Linear v\s Non-Linear Color Spaces

Ramit Pahwa 14MA20029

September 3, 2017

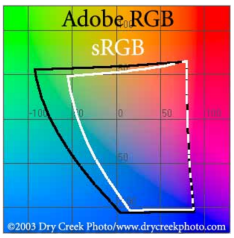
1 Introduction

A color space is a specific organization of colors. In combination with physical device profiling, it allows for reproducible representations of color, in both analog and digital representations. A color space may be arbitrary, with particular colors assigned to a set of physical color swatches and corresponding assigned names or numbers such as with the Pantone collection, or structured mathematically, as with **NCS System, Adobe RGB or sRGB**.
A color model is mathematically representation describing the are represented as tuples of numbers (e.g. triples in RGB or quadruples in CMYK). A specific mapping function between a color model and a reference color space establishes within the reference color space a definite "footprint", known as a **gamut**, and for a given color model this defines a color space.
Adobe RGB and sRGB are two different absolute color spaces, both based on the RGB color model. When defining a color space, the usual reference standard is the CIELAB or CIEXYZ color spaces, which were specifically designed to encompass all colors the average human can see.

2 What are Color Spaces?

In simple terms, imagine a box containing all the visible colors. The farther from the center of the box you go, the more saturated the colors become — Red towards one corner, Blue towards another, Green towards the third (our box has a curious shape). A Cyan, Magenta, Yellow color space works the same way, except that the primary colors are CMY rather than RGB. For simplicity, we will refer only to RGB spaces, but the comments apply equally to CMY(K) color spaces. A color space can be represented as a balloon blown up inside the box. The space taken up by the balloon is the portion of the total number of visible colors that fall within the particular color space. Larger balloons contain more colors, or have a larger gamut, while smaller balloons hold fewer colors. The surface of the balloon has the most saturated colors that the color space can hold. Any colors falling outside the balloon can't be reproduced in that color space.
Colors inside the balloon are described using (R,G,B) coordinates. The most saturated (i.e. purest) red in any color space has an R-value of 255. Since larger color spaces have larger balloons, they contain both more air volume (i.e. more colors), and the surface of the balloon is farther from the center of the box (i.e.

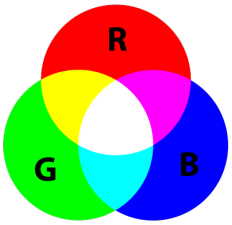
the colors are more saturated).



Therefore, larger color spaces such as Adobe RGB contain both more colors and more highly saturated colors than smaller spaces like sRGB. A comparison of the Adobe RGB and sRGB gamuts is below. As you can see, working with Adobe RGB allows you to see and print more of most colors. Adobe RGB was designed to contain the entire color gamut available from most CMYK printers. sRGB is an HP/Microsoft defined color space that describes the colors visible on a low end monitor.
In general, you want to use color spaces that are as large as is practical. For example, if your printer is capable of producing output in a color space larger than sRGB, there is no reason to hobble your work by limiting output to the small sRGB gamut. If you do, you'll lose the saturated cyans and greens that can make your prints stand out.

3 Various Color Spaces

CIE 1931 XYZ color space was one of the first attempts to produce a color space based on measurements of human color perception and it is the basis for almost all other color spaces. The **CIERGB** color space is a linearly-related companion of **CIE XYZ**. Additional deriatives of **CIE XYZ** include the **CIELUV, CIEUVW, and CIELAB**.

