E04: Grayscale

Here's another simple but common shader effect.

```
void main(){
  gl_FragColor = cc_FragColor*texture2D(cc_MainTexture, cc_FragTexC
oord1);

  float gray = (gl_FragColor.r + gl_FragColor.g + gl_FragColor.b)/3
.0;
  gl_FragColor.rgb = vec3(gray);
}
```

It's starts out reading the texture color and tinting it. Then averages the red, green and blue channels together. Lastly, it uses that average value to make a new color. Pretty simple.

This is a simple way to convert to grayscale, but it does not preserve *luminance*, or overall brightness. This is because the human eye is equally sensitive to red, green and blue light and computer displays are calibrated to take advantage of this.

Vector Swizzling and Slicing:

GLSL has a lot of really handy syntax for working with vectors. Hopefully the

meaning of the following example is clear just by looking at it. It creates a 3 component vector that repeats the same value 3 times. Then it replaces the rgb values in `gl_FragColor` with this new vector.

```
float gray = ...;
gl_FragColor.rgb = vec3(gray);
```

More examples!

```
color1.rgb = color.gbr;
position = vec3(vec2(x, y), z);
color3 = vec4(vec2(r, g), vec2(b, a));
color4 = vec4(r, g, color5.ba);
position2.yz = vec2(y, z).
```

That should give you a pretty good idea of what you can do with GLSL vectors.

Exercises:

- Rewrite the shader to apply the tint color after converting to grayscale.
- Read about proper grayscale conversion: http://en.wikipedia.org/wiki/Luminance_(relative).
- See if you can figure out how to use GLSL's `dot()` function to simplify the averaging.