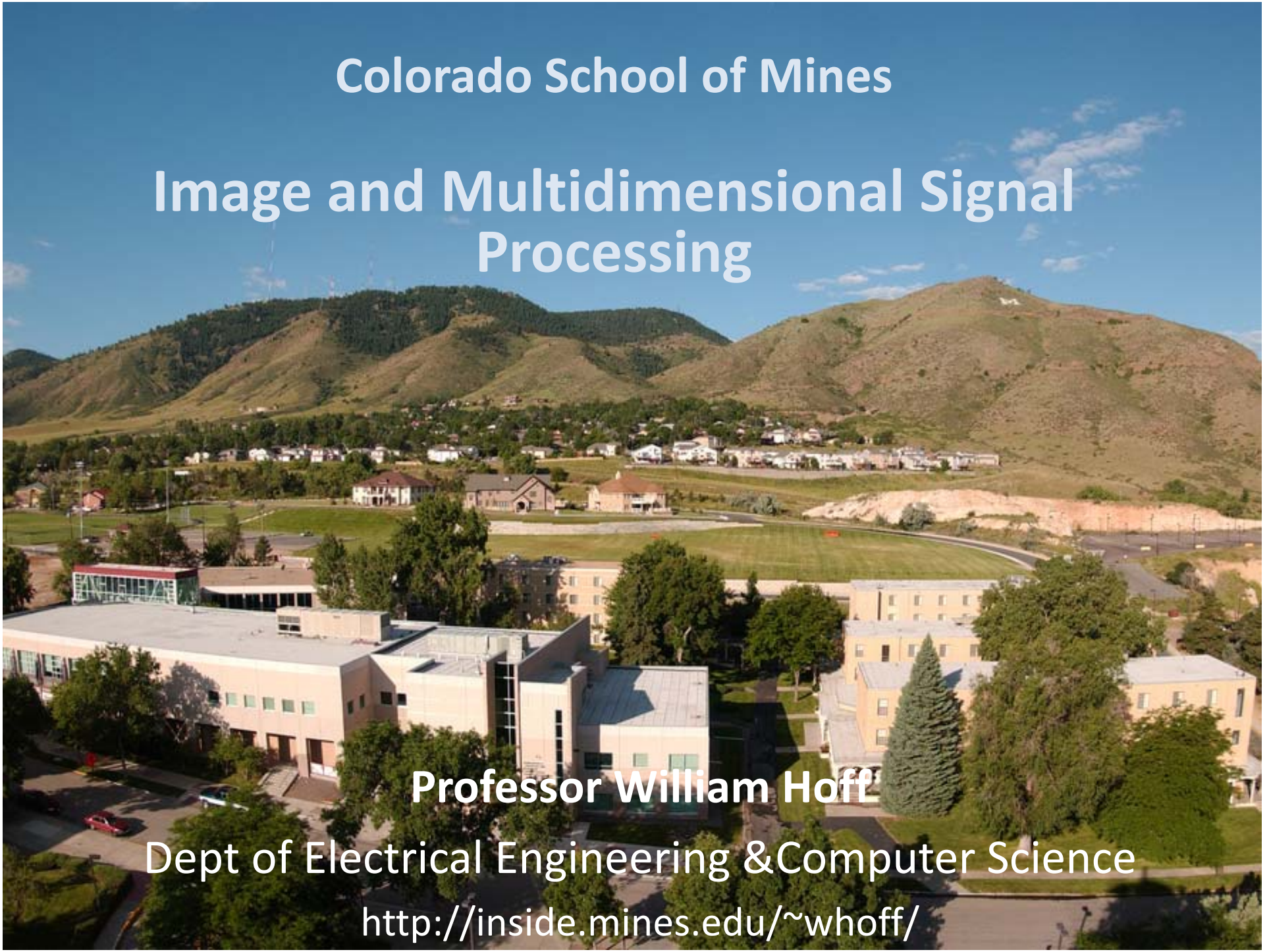

Colorado School of Mines
Image and Multidimensional Signal Processing
Professor William Hoff
Dept of Electrical Engineering &Computer Science
http://inside.mines.edu/~whoff/

Color

Color

- Image values are a vector instead of a scalar
- Example: I(x,y) = (red, green, blue)
- Most gray scale methods are directly applicable to color images

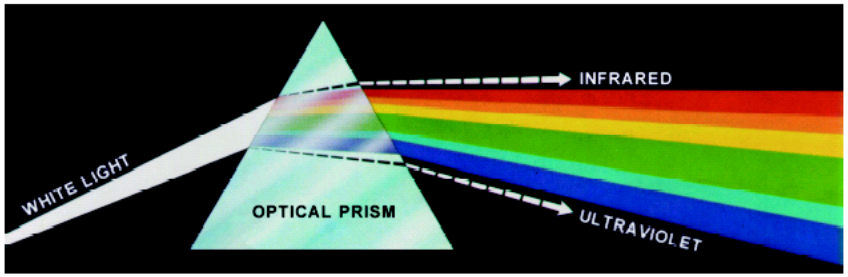


FIGURE 6.1 Color spectrum seen by passing white light through a prism. (Courtesy of the General Electric Co., Lamp Business Division.)

