Computer Graphics Coursework 1 Report

Brooklyn Mcswiney

Contents

1 Implementation

1.1 Drawing Pixels

Within the window the coordinates count from the top left of the window to the bottom right. This means that the pixel (0, 0) is in the top left corner, (w-1, 0) is in the top right corner and (0, h-1) is in the bottom left corner.

Figure 1: Particle field from the project implementation

1.2 Drawing Lines

The line implementation works by starting with two points. The start and end of the line. The program will then take these values and calculate 2 numbers d_x and d_y. Then, the program will find which axis the line moves the most along (Known as the X or Y major). This is done through seeing which value between d_x and d_y is the greatest. Should d_x be greater we step along the X axis. The same is true should d_y be greater. The program then keeps track of this by saving a 'step' variable as the corresponding major.

The final step before moving onto deciding which pixels get drawn is to calculate the increase required each iteration in the x and y directions. This increase is found by dividing d_x and dy by the step variable that was saved earlier.

In order to draw the line the program now iterates over as many 'steps' needed to draw the line from the start to the end. During each iteration the program will check that the pixel is within the bounds of the surface and should this check return true the pixel will be drawn. This is done to avoid out of bounds errors within the program. The last stage of the loop is to move to the next pixel in the line by adding the x and y increase to our current pixel location.

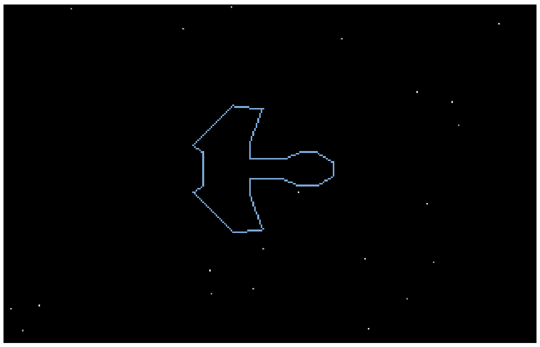

Figure 2: Ship drawn.

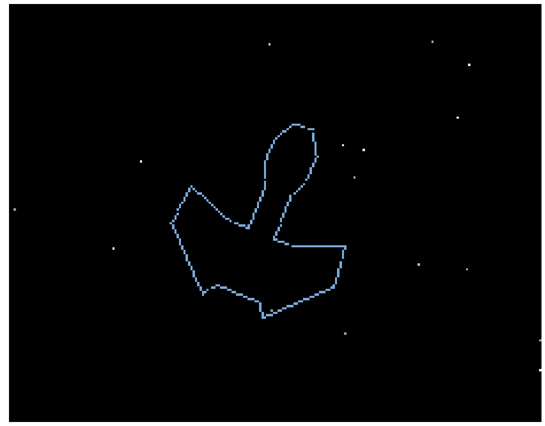

Figure 3: Ship rotated

1.3 Drawing Triangles

This algorithm begins by finding the possible bounds of the triangle by iterating through the points given to find the maximum and minimum coordinates. Once the bounds of the triangle are discovered the program will iterate through every pixel within the bounds. Every loop the program will check if the current pixel (X) is within the triangle by performing 3 half-plane tests. These tests are performed using the following algorithm $F(X) = n \cdot (X - P)$. Should $F(X) > 0$ be true for any of the lines the program continues onto the next pixel. Should the pixel pass all the half-plane tests the program will then check the pixel is within the window and if it is the pixel gets drawn.

1.4 Barycentric Interpolation

The algorithm begins exactly the same as the previous triangle algorithm, by finding the bounds of the triangle to be drawn. The algorithm then loops through this rectangle and performs calculations to determine whether or not a pixel is within the triangle to be drawn. The changes from the previous algorithm begin with these checks. These checks involve finding α, β and γ values for a given point. These values are known as Barycentric Coordinates and can be found for any given point (U) on a triangle.

Given a triangle with points A, B and C the program need to find the vectors $u = B - A$ and $y = C - A$. Once these two vectors have been found the program then uses the following formula to find the area of the triangle ABC.

$$area(ABC) = \frac{(u_x \times v_y) - (u_y \times v_x)}{2}$$

The code can then call a function for this formula for each of the three sub triangles. Once each sub area has been calculated the program can then perform the following formulas to find the values of α, β and γ.

$$\alpha = \frac{area(BCU)}{area(ABC)}, \beta = \frac{area(AUC)}{area(ABC)}, \gamma = \frac{area(ABU)}{area(ABC)}$$

Finally, should each of these values be between 0 and 1 the algorithm will draw a pixel at the given point. This pixel shall be a colour interpolated from the three points on the triangle using the following formula for each RGB value.

$$colour = A_{colour} \times \alpha + B_{colour} \times \beta + C_{colour} \times \gamma$$

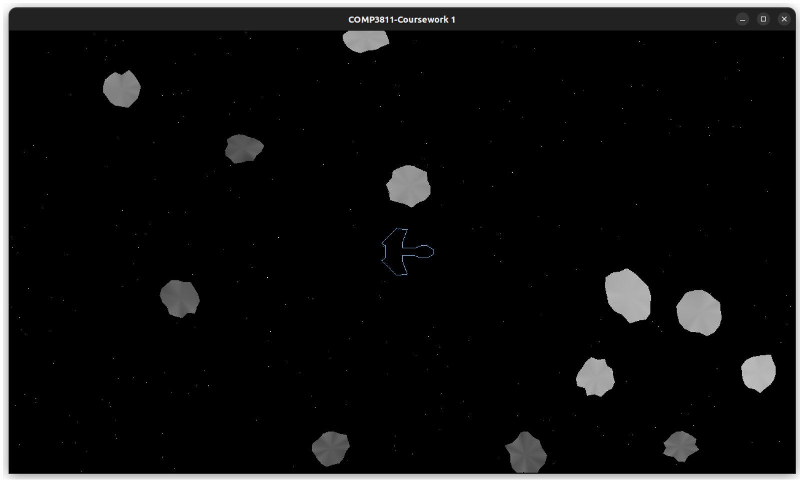

Figure 4: Asteroids drawn using Barycentric Interpolation

1.5 Blitting Images

Helper Algorithims

These helper algorithms differ slightly from the ones used in order to set pixels. This is due to the fact that they use RGBA representation as opposed to RGBx representations. This means that as well as returning the RGB values when getting a pixel the program must also return an A value. This A value is used to indicate the opacity of a pixel which is then later used to check whether a pixel should or should not be drawn on the surface from the image.

Main Blitting Algorithm

The blitting algorithm begins by starting a loop through the image to be rendered. Within this loop the program first finds the current position on the surface by adding the starting position to the x and y values being used to loop through the image. The algorithm then calls the helper algorithm to get a pixel from the image. The program then checks if the current pixel should be masked or not by checking if the A value is greater than 0. The program then makes sure that the current pixel is within the bounds of the surface and if it is the pixel gets drawn onto the surface.

One possible optimisation that could be made is improving the clipping of the image. For example, currently the program will check every pixel outside the bounds of the surface even if no more pixels could possibly be on the surface. This could be further optimised by stopping the current loop if the following pixels will always be outside the borders of the surface.

2 Testing

2.1 Testing: Lines

2.2 Testing: Triangles

3 Benchmarks

3.1 Blitting

3.2 Line drawing

4 My own spaceship