CHAPTER 16

COLOR IMAGE PROCESSING

WHAT WILL WE LEARN?

- What are the most important concepts and terms related to color perception?
- What are the main color models used to represent and quantify color?
- How are color images represented in MATLAB?
- What is pseudocolor image processing and how does it differ from full-color image processing?
- How can monochrome image processing techniques be extended to color images?

16.1 THE PSYCHOPHYSICS OF COLOR

Color perception is a psychophysical phenomenon that combines two main components:

1. The physical properties of light sources (usually expressed by their spectral power distribution (SPD)) and surfaces (e.g., their absorption and reflectance capabilities).
2. The physiological and psychological aspects of the human visual system (HVS).

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

In this section, we expand on the discussion started in Section 5.2.4 and present the main concepts involved in color perception and representation.

16.1.1 Basic Concepts

The perception of color starts with a chromatic light source, capable of emitting electromagnetic radiation with wavelengths between approximately 400 and 700 nm. Part of that radiation reflects on the surfaces of the objects in a scene and the resulting reflected light reaches the human eye, giving rise to the sensation of color. An object that reflects light almost equally in all wavelengths within the visible spectrum is perceived as white, whereas an object that absorbs most of the incoming light, regardless of the wavelength, is seen as black. The perception of several shades of gray between pure white and pure black is usually referred to as *achromatic*. Objects that have more selective properties are considered *chromatic*, and the range of the spectrum that they reflect is often associated with a color name. For example, an object that absorbs most of the energy within the 565–590 nm wavelength range is considered yellow.

A chromatic light source can be described by three basic quantities:

- *Intensity (or Radiance)*: the total amount of energy that flows from the light source, measured in watts (W).
- *Luminance*: a measure of the amount of information an observer *perceives* from a light source, measured in lumen (lm). It corresponds to the radiant power of a light source weighted by a spectral sensitivity function (characteristic of the HVS).
- *Brightness*: the *subjective* perception of (achromatic) luminous intensity.

The human retina (the surface at the back of the eye where images are projected) is coated with photosensitive receptors of two different types: *cones* and *rods*. Rods cannot encode color but respond to lower luminance levels and enable vision under darker conditions. Cones are primarily responsible for color perception and operate only under bright conditions. There are three types of cone cells (L cones, M cones, and S cones, corresponding to long (≈ 610 nm), medium (≈ 560 nm), and short (≈ 430 nm) wavelengths, respectively) whose spectral responses are shown in Figure 16.1.[1]

The existence of three specialized types of cones in the human eye was hypothesized more than a century before it could be confirmed experimentally by Thomas Young and his *trichromatic* theory of vision in 1802. Young's theory explains only part of the color vision process, though. It does not explain, for instance, why it is possible to speak of 'bluish green' colors, but not 'bluish yellow' ones. Such understanding came with the *opponent-process* theory of color vision, brought forth by Edward Herring in 1872. The colors to which the cones respond more strongly are known as the *primary colors of light* and have been standardized by the CIE

[1] The figure also shows the spectral absorption curve for rods, responsible for achromatic vision.

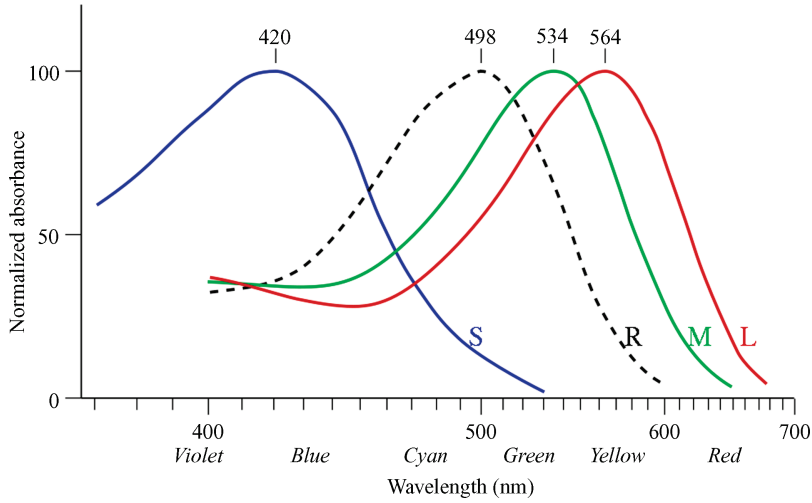

FIGURE 16.1 Spectral absorption curves of the short (S), medium (M), and long (L) wavelength pigments in human cone and rod (R) cells. Courtesy of Wikimedia Commons.

(*Commission Internationale de L'Éclairage*—International Commission on Illumination, an organization responsible for color standards) as red (700 nm), green (546.1 nm), and blue (435.8 nm).

The secondary colors of light, obtained by additive mixtures of the primaries, two colors at a time, are *magenta* (or *purple*) = red + blue, *cyan* (or *turquoise*) = blue + green, and *yellow* = green + red (Figure 16.2a).

For color mixtures using pigments (or paints), the primary colors are magenta, cyan, and yellow and the secondary colors are red, green, and blue (Figure 16.2b). It is important to note that for pigments a color is named after the portion of the spectrum

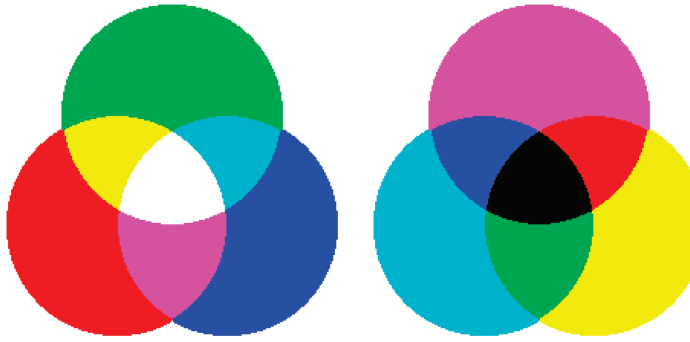

FIGURE 16.2 Additive (a) and subtractive (b) color mixtures.

that it absorbs, whereas for light a color is defined based on the portion of the spectrum that it emits. Consequently, mixing all three primary colors of light results in white (i.e., the entire spectrum of visible light), whereas mixing all three primary colors of paints results in black (i.e., all colors have been absorbed, and nothing remains to reflect the incoming light).

The use of the expression *primary colors* to refer to red, green, and blue may lead to a common misinterpretation: that *all* visible colors can be obtained by mixing different amounts of each primary color, which is not true. A related phenomenon of color perception, the existence of *color metamers*, may have contributed to this confusion. Color metamers are combinations of primary colors (e.g., red and green) perceived by the HVS as another color (in this case, yellow) that could have been produced by a spectral color of fixed wavelength (of ≈ 580 nm).

16.1.2 The CIE *XYZ* Chromaticity Diagram

In color matching experiments performed in the late 1920s, subjects were asked to adjust the amount of red, green, and blue on one patch that were needed to match a color on a second patch. The results of such experiments are summarized in Figure 16.3. The existence of negative values of red and green in this figure means that the second patch should be made brighter (i.e., equal amounts of red, green, and blue has to be added to the color) for the subjects to report a perfect match. Since adding amounts of primary colors on the second patch corresponds to subtracting them in the first, negative values can occur. The amounts of three primary colors in a three-component additive color model (on the first patch) needed to match a test color (on the second patch) are called *tristimulus values*.

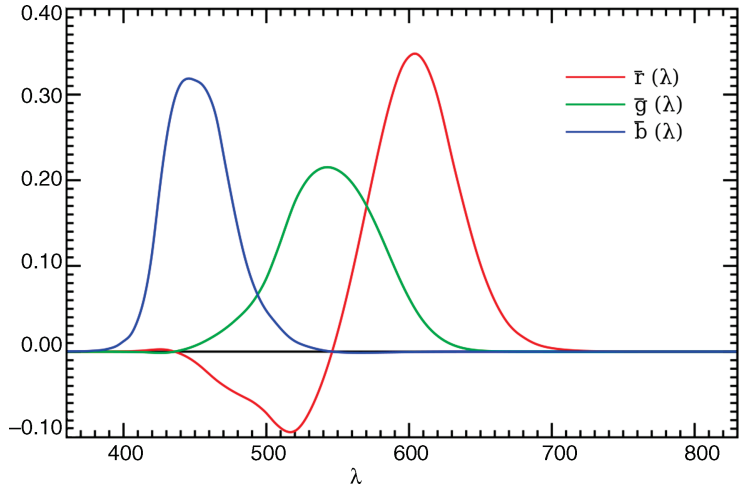

FIGURE 16.3 RGB color matching function (CIE 1931). Courtesy of Wikimedia Commons.

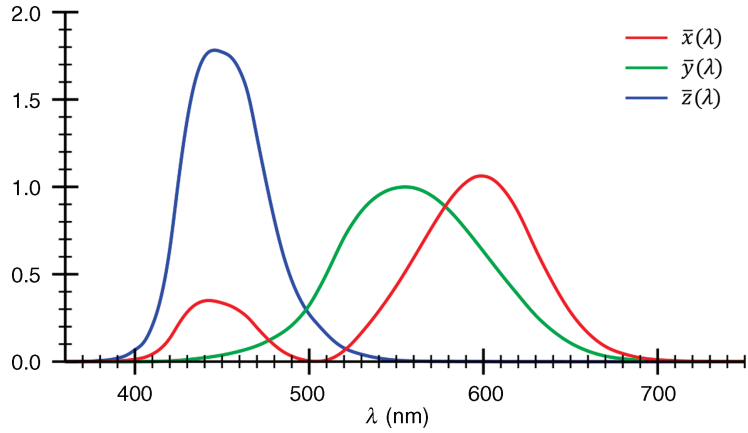

FIGURE 16.4 *XYZ* color matching function (CIE 1931). Courtesy of Wikimedia Commons.

To remove the inconvenience of having to deal with (physically impossible) negative values to represent observable colors, in 1931 the CIE adopted standard curves for a hypothetical *standard (colorimetric) observer*, considered to be the chromatic response of the average human viewing through a 2° angle, due to the belief at that time that the cones resided within a 2° arc of the fovea.[2] These curves specify how a SPD corresponding to the physical power (or *radiance*) of the light source can be transformed into a set of three numbers that specifies a color. These curves are not based on the values of R, G, and B, but on a new set of tristimulus values: X, Y, and Z.