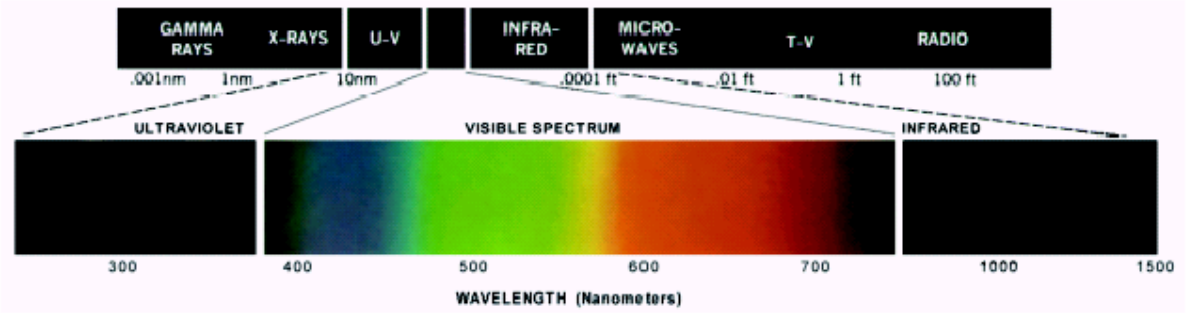
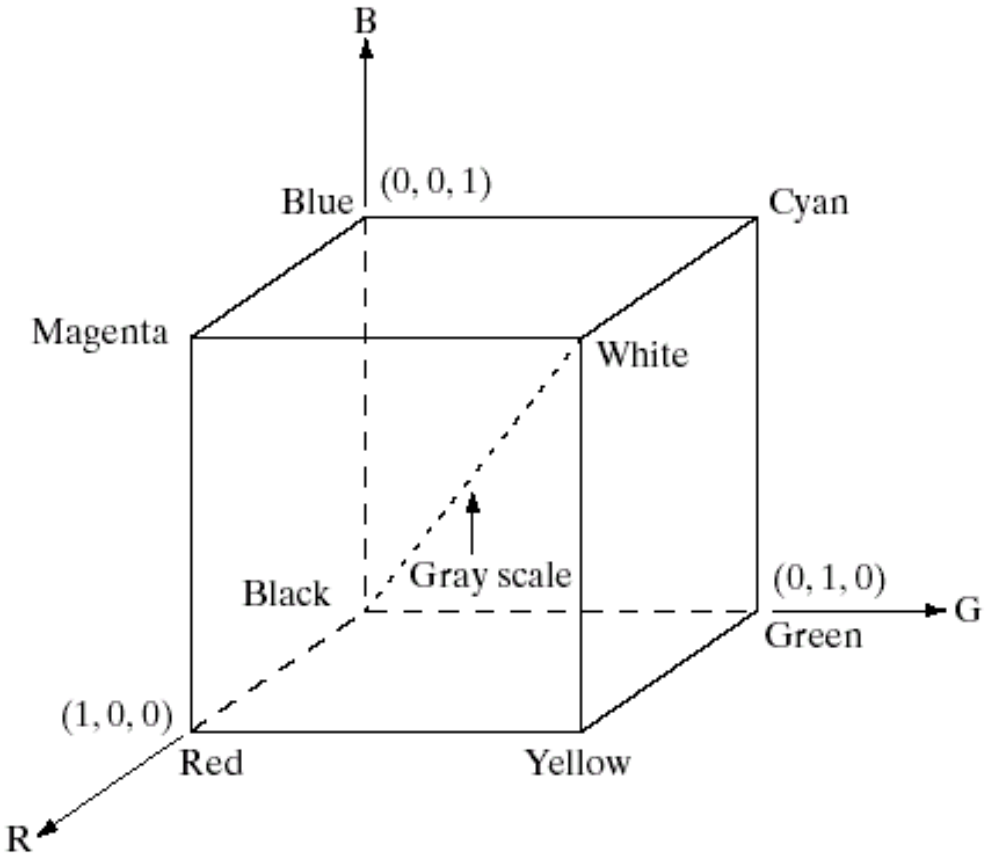
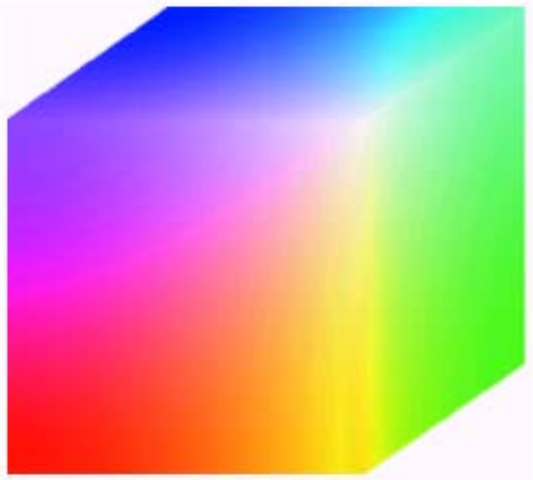


FIGURE 6.2 Wavelengths comprising the visible range of the electromagnetic spectrum. (Courtesy of the General Electric Co., Lamp Business Division.)

