
Meet iCAM: An Image Color Appearance

Model

Garrett M. Johnson

Mark D. Fairchild

Tone Reproduction for High Dynamic Range Images (converted to *Mathematica* v.6)

Rev1

Changes from original iCAM_hidyn_new.V6.nb denoted in [blue](#).

■ Initializations

■ Input

First read in the input image, and display the RGB version. We first need to do a little *Mathematica* house cleaning to get the path of the input image.

```
nbinfo = NotebookInformation[EvaluationNotebook[]];
dir = ("FileName" /. nbinfo /.
      FrontEnd`FileName[d_List, nam_, ___] :=> ToFileName[d]);
```

Read in the High-dynamic range image. This image was originally generated by Debevec, and can be found at <http://www.debevec.org/Research/HDR/>

The Images are originally in Greg Larsons' Radiance image format. Radiance was used to convert these images to a standard binary image format, and then embedded in an HDF format. Radiance can be found at:

<http://radsite.lbl.gov/radiance/HOME.html>

information on the HDF image format can be found at:

<http://hdf.ncsa.uiuc.edu/>

The HDF image is in reverse order from what *Mathematica* expects, so we must reverse the input of the Import function.

```
Import[StringJoin[dir, "memorial_small.hdf"],
        {"HDF", "Elements"}]

{Data, DataFormat, Datasets, Dimensions}
```

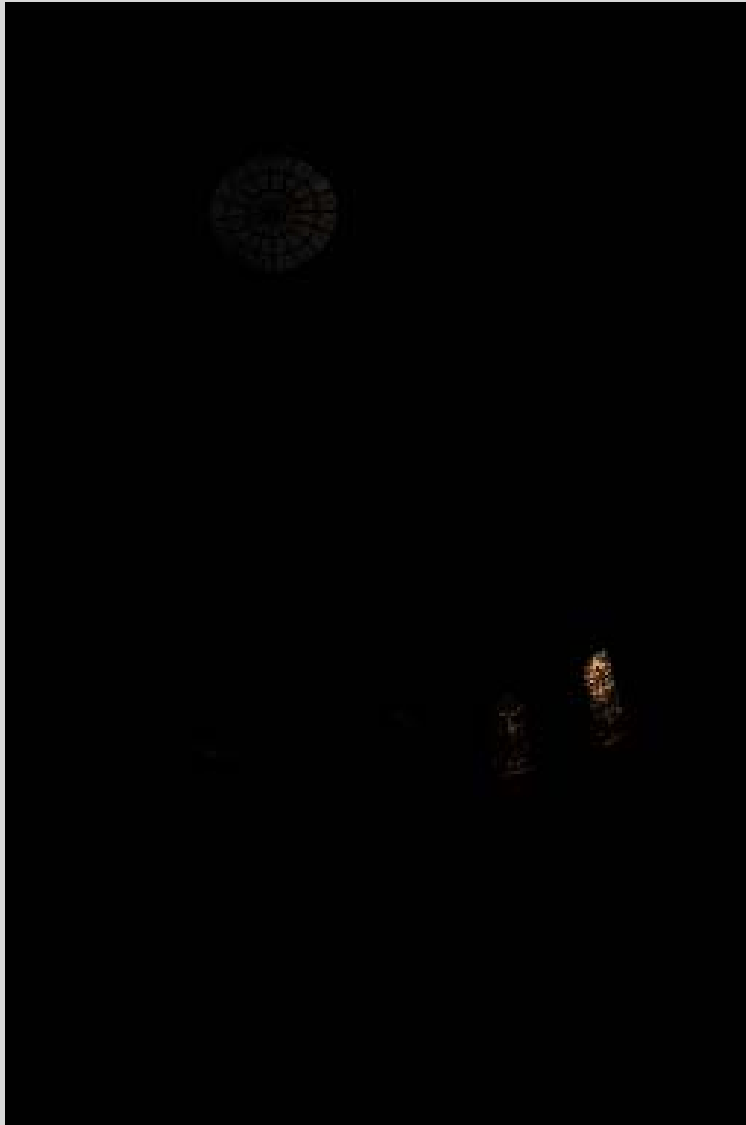
```
Import[StringJoin[dir, "memorial_small.hdf"], {"Datasets"}]

{hdr image}
```

```
rgbImage =
  Reverse[Import[StringJoin[dir, "memorial_small.hdf"],
    {"Datasets", "hdr image"}]];
```