This model, whose color matching functions are shown in Figure 16.4, is known as the *CIE XYZ* (or CIE 1931) model. The tristimulus values of X, Y, and Z in Figure 16.4 are related to the values of R, G, and B in Figure 16.3 by the following linear transformations:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.431 & 0.342 & 0.178 \\ 0.222 & 0.707 & 0.071 \\ 0.020 & 0.130 & 0.939 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{16.1}$$

and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.063 & -1.393 & -0.476 \\ -0.969 & 1.876 & 0.042 \\ 0.068 & -0.229 & 1.069 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{16.2}$$

[2]This understanding was eventually revised and a new model, CIE 1964, for a 10° standard observer, was produced. Interestingly enough, the CIE 1931 model is more popular than the CIE 1964 alternative even today.

The CIE XYZ color space was designed so that the Y parameter corresponds to a measure of the *brightness* of a color. The *chromaticity* of a color is specified by two other parameters, x and y, known as *chromaticity coordinates* and calculated as

$$x = \frac{X}{X + Y + Z} \tag{16.3}$$

$$y = \frac{Y}{X + Y + Z} \tag{16.4}$$

where x and y are also called *normalized tristimulus values*. The third normalized tristimulus value, z, can just as easily be calculated as

$$z = \frac{Z}{X + Y + Z} \tag{16.5}$$

Clearly, a combination of the three normalized tristimulus values results in

$$x + y + z = 1 \tag{16.6}$$

The resulting CIE XYZ chromaticity diagram (Figure 16.5) allows the mapping of a color to a point of coordinates (x, y) corresponding to the color's chromaticity. The complete specification of a color (chromaticity and luminance) takes the form of an xyY triple.[3]

To recover X and Z from x, y, and Y, we can use

$$X = \frac{x}{y} Y \tag{16.7}$$

$$Z = \frac{1 - x - y}{y} Y \tag{16.8}$$

The resulting CIE XYZ chromaticity diagram shows a horseshoe-shaped outer curved boundary, representing the *spectral locus* of wavelengths (in nm) along the visible light portion of the electromagnetic spectrum. The *line of purples* on a chromaticity diagram joins the two extreme points of the spectrum, suggesting that the sensation of purple cannot be produced by a single wavelength: it requires a mixture of shortwave and longwave light and for this reason purple is referred to as a *nonspectral color*. All colors of light are contained in the area in (x, y) bounded by the line of purples and the spectral locus, with pure white at its center.

The inner triangle in Figure 16.6a represents a *color gamut*, that is, a range of colors that can be produced by a physical device, in this case a CRT monitor. Different color image display and printing devices and technologies exhibit gamuts of different shape and size, as shown in Figure 16.6. As a rule of thumb, the larger the gamut, the better the device's color reproduction capabilities.

[3]This explains why this color space is also known as *CIExyY* color space.

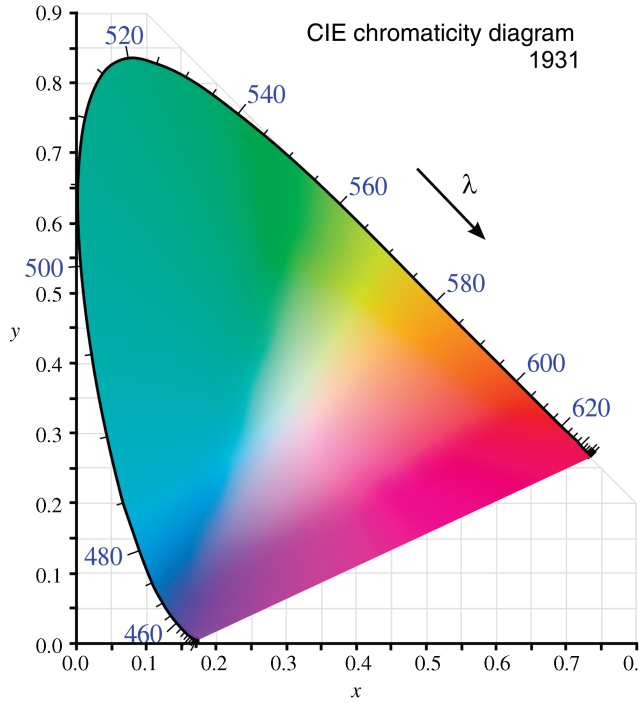

FIGURE 16.5 CIE *XYZ* color model.

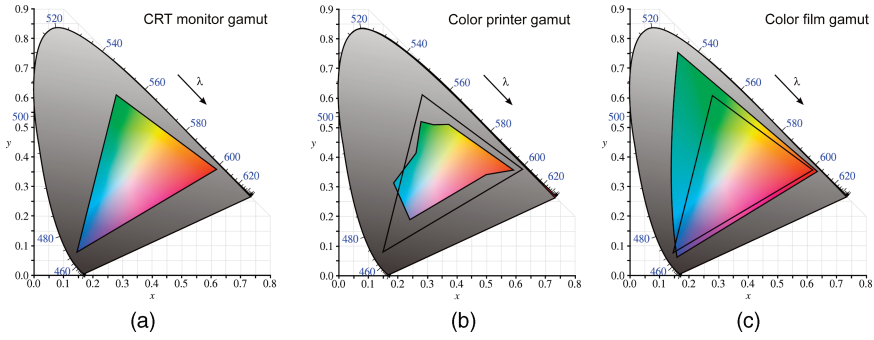

FIGURE 16.6 Color gamut for three different devices: (a) CRT monitor; (b) printer; (c) film. The RGB triangle is the same in all figures to serve as a reference for comparison.

16.1.3 Perceptually Uniform Color Spaces

One of the main limitations of the CIE *XYZ* chromaticity diagram lies in the fact that a distance on the *xy* plane does not correspond to the degree of difference between two colors. This was demonstrated in the early 1940s by David MacAdam, who conducted

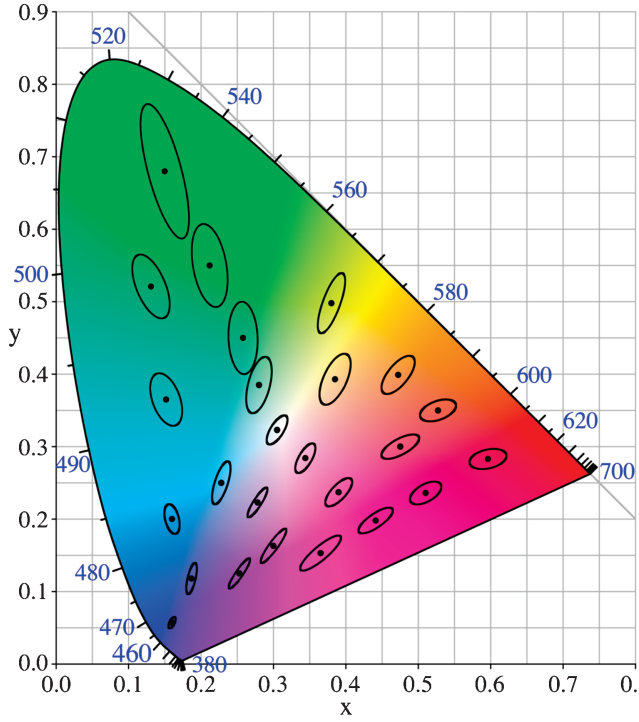

FIGURE 16.7 MacAdam ellipses overlapped on the CIE 1931 chromaticity diagram. Courtesy of Wikimedia Commons.

experiments asking subjects to report noticeable changes in color (relative to a starting color stimulus). The result of that study is illustrated in Figure 16.7, showing the resulting *MacAdam ellipses*: regions on the chromaticity diagram corresponding to all colors that are indistinguishable, to the average human eye, from the color at the center of the ellipse. The contour of the ellipse represents the just noticeable differences (JNDs) of chromaticity. Based on the work of MacAdam, several CIE color spaces—most notably the CIE L*u*v* (also known as CIELUV) and the CIE L*a*b* (also known as CIELAB)—were developed, with the goal of achieving *perceptual uniformity*, that is, have an equal distance in the color space corresponding to equal differences in color.

In MATLAB

The IPT has an extensive support for conversion among CIE color spaces. Converting from one color space to another is usually accomplished by using function `makecform` (to create a color transformation structure that defines the desired color space conversion) followed by `applycform`, which takes the color transformation structure as a parameter.

TABLE 16.1 IPT Functions for CIE *XYZ* and CIELAB Color Spaces

Function	Description
`xyz2double`	Converts an $M \times 3$ or $M \times N \times 3$ array of *XYZ* color values to `double`
`xyz2uint16`	Converts an $M \times 3$ or $M \times N \times 3$ array of *XYZ* color values to `uint16`
`lab2double`	Converts an $M \times 3$ or $M \times N \times 3$ array of L*a*b* color values to `double`
`lab2uint16`	Converts an $M \times 3$ or $M \times N \times 3$ array of L*a*b* color values to `uint16`
`lab2uint8`	Converts an $M \times 3$ or $M \times N \times 3$ array of L*a*b* color values to `uint8`
`whitepoint`	Returns a 3×1 vector of *XYZ* values scaled so that $Y = 1$

Other functions for manipulation of CIE *XYZ* and CIELAB values are listed in Table 16.1.

16.1.4 ICC Profiles

An ICC (International Color Consortium) profile is a standardized description of a color input or output device, or a color space, according to standards established by the ICC. Profiles are used to define a mapping between the device source or target color space and a *profile connection space* (PCS), which is either CIELAB (L*a*b*) or CIEXYZ.

In MATLAB

The IPT has several functions to support ICC profile operations. They are listed in Table 16.2.

TABLE 16.2 IPT Functions for *ICC* Profile Manipulation

Function	Description
`iccread`	Reads an ICC profile into the MATLAB workspace
`iccfind`	Finds ICC color profiles on a system, or a particular ICC color profile whose description contains a certain text string
`iccroot`	Returns the name of the directory that is the default system repository for ICC profiles
`iccwrite`	Writes an ICC color profile to disk file

16.2 COLOR MODELS

A *color model* (also called *color space* or *color system*) is a specification of a coordinate system and a subspace within that system where each color is represented by a single point.

There have been many different color models proposed over the last 400 years. Contemporary color models have also evolved to specify colors for different purposes (e.g., photography, physical measurements of light, color mixtures, etc.). In this section, we discuss the most popular color models used in image processing.

16.2.1 The RGB Color Model

The RGB color model is based on a Cartesian coordinate system whose axes represent the three primary colors of light (R, G, and B), usually normalized to the range [0, 1] (Figure 16.8). The eight vertices of the resulting cube correspond to the three primary colors of light, the three secondary colors, pure white, and pure black. Table 16.3 shows the R, G, and B values for each of these eight vertices.

