1. Scanline Polygon filling 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.

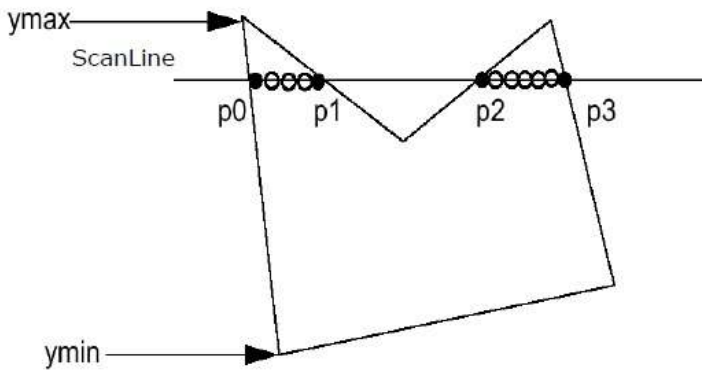


Figure 1: Scanline Polygon

Special cases of polygon vertices:

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

2. Components of algorithms

1. Edge Bucket
2. Edge Table
3. Active List

3. Algorithms

1. Step 1 Find out the Ymin and Ymax from the given polygon.
2. Step 2 ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.
3. Step 3 Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).
4. Step 4 Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

4. Code

step 1: C++ OpenGL code

```
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <fstream>
```

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#else
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/gl.h>
#endif
#include <OpenGL/gl3.h>
#include <stdio.h>
#include <math.h>
using namespace std;
#define maximum_height_of_image 800
#define maximum_width_of_image 600
#define maximum_Vertex 10000
typedef struct edgebucket
{
    int ymax;
    float xofymin;
    float slopeinverse;
} EdgeBucket;

typedef struct edgetabletup
{
    int countEdgeBucket;
    EdgeBucket buckets[maximum_Vertex];
}EdgeTableTuple;

EdgeTableTuple EdgeTable[maximum_height_of_image], ActiveEdgeTuple;
void initEdgeTable();
void insertionSort(EdgeTableTuple *ett);
void St_edgeIntuple (EdgeTableTuple *receiver,int ym,int xm,float slopInv);
void storeEdgeInTable (int x1,int y1, int x2, int y2);
void updatexbyslopeinv(EdgeTableTuple *Tup);
void Re_Edge_y_maximum(EdgeTableTuple *Tup,int yy);
void draw_polygon_rectangle();
void myInit(void);
void ScanlineFill();

void draw_polygon(void)
{
    initEdgeTable();
    draw_polygon_rectangle();
//    printf("\nTable");
    ScanlineFill();
}

