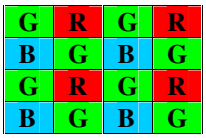
Bayer color conversion and processing

This note describes conversions from Bayer format to RGB and between RGB and YUV (YCrCb) color spaces. We also discuss two color processing operations (white balance and color correction) in the RGB domain, and derive the corresponding operations in the YUV domain. Using derived operations in the YUV domain, one can perform white balance and color correction directly in the YUV domain, without switching back to the RGB domain.

1. Conversion from Bayer format to RGB

Bayer color filter array is a popular format for digital acquisition of color images [1]. The pattern of the color filters is shown below. Half of the total number of pixels are green (G), while a quarter of the total number is assigned to both red (R) and blue (B).



To convert an image from this format to an RGB format, we need to interpolate the two missing color values in each pixel. Several standard interpolation methods (nearest neighbor, linear, cubic, cubic spline, etc.) were evaluated on this problem in [2]. The authors have measured interpolation accuracy as well as the speed of the method and concluded that the best performance is achieved by a correlation-adjusted version of the linear interpolation. The suggested method is presented here.

1.1 Interpolating red and blue components

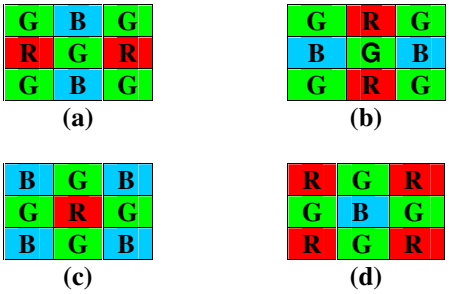


Figure 1: Four possible cases for interpolating R and B components

As suggested in [2], R and B values are interpolated linearly from the nearest neighbors of the same color. There are four are possible cases, as shown in Figure 1. When interpolating the missing values of R and B on a green pixel, as in Figure 1 (a) and (b), we take the average values of the two nearest neighbors of the same color. For example, in Figure 1 (a), the value for the blue component on a shaded G pixel will be the average of the blue pixels above and below the G pixel, while the value for the red component will be the average of the two red pixels to the left and right of the G pixel.
Figure 1 (c) shows the case when the value of the blue component is to be interpolated for an R pixel. In such case, we take the average of the four nearest blue pixels cornering the R pixel. Similarly, to determine the value of the red component on a B pixel in Figure 2 (d) we take the average of the four nearest red pixels cornering the B pixel.

1.2 Interpolating the green component

By [2], green component is adaptively interpolated from a pair of nearest neighbors. To illustrate the procedure, consider two possible cases in Figure 2.

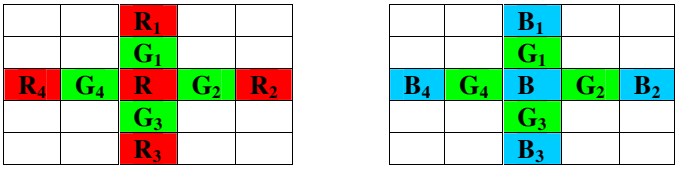


Figure 2: Two possible cases for interpolating G component

In Figure 2 (a), the value of the green component is to be interpolated on an R pixel. The value used for the G component here is

$$G(R) = \begin{cases} (G_1 + G_3)/2, & \text{if } |R_1 - R_3| < |R_2 - R_4| \\ (G_2 + G_4)/2, & \text{if } |R_1 - R_3| > |R_2 - R_4| \\ (G_1 + G_2 + G_3 + G_4)/4, & \text{if } |R_1 - R_3| = |R_2 - R_4| \end{cases}$$

In other words, we take into account the correlation in the red component to adapt the interpolation method. If the difference between R_1 and R_3 is smaller than the difference between R_2 and R_4, indicating that the correlation is stronger in the vertical direction, we use the average of the vertical neighbors G_1 and G_3 to interpolate the required value. If the horizontal correlation is larger, we use horizontal neighbors. If neither direction dominates the correlation, we use all four neighbors.

Similarly, for Figure 2 (b) we will have

$$G(B) = \begin{cases} (G_1 + G_3)/2, & \text{if } |B_1 - B_3| < |B_2 - B_4| \\ (G_2 + G_4)/2, & \text{if } |B_1 - B_3| > |B_2 - B_4| \\ (G_1 + G_2 + G_3 + G_4)/4, & \text{if } |B_1 - B_3| = |B_2 - B_4| \end{cases}$$

To conclude this section, note that if the speed of execution is the issue, one can safely use simple linear interpolation of the green component from the four nearest neighbors, without any adaptation

$$G = (G_1 + G_2 + G_3 + G_4)/4$$

According to [2], this method of interpolation executes twice as fast as the adaptive method, and achieves only slightly worse performance on real images, while it is actually better than the adaptive method when applied to synthetic images.