RGB color coordinates are often represented in hexadecimal notation, with individual components varying from `00` (decimal 0) to `FF` (decimal 255). For example, a pure (100% saturated) red would be denoted `FF0000`, whereas a slightly desaturated yellow could be written as `CCCC33`.

The number of discrete values of R, G, and B is a function of the *pixel depth*, defined as the number of bits used to represent each pixel: a typical value is

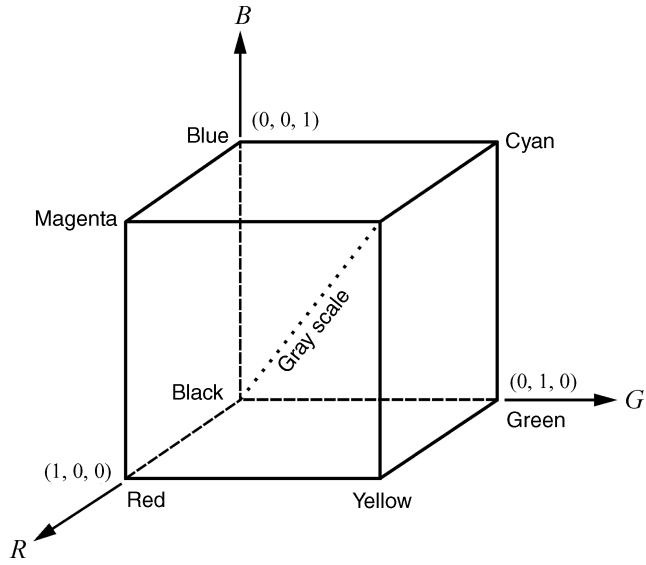

FIGURE 16.8 RGB color model.

TABLE 16.3 *R*, *G*, and *B* Values for Eight Representative Colors Corresponding to the Vertices of the *RGB* Cube

Color Name	*R*	*G*	*B*
Black	0	0	0
Blue	0	0	1
Green	0	1	0
Cyan	0	1	1
Red	1	0	0
Magenta	1	0	1
Yellow	1	1	0
White	1	1	1

24 bits = 3 image planes × 8 bits per plane. The resulting cube—with more than 16 million possible color combinations—is shown in Figure 16.9.

In MATLAB

The RGB cube in Figure 16.9 was generated using `patch`, a graphics function for creating *patch graphics objects*, made up of one or more polygons, which can be specified by passing the coordinates of their vertices and the coloring and lighting of the patch as parameters.

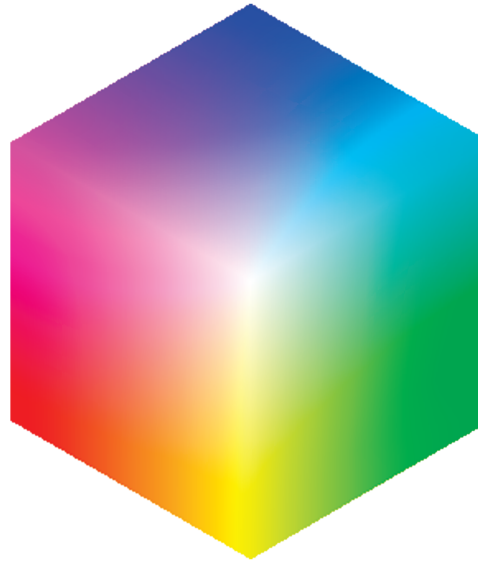

FIGURE 16.9 RGB color cube.

16.2.2 The CMY and CMYK Color Models

The *CMY* model is based on the three primary colors of pigments (*c*yan, *m*agenta, and *y*ellow). It is used for color printers, where each primary color usually corresponds to an ink (or toner) cartridge. Since the addition of equal amounts of each primary to produce black usually produces unacceptable, muddy looking black, in practice, a fourth color, *blacK*, is added, and the resulting model is called *CMYK*.

The conversion from *RGB* to *CMY* is straightforward:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{16.9}$$

The inverse operation (conversion from *CMY* to *RGB*), although it is equally easy from a mathematical standpoint, is of little practical use.

In MATLAB

Conversion between *RGB* and *CMY* in MATLAB can also be accomplished using the `imcomplement` function.

16.2.3 The HSV Color Model

Color models such as the *RGB* and *CMYK* described previously are very convenient to specify color coordinates for display or printing, respectively. They are not, however, useful to capture a typical human description of color. After all, none of us goes to a store looking for a `FFFFCC` shirt to go with the `FFCC33` jacket we got for our birthday. Rather, the human perception of color is best described in terms of hue, saturation, and lightness. *Hue* describes the color type, or tone, of the color (and very often is expressed by the "color name"), *saturation* provides a measure of its purity (or how much it has been diluted in white), and *lightness* refers to the intensity of light reflected from objects.

For representing colors in a way that is closer to the human description, a family of color models have been proposed. The common aspect among these models is their ability to dissociate the dimension of *intensity* (also called *brightness* or *value*) from the *chromaticity*—expressed as a combination of *hue* and *saturation*—of a color.

We will look at a representative example from this family: the *HSV* (hue–saturation–value) color model.[4]

The *HSV* (sometimes called *HSB*) color model can be obtained by looking at the *RGB* color cube along its main diagonal (or *gray axis*), which results in a hexagon-shaped color palette. As we move along the main axis in the pyramid in Figure 16.10,

[4]The terminology for color models based on hue and saturation is not universal, which is unfortunate. What we call *HSV* in this book may appear under different names and acronyms elsewhere. Moreover, the distinctions among these models are very subtle and different acronyms might be used to represent slight variations among them.

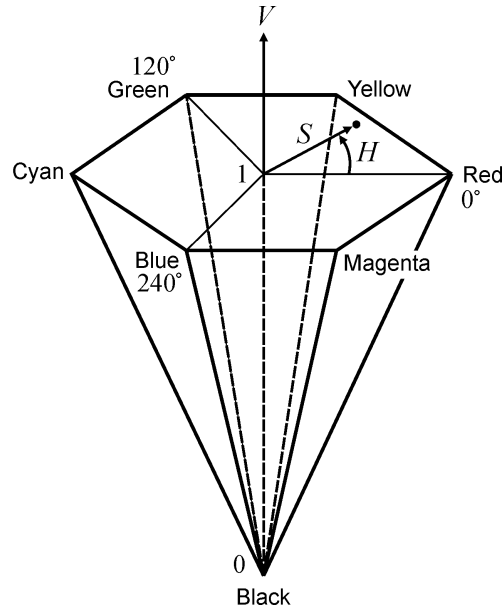

FIGURE 16.10 The HSV color model as a hexagonal cone.

the hexagon gets smaller, corresponding to decreasing values of V, from 1 (white) to 0 (black). For any hexagon, the three primary and the three secondary colors of light are represented in its vertices. Hue, therefore, is specified as an angle relative to the origin (the red axis by convention). Finally, saturation is specified by the distance to the axis: the longer the distance, the more saturated the color.

Figure 16.11 shows an alternative representation of the *HSV* color model in which the hexcone is replaced by a cylinder. Figure 16.12 shows yet another equivalent three-dimensional representation for the *HSV* color model, as a cone with circular-shaped base.

In summary, the main advantages of the *HSV* color model (and its closely related alternatives) are its ability to match the human way of describing colors and to allow for independent control over hue, saturation, and intensity (value). The ability to isolate the intensity component from the other two—which are often collectively called *chromaticity* components—is a requirement in many color image processing algorithms, as we shall see in Section 16.5. Its main disadvantages include the discontinuity in numeric values of hue around red, the computationally expensive conversion to/from RGB, and the fact that hue is undefined for a saturation of 0.

In MATLAB

Converting between *HSV* and *RGB* in MATLAB can be accomplished by the functions `rgb2hsv` and `hsv2rgb`. Tutorial 16.2 explores these functions in detail.

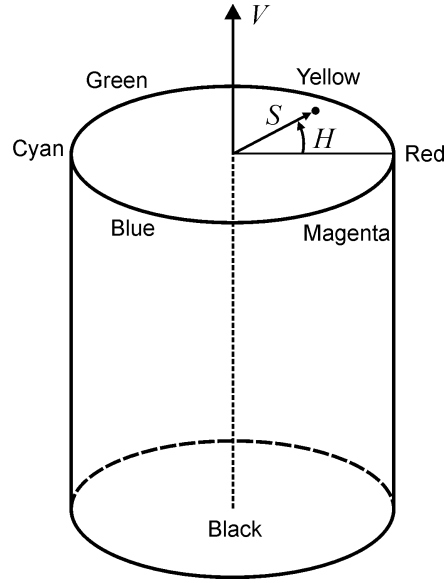

FIGURE 16.11 The *HSV* color model as a cylinder.

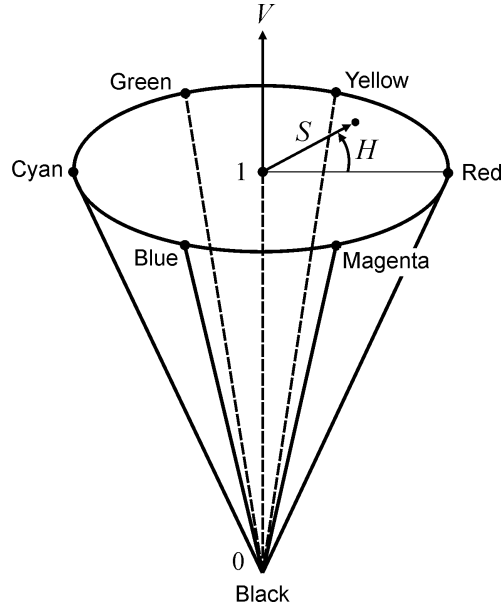

FIGURE 16.12 The HSV color model as a cone.

16.2.4 The YIQ (NTSC) Color Model

The NTSC color model is used in the American standard for analog television, which will be described in more detail in Chapter 20. One of the main advantages of this model is the ability to separate grayscale contents from color data, a major design requirement at a time when emerging color TV sets and transmission equipment had to be backward compatible with their B&W predecessors. In the NTSC color model, the three components are luminance (Y) and two color-difference signals hue (I) and saturation (Q).[5]

Conversion from RGB to YIQ can be performed using the transformation

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{16.10}$$

In MATLAB

Converting between RGB and YIQ (NTSC) in MATLAB is accomplished using functions `rgb2ntsc` and `ntsc2rgb`.