Human vision can perceive about 400-700 nm

RGB Color Model



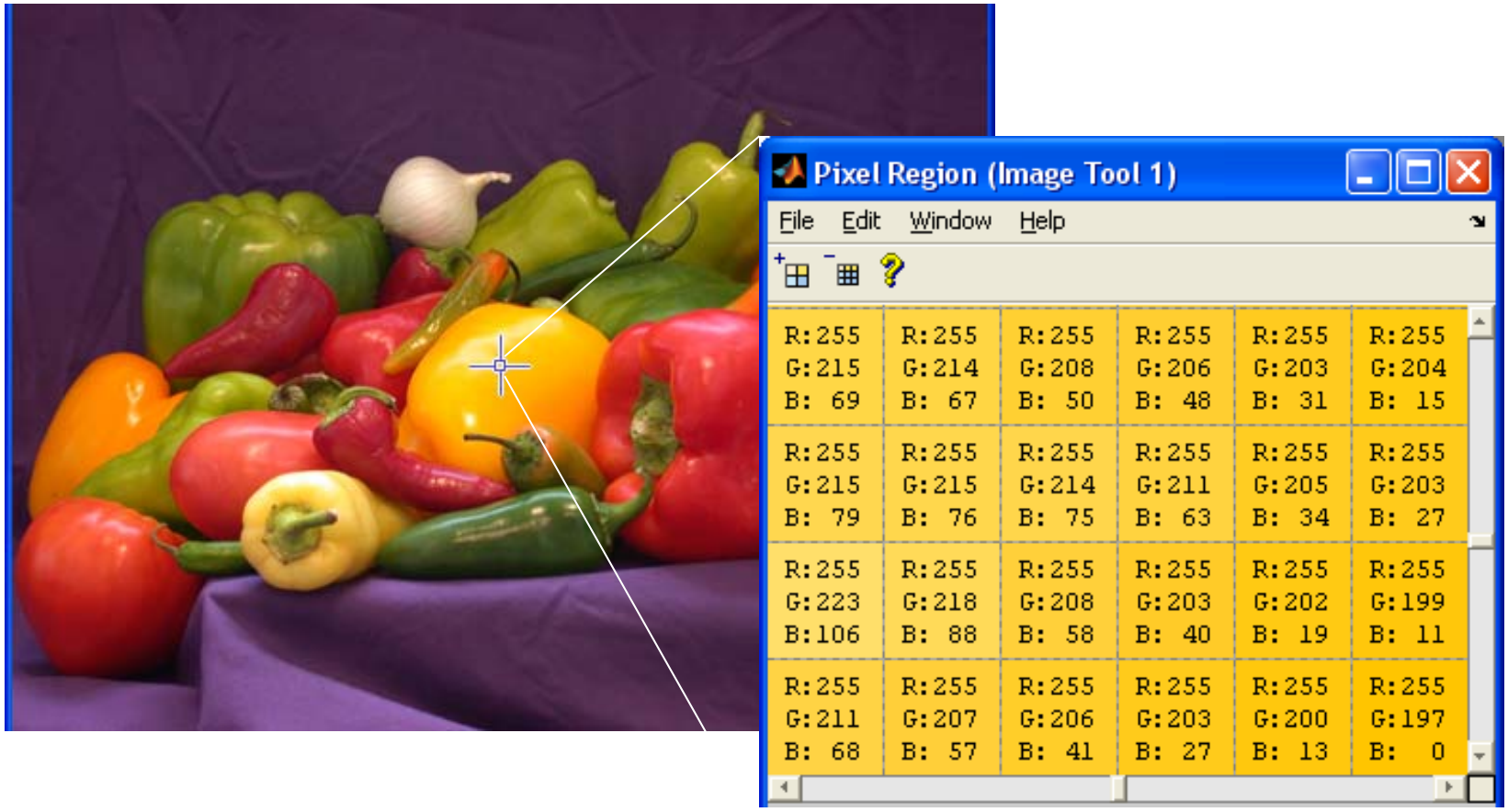


Storage of Color Images

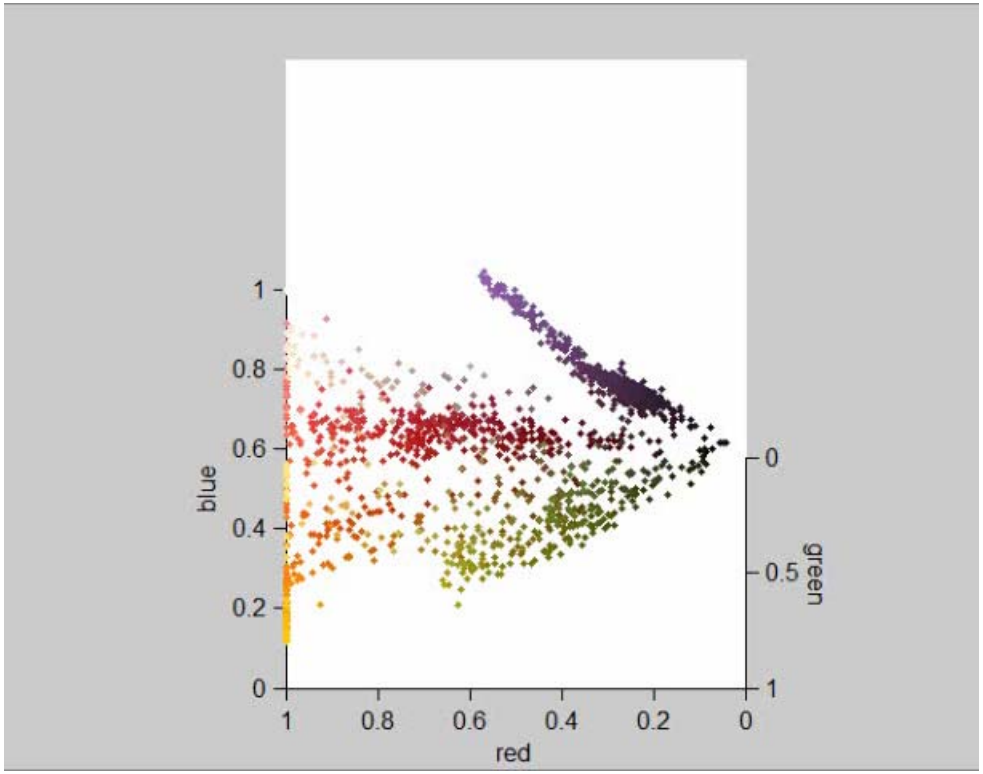
- Separate values for R,G,B
  - Example: f(x,y,c), where c=1,2,3
- Using a colormap
  - Image f(x,y) stores indices into a lookup table (colormap)
  - Colormap specifies RGB for each index

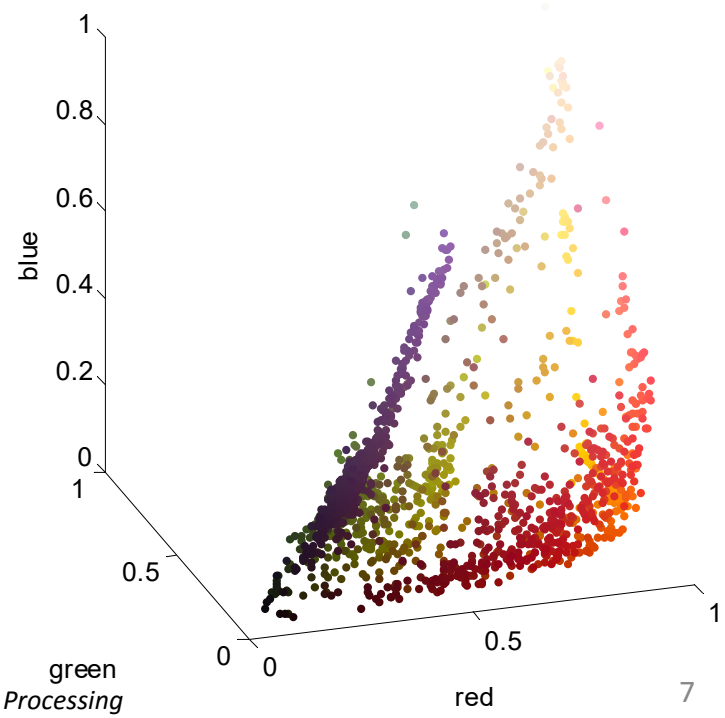
RGB Storage

- Separate values for R,G,B
  - Example: f(x,y,c), where c=1,2,3



Visualizing RGB Image Values





Principal Component Analysis (PCA)

