Adventures in HSV Space

Darrin Cardani
dcardani@buena.com

Abstract: Describes how to convert RGB images into HSV space and why you would want to do so. It also describes some new techniques for choosing parameters in HSV space, and presents code for some interesting image and video filters that work in HSV space, including cleaning up skin tones, creating cartoon-like effects, and manipulating individual colors in a complex scene.

What is HSV Space?

Most operating systems, image processing programs and texts treat images as collections of pixels comprised of red, green and blue values. This is very convenient for display purposes, since computer monitors output color by combining different amounts of red, green and blue. However, most users don't think of color in these terms. They tend to think about color the same way they perceive it - in terms of hue (the English name we give colors, like "reddish" or "greenish"), purity (pastels are "washed out", saturated colors are "vibrant"), and brightness (a stop sign is "bright" red, a glass of wine is "dark" red). So scientists came up with what they call *perceptual* color spaces.

A perceptual color space represents color in terms that non-technical people understand. There are many perceptual color spaces, including the PANTONE® Color System, the Munsell Color System, HSV (Hue, Saturation, Value) space, HLS (Hue, Lightness , Saturation) space, and countless others. The one with which most Mac users are familiar is Hue, Saturation and Value space. It can be seen in Apple's HSV Color Picker, included with every Mac since at least System 6.[Apple86] (See figure 1.)

(Note: HSV space is sometimes referred to as HSI for hue, saturation and intensity, or HSB for hue, saturation, and brightness.)

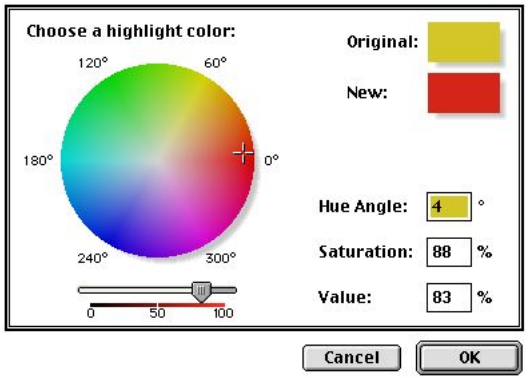

Figure 1: Apple's HSV Color Picker

Conceptually, the HSV color space is a cone. Viewed from the circular side of the cone, the hues are represented by the angle of each color in the cone relative to the 0° line, which is traditionally assigned to be red. The saturation is represented as the distance from the center of the circle. Highly saturated colors are on the outer edge of the cone, whereas gray tones (which have no saturation) are at the very center. The brightness is determined by the colors vertical position in the cone. At the pointy end of the cone, there is no brightness, so all colors are black. At the fat end of the cone are the brightest colors.

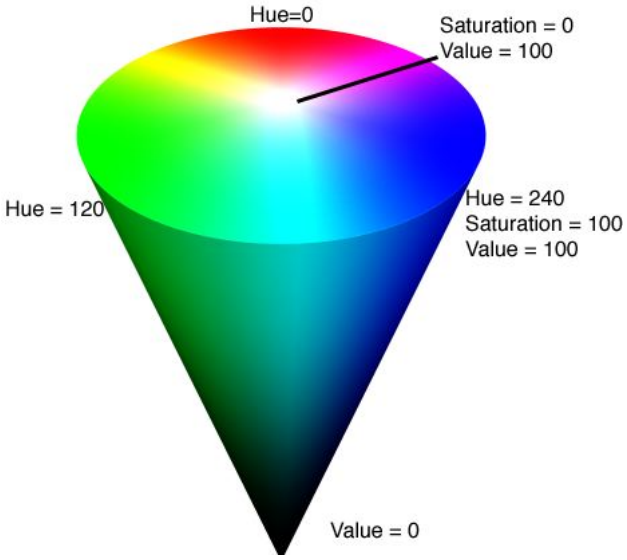

Figure 2: The HSV Cone

Why (or When) Should I Use HSV Space?

