**PRACTICAL 7: BEZIER CURVE**

NAME: RIDDHI VIPUL MEHTA

DIV\BATCH: COMPS 3 [A]

ROLL NO.: 06

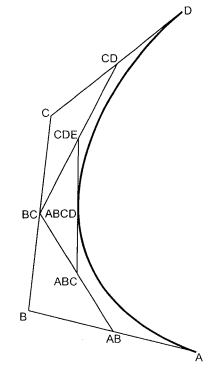
**Aim:**  To implement Bezier curve for n control points. (Midpoint approach)

**Objective:**

Draw a Bezier curves and surfaces written in Bernstein basis form. The goal of interpolation is to create a smooth curve that passes through an ordered group of points. When used in this fashion, these points are called the control points.

**Theory:**

In midpoint approach Bezier curve can be constructed simply by taking the midpoints. In this approach midpoints of the line connecting four control points (A, B, C, D) are determined (AB, BC, CD, DA). These midpoints are connected by line segment and their midpoints are ABC and BCD are determined. Finally, these midpoints are connected by line segments and its midpoint ABCD is determined as shown in the figure –



The point ABCD on the Bezier curve divides the original curve in two sections. The original curve gets divided in four different curves. This process can be repeated to split the curve into smaller sections until we have sections so short that they can be replaced by straight lines.

**Algorithm:**

1) Get four control points say A(xa, ya), B(xb, yb), C(xc, yc), D(xd, yd).

2) Divide the curve represented by points A, B, C, and D in two sections.

xab = (xa + xb) / 2

yab = (ya + yb) / 2

xbc = (xb + xc) / 2

ybc = (yb + yc) / 2

xcd = (xc + xd) / 2

ycd = (yc + yd) / 2

xabc = (xab + xbc) / 2

yabc = (yab + ybc) / 2

xbcd = ( xbc + xcd) / 2

ybcd = (ybc + ycd) / 2

xabcd = (xabc + xbcd) / 2

yabcd = (yabc + ybcd) / 2

3) Repeat the step 2 for section A, AB, ABC, ABCD and section ABCD, BCD, CD, D.

4) Repeat step 3 until we have sections so that they can be replaced by straight lines.

5) Repeat small sections by straight lines.

6) Stop.

**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

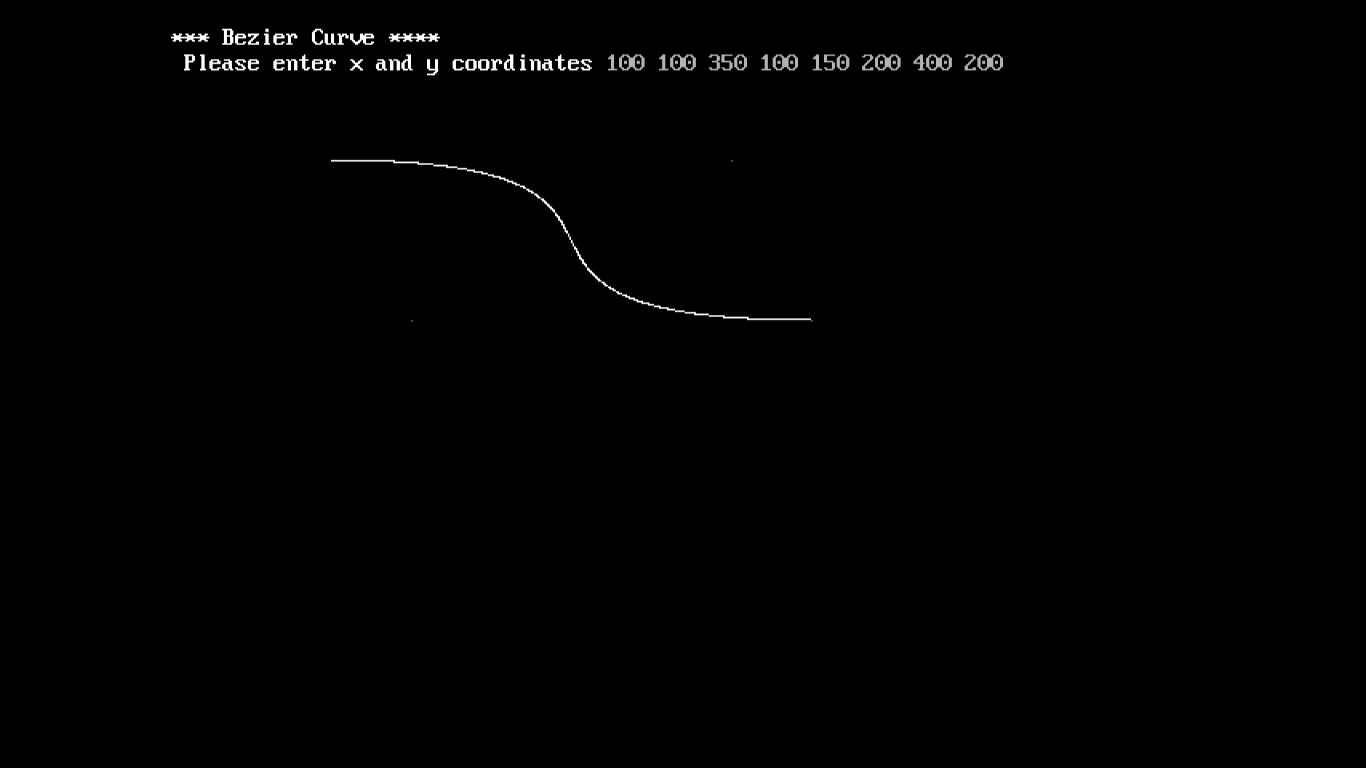
}

getch();

closegraph();

}

**Output:**

****

**Conclusion –**

1. **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.
2. **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.
3. **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.