Understanding ISP Pipeline - CCM

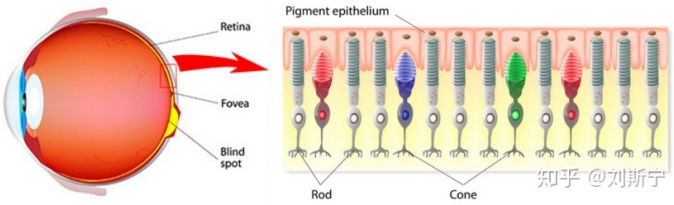
刘斯宁
Camera技术专家

已关注

34 人赞同了该文章

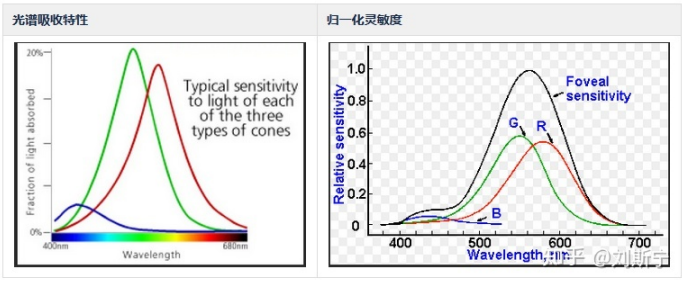
人眼的特性

人眼视网膜的微观结构如下图所示。



从图中可以看到，人眼视网膜上有三种锥细胞(cone)用于感受蓝、绿、红三种频率的色光，一种杆细胞(rod)只在低照度条件下感应亮度信息，不能分辨颜色和细节。

已知三种锥细胞的总个数大约600~800万，分布比例大约是1:16:32，杆细胞的总数大约是1200万。人眼的三种锥细胞对不同波长的电磁波的响应是有明显区别的，研究发现，人眼的红、绿、蓝三种锥细胞光谱响应曲线的峰值比例是0.54:0.575:0.053，如下图所示。



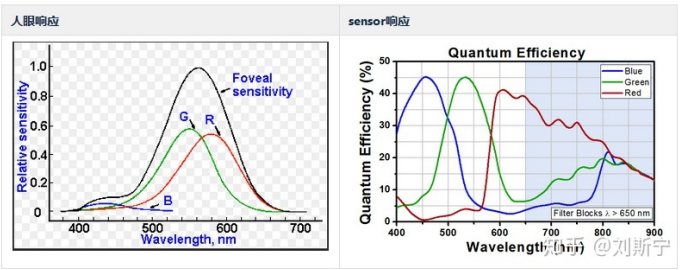
如从图中曲线可以看到，人眼对波长为555纳米的绿色光吸收率可达20%，对波长为650纳米的橙色光的吸收率在4%左右，对波长为450纳米的蓝色光的吸收率最高只有2%左右，大约是绿色光最高吸收率的1/10。

因此，对于辐射强度相同而颜色（波长）不同的光，人眼的亮度感觉是不同的：波长为555纳米的黄绿光感觉最亮，波长为450纳米的蓝色光感觉最暗。另外还发现，当光线微弱时，人眼最敏感的波长会移动至507nm。

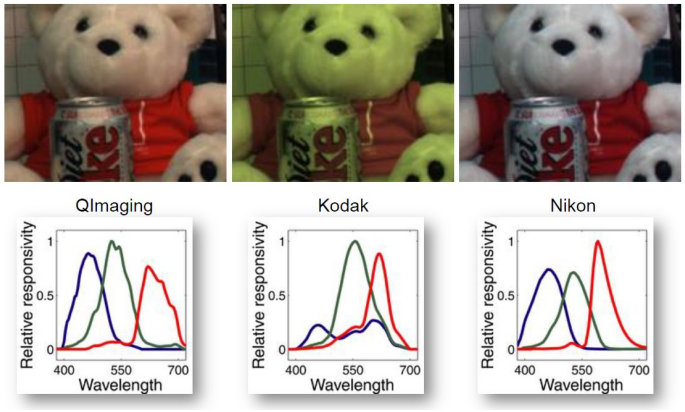
人眼对不同波长的光有不同的色调感觉，严格地讲,只有波长为572nm的黄光、503nm的绿光和478nm的蓝光,其色调不随光强而变化,其它波长的色光都随光强的改变略有变化。在可见光谱中,从紫到红分布着各种不同的颜色,人眼能分辨出色调差别的最小波长变化称为色调分辨阈.其数值随波长而改变.人眼对480~640nm区间色光的色调分辨力较高,其中,对500nm (青绿色)和600nm(橙黄色）两个波长来说,只要波长变化约1nm,便可分辨出色调的变化。而从655nm的红色到可见光谱长波末端,以及从430nm的紫色到可见光谱短波末端，人眼几乎感觉不到色调的差别。当饱和度减小时,人眼的色调分辨力将下降;当亮度太大或太小时,色调分辨力也会下降。