So why should you use the HSV color space if the OS itself needs colors to be in RGB space to display them? There are two main times RGB is inconvenient. The first is when you want to get a color from a typical user. Since most users don't understand the nuances of RGB, you need to present them with a way to pick colors which they can understand. Apple's HSV color picker does this very well. The second time you want to use HSV space is when you have to match colors, or programatically determine if one color is similar to another color. Let's look at why RGB comparisons are difficult.

The RGB color space is conceptually a cube with one axis representing red, one representing green, and one representing blue, as shown below.

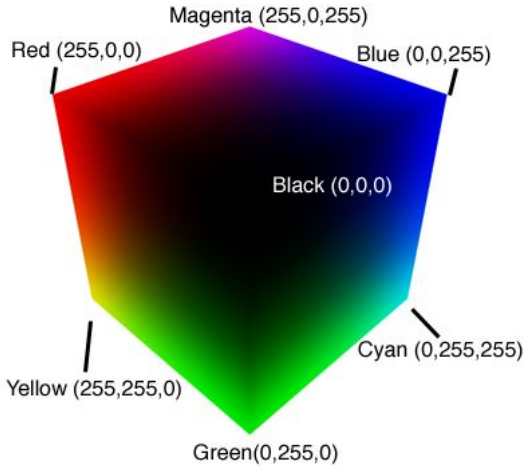

Figure 3a: The black corner (0,0,0) of the RGB Cube

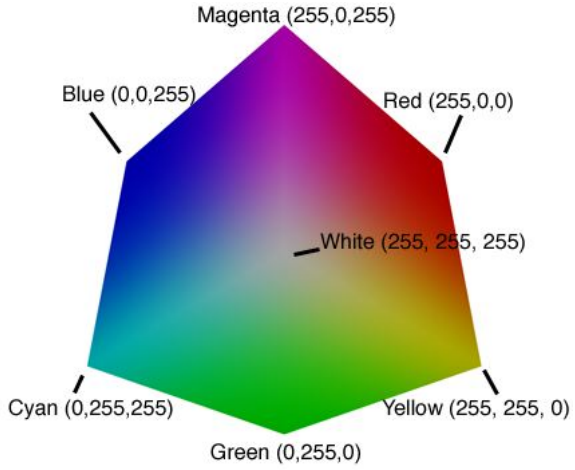

Figure 3b: The white corner (255,255,255) of the RGB Cube

As you can see, where the axes meet at (0,0,0), we have black, and at (255,255,255) (or (1.0,1.0,1.0) if you prefer), we have white. How do we tell if a color in an image is close to a color we picked? We could take the Euclidian distance between the two colors and see if it's less than a "similarity" parameter. That sounds reasonable, but lets look at how this works from a perceptual standpoint. Let's say you allow the user to set a "similarity" threshold for matching all colors that are similar to a chosen color. In RGB space, all points less than or equal to the "similarity" distance from the chosen color form a sphere inside the RGB cube. The user probably thinks of matching a color by choosing "all the bluish tones", or some similar perceptual way. But the sphere we get in the RGB cube doesn't

include many of the values that would meet this criteria, and it does include many that probably wouldn't. You can shrink the sphere to remove those which don't match, but there is no obvious transform you can perform to get more of those colors which don't match, but that you want to include.

Another way to match colors in RGB space would be to pick a range of red, green, and blue in which colors must fall. So now you've cut out a smaller cube from the RGB cube. If you want to, say, match purplish colors, you run into a similar problem as with using a Euclidian distance. Since the purplish colors run along the diagonal between the red and blue axes, you either end up including a bunch of points you don't want, not including points you do want, or doing a whole bunch of math for what should be an easy problem.

