With J: Image Processing 2: Color Spaces

Cliff Reiter
Mathematics Department
Lafayette College,
Easton PA 18042, USA
reiterc@lafayette.edu

Using Red-Green-Blue triples to describe colors is common and often quite convenient. However, other color models may be more convenient or appropriate for many types of image analysis and processing. We will discuss conversion between RGB and YUV, YIQ, HSV, HLS, HSI color spaces. We create histograms of the distribution of the color components in those spaces and give examples of how to correct defects and make improvements in various color spaces.

Color Spaces

While there is general agreement about what various color spaces are, many details vary from presentation to presentation [3-6,10]. Some differences account for different display qualities of specific hardware and there is variation in how certain measures should be scaled. While the conversions are one-to-one correspondences for typical values, choices near the edges of one model may correspond to out range values in another and rounding details can have significant impact on the algorithms. Thus, the details of the conversions are a bit of an art and certainly should vary for particular applications.

The RGB color space model is especially convenient for monitor display since the phosphors used on monitors are typically red, green and blue. Figure 1 shows RGB color space envisioned as a cube. Grays appear along the diagonal of the cube running

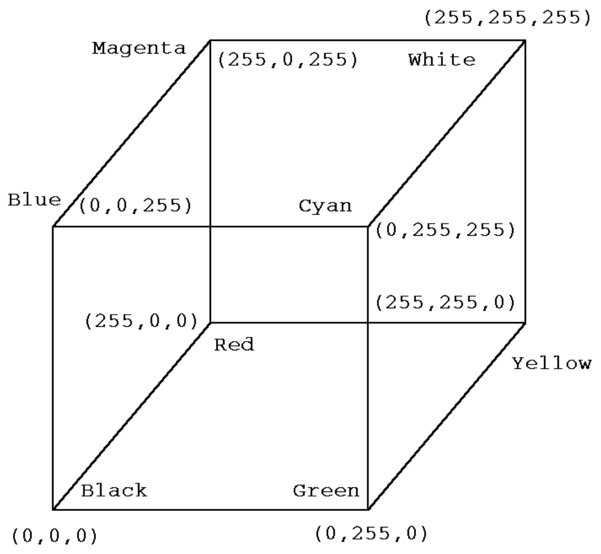

Figure 1. The RGB Cube

from black at one vertex to white at the opposite vertex. We can trace through all the fully saturated hues by running along the edges from red to yellow to green to cyan to blue to magenta and back to red. Notice there is inherent ambiguity. For example, exactly what frequency of red is "red". Different monitors use different red phosphors, so the same RGB triple may appear differently on different monitors. Moreover, it is well known that human perception is affected by a number of factors. Hence, even identical frequency light will be perceived differently in different contexts. From our description of hue, it is clear that hue should measure in some manner the angle around the diagonal for a given point in the RGB cube. How to make that precise is tricky given the jagged nature of the hue path that we described. Indeed, there is not a unique answer to how to accomplish the conversion. Despite the concerns, it is useful to consider various color models in order to manipulate images in a manner best suited to the situation even if we don't know every detail of their eventual display and observation.

The first two color spaces that we will describe are obtained as affine transformations from RGB coordinates. That is, they have the form $T(X)=B+XA$ where X is the input RGB-triple, A is a 3 by 3 matrix and B is a length three vector. The arrays A and B are given by `rgbyuv1` and `rgbyuv2` in Table 1, respectively, for conversion to YUV space.

```
NB. general utilities
to01=:%&255                            NB. convert from [0,255] to [0,1]
to255=: <.@:(255.999&*)                NB. convert from [0,1] to [0,255]
to_int_in=: 1 : '<.@:((m.+0.999)&*"1)'  NB. convert to integer in range

NB. force into range [0,255]
clamp=: 255&<.@(>.&0)

NB. similar YUV conversions also are provided in movie3.ijs
NB. yuv is close to YCbCr which is given here
rgbyuv1=.0.257 0.504  0.098,. _0.148 _0.291 0.439,. 0.439 _0.368 _0.071
rgbyuv2=.16 128 128%255

NB. RGB in [0,255], yuv in [0,1]
RGB_to_yuv=:(rgbyuv2"_ +"1 (+/ . *)&(rgbyuv1%255)) :. yuv_to_RGB
yuv_to_RGB=:clamp@:<.@:((+/ .*)&(255.999*%.rgbyuv1))@:(-&rgbyuv2"1) :. RGB_to_yuv

NB. Versions with YUV in [0,255]
RGB_to_YUV=:to255@:RGB_to_yuv :. YUV_to_RGB
YUV_to_RGB=:yuv_to_RGB@:to01 :. RGB_to_YUV
```

Table 1. Conversion to and from YUV.