Render the HDR linear image. Except for a few highlights, it is almost completely dark.

```
Show[  
Graphics[Raster[Apply[{#1, #2, #3} &,  $\frac{\text{rgbImage}}{\text{Max}[\text{rgbImage}]}$ , {2}]]],  
AspectRatio -> Automatic]
```



Find the dimensions of the image, for future calculations

```
imDim = Dimensions[rgbImage]

{384, 256, 3}
```

```
imWidth = imDim[[2]];
```

```
imHeight = imDim[[1]];
```

■ Gamma Correction

Generally this stage involves linearizing the image, before transforming to XYZ values. Since the HDR image is already linear, this is not necessary. We will normalize the data to be between 0-1, and display it with a gamma correction, just to illustrate that a gamma correction is not enough for HDR images.

Generally, PC users have $\gamma=2.4$; Mac users 1.8

```
 $\gamma = 1.8;$ 
```

```
 $\gamma = 2.4;$ 
```

```
dcRGB =  $\left( \frac{\text{rgbImage}}{\text{Max}[\text{rgbImage}]} \right);$ 
```

```
Show[Graphics[Raster[Apply[{{#1, #2, #3} &, dcRGB1/γ, {2}}]],  
AspectRatio → Automatic]
```



■ RGB to XYZ Color Conversion

Convert the linearized RGB signal into XYZ using a standard sRGB matrix normalized for D65. Note that the *Mathematica* matrix convention is transposed from typical matrix descriptions.

```
sRGBmat =  $\begin{pmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{pmatrix}$ ;
```

```
xyzIm = (rgbImage.Transpose[sRGBmat]);
```

Isolate the individual X, Y, and Z channels

```
{xIm = Take[xyzIm[[All, All, 1]]],  
 yIm = Take[xyzIm[[All, All, 2]]],  
 zIm = Take[xyzIm[[All, All, 3]]];
```

Plot the XYZ images side-by-side for examination

```
Show[  
 GraphicsRow[{Graphics[Raster[xIm], AspectRatio → Automatic],  
 Graphics[Raster[yIm], AspectRatio → Automatic],  
 Graphics[Raster[zIm], AspectRatio → Automatic]}]]
```



```
Max[yIm]
```

```
184.114
```

```
Min[yIm]

0.000677667
```

■ Determine Image "White" for Chromatic Adaptation

Create a smoothing kernel to use as the low-pass filter for the discrete convolution. This can be replaced with frequency filtering, if desired.

```
kernSize =  $\frac{\text{imWidth}}{1}$ ;
```

Define a Gaussian convolution kernel

```
kern = N[Table[e- $\left(\frac{\sqrt{x^2+y^2}}{\left(\frac{\text{kernSize}}{40}\right)^2}\right)$ , {x, - $\frac{\text{kernSize}-1}{2}$ ,  $\frac{\text{kernSize}-1}{2}$ },
  {y, - $\frac{\text{kernSize}-1}{2}$ ,  $\frac{\text{kernSize}-1}{2}$ }]]];
```

Normalize the convolution kernel so that it sums to 1.