Now let's see how you would make such a match in HSV space. The user picks a color, and sets a similarity control. You convert the color to HSV space, if it isn't already in HSV space. Now you can see if other colors match the chosen one based on their hue angle. If the user wants only aqua colors, they will likely choose a color with a hue angle of 180°. Colors that match have hues of roughly 165° to 195°. Using those parameters you cut a pie slice out of the HSV cone. The user probably doesn't want very dark cyans, since colors that are close to black often appear to be black. And she probably doesn't want colors that are too close to gray or white, either. So we can limit the colors that match to not only be within a given hue range, but also a given saturation and value range. Brief experimentation with the HSV color picker suggests that a saturation of 25% or greater, and a value of 50% or greater gets us a nice range of colors that most users would probably qualify as "close to aqua". So allowing the user to choose a range for the hue, a range for the saturation, and a range for the value gets us reasonable results. This is easier for the user, and as you'll see below, fairly easy for the programmer, too.

How Do I Convert from RGB to HSV Space?

Foley, van Dam, et al. describe a fairly straightforward way of converting from RGB space to HSV.[Foley90] Here is a C translation of their pseudocode which I've used in production software. It produces 16 bit signed integers in the range 0-360 for hue, 0-255 for saturation, and 0-255 for value. Values can be scaled to be in any range that's convenient for your application, and the function can be rewritten to use only integers if this version proves to be too slow.

```
void RGBtoHSV (RGBColor* prgbcIn, short*
piOutHue, short* piOutSaturation, short*
piOutValue)
{
   short  iMax, iMin;

   // Calculate the value component
   if (prgbcIn->red > prgbcIn->green) {
      iMax = prgbcIn->red;
      iMin = prgbcIn->green;
   } else {
      iMin = prgbcIn->red;
      iMax = prgbcIn->green;
   }

   if (prgbcIn->blue > iMax)
      iMax = prgbcIn->blue;

   if (prgbcIn->blue < iMin)
      iMin = prgbcIn->blue;

   *piOutValue = iMax;

   // Calculate the saturation component
   if (iMax != 0) {
      *piOutSaturation = 255 *
         (iMax - iMin) / iMax;
   } else {
      *piOutSaturation = 0;
   }

   // Calculate the hue
   if (*piOutSaturation == 0) {
      *piOutHue = kHueUndefined;
   } else {
      float  fHue;
      float  fDelta;
      fDelta = iMax - iMin;

      if (prgbcIn->red == iMax) {
         fHue = (float)(prgbcIn->green -
            prgbcIn->blue) / fDelta;
      } else if (prgbcIn->green == iMax) {
         fHue = 2.0 + (prgbcIn->blue -
            prgbcIn->red) / fDelta;
```

```
        } else {
            fHue = 4.0 + (prgbcIn->red -
                prgbcIn->green) / fDelta;
        }

        fHue *= 60.0;

        if (fHue < 0)
            fHue += 360;

        *piOutHue = (short)fHue;
    }
}
```

"That's great for color matching," you say, "but once I've converted the image and matched the colors, the image is in HSV space, which my OS doesn't understand!" You simply need the reverse transformation. Here's the code to get back into RGB from HSV space (also translated from [Foley90]):

```
void HSVtoRGB (short iInHue, short
iInSaturation, short iInValue, short* piRed,
short* piGreen, short* piBlue)
{
    if (iInSaturation == 0) {
        *piRed = iInValue;
        *piGreen = iInValue;
        *piBlue = iInValue;
    } else {
        float  fHue, fValue, fSaturation;
        SInt32 i;
        float  f;
        float  p,q,t;

        if (iInHue == 360)
            iInHue = 0;

        fHue = (float)iInHue / 60;
        i = fHue;
        f = fHue - (float)i;

        fValue = (float)iInValue / 255;
        fSaturation = (float)iInSaturation /
            255;

        p = fValue * (1.0 - fSaturation);
        q = fValue * (1.0 - (fSaturation *
            f));
        t = fValue * (1.0 - (fSaturation *
            (1.0 - f)));

        switch (i) {
            case 0:
                *piRed = fValue * 255;
                *piGreen = t * 255;
                *piBlue = p * 255;
            break;

            case 1:
                *piRed = q * 255;
                *piGreen = fValue * 255;
                *piBlue = p * 255;
            break;

            case 2:
                *piRed = p * 255;
                *piGreen = fValue * 255;
                *piBlue = t * 255;
            break;

            case 3:
                *piRed = p * 255;
                *piGreen = q * 255;
                *piBlue = fValue * 255;
            break;

            case 4:
                *piRed = t * 255;
                *piGreen = p * 255;
                *piBlue = fValue * 255;
            break;

            case 5:
                *piRed = fValue * 255;
                *piGreen = p * 255;
                *piBlue = q * 255;
            break;
        }
    }
}
```

