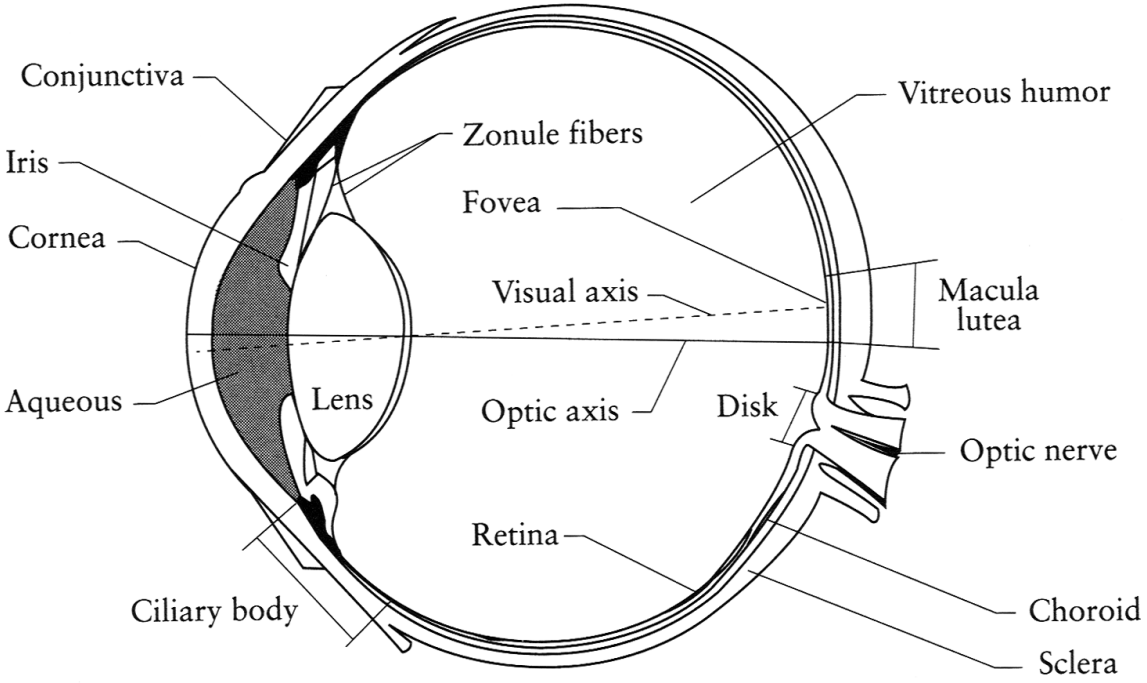
Lecture 3
Color &
Color Quantization

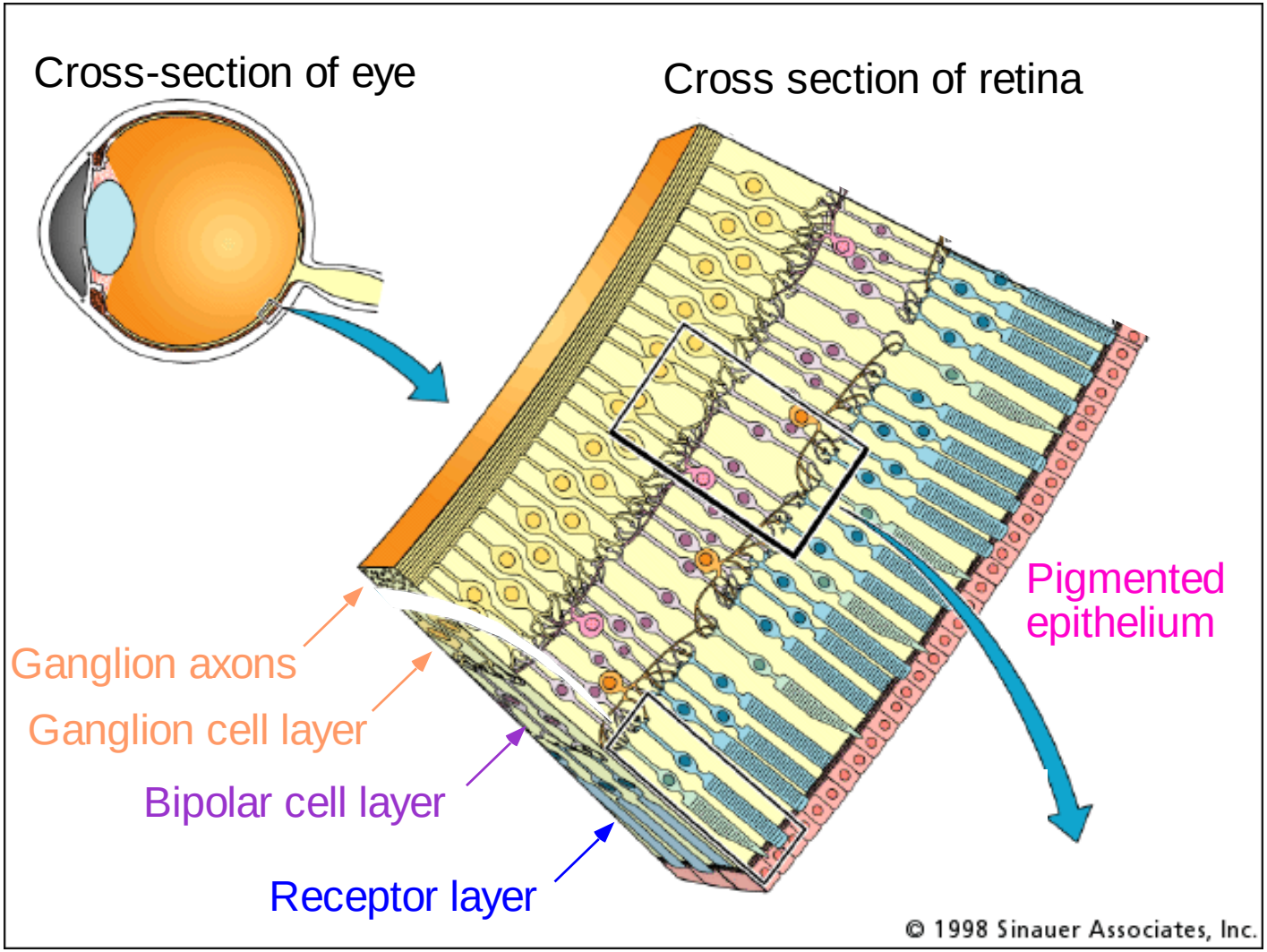
Multimedia Systems
Spring 2020

The Eye

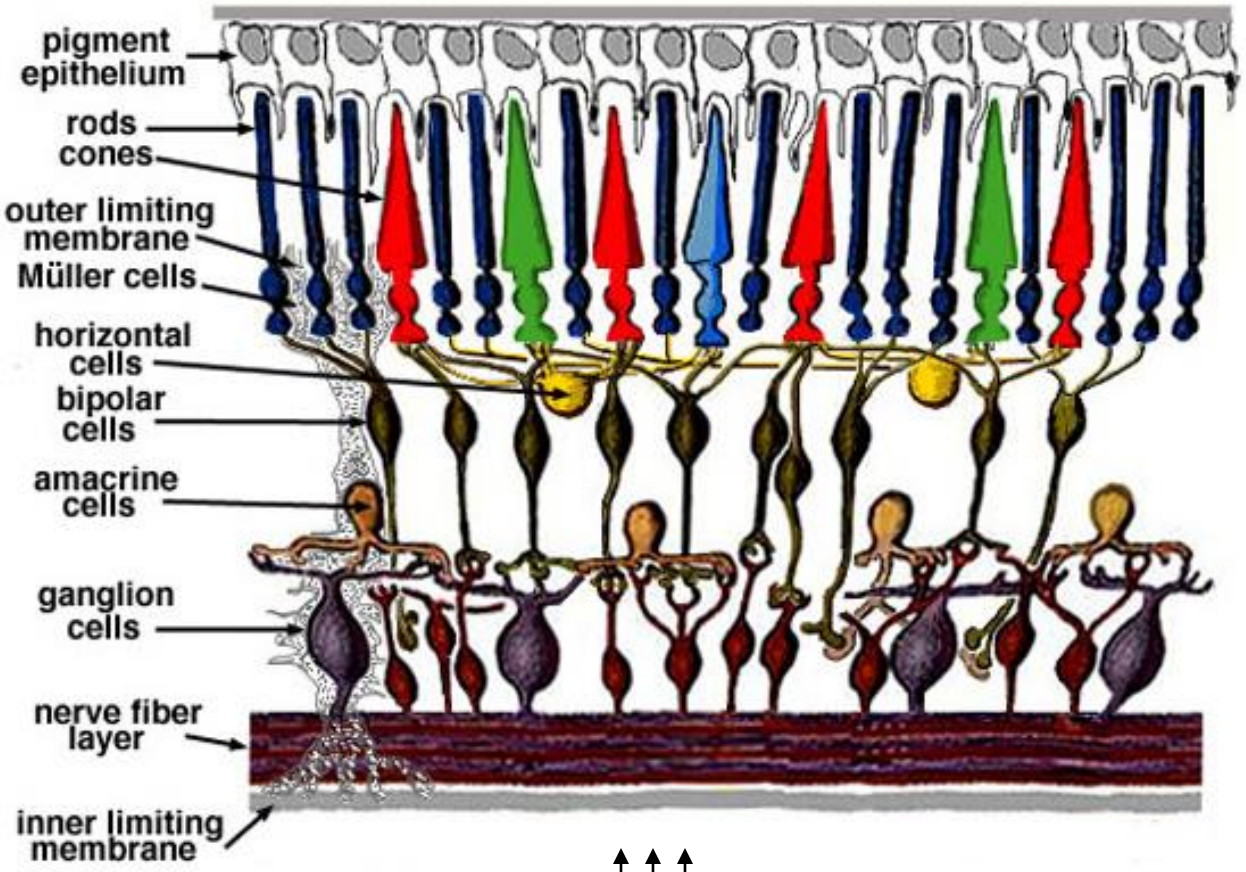


- The human eye is a camera!
  - **Iris** – colored annulus with radial muscles
  - **Pupil** – the hole (aperture) whose size is controlled by the iris
  - What's the "film"?
    - photoreceptor cells (rods and cones) in the **retina**

The Retina


Cross-section of eye
Cross section of retina
Pigmented epithelium
Ganglion axons
Ganglion cell layer
Bipolar cell layer
Receptor layer
© 1998 Sinauer Associates, Inc.

