Where, Δx & Δy are the differences between the edge endpoint x and y co-ordinate va
Thus, incremental calculations of x intercepts along an edge for successive scan lines ca
expressed as

$$x_k + 1 = x_k + \frac{\Delta x}{\Delta y}$$

Algorithm Steps

1. the horizontal scanning of the polygon from its lowermost to its topmost vertex
2. identify the edge intersections of scan line with polygon
3. Build the edge table
   - Each entry in the table for a particular scan line contains the maximum y valu that edge, the x-intercept value (at the lower vertex) for the edge, and the inv slope of the edge.
4. Determine whether any edges need to be splitted or not. If there is need to split, spli 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 intersecti scan line with polygon edges.

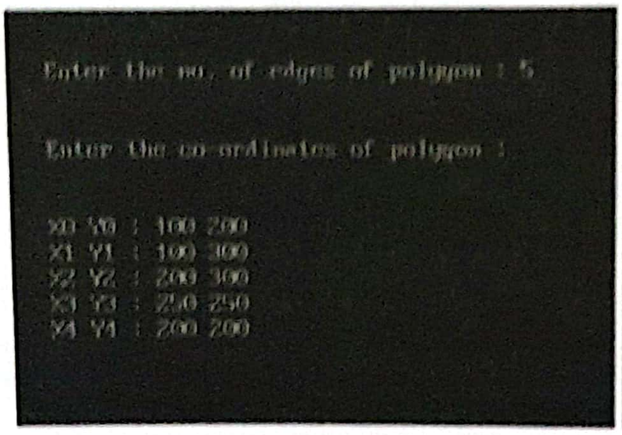
Lab problem 1.6: Program for scan line polygon fill algorithm

```
#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();
        printf("\n\n\t Enter the no. of edges of polygon : ");
        scanf("%d", &n);
        printf("\n\n\t Enter the co-ordinates of polygon :\n\n\n ");
        for(i=0;i<n;i++)
        {
                printf("\t X%d Y%d : ",i ,i);
                scanf("%d %d", &a[i][0],&a[i][1]);
        }
        a[n][0]=a[0][0];
        a[n][1]=a[0][1];
```

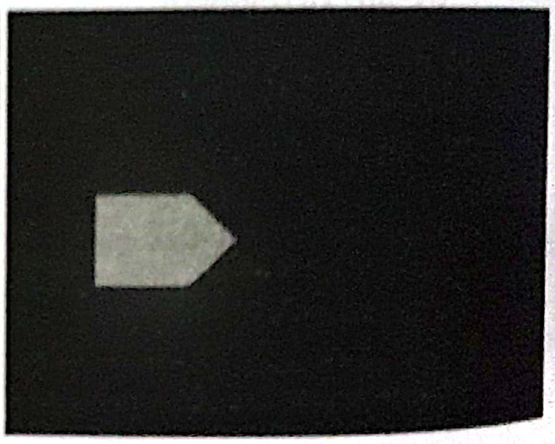
```
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
for(i=0;i<n;i++)
{
        line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);
}
getch();
for(i=0;i<n;i++)
{
        dy=a[i+1][1]-a[i][1];
        dx=a[i+1][0]-a[i][0];
        if(dy==0)
                slope[i]=1.0;
        if(dx==0)
                slope[i]=0.0;
        if((dy!=0)&&(dx!=0))
        {
                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(7);
                for(i=0;i<k;i+=2)
                {
                        line(xi[i],y,xi[i+1]+1,y);
                        getch();
                }
        }
}
```

Output

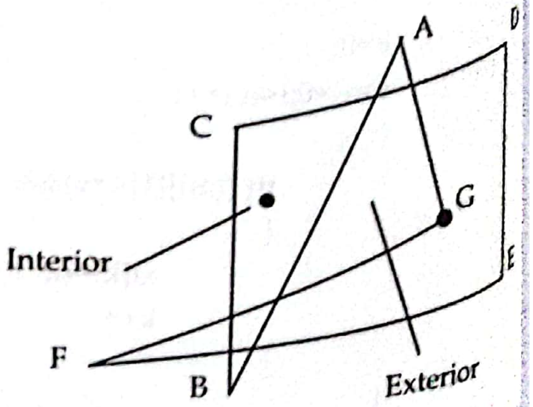


```
Enter the no. of edges of polygon : 5

Enter the co-ordinates of polygon :

X0 Y0 : 100 200
X1 Y1 : 100 300
X2 Y2 : 200 300
X3 Y3 : 250 250
X4 Y4 : 200 200
```



Inside-Outside Test

One simple way of finding whether the point is inside or outside a simple polygon is to test how many times a ray, starting from the point and going in any fixed direction, intersects the edges of the polygon. If the point is on the outside of the polygon the ray will intersect its edge an even number of times. If the point is on the inside of the polygon, then it will intersect the edge an odd number of times. Unfortunately, this method won't work if the point is on the edge of the polygon.



Area filling algorithms and other graphics package often need to identify interior and ext region for a complex polygon in a plane. For example, in figure below, it needs to ide interior and exterior region.

This method is also known as **counting number method**. While filling an object, we often n to identify whether particular point is inside the object or outside it. There are two methods which we can identify whether particular point is inside an object or outside.

- Odd-Even method
- Nonzero winding number method