```
L2=:L1-<./,L1
L3=:<.L2*255%>./,L2
view_image BW256;3 spix L3
```

Notice that the Laplacian also marks the edges with a kind of up down pattern that can be seen in Figure 8.3.2. Adding the Laplacian to the original image data smooths the noisy parts and can sometimes clarify edges. The expression below compares the original image with the image with its Laplacian added to it.

```
view_image BW256;3 spix Y,.Y+round L1
```

The result is shown in Figure 8.3.3.

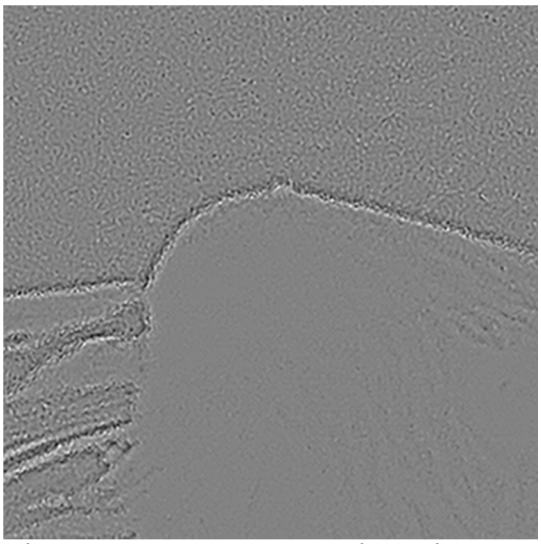

Figure 8.3.2 The Laplacian Filter

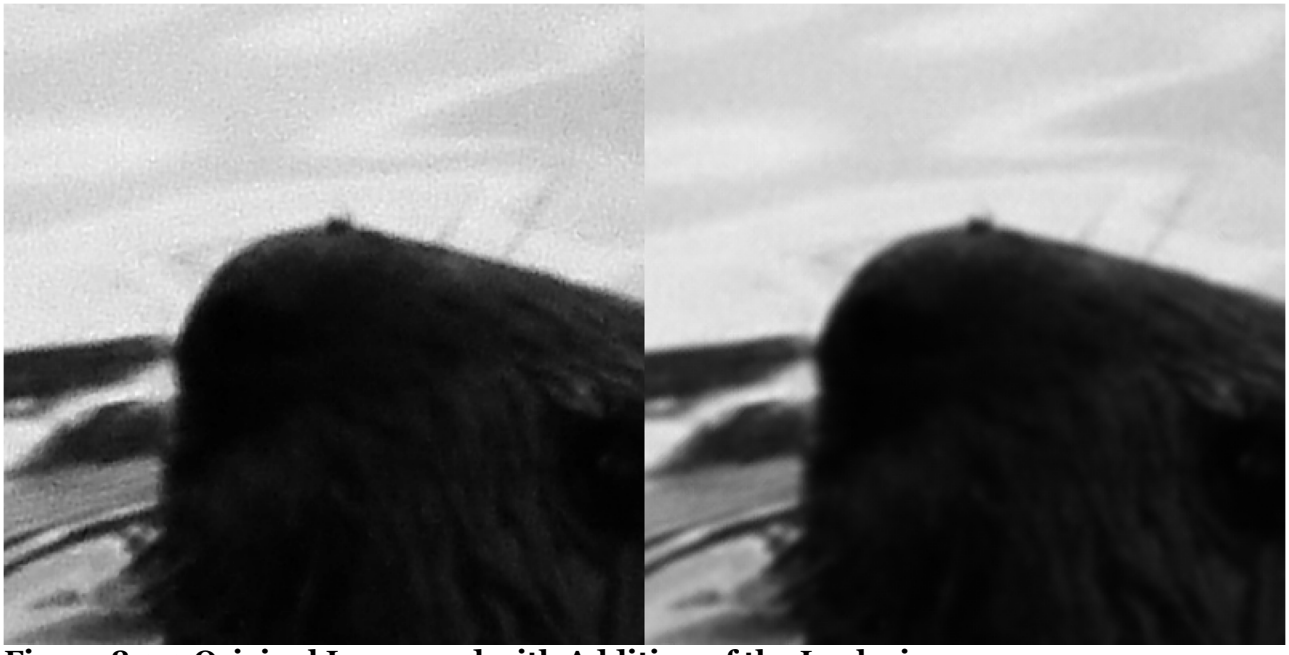

Figure 8.3.3 Original Image and with Addition of the Laplacian

8.4 Experiment: Color Spaces

So far we have limited our attention to the RGB color model that we explicitly discussed in Section 5.1. The RGB model is especially convenient for monitor and LCD displays since the phosphors (respectively lcds) are usually red, green and blue. Figure 8.4.1 repeats the geometric view of the RGB model. Other color models may be better suited to printing, image compression, or other types of image manipulation. In looking at the RGB model, we can imagine moving along the central diagonal from black (0 0 0) to white (255 255 255) with the diagonal being intermediate grayscales. As we move from corner to corner along that diagonal the brightness (or intensity) increases. The angle around the diagonal gives the hue, which basically determines the color. The saturation measures how pure (or muddy) the color is, with colors on the surface of the cube being fully saturated, and the diagonal being totally unsaturated (grayscale). All these descriptions are imprecise and different color models include various choices for the coordinates and the details of how they are implemented varies considerably.

We will briefly describe three models that are useful and illustrative. Reiter (2004b) gives some further examples. One of the simplest ways to modify RGB image data is to transform the data according to an affine transformation. That is, a map of the form $T(x) = Ax + B$ where A is a 3 by 3 matrix and B and x are vectors with 3 coordinates. The color space YUV is close to the color space used for color TV. In particular, the Y component is brightness which is what is seen on black and white (PAL) television sets. The U and V components together describe hue and saturation with U being bluish and V being reddish. If we load the script *color_space.ijs*, we will have conversion utilities defined. We will use the convention that upper case letters are used for a color space components if they are integers between 0 and 255. If they are real values between 0 and 1, we use lower case letters.

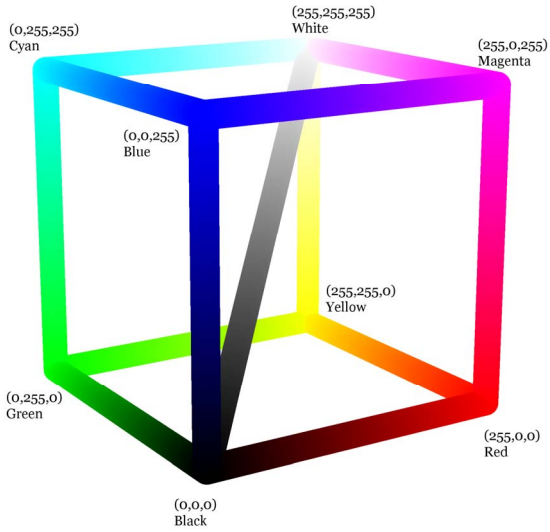

Figure 8.4.1 The RGB Color Model

```
   load '~addons/media/imagekit/color_space.ijs'

   RGB_to_yuv 0 0 0                conversion function applied
0.0627451 0.501961 0.501961

   RGB_to_yuv 255 255 255
0.921745 0.501961 0.501961

   RGB_to_yuv 0 255 0
0.566745 0.210961 0.133961

   RGB_to_YUV 0 255 0
145 54 34
```

So we see that the RGB triple 255 255 255 is near full brightness (Y) and neutral in U and V. Next we consider a sample image that can be improved various ways. The image is of Chawne Kimber discussing Angela Coxe's poster with her at a mathematics meeting. It was taken under adverse conditions. Below we see that if we use yuv coordinates, an example image can be exactly recovered, but if we use YUV coordinates, a small amount of change occurs.

```
   fn=: jpath '~addons/media/imagekit/poster.jpg'

   $b=:read_image fn
1280 960 3

   b-: yuv_to_RGB RGB_to_yuv b          completely reversible
1

   >./|,suub-YUV_to_RGB RGB_to_YUV b    almost reversible
3
```

Since this conversion is rapid and give a more sophisticated sense of brightness than averaging RGB triples, it can be used as a method for obtaining a grayscale image.

```
  'Y U V'=:0 1|: RGB_to_YUV b

   view_image 3#"0 Y
```

Next, we turn to considering the HSV model. Figure 8.4.2 shows a geometric model for HSV space. This space can be thought of as a hexagonal cone with black at the vertex and white at the center of the

hexagonal face. Saturation and hue are determined by a sort of polar coordinate conversion around the diagonal. The value V, which measures brightness, is determined by the maximum of the RGB components. It is not necessary, but traditional to allow the integer version of hues to vary from 0 to 359, corresponding to degrees when using the HSV model. Since value V is not as natural a parameter as intensity, often one prefers to use HSI space where the parameters are hue, saturation and intensity. Details of the conversions vary in the literature, although the basic idea is to provide parameters that give color as one parameter (hue), the purity of the color as another (saturation), and intensity as the last. We create a histogram of these parameters for our sample image above; it is shown in Figure 8.4.3. Notice that the distribution of intensities has few entries above 196. This suggests that the image is slightly underexposed. A standard way to adjust that might be to replace the intensity `i` by something such as `i^0.8`; however, here we will try a linear adjustment of intensity.

