2. Color spaces

2.1 Introduction

The purpose of the second laboratory work is to teach the basic color manipulation techniques, applied to the bitmap digital images.

2.2 The RGB color space

The color of each pixel, either in image acquisition devices such as cameras, and in image displaying devices such as the computer monitor and the TV screen, is obtained by combining three primary colors: Red, Green and Blue (additive color space – Fig. 2.1 and 2.2).

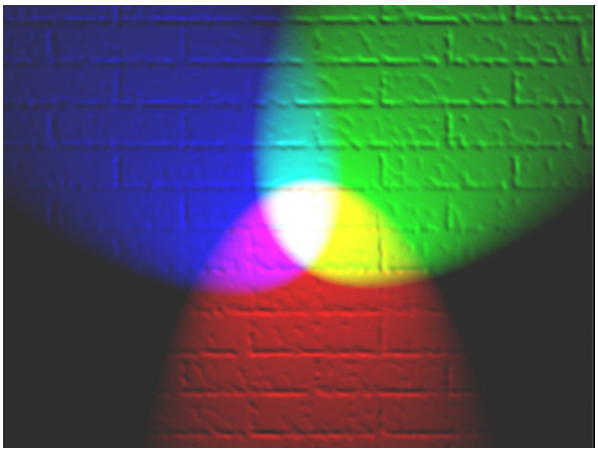


Fig. 2.1. Additive mixing of colors. When the primary colors are superposed, the secondary colors appear. When all three primary colors are superposed, the white color is obtained [1].

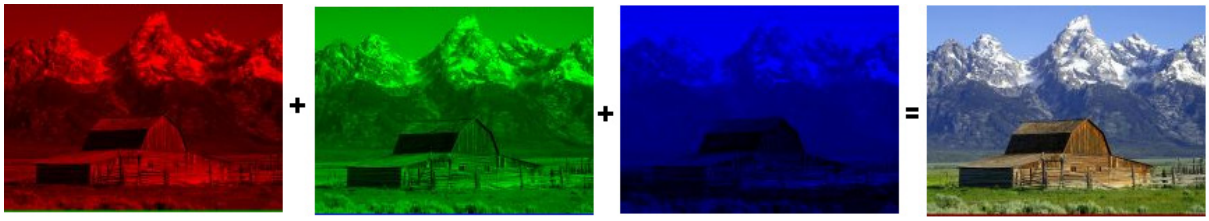


Fig. 2.2. The color image is obtained by pixel level combination of the primary colors. The three color channels are displayed.

Each image pixel will be defined by a triplet, containing a numerical value for each primary color. The color can be regarded as a point in a 3D RGB color space (Fig. 2.3). The origin of the coordinate axes corresponds to the color Black (0, 0, 0), and the opposite corner of the color space cube corresponds to the color White (255, 255, 255). The cube's diagonal, between black and white, corresponds to levels of gray (grayscale), defined by (R=G=B). Three of the corners correspond to the primary colors Red, Green and Blue. The other corners correspond to the complementary colors of Cyan, Magenta and Yellow. If the origin of the color space is moved to the White point, and the axes of the system are renamed as C, M and Y, one gets the complementary CMY color space, which is used in color printing devices.