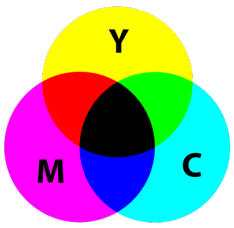


RGB uses additive color mixing, because it describes what kind of light needs to be emitted to produce a given color. RGB stores individual values for red, green and blue. RGBA is RGB with an additional channel, alpha, to indicate transparency. Common color spaces based on the RGB model include **sRGB, Adobe RGB, ProPhoto RGB, scRGB, and CIE RGB.**

CMYK uses subtractive color mixing used in the printing process, because it describes what kind of inks need to be applied so the light reflected from the substrate and through the inks produces a given color. One starts with a white substrate (canvas, page, etc.), and uses ink to subtract color from white to create an image. CMYK stores ink values for cyan, magenta, yellow and black. There are many CMYK color spaces for different sets of inks, substrates, and press characteristics (which change the dot gain or transfer function for each ink and thus change the appearance).
YIQ was formerly used in NTSC (North America, Japan and elsewhere) television broadcasts for historical reasons. This system stores a luma value roughly analogous to (and sometimes incorrectly identified as) luminance, along with two chroma values as approximate representations of the relative amounts of blue and red in the color. It is similar to the YUV scheme used in most video

capture systems and in PAL (Australia, Europe, except France, which uses SECAM) television, except that the YIQ color space is rotated 33 degrees with respect to the YUV color space and the color axes are swapped. The YDbDr scheme used by SECAM television is rotated in another way.
YPbPr is a scaled version of YUV. It is most commonly seen in its digital form, YCbCr, used widely in video and image compression schemes such as MPEG and JPEG.
xvYCC is a new international digital video color space standard published by the IEC (IEC 61966-2-4). It is based on the ITU BT.601 and BT.709 standards but extends the gamut beyond the R/G/B primaries specified in those standards.
HSV (hue, saturation, value), also known as HSB (hue, saturation, brightness) is often used by artists because it is often more natural to think about a color in terms of hue and saturation than in terms of additive or subtractive color components. HSV is a transformation of an RGB color space, and its components and colorimetry are relative to the RGB color space from which it was derived.
HSL (hue, saturation, lightness/luminance), also known as HLS or HSI (hue, saturation, intensity) is quite similar to HSV, with "lightness" replacing "brightness". The difference is that the brightness of a pure color is equal to the brightness of white, while the lightness of a pure color is equal to the lightness of a medium gray.

4 Linear v\s Non-Linear color gamet



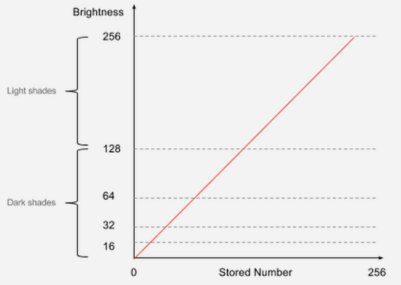
In a linear color-space, the relationship between the numbers you store and the intensities they represent is linear. Practically, this means that if you double the number, you double the intensity (the lightness of the gray). If you want to add two intensities together (because you're computing an intensity based on the contributions of two light sources, or because you're adding a transparent object on top of an opaque object), you can do this by just adding the two numbers together. **If you're doing any kind of 2D blending or 3D shading, or almost any image processing, then you want your intensities in a linear color-space, so you can just add, subtract, multiply, and divide numbers to have the same effect on the intensities**. Most color-processing and rendering algorithms only give correct results with linear RGB, unless you add extra weights to everything.
That sounds really easy, but there's a problem. The human eye's sensitivity to light is finer at low intensities than high intensities. That's to say, if you make a list of all the intensities you can distinguish, there are more dark ones than light ones. To put it another way, you can tell dark shades of gray apart better than you can with light shades of gray. In particular, if you're using 8 bits to represent your intensity, and you do this in a linear color-space, you'll end up with too many light shades, and not enough dark shades. You get banding in your dark areas, while in your light areas, you're wasting bits on different shades of near-white that the user can't tell apart. To avoid this problem, and make

the best use of those 8 bits, we tend to use sRGB. The sRGB standard tells you a curve to use, to make your colors non-linear. The curve is shallower at the bottom, so you can have more dark grays, and steeper at the top, so you have fewer light grays. If you double the number, you more than double the intensity. This means that if you add sRGB colors together, you end up with a result that is lighter than it should be. These days, most monitors interpret their input colors as sRGB. So, when you're putting a color on the screen, or storing it in an 8-bit-per-channel texture, store it as sRGB, so you make the best use of those 8 bits.

