Color

Color models and color spaces

While the previous chapter traced some of the important developments in the history of color theory, this chapter takes a deeper look at the current landscape of digital color theory. When working with color in programming languages, one will encounter quite a few terms used – often interchangeably – to describe a color's position within the color spectrum. In this chapter, we will look at three of these terms – color models, color spaces, and color profiles – and examine why it is important to develop a decent understanding of these concepts when working with color in code.

Color models

To understand the nature of something, it can be helpful to create a visual representation of the subject. In fact, humans tend to do this quite often, from scribbling notes in lectures, to drawing charts and maps to explain specific datasets. We do this because many of us are visual learners, and seeing something is different than hearing it. Throughout history, artists and scientists have depicted the color spectrum in all sorts of different models, with the goal of turning the abstract concept of the color spectrum into something comprehensible.

A color model is a visualization that depicts the color spectrum as a multidimensional model. Most modern color models have 3 dimensions (like RGB), and can therefore be depicted as 3D shapes, while other models have more dimensions (like CMYK). In the following, we will look at the RGB, HSV, and HSL color models, which are all prevalent in current digital design tools and programming languages. These color models all use the same RGB primary colors, which makes them good examples of how color models can visualize the same color spectrum in widely different dimensions.

RGB is a color model with three dimensions – red, green, and blue – that are mixed to produce a specific color. When defining colors in these dimensions, one has to know the sequence of colors in the color spectrum, e.g. that a mix of 100% red and green produces yellow. The RGB color model is often depicted as a cube by mapping the red, green, and blue dimensions onto the x, y, and z axis in 3D space. This is illustrated in the interactive example below, where all possible color mixes are represented within the bounds of the cube.

Drag the sliders to see the resulting color.
Click and drag the color model to rotate.

50% red

50% green

100% blue

The RGB color model is not an especially intuitive model for creating colors in code. While you might be able to guess the combination of values to use for some colors such as yellow (equal amounts of red and green) or the red color used on Coca-Cola bottles (lots of red with a little bit of blue), less pure colors are much harder to guess in this color model. What values would you use for a dark purple? How about finding the complimentary color for cyan? If you cannot find the answer, it is because humans do not think about colors as mixes of red, green, and blue lights.

HSV is a cylindrical color model that remaps the RGB primary colors into dimensions that are easier for humans to understand. Like the Munsell Color System, these dimensions are hue, saturation, and value.

- *Hue* specifies the angle of the color on the RGB color circle. A 0° hue results in red, 120° results in green, and 240° results in blue.
- *Saturation* controls the amount of color used. A color with 100% saturation will be the purest color possible, while 0% saturation yields grayscale.
- *Value* controls the brightness of the color. A color with 0% brightness is pure black while a color with 100% brightness has no black mixed into the color. Because this dimension is often referred to as brightness, the HSV color model is sometimes called HSB, including in P5.js.

It is important to note that the three dimensions of the HSV color model are interdependent. If the value dimension of a color is set to 0%, the amount of hue and saturation does not matter as the color will be black. Likewise, if the saturation of a color is set to 0%, the hue does not matter as there is no color used. Because the hue dimension is circular, the HSV color model is best depicted as a cylinder. This is illustrated in the interactive example below, where all possible color mixes are represented within the bounds of the cylinder.

Drag the sliders to see the resulting color.
Click and drag the color model to rotate.

0° hue

50% saturation

70% brightness

HSL is another cylindrical color model that shares two dimensions with HSV, while replacing the value dimension with a lightness dimension.

- *Hue* specifies the angle of the color on the RGB color circle, exactly like HSV.
- *Saturation* controls the purity of the color, exactly like HSV.
- *Lightness* controls the luminosity of the color. This dimension is different from the HSV value dimension in that the purest color is positioned midway between black and white ends of the scale. A color with 0% lightness is black, 50% is the purest color possible, and 100% is white.

Even though the saturation dimension theoretically is similar between the two color models (controlling how much pure color is used), the resulting saturation scales differ between the models caused by the brightness to lightness remapping. Like HSV, the HSL color model is best depicted as a cylinder, which is illustrated in the interactive example below.

Drag the sliders to see the resulting color.
Click and drag the color model

to rotate.