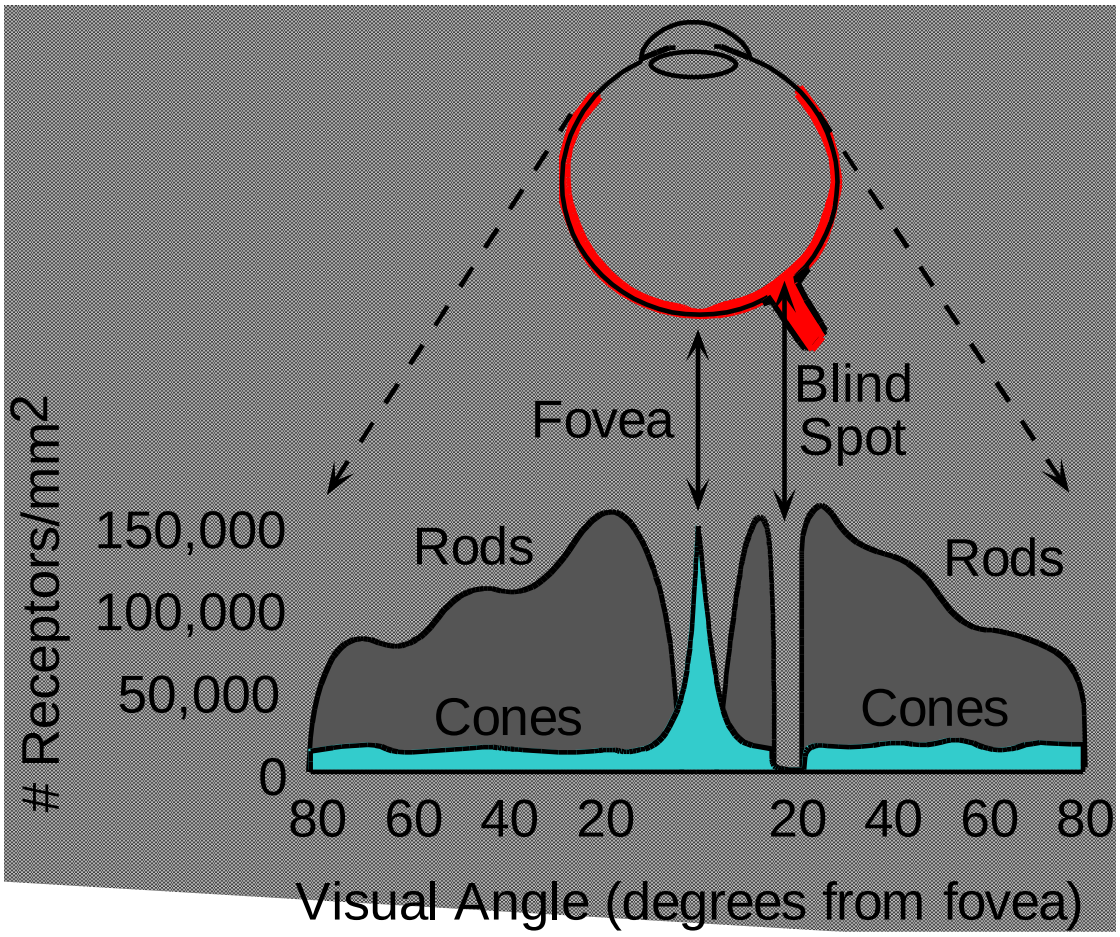
Retina up-close



↑↑↑

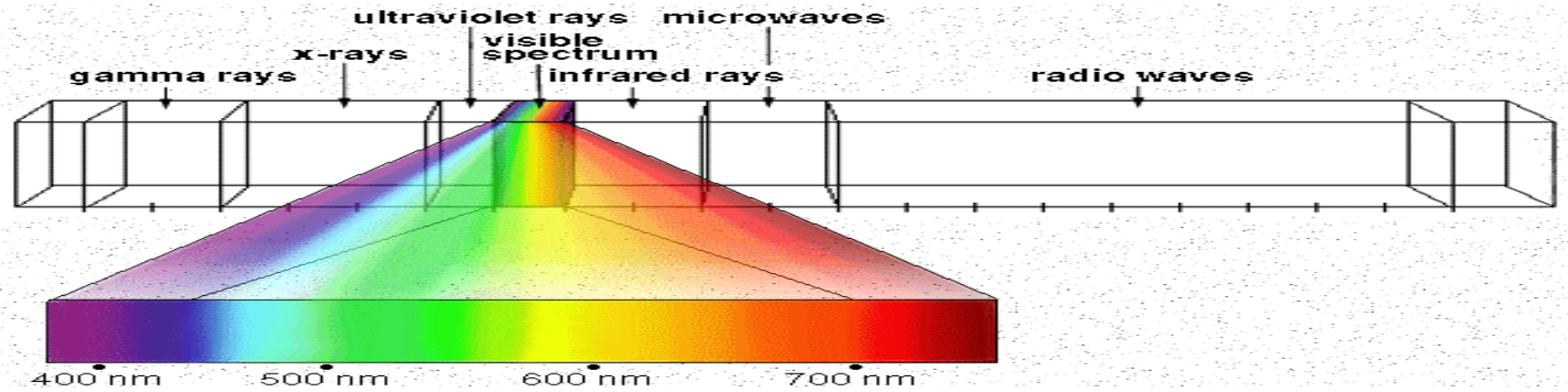
Light

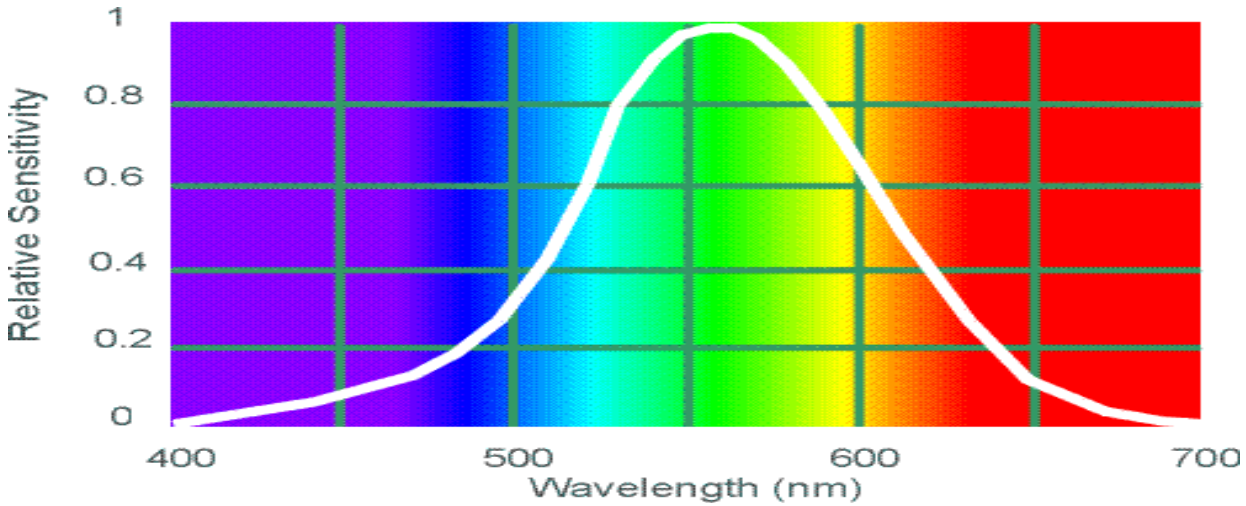
Distribution of Rods and Cones



Night Sky: why are there more stars, off-center?

Electromagnetic Spectrum





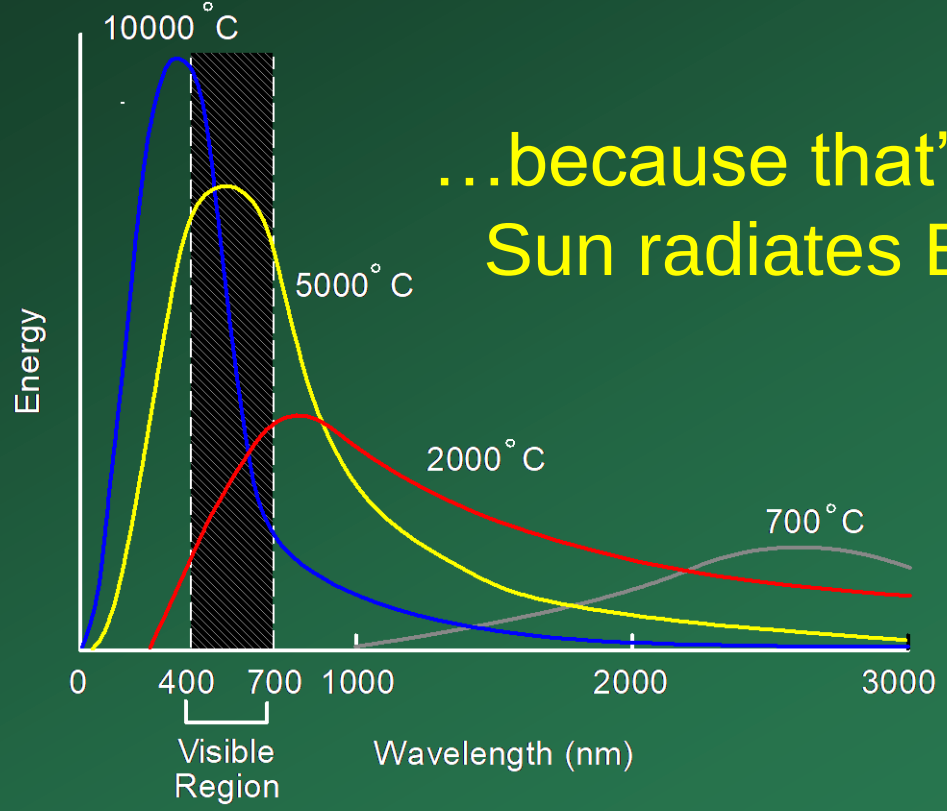
Human Luminance Sensitivity Function

http://www.yorku.ca/eye/photopik.htm

Visible Light

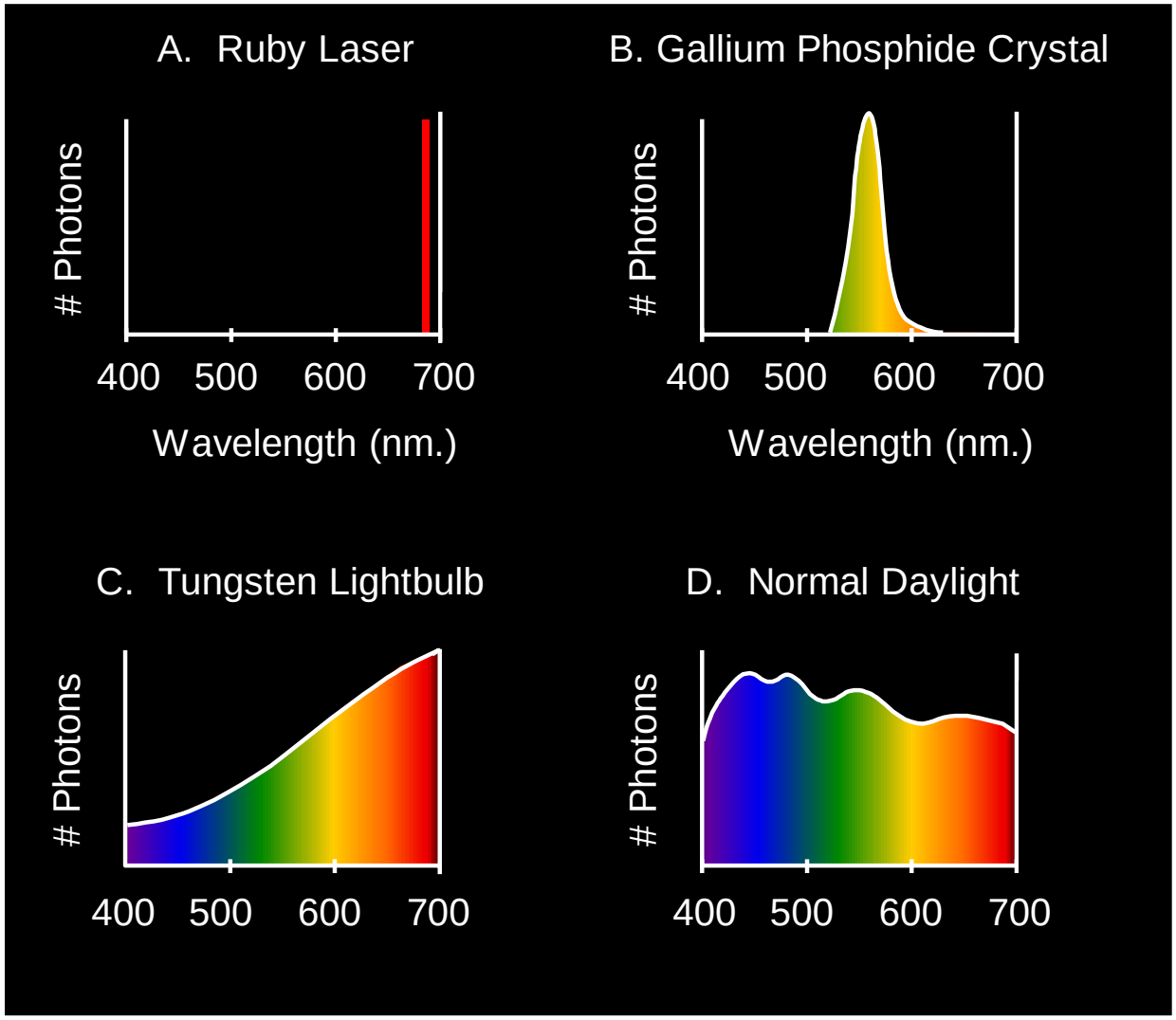
Plank's law for Blackbody radiation
Surface of the sun: ~5800K

Why do we see light of these wavelengths?



