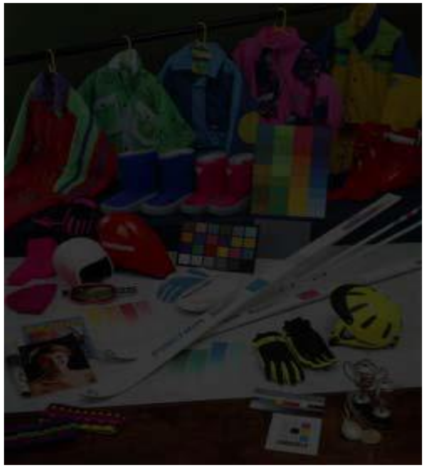
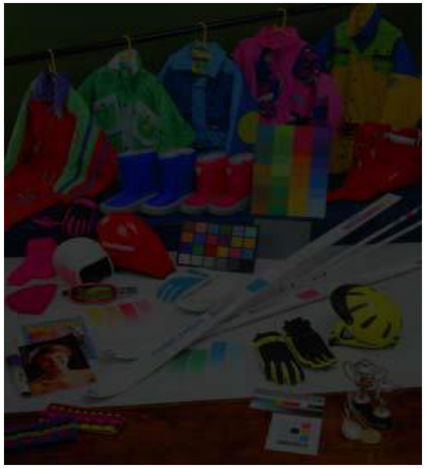
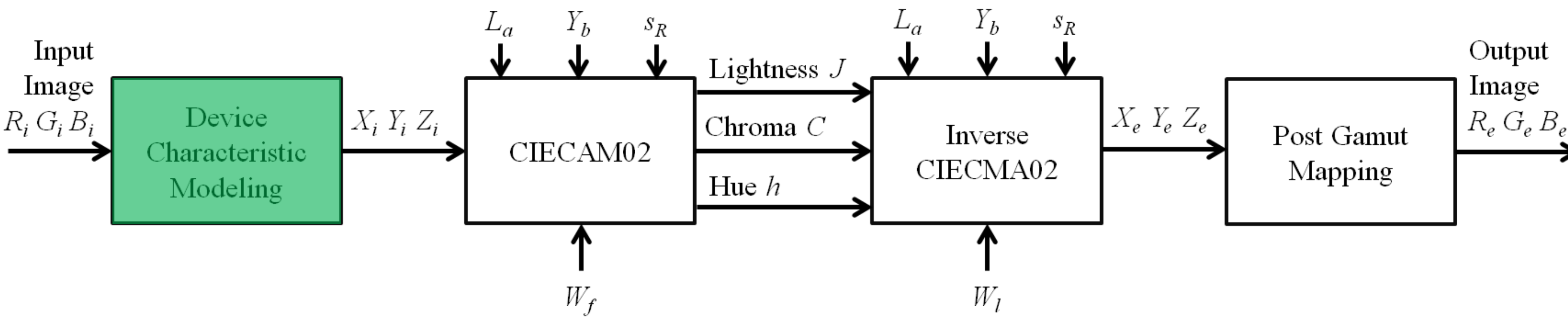
Exploiting Perceptual Anchoring for Color Image Enhancement

