TinyRaster - Report

In this report, I will discuss my approach to the TinyRaster assignment, and assess my solution's results.

My initial approach to drawing 2D lines involved refactoring an existing implementation of the Bresenham algorithm (Flanagan, 2018) from a previous project (Chen, ASCiiDraw).

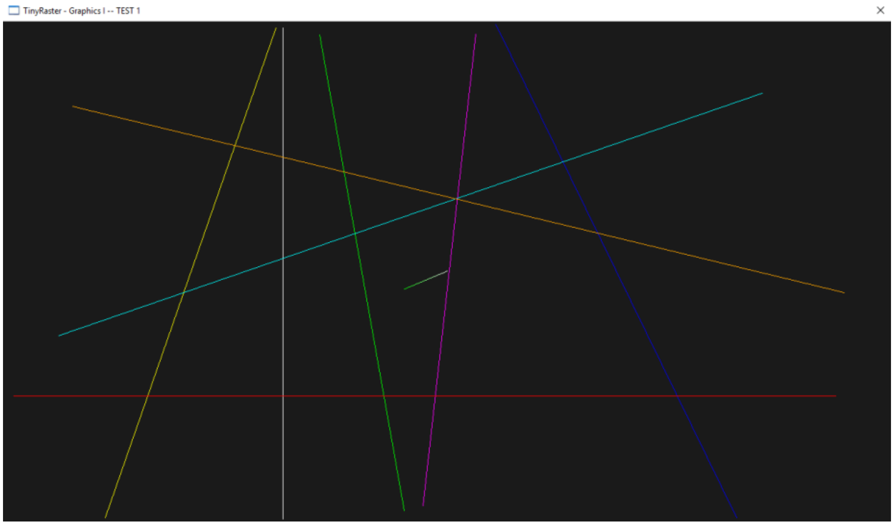

Figure 1 - Drawing standard lines

For line thickness, my initial approach involved adding extra pixels next to the pixel being drawn based on the given thickness value. The final version determines which axis to draw thickness pixels on, and ensures these are distributed evenly on either side of the current pixel.

Colour interpolation for lines incorporates the interpolation formula from our lecture (see Figure 2).

If P, P_0 and P_1 are known, then
$t = \frac{|P_0 - P|}{|P_1 - P_0|}$
Therefore the colour for P is:
$C = t * C_1 + (1 - t) * C_0$

Figure 2 - Chen, Colour and Interpolation

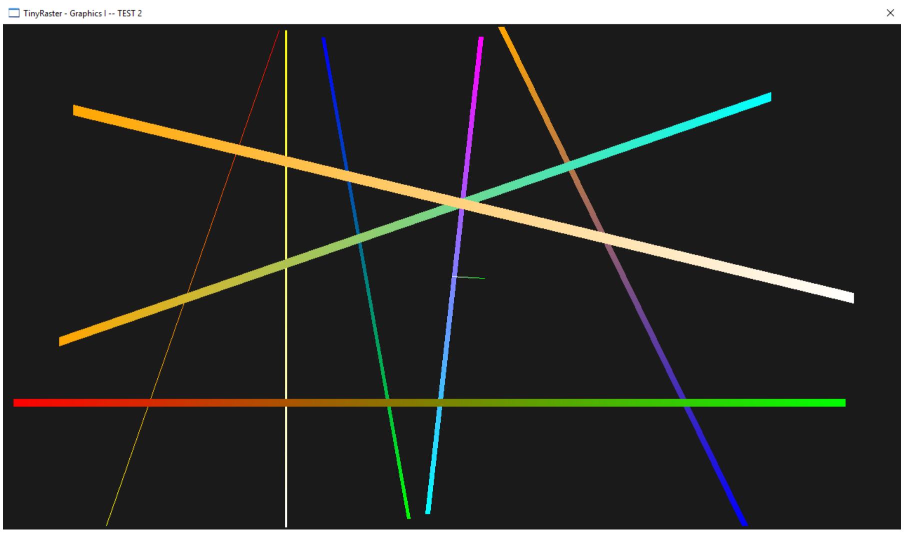

Figure 3 - Drawing lines with thickness and colour interpolation

Implementing unfilled polygon drawing simply involved iterating over the given vertices and drawing lines between each point in order. There's a check for the last point in the array that joins it up with the first.

