we use function imapprox, whose syntax is

  [V, newmap] = imapprox(X, map, n)

- This function returns an array Y with color map newmap, which has at most n colors.
- The input array X can be of class uint8, uint16, or double.
- The output Y is of class uint8 if n is less than or equal to 256. If n is greater than 256, Y is of class double.

# Example

- When the number of rows in a map is less than the number of distinct integer values in X, multiple values in X are assigned the same color in the map.
- For example, suppose that X consists of four vertical bands of equal width, with values 1,64,128, and 256.

```
X = ones(200,200);
X(:,1:50) = 1;
X(:,50:100) = 64;
X(:,100:150) = 128;
X(:,150:200) = 256;
map = [0 0 0; 1 1 1];
```

- If we specify the color map map = [0 0 0; 1 1 1] , then all the elements in X with value 1 would point to the first row (black) of the map and all the other elements would point to the second row (white).
- The command imshow (X, map) would display an image with a black band followed by three white bands.

# Predefined color maps

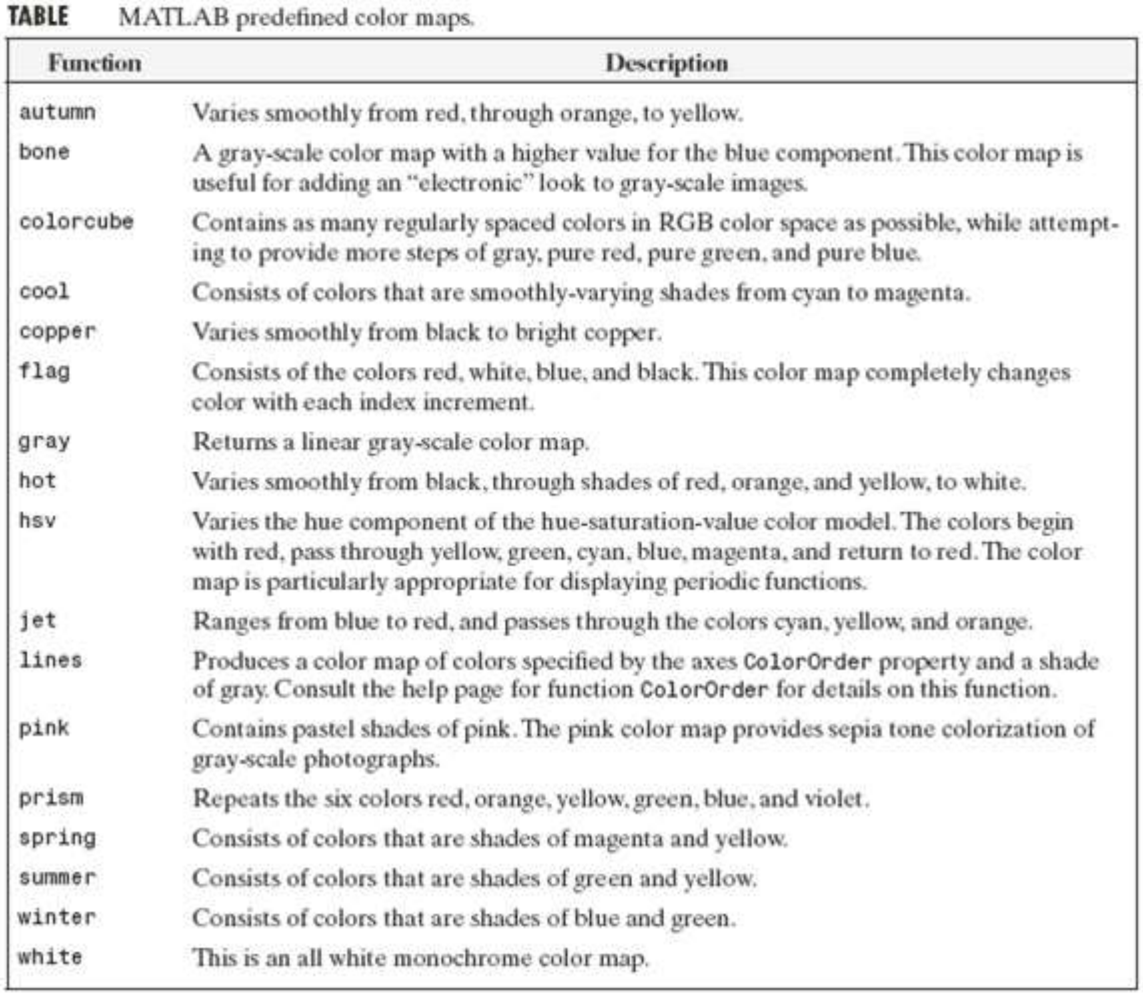


**TABLE** MATLAB predefined color maps.

| Function | Description |
|---|---|
| `autumn` | Varies smoothly from red, through orange, to yellow. |
| `bone` | A gray-scale color map with a higher value for the blue component. This color map is useful for adding an "electronic" look to gray-scale images. |
| `colorcube` | Contains as many regularly spaced colors in RGB color space as possible, while attempting to provide more steps of gray, pure red, pure green, and pure blue. |
| `cool` | Consists of colors that are smoothly-varying shades from cyan to magenta. |
| `copper` | Varies smoothly from black to bright copper. |
| `flag` | Consists of the colors red, white, blue, and black. This color map completely changes color with each index increment. |
| `gray` | Returns a linear gray-scale color map. |
| `hot` | Varies smoothly from black, through shades of red, orange, and yellow, to white. |
| `hsv` | Varies the hue component of the hue-saturation-value color model. The colors begin with red, pass through yellow, green, cyan, blue, magenta, and return to red. The color map is particularly appropriate for displaying periodic functions. |
| `jet` | Ranges from blue to red, and passes through the colors cyan, yellow, and orange. |
| `lines` | Produces a color map of colors specified by the axes `ColorOrder` property and a shade of gray. Consult the help page for function `ColorOrder` for details on this function. |
| `pink` | Contains pastel shades of pink. The pink color map provides sepia tone colorization of gray-scale photographs. |
| `prism` | Repeats the six colors red, orange, yellow, green, blue, and violet. |
| `spring` | Consists of colors that are shades of magenta and yellow. |
| `summer` | Consists of colors that are shades of green and yellow. |
| `winter` | Consists of colors that are shades of blue and green. |
| `white` | This is an all white monochrome color map. |