- See if PCA can represent the RGB image more concisely, using fewer than 3 values per pixel
- We treat the image as a collection of vectors; each vector represents a pixel (its R,G,B values)
- We compute the covariance matrix of this collection of vectors
- The eigenvectors of the covariance matrix are the principal components

```
RGB = im2double(imread('peppers.png'));

% Convert 3-dimensional array array to 2D, where each row is a pixel (RGB)
X = reshape(RGB, [], 3);
N = size(X,1);  % N is the number of pixels

% Get mean and covariance
mx = mean(X);
Cx = cov(X);
```

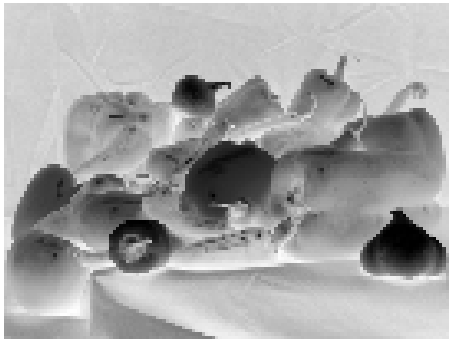
Principal Component Analysis (PCA)

- We project the original input vectors onto the space of principal components, using

$$\mathbf{y} = \mathbf{A}\left(\mathbf{x} - \mathbf{m}_{\mathbf{x}}\right)$$

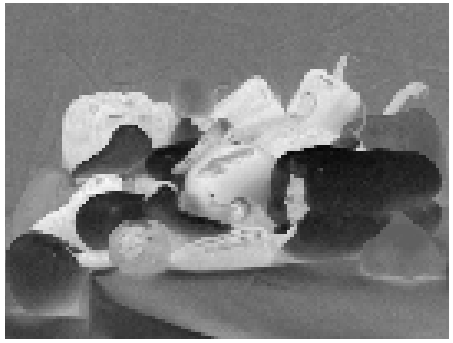
- Here are the y vectors, shown as images

y1



y2

y3



Principal Component Analysis (PCA)

- We reconstruct the original input vectors using only the first two principal components, using

$$\mathbf{x'} = \mathbf{A}_k^T\mathbf{y} + \mathbf{m}_x$$

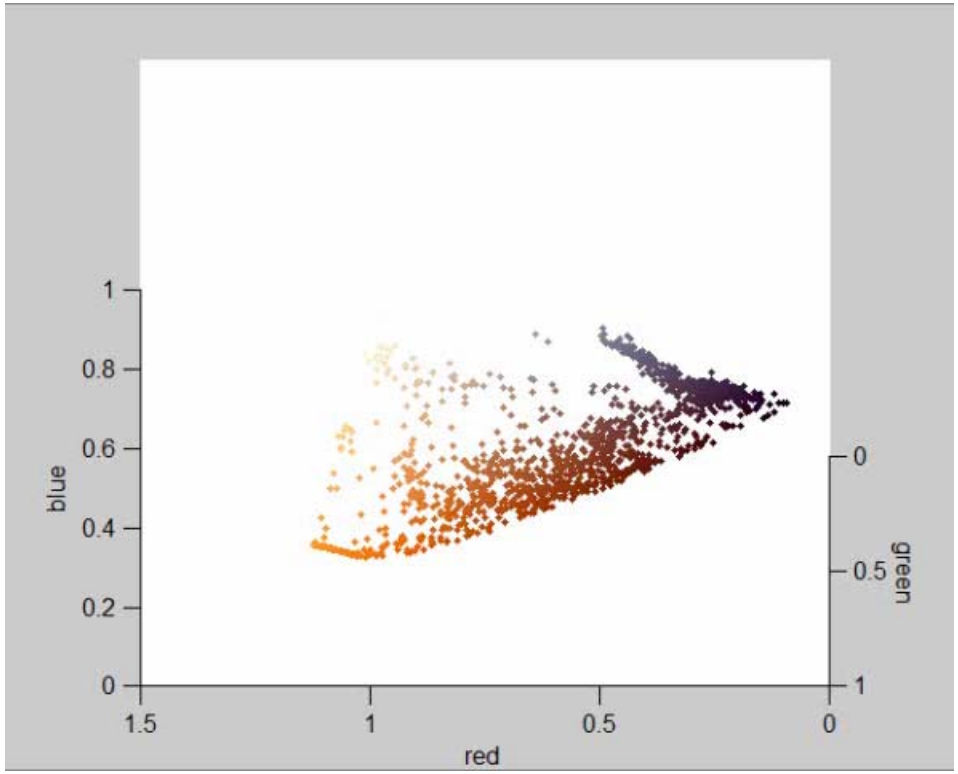
- Here are the reconstructed vectors, shown as an RGB image

