Computer Graphics Coursework 1 Report

Assignment 1

For a line, L, given by the values $x1,\ y1,\ x2,\ y2$ where each is an x or y coordinate for the two ends of the line, the parameterized equation for the line is

$$L = t(x1, y1) + (1 - t)(x2, y2)\ where\ 0 \leq t \leq 1$$

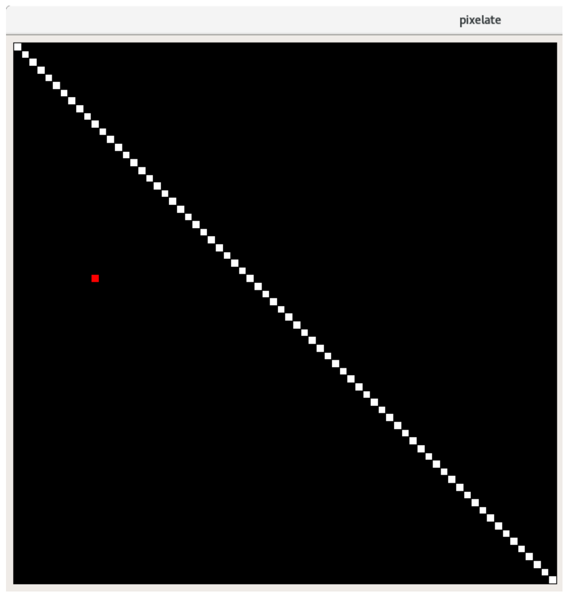
So the x and y coordinates for any point on the line can be found by:

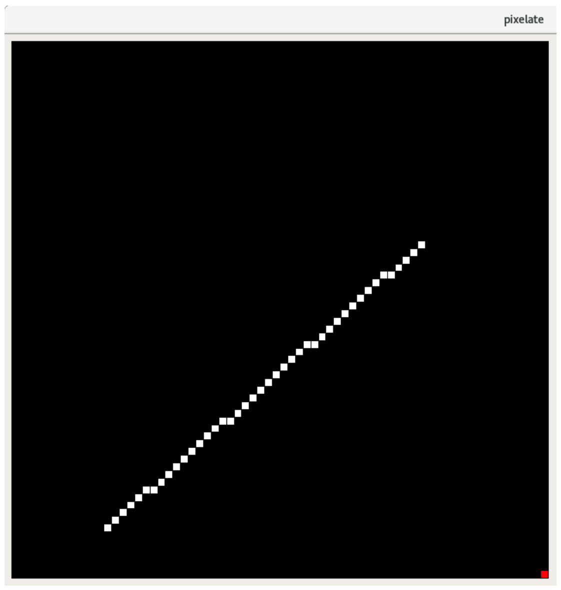
$$L_x = tx1 + (1 - t)x2,\ L_y = ty1 + (1 - t)y2\ \ where\ 0 \leq t \leq 1$$

Which is the equations I have used in my method 'DrawLine()' (Lines 27-31). To map a line the equations above must be done for a range of values of t such that $0 \leq t \leq 1$, with more values in that range giving more resolution. By doing this in a for loop over the variable t between 0 and 1, I can define the step size as the increment.

For the number of values of t that I used, I decided to calculate the distance between the x coordinates of the line, and the distance between the y coordinates, and take the larger of the two. This allowed me to ensure that any line will be drawn completely as it cannot have more pixels than it's height/width. (Line 24)

Screenshots of lines (0,0) to (70,70) and (12,63) to (54,25)