# Converting between images

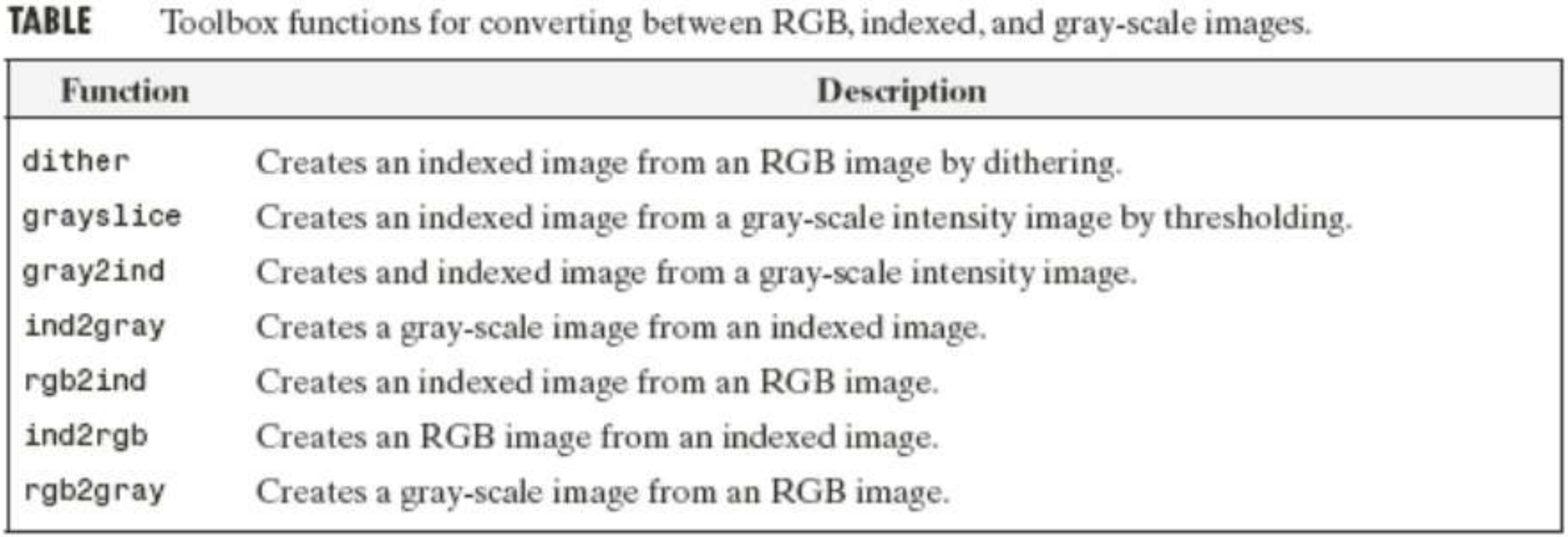


**TABLE** Toolbox functions for converting between RGB, indexed, and gray-scale images.

| Function | Description |
|---|---|
| `dither` | Creates an indexed image from an RGB image by dithering. |
| `grayslice` | Creates an indexed image from a gray-scale intensity image by thresholding. |
| `gray2ind` | Creates and indexed image from a gray-scale intensity image. |
| `ind2gray` | Creates a gray-scale image from an indexed image. |
| `rgb2ind` | Creates an indexed image from an RGB image. |
| `ind2rgb` | Creates an RGB image from an indexed image. |
| `rgb2gray` | Creates a gray-scale image from an RGB image. |

# rgb2ind

[X, map] = rgb2ind(rgb_image, n, dither_option)

- n determines the number of colors of map, and dither_option can have one of two values:
  - 'dither' (the default) dithers, if necessary, to achieve better color resolution at the expense of spatial resolution;
  - 'nodither' maps each color in the original image to the closest color in the new map (depending on the value of n); no dithering is performed.
- The input image can be of class uint8, uint16, or double.
- The output array, X, is of class uint8 if n is less than or equal to 256; otherwise it is of class uint 16.

rgb_image = ind2rgb(X, map)

- converts the matrix X and corresponding color map map to RGB format; X can be of class uint8, uint16, or double.
- The output RGB image is an $M$ X $N$ X 3 array of class double.

# rgb2gray

gray_image = rgb2gray(rgb_image)

- converts an RGB image to a gray-scale image.
- The input RGB image can be of class uint8, uint16, or double; the output image is of the same class as the input.

# Example

```
F = imread('Fig0704(a).tif');
[X1, map1] = rgb2ind(f, 8, 'nodither');
imshow(X1, map1)
[X2, map2] = rgb2ind(f, 8, 'dither');
figure, imshow(X2, map2)
```

# 3.2 Popular File Formats

- **8-bit GIF**: one of the most popular formats because of its historical connection to the WWW and HTML markup language as the first image type recognized by net browsers.
- **JPEG**: currently the most important common file format.
- **PNG:** most popular lossless image format.
- **TIFF:** flexible file format due to the addition of tags.
- **BMP:** standard image file format for Windows.
- **PS and PDF:** vector based language, popular in publishing and academia
- **Many others**

# 3.2.1 GIF

- **GIF standard**: (We examine GIF standard because it is so simple, yet contains many common elements.)

  Limited to 8-bit (256) color images only, which, while producing acceptable color images, is best suited for images with few distinctive colors (e.g., graphics or drawing).