Original

Reconstructed

Visualizing RGB Image Values

- Reconstructed image RGB vectors



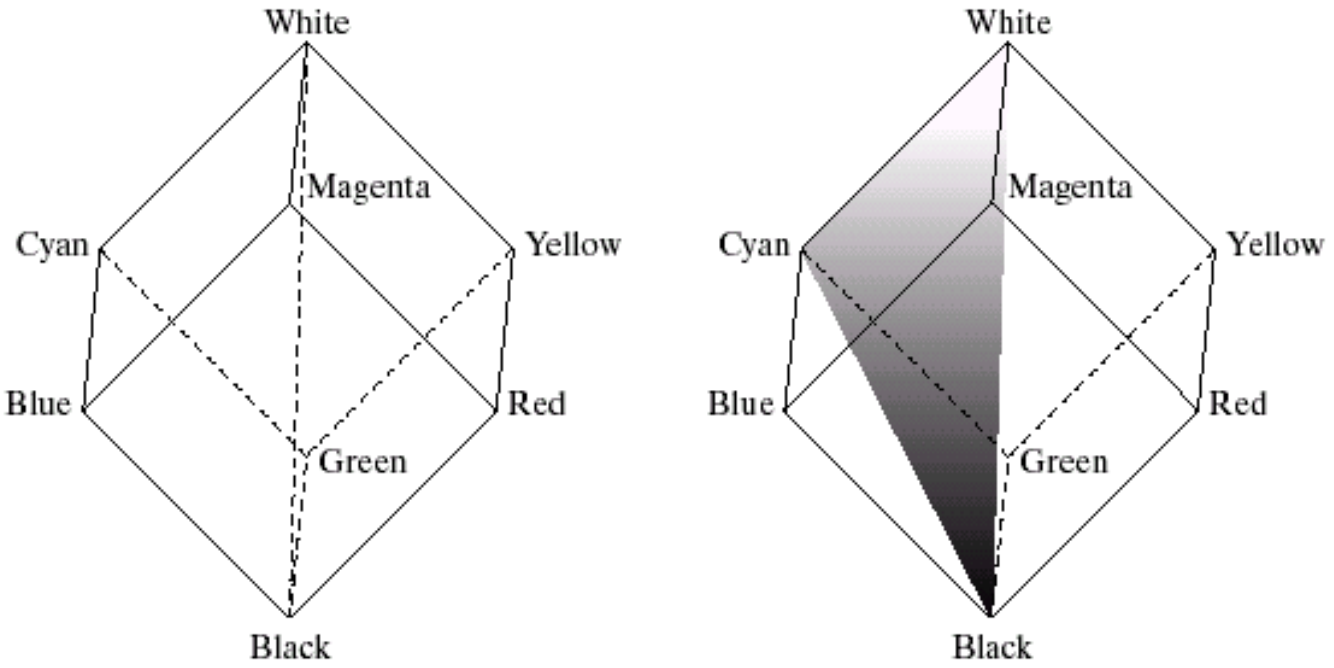
Code for generating plots

- This generates a plot of the RGB vectors, and creates an “avi” format movie

```
% Plot pixels in color space
figure
hold on
for i=1:100:size(X,1)
    mycolor = X(i,:);
    mycolor = max(mycolor, [0 0 0]);
    mycolor = min(mycolor, [1 1 1]);
    plot3(X(i, 1), X(i, 2), X(i, 3), ...
        '.', 'Color', mycolor);
end
xlabel('red'), ylabel('green'), zlabel('blue');
xlim([0 1]), ylim([0 1]), zlim([0 1]);
hold off
axis equal
movie1 = avifile('movie1.avi', 'compression', 'None', 'fps', 15);
for az=-180:3:180
    view(az,30);    % set azimuth, elevation
    drawnow;
    F = getframe(gcf);
    movie1 = addframe(movie1,F);
end
movie1 = close(movie1);
```