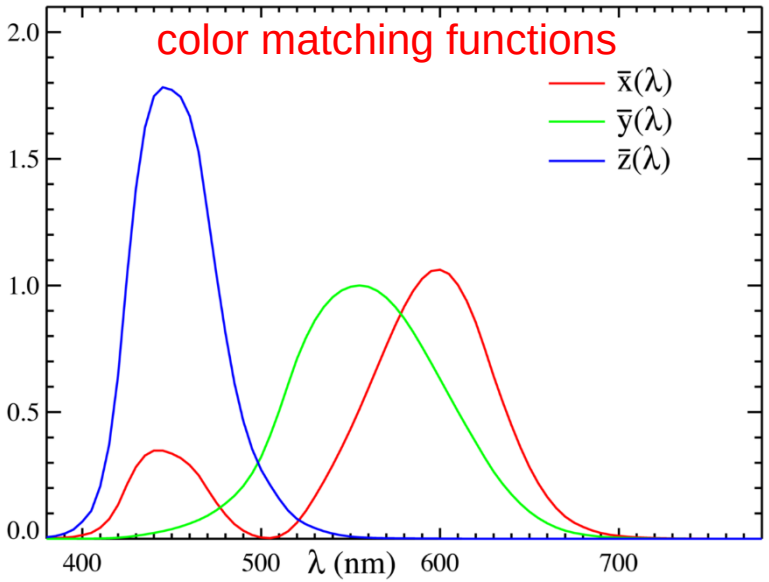
…because that's where the Sun radiates EM energy

The Physics of Light



CIE Color Space

- *CIE* (Commission Internationale d'Eclairage) came up with 3 hypothetical lights X, Y, and Z with these spectra:



spectral power distribution of the illuminant

$$X = \int_0^\infty I(\lambda)\,\bar{x}(\lambda)\,d\lambda$$

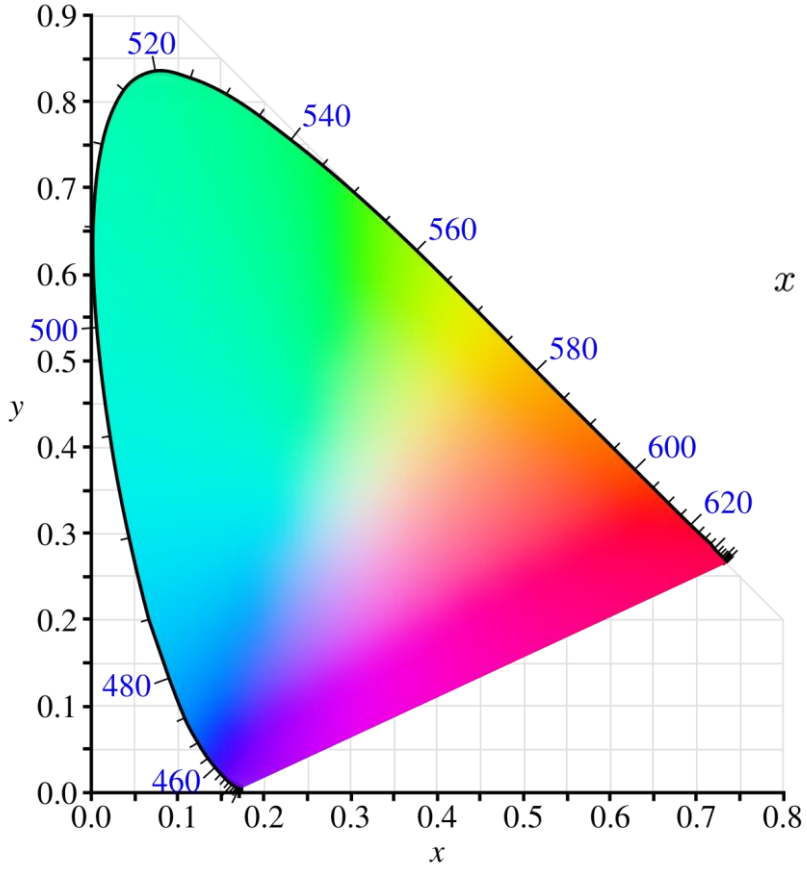
$$Y = \int_0^\infty I(\lambda)\,\bar{y}(\lambda)\,d\lambda$$

$$Z = \int_0^\infty I(\lambda)\,\bar{z}(\lambda)\,d\lambda$$

- **Idea:** any wavelength λ can be matched perceptually by *positive* combinations of X,Y,Z
  - CIE created table of XYZ values for all *visible* colors

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

CIE Chromaticity Diagram (1931)



- For simplicity, we often project to the 2D plane
- Also normalize

$$x + y + z = 1$$

$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z} \qquad z = \frac{Z}{X+Y+Z}$$

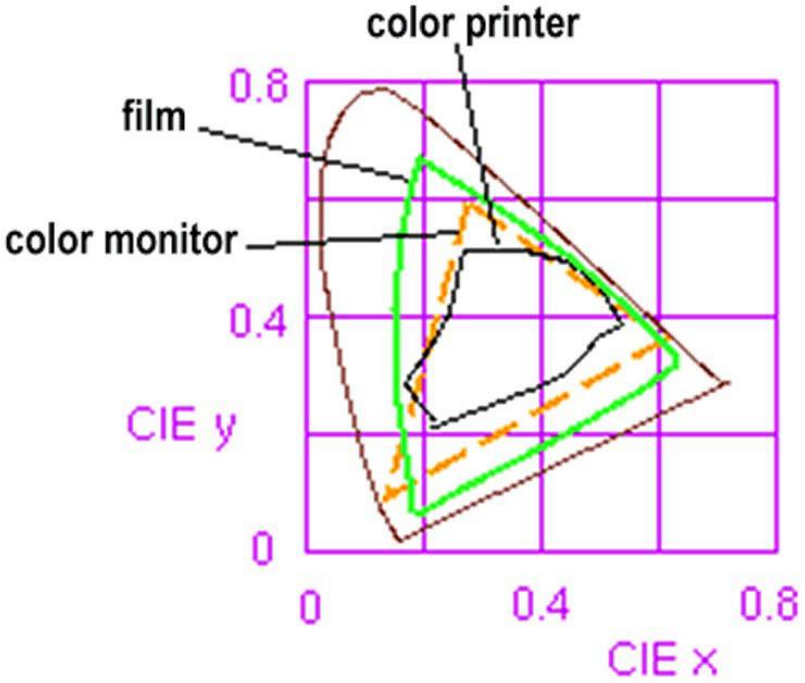
- Note: Inside horseshoe visible, outside invisible to eye

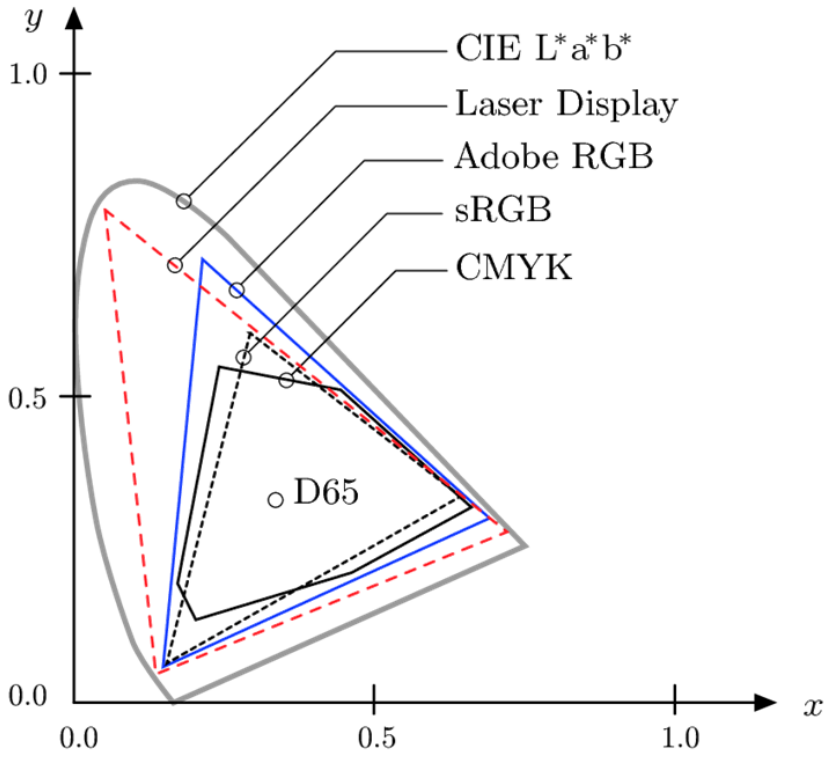
Note: Look up *x, y*
Calculate z as *1 – x - y*

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

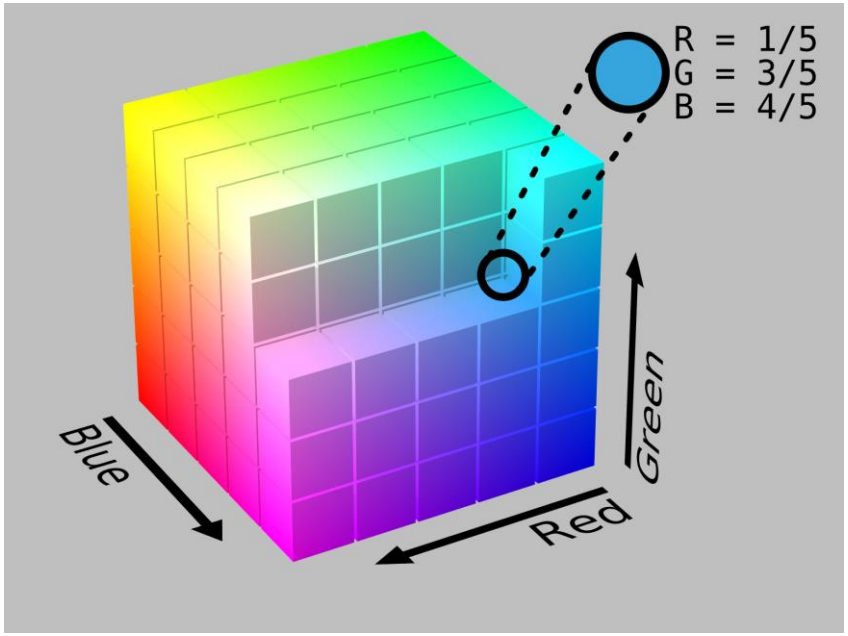
Device Color Gamuts

- The RGB color cube sits within CIE color space
- We can use the CIE chromaticity diagram to compare the gamuts of various devices
- E.g. compare color printer and monitor color gamuts





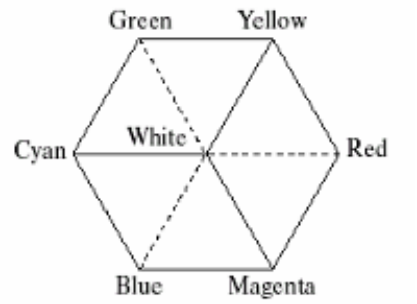
RGB color space



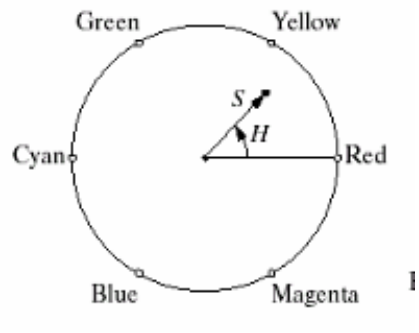
- RGB cube
  - Easy for devices
  - But not perceptual
  - Where do the grays live?
  - Where is hue and saturation?

