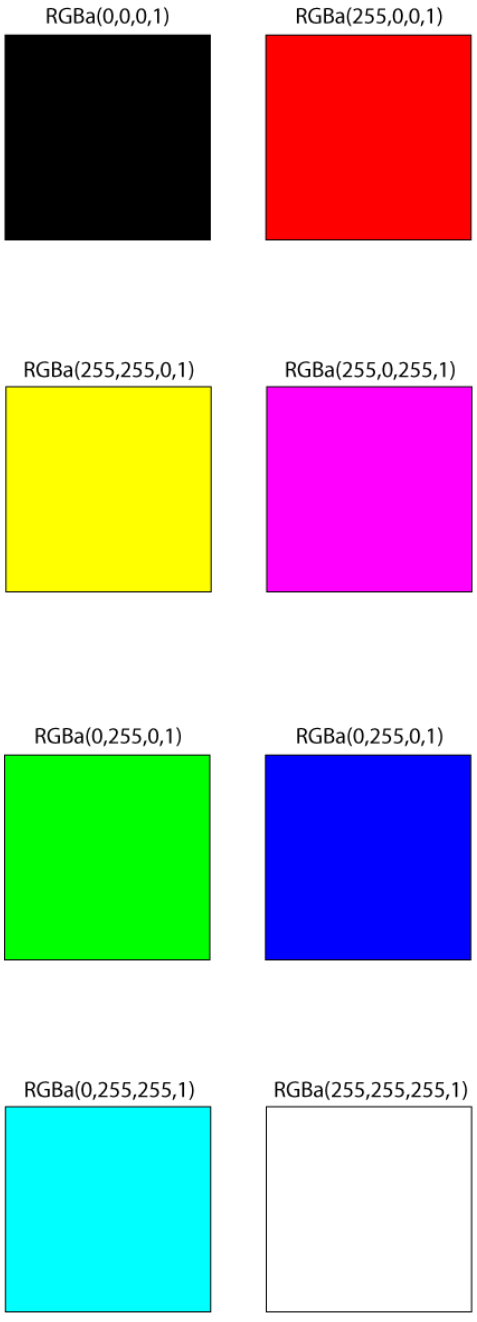
TIQC Colorblind Simulator Application

Philip Parzygnat (TIQC)
Queens College - Computer Science

Model Simulator

The model for the simulator created for this project is available on the world wide web via goo.gl/Q8UwjO. The simulator is called Coblis. Also contained on the site is an introductory treatment of the deficiency and a listing of the most common eight variations (below you will find a list of the eight variations) considered.

1. Achromatomaly
2. Achromatopsia
3. Deuteranomaly
4. Deuteranopia
5. Protanomaly
6. Protanopia
7. Tritanomaly
8. Tritanopia