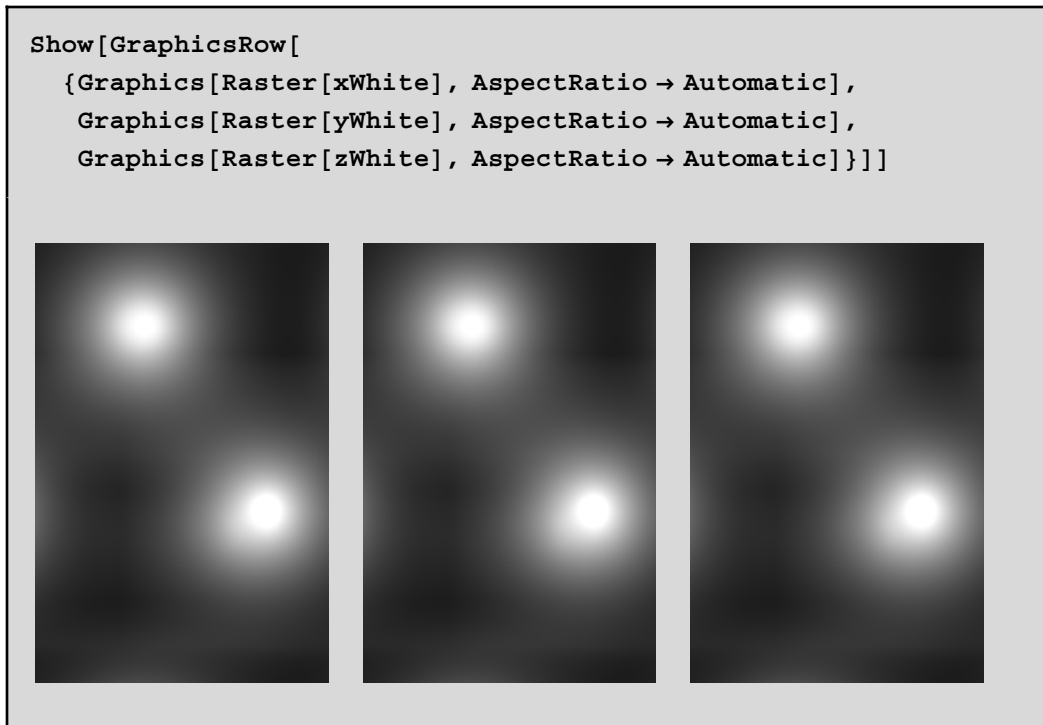
```
kern = kern / Apply[Plus, Flatten[kern]];
```

```
yWhite = RotateLeft[ListConvolve[kern, yIm, {{1, 1}, {1, 1}},
  { kernSize / 2, kernSize / 2 }];
```

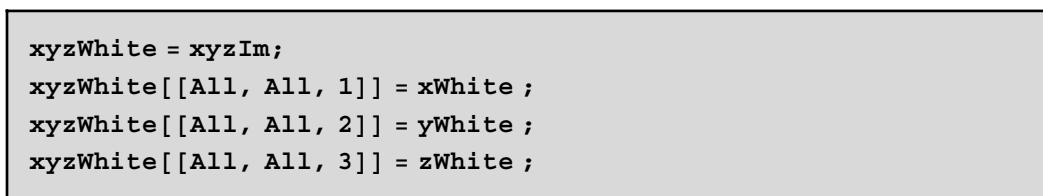
```
xWhite = yWhite;
```

```
zWhite = yWhite;
```

Plot the blurred XYZ images



Combine the individual XYZ images back into a single [x,y,3] image.



■ Chromatic Adaptation

Set up some variables for the chromatic adaptation transform. This utilizes a slightly modified version of the CIECAM02 transform to include non-linear exponential functions, if desired.

■ Adapting Luminance



■ Surround Conditions



Degree of Adaptation

$$D_{\text{fact}} = F \left(1 - \frac{1}{1 + 2 \text{La}^{\frac{1.0}{4.0}} + \frac{\text{La}^2}{300.0}} \right)$$

0.975405

$$CIED_f = F \left(1 - (1 / 3.6) e^{\left(\frac{-\text{La} - 42}{92} \right)} \right)$$