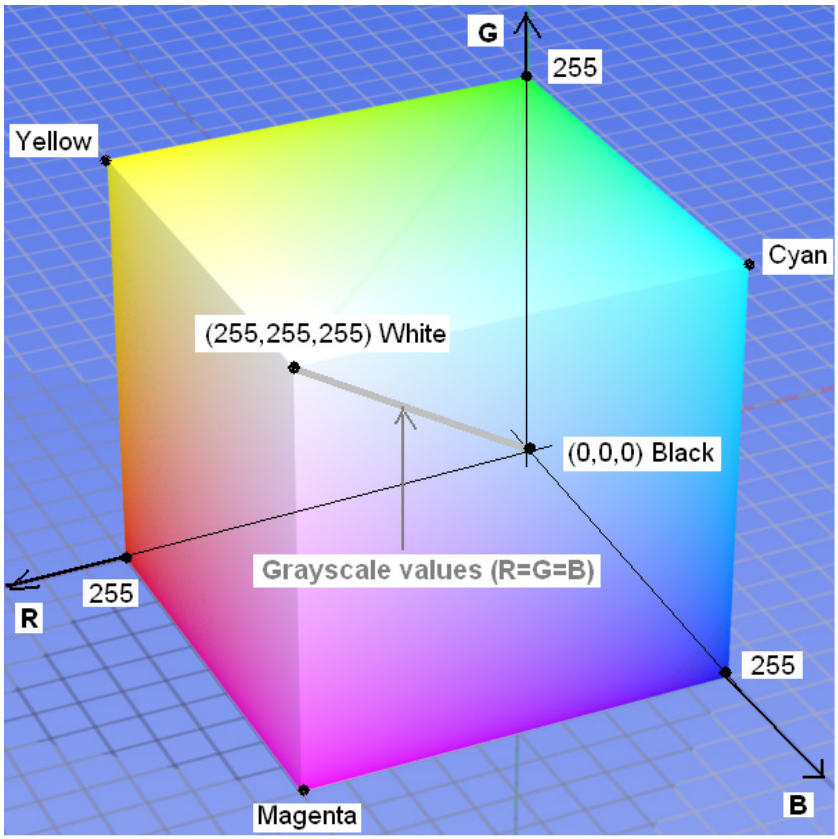


Fig. 2.3. The RGB color space mapped on a cube. Here, each color axis is represented on 8 bits (256 levels) (RGB24 bitmap images). The total number of colors is $2^8 x 2^8 x 2^8 = 2^{24} = 16.777.216$.

For RGB24 images, all possible color combinations can be displayed simultaneously. If the image contains a palette, and the color of a pixel is an index in the palette, only a subset of the colors can be displayed. In this context, the number of bits/pixel (the number of bits used to encode a color) is called "color depth" (Table 2.1):

Table 2.1. Color depth and image type

Color depth	Number of colors	Color mode	Palette (LUT)
1 bit	2	Indexed Color	Yes
4 bits	16	Indexed Color	Yes
8 bits	256	Indexed Color	Yes
16 bits	65536	True Color	No
24 bits	16.777.216	True Color	No
32 bits	16.777.216	True Color	No

There are other color models [2], which will not be discussed here.

2.3 Conversion of a color image to grayscale

In order to convert a color pixel to a grayscale pixel, its color components must be made equal. A widely used conversion method is to compute the intensity as the average of the three channels:

$$R_{Dst} = G_{Dst} = B_{Dst} = \frac{R_{Src} + G_{Src} + B_{Src}}{3} \quad (2.1)$$

2.4 Conversion of a grayscale image to binary (black and white)

A binary image, having only two pixel values (black and white) is obtained from a grayscale image through an operation called thresholding. This operation involves the comparison of the graylevel pixels with a value called "threshold". Thresholding is the simplest segmentation technique, which allows the separation of foreground objects from the background (Fig. 2.4).

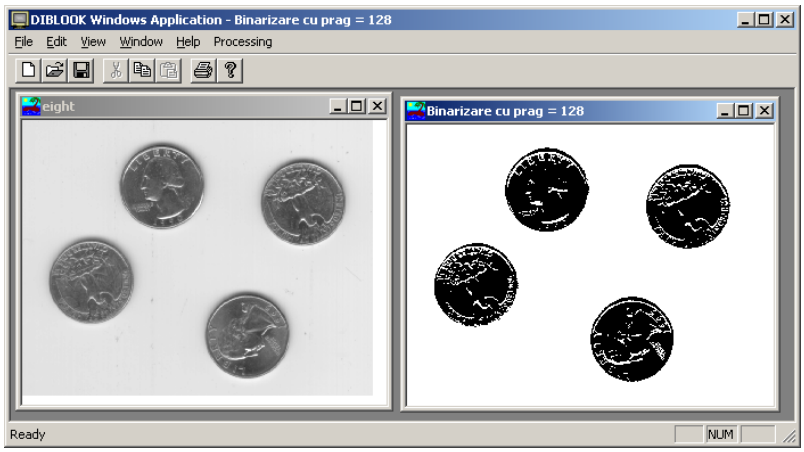


Fig. 2.4. Thresholding.

In this laboratory work you will implement the thresholding operation using a fixed, user defined threshold, for grayscale 8 bit images. The pixels from the source image will be compared to the threshold value, and the destination will be set to:

$$Dst(i,j) = \begin{cases} 0 \quad (black) & , \quad if \quad Src(i,j) < threshold \\ 255 \quad (white) & , \quad if \quad Src(i,j) \geq threshold \end{cases} \tag{2.2}$$

2.5 The HSV (Hue Saturation Value) color space

This color space tries to mimic the way the humans perceive color. The H component (hue) is the color itself, independent (invariant) of illumination, the S component (saturation) is the color's "purity" (how well defined the color is), and V (value, or intensity) is the brightness. This space is represented as a pyramid with a hexagonal base, or as a cone.

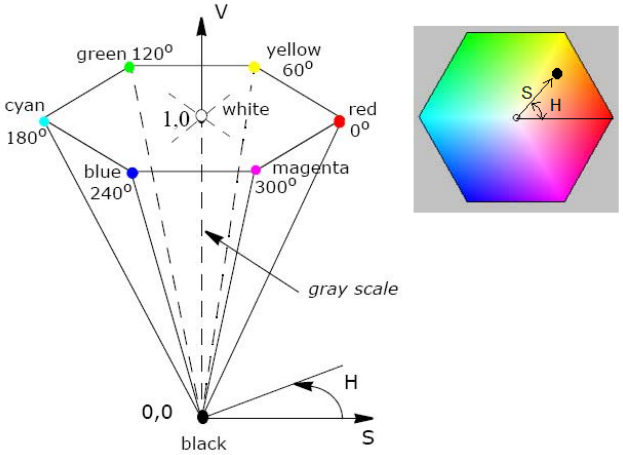


Fig. 2.5. The HSV color space.