- GIF standard supports **interlacing** — successive display of pixels in widely-spaced rows by a 4-pass display process.

- GIF actually comes in two flavors:

  1. **GIF87a**: The original specification.
  2. **GIF89a**: The later version. Supports simple animation via a Graphics Control Extension block in the data, provides simple control over delay time, a transparency index, etc.

# GIF87

- For the standard specification, the general file format of a GIF87 file is as in Fig. 3.12.

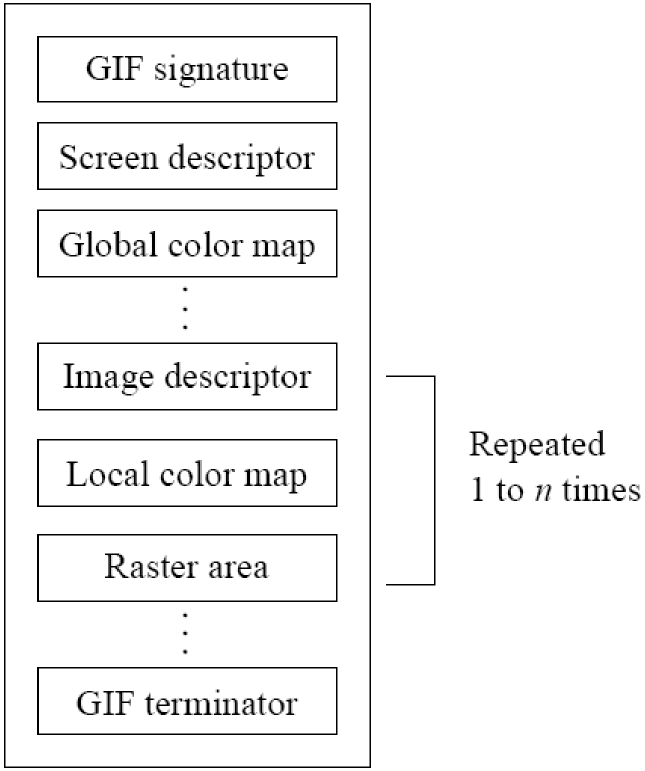


**Fig. 3.12:** GIF file format.

- **Screen Descriptor** comprises a set of attributes that belong to every image in the file. According to the GIF87 standard, it is defined as in Fig. 3.13.

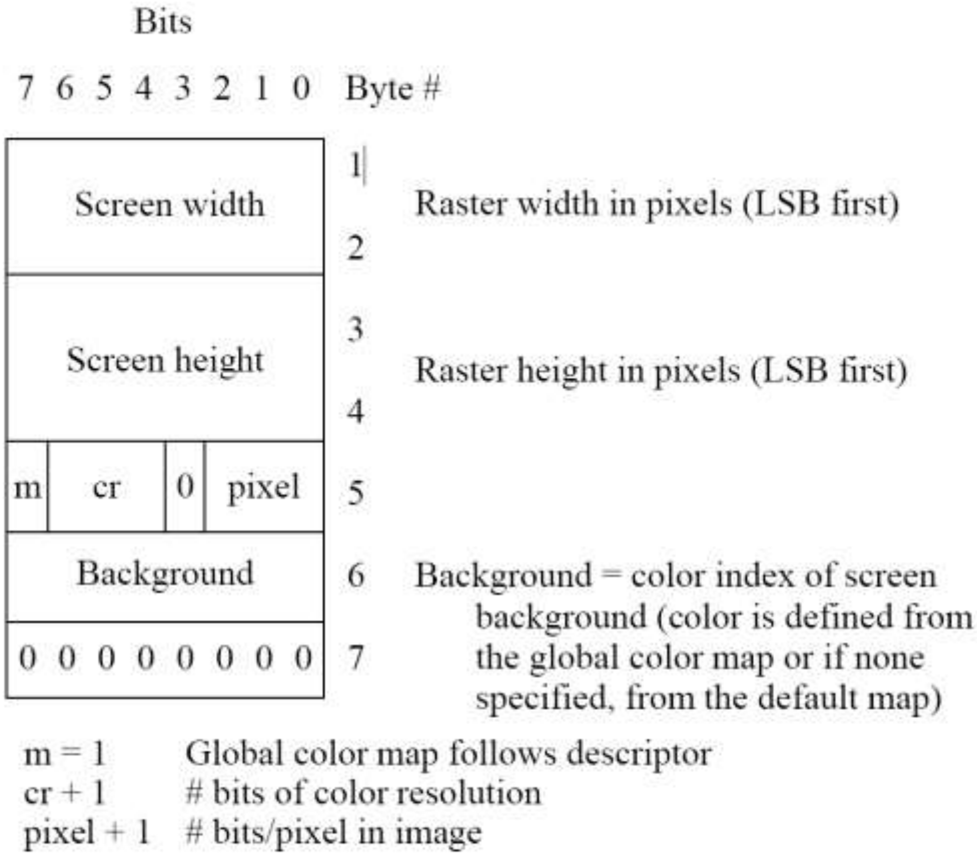
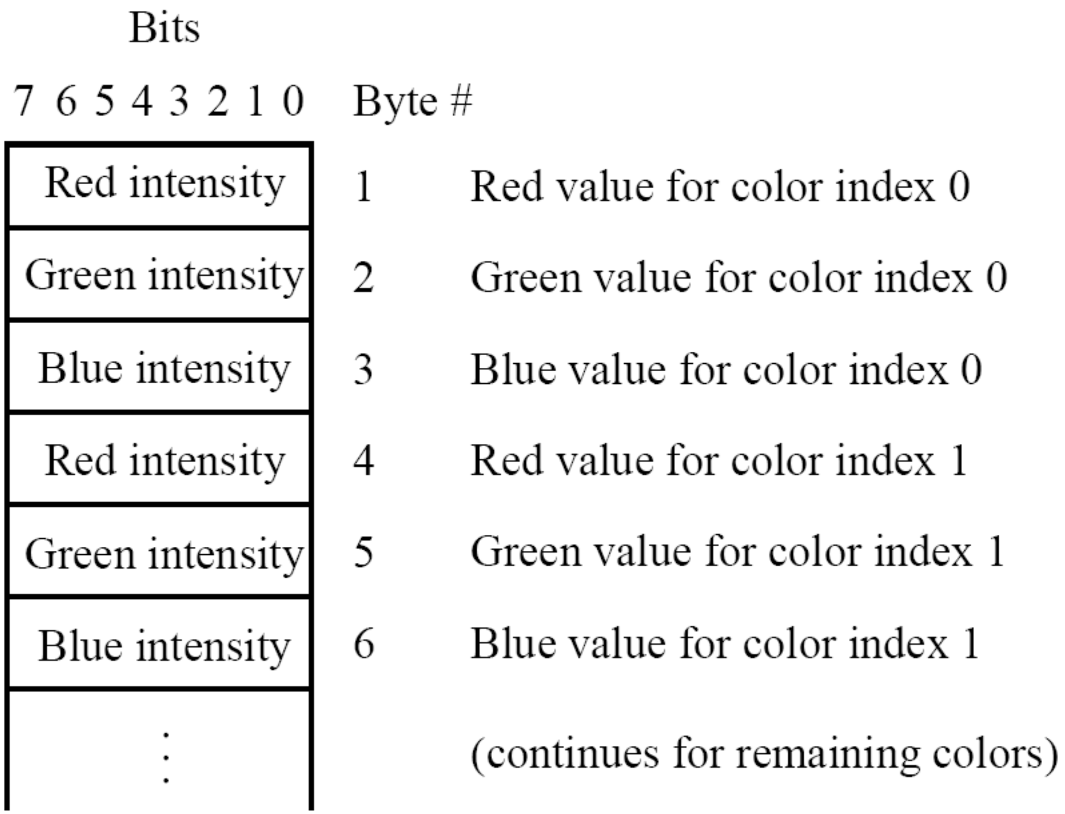