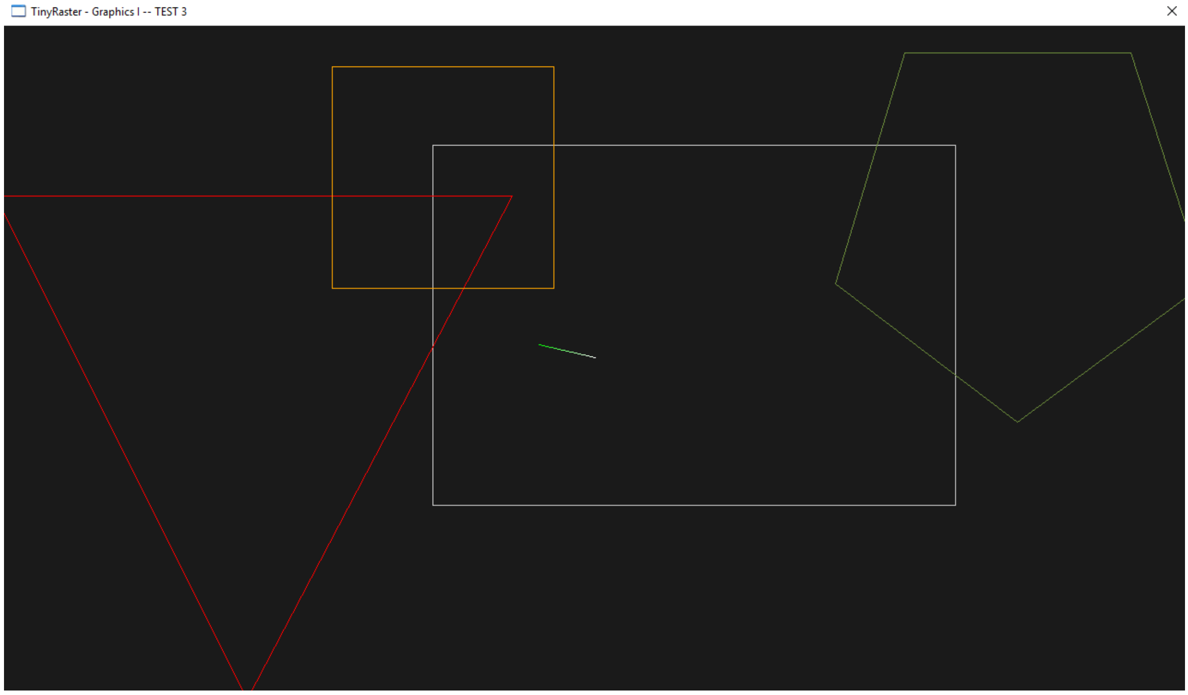

Figure 4 - Unfilled simple polygons

For solid polygon filling, I originally used the second LUT (see Figure 5) filling method from our tutorial (see Figure 6).