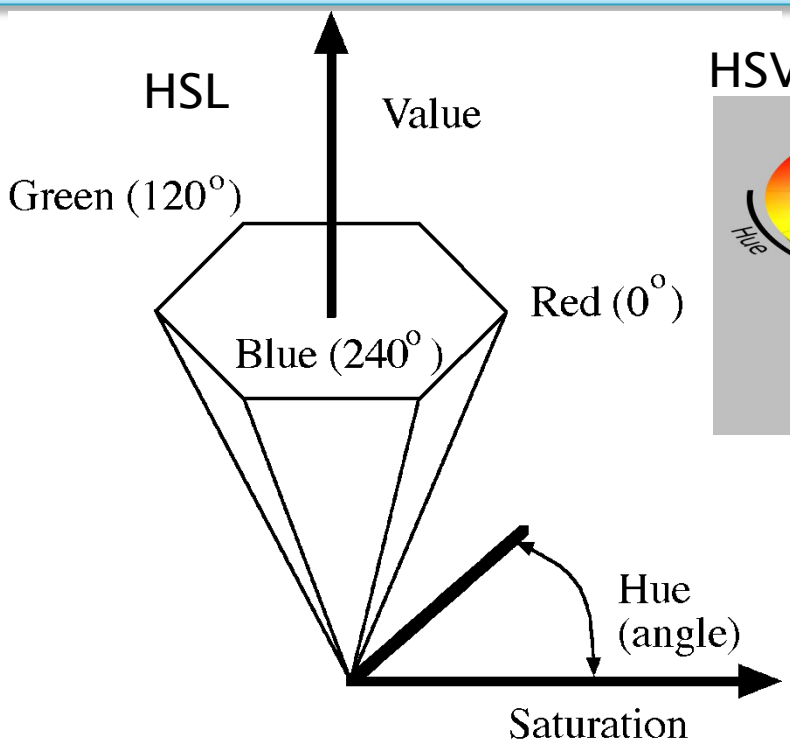
HSV

HSL

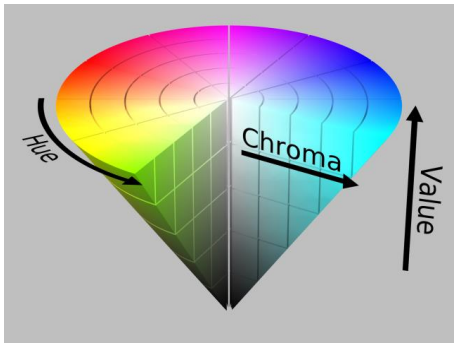


HSV





HSV



- Hue, Saturation, Value (Intensity)
  - RGB cube on its vertex
- Decouples the three components (a bit)
- Use rgb2hsv() and hsv2rgb() in Matlab

RGB to HSV Conversion

- Define the following values

$$C_{\text{high}} = \max(R, G, B), \quad C_{\text{low}} = \min(R, G, B), \quad \text{and}$$
$$C_{\text{rng}} = C_{\text{high}} - C_{\text{low}}.$$

- Find saturation of RGB color components (C_{max}= 255)

$$S_{\text{HSV}} = \begin{cases} \frac{C_{\text{rng}}}{C_{\text{high}}} & \text{for } C_{\text{high}} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- And luminance (Value) value

$$V_{\text{HSV}} = \frac{C_{\text{high}}}{C_{\text{max}}}$$

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

RGB to HSV Conversion

- Normalize each component using

$$R' = \frac{C_{\text{high}} - R}{C_{\text{rng}}} \qquad G' = \frac{C_{\text{high}} - G}{C_{\text{rng}}} \qquad B' = \frac{C_{\text{high}} - B}{C_{\text{rng}}}$$

- Calculate preliminary hue value *H'* as

$$H' = \begin{cases} B' - G' & \text{if } R = C_{\text{high}} \\ R' - B' + 2 & \text{if } G = C_{\text{high}} \\ G' - R' + 4 & \text{if } B = C_{\text{high}} \end{cases}$$

- Finally, obtain final hue value by normalizing to interval [0,1]

$$H_{\text{HSV}} = \frac{1}{6} \cdot \begin{cases} (H' + 6) & \text{for } H' < 0 \\ H' & \text{otherwise} \end{cases}$$

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

RGB to HSV Conversion

Original RGB image

H_{HSV} S_{HSV} V_{HSV}

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

TV Color Spaces – YUV, YIQ, YC_bC_r

- YUV, YIQ: color encoding for analog NTSC and PAL
- YC_bC_r: Digital TV encoding
- Key common ideas:
  - Separate luminance component Y from 2 chroma components
  - Instead of encoding colors, encode color differences between components (maintains compatibility with black and white TV)

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

YUV

- Basis for color encoding in analog TV in north america (NTSC) and Europe (PAL)
- Y components computed from RGB components as

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

- UV components computed as:

$$U = 0.492 \cdot (B - Y) \qquad \text{and} \qquad V = 0.877 \cdot (R - Y)$$

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

YUV

- Entire transformation from RGB to YUV

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- Invert matrix above to transform from YUV back to RGB

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.000 & 0.000 & 1.140 \\ 1.000 & -0.395 & -0.581 \\ 1.000 & 2.032 & 0.000 \end{pmatrix} \cdot \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

YIQ

- Original NTSC used variant of YUV called YIQ
- *Y* component is same as in YUV
- Both U and V color vectors rotated and mirrored so that

$$\begin{pmatrix} I \\ Q \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{pmatrix} \cdot \begin{pmatrix} U \\ V \end{pmatrix}$$

2D rotation matrix

where β = 0.576 (33 degrees)

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

YC_bC_r

- Internationally standardized variant of YUV
- Used for digital TV and image compression (e.g. JPEG)
- *Y*, C_b, C_r components calculated as

$$Y = w_R \cdot R + (1 - w_B - w_R) \cdot G + w_B \cdot B$$

$$C_b = \frac{0.5}{1 - w_B} \cdot (B - Y)$$

$$C_r = \frac{0.5}{1 - w_R} \cdot (R - Y)$$

- Inverse transform from YC_bC_r to RGB

$$R = Y + \frac{1 - w_R}{0.5} \cdot C_r$$

$$G = Y - \frac{w_B \cdot (1 - w_B) \cdot C_b \; - \; w_R \cdot (1 - w_R) \cdot C_r}{0.5 \cdot (1 - w_B - w_R)}$$

$$B = Y + \frac{1 - w_B}{0.5} \cdot C_b$$

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

YC_bC_r

- ITU recommendation BT.601 specifies values:

  $w_R = 0.299$, $w_B = 0.114$, $w_G = 1 - w_B - w_R = 0.587$

- Thus the transformation

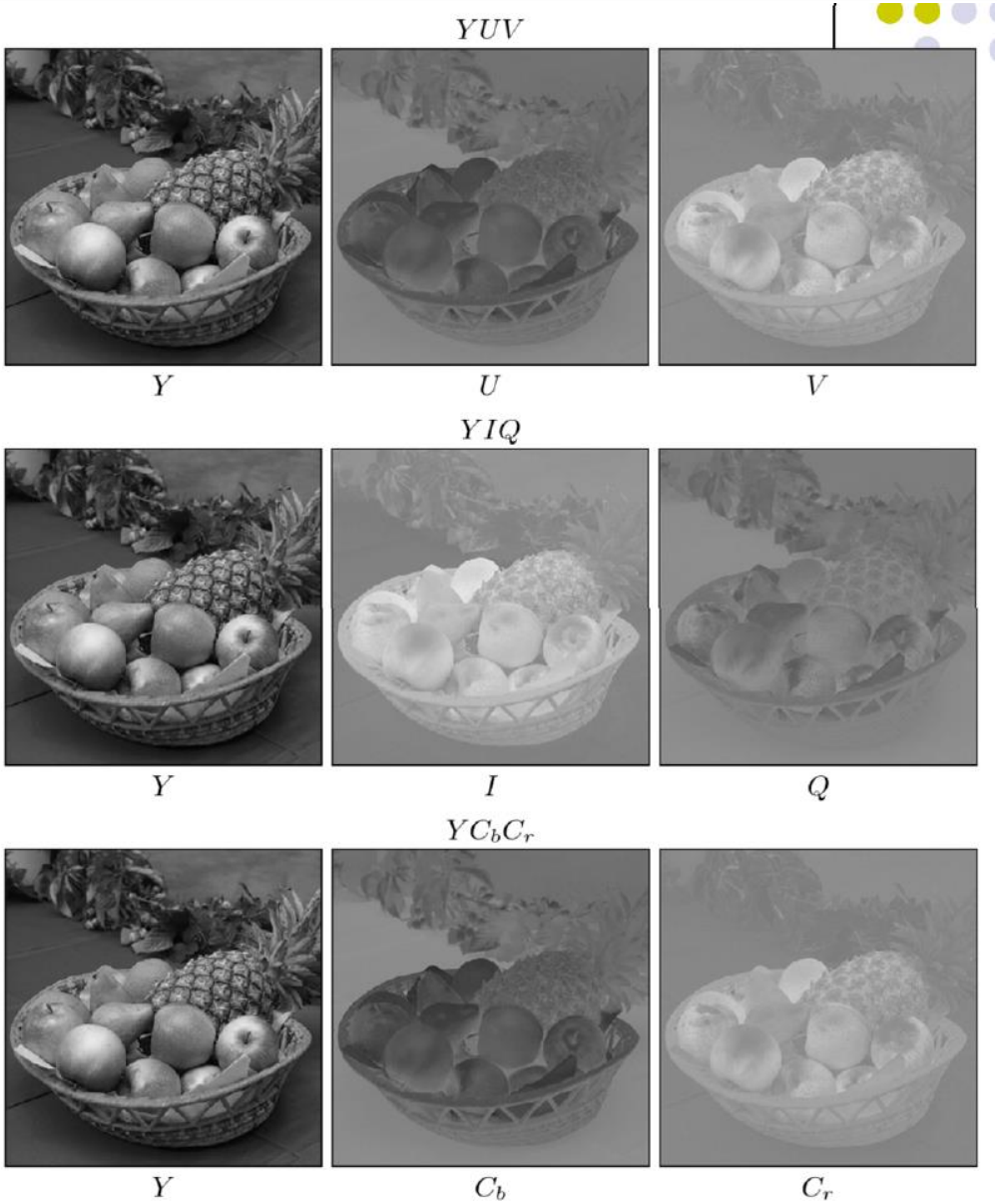
$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- And the inverse transformation becomes

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.000 & 0.000 & 1.403 \\ 1.000 & -0.344 & -0.714 \\ 1.000 & 1.773 & 0.000 \end{pmatrix} \cdot \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}$$

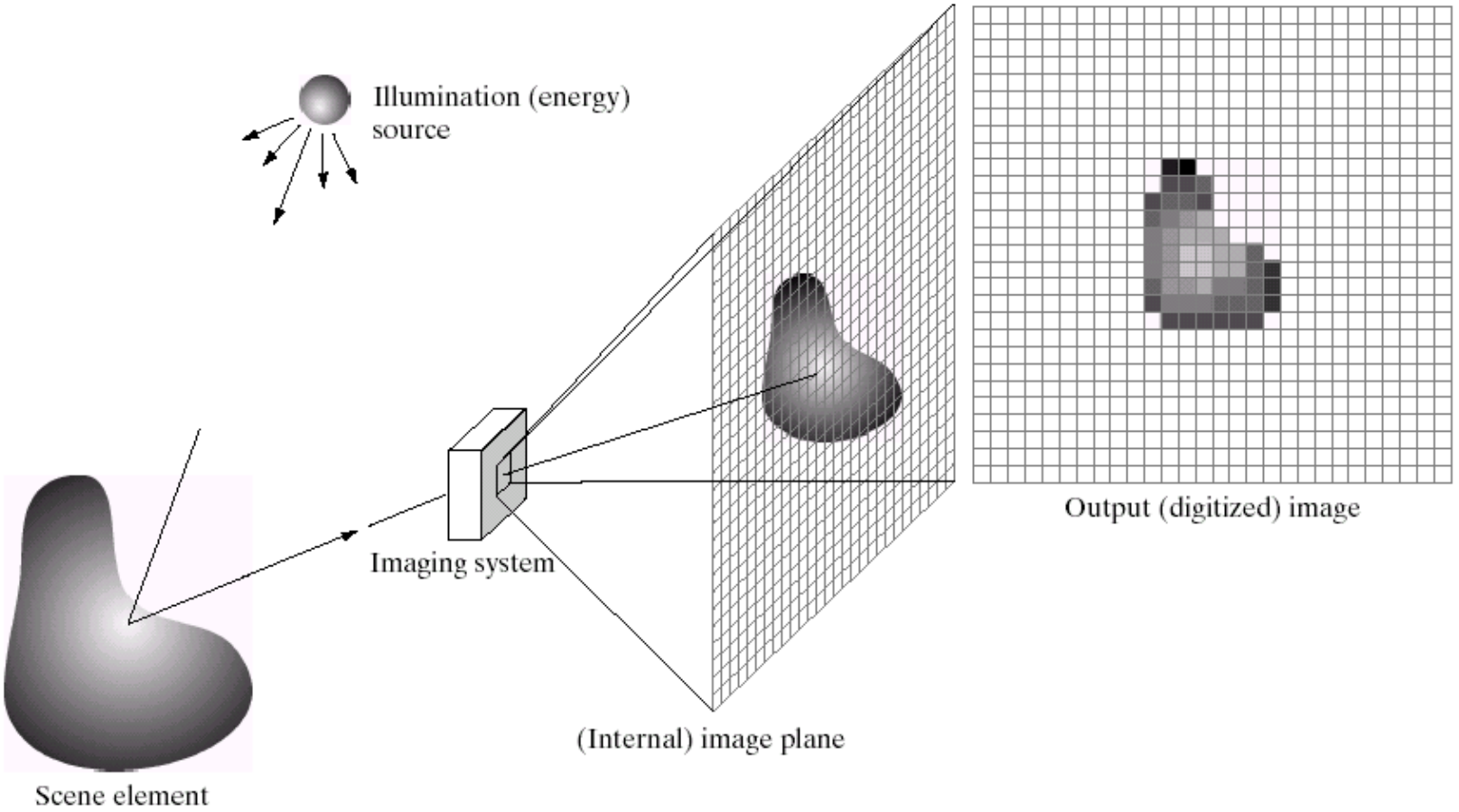
Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