Bits
7 6 5 4 3 2 1 0
Byte #
Screen width
1
2
Raster width in pixels (LSB first)
Screen height
3
4
Raster height in pixels (LSB first)
m
cr
0
pixel
5
Background
6
Background = color index of screen background (color is defined from the global color map or if none specified, from the default map)
0 0 0 0 0 0 0 0
7
m = 1 Global color map follows descriptor
cr + 1 # bits of color resolution
pixel + 1 # bits/pixel in image

**Fig. 3.13:** GIF screen descriptor

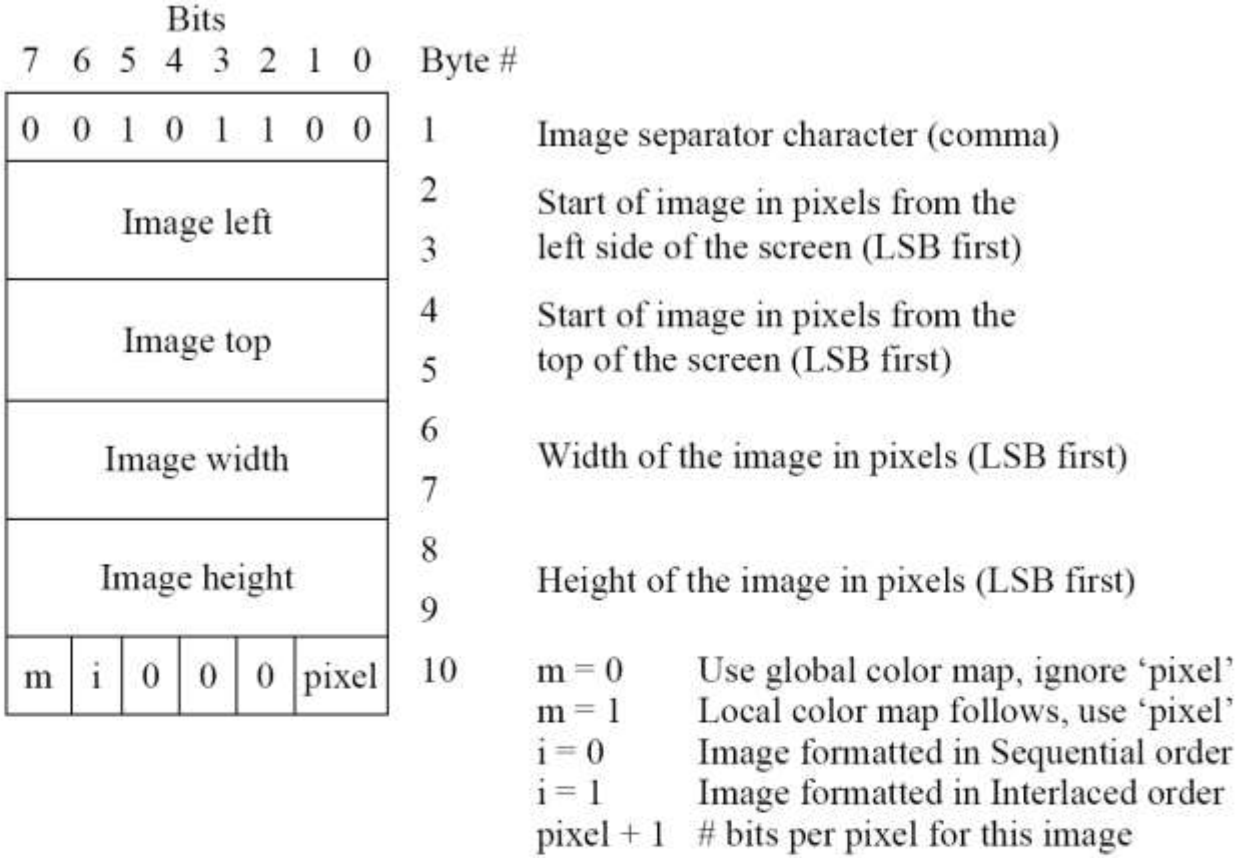
- **Color Map** is set up in a very simple fashion as in Fig. 3.14. However, the actual length of the table equals $2^{(pixel+1)}$ as given in the Screen Descriptor.