颜色校正 (Correction Correction)

人眼在可见光波段的频谱响应度和半导体传感器的频谱响应以及显示器的激励响应都存在较大的差别，这些差别会对摄像机的色彩还原造成较大的影响。举例来说，从下图所示的光谱响应曲线中可以了解到，一个典型的硅材料sensor在500nm处对蓝、绿光的响应几乎是相等的，但是人眼的蓝色锥细胞对500nm的蓝绿光响应却几乎为零。假设sensor按照自己的特性忠实地记录下它对500nm波长的响应值为b，根据CIE的标准，显示器会在b值的驱动下发出波长为435.8nm的蓝光，使人眼感知到明亮的蓝光。这个过程在人眼看来，就是成像系统在绿色中凭空增加了很多蓝色的成分，降低了绿色的饱和度，造成了颜色失真。

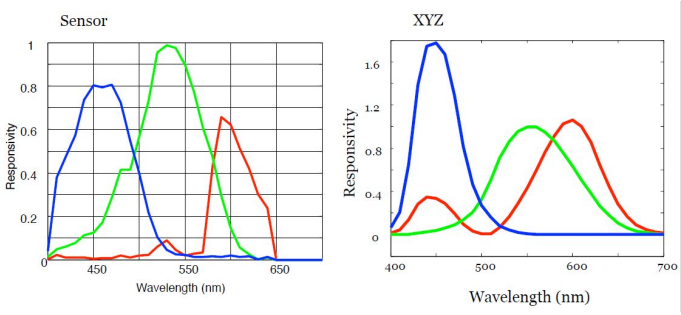


不仅人眼的响应与sensor不同，不同厂家制造的sensor响应也是不同的。下图显示了三个camera在相同参数下拍摄到的颜色效果。



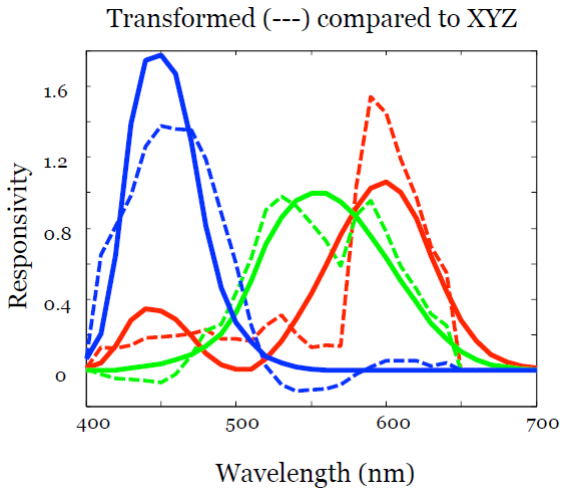
此外，摄像机光路上一般还存在镜头、滤波片等光学元件，镜头的镀膜、滤光片的频率响应等参数也会对色彩还原造成影响，这些因素综合作用的结果，就是人眼在显示器上看到的RGB颜色与真实世界中感知到的物体颜色存在偏差， 尤其是色饱和度受到较大影响，因此必须对摄像机记录的颜色进行校正以还原人眼的感知效果。

颜色校正，英文color correction，是在RGB空间中完成的处理任务，主流的做法是用一个3x3 的矩阵将一个输入像素值(R, G, B)线性地映射为一个新的像素值(R', G', B')，通过审慎地选择矩阵参数使映射后的颜色更符合人的认知习惯。