Comparing YUV, YIQ and YC_bC_r



Color Quantization

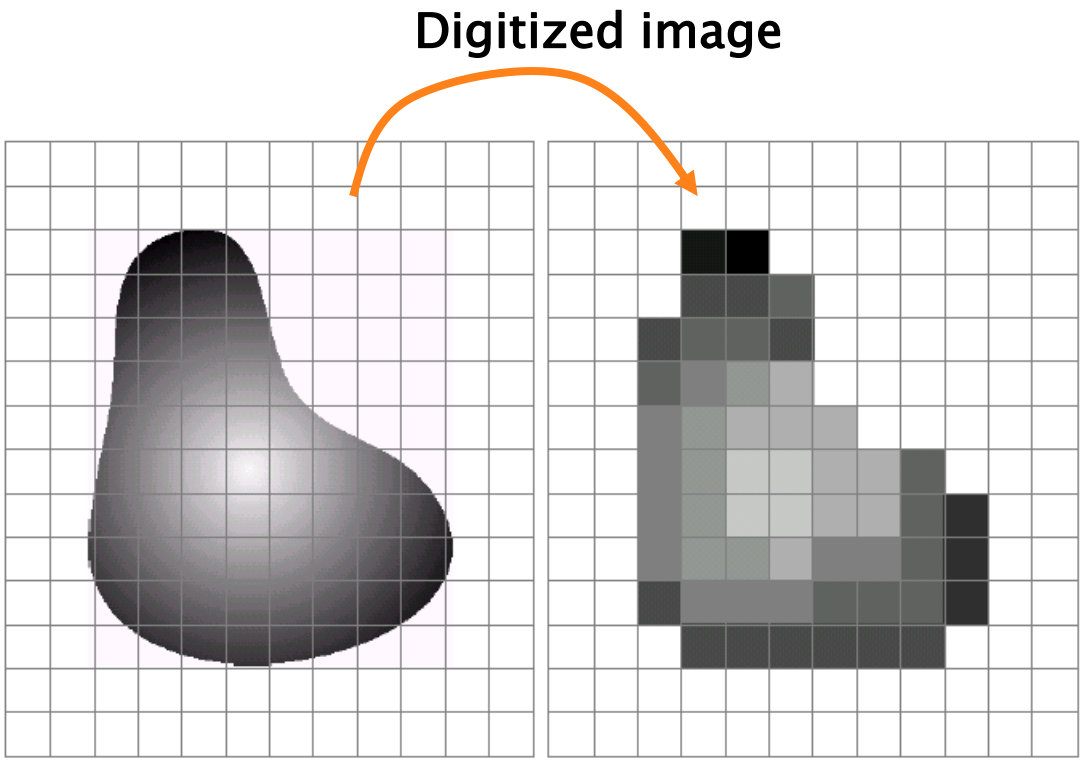
Image formation



a
b c d e

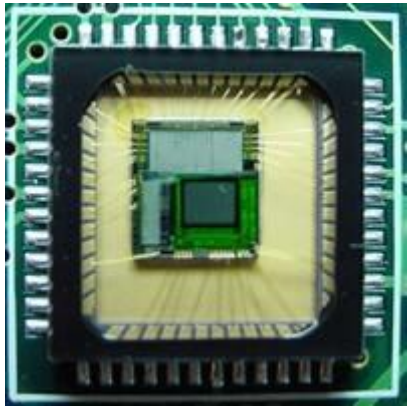
FIGURE 2.15 An example of the digital image acquisition process. (a) Energy ("illumination") source. (b) An element of a scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

Sensor Array



a b

FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.



CMOS sensor

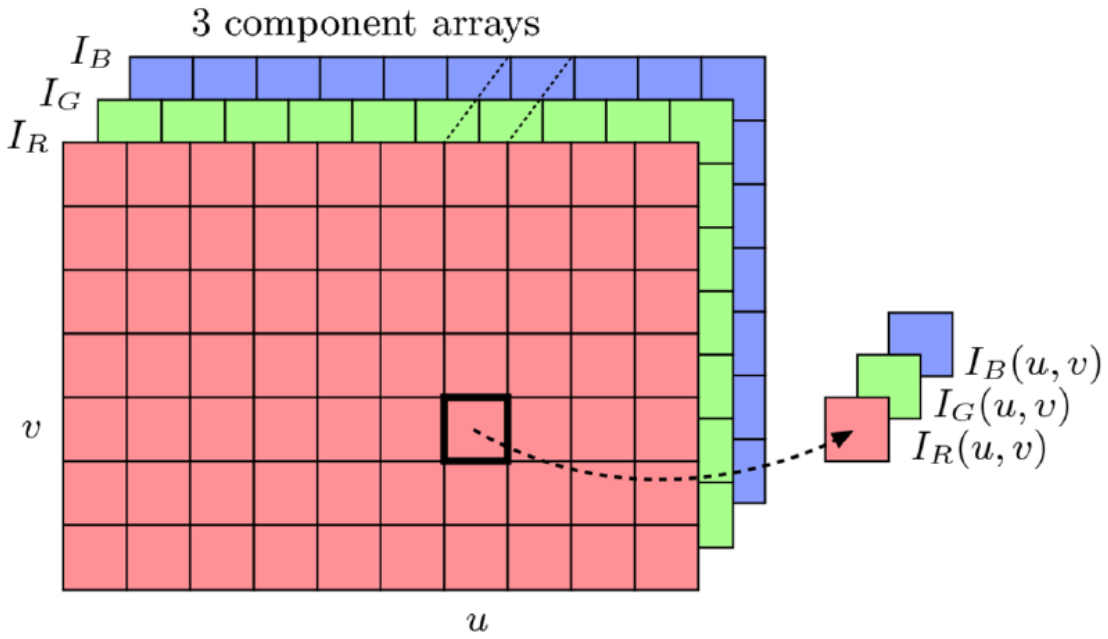
Color Quantization

- **True color**: Uses all colors in color space
- **Indexed color**: Uses only some colors
  - Which subset of colors to use? Depends on application
- True color:
  - used in applications that contain many colors with subtle differences
  - E.g. digital photography or photorealistic rendering
- Two main ways to organize true color
  - Component ordering
  - Packed ordering

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

True Color: Component Ordering

- Colors in 3 separate arrays of similar length
- Retrieve same location (u,v) in each R, G and B array

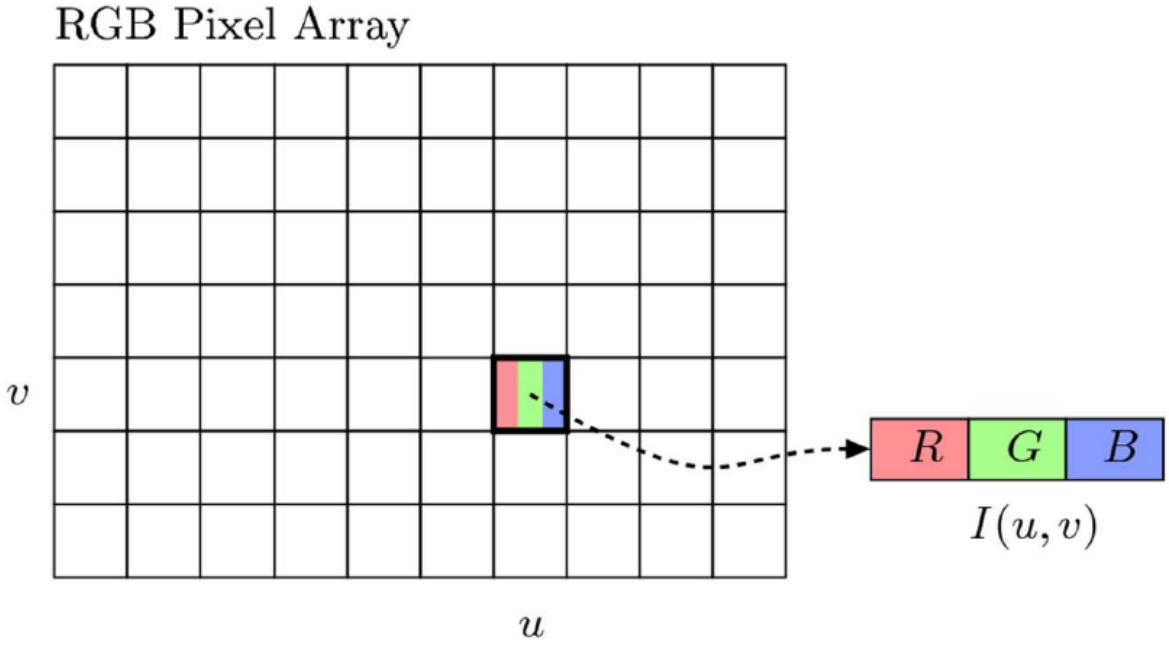


$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} \leftarrow \begin{pmatrix} I_R(u,v) \\ I_G(u,v) \\ I_B(u,v) \end{pmatrix}$$

Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

True Color: Packed Ordering

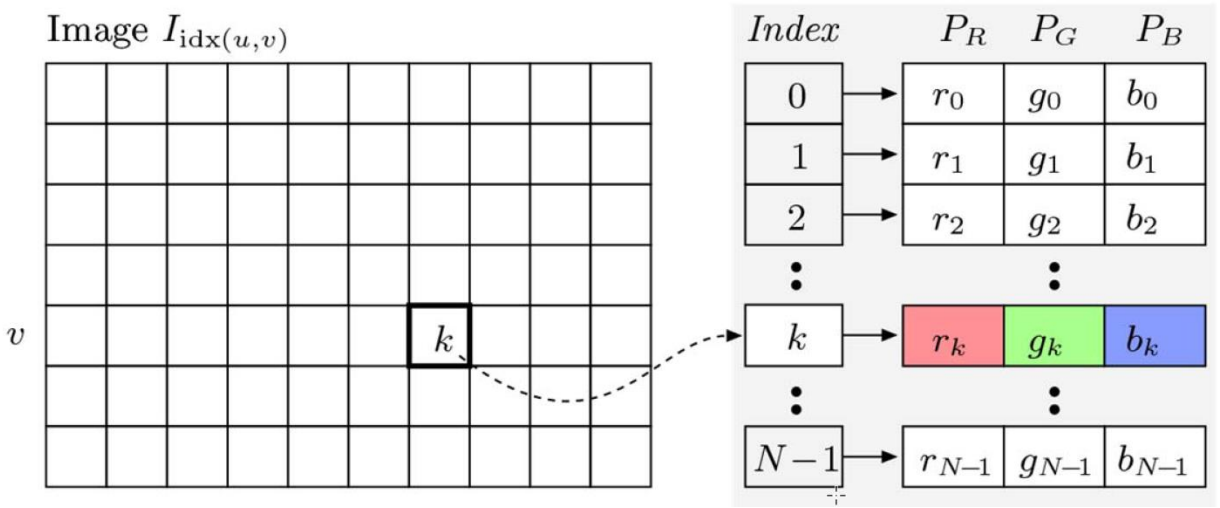
- Component (RGB) values representing a pixel's color is packed together into single element



Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

Indexed color

- Permit only limited number of distinct colors (N = 2 to 256)
- Used in illustrations or graphics containing large regions with same color
- Instead of intensity values, image contains indices into color table or palette
- Palette saved as part of image
- Converting from true color to indexed color requires quantization



Slide from Prof Emmanuel Agu, Computer Science Dept. Worcester Polytechnic Institute (WPI)

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).

