Prakash Paudel
Midterm Project

MATH 397
03/22/2014

Pixel Representation with Tri-Color Triangles

Introduction

Inspired by the use of *truchet tiles* to create two-colored representations of figurative mosaics, this project explores the addition of color into the equation. First and foremost, to represent the full dynamic range of color, our tile must incorporate at-least three colors. In this case, I choose the RGB color model, which uses red, green and blue in combination to represent the full spectrum of color.

Now, the next step was to pick an appropriate 'tile' system for this representation. The equilateral triangle with its simplicity and symmetry presented itself as an elegant option. As we all learned in school, an equilateral triangle is made up of 3 sides of equal length and, as a result, all three vertices of the triangle have an angle of 60^{o}. Furthermore, the bisectors of these three vertices intersect at one single point in the middle, the center 'vertex' of the triangle. As shown in *figure 1*, this vertex is also where the lines perpendicular to the midpoints of all three edges meet.

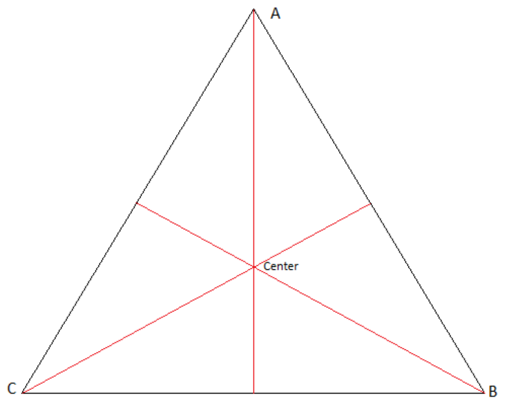
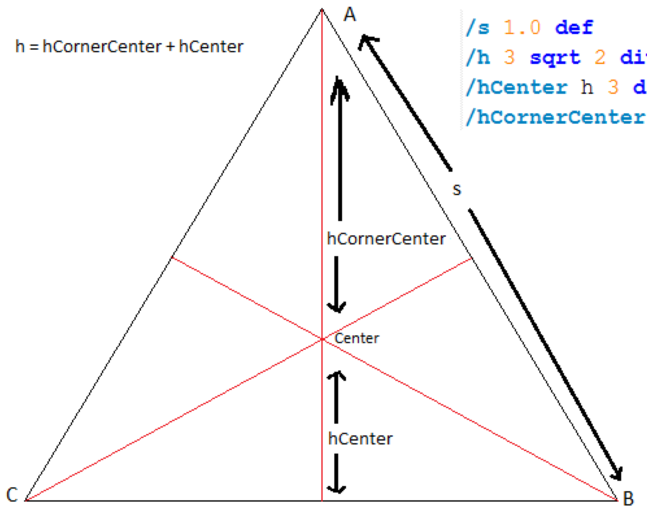


Figure 1: The Symmetry of the Equilateral Triangle

Vertex-based representation

Our first representation, consists of using the center vertex in conjunction with the three outer vertices as the meeting points of the separate colored areas. Using geometry, we see that we can represent all the different lengths of the equilateral triangle in terms of the length of a side, s.

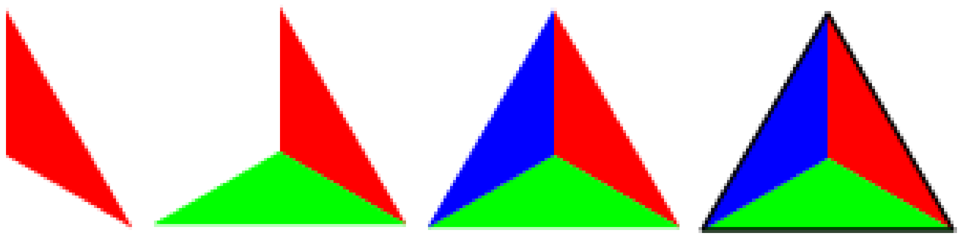


```
/s 1.0 def                              % length of side
/h 3 sqrt 2 div s mul def               % Height = (sqrt(3)/2)*s
/hCenter h 3 div def                    % h/3
/hCornerCenter 2 h 3 div mul def        % 2*(h/3)
```

Figure 2: Global variables and Constants

Note that rotating the above figure by 120^{o} or 240^{o} doesn't change the relationship between the variables due to rotational symmetry. Thus, no further constants are needed for drawing the basic pixel.

Using postscript, we are able to carve a red, green and blue portion of such a triangle as shown in *figure 3*.



```
% Red
r br mul 0 0 setrgbcolor
newpath
dotx doty moveto
dotx doty hCornerCenter add lineto
dotx .5 add doty hCenter neg add lineto
closepath
fill
```

```
% green
0 g br mul 0 setrgbcolor
newpath
dotx doty moveto
dotx .5 add doty hCenter neg add lineto
dotx -.5 add doty hCenter neg add lineto
closepath
fill
```

```
% blue
0 0 b br mul setrgbcolor
newpath
dotx doty moveto
dotx -.5 add doty hCenter neg add lineto
dotx doty hCornerCenter add lineto
closepath
fill
```

Figure 3: Pixel drawing progression

Problems with capturing the spectrum

The next step in the pixel representation is to vary the areas of the red, green and blue so that we can represent more of the color spectrum. Initially, I thought of simply moving the central vertex according to the ratio of the three colors. However, this doesn't allow for the representation of changes in brightness. Thus, I added in a brightness variable to account for that need and was able to vary the brightness of each pixel by scaling the r,g,b values by this brightness variable. The result of this effect is shown in *figure 4*.

