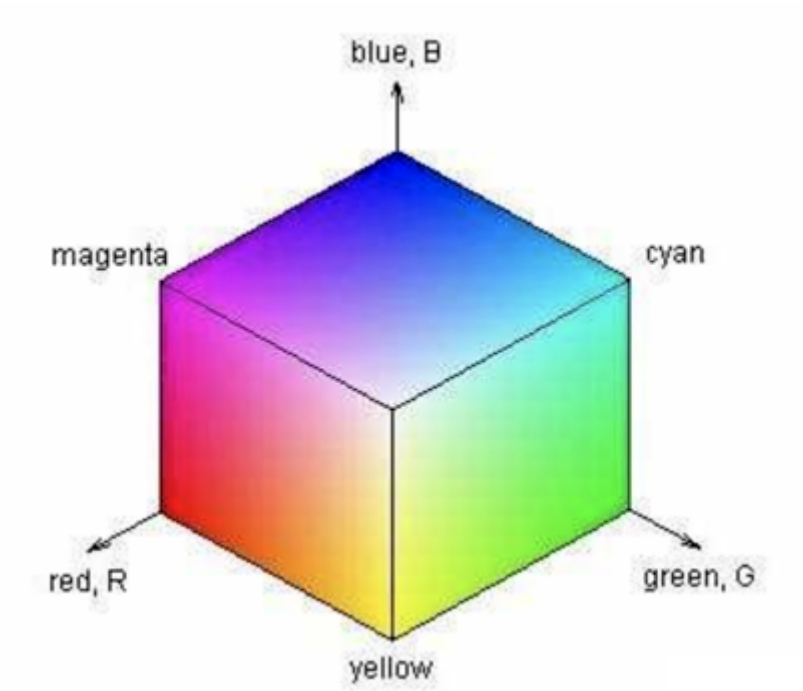
Module 1a: Color Systems & File Types

Three Axis Color

In the RGB color space, colors are often visualized as points within a cube. This cube is a three-dimensional Cartesian coordinate system where each axis represents one of the primary colors: Red, Green, and Blue. The cube provides a way to visualize how the primary colors mix to produce other colors.



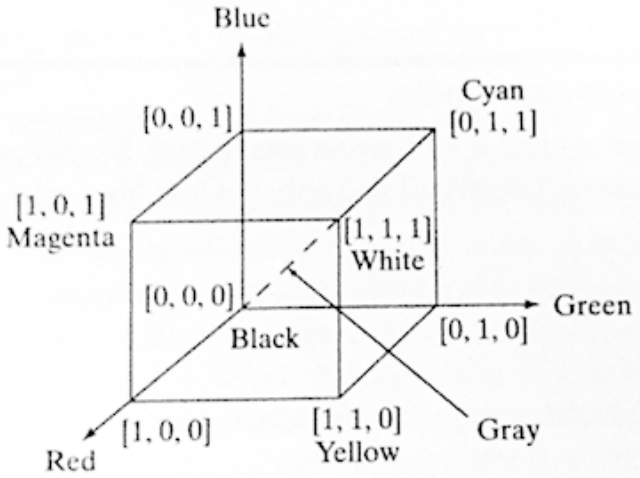


Figure 6.7 Color cube for normalized RGB coordinates:

Structure of the RGB Cube

(note: you can substitute 255 for 1 in the image above)

- **Origin (0,0,0)**: At the origin, all RGB values are zero, which corresponds to the color black.
- **Cube Corners**: The opposite corner from the origin is white, where all RGB values are at their maximum (255, 255, 255 in an 8-bit color scheme).
- **Primary Colors**: The three primary colors are situated along the three axes:
  - The Red axis extends from black (0,0,0) to red (255,0,0).
  - The Green axis extends from black (0,0,0) to green (0,255,0).
  - The Blue axis extends from black (0,0,0) to blue (0,0,255).

Characteristics of the RGB Cube:

1. **Grayscale Line**: If you were to draw a line from the black corner (0,0,0) to the white corner (255,255,255), all points along that line would be shades of gray. This is because, at any point along this line, the values of R, G, and B are equal.
2. **Secondary Colors**: The corners of the cube where only two primary colors mix (ignoring black and white) are the secondary colors. For example, magenta is a mixture of red and blue (255,0,255).
3. **Tertiary Colors and Beyond**: As you move through the cube, the various other colors are combinations of the primary colors in different ratios.
4. **Faces of the Cube**: Each face of the cube represents all possible values of two primary colors while keeping the third one constant. For example, one face of the cube would show all combinations of red and green while keeping blue at 0.
5. **Volume**: Every point inside the cube represents a unique color. The x, y, and z coordinates of that point correspond to the R, G, and B values, respectively.

Other color spaces exist such as HSV (HSB) & YCrCb.