Original image

Monochrome 1-bit Lena image.

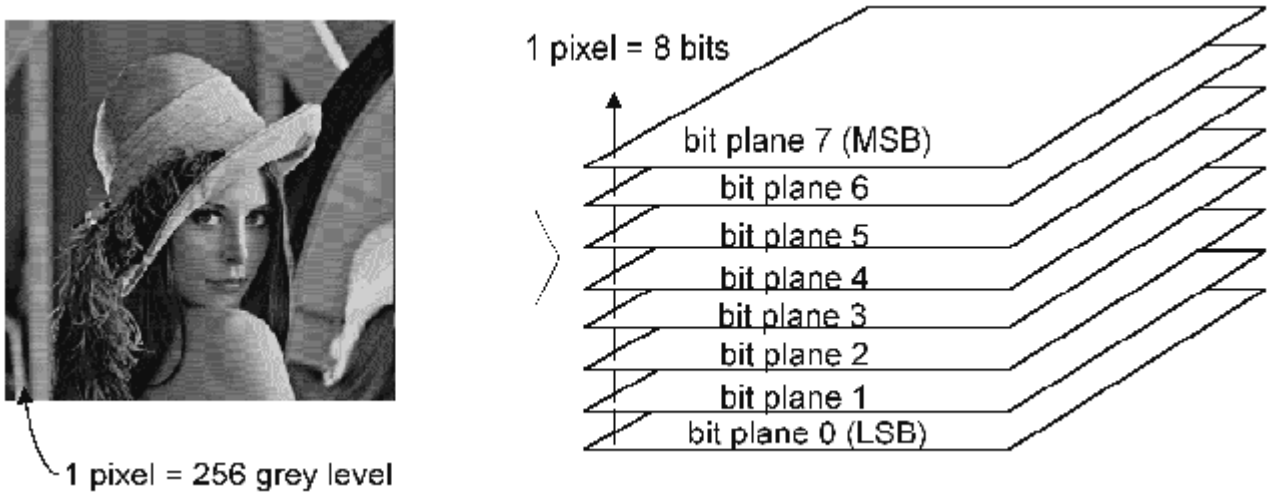
Fig. 3.1

8-bit Gray-level Images

- Each pixel has a gray-value between 0 and 255.
  - Represented by a single byte
- **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.
- **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.

8-bit Gray-level Images

- **bit-planes:**
  - Consider the 8-bit image as a set of 1-bit bitplanes
  - Each plane consists of a 1-bit representation of the image
- Graphical concept of bit-planes



Bit-planes for 8-bit grayscale image.

Multimedia Presentation

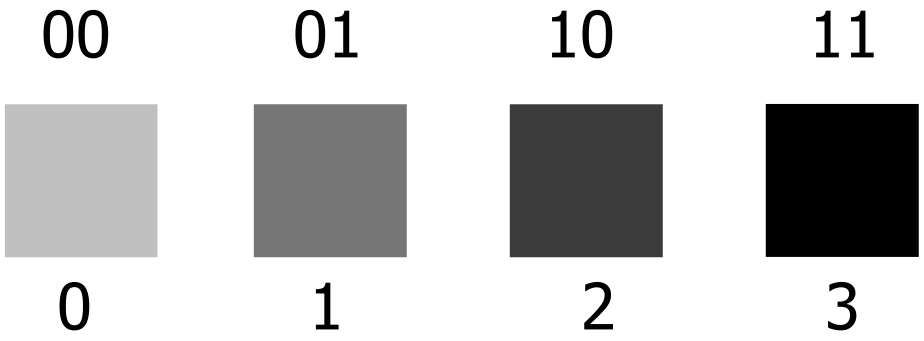
- Each pixel is usually stored as a byte (a value between 0 to 255), so a 640 x 480 grayscale image requires 300 kB of storage (640 x 480 = 307,200).

- Dithering : 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.

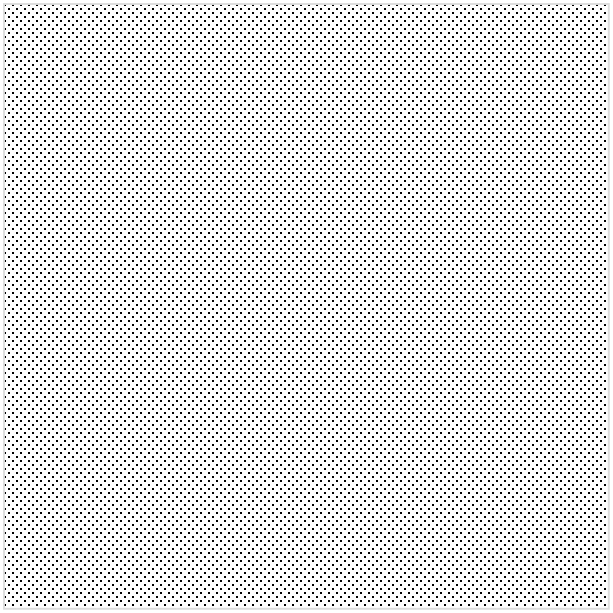
Dither algorithm

- 2-bit image with 4 gray levels.



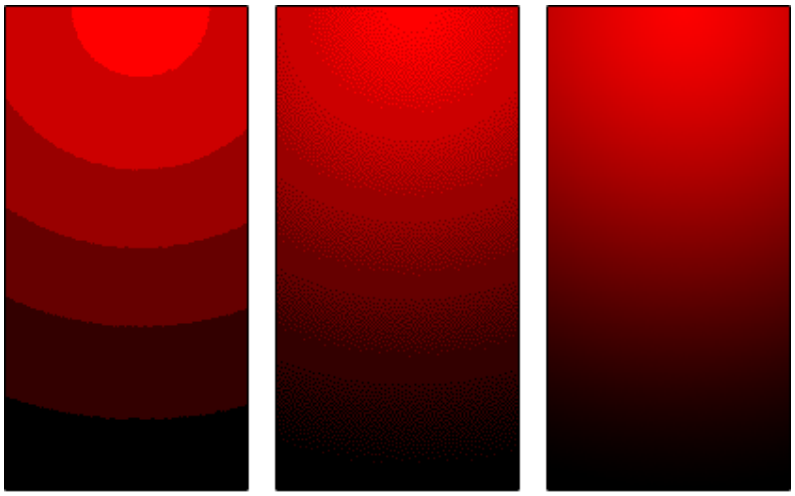
- 1-bit printer with 2 gray levels.

Dither algorithm

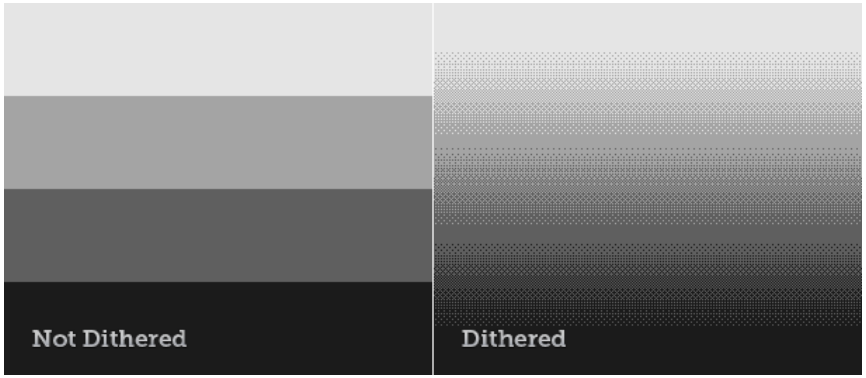


Dither algorithm

- Example of dithering



Display in dark grey levels



Display using low-bit grey levels

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 2x2 or 4x4, 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 2x2 **dither matrix**

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

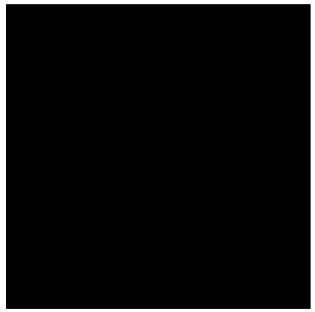
Ordered dither algorithm

- Print 2-bit image using 1-bit printer based on the dither matrix

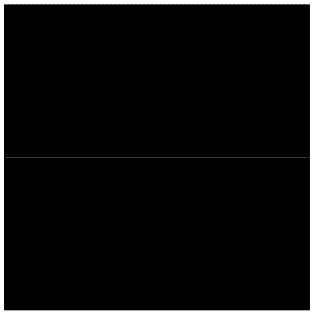
Dither Matrix

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

Image



Dither result



Ordered dither algorithm

- 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 2x2 area of printer output. But if the pixel value is 4 we print all four dots.
- The rule is:

  If (intensity > the dither matrix entry) 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.

Ordered dither algorithm

- A clever trick can get around this problem. Suppose we wish to use a larger, 4x4 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 output 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).