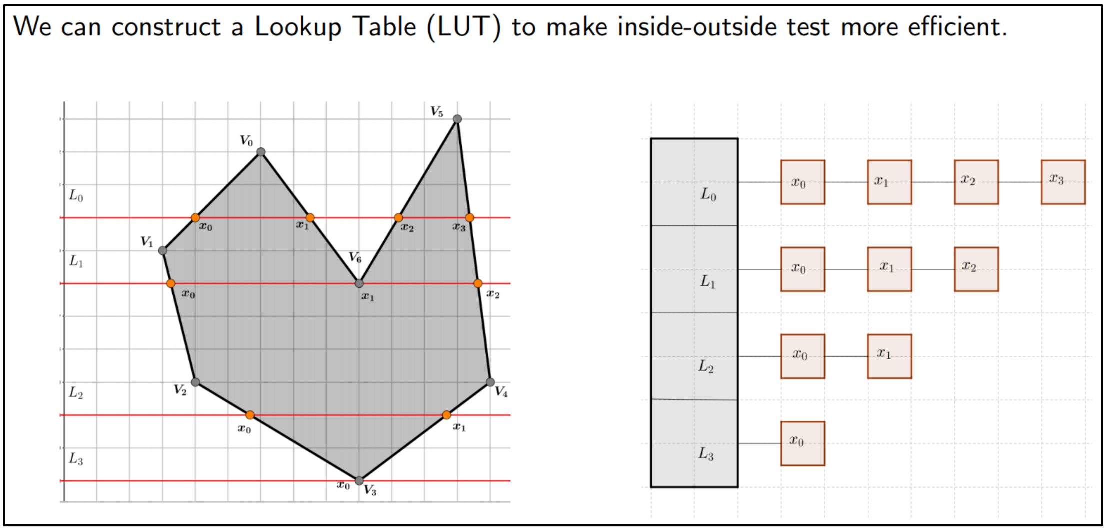

Figure 5 - Chen, Polygon Filling

```
void DrawLine2D(const Vertex2d &v1, const Vertex2d &v2, int thickness)
{
   ...
   //Given a point x,y and its colour
   ScanlineLUTItem newitem = {colour, x};
   mScanlineLUT[y].push_back(newitem);
   ...
}
```

Figure 6 - Chen, Scanline Filling Polygons

This meant populating the LUT when drawing the outline of each shape. This worked well for solid, interpolated, and circle filling. However, this made complex polygon filling unstable. Issues arose due to multiple points being added to the same scanline for each line being drawn. After attempting to work with this, no variation could fill complex polygons perfectly. I then transitioned to using the formula in the first method from our tutorial (see Figure 7) to populate the LUT.

Given two vertices of an edge, $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, and the i-th scanline $y = i$. The edge intersects with the scanline if $y_1 \leq y \leq y_2$ or $y_1 \geq y \geq y_2$. The x coordinate of the edge point on the scanline is then given by:

$$x = x_1 + (y - y_1)\frac{x_1 - x_2}{y_1 - y_2} \tag{1}$$

The location (x, y) can now be added to the LUT, e.g.

```
ScanlineLUTItem newitem = {colour, x};
mScanlineLUT[y].push_back(newitem);
```

Figure 7 - Chen, Scanline Filling Polygons

A vector is populated with certain points from the given shape that should be treated uniquely when filling, and lines are drawn between pairs of points in each scanline, taking into account the points in the vector. I ensured this solution was robust by adjusting the points of existing tests, confirming that it could still fill correctly.

Figure 8 - Filled simple polygons

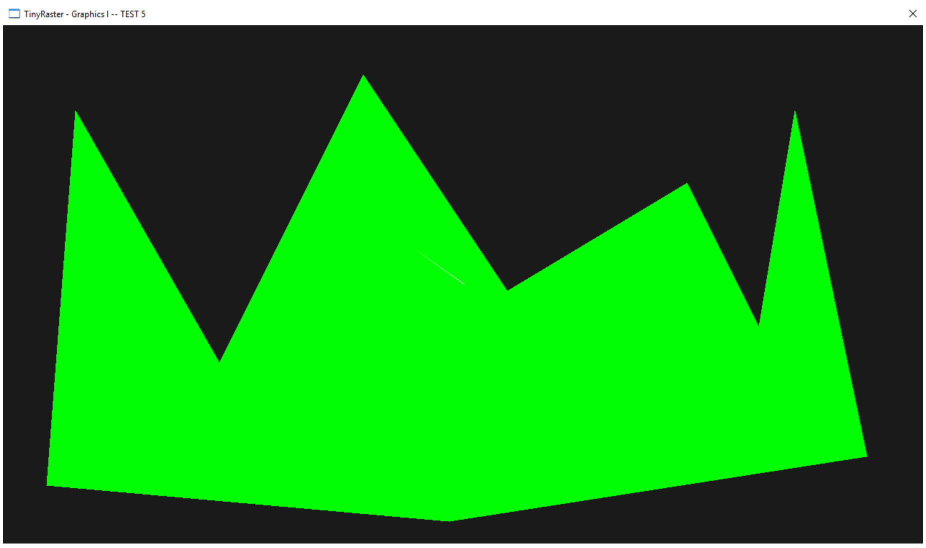

Figure 9 - Filled complex polygon

Alpha blending occurs when pixels are being added to the framebuffer. The equation found in our lecture (see Figure 10) is used to calculate the new colour.