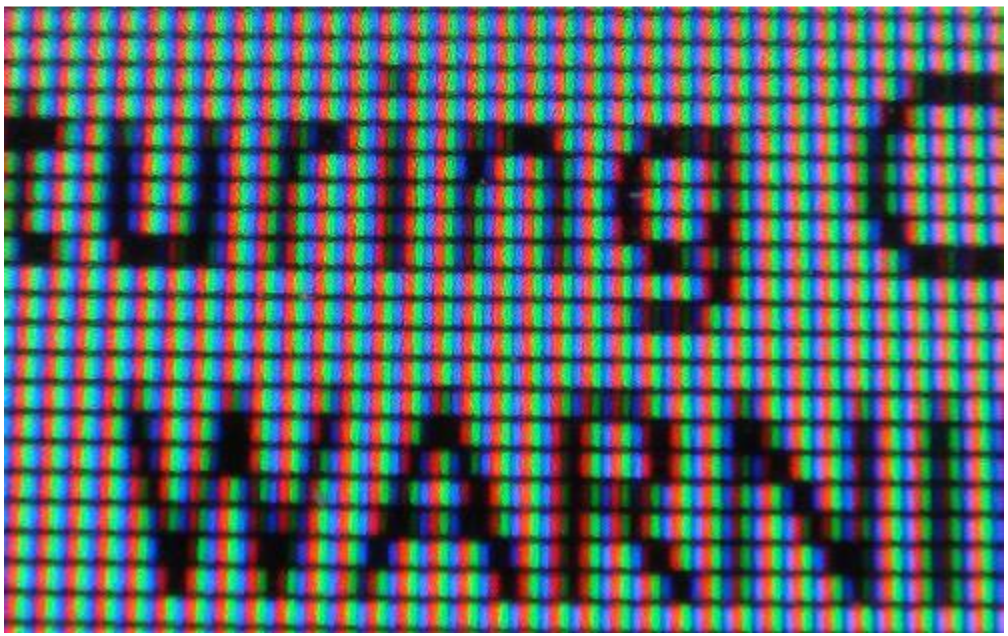




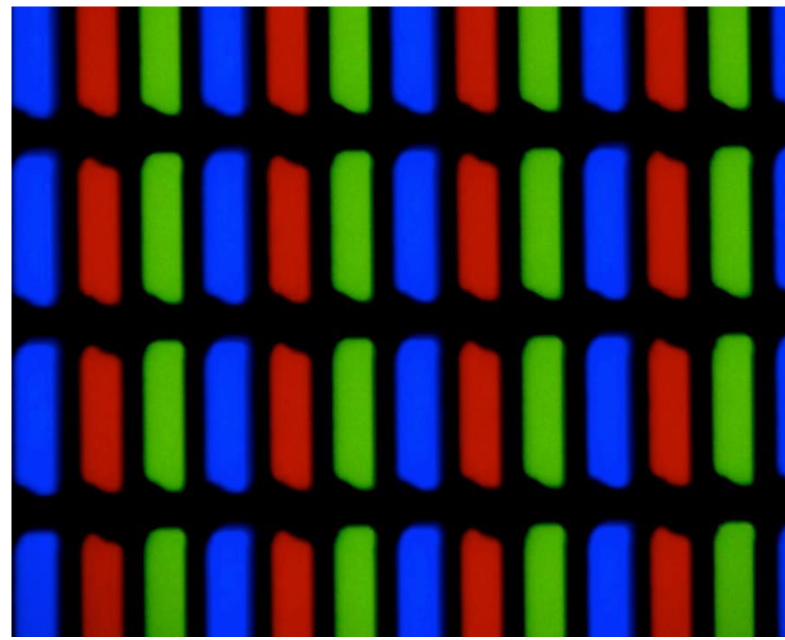
1. K.-T. Shih and H. H. Chen, “Exploiting perceptual anchoring for color image enhancement,” IEEE Trans. Multimedia, vol. 18, no. 2, pp. 300-310, Feb. 2016
2. T.-H. Huang, T.-C. Wang, and H. H. Chen, “Radiometric compensation of images projected on non-white surfaces by exploiting chromatic adaptation and perceptual anchoring,” IEEE Trans. Image Process., vol. 26, no. 1, pp. 147-159, Jan. 2017.

Flow Chart of the Algorithm

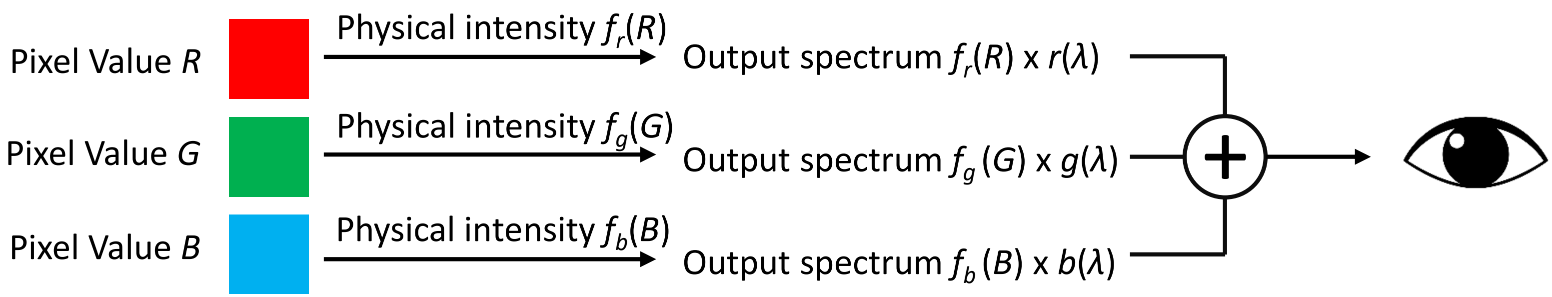


Device Characteristic Modeling





Device Characteristic Modeling



$$X = \int_{400nm}^{700nm} \left\{ f_r(R) r(\lambda) + f_g(G) g(\lambda) + f_b(B) b(\lambda) \right\} x(\lambda) d\lambda$$

$$= f_r(R) \int_{400nm}^{700nm} r(\lambda) x(\lambda) d\lambda + f_g(G) \int_{400nm}^{700nm} g(\lambda) x(\lambda) d\lambda + f_b(B) \int_{400nm}^{700nm} b(\lambda) x(\lambda) d\lambda$$