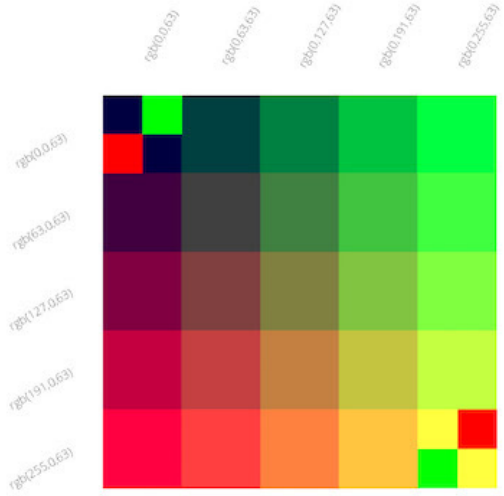


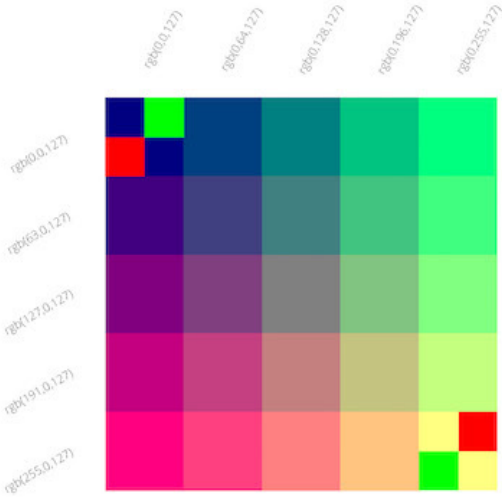
About RGBa

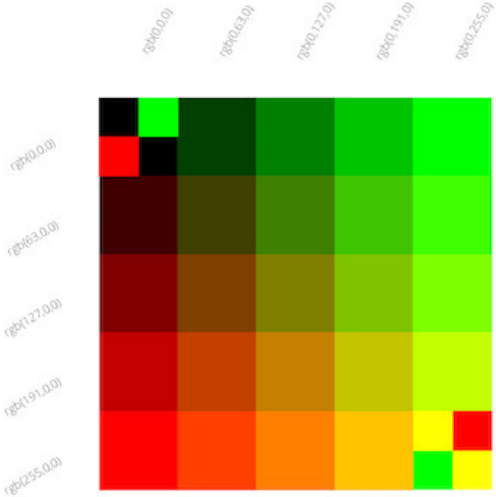
RGBa is a binary or hexadecimal encoding for color in digital formats. The encoding is composed of three 8-bit values (or three 2-hex values) along with an alpha value that scales from 0 to 1 and defines the opacity of the color encoded (a few examples of colors and their RGBa encodings are available below).

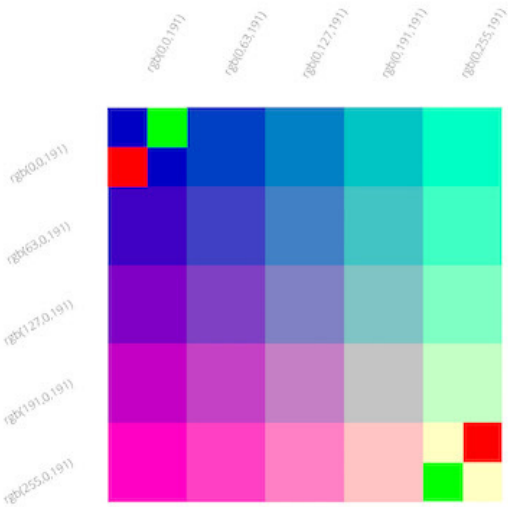
Coblis and Filtering Images

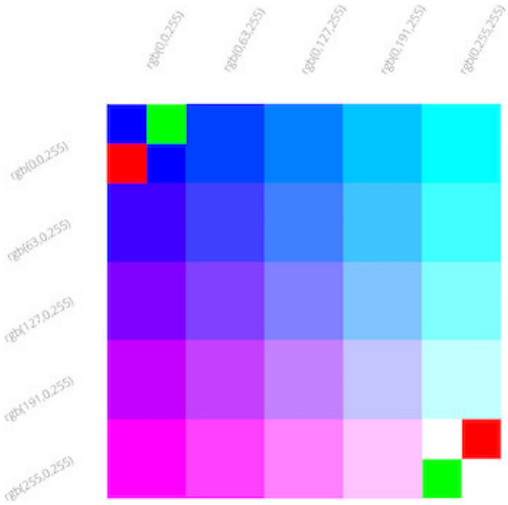
The Coblis simulator takes any image in Portable Network Graphics or JPEG and filters that image to produce an output mirroring one of the eight variations of color blindness. In working with the simulator it became clear that one way to determine how colors in RGB space are being mapped for any given filter is to test the simulator with images that have well defined color ranges. In effect a series of test strips were produced to determine how any particular filter converted the full range of colors on RGB (below is a set of these such test strips prior to being converted by Coblis).