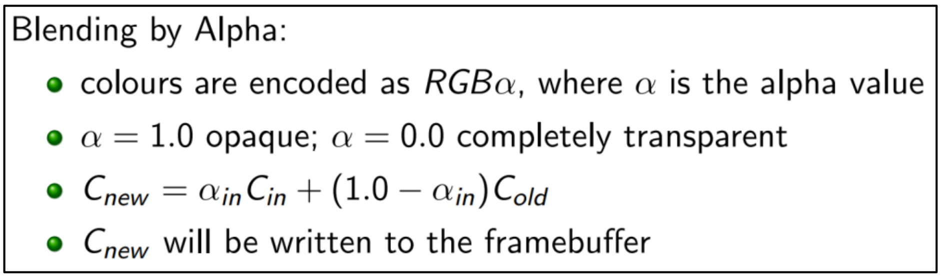

Figure 10 - Chen, Colour Blending

Figure 11 – Polygon alpha blending

For interpolated filling, the standard filling method checks if the fill mode is set to interpolated, and if so, a modified version of the line drawing's colour interpolation is used to calculate the colour of the point being added to the LUT. The interpolated fill method simply calls the standard fill method using the given shape vertices, meaning the standard filling code doesn't need to be unnecessarily repeated.

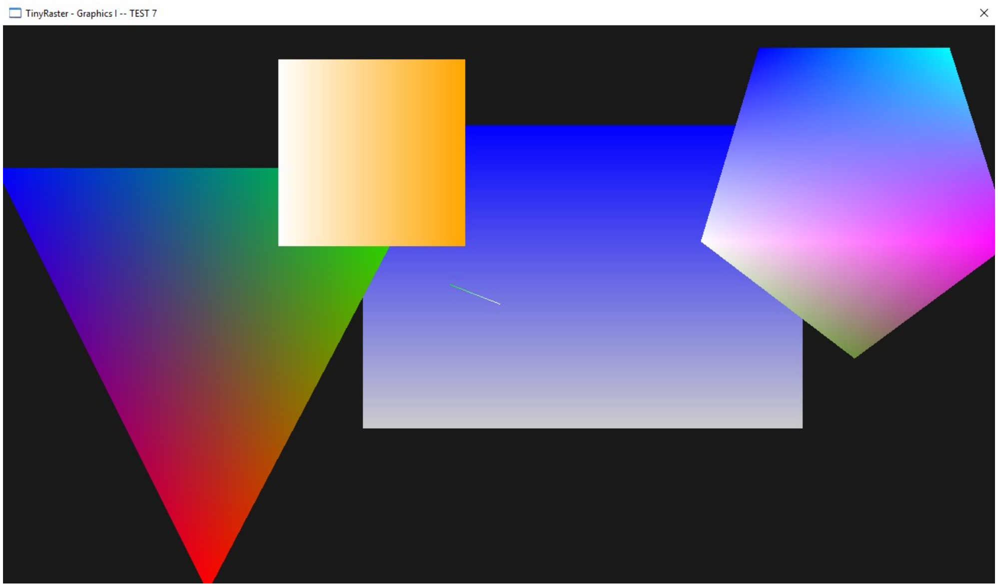

Figure 12 - Polygon colour interpolation

For circle drawing, I initially drew circles with a set number of vertices, using the ‘parametric form’ equation found in our lecture (see Figure 13).

- A circle with radius r centred at $C(c_x, c_y)$ can be written in parametric form

$$x = rcos(t) + c_x$$
$$y = rsin(t) + c_y$$

- where $t \in [0, 2\pi)$

Figure 13 - Chen, Polygons and Circles

The generated vertices are passed to the unfilled or filled polygon drawing methods depending on the given ‘fill’ value. The set number of vertices was later replaced with the circle’s circumference in order to make the circle’s edge as smooth as possible without unnecessary vertices being calculated. Finally, the two/four/eight-way symmetry optimisation suggested in our lecture (see Figure 14) was implemented.