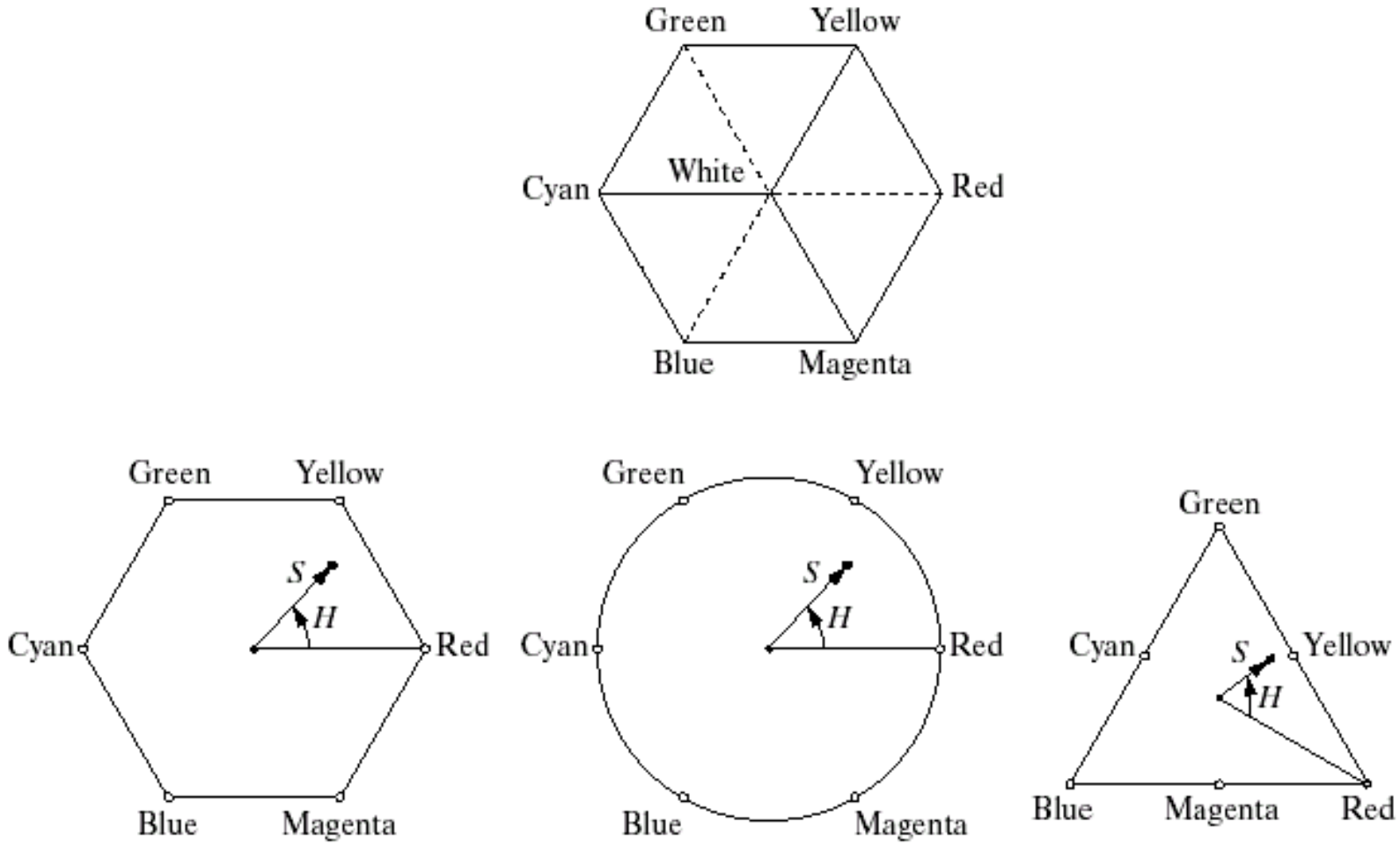
HSI Color Model

- Hue, saturation, intensity



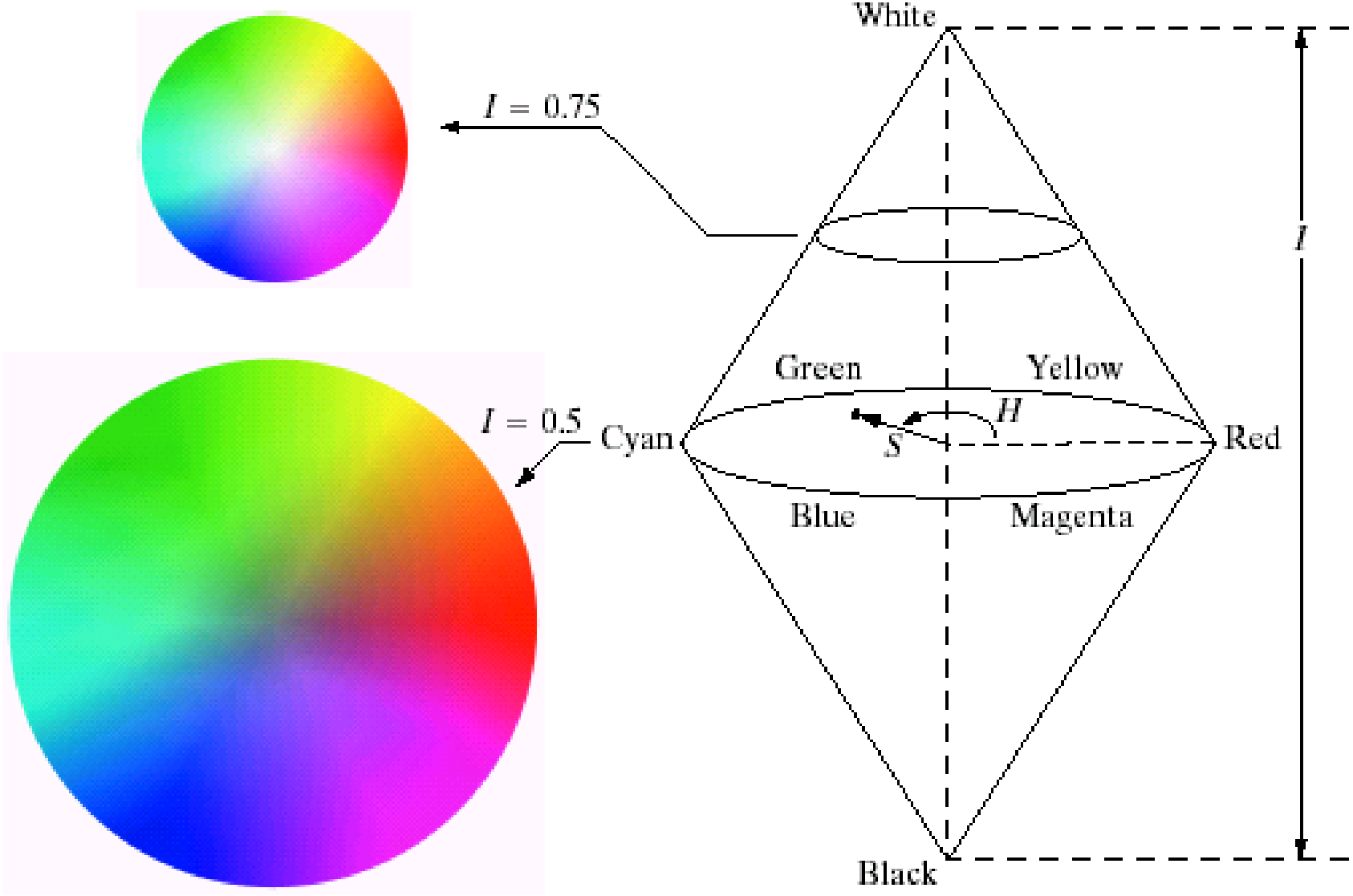
a b

FIGURE 6.12 Conceptual relationships between the RGB and HSI color models.



a
b c d

FIGURE 6.13 Hue and saturation in the HSI color model. The dot is an arbitrary color point. The angle from the red axis gives the hue, and the length of the vector is the saturation. The intensity of all colors in any of these planes is given by the position of the plane on the vertical intensity axis.