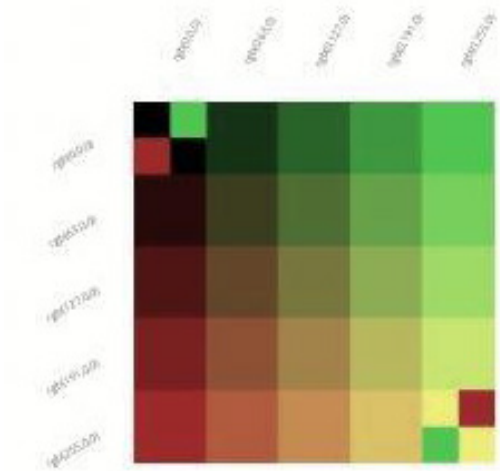


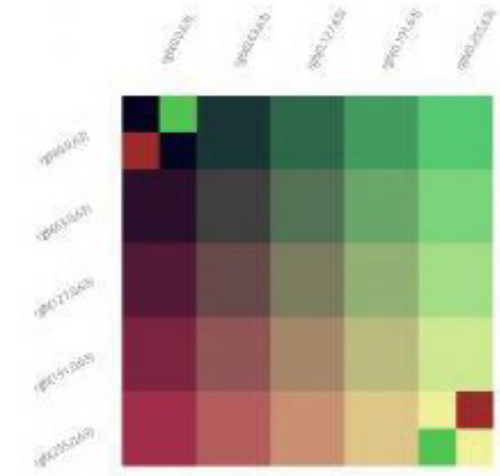
Test Strip Description

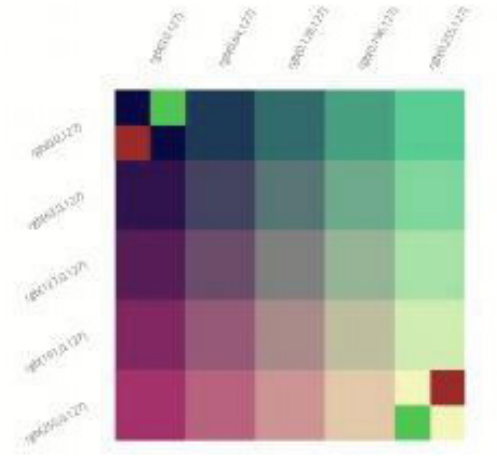
Each of five test strips contains in 6-bit increments all R and G (where R and G imply Red and Green) values for B (where B implies Blue) fixed. Then each consecutive test strip is a 6-bit increment with respect to B of the preceding test strip (to more clearly present the RGB values of these color blocks on the test strips see goo.gl/BwWVDj).

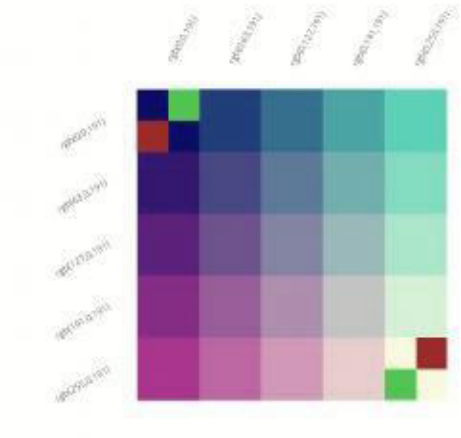
Applying Test Strips to Achromatomaly

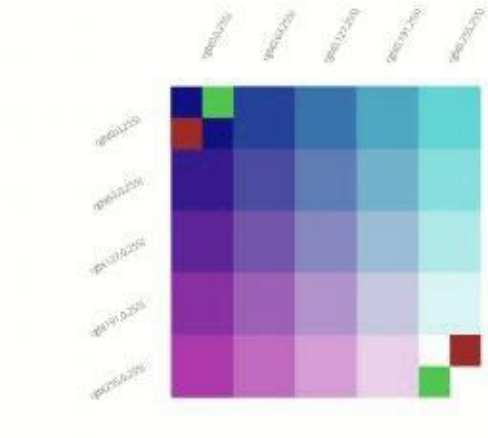
When using the test strips in Coblis to filter for achromatomaly the following results are returned:











Recording These Results

Using Digital Color Meter in Apple OS X each square of the test strips can be read and associated with a resultant RGB value. These results are then collected into a spreadsheet and analyzed (below is a sample, view the complete table of results via goo.gl/gPafl6).