| Bits<br>7 6 5 4 3 2 1 0 | Byte # | |
|---|---|---|
| Red intensity | 1 | Red value for color index 0 |
| Green intensity | 2 | Green value for color index 0 |
| Blue intensity | 3 | Blue value for color index 0 |
| Red intensity | 4 | Red value for color index 1 |
| Green intensity | 5 | Green value for color index 1 |
| Blue intensity | 6 | Blue value for color index 1 |
| ⋮ | | (continues for remaining colors) |

**Fig. 3.14:** GIF color map.

- Each image in the file has its own **Image Descriptor**, defined as in Fig. 3.15.

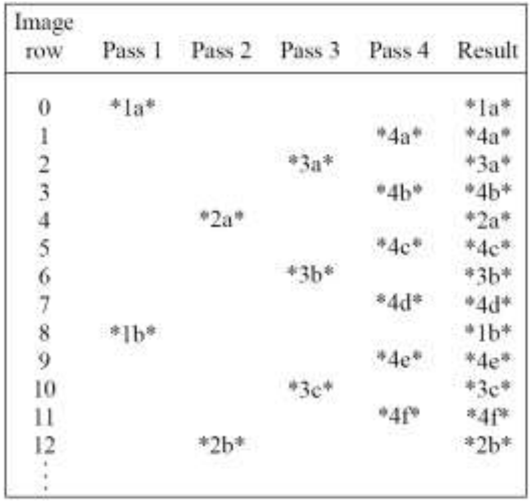


| Bits<br>7 6 5 4 3 2 1 0 | Byte # | |
|---|---|---|
| 0 0 1 0 1 1 0 0 | 1 | Image separator character (comma) |
| Image left | 2<br>3 | Start of image in pixels from the left side of the screen (LSB first) |
| Image top | 4<br>5 | Start of image in pixels from the top of the screen (LSB first) |
| Image width | 6<br>7 | Width of the image in pixels (LSB first) |
| Image height | 8<br>9 | Height of the image in pixels (LSB first) |
| m \| i \| 0 \| 0 \| 0 \| pixel | 10 | |

| | |
|---|---|
| m = 0 | Use global color map, ignore ‘pixel’ |
| m = 1 | Local color map follows, use ‘pixel’ |
| i = 0 | Image formatted in Sequential order |
| i = 1 | Image formatted in Interlaced order |
| pixel + 1 | # bits per pixel for this image |

**Fig. 3.15:** GIF image descriptor.

- If the “interlace” bit is set in the local Image Descriptor, then the rows of the image are displayed in a four-pass sequence (Fig.3.16).
- four passes as follows: Pass 1 - starting at row 0, display every eighth row; Pass 2 - starting at row 4, display every eighth row; Pass 3 - starting at row 2, display every fourth row; Pass 4 - starting at row 1, display every other row



| Image row | Pass 1 | Pass 2 | Pass 3 | Pass 4 | Result |
|---|---|---|---|---|---|
| 0 | *1a* | | | | *1a* |
| 1 | | | | *4a* | *4a* |
| 2 | | | *3a* | | *3a* |
| 3 | | | | *4b* | *4b* |
| 4 | | *2a* | | | *2a* |
| 5 | | | | *4c* | *4c* |
| 6 | | | *3b* | | *3b* |
| 7 | | | | *4d* | *4d* |
| 8 | *1b* | | | | *1b* |
| 9 | | | | *4e* | *4e* |
| 10 | | | *3c* | | *3c* |
| 11 | | | | *4f* | *4f* |
| 12 | | *2b* | | | *2b* |
| ⋮ | | | | | |

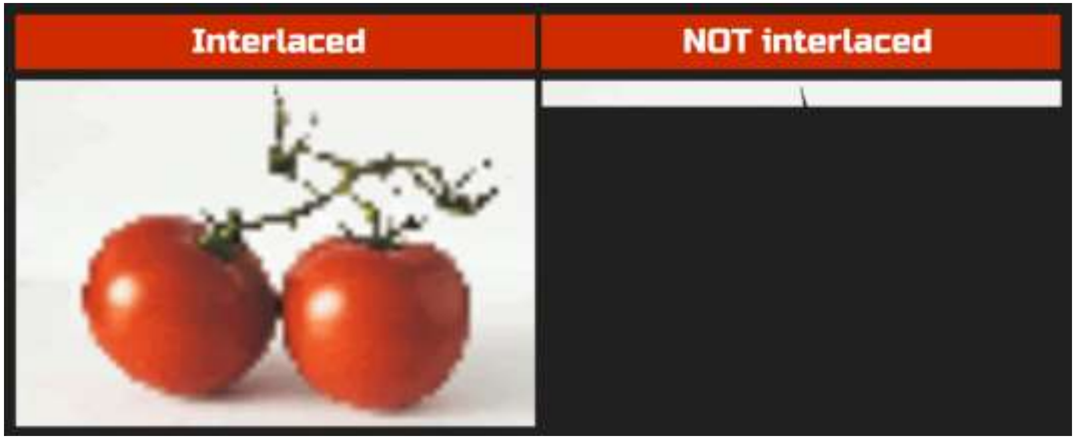


**Fig. 3.16:** GIF 4-pass interlace display row order.

# 3.2.2 JPEG

- **JPEG:** The most important current standard for image compression.

- The human vision system has some specific limitations and JPEG takes advantage of these to achieve high rates of compression.

- JPEG allows the user to set a desired level of quality, or compression ratio (input divided by output).

