Introduction

We are living in a world where good and attractive graphics are more important than anything, like you have created an working program but if it is not attractive than there is high probability that less people will use it. Current scenarios are that like if you are playing a game, the quality of that game is judge on the basis that how good are the graphics of that game. So computer graphics are an important part of our day-to-day real life like while playing games or animations which we see etc. So here what we will be doing is using the basics function like plotting point we will try to make the whole animation of moving ship without using any of the inbuilt function for translate or color filling or etc.

So what all things are in the basic project?

For line drawing – mid-point line algorithm

For circle drawing – mid-point circle algorithm

For color filling – Scanline color fill algorithm

For plotting points – OpenGL inbuilt function

So using all this things we can say at the end of the project you will have an beautiful project to demonstrate and have a start in computer graphics.

Mid – Point Line Drawing Algorithm

Given coordinate of two points A(x1, y1) and B(x2, y2) such that x1 < x2 and y1 < y2. The task to find all the intermediate points required for drawing line AB on the computer screen of pixels. Note that every pixel has integer coordinates.

In Mid-Point algorithm we do following (**E** stands for East and **NE** stands for North-East).:

- Find middle of two possible next points. Middle of E(X_p+1, Y_p) and NE(X_p+1, Y_p+1) is M(X_{p+1}, Y_p+1/2).
- If M is above the line, then choose E as next point.
- If M is below the line, then choose NE as next point.

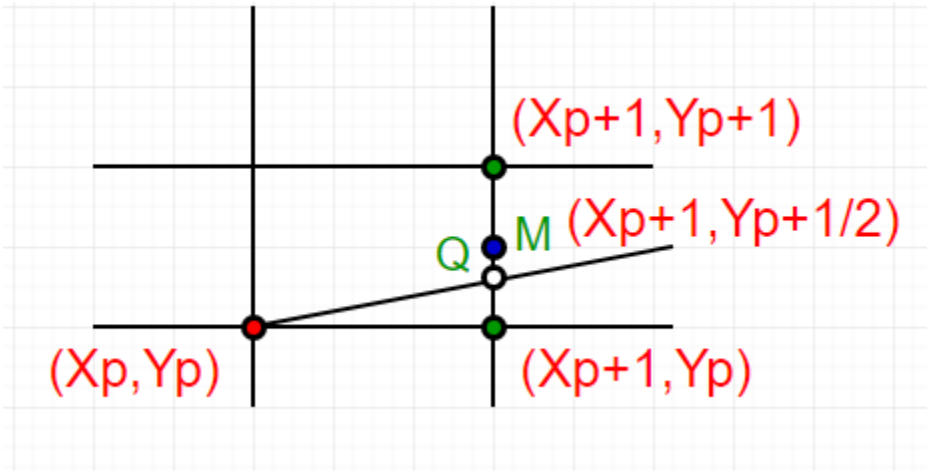

Figure 1

How to find if a point is above a line or below a line?

Below are some assumptions to keep algorithm simple.

- We draw line from left to right
  .
- x1 < x2 and y1< y2
- Slope of the line is between 0 and 1. We draw a line from lower left to upper right.

Cases other than above assumptions can be handled using reflection

Let us consider a line ***y = mx + B.***

We can re-write the equation as :

y = (dy/dx)x + B or

(dy)x + B(dx) - y(dx) = 0

Let ***F(x, y) = (dy)x - y(dx) + B(dx)*** -----(1)

Let we are given two end points of a line (under above assumptions)

- For all points (x,y) on the line, the solution to F(x, y) is 0.
- For all points (x,y) above the line, F(x, y) result in a negative number.
- And for all points (x,y) below the line, F(x, y) result in a positive number.