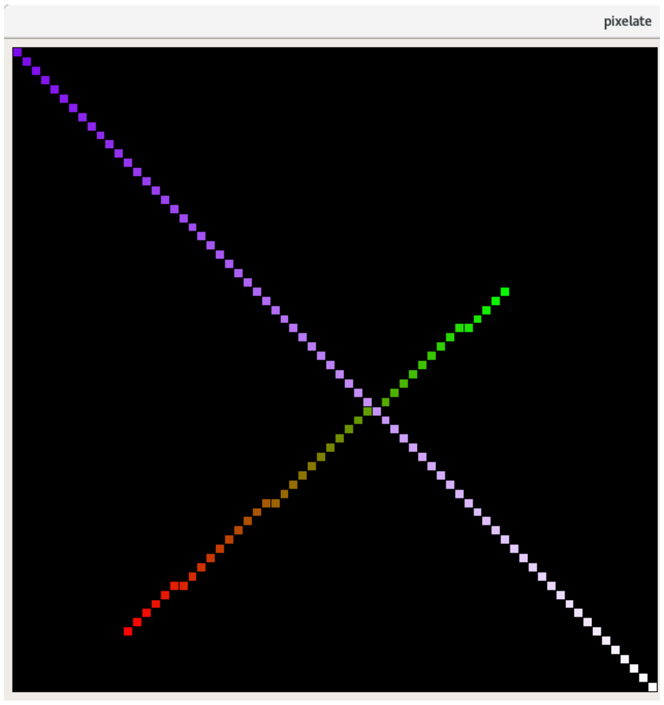


Assignment 2

To interpolate the colour values I decided to treat the two RGBVal variables as points in 3D space, where each of the red, green and blue are the positions on each axis. By doing this I am able to interpolate them in the same was that I did with the two ends of the line, using the same values of t. This meant that at each value of t, I got a colour that is midway between the two given RGB values. (Lines 34-36)

Screenshot shows the lines:

- (12,63) to (54,25) with end colours (255,0,0) and (0,255,0)
- (0,0) to (69,69) with the end colours (125,10,236) and (255,255,255)



Assignment 3

By using barycentric coordinates, a triangle, T, with corners at (x_1, y_1), (x_2, y_2), (x_3, y_3) can be represented as:

$$T = s(x_1, y_1) + t(x_2, y_2) + (1 - s - t)(x_3, y_3) \; where \; 0 \le s + t \le 0, \; t \ge 0, \; s \ge 0$$

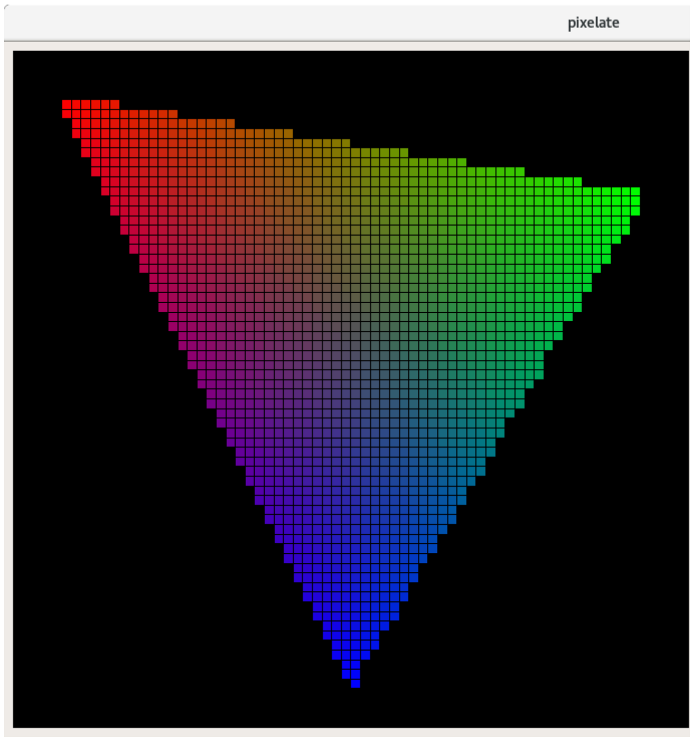
So, like with the parameterized line, the (x, y) coordinates of any point within T are:

$$T_x = sx_1 + tx_2 + (1 - s - t)x_3, \; T_y = sy_1 + ty_2 + (1 - s - t)y_3 \; where \; 0 \le s + t \le 0, \; t \ge 0, \; s \ge 0$$

This we can loop over a range of (s, t) values to find a set of pixels within T. If we do this with enough values of (s, t) and draw each resultant (x, y) then we will have drawn T (Lines 56-65).

In order to find how many different values of s and t should be used, I decided to find what was greater out of the width and the height of the triangle. To do this I found the maximum distance between any pair of the x coordinates, and the same for the y coordinates, then took the larger of these two values as the number of steps. (Lines 49-53)

Screenshot shows the triangle with corners (5,5), (65,15), (35,65) with red, green and blue being the corner colours respectively.