What Can I Do With It?

Now that you know how to convert between RGB and HSV color spaces, let's look at some of the useful things you can do with it.

Color Picking

As stated previously, users tend to have an easier time choosing colors in HSV space. Apple's HSV color picker is pretty good. Adobe also has two interesting HSV color pickers in Premiere and Photoshop that are also worth noting.

In Photoshop, they decided to put the hue in a control by itself, rather than the value. The user chooses a hue on the color strip in the center of the dialog box. Then the large area to the left of it displays the hue with its value varying from

0% at the bottom to 100% at the top, and its saturation varying from 0% at the left to 100% at the right.[Adobe2000]

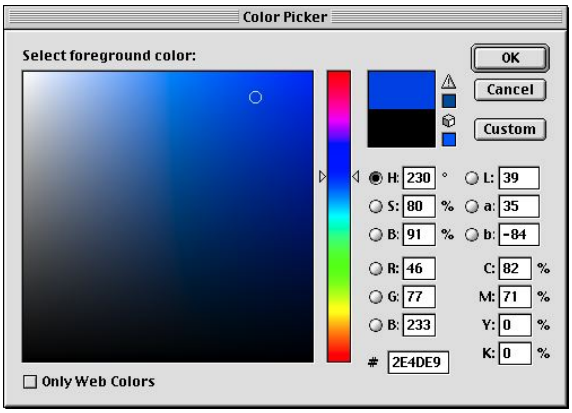

Figure 4: Adobe Photoshop Color Picker

Premiere has a conceptually more complicated color picker, although it lays out the colors quite nicely from a perceptual standpoint. For the top half of the color picker, the value is set to 100%. The hue varies from left to right, and the saturation is varied from 0% to 100% from top to the middle of the image. For the bottom half of the color picker, the value varies from 100% down to 0% from the middle to bottom but the saturation remains at 100%. On the left, there is also a gray scale so you can choose pure gray tones.[Adobe94] This is essentially the exterior of the HSV cone mapped onto a rectangle.

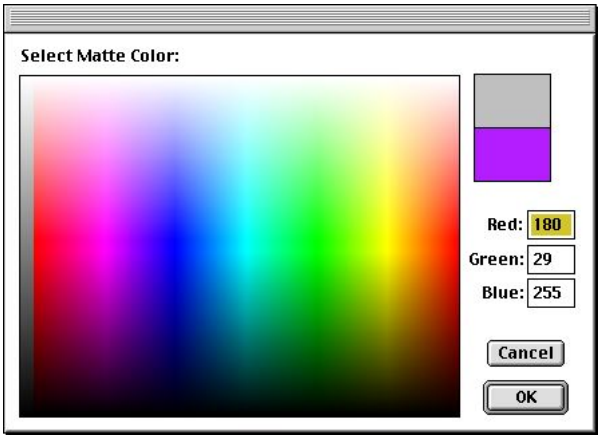

Figure 5: Adobe Premiere Color Picker

New Ways of Choosing Color Ranges in HSV Space

Since non-technical users have an easier time of selecting color in a perceptual space, it stands to reason that they would have an easier time picking a color range in a perceptual space, as well. I'd like to look at some possible ways to make choosing ranges of colors easier by doing the picking in HSV Space.