void myInit(void)
{
    glClearColor(1.0,1.0,0.0,0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,maximum_height_of_image/6,0,maximum_width_of_image/6);
    glClear(GL_COLOR_BUFFER_BIT);
}
```

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(maximum_height_of_image,maximum_width_of_image);
    glutInitWindowPosition(400, 350);
    glutCreateWindow("Scanline filled polygon");
    myInit();
    glutDisplayFunc(draw_polygon);
    glutMainLoop();
}
void initEdgeTable()
{
    int i;
    for (i=0; i<maximum_height_of_image; i++)
    {
        EdgeTable[i].countEdgeBucket = 0;
    }
    ActiveEdgeTuple.countEdgeBucket = 0;
}
void insertionSort(EdgeTableTuple *ett)
{
    int i,j;
    EdgeBucket temp;
    for (i = 1; i < ett->countEdgeBucket; i++)
    {
        temp.ymax = ett->buckets[i].ymax;
        temp.xofymin = ett->buckets[i].xofymin;
        temp.slopeinverse = ett->buckets[i].slopeinverse;
        j = i - 1;

        while ((temp.xofymin < ett->buckets[j].xofymin) && (j >= 0))
        {
            ett->buckets[j + 1].ymax = ett->buckets[j].ymax;
            ett->buckets[j + 1].xofymin = ett->buckets[j].xofymin;
            ett->buckets[j + 1].slopeinverse = ett->buckets[j].slopeinverse;
            j = j - 1;
        }
        ett->buckets[j + 1].ymax = temp.ymax;
        ett->buckets[j + 1].xofymin = temp.xofymin;
        ett->buckets[j + 1].slopeinverse = temp.slopeinverse;
    }
}

void St_edgeIntuple (EdgeTableTuple *receiver,int ym,int xm,float slopInv)
{
    (receiver->buckets[(receiver)->countEdgeBucket]).ymax = ym;
    (receiver->buckets[(receiver)->countEdgeBucket]).xofymin = (float)xm;
    (receiver->buckets[(receiver)->countEdgeBucket]).slopeinverse = slopInv;
    insertionSort(receiver);
    (receiver->countEdgeBucket)++;
}
void storeEdgeInTable (int x1,int y1, int x2, int y2)
{
    float m,minv;
    int ymaxTS,xwithyminTS, scanline;
```

```
    if (x2==x1)
    {
        minv=0.000000;
    }
    else
    {
        m = ((float)(y2-y1))/((float)(x2-x1));
        if (y2==y1)
            return;
        minv = (float)1.0/m;
    }
    if (y1>y2)
    {
        scanline=y2;
        ymaxTS=y1;
        xwithyminTS=x2;
    }
    else
    {
        scanline=y1;
        ymaxTS=y2;
        xwithyminTS=x1;
    }
    St_edgeIntuple(&EdgeTable[scanline],ymaxTS,xwithyminTS,minv);
}
void Re_Edge_y_maximum(EdgeTableTuple *Tup,int yy)
{
    int i,j;
    for (i=0; i< Tup->countEdgeBucket; i++)
    {
        if (Tup->buckets[i].ymax == yy)
        {
            for ( j = i ; j < Tup->countEdgeBucket -1 ; j++ )
            {
                Tup->buckets[j].ymax =Tup->buckets[j+1].ymax;
                Tup->buckets[j].xofymin =Tup->buckets[j+1].xofymin;
                Tup->buckets[j].slopeinverse = Tup->buckets[j+1].slopeinverse;
            }
            Tup->countEdgeBucket--;
            i--;
        }
    }
}


void updatexbyslopeinv(EdgeTableTuple *Tup)
{
    int i;

    for (i=0; i<Tup->countEdgeBucket; i++)
    {
        (Tup->buckets[i]).xofymin =(Tup->buckets[i]).xofymin + (Tup->buckets[i]).slopeinverse;
    }
}
```

```
void ScanlineFill()
{
    int i, j, x1, ymax1, x2, ymax2, FillFlag = 0, coordCount;

    for (i=0; i<maximum_height_of_image; i++)
    {

        for (j=0; j<EdgeTable[i].countEdgeBucket; j++)
        {
            St_edgeIntuple(&ActiveEdgeTuple,EdgeTable[i].buckets[j].
                           ymax,EdgeTable[i].buckets[j].xofymin,
                           EdgeTable[i].buckets[j].slopeinverse);
        }
        Re_Edge_y_maximum(&ActiveEdgeTuple, i);

        insertionSort(&ActiveEdgeTuple);
        j = 0;
        FillFlag = 0;
        coordCount = 0;
        x1 = 0;
        x2 = 0;
        ymax1 = 0;
        ymax2 = 0;
        while (j<ActiveEdgeTuple.countEdgeBucket)
        {
            if (coordCount%2==0)
            {
                x1 = (int)(ActiveEdgeTuple.buckets[j].xofymin);
                ymax1 = ActiveEdgeTuple.buckets[j].ymax;
                if (x1==x2)
                {
                    if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))
                    {
                        x2 = x1;
                        ymax2 = ymax1;
                    }

                    else
                    {
                        coordCount++;
                    }
                }

                else
                {
                    coordCount++;
                }
            }
            else
            {
                x2 = (int)ActiveEdgeTuple.buckets[j].xofymin;
                ymax2 = ActiveEdgeTuple.buckets[j].ymax;

                FillFlag = 0;
                if (x1==x2)
                {
```

```
                    if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))
                    {
                        x1 = x2;
                        ymax1 = ymax2;
                    }
                    else
                    {
                        coordCount++;
                        FillFlag = 1;
                    }
                }
                else
                {
                    coordCount++;
                    FillFlag = 1;
                }


                if(FillFlag)
                {
                    glColor3f(0.0,1.0,1.0);

                    glBegin(GL_LINES);
                    glVertex2i(x1,i);
                    glVertex2i(x2,i);
                    glEnd();
                    glFlush();
                }

            }

            j++;
        }

        updatexbyslopeinv(&ActiveEdgeTuple);
    }
    printf("\nScanline filling complete");
}
void draw_polygon_rectangle(){
    glColor3f(0.0f,1.0f,0.0f);
    vector<pair< pair<int,int >,pair<int, int > > >vec;
    vec.push_back({{2,3},{7,1}});
    vec.push_back({{7, 1},{13, 6}});
    vec.push_back({{13, 6},{13, 11}});
    vec.push_back({{13, 11},{ 7, 7}});
    vec.push_back({{7, 7},{ 2, 10}});
    vec.push_back({{2, 10}, {2, 3}});
    glLineWidth(2.0);

    for (auto x : vec){
        int x1=x.first.first,y1=x.first.second,x2=x.second.first,y2=x.second.second;
        glBegin(GL_LINES);
        glVertex2i( x1, y1);
        glVertex2i( x2, y2);
        glEnd();
        storeEdgeInTable(x1, y1, x2, y2);
```

```
        printf("\n%d %d %d %d" , x1, y1,x2,y2);
        glFlush();
    }
}
```