HSV, HSB, HSI

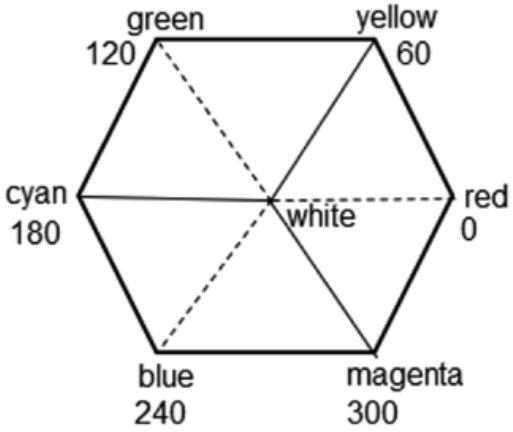
HSV stands for Hue, Saturation, and Value. You may see it called HSB or HSI (brightness and intensity). You can visualize the HSV color space in a cylinder or more precisely in a hexacone.

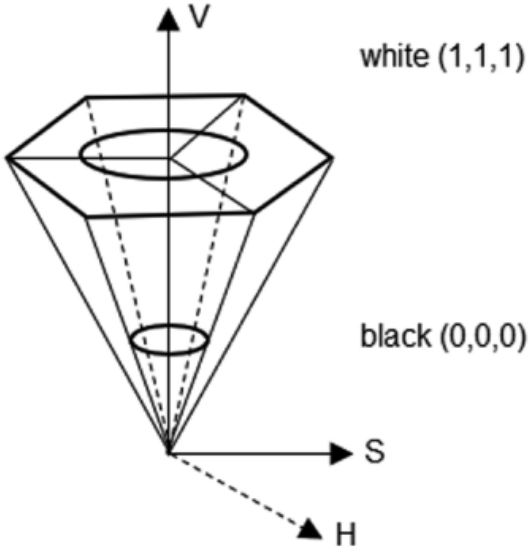
HSV encodes color information by seperating out the brightness (value or intensity) value from the two values encoding chromaticity, Hue and Saturation.

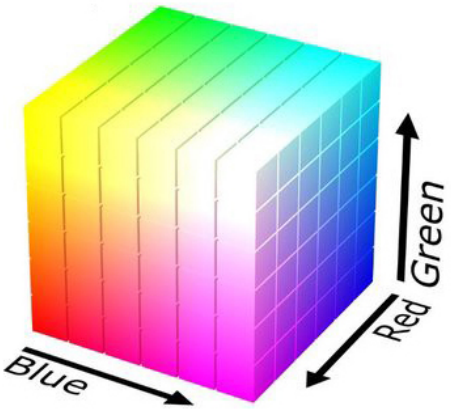
Hue: Represents the color.
Saturation: Represents the purity of the hue or color. The amount of grey mixed in. Value: The amount of of black or white mixed with a given hue

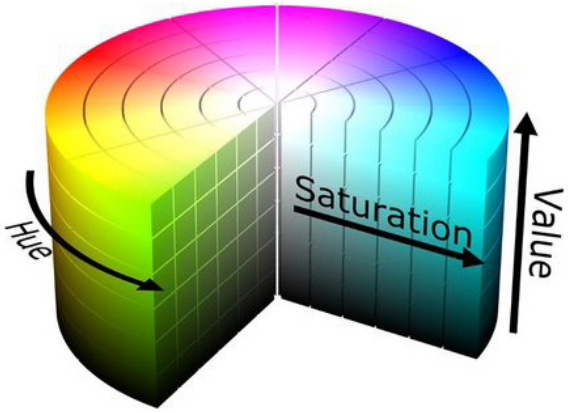
The hue is defined by the angle between 0 and 360° or 2π. π(radians) being equal to 180° and 2π being equal to 360°. π ≈ 3.14 | 2π ≈ 6.28

- Pure red is an angle of 0°
- Pure green is at 120° or 2π/3.
- Pure blue is at 240° or 4π/3.









Color Spaces & Theory

RGB -> HSV

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-min(R,G,B)) & \text{if } V = R \\ 120+60(B-R)/(V-min(R,G,B)) & \text{if } V = G \\ 240+60(R-G)/(V-min(R,G,B)) & \text{if } V = B \\ 0 & \text{if } R = G = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

The values are then converted to the destination data type:

- 8-bit images: $V \leftarrow 255V, S \leftarrow 255S, H \leftarrow H/2$(to fit to 0 to 255)
- 16-bit images: (currently not supported) $V < -65535V, S < -65535S, H < -H$
- 32-bit images: H, S, and V are left as is