Round Range Picker

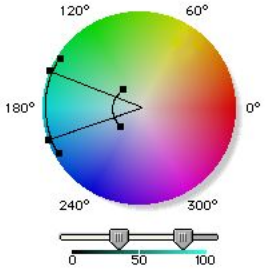

Figure 6: Round HSV Range Picker

In the picture above, the user selects a range of colors using a control similar to Apple's round HSV color picker. Instead of selecting a single color, though, they can select a maximum and minimum hue by adjusting the lines coming from the center of the circle. The user then selects a saturation range by moving the semi-circular controls in and out. Finally, they can use the double slider below the circle to choose a range for the value parameter.

The advantage to using a control like this is that it uses controls similar to Apple's HSV color picker which is familiar to users. The disadvantage is that programming and using curved controls is cumbersome.

Square Range Picker

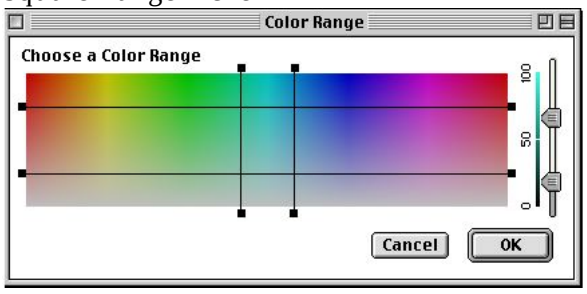

Figure 7: Square HSV Range Picker

With the Square Range Picker, pictured above, the hues are laid out horizontally, and vary in saturation vertically. The user can then choose a value range using the double slider on the right. (You could also switch parameters around so

that saturation is on a slider, and value is displayed along with hue.) This is significantly easier to program, and the user should have an easier time manipulating the controls, as well. However, choosing shades of red can be somewhat problematic, since they are split at 0° and 360°. Allowing the user to rotate the hues should solve the problem, although it's not immediately obvious to the user that they can do so.

Eye Dropper Picker

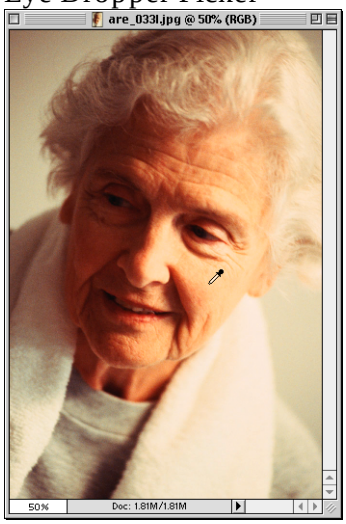

Figure 8: Eye Dropper Range Picker

A third method of choosing a range of colors is extremely simple for the user, but presents some challenges to the programmer. If you display the content to be manipulated and allow the user to simply click on it at different places, you can get a list of samples from the image. You can then base your parameters on the maximum and minimum hue, saturation, and value of the samples chosen. However, if the user chooses wildly varying hues, covering more than 180°, how do you decide what the range you should cover is? For example, if a user chooses yellow at 60°, then chooses a bluish cyan, at 200°, and then chooses a bluish magenta at 265°, should the hue range be 60° through 265°? Or should it be -105° (265° coming from the other direction) through 200°?

Using Matched Colors During Processing

The other major useful application of HSV space is that you can more easily match colors in a way that is fairly consistent with human color perception. So what does that get you? It allows you to apply manipulations (in other words filters and effects) only to selected areas of an image or video clip. This is useful because it means you can apply your effects to objects in complex scenes without having to film two scenes and composite them together later. It also saves time by not applying a complex filter to an entire image when only part of the image needs to be manipulated.

HSV Curves

If you've ever used photo editing software, you've probably dealt with the "curves" dialog. In Photoshop, for example, you can change the output of any RGB color channel by adjusting the graph in the curves dialog as shown below.