Using the pyramid representation, the significance of the components is:

H – the angle between the current color and the ray corresponding to the color Red.
S – the distance from the current color to the central axis of the pyramid/code.
V – the height of the current color in the pyramid/cone.

2.6 The RGB → HSV transform

The equations for obtaining the HSV components from RGB are [3]:

```
r = R/255;  // r : the normalized R component
g = G/255; // g : the normalized G component
b = B/255; // b : the normalized B component
// Attention: please declare all variables as float
// If you have declared R as uchar, you have to use a cast: r = (float)R/255 !!!

M = max (r, g, b); //Attention: there is a default macro in Visual C for max and min, but
m = min (r, g, b);  //it only takes two parameters (no compiler error if you provide three)
C = M - m;
```

Value:

```
V = M;
```

Saturation:

```
If (V!=0)
    S = C / V;
Else // black
        S = 0;
```

Hue:

```
If (C!=0) {
    if (M == r) H = 60 * (g - b) / C;
    if (M == g) H = 120 + 60 * (b - r) / C;
    if (M == b) H = 240 + 60 * (r - g) / C;
    }
Else  // grayscale
    H = 0;
If (H < 0)
        H = H + 360;
```

The values for H, S and V computed with the previous equations will have the following range:

H = 0 .. 360
S = 0 .. 1
V = 0 .. 1

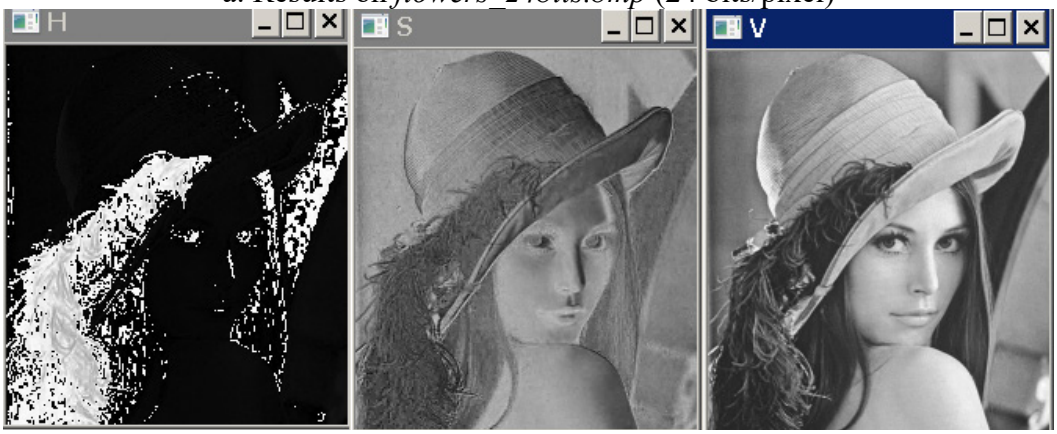
In order to display them as 8-bit grayscale images, you will need to scale them to the 0…255 interval:

H_norm = H*255/360
S_norm = S*255
V_norm=V*255

2.7 Practical work

1. Create a function that will copy the R, G and B channels of a color, RGB24 image (CV_8UC3 type) into three matrices of type CV_8UC1 (grayscale images). Display these matrices in three distinct windows.
2. Create a function that will convert a color RGB24 image (CV_8UC3 type) to a grayscale image (CV_8UC1), and display the result image in a destination window.
3. Create a function for converting from grayscale to black and white (binary), using (2.2). Read the threshold from the console. Test the operation on multiple images, and using multiple thresholds.
4. Create a function that will compute the H, S and V values from the R, G, B channels of an image, using the equations from 2.6. Store each value (H, S, V) in a CV_8UC1 matrix. Display these matrices in distinct windows. Check the correctness of your implementation using the example below.
5. Implement a function called *isInside(img, i, j)* which checks if the position indicated by the pair *(i,j)* (row, column) is inside the image *img*.
6. **Save your work. Use the same application in the next laboratories. At the end of the image processing laboratory you should present your own application with the implemented algorithms.**

a. Results on *flowers_24bits.bmp* (24 bits/pixel)



b. Results on *Lena_24bits.bmp* (24 bits/pixel)

Fig. 2.6. Examples of RGB to HSV conversion.

2.8 References

[1] http://en.wikipedia.org/wiki/RGB_color_model
[2] http://en.wikipedia.org/wiki/Color_models
[3] Open Computer vision Library, Reference guide, cvtColor() function, https://docs.opencv.org/4.5.1/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab