**Scan Line Algorithm**

**Object:** To demonstrate the scan line algorithm

**Theory:** This algorithm is based on the Image-space method and concept of coherence. As its name suggests itself Scan-line algorithm, so it processes one line at a time rather than processing one pixel(a point on raster display) at a time. The algorithm works as follows:

1. Here each point at whsich the scan- line intersects the polygon surfaces are examined(processed) from left to right and in this process.
2. The depth calculation (if there are overlapping surfaces found) is performed to identify the Hidden region(Visible surface) of the polygons which is nearer to the viewing plane.
3. As soon as the visible surfaces(Hidden surfaces) are identified then the corresponding color-intensity values are updated into the refresh buffer(Frame buffer) if and only if the Flag of the corresponding surface is on.
4. This algorithm works efficiently with one or more than one polygon surface and this algorithm is just an extension of the Scan line algorithm of Polygon filling.

**Program**

#include <conio.h> #include <iostream> #include <graphics.h> #include <stdlib.h> using namespace std;

//Declaration of class point 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<<"\n\t SCAN\_FILL ALGORITHM";

cout<<"\n Enter the no of vertices of polygon:"; cin>>v;

if(v>2)

{

for(i=0;i<v; i++) //ACCEPT THE VERTICES

{

cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : "; cout<<"\n\tx"<<(i+1)<<"=";

cin>>p[i].x; cout<<"\n\ty"<<(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<<"\n Enter valid no. of vertices.";

}

//FUNCTION FOR FINDING

void poly::calcs()

{ //MAX,MIN

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;

}

}

//DISPLAY FUNCTION

void poly::display()

{

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

{

cout<<"\n\nMENU:"; cout<<"\n\n\t1 . Scan line Fill "; cout<<"\n\n\t2 . Exit "; cout<<"\n\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) //DEFINE FUNCTION INTS

{

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 // used to make changes in x. so that we can fill our polygon after cerain distance

{

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

}

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

}

}

}

void poly::sort(int z) //SORT FUNCTION

{

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); // used to make hollow outlines of a polygon

}

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

{

delay(100);

line(inter[i],z,inter[i+1],z); // Used to fill the polygon ....

}

}

int main() //START OF MAIN

{

int cl; initwindow(500,600); cleardevice();

poly x; x.read();

x.calcs(); cleardevice();

cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour cin>>cl;

setcolor(cl); x.display();

closegraph(); //CLOSE OF GRAPH getch();

return 0;