$$\triangleq m_{rx} f(R) + m_{gx} g(G) + m_{bx} h(B)$$

Similarly,

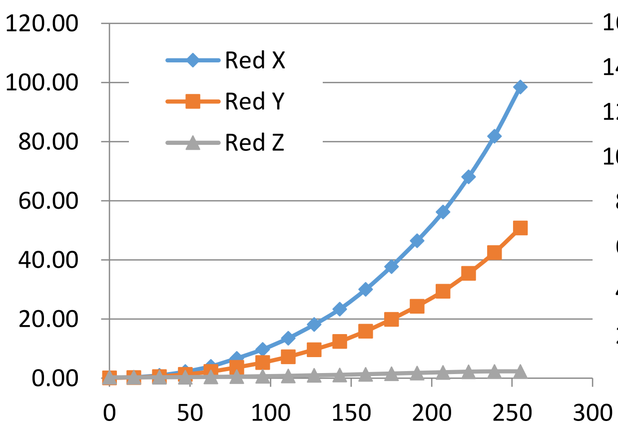
$$Y = m_{ry} f(R) + m_{gy} g(G) + m_{by} h(B)$$

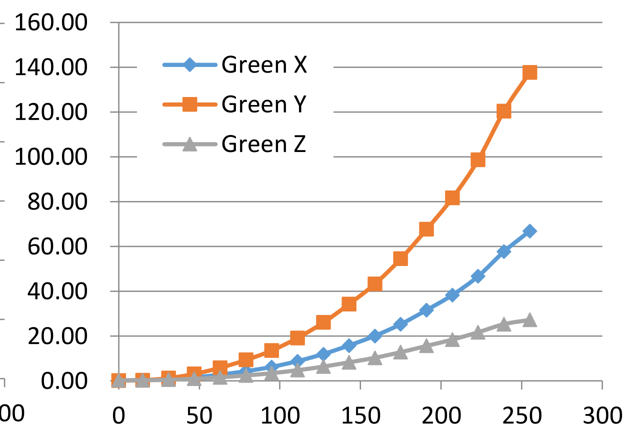
$$Z = m_{rz} f(R) + m_{gz} g(G) + m_{bz} h(B)$$

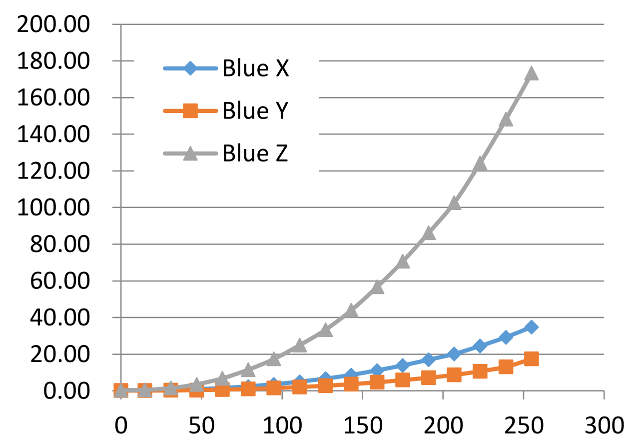
Device Characteristic Modeling

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} m_{rx} & m_{gx} & m_{bx} \\ m_{ry} & m_{gy} & m_{by} \\ m_{rz} & m_{gz} & m_{bz} \end{bmatrix} \begin{bmatrix} R^{\gamma_r} \\ G^{\gamma_g} \\ B^{\gamma_b} \end{bmatrix} = \mathrm{M} \begin{bmatrix} R_l \\ G_l \\ B_l \end{bmatrix},$$

Note that R, G, and B are *normalized* pixel values ranging from 0 to 1 (not the 8-bit number from 0 to 255)







The Estimated Display Parameters

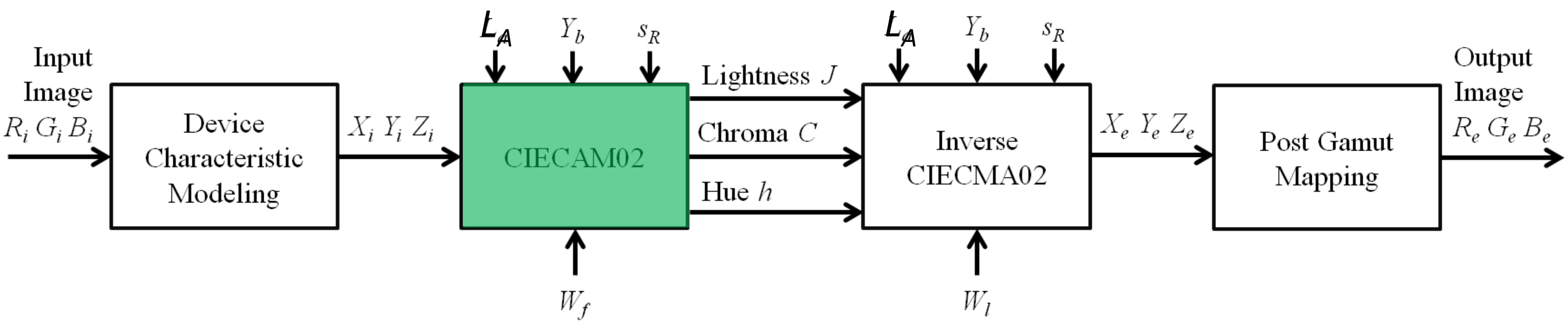
The full-backlight display (subscript *f*)

Parameter*	Value
$\gamma_{r,f}$	2.4767
$\gamma_{g,f}$	2.4286
$\gamma_{b,f}$	2.3792
$\mathbf{M}_f$	$\begin{bmatrix} 95.57 & 64.67 & 33.01 \\ 49.49 & 137.29 & 14.76 \\ 0.44 & 27.21 & 169.83 \end{bmatrix}$

The low-backlight display (subscript *l*)

Parameter*	Value
$\gamma_{r,l}$	2.2212
$\gamma_{g,l}$	2.1044
$\gamma_{b,l}$	2.1835
$\mathbf{M}_l$	$\begin{bmatrix} 4.61 & 3.35 & 1.78 \\ 2.48 & 7.16 & 0.79 \\ 0.28 & 1.93 & 8.93 \end{bmatrix}$

Flow Chart of the Algorithm



The Prediction of Appearance

Inputs

Stimulus: (X, Y, Z)
White point: (X_w, Y_w, Z_w)
Adapting luminance: L_A
Relative luminance of the background: Y_b
Surround condition

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = M_f \begin{bmatrix} 1^{\gamma_r} \\ 1^{\gamma_g} \\ 1^{\gamma_b} \end{bmatrix} = \begin{bmatrix} m_{f,rx} + m_{f,gx} + m_{f,bx} \\ m_{f,ry} + m_{f,gy} + m_{f,by} \\ m_{f,rz} + m_{f,gz} + m_{f,bz} \end{bmatrix}$$

Suggested values: $L_A = 60$, $Y_b = 25$

LMS Space Conversion → Gain Control → Non-Linear Compression → Opponent Color Conversion → Computing Perceptual Attributes

Chromatic Transform

Lightness J
Chroma C
Hue h
Brightness Q
Colorfulness M
Saturation s

Step 1: Determining Parameters

Surround condition	c	N_c	F
Average surround	0.69	1.0	1.0
Dim surround	0.59	0.9	0.9
Dark surround	0.525	0.8	0.8

c: an exponential nonlinearity (used in the computation of lightness and brightness)
N_c: the chromatic induction factor (used in the computation of chroma)
F: the maximum degree of adaptation (used in chromatic transform)

Step 2: Chromatic Transform

- Step 2a: LMS space conversion (XYZ → LMS)
- For the target stimulus

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \mathrm{M}_{CAT02} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \quad \mathbf{M}_{CAT02} = \begin{bmatrix} 0.7328 & 0.4296 & -0.1624 \\ -0.7036 & 1.6975 & 0.0061 \\ 0.0030 & 0.0136 & 0.9834 \end{bmatrix}$$

- For the reference white

$$\begin{bmatrix} L_w \\ M_w \\ S_w \end{bmatrix} = \mathrm{M}_{CAT02} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

Step 2: Chromatic Transform (cont'd)

- The gain control is independent in each of the three types of cone cells
- Step 2b: Compute the degree of adaptation

$$D = F\left[1 - \left(\frac{1}{3.6}\right)e^{-(L_A+42)/92}\right]$$

- Step 2c: Von-Kries-Type Gain control

$$L_c = \left(\frac{100}{L_w}D + 1 - D\right)L$$

$$M_c = \left(\frac{100}{M_w}D + 1 - D\right)M$$

$$S_c = \left(\frac{100}{S_w}D + 1 - D\right)S$$

Step 3: Non-Linear Compression

- In CIECAM02, the non-linear compression and the von-Kries-type gain control are carried out in different color spaces for better accuracy
- Step 3a: first compute the necessary parameter F_L:

$$k = \frac{1}{5L_A + 1}$$

$$F_L = 0.2k^4 (5L_A) + 0.1(1 - k^4)^2 (5L_A)^{1/3}$$

luminance-level adaptation factor

Step 3: Non-Linear Compression (cont'd)

- Step 3b: Convert the adapted LMS value (L_c, M_c, S_c) to Hunt-Pointer-Estévez (HPE) space for response compression

$$\begin{bmatrix} L' \\ M' \\ S' \end{bmatrix} = \mathbf{M}_H \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \underbrace{\mathbf{M}_H}_{\text{LMS}\rightarrow\text{HPE}} \underbrace{\mathbf{M}_{CAT02}^{-1}}_{\text{LMS}\rightarrow\text{XYZ}} \begin{bmatrix} L_c \\ M_c \\ S_c \end{bmatrix},$$

where $\mathbf{M}_H = \begin{bmatrix} 0.38971 & 0.68898 & -0.07868 \\ -0.22981 & 1.18340 & 0.04641 \\ 0.00000 & 0.00000 & 1.00000 \end{bmatrix}$

Step 3: Non-Linear Compression (cont'd)

- Step 3c: Non-linear compression

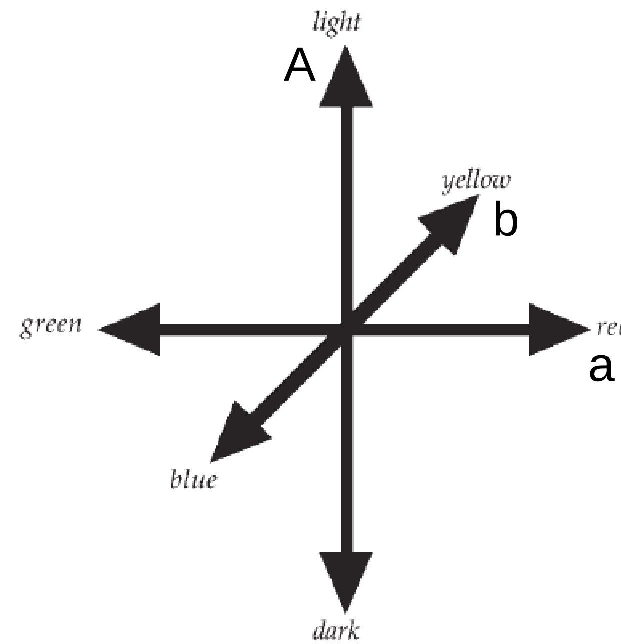
$$L'_a = \frac{400(F_L L'/100)^{0.42}}{27.13 + (F_L L'/100)^{0.42}} + 0.1$$

$$M'_a = \frac{400(F_L M'/100)^{0.42}}{27.13 + (F_L M'/100)^{0.42}} + 0.1$$

$$S'_a = \frac{400(F_L S'/100)^{0.42}}{27.13 + (F_L S'/100)^{0.42}} + 0.1$$

Step 4: Opponent Color Conversion

- Conversion from the cone-response space to opponent space



$$C_1 = L'_a - M'_a$$
$$C_2 = M'_a - S'_a$$
$$C_3 = S'_a - L'_a$$

$$A = \left(2L'_a + M'_a + \tfrac{1}{20}S'_a - 0.305\right)N_{bb}$$
$$a = C_1 - \tfrac{1}{11}C_2 \qquad = L'_a - \tfrac{12}{11}M'_a + \tfrac{1}{11}S'_a$$
$$b = \tfrac{1}{2}\left(C_2 - C_1 + C_1 - C_3\right)/4.5 \quad = \tfrac{1}{9}\left(L'_a + M'_a - 2S'_a\right)$$

Step 5: Computing the Perceptual Attributes

- Step 5a: compute the necessary parameters:

$$n = \frac{Y_{\mathrm{b}}}{Y_{\mathrm{W}}}$$

$$N_{\mathrm{bb}} = N_{\mathrm{cb}} = 0.725\left(\frac{1}{n}\right)^{0.2}$$ Induction factors

$$z = 1.48 + \sqrt{n}$$ Base exponential non-linearity

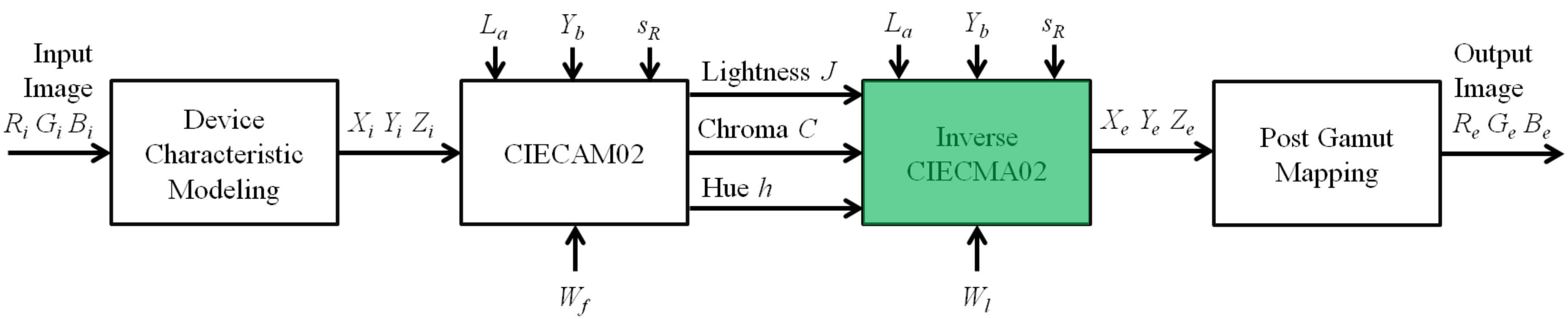
Step 5: Computing the Perceptual Attributes (cont'd)