0.940656

$D_f = 0.1$

0.1

■ Adapting Whitepoint

```
whitePoint = ( 95.05 100.0 108.9 ) / 100;
```

■ Chromatic Adaptation Matrix

$$mCAT02 = \begin{pmatrix} 0.7328 & 0.4296 & -0.1624 \\ -0.7036 & 1.6974 & 0.0061 \\ 0.0030 & 0.0136 & 0.9834 \end{pmatrix};$$

```
iCAT02 = Inverse[mCAT02];
```

```
MatrixForm[iCAT02]
```

$$\begin{pmatrix} 1.09611 & -0.278882 & 0.182743 \\ 0.454391 & 0.473556 & 0.0721012 \\ -0.00962787 & -0.0056983 & 1.01533 \end{pmatrix}$$

Calculate RGB values for input image, blurred image, and whitepoint

```
rgbI = xyzIm.Transpose[mCAT02];
```

```
rgbWhitePoint = whitePoint.Transpose[mCAT02]
{{0.949273, 1.03527, 1.08737}}
```

Calculate the RGB of the blurred "whitepoint" image. First define a function to threshold the minimum of the image to 1. This means the darkest RGB adapting value is 1, and prevents division by zero, or division by a small number.

```
threshold[x_] := If[x > .001, x, .001];
```

```
rgbWhite = xyzWhite.Transpose[mCAT02];
```

```
rgbWhite = Max[rgbI] rgbWhite / Max[rgbWhite];
```

■ Calculate the Maximum R, G, B value

```
{maxR = Max[rgbI[[All, All, 1]]],
 maxG = Max[rgbI[[All, All, 2]]],
 maxB = Max[rgbI[[All, All, 3]]]}
{192.228, 184.631, 129.467}
```

■ Calculate Adapted RGB Signals

$$R_c = \left(\frac{Df \text{ rgbWhitePoint}[[1, 1]]}{\text{rgbWhite}[[\text{All}, \text{All}, 1]]} + 1 - Df \right) * \text{rgbI}[[\text{All}, \text{All}, 1]];$$

$$G_c = \left(\frac{Df \text{ rgbWhitePoint}[[1, 2]]}{\text{rgbWhite}[[\text{All}, \text{All}, 2]]} + 1 - Df \right) * \text{rgbI}[[\text{All}, \text{All}, 2]];$$

$$Bc = \left(\frac{Df \text{rgbWhitePoint}[[1, 3]]}{\text{rgbWhite}[[A11, A11, 3]]} + 1 - Df \right) * \text{rgbI}[[A11, A11, 3]];$$

- **Convert Adapted RGB Signals back to XYZ, Examine the RGB color image for reference**

```
RGBc = rgbWhite * 0 ;
```

```
{RGBc[[A11, A11, 1]] = Rc,  
 RGBc[[A11, A11, 2]] = Gc, RGBc[[A11, A11, 3]] = Bc};
```

```
xyzAdapt = RGBc.Transpose[iCAT02];
```

```
rgbAdapt = xyzAdapt.Transpose[Inverse[sRGBmat]];
```

```
rgbAdapt = Map[threshold, rgbAdapt, {3}];
```

```
Show[Graphics[  
  Raster[Apply[{{#1, #2, #3} &,  $\left(\frac{\text{rgbAdapt}}{\text{Max}[\text{rgbAdapt}]}\right)^{1/\gamma}$ , {2}}]]],  
  AspectRatio -> Automatic]
```



■ Convert to IPT color space

The IPT matrix is defined to transform LMS cone signals into the opponent IPT space. The LMS functions are modified using a non-linear exponent function. Typically this is a value of 0.43, but in iCAM we modify this exponent based on a low-passed Y image.

$$\mathbf{lmsIPTm} = \begin{pmatrix} 0.4000 & 0.4000 & 0.2000 \\ 4.4550 & -4.8510 & 0.3960 \\ 0.8056 & 0.3572 & -1.1628 \end{pmatrix};$$

$$\mathbf{xyzLMSm} = \begin{pmatrix} 0.3971 & 0.68898 & -0.07868 \\ -0.22981 & 1.18340 & 0.04641 \\ 0.0 & 0.0 & 1.0 \end{pmatrix};$$