16.2.5 The YCbCr Color Model

The $YCbCr$ color model is the most popular color representation for digital video.[6] In this format, one component represents luminance (Y), while the other two are color-difference signals: Cb (the difference between the blue component and a reference value) and Cr (the difference between the red component and a reference value).

Conversion from RGB to $YCbCr$ is possible using the transformation

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{16.11}$$

In MATLAB

Converting between $YCbCr$ and RGB in MATLAB can be accomplished by the functions `rgb2ycbcr` and `ycbcr2rgb`.

16.3 REPRESENTATION OF COLOR IMAGES IN MATLAB

As we have seen in Chapter 2, color images are usually represented as RGB (24 bits per pixel) or indexed with a palette (color map), usually of size 256. These representation

[5]The choice for letters I and Q stems from the fact that one of the components is *i*n phase, whereas the other is off by 90°, that is, in *q*uadrature.

[6]More details in Chapter 20.

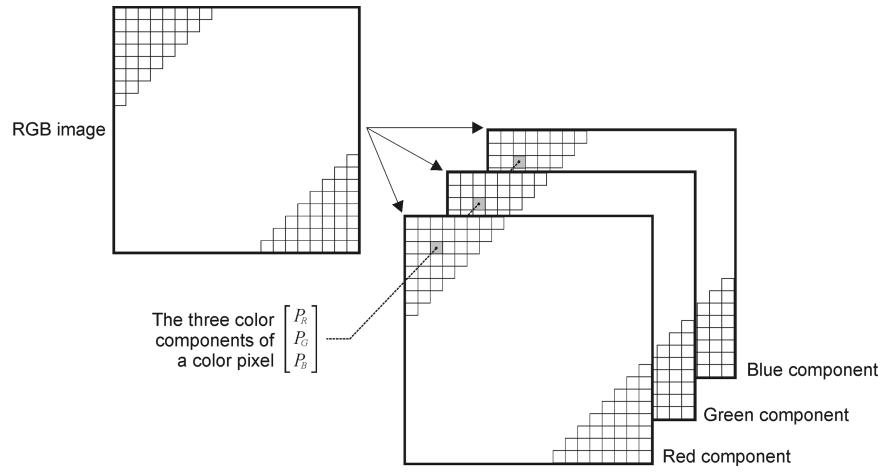

FIGURE 16.13 RGB color image representation.

modes are independent of file format (although GIF images are usually indexed and JPEG images typically are not). In this section, we provide a more detailed analysis of color image representation in MATLAB.

16.3.1 RGB Images

An RGB color image in MATLAB corresponds to a 3D array of dimensions $M \times N \times 3$, where M and N are the image's height and width (respectively) and 3 is the number of color planes (channels). Each color pixel is represented as a triple containing the values of its R, G, and B components (Figure 16.13). Each individual array of size $M \times N$ is called a *component image* and corresponds to one of the color channels: red, green, or blue. The data class of the component images determines their range of values. For RGB images of class `double`, the range of values is [0.0, 1.0], whereas for classes `uint8` or `uint16`, the ranges are [0, 255] and [0, 65535], respectively. RGB images typically have a *bit depth* of 24 bits per pixel (8 bits per pixel per component image), resulting in a total number of $(2^8)^3 = 16,777,216$ colors.

■ **EXAMPLE 16.1**

The following MATLAB sequence can be used to open, verify the size (in this case, $384 \times 512 \times 3$) and data class (in this case, `uint8`), and display an RGB color image (Figure 16.14).

```
I = imread('peppers.png');
size(I)
class(I)
subplot(2,2,1), imshow(I), title('Color image (RGB)')
```

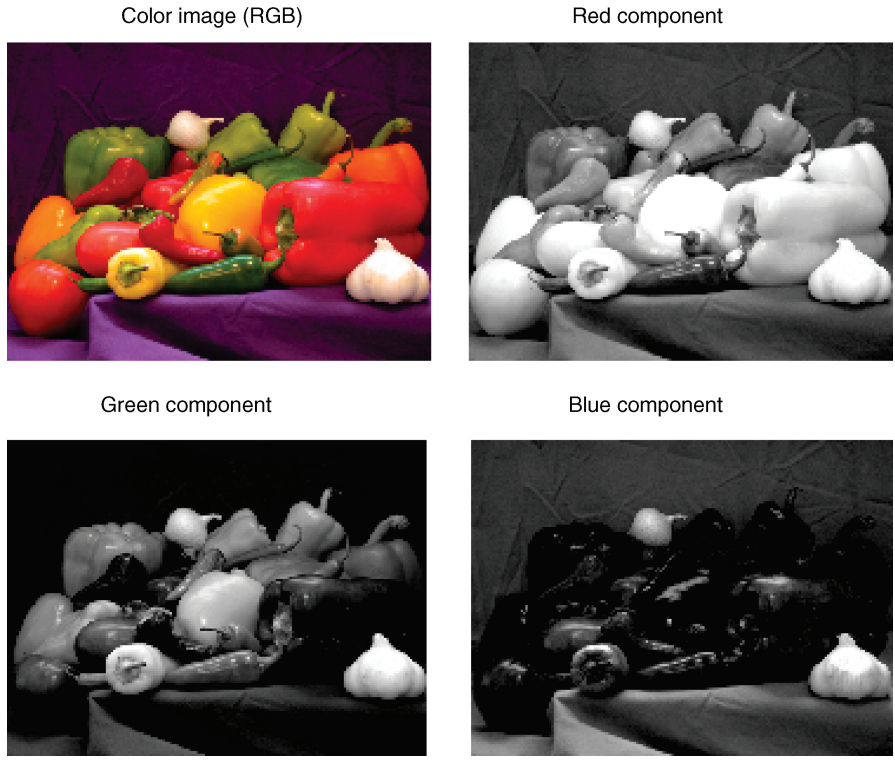

FIGURE 16.14 *RGB* image and its three color components (or *channels*). Original image: courtesy of MathWorks.

```
subplot(2,2,2), imshow(I(:,:,1)), title('Red component')
subplot(2,2,3), imshow(I(:,:,2)), title('Green component')
subplot(2,2,4), imshow(I(:,:,3)), title('Blue component')
```

16.3.2 Indexed Images

An indexed image is a matrix of integers (`X`), where each integer refers to a particular row of RGB values in a secondary matrix (`map`) known as a *color map*. The image can be represented by an array of class `uint8`, `uint16`, or `double`. The color map array is an $M \times 3$ matrix of class `double`, where each element's value is within the range [0.0, 1.0]. Each row in the color map represents R (red), G (green), and B (blue) values, in that order. The indexing mechanism works as follows (Figure 16.15): if `X` is of class `uint8` or `uint16`, all components with value 0 point to the first row in `map`, all components with value 1 point to the second row, and so on. If `X` is of class `double`, all components with value less than or equal to 1.0 point to the first row and so on.

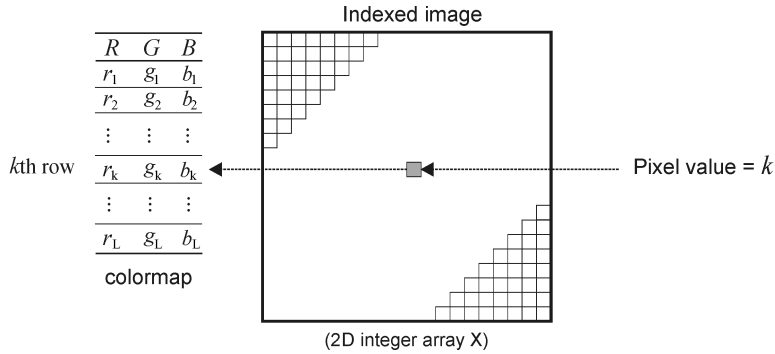

FIGURE 16.15 Indexed color image representation.

MATLAB has many built-in color maps (which can be accessed using the function `colormap`), briefly described in Table 16.4. In addition, you can easily create your own color map by defining an array of class `double` and size $M \times 3$, where each element is a floating-point value in the range [0.0, 1.0].

■ EXAMPLE 16.2

The following MATLAB sequence can be used to load a built-in indexed image, verify its size (in this case, 200 × 300) and data class (in this case, `double`), verify the

TABLE 16.4 Color Maps in MATLAB

Name	Description
`hsv`	Hue–saturation–value color map
`hot`	Black–red–yellow–white color map
`gray`	Linear gray-scale color map
`bone`	Gray scale with tinge of blue color map
`copper`	Linear copper-tone color map
`pink`	Pastel shades of pink color map
`white`	All white color map
`flag`	Alternating red, white, blue, and black color map
`lines`	Color map with the line colors
`colorcube`	Enhanced color-cube color map
`vga`	Windows color map for 16 colors
`jet`	Variant of HSV
`prism`	Prism color map
`cool`	Shades of cyan and magenta color map
`autumn`	Shades of red and yellow color map.
`spring`	Shades of magenta and yellow color map
`winter`	Shades of blue and green color map
`summer`	Shades of green and yellow color map

FIGURE 16.16 A built-in indexed image. Original image: courtesy of MathWorks.

color map's size (in this case, 81 × 3) and data class (in this case, `double`), and display the image (Figure 16.16).

```
load clown
size(X)
class(X)
size(map)
class(map)
imshow(X,map), title('Color (Indexed)')
```

MATLAB has many useful functions for manipulating indexed color images:

- If we need to approximate an indexed image by one with fewer colors, we can use MATLAB's `imapprox` function.
- Conversion between RGB and indexed color images is straightforward, thanks to functions `rgb2ind` and `ind2rgb`.
- Conversion from either color format to their grayscale equivalent is equally easy, using functions `rgb2gray` and `ind2gray`.
- We can also create an index image from an RGB image by dithering the original image using function `dither`.
- Function `grayslice` creates an indexed image from an intensity (grayscale) image by thresholding and can be used in pseudocolor image processing (Section 16.4).
- Function `gray2ind` converts an intensity (grayscale) image into its indexed image equivalent. It is different from `grayslice`. In this case, the resulting

image is monochrome, just as the original one;[7] only the internal data representation has changed.

16.4 PSEUDOCOLOR IMAGE PROCESSING

The purpose of pseudocolor image processing techniques is to enhance a monochrome image for human viewing purposes. Their rationale is that subtle variations of gray levels may very often mask or hide regions of interest within an image. This can be particularly damaging if the masked region is relevant to the application domain (e.g., the presence of a tumor in a medical image). Since the human eye is capable of discerning thousands of color hues and intensities, compared to only less than 100 shades of gray, replacing gray levels with colors leads to better visualization and enhanced capability for detecting relevant details within the image.