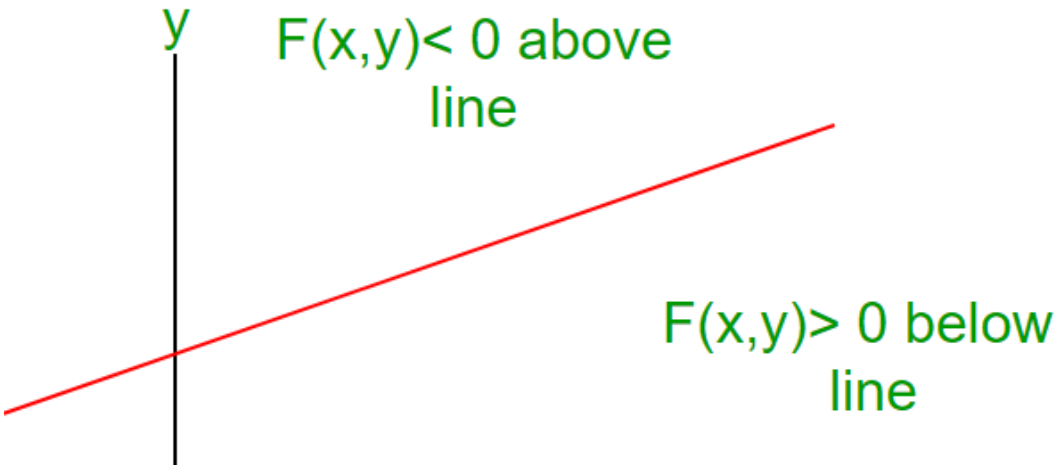

Figure2

This relationship is used to determine the relative position of M

M = (Xp+1, Yp+1/2)

So our decision parameter d is,

d = F(M) = F(Xp+1, Yp+1/2)

How to efficiently find new value of d from its old value?

For simplicity, let as write F(x, y) as *ax + by + c*.

Where $a = dy$

$b = -dx$

$c = B*dx$

We got these values from above equation (1)

Case 1: If E is chosen then for next point :

$dnew = F(Xp+2, Yp+1/2) = a(Xp+2) + b(Yp+1/2) + c$

$dold = a(Xp+1) + b(Yp+1/2) + c$

Difference (Or delta) of two distances:

$DELd = dnew - dold = a(Xp+2) - a(Xp+1) + b(Yp+1/2) - b(Yp+1/2) + c - c$

$= a(Xp) + 2a - a(Xp) - a$

$= a.$

Therefore,

$dnew = dold + dy.$ (as $a = dy$)

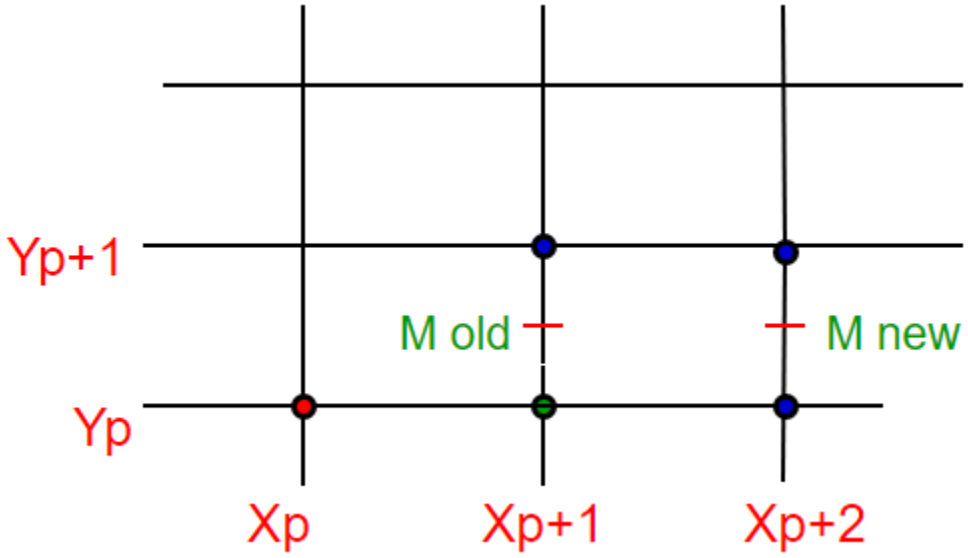

Figure 3

Case 2: If NE is chosen then for next point :

$dnew = F(Xp+2, Yp+3/2) = a(Xp+2) + b(Yp+3/2) + c$

$dold = a(Xp+1) + b(Yp+1/2) + c$

Difference (Or delta) of two distances:

$DELd = dnew - dold = a(Xp+2) - a(Xp+1) + b(Yp+3/2) - b(Yp+1/2) + c - c$

$= a(Xp) + 2a - a(Xp) - a + b(Yp) + 3/2b - b(Yp) - 1/2b$

$= a + b$

Therefore,

$$dnew = dold + dy - dx. \text{ (as } a = dy, b = -dx)$$

Calculation For initial value of decision parameter d0:

$$d0 = F(X1+1, Y1+1/2) = a(X1 + 1) + b(Y1 + 1/2) + c$$
$$= aX1 + bY1 + c + a + b/2$$
$$= F(X1,Y1) + a + b/2$$
$$= a + b/2 \text{ (as } F(X1, Y1) = 0)$$

$$d0 = dy - dx/2. \text{ (as } a = dy, b = -dx)$$

Algorithm:

```
Input (X1,Y1) and (X2,Y2)

dy = Y2- Y1
dx = X2 - X1
// initial value of
// decision parameter d
d = dy - (dx/2)
x = X1 , y = Y1
// plot initial given point
Plot(x , y)

// iterate through value of X
while(x < X2)
                    x = x+1
                    // 'E' is chosen
                    if (d < 0)
                              d = d + dy
                    // 'NE' is chosen
                    else
                              d = d + dy - dx
                    y = y+1
                    Plot(x,y)
```

Mid Point Circle Drawing

The algorithm is very similar to the Mid-Point Line Generation Algorithm. Here, only the boundary condition is different.
For any given pixel (x, y), the next pixel to be plotted is either **(x, y+1)** or **(x-1, y+1)**. This can be decided by following the steps below.

- Find the mid-point **p** of the two possible pixels i.e (x-0.5, y+1)
- If **p** lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it's outside we plot the pixel (x-1, y+1)

Boundary Condition : Whether the mid-point lies inside or outside the circle can be decided by using the formula:-

Given a circle centered at (0,0) and radius r and a point p(x,y)

$$F(p) = x2 + y2 - r2$$

- if F(p)<0, the point is inside the circle
- F(p)=0, the point is on the perimeter
- F(p)>0, the point is outside the circle

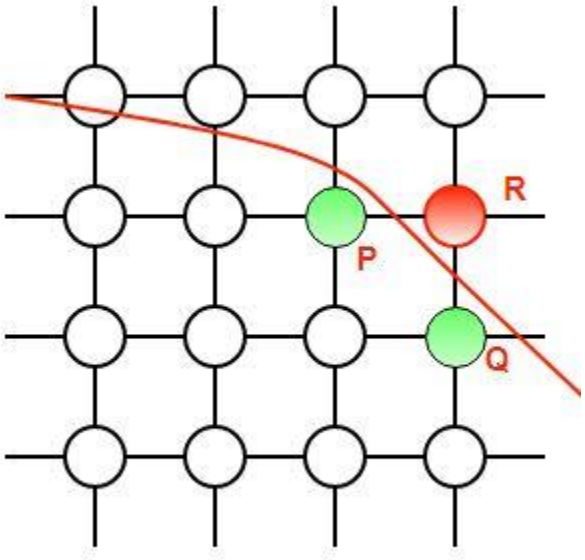

Figure 4

In our program we denote F(p) with P. The value of P is calculated at the mid-point of the two contending pixels i.e. (x-0.5, y+1). Each pixel is described with a subscript k.

$$Pk = (Xk — 0.5)2 + (yk + 1)2 – r2$$

Now,

$$xk+1 = xk \text{ or } xk-1 \text{ , } yk+1 = yk + 1$$

$$\therefore Pk+1 = (xk+1 – 0.5)2 + (yk+1 +1)2 – r2$$
$$= (xk+1 – 0.5)2 + [(yk +1) + 1]2 – r2$$
$$= (xk+1 – 0.5)2 + (yk +1)2 + 2(yk + 1) + 1 – r2$$

$$= (xk+1 – 0.5)2 + [– (xk – 0.5)2 + (xk – 0.5)2] + (yk + 1)2 – r2 + (yk + 1) + 1$$

$$= Pk + (xk+1 – 0.5)2 – (xk – 0.5)2 + 2(yk + 1) + 1$$

$$= Pk + (x2k+1 – x2k)2 + (xk+1 – xk)2 + 2(yk + 1) + 1$$

$= Pk + 2(yk +1) + 1$, when Pk <=0 i.e the midpoint is inside the circle ($xk+1 = xk$)

$Pk + 2(yk +1) – 2(xk – 1) + 1$, when Pk>0 I.e the mid point is outside the circle($xk+1 = xk-1$)

The first point to be plotted is (r, 0) on the x-axis. The initial value of P is calculated as follows:-

$$P1 = (r – 0.5)2 + (0+1)2 – r2$$
$$= 1.25 – r$$
$= 1 - r$ (When rounded off)

Scanline Polygon Fill Algorithm

Scanline filling is basically filling up of polygons using horizontal lines or scanlines. The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure. To understand Scanline, think of the image being drawn by a single pen starting from bottom left, continuing to the right, plotting only points where there is a point present in the image, and when the line is complete, start from the next line and continue.
This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.