这个3x3 的矩阵叫做颜色校正矩阵，英文color correction matrix，简称CCM，其变换公式如下。

CCM公式的物理意义是从一种颜色种减除另外两种颜色的成分，以增加该颜色的饱和度，使变换的结果接近人的视觉感受，或者更符合人的主观审美。由于输入颜色可以有上千万种组合，而CCM参数却只有9个，所以CCM实际上只能优先保证几个最重要的颜色在人看来是“正确”的，而不可能面面俱到地保证所有颜色在所有条件下都是最优的。下图显示了变换值（虚线）与理想值（实线）之间的差异。



CCM公式的一个基本约束就是不能破坏白平衡，即对于任何R=G=B的输入，必须保证输出满足R'=G'=B'。正式由于这个原因，颜色校正操作只能放在白平衡调整之后执行。

实践中通常使用X-Rite 24色卡上的18个彩色色块为标准计算校正系数，基本原理是用摄像机拍摄色卡，提取18个色块的平均颜色（Rn, Gn, Bn），n=1..18 构成输入矩阵

用24色卡上的18个彩色色块的标准RGB值构成目标矩阵

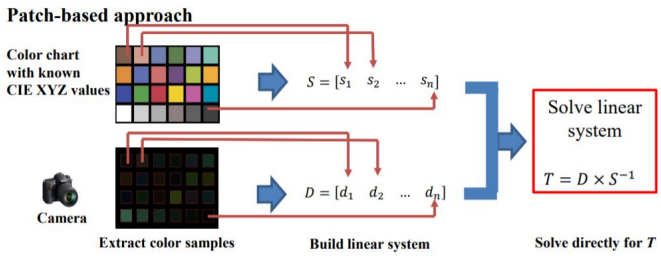
则有关于CCM的矩阵方程

$$S' = M \cdot S$$

$$\begin{bmatrix} R'_1 & R'_2 & \cdots & R'_n \\ G'_1 & G'_2 & \cdots & G'_n \\ B'_1 & B'_2 & \cdots & B'_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} R_1 & R_2 & \cdots & R_n \\ G_1 & G_2 & \cdots & G_n \\ B_1 & B_2 & \cdots & B_n \end{bmatrix}$$

求解M的过程如下

上述过程需要使用某种色卡，因此称为patch-based方法。



--

-- This VHDL file was generated by EASE/HDL 7.2 Revision 8 from HDL Works B.V.