The typical solution consists of using a color lookup table (LUT) designed to map the entire range of (typically 256) gray levels to a (usually much smaller) number of colors. For better results, contrasting colors should appear in consecutive rows in the LUT. The term *pseudo*color is used to emphasize the fact that the assigned colors usually have no correspondence whatsoever with the truecolors that might have been present in the original image.

16.4.1 Intensity Slicing

The technique of *intensity* (or *density*) *slicing* is the simplest and best-known pseudocoloring technique. If we look at a monochrome image as if it were a 3D plot of gray levels versus spatial coordinates, where the most prominent peaks correspond to the brightest pixels, the technique corresponds to placing several planes parallel to the coordinate plane of the image (also known as the xy plane). Each plane "slices" the 3D function in the area of intersection, resulting in several gray-level intervals. Each side of the plane is then assigned a different color. Figure 16.17 shows an example of intensity slicing using only one slicing plane at $f(x, y) = l_i$ and Figure 16.18 shows an alternative representation, where the chosen colors are indicated as c_1, c_2, c_3, and c_4. The idea can easily be extended to M planes and $M + 1$ intervals.

In MATLAB

Intensity slicing can be accomplished using the `grayslice` function. You will learn how to use this function in Tutorial 16.1.

[7] The only possible differences would have resulted from the need to requantize the number of gray levels if the specified color map is smaller than the original number of gray levels.

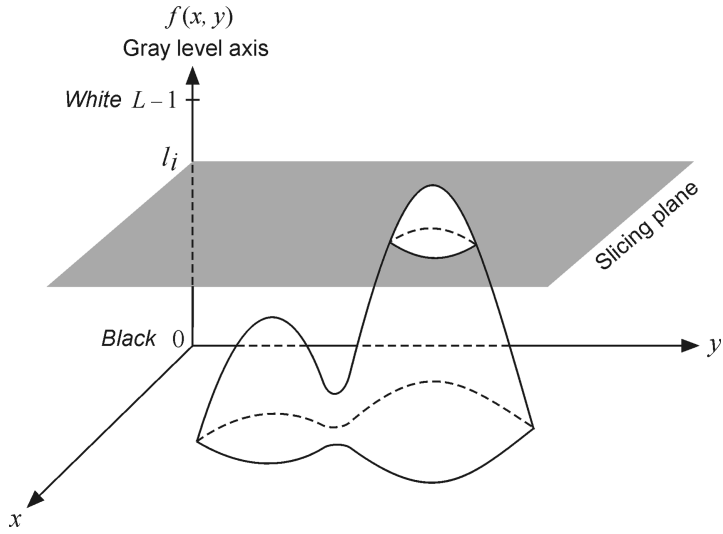

FIGURE 16.17 Pseudocoloring with intensity slicing.

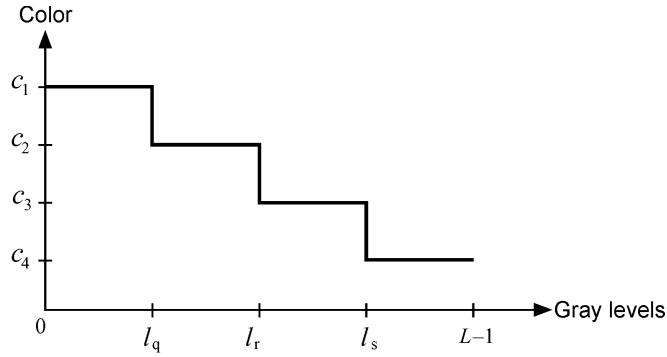

FIGURE 16.18 An alternative representation of the intensity slicing technique for an image with L gray levels pseudocolored using four colors.

■ EXAMPLE 16.3

Figure 16.19 shows an example of pseudocoloring with intensity slicing using 16 levels, the same input image (a), and three different color maps `summer` (b), `hsv` (c), and `jet` (d).

16.4.2 Gray Level to Color Transformations

An alternative approach to pseudocoloring consists of using three independent transformation functions on each pixel of the input image and assigning the results of each

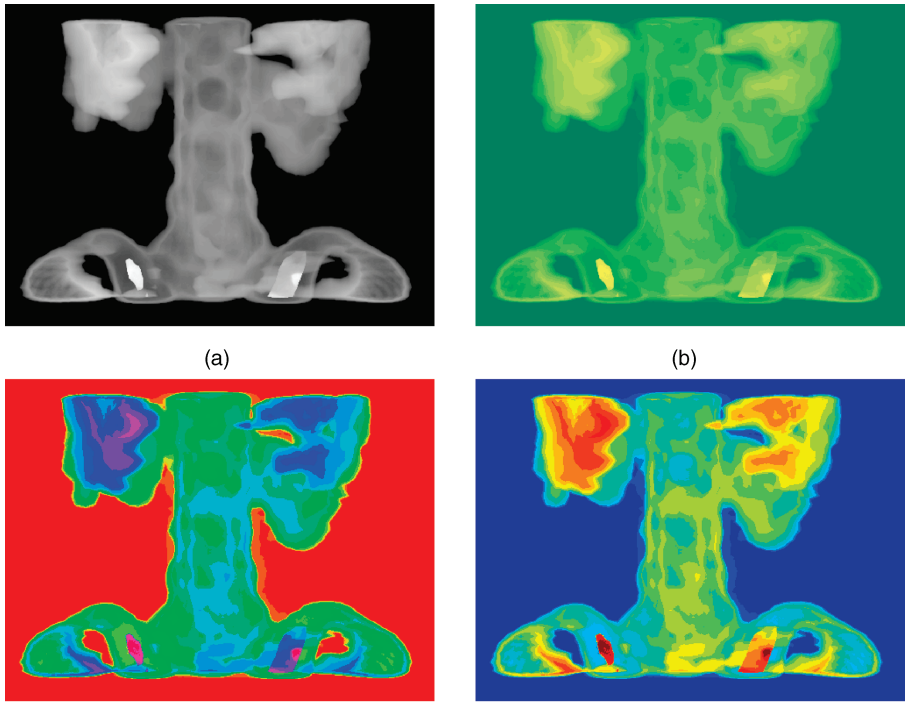

FIGURE 16.19 Pseudocoloring using intensity slicing: original image (a) and results of pseudocoloring using different color maps (b–d). Original image: courtesy of MathWorks.

function to a color channel (Figure 16.20). This method provides additional flexibility, since it allows creation of a composite color image whose contents can be modulated by each individual transformation function. Recall from our discussion in Chapter 8 that these are point functions, that is, the resulting value for each pixel does not depend on its spatial location or the gray level of its neighbors.

The intensity slicing method described in Section 16.4.1 is a particular case of gray level to color transformation, in which all transformation functions are identical and shaped like a staircase.

16.4.3 Pseudocoloring in the Frequency Domain

Pseudocoloring can also performed in the frequency domain[8] by applying a Fourier transform (FT) to the original image and then applying a low-pass, bandpass, and high-pass filters to the transformed data. The three individual filter outputs are then inverse transformed and used as the R, G, and B components of the resulting image.

[8] Frequency-domain techniques were discussed in Chapter 11.

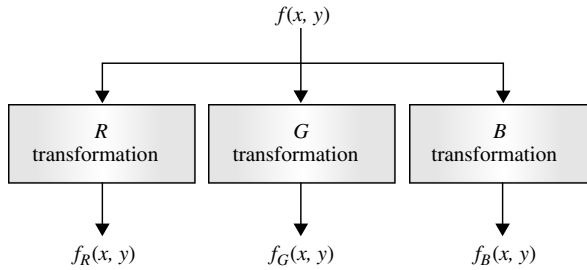

FIGURE 16.20 Block diagram for pseudocoloring using color transformation functions.

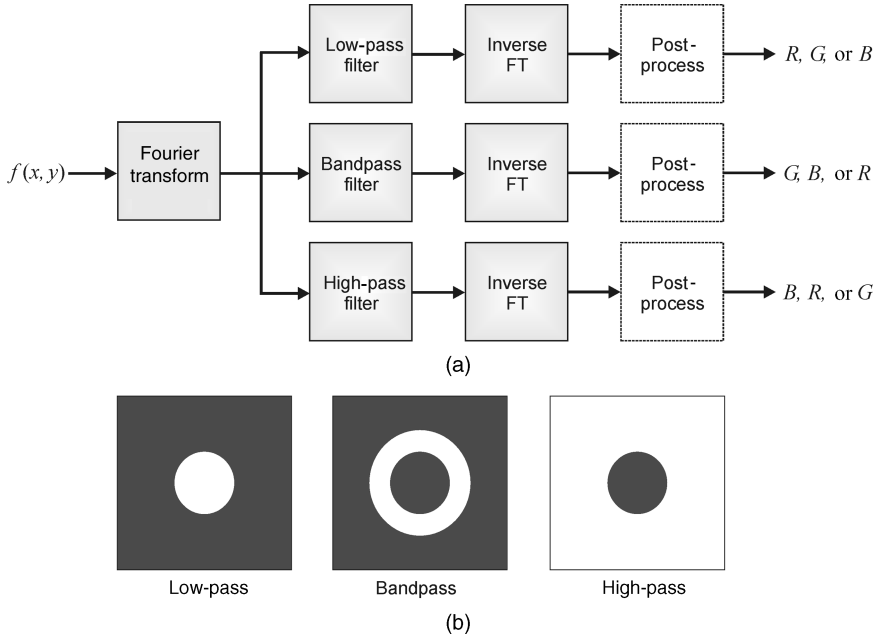

FIGURE 16.21 (a) Block diagram for pseudocoloring in the frequency domain; (b) frequency response of the filters. Redrawn from [Umb05].

Figure 16.21 shows a block diagram of the method as well as representative examples of the frequency response of the filters used in the process. The *postprocessing* stage is optional and application dependent.

16.5 FULL-COLOR IMAGE PROCESSING

This section examines techniques that process the full contents of a digital color image. Full-color image processing is a relatively young branch of digital image processing

that has become increasingly popular and relevant in recent years, thanks to the wide availability of inexpensive hardware for capturing, storing, displaying, and printing color images.

There are several technical challenges involved in extrapolating monochrome image processing techniques to their color image equivalent, one of which is particularly important: the choice of the appropriate color model for the task. The impact of this choice will become evident as we look at specific examples in this section.

There are two ways to achieve color image processing:

- *Componentwise*: Having selected an appropriate color model, each component image (e.g., R, G, and B) is processed individually and then forms a composite processed image.
- *Vector Methods*: Color pixels are treated as vectors:

$$\mathbf{c}(x, y) = \begin{bmatrix} c_R(x, y) \\ c_G(x, y) \\ c_B(x, y) \end{bmatrix} = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix} \tag{16.12}$$

The two methods are equivalent if (and only if)

- The process is applicable to both vectors and scalars.
- The operation on each component of a vector is independent of the other components.