The color space YUV is close to the color space YCbCr color space used by PAL TV systems. The Y component is the brightness component. This is what is seen on black and white (PAL) televisions. The U and V components describe the saturation and hue, with the U being bluish and V being a reddish component.

Table 1 shows a J implementation for the conversion between RGB and YUV color spaces. We generally use the following conventions. If the target letters are lower case, then the result is floating type numbers, usually in the range [0,1]. If the target letters are upper case, they correspond to an integer range, usually [0,255]. Thus, `RGB_to_yuv` results in floating point numbers in the range [0,1] while `RGB_to_YUV` yields integers in the range [0,255].

The script *color_space.ijs* [8] contains various utilities for color space conversions, several of which we discuss in this note. That script assumes that the Image3 Addon [9] has been installed and we assume that *color_space.ijs* has been loaded for the example below. We read the image *atlkiln.jpg* that is distributed with the Image3 Addon and see that we can convert every bit of the image to yuv space and return to RGB space with no changes in any pixel value. However, if we convert to YUV integer space and back, roundoff errors may lead to changing a RGB component by up to 2.

```
   fn=:'addons\image3\atkiln.jpg'

   $b=:read_image fn
700 468 3

   b-:yuv_to_RGB RGB_to_yuv b
1

   b-:YUV_to_RGB RGB_to_YUV b
0

   >./,b-YUV_to_RGB RGB_to_YUV b
2
```

The representation in yuv space is more accurate than using YUV, but it requires more memory. Which representation of YUV color space to use depends upon the application.

A simple application of this can be used to create a grayscale version of a color image. In particular, given the image array `b` as above. We can obtain a grayscale version and view it via the following.

```
   gray_b=: 3&#@{."1 RGB_to_YUV b

   load 'addons\image3\view_m.ijs'

   view_image gray_b
468 700
```

That image is shown in Figure 2. Grayscale construction using YUV is usually better than simply averaging of RGB triples via `(+/%#)"1` since the weight associated with green is larger in the YUV

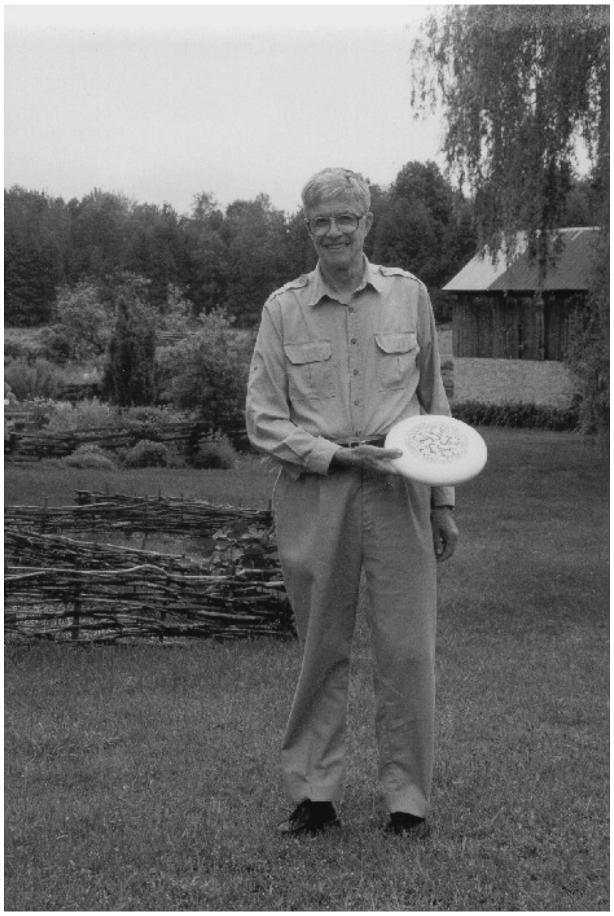

Figure 2. Gray Scale via YUV

model and that better corresponds to human perception.