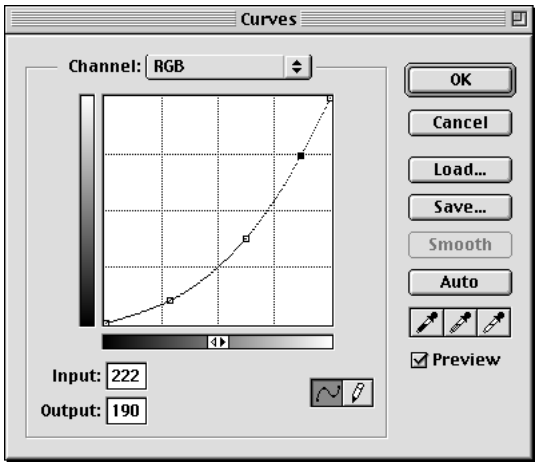

Figure 9: Photoshop's Curves Tool

We can create a similar tool for adjusting the hue, saturation and value of an image. This allows us some unique opportunities that aren't available with other tools. In RGB space it doesn't usually make sense to have the input of a curve be one channel, and the output to be another channel. Usually, you want the input and output to be the same. But in HSV space, we can use the fact the hue is in one component, and the saturation or value is another component to create some interesting results. Let's say you want the blues in your image to be darker. If you have hue as the input and value as the output, you can adjust the brightness of only the blue values.

Figure 10: Sample image before manipulation

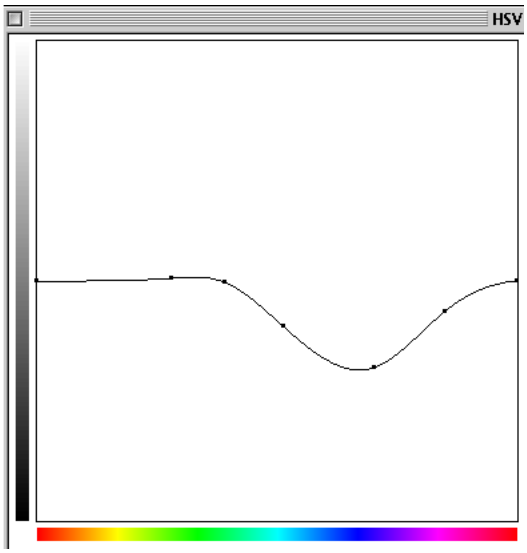

Figure 11a: Hue/Value curve

Figure 11b: Result of Hue/Value manipulation

Unlike with the RGB curves, where adjusting the blue channel adjusts any color that has blue in it, including cyan and magenta and gray tones, adjusting the blues in HSV space only adjusts those colors that users think of as actually being blue.

If you want to change one color to another, you can do that by making the hue be the input and the output. Let's say you wanted to change yellow to green. Just adjust the curves so that hues in the yellow range are moved to the green range, while leaving the rest alone.