R-Control	G-Control	B-Control
0	0	0
0	0	63
0	0	127
0	0	191
0	0	255
0	63	0
0	63	63
0	63	127
0	63	191
0	63	255
0	127	0
0	127	63
0	127	127
0	127	191
0	127	255
0	191	0
0	191	63
0	191	127
0	191	191
0	191	255
0	255	0
0	255	63
0	255	127
0	255	191
0	255	255

R-Test	G-Test	B-Test
1	0	2
7	4	34
16	8	66
20	17	101
18	25	118
24	41	19
28	53	54
31	58	86
36	61	120
42	66	150
43	99	44
50	103	77
53	107	108
59	111	140
61	116	171
66	150	67
70	156	97
75	160	128
80	164	164
84	168	191
85	197	88
91	202	117
96	205	150
100	208	184
105	215	213

Analysis and Conclusions on Achromatomaly Filter

Now that the test results have been recorded it is necessary to identify a transformation. Prior to this the results need to be reviewed and corrected as the Digital Color Meter and Colbis filter the image with a level of error. Therefore under the assumption that the achromatomaly filter produces a linear transformation of the RGB space (which is a fair assumption based on the data) a series of steps are taken to take the result measures and produce a set of proposed measures that describe the filter as linear. The process of analysis can be followed via this table.

R-Proposed	G-Proposed	B-Proposed
0	0	0
4	8	32
8	16	64
12	24	96
16	32	128
20	40	24
24	48	56
28	56	88
32	64	120
36	72	152
40	80	48
44	88	80
48	96	112
52	104	144
56	112	176
60	120	72
64	128	104
68	136	136
72	144	168
76	152	200
80	160	96
84	168	128
88	176	160
92	184	192
96	192	224

Pixels and RGB Encoded Images

To best describe the filter process that this application will employ, we will consider PNG images. At core PNG images are simply a collection of pixels (where a pixel is a small point of color). Each pixel has an

RGBa value where for the sake of this project the alpha transparency values will be set to 1 or fully opaque. Then any image we wish to filter can in essence be considered a large scale collection of pixels where each pixel is a definite RGB value. Based on our data we see how the achromatomaly filter transforms certain pixels that are listed in our test strips. We interpolate linearly all datapoints between these 6-bit increments accross RGB to determine how any RGB value is converted (where there is a total of 256 x 256 x 256 color definitions possible). Based on our data we will want to identify a function say f_achromatomaly(r,g,b) -> (r_achromatomaly, g_achromatomaly, b_achromatomaly), that receives as input any collection of RGB values and outputs the filtered values. We will need to identify this function from the data.

Our Function for the Achromatomaly Filter

Now that we have a collection of data points that describe the Achromatomaly Filter in increments of 64. We can derive a function or function set that models the data. Consider a function $F_{AF}(r,g,b)$ that maps our initial values to our resultant or proposed values. We know that each value in the proposed set is resultant of a composition of the initial r,g,b values. Then to properly describe our function $F_{AF}(r,g,b)$ we actually need three distinct subfunctions for r,g,b respectfully. Then consider a function $F_{AFR}(r,g,b)$ that as input takes r,g,b values and as output produces an r value for achromatomaly. This r value is only a component of F_{AF}. More derictly the r component. Then we devise two more functions in the same fashion for g and b. In effect we have the complete mapping $F_{AF}(r,g,b)$.

To further motivate the process consider $F_{IF}(r,g,b)$ where this function is the identity function for any r,g,b encoding. Below are listed the three components of this identity function.

$$F_{IFR}(r,g,b) = r \; F_{IFG}(r,g,b) = g \; F_{IGB}(r,g,b) = b$$

With Excel we can begin to formulate $F_{AF}(r,g,b)$ by iterating over the r,g,b (proposed) data.

Here are excel functions that precisely model the data.

Please note that working with even numbers was easier when interpolating the test data. Therefore the final functions have an adjustment to make their results valid RGB values.

