**PRACTICAL 6**

**NAME: SHREYA SANJAY CHAUDHARI**

**ROLL NO: 04**

**BATCH: A DIV: COMPS 3**

**Aim:**  To implement Scan line Polygon Fill Algorithm

**Objective:**

Polygon is an ordered list of vertices as shown in the following figure. For filling polygons with particular colors, we need to determine the pixels falling on each scan line of the polygon and those which fall inside the polygon. Objective is to demonstrate the procedure for filling polygons using different techniques.

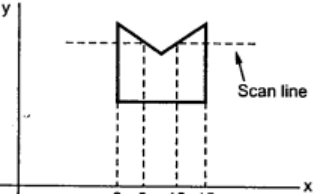
**Theory**

The basic scan-line algorithm is as follows:

* Find the intersections of the scan line with all edges of the polygon.
* Sort the intersections by increasing x coordinate.
* Fill in all pixels between pairs of intersections that lie interior to the polygon.

Process of scan-line polygon-filling algorithm involves -

* the horizontal scanning of the polygon from its lowermost to its topmost vertex,
* identifying which edges intersect the scan-line, and
* finally drawing the interior horizontal lines with the specified fill color process.



**Algorithm –**

1. The horizontal scanning of the polygon from its lowermost to its topmost vertex.

2. Identify the edge intersections of scan line with polygon edges.

3. Build the edge table.

4. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.

4. Determine whether any edges need to be splitted or not. If there is need to split, split the

edges.

5. Add new edges and build modified edge table.

6. Build Active edge table for each scan line and fill the polygon based on intersection of

scanline with polygon edges.

**Program:**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

void main()

{

int n,i,j,k,gd,gm,dy,dx;

int x,y,temp;

int a[20][2],xi[20];

float slope[20];

clrscr();

detectgraph(&gd,&gm);

initgraph(&gd,&gm,"..\\BGI");

printf("\n\n\tEnter the no. of edges of polygon :");

scanf("%d",&n);

printf("\n\n\tEnter the cordinates of polygon :\n\n\n");

for(i=0;i<n;i++)

{

printf("\tX%dY%d : ",i,i);

scanf("%d %d",&a[i][0],&a[i][1]);

}

a[n][0]=a[0][0];

a[n][1]=a[0][1];

/\*- draw polygon -\*/

for(i=0;i<n;i++)

{

line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);

}

for(i=0;i<n;i++)

{

dy=a[i+1][1]-a[i][1];

dx=a[i+1][0]-a[i][0];

if(dy==0) slope[i]=1.0;

if(dx==0) slope[i]=0.0;

if((dy!=0)&&(dx!=0)) /\*- calculate inverse slope -\*/

{

slope[i]=(float) dx/dy;

}

}

for(y=0;y< 480;y++)

{

k=0;

for(i=0;i<n;i++)

{

if( ((a[i][1]<=y)&&(a[i+1][1]>y))||

((a[i][1]>y)&&(a[i+1][1]<=y)))

{

xi[k]=(int)(a[i][0]+slope[i]\*(y-a[i][1]));

k++;

}

}

for(j=0;j<k-1;j++) /\*- Arrange x-intersections in order -\*/

for(i=0;i<k-1;i++)

{

if(xi[i]>xi[i+1])

{

temp=xi[i];

xi[i]=xi[i+1];

xi[i+1]=temp;

}

}

setcolor(52);

for(i=0;i<k;i+=2)

{

line(xi[i],y,xi[i+1]+1,y);

delay(50);