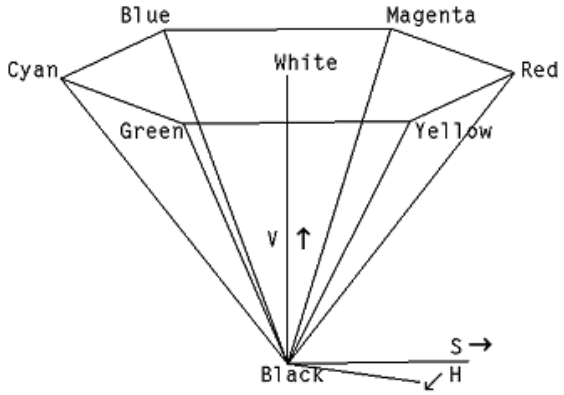

Figure 8.4.2 The HSV Color Model

```
'h s i'=:0 1|: RGB_to_hsi b
view_image c=:hsi_to_RGB 0|:h,s,:(255%196)*i
hsi_hist c
```

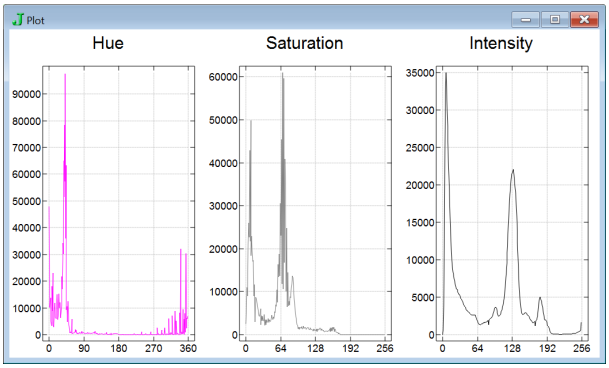

Figure 8.4.3 HSI Histograms

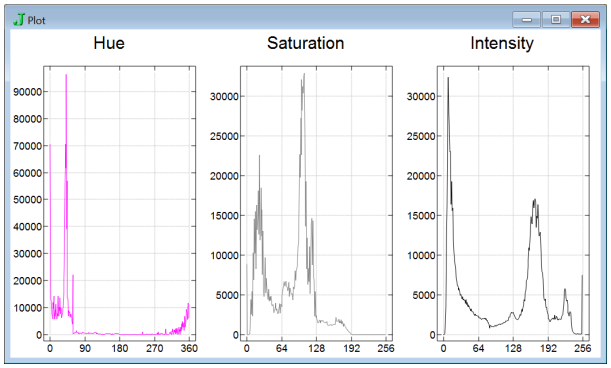

Figure 8.4.4 Modified HSI Histograms

A look at the histogram shows intensities spread more uniformly through their range. While the saturations in the image are probably correct, we also observe that they are low. We can make the image more vibrant by raising the saturation levels too.

```
view_image c=:hsi_to_RGB 0|:h,(s^0.7),:(255%196)*i
hsi_hist c
```

The resulting histogram showing the raised saturation and spread intensities is shown in Figure 8.4.4. The resulting image is shown in Figure 8.4.5.

Now load an image of your choice as `b` and experiment as follows with changes in intensity and saturation. Use your own filename for `fn`.

```
fn=:'your_image.jpg'
view_image b=:read_image fn
```

```
   'h s i'=:0 1|: RGB_to_hsi b
   view_image hsi_to_RGB 0|:h,s,:i^0.7
   view_image hsi_to_RGB 0|:h,s,:i^1.3
   view_image hsi_to_RGB 0|:h,(s^0.7),:i
   view_image hsi_to_RGB 0|:h,(s^1.3),:i
   view_image hsi_to_RGB 0|:h,(s^0.7),:i^0.7
```

Do any of these improve the image? What happens if you adjust hue in a similar manner?

Figure 8.4.5 The Modified Image

8.5 Rotation, Tilt and Barrel Distortion

Images can also have various sorts of distortions that can be corrected or mitigated with mathematical transformations. Each of the distortions we will discuss may be corrected using a GUI interface using functions from the script *~addons/media/imagekit/transform_m.ijs*. Additional advice may be found at Reiter (2003a) and by looking at the help for the `transform_image` form. However, here we briefly discuss the mathematics and refer to functions from that script.

Consider a digital image where the horizon is not horizontal. This can be corrected using a rotation. For example, Figure 8.5.1 shows the *keys.jpg* image with two points near the horizon marked by clicking with the shift key. The image was tilted because the camera was set on a log and a timer used.

```
   load '~addons/media/imagekit/transform_m.ijs'
   transform_image jpath,'~addons/graphics/fvj4/keys.jpg'
450 300
   SEL_pts
59 118
341  97
   get_sel_pts_angle
3 : '12 o. j./(**@{.)-/SEL_pts_mkit_'

   ]X=:get_sel_pts_angle ''
_0.0743309
```