Ordered dither algorithm

- 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)/16 is the input, O(x, y) is the output,
            // D is the dither matrix.
            if I(x, y) > D(i, j)
                O(x, y) = 1;
            else
                O(x, y) = 0;
END
```

Ordered dither algorithm

(a) (b) (c)

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.

- Reference: http://www.cs.indiana.edu/~dmiguse/Halftone/
  http://www.lenna.org

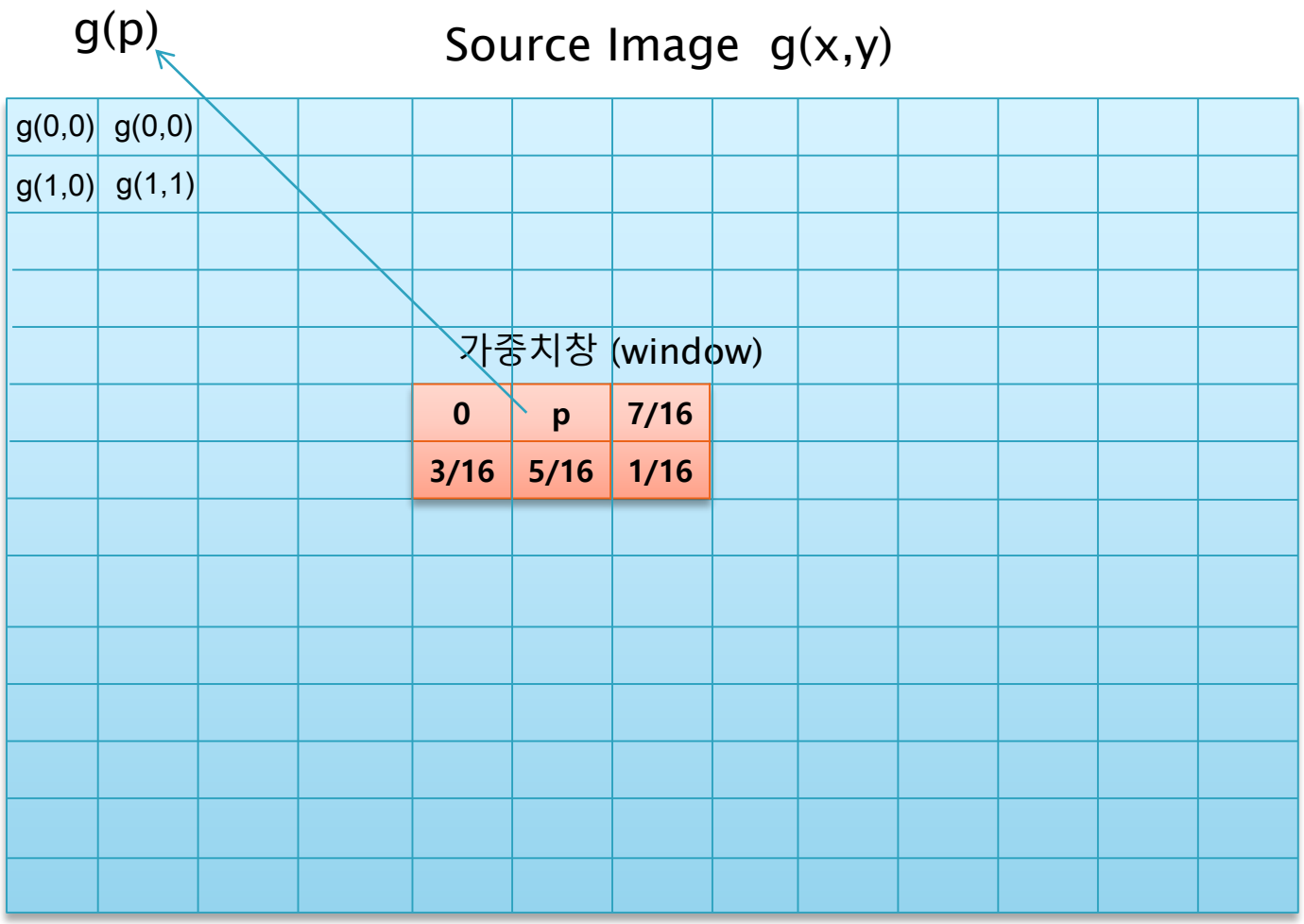
Error diffusion dithering

- 오차확산 디더링은 픽셀값과 가까운 최대, 최소값 (255 또는 0) 과의 오차를 이웃한 픽셀로 확산하는 알고리듬이다. 그림에서 픽셀 p의 오차는 이웃하는 픽셀로 오차가 그림과 같은 가중치를 가지고 확산된다.

0	p	7/16
3/16	5/16	1/16

- 픽셀 p의 값이 g(p)라 할 때 g(p)가 127보다 크면 g(p)-255가 오차가 되고, 작으면 g(p)가 오차가 된다.
- 오차에 가중치를 곱하여 원영상의 위치 p의 주변 픽셀에 더해주는 과정을 모든 픽셀에 대하여 처리하면 첫 단계가 종료된다.
- 두 번째 과정은 각 픽셀의 값이 127보다 적으면 0으로 127보다 크면 1로 변환한다.

Error diffusion dithering



Comparison of different dither algorithms

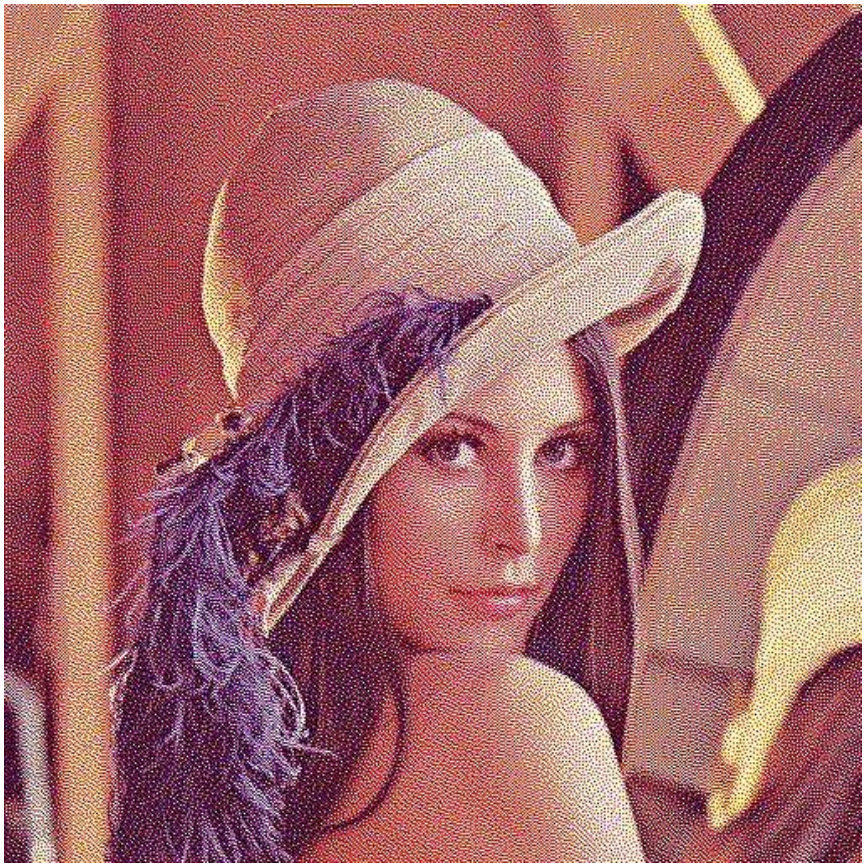
Original one
(8-bit)

Ordered dither
(1-bit)

Error diffusion dither
(1-bit)

Color Dithering (Error Diffusion)

Original image



Dithered image

Color Image 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**.

24-bit Color Images

- In a color 24-bit image, each pixel is represented by three bytes, usually representing RGB.
  - This format supports 256x256x256 possible colors
  - 640x480 24-bit color image requires 921.6 kB
- **An important point**: 24-bit color images are stored as 32-bit images, an *alpha* value representing special effect information
- Fig. 3.5 shows the image 'forestfire.bmp', a 24-bit image in Microsoft Windows BMP format. Also shown are the grayscale images for just 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.

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. 3.6 shows a 3D histogram of the RGB values of the pixels in “forestfire.bmp”.

8-bit Color Images

- 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 640x480 8-bit color image only requires 300 kB of storage, compared to 921.6 kB for a color image (again, without any compression applied).

Color Quantization

- What is color quantization?
  - A process uses a limited number of colors to represent an image.
  - The set of limited color is called a color palette.

- Application
  - displaying images with low--end display units
  - playing video clips with low--end display units
