**Experiment No:- 08 Date:-\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim**:- To fill a polygon using scan-line polygon fill algorithm.

**Theory:-**

The scan-line procedure for solid filling of polygon areas, for each scan-line crossing polygon, the area fill algorithm locates the intersection points of the scan-line with the polygon edges. These intersection points are then sorted from left to right and the corresponding frame-buffer positions between each intersection pair are set to the specified fill color.

Some scan-line intersections at polygon vertices require special handling. A scan-line passing through a vertex intersects 2 polygon edges at that position, adding 2 points to the list of intersections for the scan-line. We can identify these vertices by tracing around the polygon boundary either in clockwise or counter-clockwise order and observing the relative changes in vertex y coordinates as we move from one edge to the next.

Calculations performed in scan conversion and other graphics algorithms typically take advantage of various coherence properties of a scene that is to be displayed. What we mean by coherence is that the properties of one part of a scene are related in some way to other parts of the scene so that the relationship can be used to reduce processing.

The slope of this polygon boundary line can be expressed in terms of the scan-line intersection co-ordinates:

m=(yk+1-yk)/(xk+1-xk)

Since the change in y co-ordinate between the two scan-lines is simply

Yk+1-yk=1

The x intersection value xk+1 on the upper scan-line can be determined from the x-intersection value xk on the preceding scan-line as :

Xk+1=xk+1/m

Each successive intercept can then be calculated by adding the inverse of the slope and rounding to the nearest integer.

Along an edge with slope m, the intersection xk value for scan-line k above the initial scan-line can be calculated as :

xk=x0+k/m

m=Δy/Δx

Incremental calculation of x intercepts along an edge for successive scan line.

xk+1=xk+Δx/Δy

To efficiently perform a polygon-fill, we can first store the polygon boundary in a sorted edge table that contains all the information necessary to process the scan-lines efficiently. Proceeding around the edges in either a clockwise or counter-clockwise order, we can use a bucket sort to store the edges, sorted on the smallest y-value of each edge, in the correct scan-line positions.

Next we process the scan-lines from the bottom of the polygon towards its top, producing an active edge list for each scan-line crossing the polygon boundaries. Implementation of edge intersection calculations can also be facilitated by storing Δx and Δy values in the sorted edge table.

The following procedure performs a solid-fill scan conversion for an input set of polygon vertices. For each scan-line within the vertical extents of the polygon, an active list is set up and edge intersections are calculated. Across each scan-line, the interior fill is then applied between successive pairs of edge intersections, processed from left to right.

**Program:-**

/\* Define the structure to store the edges\*/

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include <dos.h>

struct edge{

int x1,y1,x2,y2,flag;};

int main(){

int gd=DETECT,gm,n,i,j,k;

struct edge ed[10],temped;

float dx,dy,m[10],x\_int[10],inter\_x[10];

int x[10],y[10],ymax=0,ymin=480,yy,temp;

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

/\*read the number of vertices of the polygon\*/

cout<<"Enter the no.of vertices of the graph :";

cin>>n;

/\*read the vertices of the polygon and also find ymax and ymin\*/

cout<<"Enter the vertices";

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

// cout<<x[i];

cin>>x[i];

//cout<<y[i];

cin>>y[i];

if(y[i]>ymax)

ymax=y[i];

if(y[i]<ymin)

ymin=y[i];

ed[i].x1=x[i];

ed[i].y1=y[i];

}

/\*store the edge information\*/

for(i=0;i<n-1;i++) {

ed[i].x2=ed[i+1].x1;

ed[i].y2=ed[i+1].y1;

ed[i].flag=0;

}

ed[i].x2=ed[0].x1;

ed[i].y2=ed[0].y1;

ed[i].flag=0;

/\*Check for y1>y2, if not interchnge y1 and y2 \*/

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

if(ed[i].y1 < ed[i].y2) {

temp=ed[i].x1;

ed[i].x1=ed[i].x2;/////////////

ed[i].x2=temp;

temp=ed[i].y1;

ed[i].y1=ed[i].y2;

ed[i].y2=temp;

}

}

/\*Draw the polygon\*/

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

line(ed[i].x1, ed[i].y1,ed[i].x2,ed[i].y2);

}

/\*sorting of edges in the order of y1,y2,x1\*/

for(i=0;i<n-1;i++){

for(j=0;j<n-1;j++) {

if(ed[j].y1<ed[j+1].y1) {

temped=ed[j];

ed[j]=ed[j+1];

ed[j+1]=temped;

}

if(ed[j].y1==ed[j+1].y1) {

if(ed[j].y2<ed[j+1].y2) {

temped=ed[j];

ed[j]=ed[j+1];

ed[j+1]=temped;

}

if (ed[j].y2==ed[j+1].y2){

if(ed[j].x1<ed[j+1].x1) {

temped=ed[j];

ed[j]=ed[j+1];

ed[j+1]=temped;

}

}

}

}

}

/\*calculating 1/slope of each edge and storing top\*/

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

dx=ed[i].x2-ed[i].x1;

dy=ed[i].y2-ed[i].y1;

if(dy==0){

m[i]=0;

}

else {

m[i]=dx/dy;

}

inter\_x[i]=ed[i].x1;

}

/\*making the Actual edges\*/

yy=ymax;

while(yy>ymin){

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

if(yy>ed[i].y2 && yy<=ed[i].y1) {

ed[i].flag=1;

}

else

ed[i].flag=0;

}

j=0;

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

if(ed[i].flag==1){

if(yy==ed[i].y1){

x\_int[j]=ed[i].x1;

j++;

if(ed[i-1].y1==yy && ed[i-1].y1<yy) {

x\_int[j]=ed[i].x1;

j++;

}

if(ed[i+1].y1==yy && ed[i+1].y1<yy){

x\_int[j]=ed[i].x1;

j++;

}

}

else {

x\_int[j]=inter\_x[i]+(-m[i]);

inter\_x[i]=x\_int[j];

j++;

}

}

}

/\*sorting the x intersaction\*/

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

{

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

{

if(x\_int[k]>x\_int[k+1])

{

temp=(int)x\_int[k];

x\_int[k]=x\_int[k+1];

x\_int[k+1]=temp;

}

}

}

/\*extracting pairs of values to draw lilnes\*/

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

{

line((int)x\_int[i],yy,(int)x\_int[i+1],yy);

}

yy--;

delay(50);

}

getch();

closegraph();

}