White
I = 0.75
I
Green
Yellow
I = 0.5
Cyan
H
S
Red
Blue
Magenta
Black

Conversion from RGB to HSI

$$I = \tfrac{1}{3}\left(R+G+B\right)$$

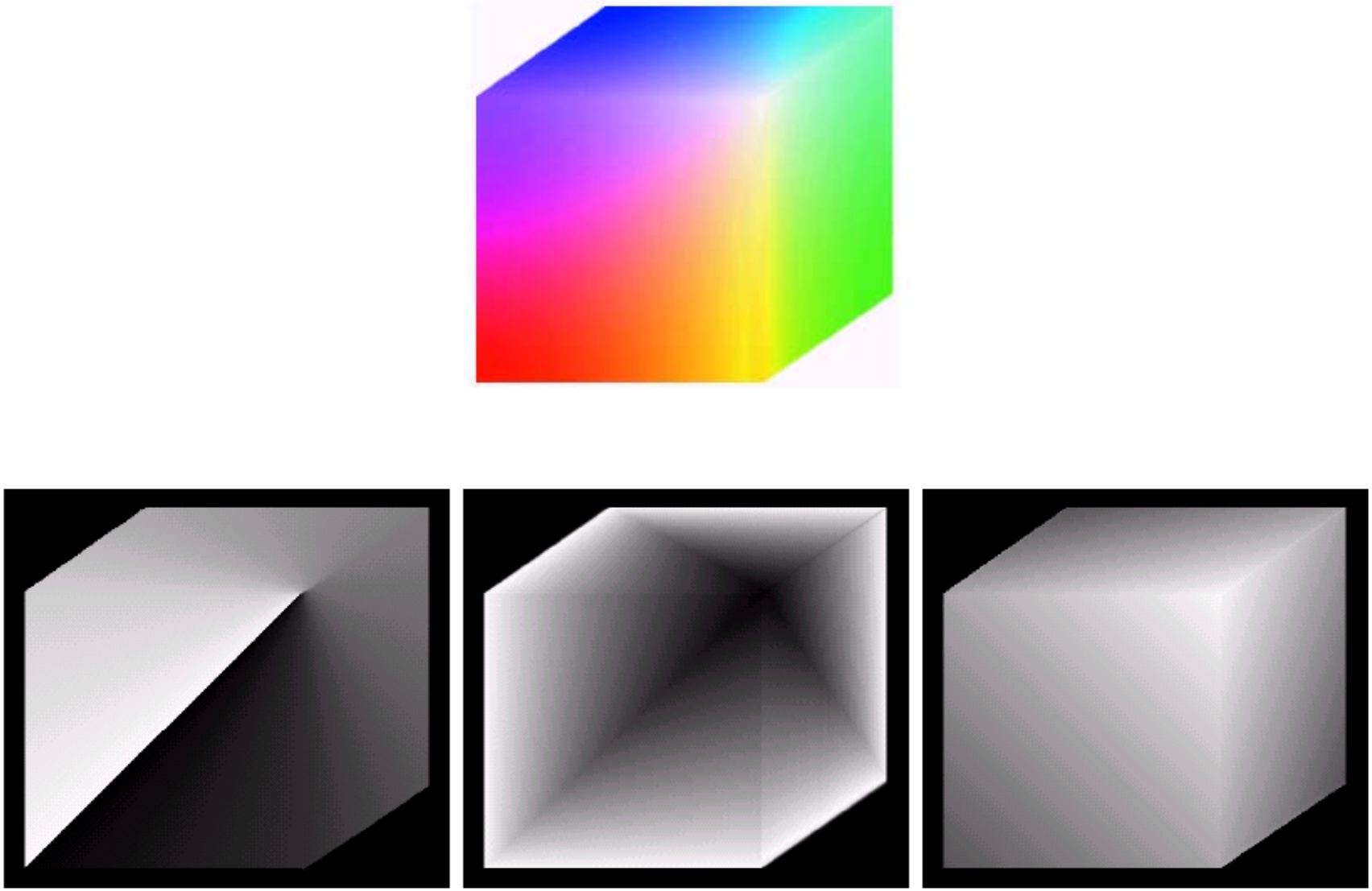
$$S = 1 - \frac{3}{R+G+B}\min(R,G,B)$$

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

R,G,B are 0..1

where

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



a b c

FIGURE 6.15 HSI components of the image in Fig. 6.8. (a) Hue, (b) saturation, and (c) intensity images.

Matlab Example

- rgb2hsv

RGB image

Hue

Saturation

Value

Segmentation

```
% Segment blue
Hmask = (H>0.4) & (H<0.6);  % blue
Smask = (S>0.5);
Vmask = (V>0.3);

figure;
subplot(1,3,1), imshow(Hmask,[]);
subplot(1,3,2), imshow(Smask,[]);
subplot(1,3,3), imshow(Vmask,[]);

% Combine
Result = Hmask & Smask & Vmask;

% Clean up
Result = imopen(Result, strel('disk', 2));
Result = imclose(Result, strel('disk', 2));
figure, imshow(Result);

% Overlay
boundaries = bwboundaries(Result);
figure, imshow(RGB);
hold on
for k=1:length(boundaries)
   b = boundaries{k};
   plot(b(:,2),b(:,1),'g','LineWidth',3);
end
hold off
```

OpenCV example

```
#include <iostream>
#include <opencv2/opencv.hpp>

int main(int argc, char* argv[])
{
   printf("Hit ESC key to quit ...\n");

   cv::VideoCapture cap(0);       // open the default camera
   if(!cap.isOpened())  {         // check if we succeeded
      printf("error - can't open the camera\n");
      system("PAUSE");
      return -1;
   }
   double WIDTH = cap.get(CV_CAP_PROP_FRAME_WIDTH);
   double HEIGHT = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
   printf("Image width=%f, height=%f\n", WIDTH, HEIGHT);

   // Create image windows.  Meaning of flags:
   //CV_WINDOW_NORMAL enables manual resizing; CV_WINDOW_AUTOSIZE is automatic
   //  You can "or" the above choice with CV_WINDOW_KEEPRATIO, which keeps aspect ratio
   cv::namedWindow("Input image", CV_WINDOW_AUTOSIZE);

   // Run an infinite loop until user hits the ESC key
   while (1){
      cv::Mat imgInput;
      cap >> imgInput;       // get image from camera
      cv::imshow("Input image", imgInput);

      // wait for x ms (0 means wait until a keypress)
      if (cv::waitKey(33) == 27)
         break;              // ESC is ascii 27
   }

      return EXIT_SUCCESS;
}
```