0° hue

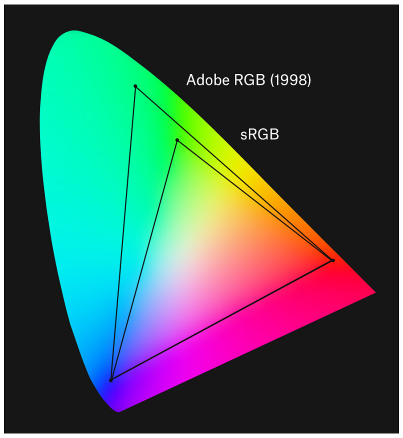
50% saturation

70% lightness

There are plenty of other ways to visualize the color spectrum in a multi-dimensional space. The CMYK color model has four dimensions, which means that one has to use either animation or multiple 3D shapes to visualize the states of the model. Another color model called CIELAB is modeled on the opponent-process theory of human perception with two of three dimensions representing scales from red to green and yellow to blue – two opponent color pairs that humans cannot perceive simultaneously.

Color spaces

Color models provide for a good way to visualize the color spectrum, but they are inadequate when it comes to defining and displaying colors on computer screens. To explain this, let us assume that you own a laptop computer as well as a larger, external screen for your home office. Now, let us also assume that you are running a P5.js sketch showing a yellow ellipse on both screens. In a world without color spaces, these two screens would turn on their red and green subpixels and be done with it. However, what if your larger screen has more expensive lights that look wildly different from the ones on your laptop screen? This would result in two very different kinds of yellow. This is the problem that color spaces set out to solve.

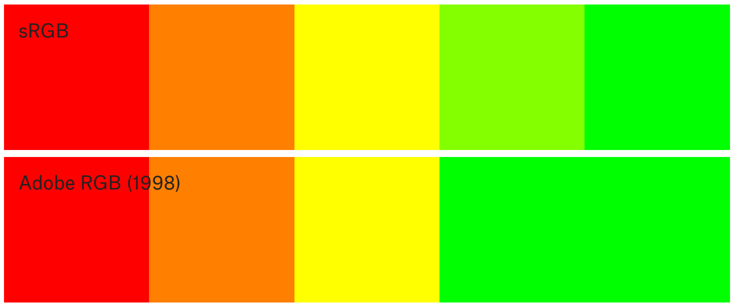


The CIE chromaticity diagram showing the color gamuts of the Adobe RGB (1998) and sRGB color spaces.

This chromaticity diagram was created by the International Commission on Illumination (CIE). It was based on a number of vision experiments on human subjects in the 1930's, and it accurately defines the relationship between the wavelength of a color and the perceived effect on the human eye. This diagram – which is also a color space called CIEXYZ – is very important as all modern color spaces define their absolute range of colors (called a color gamut) in relation to this color space. The two triangles inside the curved shape indicate the color gamuts of two popular color spaces: sRGB and Adobe RGB (1998). The corners of each triangle define the primary colors of each color gamut, and you might notice that while the two color spaces share the same red and blue primaries, the green primary color is different between the two. To put it another way, a primary color only has absolute meaning when it refers to a specific color space. In our example from above, color spaces allow your two computer monitors to show an identical yellow color by following a standard process: First, it converts the yellow color from the color space of the P5 sketch to the CIEXYZ color space (also called a reference color space). Then, because every monitor knows the exact color of their primary lights in relation to the

CIEXYZ color space, it can determine the amount of primary lights to mix.

The sRGB color space has the smallest color gamut of the two color spaces, which means that it covers the smallest range of colors. It was created for use by computer monitors, and the smaller gamut reflects the exact colors of the primary lights in most HDTVs and computer monitors. This also means that the sRGB color space is easy to adapt for hardware manufacturers, which is why it has become the most widely used color space for digital files. Whenever you come across a color or an image on a website, it is most likely an sRGB color. Even though sRGB is a great color space for the range of colors that can be shown on a screen, the color gamut is not wide enough to support colors printed in ink – especially in the green-blue parts of the spectrum. The Adobe RGB (1998) color space has a much wider RGB color gamut that was carefully chosen to cover most of the colors that CMYK printers can produce. This also means that a specific set of colors can look very different depending on the color space it adheres to, as demonstrated in the example below that shows the same RGB values in the two color spaces. Notice how the last two green colors look identical in the Adobe RGB (1998) color space, because most screens cannot display the green primary color of the wider color gamut.