- Step 5b: compute the perceptual attributes (final output):

Hue	$h = \angle(a, b),\ (0 < h < 360^\circ)$
Lightness	$J = 100\,(A/A_w)^{cz}$
Brightness	$Q = (4/c)\sqrt{\frac{1}{100}J}\,(A_w + 4)\,F_L^{1/4}$
Chroma	$C = t^{0.9}\sqrt{\frac{1}{100}J}(1.64 - 0.29^n)^{0.73}$
Colorfulness	$M = C \cdot F_L^{1/4}$
Saturation	$s = 100\sqrt{M/Q}$

- In this work, only lightness, chroma, and hue are used

Flow Chart of the Algorithm



Eccentricity factor

$$e_t = \frac{1}{4}\left[\cos\left(h\frac{\pi}{180}+2\right)+3.8\right]$$

Inversion of the Appearance Model

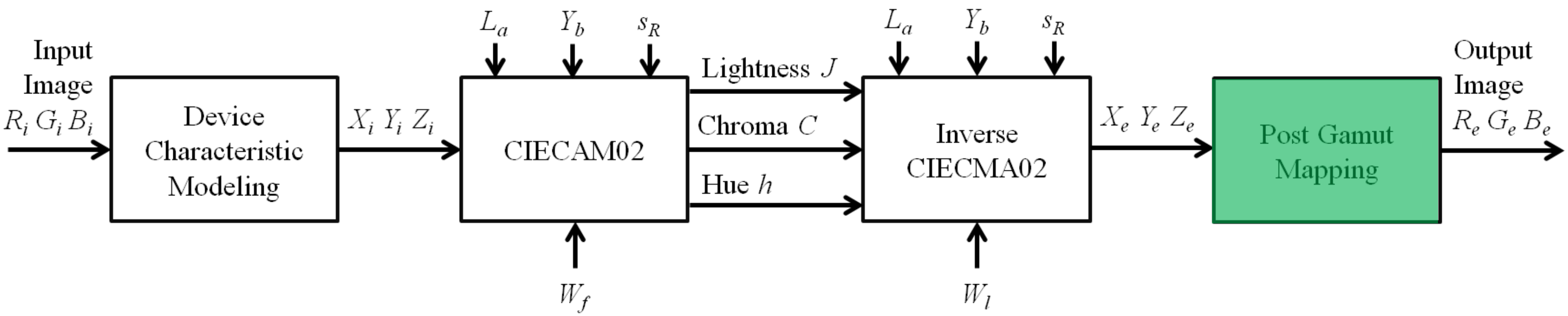
- A rough sketch of the computation of inverse CIECAM02:

  Step 1. Calculate t from C and J.
  ~~Step 2. Calculate e_t from h.~~
  Step 3. Calculate A from A_w and J.
  Step 4. Calculate a and b from t, $\cancel{e_t}$, h, and A.
  Step 5. Calculate L_a', M_a', and S_a' from A, a, and b.
  Step 6. Use the inverse nonlinearity to compute L', M', and S'.
  Step 7. Convert to L_c, M_c, and S_c via linear transform.
  Step 8. Invert the chromatic adaptation transform to compute L, M, and S and then X, Y, and Z.

- Replace the white point with the low-backlight one:

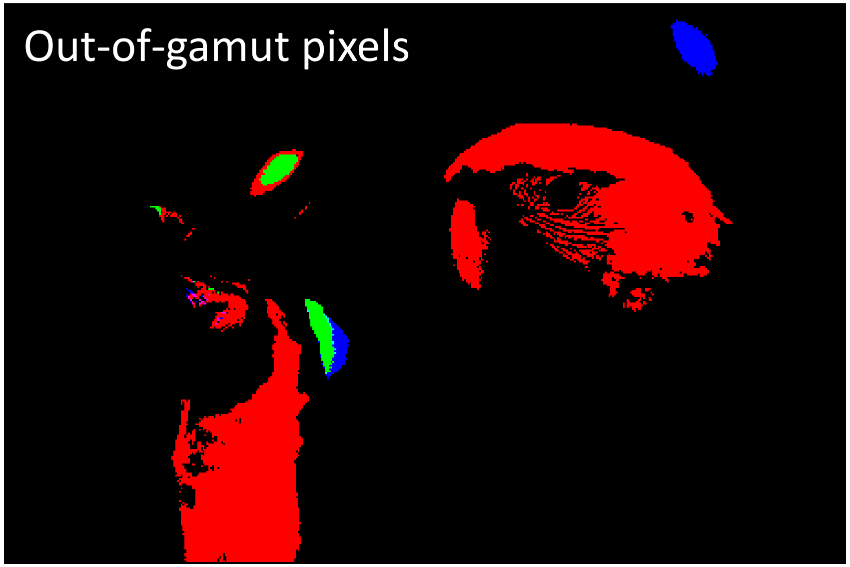
$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = M_l \begin{bmatrix} 1^{\gamma_r} \\ 1^{\gamma_g} \\ 1^{\gamma_b} \end{bmatrix} = \begin{bmatrix} m_{l,rx} + m_{l,gx} + m_{l,bx} \\ m_{l,ry} + m_{l,gy} + m_{l,by} \\ m_{l,rz} + m_{l,gz} + m_{l,bz} \end{bmatrix}$$

Flow Chart of the Algorithm



Post Gamut Mapping

- Some enhanced pixels are not displayable by the low-backlight display
- We need to put those out-of-gamut pixels back into the low-backlight display gamut



Post Gamut Mapping (cont'd)

- Step 1: Determine whether a pixel is in or out of the gamut by converting the XYZ values (X_e, Y_e, Z_e) to RGB values (R', G', B')
  - A pixel is out-of-gamut if and only if R', G', or B' is not in the interval [0 1].

Gamma values of the low-backlight display

$$(R', G', B') = (R_{e,l}{}^{1/\gamma_{r,l}}, G_{e,l}{}^{1/\gamma_{g,l}}, B_{e,l}{}^{1/\gamma_{b,l}}),$$

where

Color mixing matrix of the low-backlight display

$$[R_{e,l} \quad G_{e,l} \quad B_{e,l}]^{\mathrm{T}} = \mathbf{M}_l^{-1}[X_e \quad Y_e \quad Z_e]^{\mathrm{T}}$$

Post Gamut Mapping (cont'd)

- Step 2: Clipping the RGB values with a hard threshold

$$(R_c, G_c, B_c) = (f(R'), f(G'), f(B')),$$

where $$f(x) = \begin{cases} x, & \text{if } 0 \le x \le 1 \\ 1, & \text{if } x > 1 \\ 0, & \text{if } x < 0 \end{cases}$$

Post Gamut Mapping (cont'd)

- Problem of hard clipping: loss of details

Original

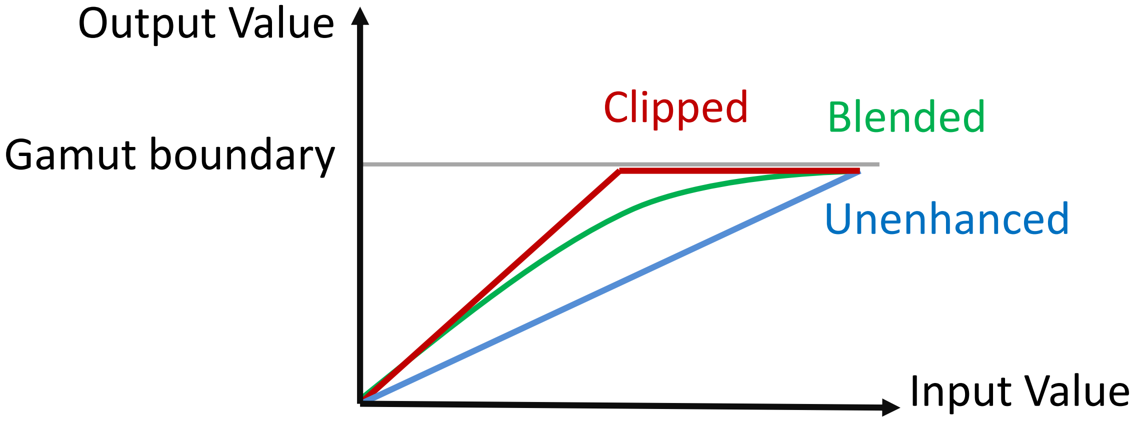
Clipped

Post Gamut Mapping (cont'd)

- Step 3: Blend the clipped pixel value with the original pixel value:

$$\begin{bmatrix} R_e \\ G_e \\ B_e \end{bmatrix} = (1 - JC) \begin{bmatrix} R_c \\ G_c \\ B_c \end{bmatrix} + JC \begin{bmatrix} R_i \\ G_i \\ B_i \end{bmatrix}$$

where (R_i, G_i, B_i) is the original unenhanced pixel value, and J and C, respectively, represents lightness and chroma of that pixel