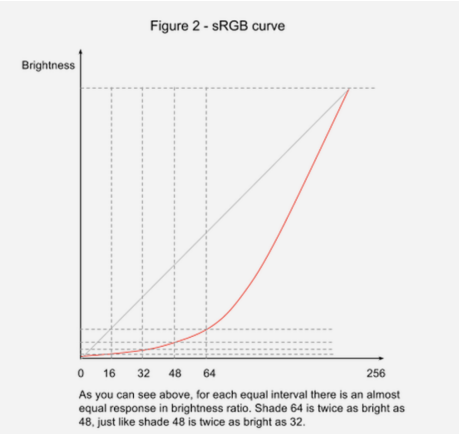
5 Analysis

I noticed we now have a problem: we want our colors processed in **linear space, but stored in sRGB.** This means you end up doing sRGB-to-linear conversion on read, and linear-to-sRGB conversion on write.As we've already said that linear 8-bit intensities don't have enough darks, this would cause problems, so there's one more practical rule: don't use 8-bit linear colors if you can avoid it. It's becoming conventional to follow the rule that 8-bit colors are always sRGB, so you do your sRGB-to-linear conversion at the same time as widening your intensity from 8 to 16 bits, or from integer to floating-point; similarly, when you've finished your floating-point processing, you narrow to 8 bits at the same time as converting to sRGB. If you follow these rules, you never have to worry about gamma correction.

Figure 1 - Linear color space



You can notice that shade 64 is twice as bright as 32, just like shade 256 is twice as bright as 128. This means that we end up with a lot of very bright shades (128-256), while having too few in the very dark spectrum (0-16)



6 Conversions

When you're reading an sRGB image, and you want linear intensities, apply this formula to each intensity:

```
float s = read_channel();
float linear;
if (s <= 0.04045)
    linear = s / 12.92;
else
    linear = pow((s + 0.055) / 1.055, 2.4);
```

Going the other way, when you want to write an image as sRGB, apply this formula to each linear intensity:

```
float linear = do_processing();
float s;
if (linear <= 0.0031308)
    s = linear * 12.92;
else
    s = 1.055 * pow(linear, 1.0/2.4) - 0.055;
```

In both cases, the floating-point s value ranges from 0 to 1, so if we are reading 8-bit integers you want to divide by 255 first, and if we are writing 8-bit integers you want to multiply by 255 last, the same way you usually would. Up to now, we have dealt with one intensity only, but there are cleverer things to do with colors. The human eye can tell different brightnesses apart better than different tints (more technically, it has better luminance resolution than chrominance), so you can make even better use of your 24 bits by storing the brightness separately from the tint. This is what YUV, YCrCb, etc. representations try to do. The Y channel is the overall lightness of the color, and uses more bits (or has more spatial resolution) than the other two channels. This way, you don't (always) need to apply a curve like you do with RGB intensities. YUV is a linear color-space, so if you double the number in the Y channel, you double the lightness of the color, but you can't add or multiply YUV colors together like you can with RGB colors, so it's not used for image processing, only for storage and

transmission.
Before sRGB, old CRTs used to have a non-linearity built into them. If you doubled the voltage for a pixel, you would more than double the intensity. How much more was different for each monitor, and this parameter was called the gamma. This behavior was useful because it meant you could get more darks than lights, but it also meant you couldn't tell how bright your colors would be on the user's CRT, unless you calibrated it first. Gamma correction means transforming the colors you start with (probably linear) and transforming them for the gamma of the user's CRT. OpenGL comes from this era, which is why its sRGB behavior is sometimes a little confusing. But GPU vendors now tend to work with the convention described above: that when you're storing an 8-bit intensity in a texture or framebuffer, it's sRGB, and when you're processing colors, it's linear.

7 References

- https://www.drycreekphoto.com/Learn/color_spaces.html
- http://www.tiberius-viris.com/single-post/2016/05/16/Color-Space-Management-Linear-Log-sRGB
- https://en.wikipedia.org