- As an example, Fig. 3.17 shows our **forestfire** image, with a quality factor Q=10.
  - This image is a mere 1.5% of the original size. In comparison, a JPEG image with Q=75 yields an image size 5.6% of the original, whereas a GIF version of this image compresses down to 23.0% of uncompressed image size.

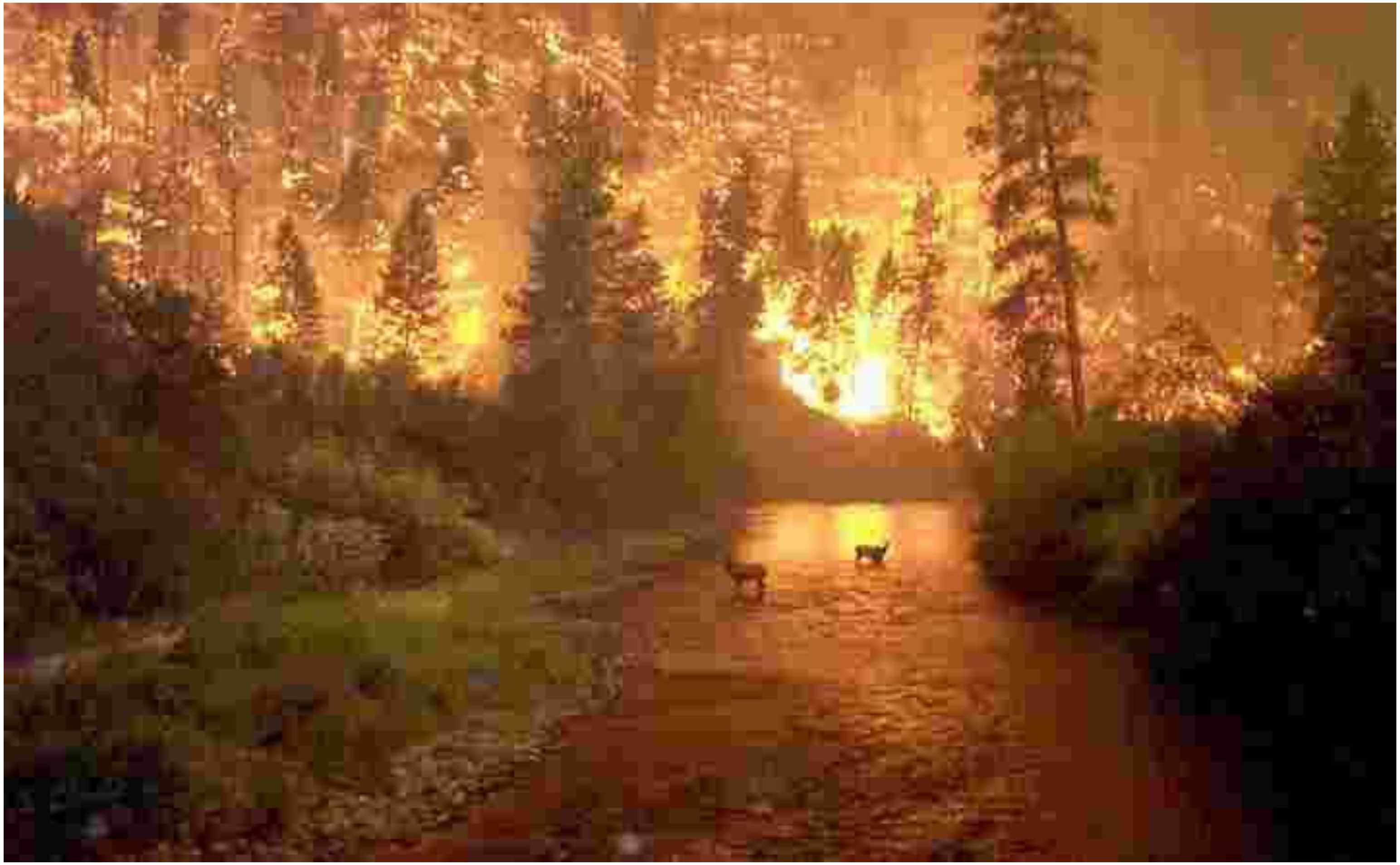


**Fig. 3.17:** JPEG image with low quality specified by user.

# 3.2.3 PNG

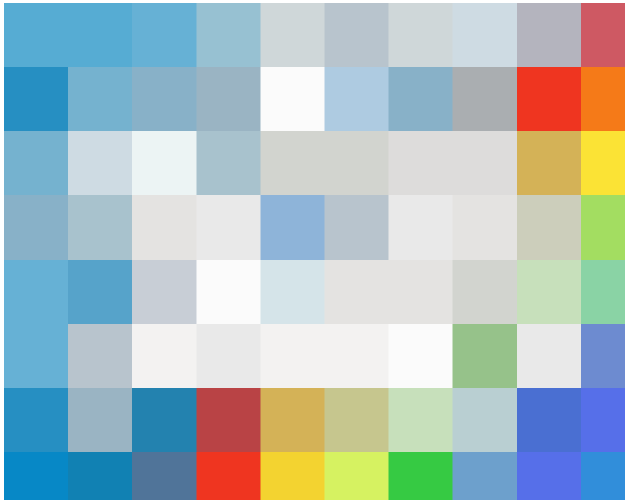
- **PNG**: stands for **Portable Network Graphics** — meant to supersede the GIF standard, and extends it in important ways.

- Special features of PNG files include:

1. Support for up to 48 bits of color information — a large increase.

2. Files may contain gamma-correction information for correct display of color images, as well as alpha-channel information for such uses as control of transparency.

3. The display progressively displays pixels in a 2-dimensional fashion by showing a few pixels at a time over seven passes through each 8 x 8 block of an image.



# 3.2.4 TIFF

- **TIFF**: stands for **Tagged Image File Format.**

- The support for attachment of additional information (referred to as "tags") provides a great deal of flexibility.