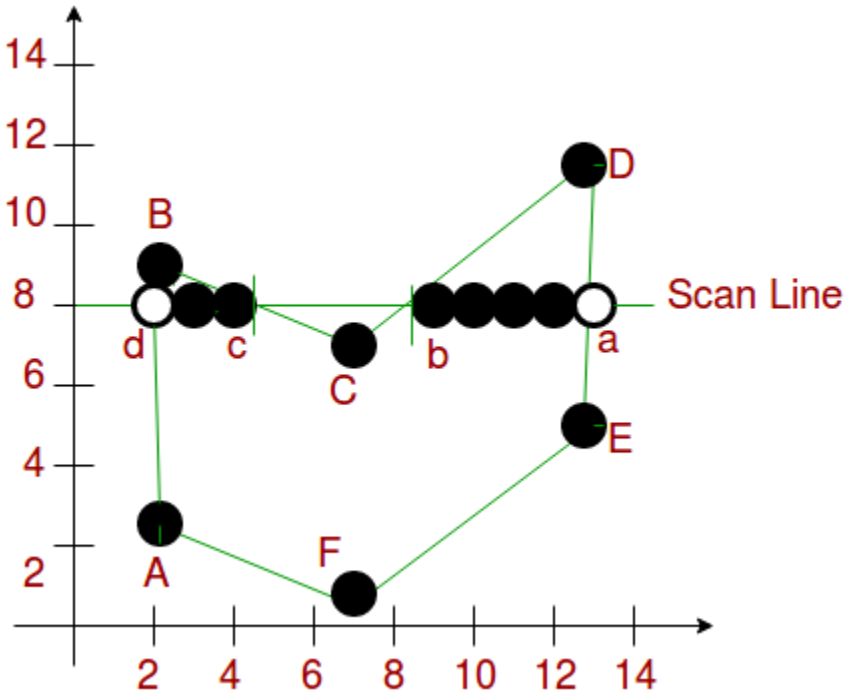

Figure 5

Special cases of polygon vertices:

- If both lines intersecting at the vertex are on the same side of the scanline, consider it as two points.
- If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point.

Components of Polygon fill:

- **Edge Buckets:** It contains an edge's information. The entries of edge bucket vary according to data structure you have used.In the example we are taking below, there are three edge
  Buckets namely : ymax, xofmin, slopeinverse.
- **Edge Table:** It consistsof several edge lists -> holds all of the edges that compose the figure. When creating edges, the vertices of the edge need to be ordered from left to right and thee edges are maintained in increasing yMin order. Filling is complete once all of the edges are removed from the ET
- **Active List:** IT maintains the current edges being used to fill in the polygon.Edges aree pushed into the AL from the Edge Table when an edge's yMin is equal to the current scanline being processed.

  The Active List will be re-sorted after every pass

Data Structure:

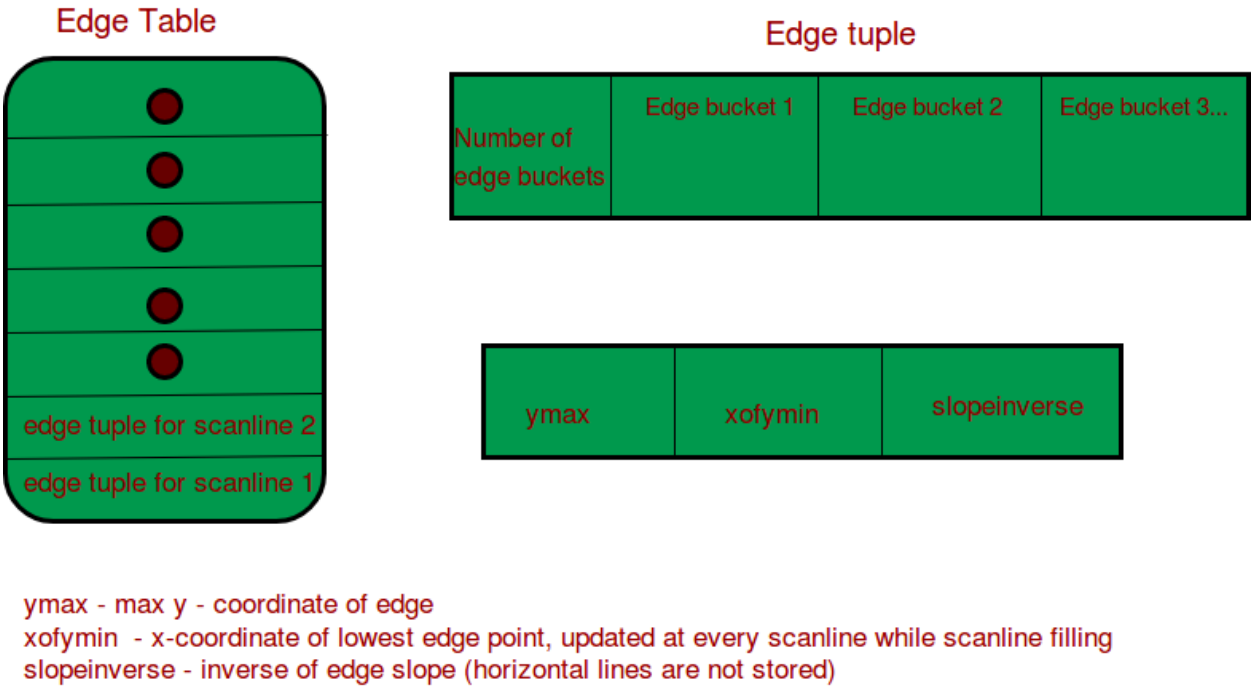

Figure 6

Algorithm:

- We will process the polygon edge after edge, and store in the edge Table.
- Storing is done by storing the edge in the same scanline edge tuple as the lowermost point's y-coordinate value of the edge.
- After addition of any edge in an edge tuple, the tuple is sorted using insertion sort, according to its xofymin value.
- After the whole polygon is added to the edge table, the figure is now filled.
- Filling is started from the first scanline at the bottom, and continued till the top.
- Now the active edge table is taken and the following things are repeated for each scanline:
    - Copy all edge buckets of the designated scanline to the active edge tuple
    - Perform an insertion sort according to the xofymin values
    - Remove all edge buckets whose ymax is equal or greater than the scanline
    - Fillup pairs of edges in active tuple, if any vertex is got, follow these instructions:
        - If both lines intersecting at the vertex are on the same side of the scanline, consider it as two points.
        - If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point.
    - Update the xofymin by adding slopeinverse for each bucket.

Implementing Above mentioned Algorithms to solve the Problem:

Steps Followed to solve the problem statement mentioned above:

- First for the base of designing the first work was to pointing out the points where to draw the edges and the edges should be in order to get drawn
- Step two is to draw the lines between these edges inorder to get a boundary of the object that we want to draw.
- Step three is to fill the color in the objects that we had drawn using the above algorithm be careful while filling as the color will be overwritten by scan-line polygon fill algorithm.
- Till step 3 we have the basic static layout of what we want to draw now adding some movement to it
- First adding translation to boats make a global variable which will increase by some value x and use this while plotting the x coordinate this will appear like that the boat is translating similarly do this for clouds also
- Now for the smoke bubble apply the step 5 in the similar manner
- For using key make a specific function and then added the functionality code in the keys
- After reading these lines refer the code for the best understanding of this project.

Figure 7

Language and Tools Used:

We have used the following:

- Code Language (C++)
- IDE (CodeBlocks)
- Library (OpenGL)

System Requirements for Developers:

Your system should have the following required libraries to run the project

- gcc compiler for compiling C/C++ code
- OpenGL/Glut dependencies installed
- IDE to work (eg. codeblocks)

Result:

So the result of the above program and the algorithms that we had used is that we build an animation for the moving of ship in the ocean in day or night without using any inbuilt function of OpenGL and got a beautiful graphic of it as the output

Conclusion and Future Work:

There are more implementation of the above algorithms together on a large scale. One can build efficient packages with these algorithm and use them normally like to draw line and fill color. This is a very basic project in computer graphics as we all know graphics are attractive to everyone, so the future of graphics is vast and one can do many animations using these basic things only.

References:

- http://groups.csail.mit.edu/graphics/classes/6.837/F00/Lecture05/lecture5.pdf
- http://agnigarh.tezu.ernet.in/~zubin/cg/lecture_notes/08-cg_class1.pdf
- https://www.geeksforgeeks.org/