Assignment 4

To implement the IsInside() function I first calculated the x and y components of the vector for each edge of the triangle, and then the normal to each of those vectors. (Lines 76-93)

After this I used the equation $f(X) = (X - P) \cdot n$ where X is the point being checked, P is a point on the line and n is the normal to the line QP. The result of this tells me whether the point X is above or below the line QP. If the point is above all three edges, then it must be inside the triangle. (Lines 98-100)

Because I am using C++ without any vertices objects I have to consider x and y components separately. Therefore for each point (x,y): $f(x,y) = ((x - P_x) * n_x) + ((y - P_y) * n_y)$.

This will give a result for each edge and if all three are greater than or equal to 0, the point is inside the triangle. (Lines 103)

I also created a function (CheckTriangle()) to loop over all pixels in the viewport, calculate the barycentric coordinates of the pixel relative to the triangle, then call IsInside() with the triangle and that pixel, printing the results and coordinates to a file (output.txt) in the format "IsInside():alpha:beta". (Method starts on line 110)

To calculate the barycentric coordinates of a point (x,y) with respect to the triangle: (x_1,y_1), (x_2,y_2), (x_3,y_3) I used the equation:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} (x_1 - x_3) & (x_2 - x_3) \\ (y_1 - y_3) & (y_2 - y_3) \end{bmatrix} \begin{bmatrix} x - x_3 \\ y - y_3 \end{bmatrix}$$

Where alpha and beta are the barycentric coordinates of (x,y). (Lines 116-134)

To check the output of the IsInside() function I created a simple python script (output_check.py) that goes through and check that the statement below is true:

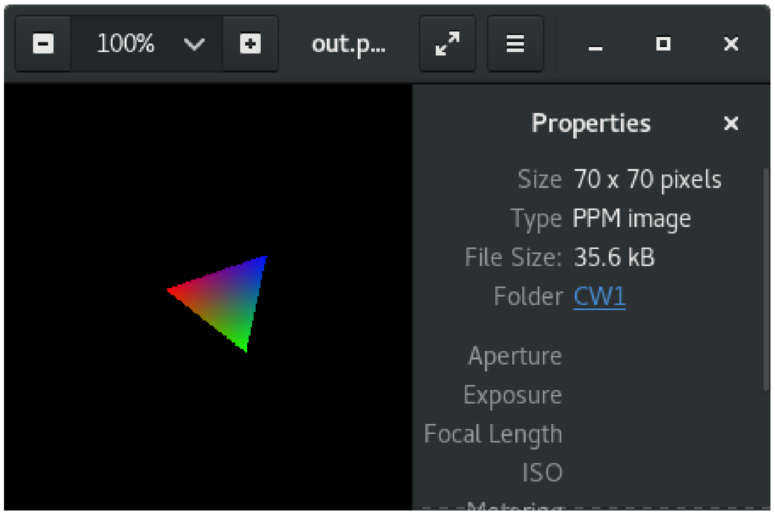
- Iff the pixel is inside the triangle, then the barycentric coordinates (alpha and beta) satisfy the conditions: alpha >= 0, beta >= 0, alpha + beta <= 0.
- Iff meaning that the inverse is also true; If alpha and beta satisfy the conditions stated, then the pixel must also be in the triangle.

When I ran this script on the output.txt file and it returned true, meaning that the statement above is in fact correct for all pixels in the image. To ensure that the script actually works, I changed a few of the truth values in the output.txt file and ensured that with the erroneous data that the script outputted false, which it did.

Assignment 5

For assignment 5 i created a method 'OutputPPM' to create the PPM file with a name given as a parameter, which uses the class variables to fill out the header of the ppm file, then loop over every pixel and find it's colour components, writing each to the file in the correct format. I set the 'magic' value of the PPM header to 'P3' as this allows me to write the file as strings, which is easier and the performance penalty is not too large with this size image.

The output file:



Size 70 x 70 pixels
Type PPM image
File Size: 35.6 kB
Folder CW1
Aperture
Exposure
Focal Length
ISO