**Experiment No. 06**

**Scan Line Polygon Fill Algorithm**

**Aim:**

Implement Scan Line Polygon Filling Algorithm.

**Theory:**

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.

**Procedure:**

Step 1: The algorithm begins with first scan line that the polygon occupies i.eymax and proceeds line by line towards the last scan line i.eymin.

Step 2: Sort Xmax, Xmin, Ymax, Ymin of the edges of polygon along with their slopes.

Step 3: To decide which edges are getting intersected by scan line we are making use of Ymax of particular edge.

Step 4: Every time we are decreasing scan line by 1 from Ymax to Ymin of the polygon.

Step 5: It may happen that the edge which we have selected to find intersection point may get finished i.eYmin goes below the Ymin of selected edge. In that case discard the edge and select next edge from the sorted table and continue to sac line.

Step 6: Decreasing Ymax by 1 unit i.e Ymax-1 to find corresponding value of intersection point. Xnew=Xold+1/m

Step 7: Find the intersection of scan line with every edge of polygon.

**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(35);  for(i=0;i<k;i+=2)  {  line(xi[i],y,xi[i+1]+1,y);  delay(20);  }  }  getch();  closegraph();  } |
| --- | --- |

**Outcome:**