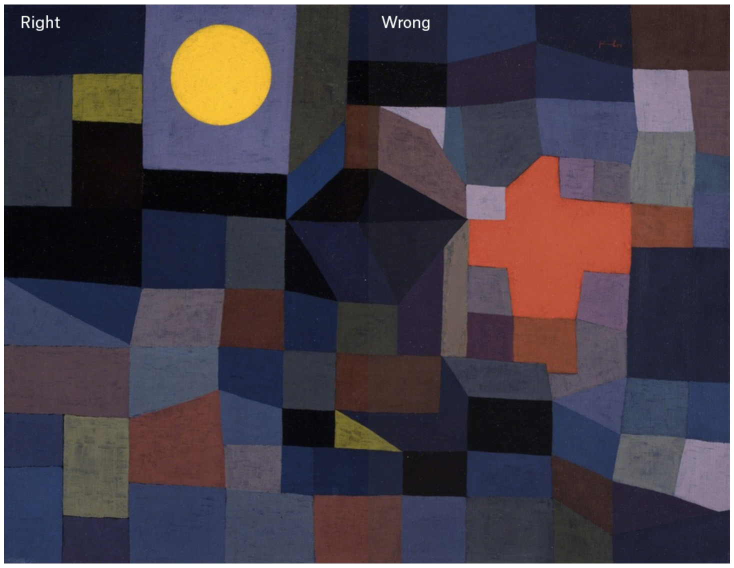


A simulation of how most monitors render the two color spaces. The last two colors in the Adobe RGB (1998) color space will look identical.

It is important to note that although color models are abstract mathematical concepts, it is impossible to visualize a color model without an accompanying color space. The RGB, HSV, and HSL color model examples from above are all visualized within the sRGB color space, because that is the default color space of the internet.

Color profiles

A digital image can adhere to a specific color space by embedding a color profile in its metadata. This tells any program that wants to read the image that the pixel values are stated according to a particular color space, and images without a color profile are often assumed to be sRGB. Color profiles are important in order to correctly reproduce identical colors across multiple devices, and you will often see professional printing services require image files to be set to a specific color space (most likely Adobe RGB (1998) or ProPhoto RGB, a color space with a very wide color gamut). This assures that the colors in your image are not interpreted to be the wrong color space. If you have ever pasted an image into an existing Photoshop project only to have the colors look wrong, you have been a victim of this. As an example, if you paste an image with an Adobe RGB (1998) profile into a Photoshop file with a sRGB profile, Photoshop will interpret the pixel values to be within the smaller color gamut, changing the colors of your pasted image. Because of this, most digital design tools have built-in commands to convert between color spaces, and Photoshop actually does a good job of alerting the user before reinterpreting color profiles. Conversion between color spaces is especially beneficial for designers wanting to design print products in code, as their digital assets will need to be converted from sRGB to a print-specific color profile before printing.



The left-hand side of this Paul Klee painting was correctly converted from Adobe RGB (1998) to sRGB, while the right-hand side wrongly reinterpreted the colors into sRGB without conversion. ©

If a digital image uses a color profile with a wide color gamut, it is almost guaranteed to lose colors on most screens because most screens can only show colors within the sRGB gamut. However, many newer screens support wider color gamuts. The Apple iMac retina screen uses a RGB color space called DCI-P3 with a color gamut that spans about the same range as Adobe RGB (1998), but it includes more red-yellow colors and excludes some green-blue colors. To highlight the complexity of color management, some browsers running on retina computers may oversaturate the colors of sRGB images without color profiles, while other browsers will correctly convert the sRGB pixel values into DCI-P3. Although this book will not dive further into the complicated aspects of color management, there are plenty of good resources out there for the interested reader.

Color in P5.js

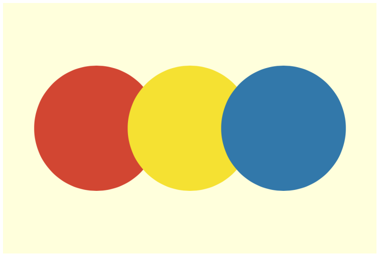
As a browser-based JavaScript library, all color values in P5.js adhere to sRGB, the standard color space for the internet. You can define these colors in all three of the aforementioned color models: RGB, HSV (called HSB), and HSL. Passing color values to the `fill()` and `stroke()` functions is the main way to color shapes in P5.js. This sets the current fill and stroke colors for all subsequent shapes, and this setting is remembered until you `fill()` or `stroke()` again, or disable the stroke or fill entirely with the `noStroke()` or `noFill()` functions.