Now convert the XYZ adapted image into LMS cone space

```
lmsImage = xyzAdapt.Transpose[xyzLMSm];
```

■ Determine the exponents, based on the luminance of the input image

$$\mathit{iptKernSize} = \frac{\mathit{imWidth}}{1};$$

```
iptKern =
```

$$\mathbf{N}\left[\mathbf{Table}\left[e^{-\frac{\sqrt{x^2+y^2}}{\left(\frac{\mathit{iptKernSize}}{40}\right)^2}}, \left\{x, -\frac{\mathit{iptKernSize}-1}{2}, \frac{\mathit{iptKernSize}-1}{2}\right\}, \left\{y, -\frac{\mathit{iptKernSize}-1}{2}, \frac{\mathit{iptKernSize}-1}{2}\right\}\right]\right];$$

```
iptKern = iptKern / Apply[Plus, Flatten[iptKern]];
```

```
yLow = RotateLeft[ListConvolve[iptKern, yIm, {{1, 1}, {1, 1}}, {kernSize/2, kernSize/2}];
```

```
Show[Graphics[Raster[yLow]], AspectRatio -> Automatic]
```



```
Efact[L_] =
```

$$\left(\frac{1.0}{1.7}\right) \left(0.2 \left(\frac{1}{5L + 1.0}\right)^4 (5L) + 0.1 \left(1 - \left(\frac{1}{5L + 1.0}\right)^4\right)^2 (5L)^{1/3}\right)$$

$$0.588235 \left(\frac{1. L}{(1. + 5 L)^4} + 0.170998 L^{1/3} \left(1 - \frac{1}{(1. + 5 L)^4}\right)^2\right)$$

```
iptExp = Efact[ (1000 yLow) ];
```

Create a minimum threshold for the exponential, otherwise we will clip dark values to zero.

```
Max[iptExp] * .43
```

```
0.45753
```

```
minThresh[x_] := If[x < 0.3, 0.3, x];
```

```
iptExp = Map[minThresh, iptExp, {2}];
```

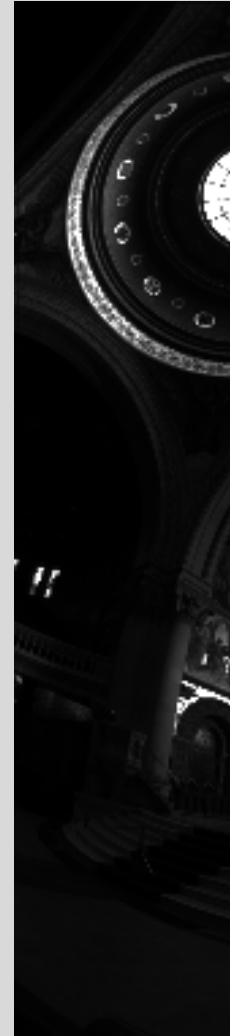
```
lmsImNL = lmsImageiptExp*.43;
```

- Convert the non-linear LMS signals into IPT space

```
iptImage = lmsImNL.Transpose[lmsIPTm];
```

Plot the "I" dimension, along with the Luminance (Y) image

```
Show[GraphicsRow[{Graphics[Raster[iptImage[All, All, 1]]1/1],  
  AspectRatio → Automatic], Graphics[  
  Raster[xyzIm[All, All, 2]], AspectRatio → Automatic]}]]
```



■ Inverse IPT transform

```
iptLMSm = Inverse[lmsIPTm];
```

```
lmsXYZm = Inverse [xyzLMSm];
```

```
xyzRGBm = Inverse [sRGBmat];
```

```
MatrixForm [iptLMSm]
```

$$\begin{pmatrix} 1. & 0.0975689 & 0.205226 \\ 1. & -0.113876 & 0.133217 \\ 1. & 0.0326151 & -0.676887 \end{pmatrix}$$

```
MatrixForm [lmsXYZm]
```