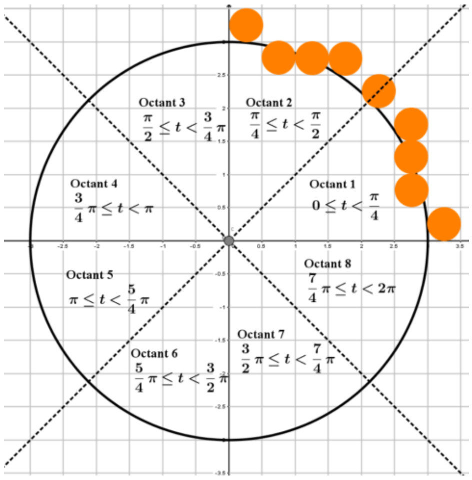

Figure 14 - Chen, Polygons and Circles

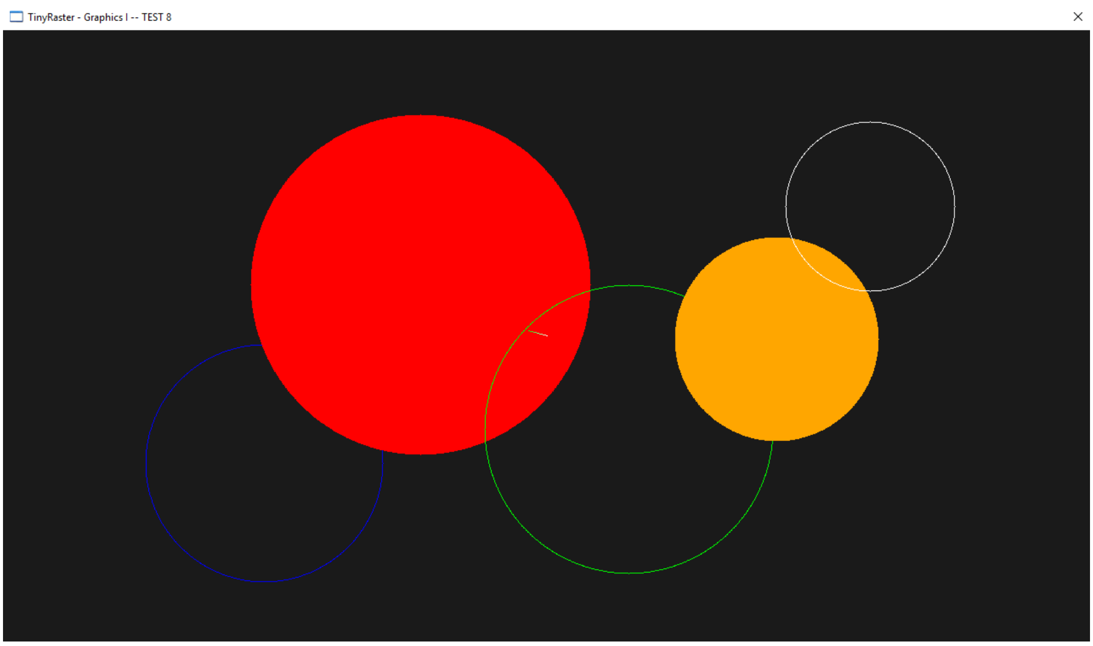

Figure 15 - Unfilled and filled circles

Overall, the strengths of my solution are that all tests perform as intended, and the code has been kept optimised. A weakness is that the original method of LUT filling seemingly produced smoother edges, due to it utilising the Bresenham algorithm (Flanagan, 2018).

References

Chen, M. (n.d.). *ASCiiDraw Project.* Retrieved from UniLearn: https://unilearn.hud.ac.uk/bbcswebdav/pid-2323330-dt-content-rid-3645963_1/xid-3645963_1

Chen, M. (n.d.). *Colour and Interpolation.* Retrieved from UniLearn: https://unilearn.hud.ac.uk/bbcswebdav/pid-2334072-dt-content-rid-3679848_1/courses/CFT2112-1718/CFT2112_Lec16.pdf

Chen, M. (n.d.). *Colour Blending.* Retrieved from UniLearn: https://unilearn.hud.ac.uk/bbcswebdav/pid-2335227-dt-content-rid-3684517_1/courses/CFT2112-1718/CFT2112_Lec17.pdf

Chen, M. (n.d.). *Polygon Filling.* Retrieved from UniLearn: https://unilearn.hud.ac.uk/bbcswebdav/pid-2332420-dt-content-rid-3675198_1/courses/CFT2112-1718/CFT2112_Lec16-s.pdf

Chen, M. (n.d.). *Polygons and Circles.* Retrieved from UniLearn: https://unilearn.hud.ac.uk/bbcswebdav/pid-2332419-dt-content-rid-3675193_1/courses/CFT2112-1718/CFT2112_Lec15-s.pdf

Chen, M. (n.d.). *Scanline Filling Polygons.* Retrieved from UniLearn: https://unilearn.hud.ac.uk/bbcswebdav/pid-2334073-dt-content-rid-3679849_1/courses/CFT2112-1718/scanline_fill.pdf

Flanagan, C. (2018, April 26). *The Bresenham Line-Drawing Algorithm*. Retrieved from University of Helsinki: https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html