The adjustment works as follows:
If F_{IFR} or F_{IFG} or F_{IFB} equal 0 then 0
else F_{IFN} equals F_{IFN} - 1 for n in {r, g, b}

F_{IFR} = ROUNDUP(r · 40/64, 0)
+ ROUNDUP(g · 20/64, 0)
+ ROUNDUP(b · 4/64, 0)

F_{IFG} = ROUNDUP(r · 16/64,0)
+ ROUNDUP(g · 40/64,0)

+ ROUNDUP(b · 8/64,0)

F_{IFB} = ROUNDUP(r · 8/64,0)
+ ROUNDUP(g · 24/64,0)
+ ROUNDUP(b · 32/64,0)

So in psuedo code:

$F_{IFR} = \lceil r \cdot 40/64 \rceil + \lceil g \cdot 20/64 \rceil + \lceil b \cdot 4/64 \rceil$
$F_{IFG} = \lceil r \cdot 16/64 \rceil + \lceil g \cdot 40/64 \rceil + \lceil b \cdot 8/64 \rceil$
$F_{IFB} = \lceil r \cdot 8/64 \rceil + \lceil g \cdot 24/64 \rceil + \lceil b \cdot 32/64 \rceil$

Our Implementation for the Achromatomaly Function

Using HTML5 Canvas a simple implementation of the function can be constructed. The code for the filter is presented below as well as a prototype. To view the working filter visit goo.gl/Aasu1u.

```
Filters.achromatomaly = function(pixels, args) {
var d = pixels.data;
    for (var i = 0; i < d.length; i += 4) {
        var r = d[i];
        var g = d[i + 1];
        var b = d[i + 2];

        d[i] = (d[i] === 0) ? 0 : Math.ceil(r * (40 / 64)) + Math.ceil(g * (20 / 64)) + Math.ceil(b * (4 / 64));
        d[i + 1] = (d[i + 1] === 0) ? 0 : Math.ceil(r * (16 / 64)) + Math.ceil(g * (40 / 64)) + Math.ceil(b * (8 / 64));
        d[i + 2] = (d[i + 2] === 0) ? 0 : Math.ceil(r * (8 / 64)) + Math.ceil(g * (24 / 64)) + Math.ceil(b * (32 / 64));
    }
    return pixels;
};
```

QUIC (TIQC) Color Blindness Simulator

The original test image

The original test image filtered by Coblis Simulator (Our Model)

The original test image filtered by QUIC Simulator (Our Application)

Apply Achromatomaly filter to the image

Before Filter Application

QUIC (TIQC) Color Blindness Simulator

The original test image

The original test image filtered by Coblis Simulator (Our Model)

The original test image filtered by QUIC Simulator (Our Application)

Apply Achromatomaly filter to the image

After Filter Application

In Closing

Here in was demonstrated a procedure for determining one of the eight in question filters to simulate color blindness. The complete project entails the creation of a mobile application to give an end user access to the tool and allow them to convert any JPEG or PNG image. At present version 1.0 of the application is available via the Apple iTunes App store and the Google Play store. Version 1.0 has certain known issues including filter quality, yet now that a release workflow is final, all that remains is to tweak the filters accordingly and release an update. The project will continue to progress as time permits.

Thank You to Tech Incubator @ Queens College. Thank You All for listening.

About Philip Parzygnat

Philip Parzygnat holds a BA in Computer Science and is pursuing his MA (Queens College). He has over a decade of experience with technology that started with DIY electronics along with legacy operating systems which later progressed to hardware support, networking, and systems administration then to lecturing and elementary web development then to graphic design along with front end development then to writing and publications and at present to full stack development and dev-ops. Parzygnat works at a NYC financial technology company (Dealflow.com) where he manages the company's internet presence along with marketing and publications infrastructure. Parzygnat has taugh elementary courses at York College and spends a deal of time writing creatively. Beyond technology, he is interested in art, philosophy and mathematics.
http://linkedin.com/in/pparzygnat.