  - compressing images & video clips

Color Look-up Tables (LUTs)

- The idea used in 8-bit color images is to store only the index, or code values, 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).

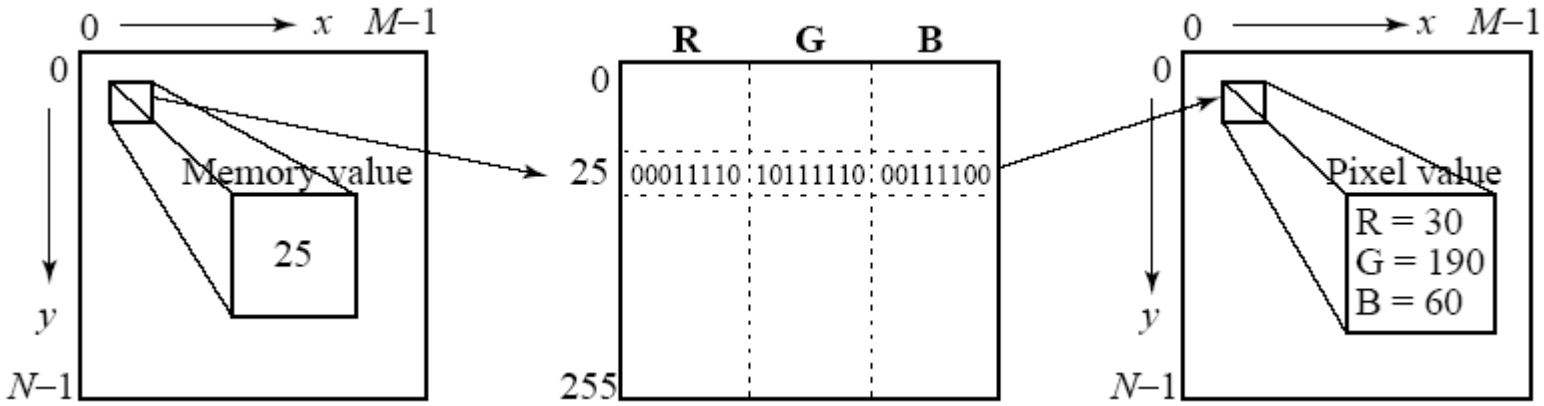


Fig. 3.8: Color LUT for 8-bit color images.

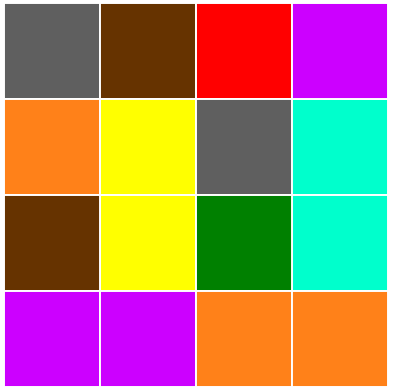
Indexed Color

Color Table

Pixel Data

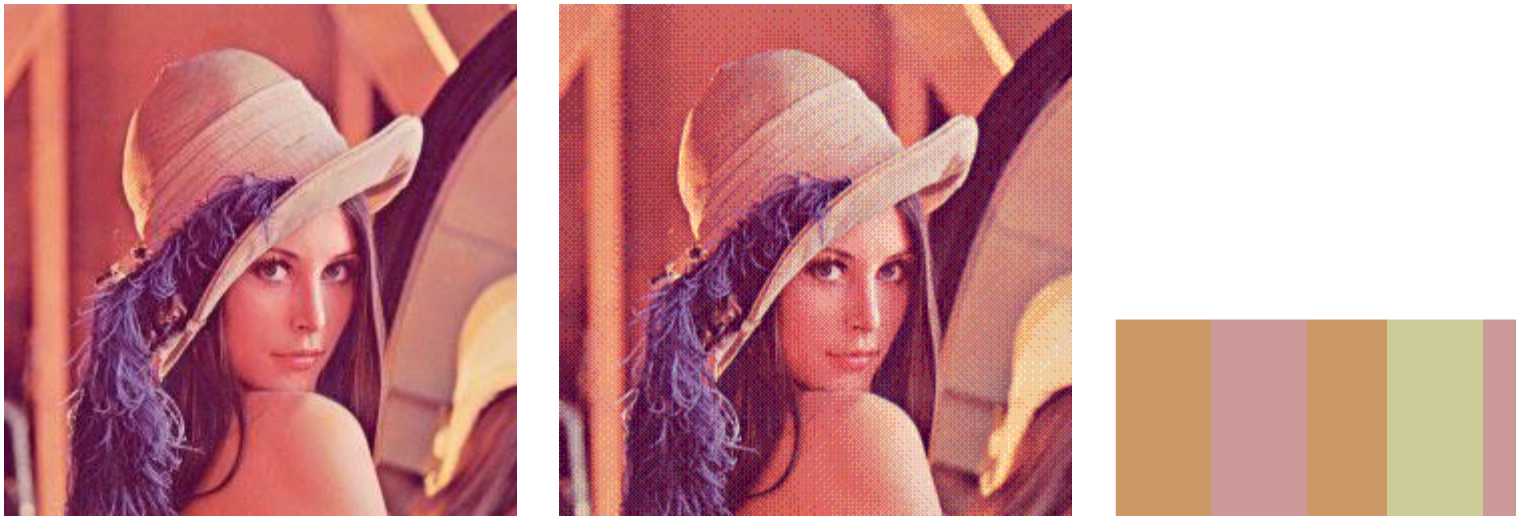
4	3	0	2
1	7	4	5
3	7	6	5
2	2	1	1

Image



Only makes sense if you have lots of pixels and not many colors

- 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.

Median-cut algorithm

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

  a. 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.
  b. This type of scheme will indeed concentrate bits where they most need to differentiate between high populations of close colors.
  c. One can most easily visualize finding the median by using a histogram showing counts at position 0..255.
  d. Fig. 3.11 shows a histogram of the R byte values for the `forestfire.bmp` image along with the median of these values, shown as a vertical line.

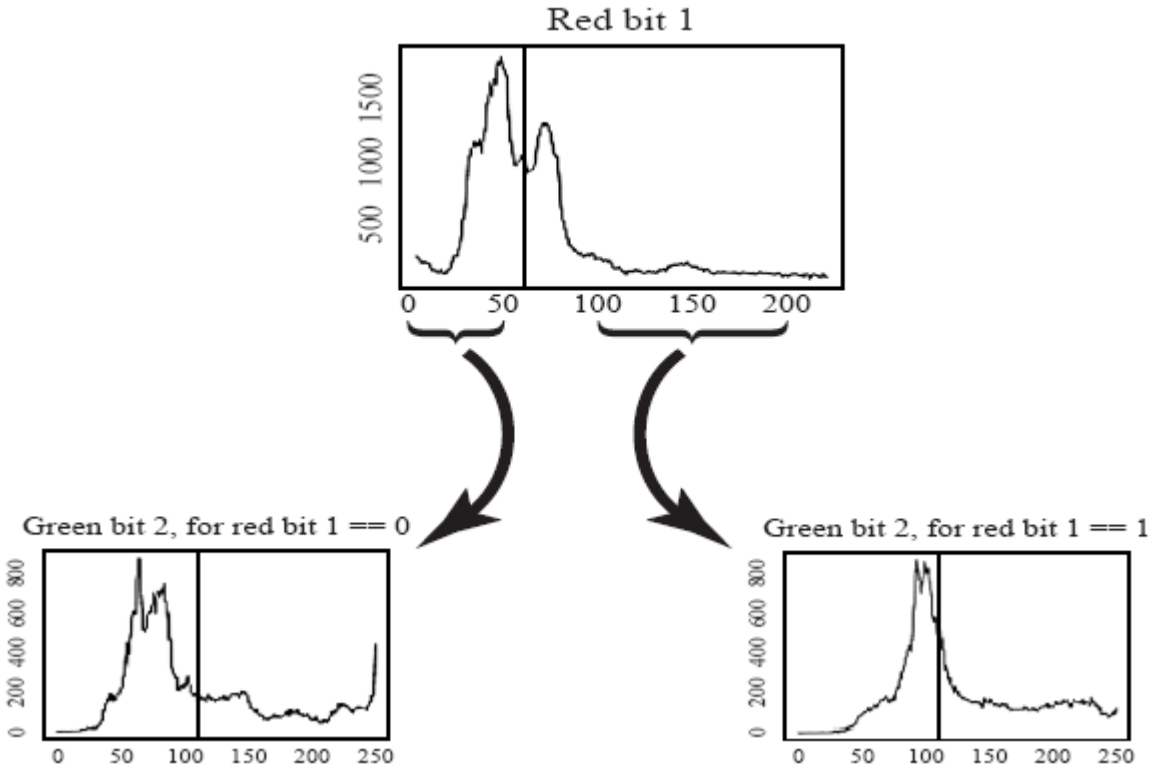
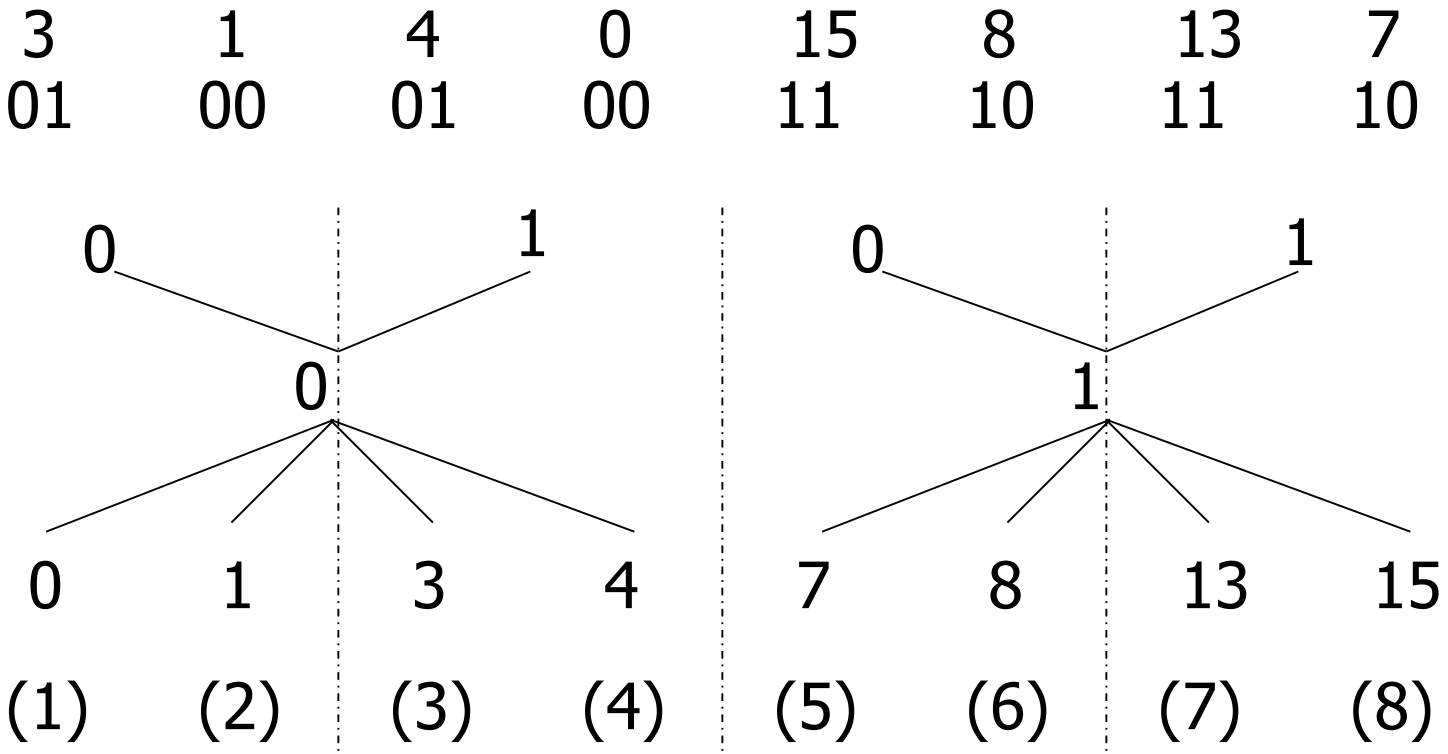


Fig. 3.11: Histogram of R bytes for the 24-bit color image "forestfire.bmp" results in a "0" bit or "1" label for every pixel. For the second bit of the color table index being built, we take R values less than the R median and label just those pixels as "0" or "1" according as their G value is less than or greater than the median of the G value, just for the "0" Red bit pixels. Continuing over R, G, B for 8 bits gives a color LUT 8-bit index.

Median-cut algorithm

- Example (4-bit -> 2-bit quantization)



Color Quantization

2 colors

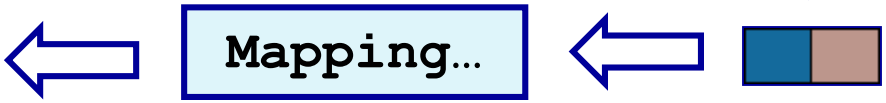
16 colors

4 colors

256 colors

Quantization phases

- Sample the original image for color statistics
- Select color map based on those statistics
- Map the colors to their representative in the color map
- Redraw the image, quantizing each pixel

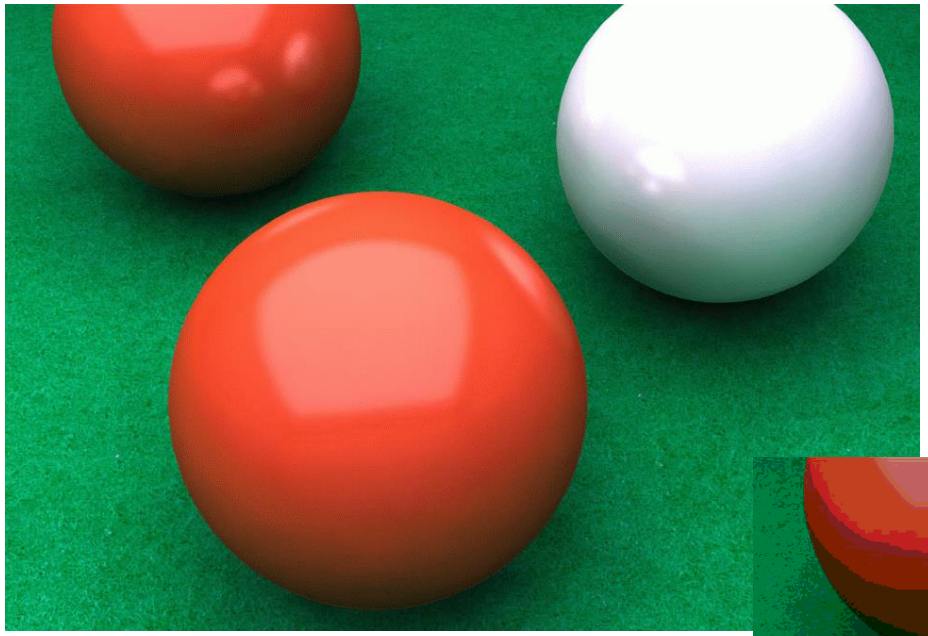


Naïve Color Quantization

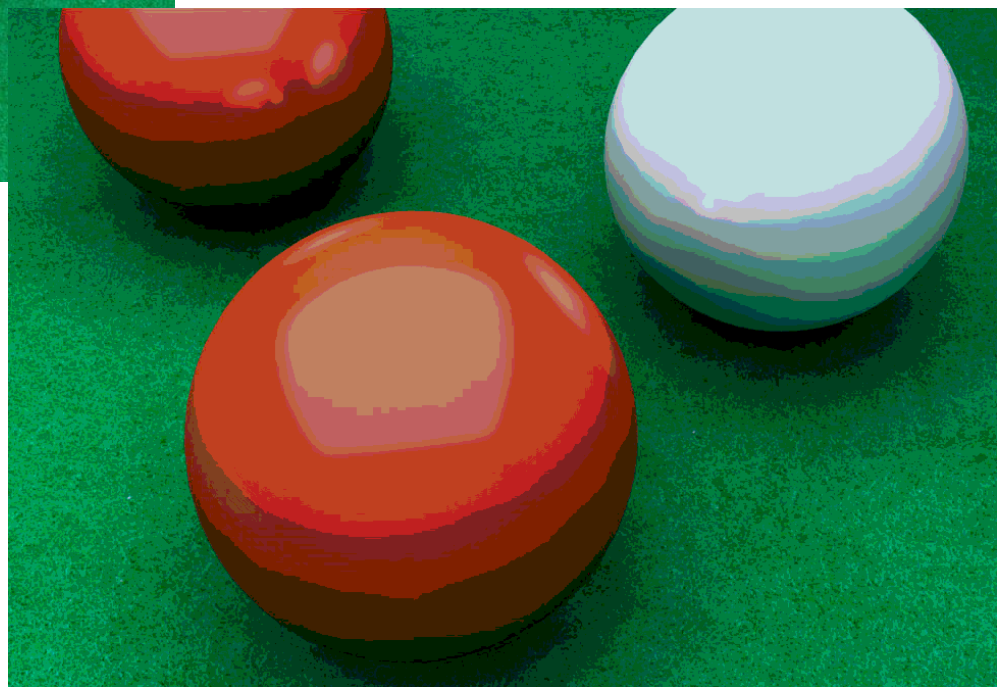
24 bit to 8 bit:

Retaining 3-3-2 most significant bits of the R,G and B components.

24 bit original



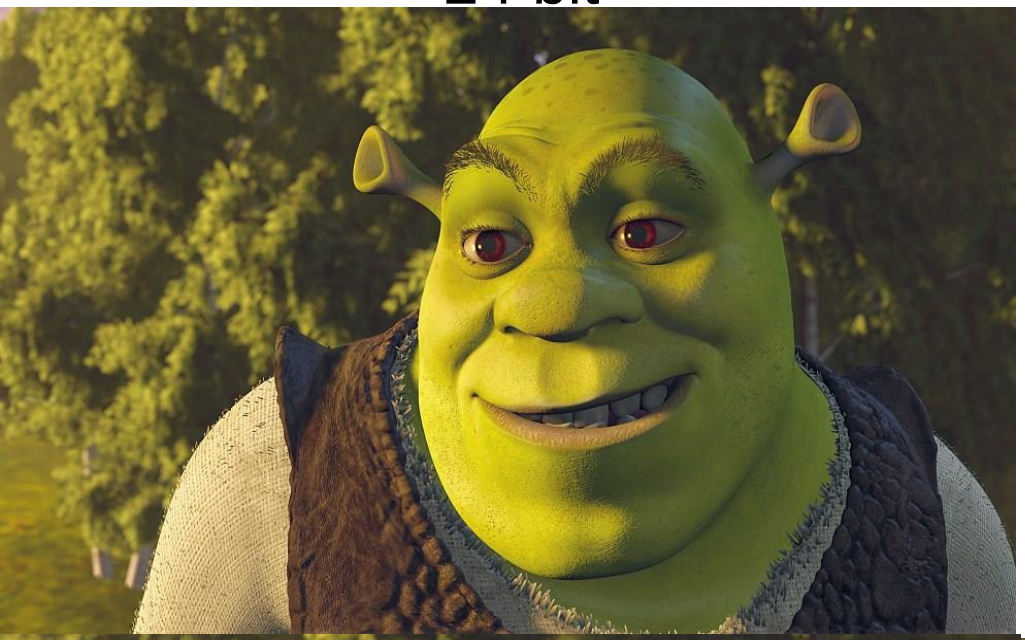
3-3-2 (fixed quantization)



24 bit original

3-3-2 (fixed quantization)

24 bit



Median-cut

8 bit

4 bit