1. The most important tag is a format signifier: what type of compression etc. is in use in the stored image.
2. TIFF can store many different types of image: 1-bit, grayscale, 8-bit color, 24-bit RGB, etc.
3. TIFF was originally a lossless format but now a JPEG tag allows one to opt for JPEG compression.
4. The TIFF format was developed by the Aldus Corporation in the 1980's and was later supported by Microsoft.

# 3.2.5 Windows BMP

- *BitMap* (BMP), aka. *Device Independent Bitmap* (DIB) is a major system standard graphics file format for Microsoft Windows, recognized by many programs.

- It uses raster graphics.

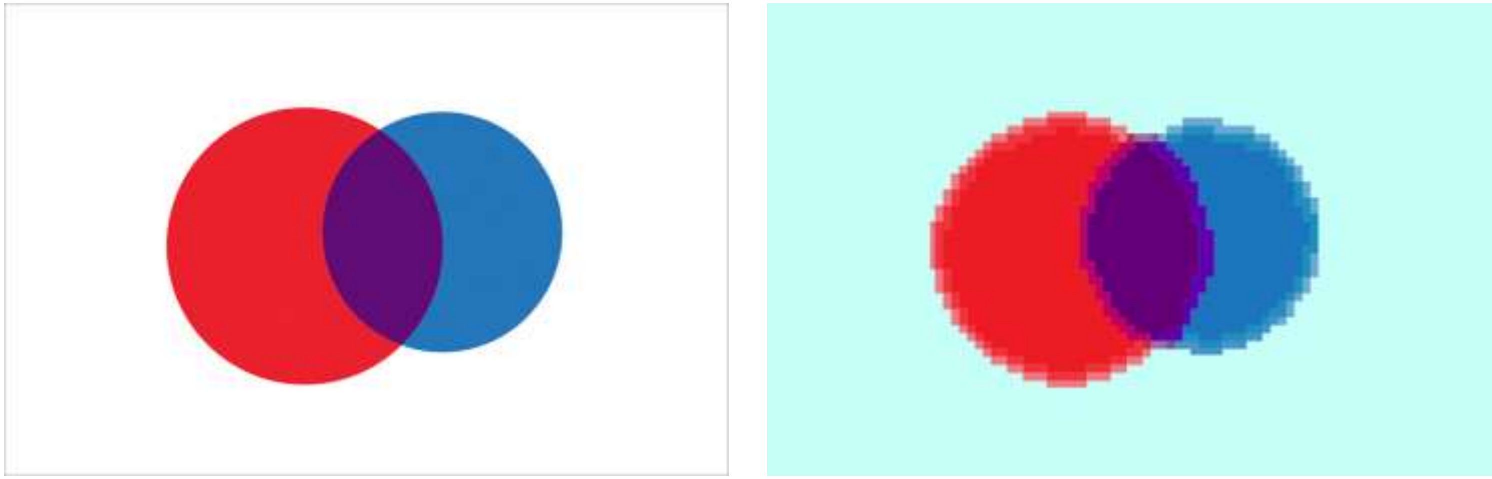
  - represents a two-dimensional picture as a rectangular matrix or grid of pixels, viewable via a computer display, paper, or other display medium

- BMP supports many pixel formats, including indexed color (up to 8 bits per pixel), and 16, 24, and 32-bit color images.

- There are many sub-variants within the BMP standard.

# Vector Images

Vector files are images that are built by mathematical formulas (specified as a code script) that establish points on a grid



# 3.2.6 Windows WMF

- The native vector file format for the Microsoft Windows operating environment:

1. Consist of a collection of GDI (Graphics Device Interface) function calls, also native to the Windows environment.
2. When a WMF file is "played" (typically using the Windows PlayMetaFile() function) the described graphics is rendered.
3. WMF files are ostensibly device-independent and are unlimited in size.

# 3.2.8 EXIF

- **EXIF** (Exchange Image File) is an image format for digital cameras:

  1. Compressed EXIF files use the baseline JPEG format.
  2. A variety of tags (many more than in TIFF) are available to facilitate higher quality printing, since information about the camera and picture-taking conditions (flash, exposure, light source, white balance, type of scene, etc.) can be stored and used by printers for possible color correction algorithms.
  3. The EXIF standard also includes specification of file format for audio that accompanies digital images.

# 3.2.9 HEIF

- *High Efficiency Image File* (HEIF) is the still-image version of the HEVC (or H. 265) video format.

- Adopted by Apple to replace JPEG in all of its iDevices. HEIF is more efficient than JPEG, i.e., the image size can be reduced by half with almost imperceptible changes to image quality.

- HEIF is said to be a container to wrap still images compressed with the HEVC codec.

# 3.2.10 PS and PDF

- **Postscript** is an important language for typesetting, and many high-end printers have a Postscript interpreter built into them.

- Postscript is a vector-based picture language, rather than pixel-based: page element definitions are essentially in terms of vectors.

  1. Postscript includes text as well as vector/structured graphics.
  2. Bit-mapped images can be included in output files.
  3. Encapsulated Postscript files (.EPS) add some additional information for inclusion of Postscript files in another document.

4. Postscript page description language itself does not provide compression; in fact, Postscript files are just stored as ASCII.

- Another text + figures language has superseded or at least paralleled Postscript: Adobe Systems Inc. includes LZW compression in its Portable Document Format (**PDF**) file format.

  - PDF files that do not include images have about the same compression ratio, 2:1 or 3:1, as do files compressed with other LZW-based compression tools.

# 3.2.11 PTM

- *Polynomial Texture Mapping (PTM), also known as Reflectance Transformation Imaging (RTI)* is a technique for storing a representation of a camera scene that contains information about a set of images taken under a set of lights that each have the same spectrum (say, a xenon flash), but with each light placed at a different direction from the scene.
- The method was originally developed by Tom Malzbender of HP Labs in order to generate enhanced 3D computer graphics and it has since been adopted for cultural heritage applications.