Vector methods for color image processing are mathematically intensive and beyond the scope of this text.[9] For the remaining of this discussion, we shall focus on componentwise color image processing and the role of the chosen color model.

Several color image processing techniques can be performed on the individual R, G, and B channels of the original image, whereas other techniques require access to a component that is equivalent to the monochrome version of the input image (e.g., the Y component in the YIQ color model or the V component in the HSV color model). The former can be called *RGB processing* (Figure 16.22), whereas the latter can be referred to as *intensity processing* (Figure 16.23).

16.5.1 Color Transformations

It is possible to extend the concept of grayscale transformations to color images. The original formulation (see Chapter 8)

$$g(x, y) = T[f(x, y)] \tag{16.13}$$

can be adapted for the case where the input and output images ($f(x, y)$ and $g(x, y)$) are color images, that is, where each individual pixel value is no longer an unsigned

[9]The interested reader will find a few useful references at the end of the chapter.

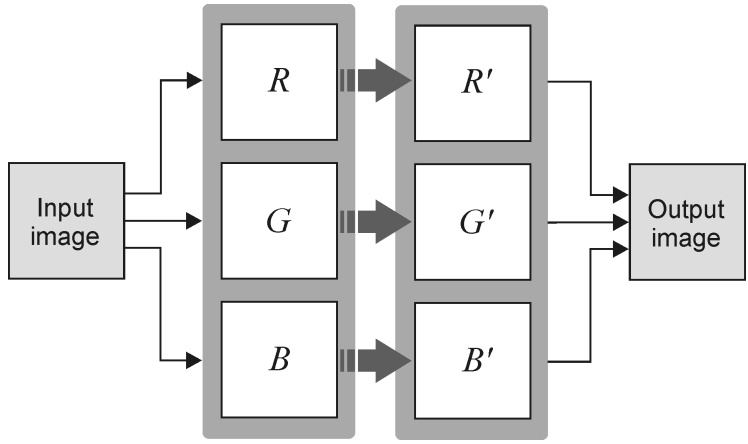

FIGURE 16.22 RGB processing.

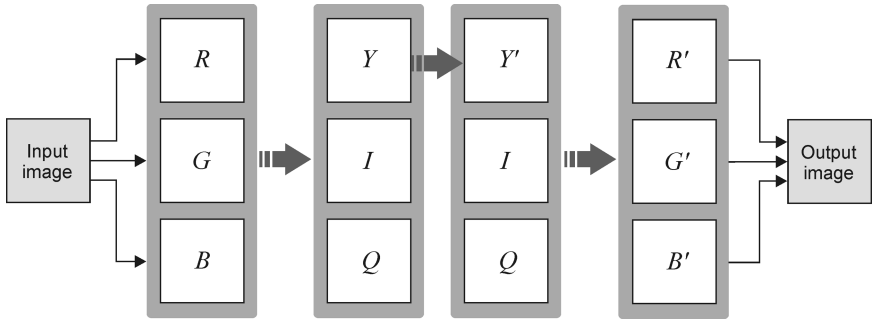

FIGURE 16.23 Intensity processing using RGB to YIQ color space conversions.

integer or double, but a triple of values (e.g., corresponding to the R, G, and B values for that pixel).

Since we know that such transformation functions are point processing functions (independent of the pixel's location or neighbors' values), we can adopt a modified version of the simplified notation introduced in Chapter 8:

$$s_i = T_i(r_1, r_2, ..., r_n), \quad i = 1, 2, ..., n \tag{16.14}$$

where r_i and s_i are the color components of the original ($f(x, y)$) and the processed ($g(x, y)$) image, respectively, n is the number of color components, and $T_1, T_2, ..., T_n$ is a set of color transformation (or *mapping*) functions that operate on r_i to produce s_i. For RGB images, $n = 3$, and r_1, r_2, and r_3 correspond to the R, G, and B values for each pixel in the input image.

Intensity Modification A simple example of color mapping function is the intensity modification function described by

$$g(x, y) = kf(x, y) \tag{16.15}$$

Clearly, if $k > 1$, the resulting image will be brighter than the original, whereas for $k < 1$, the output image will be darker than the input image.

Color Complements The color complement operation is the color equivalent of the grayscale negative transformation introduced in Chapter 8. It replaces each hue by its complement (sometimes called *opponent color*).

If the input image is represented using the RGB color model, the operation can be performed by applying a trivial transfer function to each individual color channel. If the input image is represented using the HSV color space, we must apply a trivial transfer function to V, a nontrivial transfer function—that takes care of the discontinuities in hue around 0°—to H and leave S unchanged.

In MATLAB

Color complement for RGB images can be accomplished using the `imcomplement` function.

Color Slicing Color slicing is a mapping process by which all colors outside a range of interest are mapped to a "neutral" color (e.g., gray), while all colors of interest remain unchanged. The color range of interest can be specified as a cube or a sphere centered at a prototypical reference color.

16.5.2 Histogram Processing

The concept of histogram can be extended to color images, in which case each image can be represented using three histograms with N (typically, $4 \leq N \leq 256$) bins each. Histogram techniques (e.g., histogram equalization) can be applied to an image represented using a color model that allows separation between luminance and chrominance components (e.g., HSI): the luminance (intensity, I) component of a color image is processed, while the chromaticity components (H and S) are left unchanged. Figure 16.24 shows an example that uses the YIQ (NTSC) color model (and equalizes only its Y component). The resulting image is a modified version of the original one, in which background details become more noticeable. Notice that although the colors become somewhat washed out, they are faithful to their original hue.

16.5.3 Color Image Smoothing and Sharpening

Linear neighborhood-oriented smoothing and sharpening techniques can be extended to color images under the componentwise paradigm. The original 3×3 kernel typically used in monochrome operations (see Chapters 4 and 10) becomes an array of vectors (Figure 16.25).

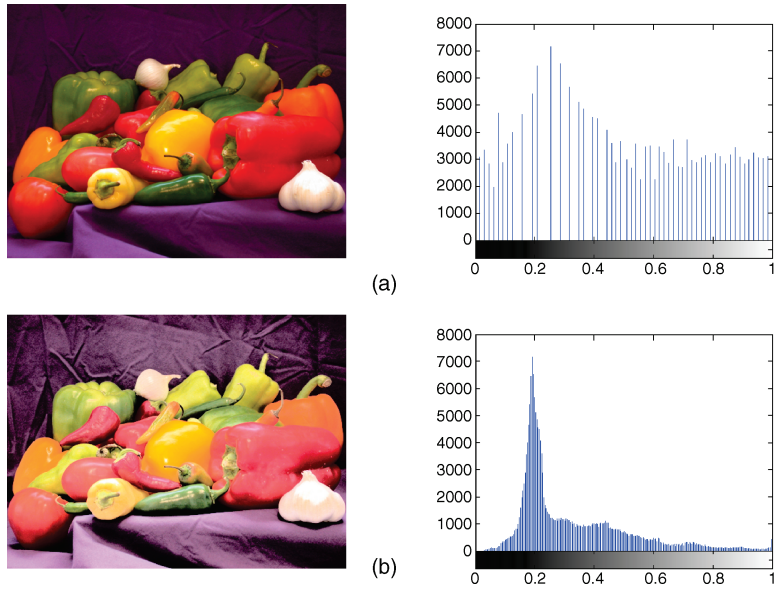

FIGURE 16.24 Example of color histogram equalization. (a) Original image and its Y channel histogram; (b) output image and its equalized Y channel histogram. Original image: courtesy of MathWorks.

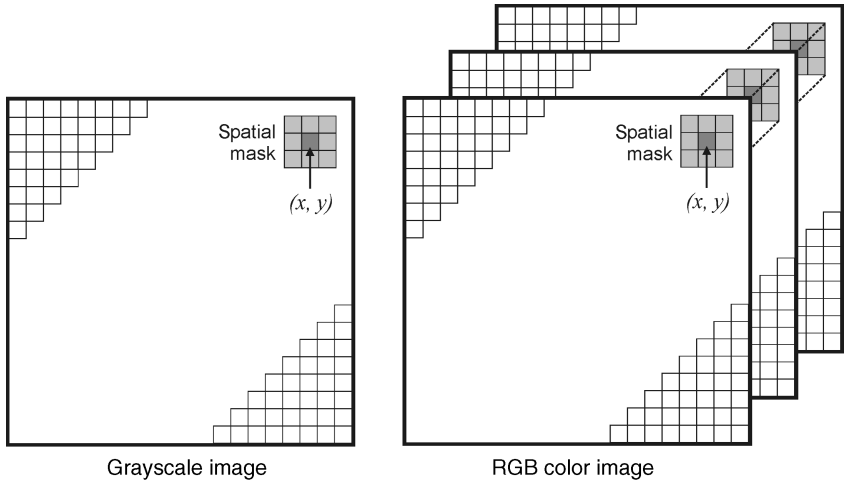

FIGURE 16.25 Spatial convolution masks for grayscale and RGB color images.

For example, the vector formulation of the averaging filter for an RGB image using a neighborhood S_{xy} centered around coordinates (x, y) becomes

$$\bar{\mathbf{c}}(x, y) = \begin{bmatrix} \frac{1}{K} \sum_{(s,t)\in S_{xy}} R(s, t) \\ \frac{1}{K} \sum_{(s,t)\in S_{xy}} G(s, t) \\ \frac{1}{K} \sum_{(s,t)\in S_{xy}} B(s, t) \end{bmatrix} \tag{16.16}$$

Equation (16.16) indicates that the result can be obtained by performing neighborhood averaging on each individual color channel using standard grayscale neighborhood processing. Similarly, sharpening a color image encoded using the RGB color model can be accomplished by applying the sharpening operator (e.g., Laplacian) to each component image individually and combining the results.

Smoothing and sharpening operations can also be performed by processing the intensity (luminance) component of an image encoded with the proper color model (e.g., YIQ or HSI) and combining the result with the original chrominance channels. Tutorial 16.2 explores smoothing and sharpening of color images.

16.5.4 Color Noise Reduction

The impact of noise on color images strongly depends on the color model used. Even when only one of the R, G, or B channels is affected by noise, conversion to another color model such as HSI or YIQ will spread the noise to all components. Linear noise reduction techniques (such as the mean filter) can be applied on each R, G, and B component separately with good results.

