**EXPERIMENT - 07**

**NAME - TRUSHANT RATHOD**

**DIV - C(3) BATCH - A ROLL NO-18**

**Aim:**  To implement Scan line Polygon Fill Algorithm

**Objective:**

Polygon is an ordered list of vertices as shown in the following figure. For filling polygons with particular colors, we need to determine the pixels falling on each scan line of the polygon and those which fall inside the polygon. Objective is to demonstrate the procedure for filling polygons using different techniques.

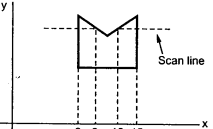
**Theory**

The basic scan-line algorithm is as follows:

* Find the intersections of the scan line with all edges of the polygon.
* Sort the intersections by increasing x coordinate.
* Fill in all pixels between pairs of intersections that lie interior to the polygon.

Process of scan-line polygon-filling algorithm involves -

* the horizontal scanning of the polygon from its lowermost to its topmost vertex,
* identifying which edges intersect the scan-line, and
* finally drawing the interior horizontal lines with the specified fill color process.



**Algorithm –**

1. The horizontal scanning of the polygon from its lowermost to its topmost vertex.

2. Identify the edge intersections of scan line with polygon edges.

3. Build the edge table.

4. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.

4. Determine whether any edges need to be splitted or not. If there is need to split, split the

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 intersection of

scanline with polygon edges.

**Program:**

**#include <graphics.h>**

**#include <math.h>**

**#include <conio.h>**

**#include <stdio.h>**

**void main()**

**{**

**int x[4], y[4], i;**

**double put\_x, put\_y, t;**

**int gd = DETECT, gm;**

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

**printf("\n\*\*\* Bezier Curve \*\*\*\*");**

**printf("\n Please enter x and y coordinates ");**

**for (i = 0; i < 4; i++)**

**{**

**scanf("%d %d", &x[i], &y[i]);**

**putpixel(x[i], y[i], 3); // Control Points**

**}**

**for (t = 0.0; t <= 1.0; t = t + 0.001) // t always lies between 0 and 1**

**{**

**put\_x = pow(1 - t, 3) \* x[0] + 3 \* t \* pow(1 - t, 2) \* x[1] + 3 \* t \* t \* (1 - t) \* x[2] + pow(t, 3) \* x[3]; // Formula to draw curve**

**put\_y = pow(1 - t, 3) \* y[0] + 3 \* t \* pow(1 - t, 2) \* y[1] + 3 \* t \* t \* (1 - t) \* y[2] + pow(t, 3) \* y[3];**

**putpixel(put\_x, put\_y, WHITE); // putting pixel**

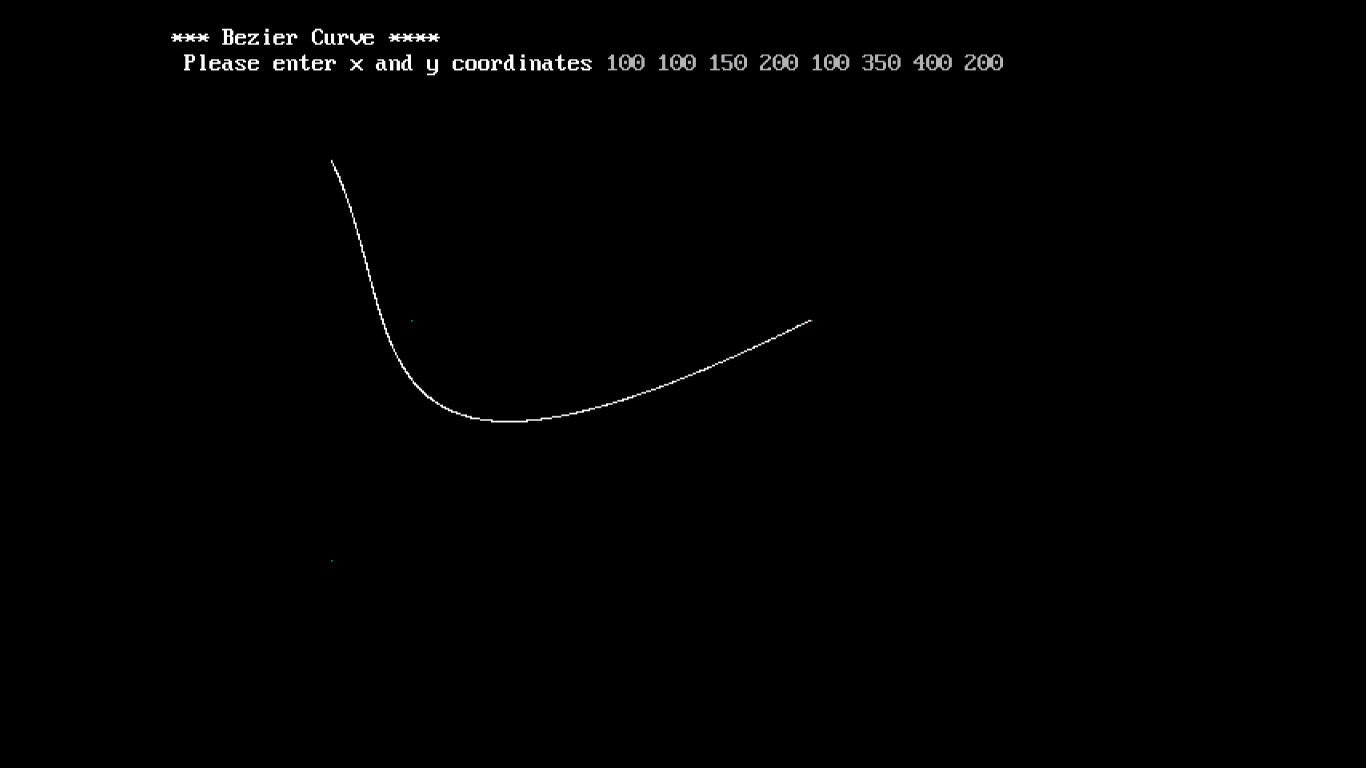
**}**

**getch();**

**closegraph();**

**}**

**Output:**

****

**Conclusion -**Difference from arc and line:The main difference between a cubic Bezier curve and a line segment is that a cubic Bezier curve can create curved paths with varying degrees of curvature, requiring four control points, while a line segment represents a straight path with no curvature, needing only two control points. Bezier curves are for curves, while line segments are for straight lines.

Importance of control point:control points in Bezier curves determine the curve's shape, direction, and smoothness, providing precision, flexibility, and the ability to create various curves, making them essential in design, animation, and computer graphics.

Applications:Bezier curves are used in computer graphics, design, animation, manufacturing, robotics, medical imaging, and more for precise and smooth curve representation and control in various applications.