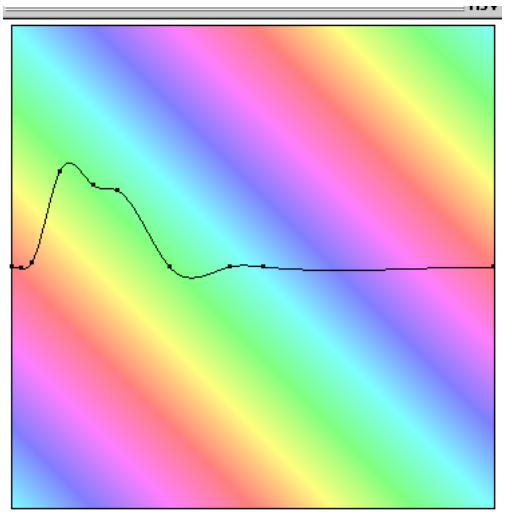

Figure 12a: Hue/Hue curve

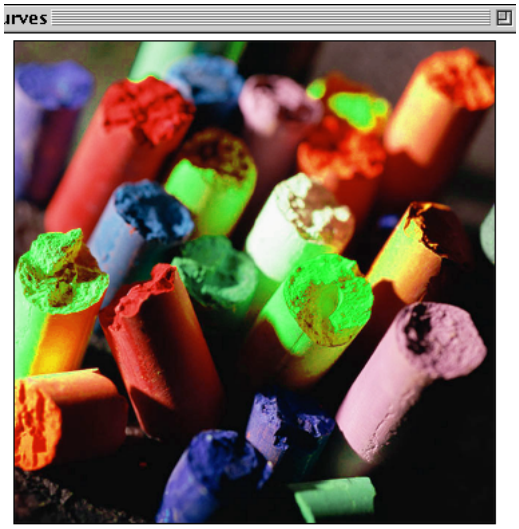

Figure 12b: Yellows are shifted into the green range using the HSV Curves dialog

You can use variations in the output saturation to remove color from objects. Maybe you want only reds to show up in your image. Adjust your HSV curves like so:

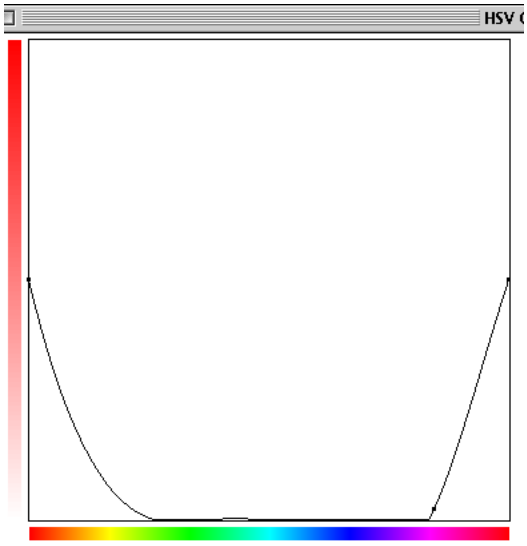

Figure 13a: Hue/Saturation Curve

Figure 13b: All colors, except for reds are desaturated

The possibilities are mind boggling. Layering the effect several times gives you an opportunity to do even more complex changes. And the great thing about it is that you can make the changes using the perceptual tools you're used to using with color.

Beer Goggles

Adjusting colors can be fun and even useful, but the real power of using HSV becomes obvious when you start applying filters only to areas that are a particular color, or in a particular color range. We all need a tool to make us look better, right? We need to get rid of wrinkles, acne, moles, body odor, etc., don't we? Well the HSV color space is just the place to do that. (Except for the body odor. You'll need to buy some deodorant for that.) By selecting the skin tones in an image and applying a filter which eliminates noise or dust and scratches, you can effectively make people look better and younger without effecting the entire frame. Here's the basic algorithm:

```
for each row in the image
  for each pixel in the row
      Convert the pixel to HSV space
      if the pixel is close to the hue,
      saturation and value chosen
          destination pixel = Filter (current
               pixel)
      else
          destination pixel = current pixel
      end if
```

```
  end for
end for
```

So how well does it work? Here are some tests I ran:

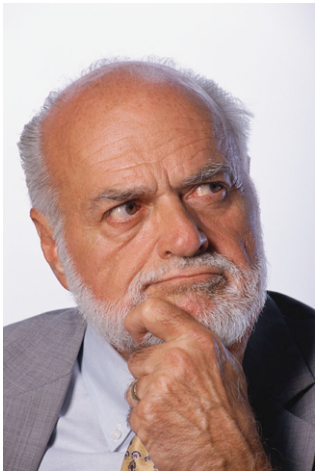

Figure 14a: A human subject

Figure 14b: Before and after Despeckle applied to subject's skin tones

Figure 14c: Before and after a 3x3 Median applied to subject's skin tones

The effects of the Despeckle version (see Figure 14b) are a bit too subtle. The 3x3 median (see Figure 14c) does a good job of removing the visible pores without altering the image unnaturally. The detail in the hair and eyebrows remains.