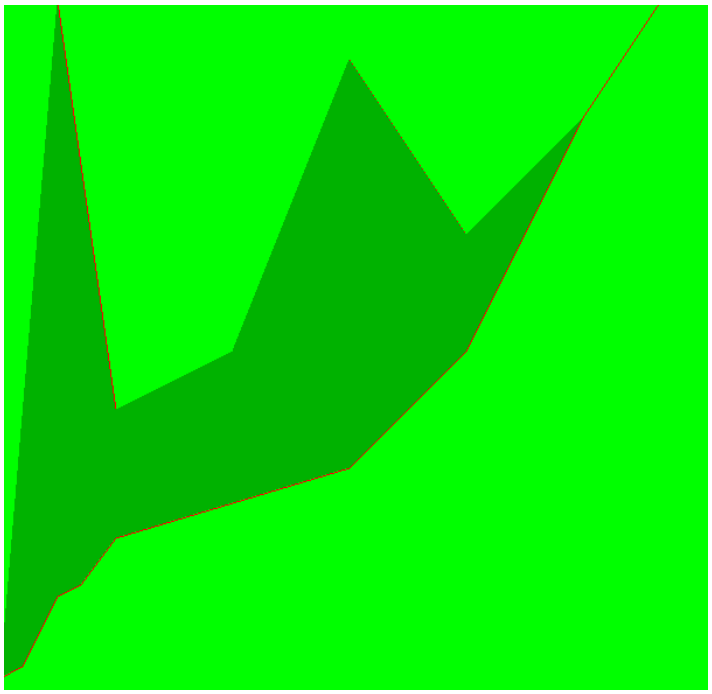
4. Code in python

step 2: Python code

```
import pylab as plt
import numpy as np
from matplotlib.path import Path
from skimage.draw import polygon
img = np.zeros((20, 20), dtype=np.uint8)
r = np.array([2,7,13,13,7,2,2])
c = np.array([3,1,6,11,7,10,3])
rr, cc = polygon(r, c)
img[rr, cc] = 1
img=np.rot90(img)
from matplotlib import pyplot as plt
plt.imshow(img, interpolation='nearest')
plt.show()
```