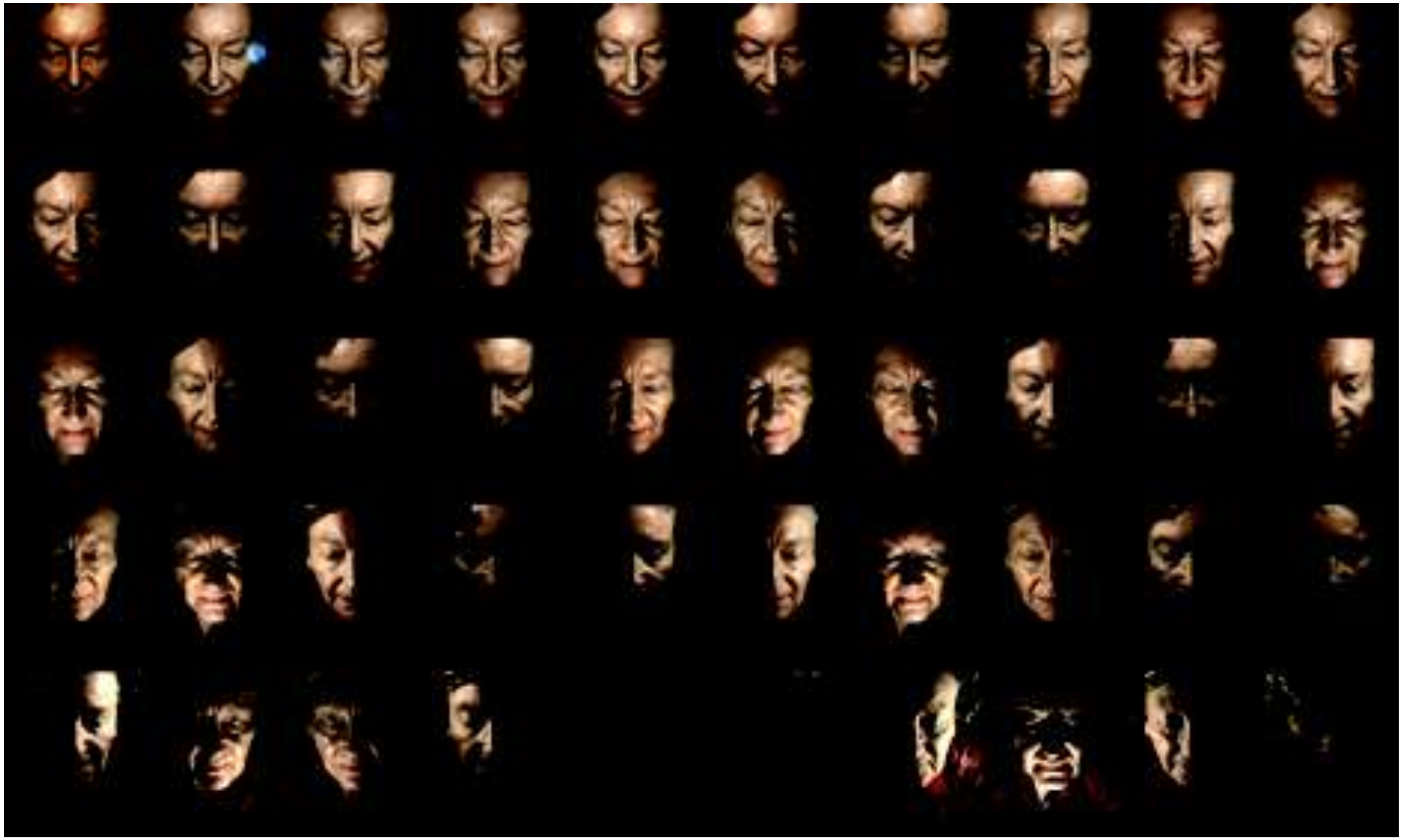
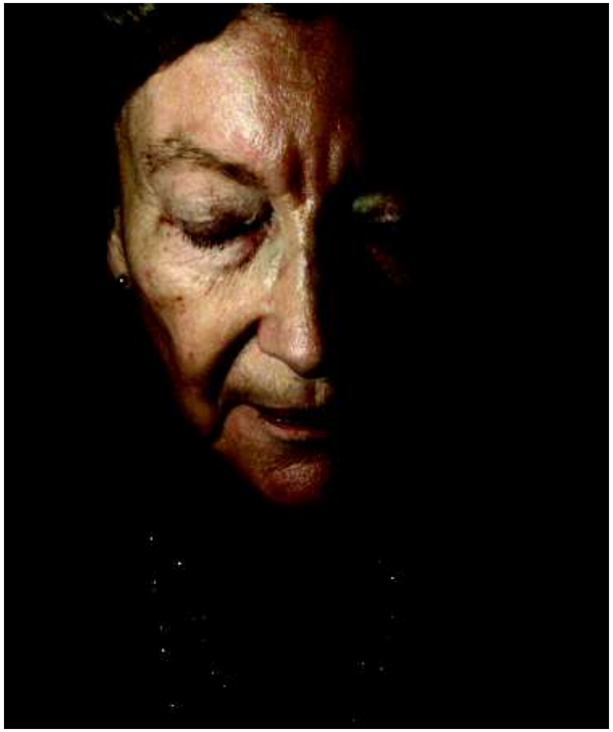


**Fig. 3.18:** (a) 50 input images for PTM: lights individually from 50 different directions ***e***$^i$, i=1..50; (b) interpolated image under *new* light ***e***.



(b)

**Fig. 3.18 (Cont’d):**

(a) 50 input images for PTM: lights individually from 50 different directions $\boldsymbol{e}^i$, i = 1..50;

(b) interpolated image under *new* light ***e***.