The default color model in P5.js is RGB, which means that the `fill()` and `stroke()` functions expect three numbers between 0 and 255, indicating the amount of red, green, and blue to use for the color. The reason behind this specific range is that a maximum of 256 values can be stored in a single byte (8 bits), allowing each RGB color to only take up 24 bits. Even though 256 different amounts of red, green, and blue might not sound like much, this can produce 16,777,216 distinct colors which is actually much more than what the human eye can perceive.

```
noStroke();
fill(210, 70, 50);
ellipse(150, height/2, 200,
200);

fill(245, 225, 50);
ellipse(300, height/2, 200,
200);

fill(50, 120, 170);
ellipse(450, height/2, 200,
```



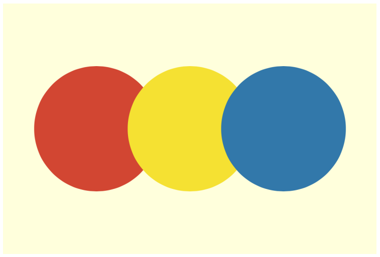
```
200);
```

P5.js also allows you to use an alternative hexadecimal syntax known from web design for specifying colors in the RGB color model. Instead of using three numbers, the hex syntax uses a hashtag followed by a six-character string to represent the primary color values. Each primary color has two characters in this string, using the numbers 0-9 to represent zero to nine and the letters A-F to represent ten to fifteen. With 16 variations per character, each primary color can therefore specify a value between 0 and 255 in just two characters.

```
noStroke();
fill("#d24632");
ellipse(150, height/2, 200,
200);

fill("#f5e132");
ellipse(300, height/2, 200,
200);

fill("#3278aa");
ellipse(450, height/2, 200,
200);
```

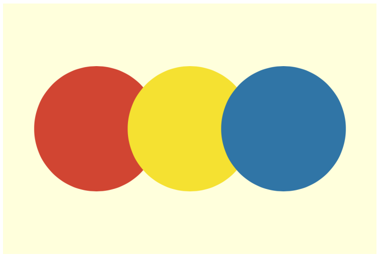


The `colorMode()` function in P5.js can be used to switch to another color model, which means that the `fill()` and `stroke()` functions will expect color ranges according to the new color model's dimensions. The default numerical ranges for HSV (called HSB in P5.js) and HSL are 0-360 for hue (indicating an angle), and 0-100 for saturation and brightness/lightness (indicating a percentage). The following code example uses both the HSV and HSL color models to draw the three ellipses.

```
noStroke();
colorMode(HSB);
fill(7, 76, 82);
ellipse(150, height/2, 200,
200);

fill(54, 80, 96);
ellipse(300, height/2, 200,
200);

colorMode(HSL);
fill(205, 55, 42);
ellipse(450, height/2, 200,
200);
```

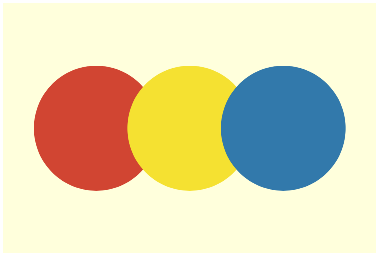


It is also possible to change the default numerical ranges for each color model. This can be done by passing in three additional numbers when calling the `colorMode()` function, as demonstrated below where all three dimensions of the HSV color model are set to 0-1 ranges.

```
noStroke();
colorMode(HSB, 1, 1, 1);
fill(0.0195, 0.76, 0.82);
ellipse(150, height/2, 200,
200);

fill(0.15, 0.80, 0.96);
ellipse(300, height/2, 200,
200);

fill(0.569, 0.71, 0.67);
ellipse(450, height/2, 200,
200);
```



In the following chapters, we will examine a range of different techniques for combining colors programmatically in P5.js. Many of these examples will use the HSV color model, as it is an intuitive way to navigate the color spectrum.

Previous Chapter
A short history of color theory

Subscribe to Newsletter

Next Chapter
Perceptually uniform color spaces