5. OpenGL Result

Polygon Filling Algorithm



6. Python result

Polygon Filling Algorithm in python

Graphics Assignment No:- 3

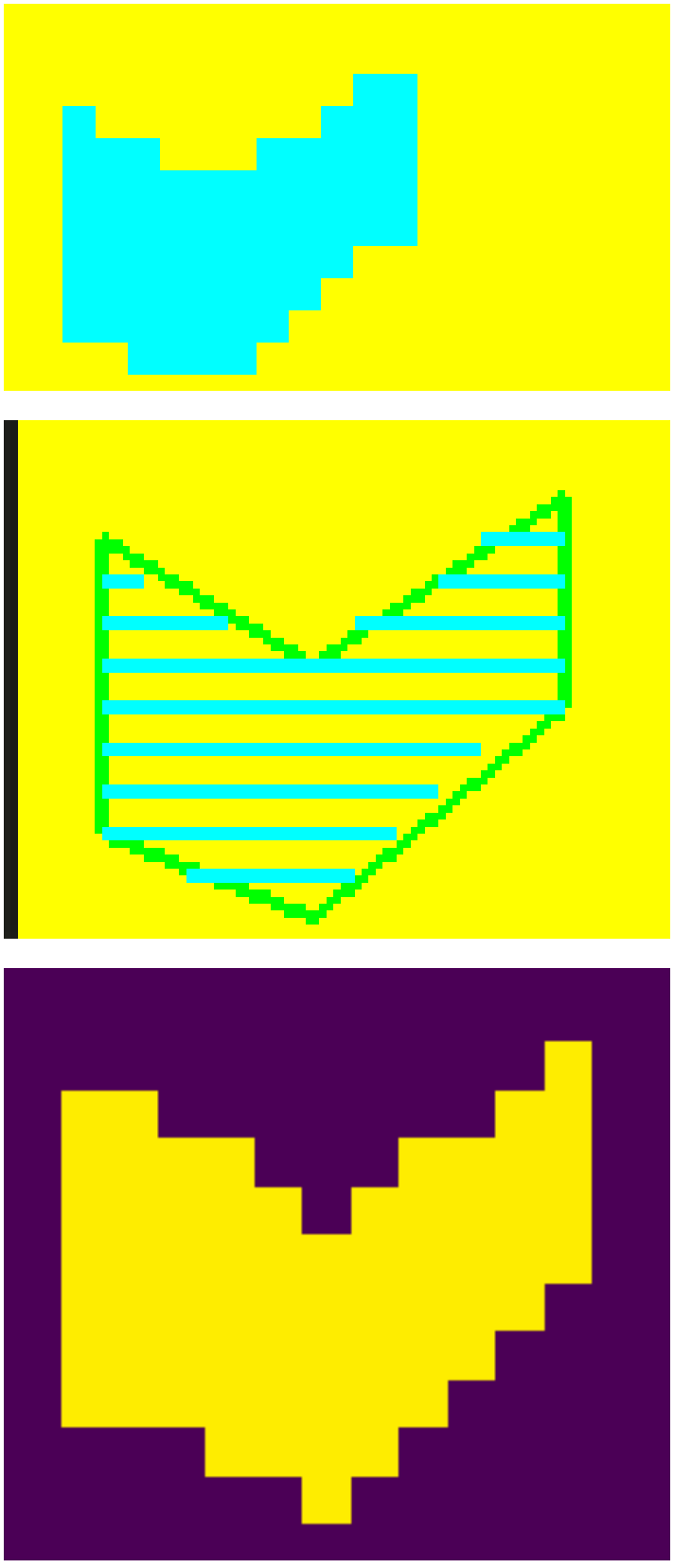


Figure 2: with given point x = np.array([2,7,13,13,7,2,2]) y = np.array([3,1,6,11,7,10,3])