A very similar color space is the YIQ color space used by NTSC color TV. It is slightly simpler. Indeed, it corresponds to multiplication by the following matrix.

```
   rgbyiq0
0.299  0.596  0.212
0.587 _0.274 _0.523
0.114 _0.322  0.311
```

A conversion to YIQ and its inverse is defined in the script *color_space.ijs*. Again, Y is the brightness which is used for the black and white broadcast signal. Although the I and Q components determine the saturation and hue, they are not in the range [0,1]. In particular, they may well be negative numbers. Thus we only include only the floating type conversion `RGB_to_yiq` and its inverse.

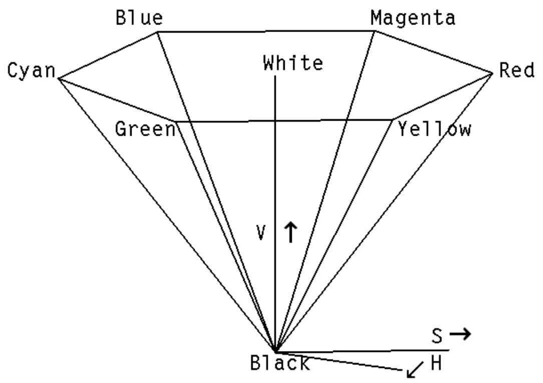

Figure 3. The HSV Hexcone

Previous remarks make it clear that saturation and hue are important features of a specific color. HSV is a simple color space that contains hue as a component. Figure 3 shows the HSV Hexcone. Notice the central axis gives V, which is a value corresponding to brightness. Hue, H, is the angle around the central axis and saturation, S, measures distance from the axis. Thus, a high saturation corresponds to a pure color, like red. A medium saturation might result in a Burgundy color (if V is low) or pinkish color (if V is high). A low saturation corresponds to a color close to gray, although it could appear anywhere along the gray axis (running from black to white). Implementation of conversions to/from HSV may be found in the script and mathematical formulas may be found in the references. We will not repeat the formulas for the conversion except to say that V is the minimum of the RGB components. Taking V for the brightness is not especially natural; however, if interest is upon hue and/or saturation, this is a color space conversion involving hue that is easily implemented. Also, when converting hues to integers, we usually use the range [0,359] to correspond to the nearest degree of the hue. Unlike other components, hue wraps around modulo 360 so that hues close to 0 are similar to those close to 360.

Another color space is HSL. It is often viewed as a double hex cone and the lightness value, L, may be computed as the average of the minimum and maximum of the RGB components. This gives what is an often a better value for brightness than HSV, while remaining reasonably simple. The H component is hue and the S component is saturation.

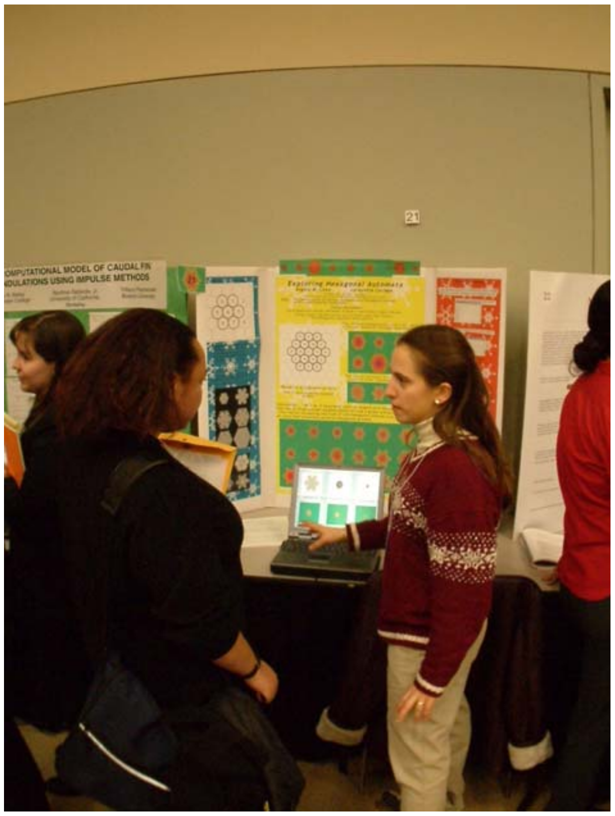
Figure 4. A Student Presenting a Poster

The HSI color space is a space which has hue, saturation and intensity as its components. It is usually viewed as resulting from a rotation of the diagonal of the RGB cube into the I direction, followed by suitable conversion to polar coordinates for the hue and saturation components. In the literature, the descriptions of this color space varies quite a lot, and we chose to rescale the approach of [5] so that the conversion of the unit RGB cube fit nicely into [0,1] ranges for each of the HSI coordinates. The intensity is then given as the average of the intensities of the RGB components. Compared to the other color spaces that include hue, the inverse of this conversion is more difficult to implement. Like the other spaces with a hue component, the components of this space are abstract, rather than based upon perception. However, the components are very convenient for thinking about color.