16.5.5 Color-Based Image Segmentation

Color Image Segmentation by Thresholding There are several possible ways to extend the image thresholding ideas commonly used for monochrome images (Chapter 15) to their color equivalent. The basic idea is to partition the color space into a few regions (which hopefully should correspond to meaningful objects and regions in the image) using appropriately chosen thresholds.

A simple option is to define one (or more) threshold(s) for each color component (e.g., R, G, and B), which results in a partitioning of the RGB cube from which the color range of interest (a smaller cube) can be isolated (Figure 16.26).

Color Image Segmentation in RGB Vector Space It is also possible to specify a threshold relative to a distance between any color and a reference color in the RGB space. If we call the reference color (R_0, G_0, B_0), the thresholding rule can be expressed as

$$g(x, y) = \begin{cases} 1 & d(x, y) \leq d_{\max} \\ 0 & d(x, y) > d_{\max} \end{cases} \tag{16.17}$$

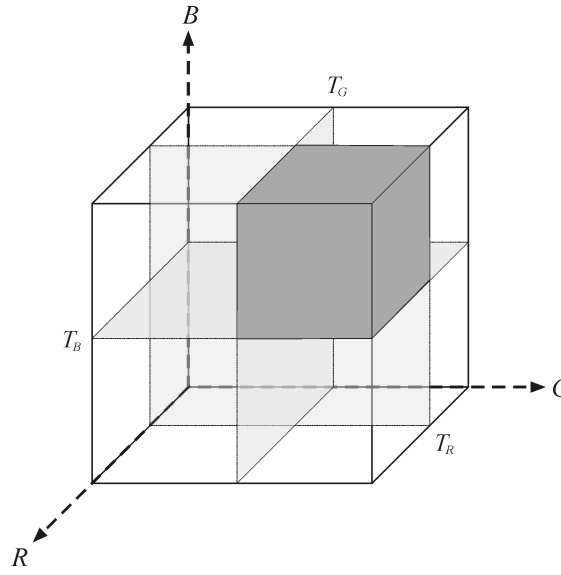

FIGURE 16.26 Thresholding in RGB space.

where

$$d(x, y) = \sqrt{[f_R(x, y) - R_0]^2 + [f_G(x, y) - G_0]^2 + [f_B(x, y) - B_0]^2} \quad (16.18)$$

Thresholding according to equations (16.17) and (16.18) in fact specifies a sphere in the RGB space, whose center is the reference color. Any pixel whose color lies inside the sphere (or on its surface) will be set to 1; all other pixels will be assigned a value of 0.

Equations (16.17) and (16.18) can be generalized by specifying different threshold values for each primary color, which will result in an ellipsoid (rather than a sphere) being defined in RGB space (Figure 16.27).

In MATLAB

A simple way to segment color images in MATLAB is by using the `rgb2ind` function. The primary use of this function is to generate an indexed image based on an input truecolor image, but with less number of colors. When a specific number of colors are specified, MATLAB quantizes the image and produces an indexed image and a color map at the output. The resulting indexed image is essentially what we want from a segmentation process: a labeled image. If there are n regions in the image, there will be $n + 1$ labels, where the additional label is for the background.

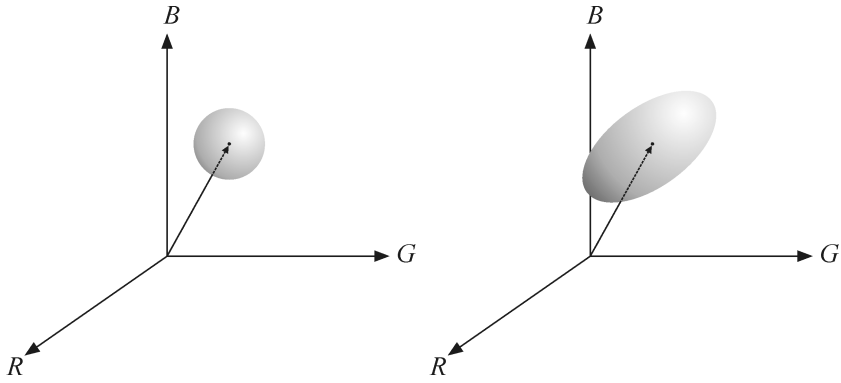

FIGURE 16.27 Defining spherical (ellipsoidal) regions in RGB space.

■ EXAMPLE 16.4

In this example, we use `rgb2ind` to segment a simple image containing three predominant colors: black (background), red (marker), and desaturated yellow (marker cap). Figure 16.28 shows the input image (a), the result of segmentation using `rgb2ind` and the default color map (b), and the result of segmentation displayed with a different color map (c). With few exceptions (caused by light reflections on the red marker), the algorithm did a good job, segmenting the input image into three main parts.

```
I = imread('marker.png');
n = 3;
[I2,map2] = rgb2ind(I,n,'nodither');
imshow(I)
figure, imshow(I2,map2)
figure, imshow(I2,hsv(3))
```

FIGURE 16.28 Example of color segmentation using requantization.

(a) (b) (c)

FIGURE 16.29 Another example of color segmentation using requantization: (a) original image; (b) requantized image with two color levels; (c) requantized image with five color levels.

In Figure 16.29, we replace the input image with a more difficult scene and show results for different values of n. The results are good: $n = 2$ provides good rough separation between "flower" and "leaves," whereas $n = 5$ segments the yellow filaments as well. Note that the uneven lighting pattern causes most of the imperfect results.

16.5.6 Color Edge Detection

In Chapter 14, we defined an edge as a boundary between two image regions having distinct characteristics. We then focused on edges in grayscale 2D images, which are relatively easy to define as "sharp variations of the intensity function across a portion of the image." In this section, we briefly look at the problem of color edge detection.

Several definitions of a color edge have been proposed in the literature [Pra07]:

1. An edge in a color image can be said to exist if and only if there is such an edge in the luminance channel. This definition ignores discontinuities in the other color channels (e.g., hue and saturation for *HSV* images).
2. A color edge is present if an edge exists in any of its three component images.
3. An edge exists if the sum of the results of edge detection operators applied to individual color channels exceeds a certain threshold. This definition is convenient and straightforward and commonly used when the emphasis is not on accuracy.

In general, extending the gradient-based methods for edge detection in monochrome images described in Chapter 14 to color images by simply computing the gradients for each component image and combining the results (e.g., by applying a logical OR operator) will lead to erroneous—but acceptable for most cases—results. If accuracy is important, we must resort to a new definition of the gradient that is applicable to vector quantities (e.g., the method proposed by Di Zenzo [DZ86]), which is beyond the scope of this book.

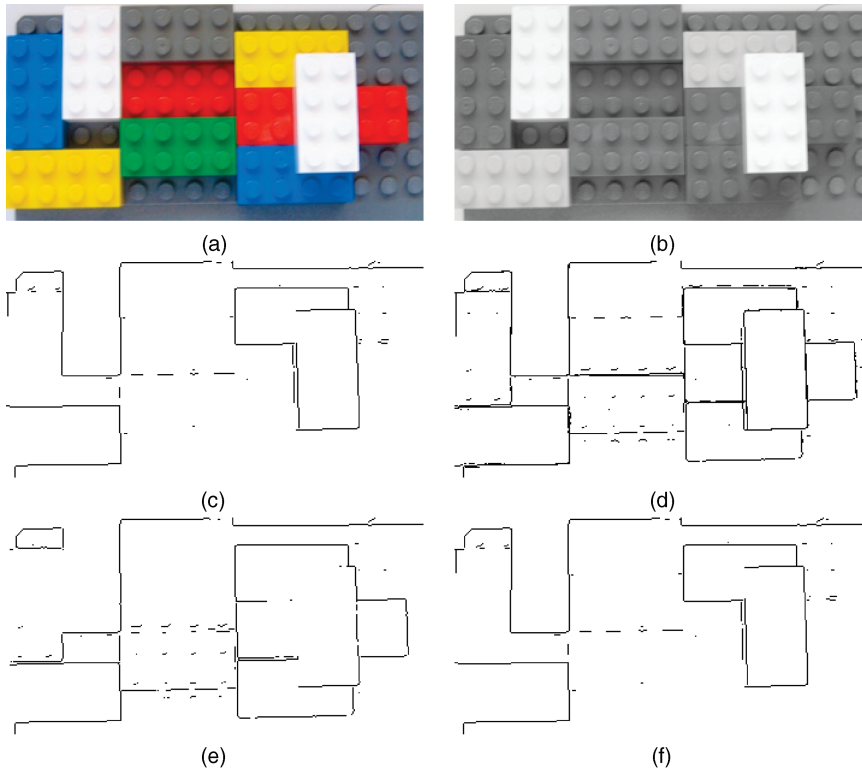

FIGURE 16.30 Color edge detection example: (a) original image; (b) grayscale equivalent; (c) edge detection on (b); (d) edge detection on individual RGB components; (e) edge detection on Y component only; (f) edge detection on V component only.

■ EXAMPLE 16.5

In this example, we show the results of using the `edge` function for different color models. More specifically, we read an RGB color image, convert it to gray, HSV, and YIQ representations, and compare the results (Figure 16.30) obtained when using

- Edge detection on the grayscale equivalent of the input image (c).
- The logical OR of edge detection applied to each individual channel in the original image (R, G, and B) (d).
- Edge detection on the Y component of the input image represented in the YIQ model (e).
- Edge detection on the V component of the input image represented in the HSV model (f).

16.6 TUTORIAL 16.1: PSEUDOCOLOR IMAGE PROCESSING

Goal

The goal of this tutorial is to learn how to display grayscale images using pseudocolors in MATLAB.

Objectives

- Learn how to use the `grayslice` function to perform intensity slicing.
- Learn how to specify color maps with a custom number of colors.

What You Will Need

- `grad.jpg`
- `mri.jpg`

Procedure

We will start by exploring the `grayslice` function on a gradient image.

1. Create and display a gradient image.

```
I = repmat(uint8([0:255]),256,1);
figure, subplot(1,2,1), subimage(I), title('Original Image');
```

2. Slice the image and display the results.

```
I2 = grayslice(I,16);
subplot(1,2,2), subimage(I2,colormap(winter(16))), ...
    title('Pseudo-colored with "winter" colormap')
```

Question 1 Why did we use the `subimage` function to display the images (instead of the familiar `imshow`)?

Question 2 What does the value 16 represent in the function call for `grayslice`?

Question 3 In the statement `subimage(I2,colormap(winter(16)))`, what does the value 16 represent?

In the above procedure, we sliced the image into equal partitions—this is the default for the `grayslice` function. We will now learn how to slice the range of grayscale values into unequal partitions.