2. Conversion between RGB and YUV

We give two commonly used forms of equations for conversion between RGB and YUV formats. The first one is recommended by CCIR [3]

$$\begin{aligned} Y &= 0.257R + 0.504G + 0.098B + 16 \\ U &= 0.439R - 0.368G - 0.071B + 128 \\ V &= -0.148R - 0.291G + 0.439B + 128 \end{aligned} \qquad (2.1)$$

The second form is used by Intel in their image processing library [4], and may be more suitable for implementation:

$$\begin{aligned} Y &= (9798R + 19235G + 3736B)/2^{15} \\ U &= (21208R - 16941G - 3277B)/2^{15} + 128 \\ V &= (-4784R - 9437G + 4221B)/2^{15} + 128 \end{aligned} \quad (2.2)$$

In either case, resulting values of Y, U and V should be clipped to fit the appropriate range for the YUV format (e.g. [0,255] for a 24-bit YUV format). The inverse conversion may be accomplished by:

$$\begin{aligned} R &= 1.164(Y - 16) + 2.018(V - 128) \\ G &= 1.164(Y - 16) - 0.813(U - 128) - 0.391(V - 128) \\ B &= 1.164(Y - 16) + 1.596(U - 128) \end{aligned} \quad (2.3)$$

3. White balance operation in RGB and YUV domains

The white balance operation is defined as a gain correction for red, green and blue components by gain factors A_R, A_G and A_B, respectively, i.e.

$$\begin{aligned} R_{wb} &= A_R R \\ G_{wb} &= A_G G \\ B_{wb} &= A_B B \end{aligned} \quad (3.1)$$

The new (white-balanced) values for red, green and blue are R_{wb}, G_{wb} and B_{wb}. To derive the equivalent form of this operation in the YUV domain, we proceed as follows. First, write equation (2.1) as

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{v} \quad (3.2)$$

where $\mathbf{x} = (R, G, B)^T$ is the vector in the RGB space, $\mathbf{y} = (Y, U, V)^T$ is the corresponding vector in the YUV space, $\mathbf{v} = (16,128,128)^T$, and $\mathbf{C}$ is the appropriate matrix of conversion coefficients. Similarly, (3.1) can be written as

$$\mathbf{x}_{wb} = \mathbf{A}\mathbf{x} \quad (3.3)$$

where $\mathbf{x}_{wb} = (R_{wb}, G_{wb}, B_{wb})^T$ is the vector in the RGB space modified by white balance operation (2.4), and $\mathbf{A} = diag(A_R, A_G, A_B)$. We want to determine what is the corresponding vector $\mathbf{y}_{wb}$ in the YUV domain, without having to revert back to the RGB domain. Vector $\mathbf{y}_{wb}$ is found by substituting $\mathbf{x}_{wb}$ for $\mathbf{x}$ in (3.2)

$$\mathbf{y}_{wb} = \mathbf{C}\mathbf{x}_{wb} + \mathbf{v} = \mathbf{C}\mathbf{A}\mathbf{x} + \mathbf{v}.$$

Let $\mathbf{D} = \mathbf{C}\mathbf{A}$, so that $\mathbf{y}_{wb} - \mathbf{v} = \mathbf{D}\mathbf{x}$. Then $\mathbf{x} = \mathbf{D}^{-1}(\mathbf{y}_{wb} - \mathbf{v})$. Substitute this expression for $\mathbf{x}$ back into (3.2) to obtain

$$\mathbf{y} = \mathbf{C}\mathbf{D}^{-1}(\mathbf{y}_{wb} - \mathbf{v}) + \mathbf{v} \quad (3.4)$$

This equation provides the connection between $\mathbf{y}$ and $\mathbf{y}_{wb}$ without involving $\mathbf{x}$ or $\mathbf{x}_{wb}$ (i.e. without going back to the RGB domain). Manipulating (3.4) and using the fact that for nonsingular matrices $\left(\mathbf{CD}^{-1}\right)^{-1} = \mathbf{DC}^{-1}$ [5], we get that white balance operation in the YUV domain is

$$\mathbf{y}_{wb} = \mathbf{DC}^{-1}\left(\mathbf{y} - \mathbf{v}\right) + \mathbf{v} = \mathbf{CAC}^{-1}\left(\mathbf{y} - \mathbf{v}\right) + \mathbf{v} \qquad (3.5)$$

Expressing components of $\mathbf{y}_{wb}$ from (3.5) we get

$$Y_{wb} \approx (0.299A_R + 0.587A_G + 0.114A_B)(Y - 16) + (0.410A_R - 0.410A_G)(U - 128) + (-0.197A_G + 0.198A_B)(V - 128) + 16$$

$$U_{wb} \approx (0.511A_R - 0.428A_G - 0.008A_B)(Y - 16) + (0.701A_R + 0.299A_G)(U - 128) + (0.144A_G - 0.143A_B)(V - 128) + 128$$

$$V_{wb} \approx (-0.172A_R - 0.339A_G + 0.511A_B)(Y - 16) + (-0.236A_R + 0.237A_G)(U - 128) + (0.114A_G + 0.886A_B)(V - 128) + 128$$

Terms with leading coefficient less than 10^{-3} have been dropped.

References

[1] B. E. Bayer, *Color imaging array*, US Patent No. 3971065.

[2] T. Sakamoto, C. Nakanishi and T. Hase, "Software pixel interpolation for digital still cameras suitable for a 32-bit MCU," *IEEE Trans. Consumer Electronics*, vol. 44, no. 4, November 1998.

{3} http://www.northpoleengineering.com/rgb2yuv.htm