$$\begin{pmatrix} 1.88361 & -1.09664 & 0.199097 \\ 0.365787 & 0.632061 & -0.000553845 \\ 0. & 0. & 1. \end{pmatrix}$$

```
MatrixForm [xyzRGBm]
```

$$\begin{pmatrix} 3.24063 & -1.53721 & -0.498629 \\ -0.968931 & 1.87576 & 0.0415175 \\ 0.0557101 & -0.204021 & 1.057 \end{pmatrix}$$

```
lmsIPTimage = iptImage.Transpose [iptLMSm];
```

```
lmsIPTimage = Map [threshold, lmsIPTimage, {3}];
```

```
xyzIPTimage = lmsIPTimage $\frac{1}{0.43}$ .Transpose [lmsXYZm];
```

■ Inverse Chromatic Adaptation

```
rgbIPTim = xyzIPTimage.Transpose [mCAT02];
```

```
monWhite = (1.0 1.0 1.0);
```

```
rgbWhite = monWhite.Transpose[mCAT02]

{{1., 0.9999, 1.}}
```

Note we are setting the Dfactor to one for the inverse chromatic adaptation transform, so that we are completely adapting.

```
Df = 1.0
```

```
1.
```

$$Rc = \left(\frac{Df \text{ rgbWhite}[[1, 1]]}{\text{rgbWhitePoint}[[1, 1]]} + 1 - Df \right) * \text{rgbIPTim}[[All, All, 1]];$$

$$Gc = \left(\frac{Df \text{ rgbWhite}[[1, 2]]}{\text{rgbWhitePoint}[[1, 2]]} + 1 - Df \right) * \text{rgbIPTim}[[All, All, 2]];$$

$$Bc = \left(\frac{Df \text{ rgbWhite}[[1, 3]]}{\text{rgbWhitePoint}[[1, 3]]} + 1 - Df \right) * \text{rgbIPTim}[[All, All, 3]];$$

```
rgbIPTadapt = rgbIPTim * 0;
```

```
{rgbIPTadapt[[All, All, 1]] = Rc,
 rgbIPTadapt[[All, All, 2]] = Gc,
 rgbIPTadapt[[All, All, 3]] = Bc};
```

```
xyzIPTadapt = rgbIPTadapt.Transpose[iCAT02];
```

```
dcIPTimage = xyzIPTadapt.Transpose[xyzRGBm];
```

```
dcIPTimage = Map[threshold, dcIPTimage, {3}];
```

```
Max[lmsIPTimage]
```

```
10.1696
```

```
maRed = Quantile[Flatten[dcIPTimage[[All, All, 1]]], .98]
```

```
1.72396
```

```
miRed = Min[dcIPTimage[[All, All, 1]]];
```

```
maGreen = Quantile[Flatten[dcIPTimage[[All, All, 2]]], .98]
```

```
0.884742
```

```
miGreen = Min[dcIPTimage[[All, All, 2]]];
```

```
maBlue = Quantile[Flatten[dcIPTimage[[All, All, 3]]], .98]
```

```
miBlue = Min[dcIPTimage[[All, All, 3]]];
```

```
0.451085
```

```
mightymax = Max[{maBlue, maRed, maGreen}]
```

```
1.72396
```

```
scaleRGB = dcIPTimage * 0;
```

```
scaleRGB[[All, All, 1]] =  $\frac{\text{dcIPTimage}[[\text{All}, \text{All}, 1]] - \text{miRed}}{\text{mightymax} - \text{miRed}};$ 
```

```
scaleRGB[[All, All, 2]] =  $\frac{\text{dcIPTimage}[[\text{All}, \text{All}, 2]] - \text{miGreen}}{\text{mightymax} - \text{miGreen}};$ 
```

```
scaleRGB[[All, All, 3]] =  $\frac{\text{dcIPTimage}[[All, All, 3]] - \text{miBlue}}{\text{mightymax} - \text{miBlue}};$ 
```

```
Show[Graphics[Raster[Apply[{{#1, #2, #3} &, scaleRGB1/γ, {2}}]],  
AspectRatio → Automatic]
```