- First, capture and display images from a camera

Program 1

Split into bands

- from OpenCV documentation (http://docs.opencv.org/)

> **split**
> Divides a multi-channel array into several single-channel arrays.
>
> **C++: void split(const Mat& src, Mat* mvbegin)**

- At the beginning of the program, add this line

```
char* windowNames[] = { "band 0", "band 1", "band 2" };
```

- After capturing image, add this code:

Program 2

```
// Split into planes
cv::Mat planes[3];
split( imgInput, planes );

// Show images in the windows
for (int i=0; i<3; i++)
   cv::imshow(windowNames[i], planes[i]);
```

Convert to HSV

- from OpenCV documentation

C++: void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)

Parameters:

src – Source image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.

dst – Destination image of the same size and depth as src .

code – Color space conversion code. See the description below.

dstCn – Number of channels in the destination image. If the parameter is 0, the number of the channels is derived automatically from src and code .

- So instead of splitting the BGR image, first convert it to HSV and then split

```
// Convert to HSV
cv::Mat imgHSV;
cv::cvtColor(imgInput, imgHSV, CV_BGR2HSV);

// Split into planes
cv::Mat planes[3];
split( imgHSV, planes );
```

Program 3

Thresholds

- We will threshold each band (H,S,V) using two thresholds tmin, tmax
  - Example: Hmask = (H>tmin) & (H<tmax)
  - We will create trackbars to interactively adjust thresholds
  - There will be two trackbars for each band (one for tmin, the other for tmax)

- Before "main", add these global variables

```
// Trackbar values
int low[] = {50, 50, 50};
int high[] = {250, 250, 250};
```

- Where you create windows, add

```
for (int i=0; i<3; i++)
   cv::namedWindow(windowNames[i], CV_WINDOW_AUTOSIZE);

// Create trackbars
for (int i=0; i<3; i++){
   cv::createTrackbar( "low", windowNames[i], &low[i], 255, NULL );
   cv::createTrackbar( "high", windowNames[i], &high[i], 255, NULL );
}
```

Thresholding

- from OpenCV documentation

C++: double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)

Parameters:

src – Source array (single-channel, 8-bit or 32-bit floating point).
dst – Destination array of the same size and type as src .
thresh – Threshold value.
maxval – Maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types.
type – Thresholding type (see the details below).

- Look at the help page
- We will use both types
  - THRESH_BINARY
  - THRESH_BINARY_INV

Thresholding

- After splitting the HSV image into planes, add this code

```
// Threshold
for (int i=0; i<3; i++){
   cv::Mat imageThreshLow, imageThreshHigh;

   threshold(planes[i],
      imageThreshLow,       // output thresholded image
      low[i],               // value to use for threshold
      255,                  // output value
      cv::THRESH_BINARY);   // threshold_type

   threshold(planes[i],
      imageThreshHigh,         // output thresholded image
      high[i],                 // value to use for threshold
      255,                     // output value
      cv::THRESH_BINARY_INV); // threshold_type

   bitwise_and(imageThreshLow, imageThreshHigh, planes[i]);
}
```

Program 4

Thresholding

- Finally, AND all the masks together and display

```
// Finally, AND all the thresholded images together
cv::Mat imgResult(
   cv::Size(WIDTH,HEIGHT),   // size of image
   CV_8UC1,                  // type: CV_8UC1=8bit, unsigned, 1 channel
   cv::Scalar(255));         // initialize to this value

for (int i=0; i<3; i++)
   bitwise_and(imgResult, planes[i], imgResult);

// Clean up binary image using morphological operators
cv::Mat structuringElmt(7,7,CV_8U,cv::Scalar(1));
morphologyEx(imgResult, imgResult, cv::MORPH_CLOSE, structuringElmt);

cv::imshow("Binary result", imgResult);
```

Program 5

Connected components

C++: void findContours(InputOutputArray image, OutputArrayOfArrays contours, int mode, int method, Point offset=Point())

Parameters:

image – Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use compare() , inRange() , threshold() , adaptiveThreshold() , Canny() , and others to create a binary image out of a grayscale or color one. The function modifies the image while extracting the contours.

contours – Detected contours. Each contour is stored as a vector of points.

mode – Contour retrieval mode (if you use Python see also a note below).

- CV_RETR_EXTERNAL retrieves only the extreme outer contours. It sets hierarchy[i][2]=hierarchy[i][3]=-1 for all the contours.
- CV_RETR_LIST retrieves all of the contours without establishing any hierarchical relationships.
- CV_RETR_CCOMP retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
- CV_RETR_TREE retrieves all of the contours and reconstructs a full hierarchy of nested contours. This full hierarchy is built and shown in the OpenCV contours.c demo.

method – Contour approximation method (if you use Python see also a note below).

- CV_CHAIN_APPROX_NONE stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, max(abs(x1-x2),abs(y2-y1))==1.
- CV_CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
- CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS applies one of the flavors of the Teh-Chin chain approximation algorithm. See [TehChin89] for details.

offset – Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

Connected components

- After creating the binary result image, add this code:

```
// Find connected components (contours)
std::vector<std::vector<cv::Point>> contours;
findContours(
   imgResult,               // input image (is destroyed)
   contours,                // output vector of contours
   CV_RETR_LIST,            // retrieve all contours
   CV_CHAIN_APPROX_NONE);   // all pixels of each contours

// Draw contours on original image
drawContours( imgInput, contours,
   -1,                      // contour number to draw (-1 means draw all)
   cv::Scalar(255,255,255), // color
   2,                       // thickness (-1 means fill)
   8);                      // line connectivity

cv::imshow("Overlay", imgInput);
```

Program 6

Summary / Questions

- Color values can be represented using red, green, blue (RGB) values.
  - An alternative representation is hue, saturation, intensity (HSI).
  - HSV is similar to HSI.

- How would you smooth a color image (e.g., with a Gaussian low pass filter)?