Open CV RGB -> HSB

In [ ]:

```
import cv2
import numpy as np
from numpy import interp

def rgb_to_hsv(rgb):
    # Input: rgb is an 1-D array [r,g,b] with values in range [0,255].
    # r = rgb[0], b = rgb[1], g = rgb[2]
    # Output: hsv is an 1-D array [h,s,v] with values in range h = [0,360], s =

    # Normalize color values.  Convert to floating point values between 0 - 1
    rgb = rgb/255

    # Initialize HSV
    h = 0.0
    s = 0.0
    v = 0.0

    # Find the max and min RGB values.
    v = np.max(rgb)
    vMin = np.min(rgb)

    # Set the saturation value.
    if(v>0.0):
        s = (v - vMin)/v
    else:
        s = 0.0

    # Calculate (v - vMin) convenience
    diff = (v - vMin)

    # Compute the hue by the relative sizes of the RGB components

    # Are r,g,b equal.
    if(rgb[0] == rgb[1] and rgb[1] == rgb[2]):
        h = 0
    # Is the point within +/- 60 degrees of the red axis
    elif(rgb[0] == v):
        h = 60 * (rgb[1] - rgb[2]) / diff
```

```
    # Is the point within +/- 60 degrees of the green axis
    elif(rgb[1] == v):
        h = 120 + 60 * (rgb[2] - rgb[0]) / diff
    # IS the point within +/- 60 degrees of the blue axis
    elif(rgb[2] == v):
        h = 240 + 60 * (rgb[0] - rgb[1]) / diff


    # Return hsv values.
    return np.array([h,s,v])

# Create a rgb value.
rgb = np.array([200,74,55])

# Call the rgb_to_hsv function.
hsv = rgb_to_hsv(rgb)

print("The original rgb value:", rgb)
print("Converted to hsv:", hsv)
```

```
The original rgb value: [200  74  55]
Converted to hsv: [7.86206897 0.725      0.78431373]
```

In [ ]:

```
rgb_cv = np.uint8([[[200,74,55]]])
hsv_cv = cv2.cvtColor(rgb_cv,cv2.COLOR_RGB2HSV)
print("OpenCv hsv:", hsv_cv)
```

```
OpenCv hsv: [[[  4 185 200]]]
```

OpenCV halves the H values to fit the range [0,255], so H value instead of being in range [0, 360], is in range [0, 180]. S and V are also in range [0, 255] instead of [0,1]. Understanding value ranges is very important. Depending on the image file, bit depth, software specifications, along with other factors; the value range for storing pixel data can change.

Let's convert our values to match OpenCv.

In [ ]:

```
# Convert to OpenCv ranges

s = np.interp(.725,[0,1],[0,255])
v = np.interp(.78431,[0,1],[0,255])
print("HSV converted to OpenCv's range:", [[[round(hsv[0]/2),round(s),round(v)]]]
```

```
HSV converted to OpenCv's range: [[[4, 185, 200]]]
```

YCrCb, YCbCr, ...

Is primary a broadcast color space. It's main advantage is in compression and subsampling. Will be discussed more in Image Compression Module 3a

- The Y stands for the Luma value (Brightness) or the image.
- Cr stands for the change in the red channel.
- Cb stands for the change in the blue channel.

YCrCb wiki

Common File Types

Raster (Pixel Array)

The raster file formats all store image data in an array structure. Different formats allow for certain types of compression, color spaces, and bit depth. Separating formats into lossy and lossless.

- **TIFF** Tag Image File Format. (1986) Orginated by the Aldus Corp and later bought by Adobe. Tiff or Tif is very general and complex format. Cross platform compatible. Can store both lossy (Jpeg) and lossless image data. TIFF Wiki

- **JPEG** (JPG) Joint Photographic Experts Group. (1992) Jpeg's main advantage is high-quality practical compression. JPEG Wiki

- **GIF** The General Graphics Interchange Format. (1987) Color must be represented in 8-bits total for all color channels so limited to 256 colors. 16-bit option also available but the 8-bit is most popular. The limited file size and ability to store multiple images in one file made the format popular in the early days of the internet to today. GIF Wiki

- **PNG** Portable Network Graphics. (1996) PNG Working Group and donated to W3. Designed as a replacement for the GIF. Lossless compression RGB 24-bit color.PNG Wiki

Vector

The vector files formats are built using coded lines and angles to draw images, type, graphics.

- **Postscript** Page Description Language. (1982) Invented by Adobe. PDF, EPS. Dominates the graphics and printing industry. Postscript Wiki

- **SVG** Scalable Vector Graphics. (1999) World Wide Web Consortium (W3C) Vector graphics stored in XML files which allows easy searching. Allows for multiple files and animation. SVG Wiki

---

Project 1 - part 1

(1) Convert the RGB to HSV code so it produces values in OpenCv's specified ranges and array structure. Your function should accept a 3D array and return HSV values in OpenCv's ranges. H -> [0,180], S -> [0,255], V -> [0,255]

To test your function use the following rgb value: `rgb = np.uint8([[[200,74,55]]])`
Your function should return: [[[4, 185, 200]]]

(2) Write the Python code for the conversion of HSV color space to RGB. The algorithm is below. The code is available online but please write it on your own. Include comments to

describe each step in your code.

HSV to RGB conversion formula

When $0 \le H < 360$, $0 \le S \le 1$ and $0 \le V \le 1$:

$$C = V \times S$$

$$X = C \times (1 - |(H / 60^\circ) \bmod 2 - 1|)$$

$$m = V - C$$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0^\circ \le H < 60^\circ \\ (X, C, 0) & , 60^\circ \le H < 120^\circ \\ (0, C, X) & , 120^\circ \le H < 180^\circ \\ (0, X, C) & , 180^\circ \le H < 240^\circ \\ (X, 0, C) & , 240^\circ \le H < 300^\circ \\ (C, 0, X) & , 300^\circ \le H < 360^\circ \end{cases}$$

$$(R,G,B) = ((R'+m)\times 255, (G'+m)\times 255, (B'+m)\times 255)$$