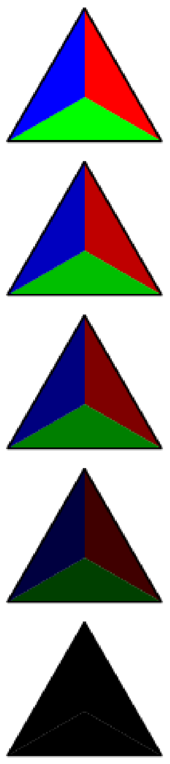


Figure 4: Brightness changes

This modification successfully allows for the representation of black, however several problems remain. First, the maximum brightness shows solid red, green and blue instead of white. Next, the brightness variable scales the red, green and blue sections by the same proportion, thus limiting the scope of representing pixels that may have varied brightness for the rgb values (e.g. bright red, dark blue and dark green). In addition, we notice that using a central vertex to move all three sections together, doesn't allow for enough variation in the ratio of the three colors. These fundamental flaws require the use of a new model that can overcome these problems within the framework of the equilateral triangle.

Independent triangles representation

This model considers each equilateral triangle being the combination of three triangles. Each of these independent triangles consist of two-fixed vertices and a flexible vertex that is adjusted based on the intensity of each color. This requires the addition of an additional global variable to represent the maximum flexibility of the flexible vertex.

```
/bend h 4 div def                   % radius of movement of vertex = h/4
```

This variable defines the maximum deviation of the flexible vertex from the center of the original equilateral triangle. In this case we have defined the maximum bend as h/4, which gives us a variability of the flexible vertex of h/2, half the height of the equilateral triangle. *Figure 5* shows the result of a flexible vertex for an independent triangle embedded into the equilateral triangle.

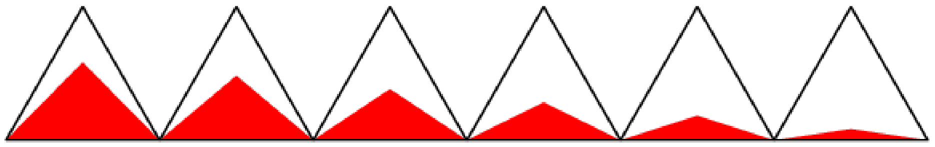


Figure 5: The bend effect

Now, consider overlaying the equilateral triangle with one such independent triangle from each side. Using a postscript function that does exactly that given the coordinates of the center and r,g,b values we get an equilateral triangle that incorporates all three colors as follows:

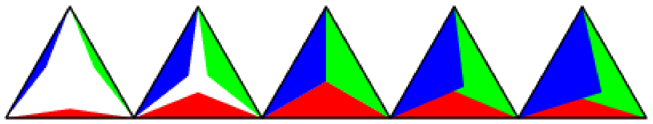


Figure 6: The basic pixel with progression of bend

Problems with capturing the spectrum - revisited

Now, with this new model, we can revisit the issues of the previous model. First off, this model immediately solves issues of the ratios of r,g and b both in terms of brightness and proportion. This is due to the independent movement of each color's triangle. Next, this model looks to address the issue of representation of white through overlapping. In order to display white, we need the overlapping region to be partially transparent. Unfortunately, this is impossible with postscript. Postscript only supports fully opaque or fully transparent filling, it does not allow partial transparency.

Another potential solution to this problem is to find the intersecting paths of the triangles and manually fill that path with the appropriate blends of r,g and b. The figure below shows how these intersections appear.

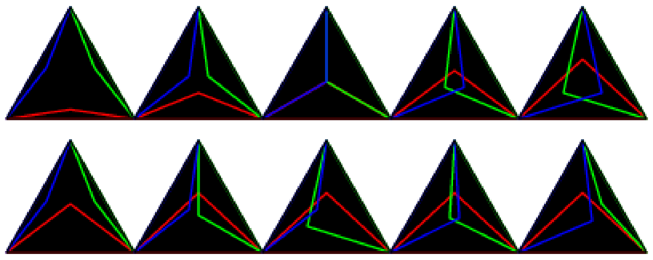


Figure 7: Intersections of the color triangles

Once again, postscript doesn't have a way to calculate the intersecting path of two or more separate paths. This could be done through our own function for triangles where the intensity of red, green and blue is equal. However, in the case of non-equal intensity (e.g. the second row in *figure 7*), it isn't feasible to calculate the intersection paths and color accordingly.

Next Steps

Given the current state of the project and my desire to make this technique work, I plan on trying more things to reach the ultimate goal of a 3-color triangular tiled representation of figurative mosaics. Given this, there are a few different approaches available. The simpler approach would be to relax the triangular constraint and to use a square instead, where the fourth color is white. This would be a workaround to the issues faced with overlapping and intersections on postscript.

However, I believe that there is a possibility of finding a solution for transparency or intersections using a higher level language such as *Java*, *C* or *Python*. Further, the idea of an elegant solution incorporating an equilateral triangle is too enticing to give up. Ultimately, given that I am able to successfully form an appropriate representation using this technique, I will seek to develop addition techniques around the equilateral triangle. One such idea revolves around using circles centered on each vertex or midpoint to represent the three colors. Given a few different techniques, I ultimately would like to compare the results of using these tiles to represent mosaics with other such tiling systems including the flexible Truchet tiles technique.