We now turn to considering how to get some basic information about an image from various color spaces.

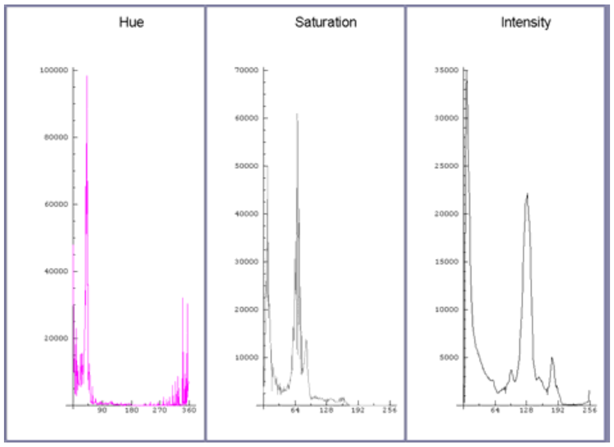

Figure 5. Histogram of HSI in an Image

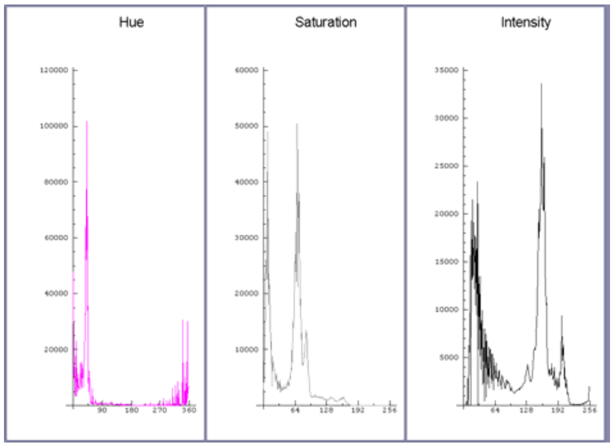

Figure 6. Histogram after Gamma Correction

Color Plane Histograms

Table 2 shows a conjunction `cs_hist` that is used to build histograms of the frequency that integer values appear in the color planes of various color spaces. The function `freq_tab` actually computes the required frequency tables. While the conjunction is not trivial, there are three histograms to create for each color space and several color spaces. The conjunction takes arguments corresponding to the color space while a loop creates the three histograms making the definition remarkably compact. The histogram function for each color space requires just one additional line. Table 2 also shows the one-line definition for the RGB-histogram function.

Figure 4 is a digital image that was taken under adverse conditions. It is photograph of a research student, Angela Coxe, presenting a poster of her work [1-2]. So as not to disturb her conversation with Professor Chawne Kimber, no flash was used. The

```
NB. create a frequency table
NB. [upper bound] freq_tab data
freq_tab=: 3 : 0
256 freq_tab y.
:
<:#/.~(i.x.),,y.
)

NB. Left: [range cutoff] (discretization levels [saturation position])
NB. Right: (labels(3),colors(3) as a boxed array)
NB. color space histogram builder conjunction
cs_hist=:2 : 0
_ _ _ m. cs_hist n. y.
:
labels=.3{.n.
colors=.3}.n.
y.=.,"_1 (i.<:#$y.)|:y.
pd 'reset'
pd 'new'
pd 'backcolor 128 128 160'
for_k. i. 3 do.
  pd 'new ',": 5 5 325 990+330 0 0 0*k
  pd 'type line;'
  pd 'color ',":>k{colors
  pd 'xrange 0 ',":n=.k{m.
  pd 'xticpos ',":n*(i.%<:)5
  pd 'title ',>k{labels
  if. (k=0) *. 3<#m. do.
     data=.(0~:(3{m.){y.)#k{y.
     else.
     data=.k{y.
     end.
  fdata=.(k{m.) freq_tab data
  if. _>k{x. do. fdata=.fdata <. k{x. end.
  pd fdata
end.
pd 'show'
wd 'pmovex 0 0 1024 768;ptop 0;pshow'
)

NB. all histogram functions take RGB [0,255] arrays
NB. as their right argument
NB. optional left argument is 3 bounds on the output frequency table heights

rgb_hist=:256 256 256 cs_hist ('Red';'Green';'Blue';255 0 0;0 255 0;0 0 255)
```