3. Slice the image into unequal partitions and display the result.

```
levels = [0.25*255, 0.75*255, 0.9*255];
I3 = grayslice(I,levels);
figure, imshow(I3,spring(4))
```

Question 4 The original image consists of values in the range [0, 255]. If our original image values ranged [0.0, 1.0], how would the above code change?

Now that we have seen how pseudocoloring works, let us apply it to an image where this visual information might be useful.

4. Clear all variables and close any open figures.
5. Load and display the `mri.jpg`.

```
I = imread('mri.jpg');
figure, subplot(1,2,1), subimage(I), title('Original Image');
```

6. Pseudocolor the image.

```
I2 = grayslice(I,16);
subplot(1,2,2), subimage(I2, colormap(jet(16))), ...
    title('Pseudo-colored with "jet" colormap');
```

Question 5 In the previous steps, we have specified how many colors we want in our color map. If we do not specify this number, how does MATLAB determine how many colors to return in the color map?

16.7 TUTORIAL 16.2: FULL-COLOR IMAGE PROCESSING

Goal

The goal of this tutorial is to learn how to convert between color spaces and perform filtering on color images in MATLAB.

Objectives

- Learn how to convert from RGB to HSV color space using the `rgb2hsv` function.
- Learn how to convert from HSV to RGB color space using the `hsv2rgb` function.
- Explore smoothing and sharpening in the RGB and HSV color spaces.

Procedure

We will start by exploring the `rgb2hsv` function.

1. Load the `onions.png` image and display its RGB components.

```
I = imread('onion.png');
figure, subplot(2,4,1), imshow(I), title('Original Image');
subplot(2,4,2), imshow(I(:,:,1)), title('R component');
subplot(2,4,3), imshow(I(:,:,2)), title('G component');
subplot(2,4,4), imshow(I(:,:,3)), title('B component');
```

2. Convert the image to HSV and display its components.

```
Ihsv = rgb2hsv(I);
subplot(2,4,6), imshow(Ihsv(:,:,1)), title('Hue')
subplot(2,4,7), imshow(Ihsv(:,:,2)), title('Saturation');
subplot(2,4,8), imshow(Ihsv(:,:,3)), title('Value');
```

Question 1 Why do we not display the HSV equivalent of the image?

When viewing the components of an RGB image, the grayscale visualization of each component is intuitive because the intensity within that component corresponds to how much of the component is being used to generate the final color. Visualization of the components of an HSV image is not as intuitive. You may have noticed that when displaying the hue, saturation, and value components, hue and saturation do not give you much insight as to what the actual color is. The value component, on the other hand, appears to be a grayscale version of the image.

3. Convert the original image to grayscale and compare it with the value component of the HSV image.

```
Igray = rgb2gray(I);
figure, subplot(1,2,1), imshow(Igray), title('Grayscale');
subplot(1,2,2), imshow(Ihsv(:,:,3)), title('Value component');
```

Question 2 How does the grayscale version of the original image and the value component of the HSV image compare?

Procedures for filtering a color image will vary depending on the color space being used. Let us first learn how to apply a smoothing filter on an RGB image.

4. Apply a smoothing filter to each component and then reconstruct the image.

```
fn = fspecial('average');
I2r = imfilter(I(:,:,1), fn);
I2g = imfilter(I(:,:,2), fn);
I2b = imfilter(I(:,:,3), fn);
I2(:,:,1) = I2r;
I2(:,:,2) = I2g;
```

```
I2(:,:,3) = I2b;
figure, subplot(1,2,1), imshow(I), title('Original Image');
subplot(1,2,2), imshow(I2), title('Averaged Image');
```

Question 3 Do the results confirm that the RGB equivalent of averaging a grayscale image is to average each component of the RGB image individually?

Now let us see what happens when we perform the same operations on an HSV image. Note that in these steps, we will use the `hsv2rgb` function to convert the HSV image back to RGB so that it can be displayed.

5. Filter all components of the *HSV* image.

```
Ihsv2h = imfilter(Ihsv(:,:,1), fn);
Ihsv2s = imfilter(Ihsv(:,:,2), fn);
Ihsv2v = imfilter(Ihsv(:,:,3), fn);
Ihsv2(:,:,1) = Ihsv2h;
Ihsv2(:,:,2) = Ihsv2s;
Ihsv2(:,:,3) = Ihsv2v;
```

6. Display the results.

```
figure, subplot(2,3,1), imshow(Ihsv(:,:,1)), ...
    title('Original Hue');
subplot(2,3,2), imshow(Ihsv(:,:,2)), ...
    title('Original Saturation');
subplot(2,3,3), imshow(Ihsv(:,:,3)), ...
    title('Original Value');
subplot(2,3,4), imshow(Ihsv2(:,:,1)), ...
    title('Filtered Hue');
subplot(2,3,5), imshow(Ihsv2(:,:,2)), ...
    title('Filtered Saturation');
subplot(2,3,6), imshow(Ihsv2(:,:,3)), ...
    title('Filtered Value');
figure, subplot(1,2,1), imshow(I), title('Original Image');
subplot(1,2,2), imshow(hsv2rgb(Ihsv2)), ...
    title('HSV with all components filtered');
```

Question 4 Based on the results, does it make sense to say that the HSV equivalent of averaging a grayscale image is to average each component of the HSV image individually?

7. Filter only the value component and display the results.

```
Ihsv3(:,:,[1 2]) = Ihsv(:,:,[1 2]);
Ihsv3(:,:,3) = Ihsv2v;
figure, subplot(1,2,1), imshow(I), title('Original Image');
subplot(1,2,2), imshow(hsv2rgb(Ihsv3)), ...
    title('HSV with only value component filtered');
```

Question 5 How does this result compare with the previous one?

We can sharpen an HSV image following a similar sequence of steps.

8. Sharpen the HSV image and display the result.

```
fn2 = fspecial('laplacian',0);
Ihsv4v = imfilter(Ihsv(:,:,3), fn2);
Ihsv4(:,:,[1 2]) = Ihsv(:,:,[1 2]);
Ihsv4(:,:,3) = imsubtract(Ihsv(:,:,3),Ihsv4v);
figure, subplot(1,2,1), imshow(I), title('Original Image');
subplot(1,2,2), imshow(hsv2rgb(Ihsv4)), ...
    title('HSV sharpened');
```

Question 6 How would we perform the same sharpening technique on an RGB image?

WHAT HAVE WE LEARNED?

- This chapter introduced the most important concepts and terms related to color perception, representation, and processing. Understand the meaning of the main terms in colorimetry is a required step toward the understanding of color image processing.
- There are many color models used to represent and quantify color information. Some of the most popular (and their use) models are as follows:
  - RGB, CMY(K): display and printing devices
  - YIQ, YCbCr: television and video
  - XYZ: color standardization
  - CIELAB, CIELUV: perceptual uniformity
  - HSV, HSI, HSL: intuitive description of color properties
- Color images can be represented in MATLAB either as an $M \times N \times 3$ array (one per color channel) or as an $M \times N$ array of indices (pointers) to a secondary (usually 256×3) color palette. The former representation is called an *RGB image*, whereas the latter is called an *indexed image*.
- Pseudocolor image processing techniques assign color to pixels based on an interpretation of the data rather than the original scene color (which may not even be known). Full-color image processing methods, on the other hand, process

the pixel values of images whose colors usually correspond to the color of the original scene.

- Several monochrome image processing techniques, from edge detection to histogram equalization, can be extended to color images. The success of applying such techniques to color images depends on the choice of color model used to represent the images.

LEARN MORE ABOUT IT

The topics of color vision, colorimetry, color science, and color image processing have been covered at great length elsewhere. A modest list of useful references is as follows.

- For a deeper understanding of color vision, we recommend Chapter 5 of [Pal99].
- The book by Wyszecki and Styles [WS82] is considered a reference in the field of color science.
- The book by Westland and Ripamonti [WR04] combines color science and colorimetry principles with MATLAB.
- Chapter 3 of [Pra07] contains an in-depth analysis of colorimetry concepts and mathematical formulation.
- There are several books devoted entirely to color image processing, such as [LP06].
- Special issues of leading journals in the field devoted to this topic include [TSV05], [Luk07], and [TTP08].
- Chapter 12 of [BB08] covers color spaces and color space conversion in great detail.
- Chapter 6 of [GWE04] includes a GUI-based MATLAB tool for color image processing, transformations, and manipulations.

ON THE WEB

- The Color Model Museum (an exhibit of color representation models and diagrams dating back to the sixteenth century).
  http://www.colorcube.com/articles/models/model.htm
- The *Colour and Vision Research Laboratories* at the Institute of Ophthalmology (London, England): a rich repository of standard data sets relevant to color and vision research.
  http://www.cvrl.org/
- The Color FAQ document by Charles Poynton: an excellent reference on colorimetry and its implications in the design of image and video systems.
  http://poynton.com/ColorFAQ.html

- International Color Consortium.
  http://www.color.org

16.8 PROBLEMS

16.1 Use the MATLAB function `patch` to display the *RGB* cube in Figure 16.9.

16.2 Write MATLAB code to add two RGB color images. Test it with two test images (of the same size) of your choice. Are the results what you had expected?

16.3 What is wrong with the following MATLAB code to add an indexed color image to a constant for brightening purposes? Fix the code to achieve the desired goal.

```
[X, map] = imread('canoe.tif');
X = X + 20;
```

16.4 In our discussion of pseudocoloring, we stated that the intensity slicing method described in Section 16.4.1 is a particular case of the more general method (using transformation functions) described in Section 16.4.2.

Assuming that a 256-level monochrome image has been "sliced" into four colors, red, green, blue, and yellow, and that each range of gray levels has the same width (64 gray levels), plot the (staircase-shaped) transformation functions for each color channel (R, G, and B).

16.5 In our discussion of color histogram equalization (Section 16.5.2), we showed an example in which the equalization technique was applied to the Y channel of an image represented using the YUQ color model. Explain (using MATLAB) what happens if the original image were represented using the RGB color model and each color channel underwent histogram equalization individually.

16.6 Use the `edge` function in MATLAB and write a script to compute and display the edges of a color image for the following cases:

- **(a)** RGB image, combining the edges from each color channel by adding them up.
- **(b)** RGB image, combining the edges from each color channel with a logical OR operation.
- **(c)** YIQ image, combining the edges from each color channel by adding them up.
- **(d)** YIQ image, combining the edges from each color channel with a logical OR operation.

16.7 Repeat Problem 16.6 for noisy versions of the input color images.