Figure 14d: Before and after a Gaussian Blur with radius 1.8 applied to subject's skin tones

Figure 14e: Gaussian Blur with radius 3.2 applied to subject's skin tones

The Gaussian Blur with a radius of 1.8 is also pretty good (see Figure 14d). Turning it up to 3.2 causes too much blurring of other areas of the image (see Figure 14e).

As you can see from the before and after picture below, the rest of the image retains its crispness.

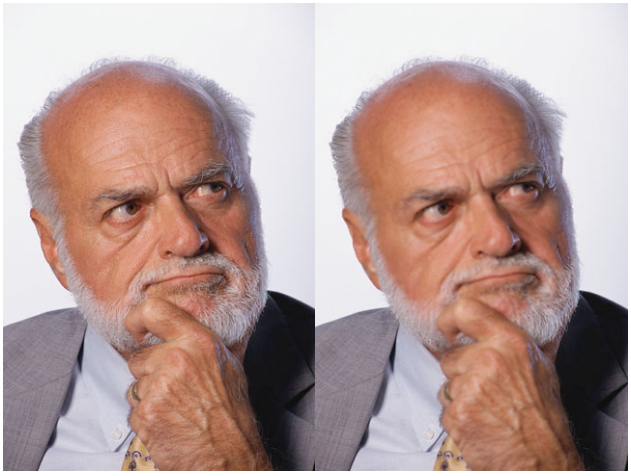

Figure 15: The entire image, before and after Gaussian blur with radius 1.8.

Cartoon Filter

Another fun filter that works in HSV space is the filter which makes photographic images look like cartoons. Simply constrain the hue of each pixel to be the nearest of the six primary or secondary colors and constrain the saturation and value to a limited number of values. Whenever there is a change in the constrained hue from one pixel to the next, replace the pixel with a black pixel. This gets you an outline around objects when the hue changes from one color to another.

In experimenting with this technique, I've found that using a wider neighborhood for deciding if a pixel should be black improves the quality of the outline. I've also found that constraining pixels with saturation under about 30% or value under about 25% to being grayscale also helps improve the quality of the output.

Some Caveats

As you can see, working in HSV space offers several opportunities for improving the ease of use of color input from users, improved color matching within your application, and lots of really cool filters for a variety of applications. However, there are some things you need to watch out for when using HSV space. The gray tones, from black to white, have undefined hue and 0 saturation. As such, they can present some special problems for both users trying to select or match colors, and programmers trying to use those colors.

In particular, colors with very low saturation tend to look like shades of gray to a user. When manipulating colors based on their hue, you can get some unexpected results when dealing with low saturation colors. The problem is made much worse by compression algorithms that use this fact to throw out data in order to save space. Colors with widely different hues that all appear to be gray to the user, may get assigned the same value in a compressed image. Applying a filter to that image will leave a large unnatural block of the image filtered, and looking terribly processed.

In addition to the discontinuity of hue when saturation is set to 0, there is the problem of the colors wrapping around at 360°. This can usually be dealt with by always setting your hues to be within the 0-360° range using modulus arithmetic. However, when subtracting one hue from another to find how close they are, you need take special care to make sure that values on the high end of the hue wheel are considered close to values on the low end. This can generally be dealt with by normalizing the difference of two hues to always be in the -180° to 180° range.

Bibliography

[Adobe94] Adobe Systems, Inc., *User Guide Adobe Premiere™ version 4.0*, Adobe Systems, Inc. 1994

[Adobe2000] Adobe Systems, Inc., *Adobe® Photoshop® 6.0 User Guide for Windows® and Macintosh*, Adobe Systems, Inc. 2000

[Apple86] Apple Computer, Inc., *Inside Macintosh Volume V*, Addison Wesley, Reading, MA. 1986

[Foley90] Foley, van Dam, Fiener, Hughes, *Computer Graphics: Principles and Practice*, Addison Wesley, Reading, MA. 1990