**Practical No 6.C**

**Implementation of Scan line polygon fill algorithm.**

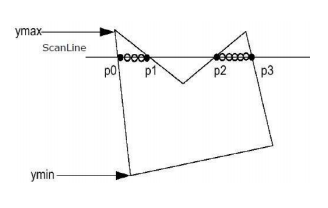
**Aim: Write a program to implement a Scan line polygon fill algorithm.**

**Theory:**

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. Scanline filling is basically filling up of polygons using horizontal lines or scanlines. The purpose of this algorithm is to fill the interior pixels of the polygon given only the vertices. To understand scanline, think it of being drawn by a single per from the bottom left containing to the right, plotting only points where there is a point and when the line is complete start from the next line end continue. This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of instructions.

**Algorithm:**

**Step 1 −** Find out the Ymin and Ymax from the given polygon.



**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.

**Step 3 −** Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

**Step 4 −** Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

**Conclusion: We have implemented a Scan line polygon fill algorithm.**

**Code:**

#include<conio.h>

#include<iostream.h>

#include<graphics.h>

#include<stdlib.h>

#include<dos.h>

class point {

public: int x,y; };

class poly {

private:

point p[20];

int inter[20],x,y;

int v,xmin,ymin,xmax,ymax;

public:

int c;

void read();

void calcs();

void display();

void ints(float);

void sort(int); };

void poly::read() {

int i;

cout<<"SCAN\_FILL ALGORITHM";

cout<<"\nEnter the no of vertices of polygon:";

cin>>v;

if(v>2) {

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

cout<<"\nEnter the co-ordinate no "<<i+1<<" : ";

cout<<"\nx"<<(i+1)<<"=";

cin>>p[i].x;

cout<<"\ny"<<(i+1)<<"=";

cin>>p[i].y; }

p[i].x=p[0].x;

p[i].y=p[0].y;

xmin=xmax=p[0].x;

ymin=ymax=p[0].y; }

else

cout<<"\nEnter valid no of vertices: "; }

void poly::calcs() {

for(int i=0;i<v;i++) {

if(xmin>p[i].x)

xmin=p[i].x;

if(xmax<p[i].x)

xmax=p[i].x;

if(ymin>p[i].y)

ymin=p[i].y;

if(ymax<p[i].y)

ymax=p[i].y; } }

void poly::display(){

int ch1; char ch='y'; float s,s2; do

{

cout<<"MENU:";

cout<<"\n1. Scan line Fill";

cout<<"\n2 .Exit";

cout<<"\nEnter your choice:";

cin>>ch1;

switch(ch1)

{

case 1:

s=ymin+0.01;

delay(100);

cleardevice();

while(s<=ymax)

{

ints(s);

sort(s);s++;

}

break;

case 2:

exit(0);

}

cout<<"Do you want to continue?: ";

cin>>ch;

}

while(ch=='y' || ch=='Y');

}

void poly::ints(float z)

{

int x1,x2,y1,y2,temp;

c=0;

for(int i=0;i<v;i++)

{

x1=p[i].x;

y1=p[i].y;

x2=p[i+1].x;

y2=p[i+1].y;

if(y2<y1)

{

temp=x1;

x1=x2;

x2=temp;

temp=y1;

y1=y2;

y2=temp;

}

if(z<=y2&&z>=y1)

{

if((y1-y2)==0)

x=x1;

else

{

x=((x2-x1)\*(z-y1))/(y2-y1);

x=x+x1; }

if(x<=xmax && x>=xmin) inter[c++]=x; } } }

void poly::sort(int z) {

int temp,j,i;

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

line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); }

delay(100); for(i=0; i<c;i+=2) {

delay(100);

line(inter[i],z,inter[i+1],z); } }

void main() {

int cl;

int gm,gd=DETECT; initgraph(&gd,&gm,"C:/TURBOC3/BGI");

cleardevice();

poly x;

x.read();

x.calcs();

cleardevice();

setcolor(cl);

x.display();

getch();

closegraph();

}

**Output:**