--

-- Object : Entity soc.color_correction

-- Last modified : Thu Oct 14 12:06:04 2010.

```
library work, ieee, unisim;

use work.global_generics_pkg.all;

use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;

use ieee.std_logic_unsigned.all;

use unisim.VCOMPONENTS.all;


entity color_correction is

port(

blue_c : out std_logic_vector(PXL_DATA_WIDTH+1 downto 0);

blue_int : in std_logic_vector(PXL_DATA_WIDTH-1 downto 0);

cc1_out : in std_logic_vector(7 downto 0);

cc2_out : in std_logic_vector(7 downto 0);

cc3_out : in std_logic_vector(7 downto 0);

cc4_out : in std_logic_vector(7 downto 0);

cc5_out : in std_logic_vector(7 downto 0);

cc6_out : in std_logic_vector(7 downto 0);

cc7_out : in std_logic_vector(7 downto 0);

cc8_out : in std_logic_vector(7 downto 0);

cc9_out : in std_logic_vector(7 downto 0);

cc_shift_now : in std_logic_vector(26 downto 0);

clk : in std_logic;

fen_out_cc : out std_logic;

fen_out_int : in std_logic;

green_c : out std_logic_vector(PXL_DATA_WIDTH+1 downto 0);

green_int : in std_logic_vector(PXL_DATA_WIDTH-1 downto 0);

len_out_cc : out std_logic;

len_out_int : in std_logic;

red_c : out std_logic_vector(PXL_DATA_WIDTH+1 downto 0);

red_int : in std_logic_vector(PXL_DATA_WIDTH-1 downto 0);

reset : in std_logic;

signs_now : in std_logic_vector(5 downto 0));


end entity color_correction ;


----------------------------------------------------------------------------------

-- Object : Architecture soc.color_correction.a0

-- Last modified : Thu Oct 14 12:06:04 2010.

----------------------------------------------------------------------------------

architecture a0 of color_correction is


signal Net_0 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);

signal Net_2 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);

signal Net_3 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);

signal Net_4 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);

signal Net_5 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
```

```
signal Net_6 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
signal Net_7 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
signal Net_8 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
signal Net_10 : std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
signal Net_29 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_30 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_31 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_32 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_33 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_34 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_35 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_36 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
signal Net_37 : std_logic_vector(PXL_DATA_WIDTH+7 downto 0);

component lut_mux_reg
port(
b1 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
b2 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
b3 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
blue1 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
blue2 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
blue3 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
cc_shift_now : in std_logic_vector(26 downto 0);
g1 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
g2 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
g3 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
green1 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
green2 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
green3 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
r1 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
r2 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
r3 : in std_logic_vector(PXL_DATA_WIDTH+7 downto 0);
red1 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
red2 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
red3 : out std_logic_vector(PXL_DATA_WIDTH+5 downto 0));

end component lut_mux_reg ;

component correct_color
port(
blue1 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
blue2 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
blue3 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
blue_c : out std_logic_vector(PXL_DATA_WIDTH+1 downto 0);
clk : in std_logic;
```

```
green1 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
green2 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
green3 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
green_c : out std_logic_vector(PXL_DATA_WIDTH+1 downto 0);
red1 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
red2 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
red3 : in std_logic_vector(PXL_DATA_WIDTH+5 downto 0);
red_c : out std_logic_vector(PXL_DATA_WIDTH+1 downto 0);
reset : in std_logic;
signs_now : in std_logic_vector(5 downto 0));

end component correct_color ;

component delay_flag
generic(
delay : Integer := 6);
port(
clk : in std_logic;
flag_in : in std_logic;
flag_out : out std_logic;
reset : in std_logic);

end component delay_flag ;

component MultiplierNxM
generic(
M : integer := 10;
N : integer := 8);
port(
clk : in std_logic;
d_in : in std_logic_vector(N-1 downto 0);
do : out std_logic_vector(M+N-1 downto 0);
reg : in std_logic_vector(M-1 downto 0);
reset : in std_logic);

end component MultiplierNxM ;

begin

u18: lut_mux_reg
port map(
b1 => Net_35,
b2 => Net_36,
b3 => Net_37,
blue1 => Net_2,
blue2 => Net_0,
blue3 => Net_10,
```

```
cc_shift_now => cc_shift_now,
g1 => Net_32,
g2 => Net_33,
g3 => Net_34,
green1 => Net_5,
green2 => Net_4,
green3 => Net_3,
r1 => Net_29,
r2 => Net_30,
r3 => Net_31,
red1 => Net_8,
red2 => Net_7,
red3 => Net_6);

u19: correct_color
port map(
blue1 => Net_2,
blue2 => Net_0,
blue3 => Net_10,
blue_c => blue_c,
clk => clk,
green1 => Net_5,
green2 => Net_4,
green3 => Net_3,
green_c => green_c,
red1 => Net_8,
red2 => Net_7,
red3 => Net_6,
red_c => red_c,
reset => reset,
signs_now => signs_now);

u9: delay_flag
generic map(
delay => 3)
port map(
clk => clk,
flag_in => fen_out_int,
flag_out => fen_out_cc,
reset => reset);

u10: delay_flag
generic map(
delay => 3)
port map(
```

```
clk => clk,
flag_in => len_out_int,
flag_out => len_out_cc,
reset => reset);

u11: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc1_out,
do => Net_29,
reg => red_int,
reset => reset);

u0: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc2_out,
do => Net_30,
reg => green_int,
reset => reset);

u1: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc3_out,
do => Net_31,
reg => blue_int,
reset => reset);

u2: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc4_out,
do => Net_32,
```

```
reg => red_int,
reset => reset);

u3: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc5_out,
do => Net_33,
reg => green_int,
reset => reset);

u4: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc6_out,
do => Net_34,
reg => blue_int,
reset => reset);

u5: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc7_out,
do => Net_35,
reg => red_int,
reset => reset);

u6: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc8_out,
do => Net_36,
reg => green_int,
reset => reset);
```

```
u7: MultiplierNxM
generic map(
M => PXL_DATA_WIDTH,
N => 8)
port map(
clk => clk,
d_in => cc9_out,
do => Net_37,
reg => blue_int,
reset => reset);
end architecture a0 ; -- of color_correction
```

编辑于 2021-01-18 15:36

Camera 图像信号处理器ISP (Image Signal Processor)