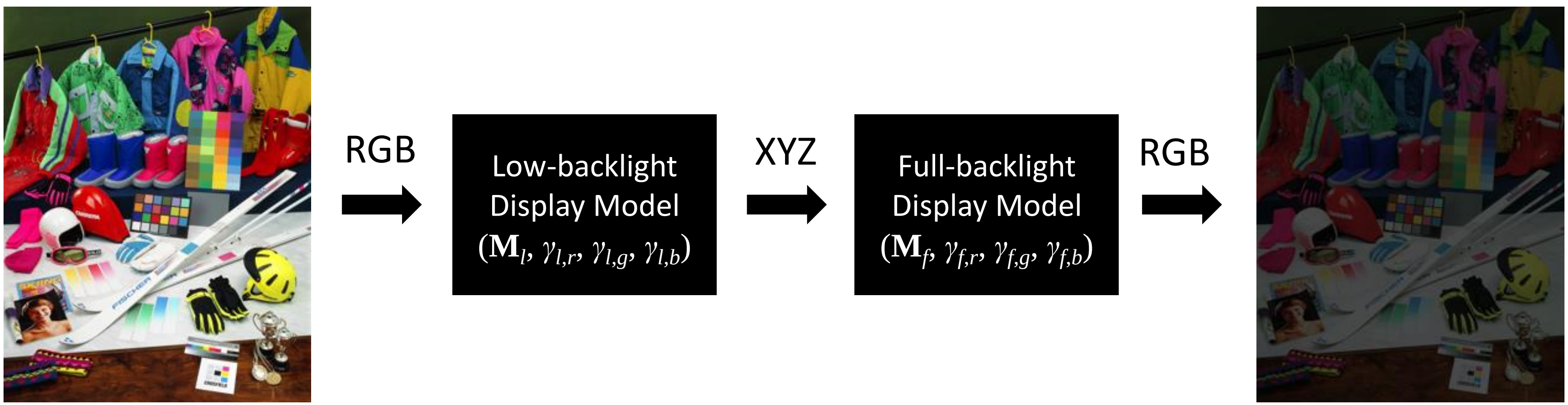
Post Gamut Mapping (cont’d)

Original

Clipped

Our method

Simulating Images Illuminated with Dim Backlight



Simulating Images Illuminated with Dim Backlight

Original image

Enhanced image

Simulating Images Illuminated with Dim Backlight

Original image illuminated with dim backlight

Enhanced image illuminated with dim backlight

Matlab Implementation Tips

- Try to think in matrix when you write Matlab codes
- Use as few “for” as possible

slow

```
A = rand(100,100);
for i = 1:100
    for j = 1:100
        A(i,j) = A(i,j) * A(i,j);
    end
end
```

fast

```
A = rand(100,100);
A = A.*A;
```

or

```
A = rand(100,100);
A = A.^2;
```

- You can reshape your matrix to avoid using “for”
  - Rearrange matrices using functions like `reshape()` and `permute()`

Matlab Implementation Tips

slow

```
A = rand(100,100);
for i = 1:100
    for j = 1:100
        if A(i,j) > 0.5,
            A(i,j) = A(i,j) * A(i,j);
        else
            A(i,j) = sqrt(A(i,j));
        end
    end
end
```

fast

```
A = rand(100,100);
mask = A > 0.5; %a binary mask
A(mask) = A(mask).^2;
A(~mask) = A(~mask).^0.5;
```

or

```
A = rand(100,100);
mask = double(A>0.5); %cast to double
A = mask.*A.^2 + (1-mask).*A.^0.5;
```

Matlab Implementation Tips

slow

```
img_rgb = imread('a_color_image.png'); %size(A) is [height width 3]
img_rgbsum = zeros(size(img,1), size(img,2));
for i = 1:size(img,1)
    for j = 1:size(img,2)
        img_rgbsum(i,j) = img_rgb(i,j,1)+img_rgb(i,j,2)+img_rgb(i,j,3);
    end
end
```

fast

```
img_rgb = imread('a_color_image.png');
img_rgbsum = img_rgb(:,:,1)+img_rgb(:,:,2)+img_rgb(:,:,3);
```

or

```
img_rgb = imread('a_color_image.png');
img_rgbsum = sum(img_rgbsum,3);
```

Matlab Implementation Tips

- Convert images to double before performing numerical operations
  - unit8 is always in the range [0 255] and may not give you the desired result
  - We usually normalize the value to [0 1]
    - For a `uint8` image, 0=black and 255=white
    - For a `double` image, 0=black and 1=white

```
img_rgb = double(imread(‘a_color_image.png’))/255;
```

- Save images in PNG format to avoid compression artifact
  - Do not save image in JPEG format using Matlab (low compression quality)

```
imwrite(img_out, ‘output.png’, ‘png’);
```