Table 2. Creation of Histogram Functions.

room was very crowded and taking the photo required a lucky break in the crowd and use of a wide-angle lens. A sport sequence of images was taken trying to capture her interaction with visitors to the poster session. The lighting was not very bright. The image seems too dark, especially if shown on a standard monitor. There is more wrong with this image than brightness (e.g., the barrel distortion caused by the wide-angle lens). However, the problem of images being too dark for display on a web page (on certain monitors) is a common problem. Notice that the intensity values in the histogram shown in Figure 5 do

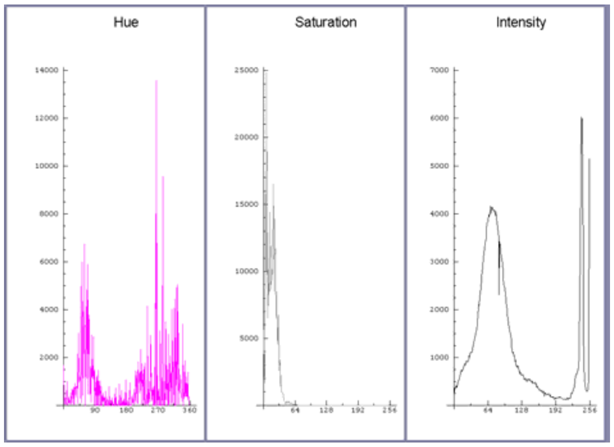

Figure 7. HSI Histogram of *atkiln.jpg*

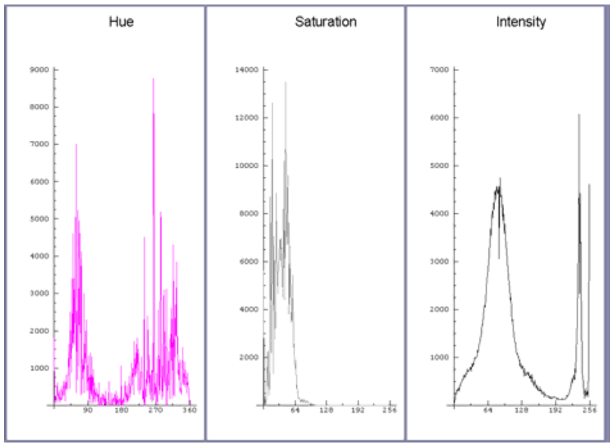

Figure 9. HSI Histogram of the Modified Image

not seem to fully utilize the upper portion of the [0,255] range of intensity values. It is not surprising that the image seems dark. We consider one method for brightening the image in the next section.

Intensity Correction

Since human perception of brightness is nonlinear, a nonlinear function is often used to map brightness values to image components. Often an exponential map with a constant in the exponent in the map is used, and the constant is often denoted as gamma. The use of an incorrect gamma would result in an image that is too bright or dark. Thus, application of a function `^&c` to the brightness component, where `c` is a constant, is commonly known as gamma correction. For this image, some experimentation leads to the choice of 0.7 for gamma. The gamma correction can be accomplished via the following J expressions. We read the original image as the array `w` and, after conversion to HSI space, we separate the component

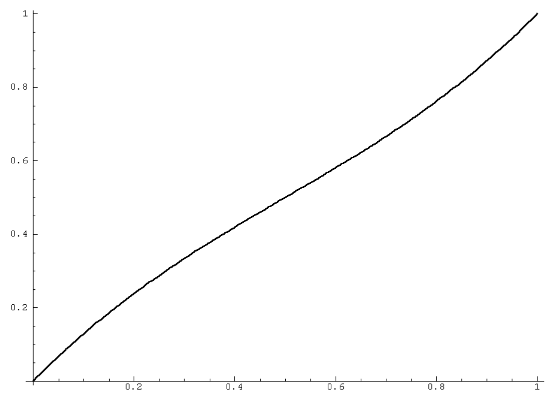

Figure 8. The Function 0.8 tw_fct

planes into the matrices `h`, `s` and `i`. The reconstructed image array, `W`, represents the gamma-enhanced image.

```
   $w=:read_image 'image1.jpg'
1280 960 3

   'h s i'=:0 1|: RGB_to_hsi w

   W=:hsi_to_RGB 0|:h,s,:i^0.7
```

Figure 6 shows the HSI histogram for `W`, the enhanced image. Notice the shift upward of the brightness components.