}

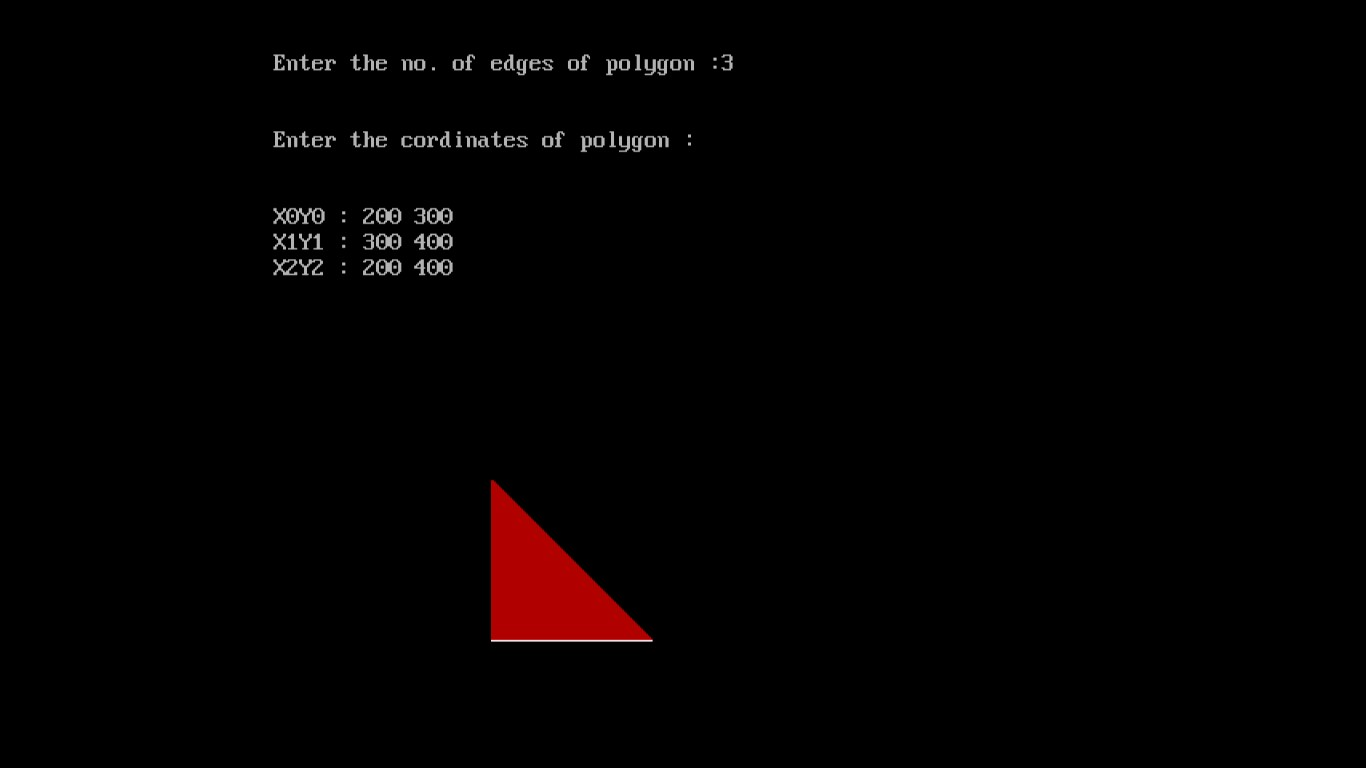
}

getch();

closegraph();

}

**Output:**



**Conclusion -**Comment on

1. **.Importance of Scan Line Polygon:**
2. **Filling Complex Shapes**: It efficiently fills complex, non-convex polygons, making it valuable for rendering intricate shapes in 2D graphics.
3. **Antialiasing**: Scanline algorithms can be extended to handle antialiasing, which is crucial for achieving smooth and high-quality rendering of images and shapes.
4. **Texture Mapping**: It is used in texture mapping, allowing textures or patterns to be applied to irregular shapes by mapping them onto the polygon's surface.
5. **Hidden Surface Removal**: Scanline algorithms are often employed as part of the hidden surface removal process to determine which parts of objects are visible and should be rendered.
6. **Rasterization**: The algorithm converts vector graphics into raster images by determining which pixels should be filled, which is fundamental in modern computer graphics pipelines.

**2. Limitation of methods:**

1. **Limited to 2D**: Primarily designed for 2D graphics and less suitable for complex 3D scenes.
2. **Complex Polygons**: Struggles with concave or self-intersecting polygons and may require preprocessing.
3. **Performance**: Efficiency can degrade with a large number of polygons or complex scenes.
4. **Edge Cases**: May require additional handling for cases involving coincident or overlapping edges.

**3. Usefulness of method:**Scanline algorithm is useful for efficiently filling complex 2D shapes with colours or textures in computer graphics.