Creative Computing II

Device-Dependent Colour Spaces

Wednesday 20th October 2010

This lab sheet explores device-dependent colour spaces and some related perceptual aspects of colour vision.

1. This part of the lab involves implementing the mathematical transformations to convert between the HSB and RGB colour spaces.

   (a) Write a *Processing* sketch including a function which converts from an RGB representation of a colour to HSB, and some way for the user of your sketch to specify an input colour. You will need to choose an appropriate data structure for your colours; one possibility is a *Processing* class, but there are others.

   (b) Implement for your sketch the reverse conversion, from HSB to RGB. Again, provide for the user of your sketch some way to specify an input HSB colour to this conversion.

   One approach for this part of the lab is demonstrated in the 'RGB HSB Colour' sketch on the course website. In that sketch, we define classes and convenient constructor functions to represent RGB and HSB colours. We also define printing methods to print out the components of the colours, and also to print built-in `color` *objects in the same format. The* `setup()` *method simply performs some conversions and prints the results, those conversions being implemented below. The user-interface aspect of this lab is unimplemented in this sketch.*

   (c) Test whether your conversions agree with the built-in conversions in *Processing*, using the various colour accessors or `colorMode()`. If you observe a difference, try to explain why.

   You should find that the RGB to HSB conversion matches exactly with the internal conversion (performed on-demand in the appropriate `colorMode()`*; however, you may be a discrepancy between the values in the reverse conversion (HSB to RGB). The reason for this discrepancy is that the Processing colour representation is quantized: as you learnt last year, a colour is represented as a 32-bit integer, with 8 bits for each of red, green and blue (along with 8 bits for the alpha channel). This means that it's not possible to represent a colour with an arbitrary hue, saturation and brightness – Processing will round to the nearest colour in the 8-bit space. In contrast, the HSB class implemented in the sketch on the course website can represent arbitrary HSB colours, and so there is no rounding.*

2. This part of the lab explores colour mixing by area.

   (a) Write a *Processing* sketch which fills a 100×100-pixel square with a checkerboard pattern of pixels, alternately full red (the maximum value in red, and zero in the green and blue channels) and full yellow (maximum values in red and green, and zero in blue).

   Something like

```
void setup() {
  size(100,100);
  for(int x = 0; x < 100; x++)
```

```
    for(int y = 0; y < 100; y++) {
      stroke(255, (x+y)%2==0 ? 255 : 0, 0);
      point(x,y);
    }
}
```

should do. (Check that you understand how the red and yellow alternation works in this sketch).

(b) Run your sketch and step far enough back from the screen that the pixellation is no longer obvious. What colour does the square appear to be?

You will most likely observe that the square appears orange from a distance.

(c) Make a guess at the RGB values of the mixture colour in part 2b. By trial and error, find the RGB colour values of the single colour that best matches the mixture. Was your guess right? Try to explain any discrepancy.

You might have guessed that the RGB colour values would be `(255,127,0)` – and if you tried that, you should find that the solid colour doesn't match at all. The reason for this will be explored in the lecture and lab for week 4.

(d) Repeat this with some other colours. Make a table of the colour values in the mixture, and the closest match in RGB colour space to the mixture that you find. (We will return to this table of values in a week's time).

3. This part of the lab explores a particular effect of the low-level details of visual perception.

(a) Construct a *Processing* sketch which works on a colour image to display alternately (switching under user control): a greyscale version of that image, and a version of that image with the hue inverted (but saturation and value unchanged).

(b) Select an image from your own collection or from the Commons, ideally with many strong colours. Incorporate it into your sketch.

(c) Run your sketch, displaying the inverted hue version. Stare at a fixed point in the image for a few seconds, before switching (without moving your focus) to the greyscale version. What do you observe? (You might want to draw a small black dot to enable you to focus on a fixed location without distraction).

See the 'Sadowski' sketch on the course website.

Further Reading:

- Stokes, M., M. Anderson, S. Chandrasekar and R. Motta, *A Standard Default Color Space for the Internet – sRGB*. `http://www.w3.org/Graphics/Color/sRGB`
- Mann, J., *Lessons Learned from Mondrians Applied to Real Images and Color Gamuts*, Proc. Seventh Color Imaging Conference (1999).