This image could be improved in several ways. In order to illustrate gamma correction, we limited ourselves to considering only this one change. We used our eye to determine a suitable level of gamma correction. Several other values of gamma were also considered. Additionally, other strategies for improving the brightness component were tried. This included linear shifts and uniformly spred of brightness changes onto the image. (Which gives a type of automated maximal contrast that can be especially useful for scientific images where one is trying to observe a vague feature. See `con_exp` in *color_space.ijs*.) However, those changes were not as pleasing to the eye. When modifying a photographic image for informal display, using one's eye is fine. However, color shifts, dilation and erosion of features are common side effects of image processing, so great care is required when modifying images for subsequent scientific analysis.

Saturation Improvement

We now take a further look at the *atkiln.jpg* image shown in grayscale in Figure 2. Since that image is

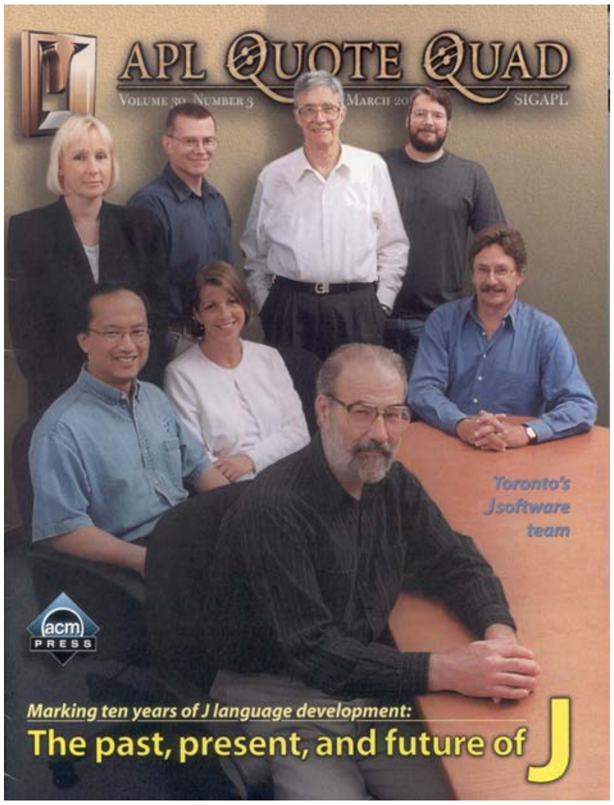

Figure 10. A Scanned Image

part of the Image3 Addon, you can duplicate the experiments that we will describe and explore your own ideas. We assume *color_space.ijs* and *view_m.ijs* have been loaded. Then we can view the original color image via the following expressions.

```
   fn=:'addons\image3\atkiln.jpg'

   $b=:read_image fn
700 468 3

   view_image b
468 700
```

Next, we create an HSI histogram, which is shown in Figure 7.

```
   hsi_hist b
```

Notice in the histograms that there is a nice distribution of hues, but the saturation seems quite low with virtually all the values occurring in the lowest quarter. If you have actually viewed the color image, you may observe that it seems somewhat washed out.

```
   'h s i'=:0 1|: RGB_to_hsi b

   s2=:s^0.7

   c=: hsi_to_RGB 0|:h,s2,:i

   view_image c
700 468 3
```

The change in saturation is the most significant change we want to make to the image. It might be considered too much of a change, but we consider it pleasing. Additionally, the intensity is bimodal, with some very bright values (look at the sky), but most of the values are in the lower half. We would like to take more advantage of the intensity values between the peaks. Consider the adverb `tw_fct` that is defined in *color_space.ijs*. It creates a function that is 0 at 0, 0.5 at 0.5, 1 at 1 and has the slope specified by its left argument at 0.5. Figure 8 shows a plot of `0.8 tw_fct` on [0,1]. It gently moves values toward 0.5. It might be better to use a function more closely designed for the histogram, perhaps twisting at a point higher than 0.5. However, `tw_fct` was available and makes the point that specialized functions can be constructed to create specialized modifications. We apply this function to the intensity component and create a final modified image.

```
   i2=:0.8 tw_fct i

   d=:hsi_to_RGB 0|:h,s2,:i2

   view_image d
468 700
```

It is worth replicating the experiments above to see the impact that these small changes make on the appearance of the color image. Figure 9 shows the HSI histogram for `d`, the modified image. Notice the raised saturation levels and slight spred of the intensity values.

Color Scanning Defects

We mentioned in [7] that correcting defects of color scanned images is significantly more difficult than dealing with grayscale images. The filtering techniques from that paper can be applied to individual components in a color space, but that should be done with an alertness to the risks. In particular, color shifting is a frequent problem.

We consider an image of the cover of Quote Quad that we wanted to use on a web page to highlight scripts and materials associated with Quote Quad writings. The cover is beautiful, but the scanned image was disappointing. We wanted to improve upon the raw scan. Figure 10 shows the scanned image while Figure 11 shows a zoom into four portions of the scan:

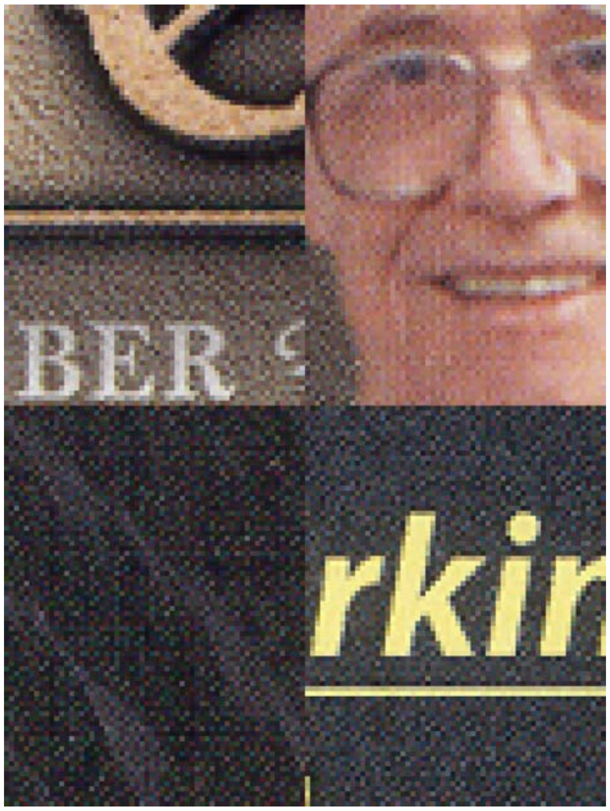

Figure 11. A Zoom into the Image

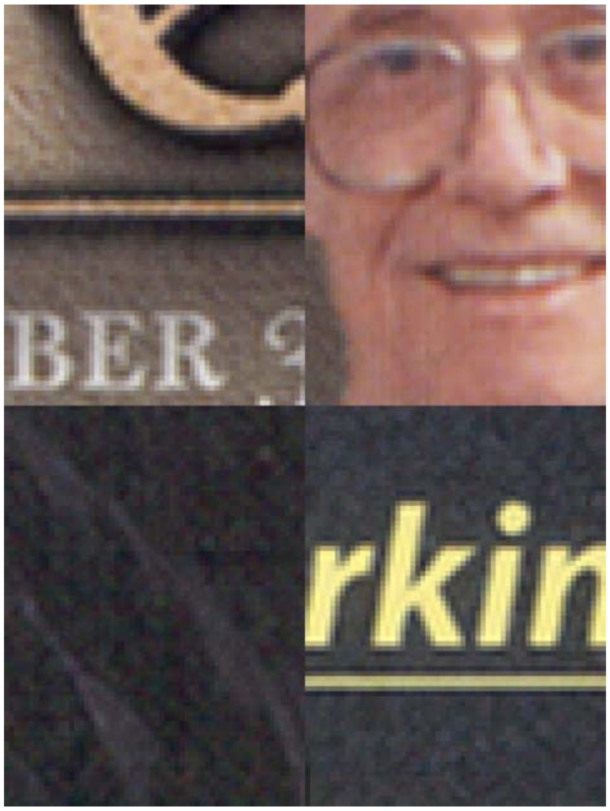

Figure 12. A Zoom into the Modified Image

letters and texture near the top, a face, a shirt, and text and carpet. Vertical and horizontal striping is visible and so are what appear to be random noise pixels. However, defect correction is quite delicate for this image since the wall texture is a real feature of the image and the text edges can easily be distorted by processing.

Most of the filters discussed in [7] do not extend in a natural way to color triplets. One exception is the maximum likelihood filter. It does a reasonable job of smoothing this image, but the results are relatively posterized. We tried applying many of the filters of the type discussed in [7] to various combinations of the color planes in various color spaces. We thought that applying the Savitsky-Golay filter of degree 2 and width 5 on the three RGB planes did the best job. Zooms using that approach are shown in Figure 12. However, several other choices were quite close. These include the following.

- Savitsky-Golay filter on S and I in HSI space
- A 3 by 3 median filter on RGB planes,
- A 3 by 3 median filter on I in HSI

We have read that the blue sensors on digital cameras may be more susceptible to noise than the other sensors. Thus, it may make sense to smooth the blue component more than other components. We have not tried that on our scanned image.

However, using the Savitsky-Golay filter on H, S and I in HSI space on the scanned image gave horrible color shifts. Indeed, it makes the people look unreal. Figure 13 shows the HSI histogram for the original image while Figure 14 shows the HSI histogram after the Savitsky-Golay filter has been applied on the HSI color planes. Notice the dramatic change in the hue component. Thus, great care needs to be taken about exactly which components are filtered and which are not filtered.

Conclusion

We see that it is possible to represent color images in a variety of color spaces. We can extract information, such as gray levels, from these spaces or analyze the image further. The distribution of various components can be visualized using histograms of the components. This makes it easy to see components that could be improved by applying simple functions

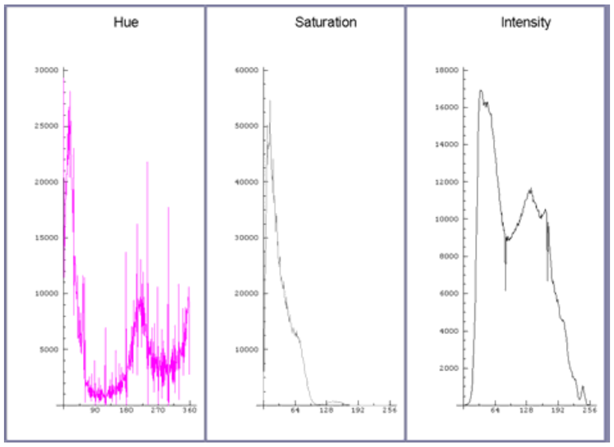

Figure 13. HSI Histogram of Scanned Image

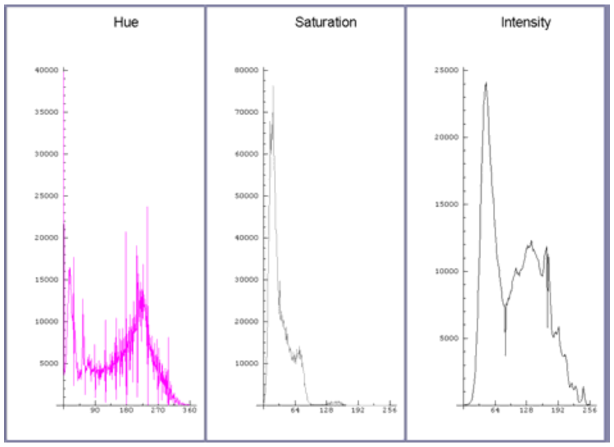

Figure 14. HSI Histogram of Bad Modification

to those components. Moreover, filters, such as those designed to smooth scanning defects can be applied to appropriate components of color spaces.

References

[1] A. M. Coxe and C. A. Reiter, Boolean Hexagonal Automata, *Vector*, 19 3 (2003) 113-121.

[2] A. M. Coxe and C. A. Reiter, Fuzzy Hexagonal Automata and Snowflakes, *Computers & Graphics*, to appear.

[3] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics, Principles and Practice, 2nd edition*, Addison-Wesley Publishing Company, 1990.

[4] R. C. Gonzalez, R. E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Company, 1992.

[5] S. Harrington, *Computer Graphics*, A Programming Approach, 2nd ed, McGraw Hill Book Co., 1987.

[6] K. Jacks, *Video Demystified*, *3rd edition*, LLH Technology, 2001.

[7] C. Reiter, With J: Image Processing 1: Smoothing Filters, submitted to *APL Quote Quad*.

[8] C. A. Reiter, *color_space.ijs* script, *http://www.lafayette.edu/~reiterc/j/withj/index.html*.

[9] Z. X. Reiter, C. A. Reiter, Image 3 Addon, *http://www.jsoftware.com*.

[10] D. Wilson, The [almost Definitive] FOURCC Definition List, RGB/YUV Conversion,

http://www.fourcc.org/fccyvrgb.htm.