CG Experiment 10.

Yash Sarang D6AD / 47.

Aim : To implement Berzier Curve.

Theory :

- Control points :-

A member of a set of points used to determine the shape of a spine curve or a surface of higher dimensional object.

- Local Control :-

When a designer maniplilates a control points a curve changes only in the region near the CP.

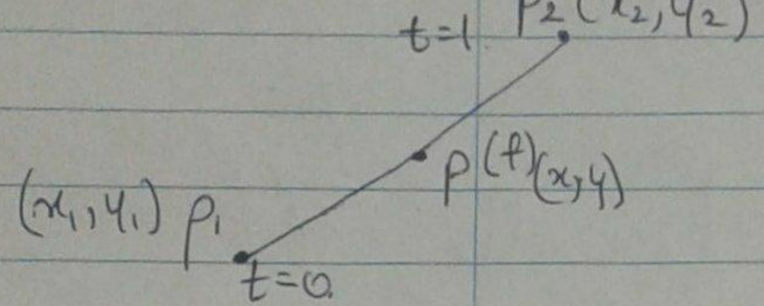
- Global Control :-

When a designer manipulates a control point, a curve changes its shape throughout.

- Benzier Curve :-

A Benzier curve is a parametric curve that is used to draw smoother lines

* Derivation of linear Benzier Curve:-



$$x = x_1 + dx \cdot t$$
$$= x_1 + (x_2 - x_1)\, t$$

$$y = y_1 + dy \cdot t$$
$$= y_1 + (y_2 - y_1)\, t.$$

In general,

$$P = P_1 + (P_2 - P_1)\, t.$$
$$= P_1 + tP_2 - tP_1.$$
$$P = P_1(1-t) + tP_2 \quad \text{—} \; (1)$$

Using eq^n (1),

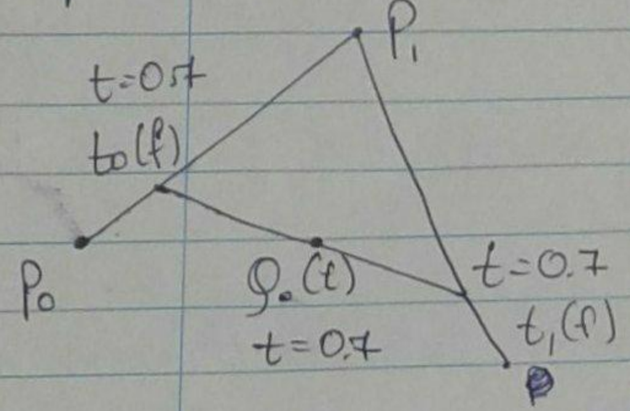
$$x = x_1(1-t) + x_2 t$$
$$y = y_2(1-t) + y_2 t.$$

* Derivation of Quadratic Benzier Curve:-

Degree = 2

No. of ~~p~~ control points = 2+1 = 3.

Consider particular value of parameter t,



① For given t, lets assume we have points L_0 on P_0P_1 & L_1 on P_1P_2

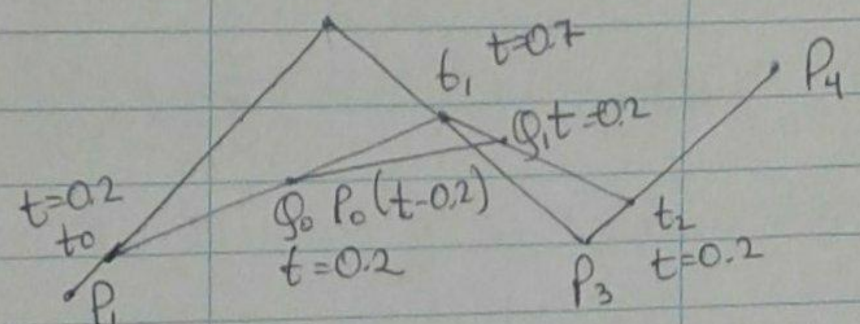
② Join these two points, we get L_0L_1, we find the point at which parameter value is t, Let's say

$$L_0 = (1-t) P_0 + t(P_1)$$
$$L_1 = (1-t) P_1 + t.P_2$$
$$Q_0 = (1-t) L_0 + (t) L_1$$
$$= (1-t)\left[(1-t) P_0 + tP_1\right] + t\left[(1-t) P_1 + tP_2\right]$$
$$= (1-t)^2 P_0 + (1-t)t P_1 + (1-t)t P_1 + t^2 P_2$$
$$\boxed{Q_0 = (1-t)^2 P_0 + 2(1-t) t P_1 + t^2 P_2}$$

We can see the degree of polynomial is two.

* Derivation of cubic Berzier Curve.

Degree = 3. , hence no of CP = 3+1 = 4.



For any given parameter find point of P_1P_2, P_2P_3, P_3P_4 say L_0, L_1, L_2.

For some value of parameter find points on L_0L_1, & L_1L_2 say φ_0, φ_1.

On φ_0, φ_1 find the point at parameter I, say C_0.

C_0 will be on curve.

$$L_0 = (1-t)P_1 + tP_2$$
$$L_1 = (1-t)P_2 + tP_3$$
$$L_2 = (1-t)P_3 + tP_4.$$
$$\varphi_0 = (1-t)L_0 + tL_1$$
$$\varphi_1 = (1-t)L_1 + t(L_2)$$
$$C_0 = (1-t)\varphi_0 + t\varphi_1$$

$$\boxed{C_0(t) = (1-t)^3 P_0 + 3t((1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3}$$

* Blending function.

$$P(U) = \sum_{i=0}^{i=n} B_{i,n}(u) = P_i$$

where,

$$B_{i,n}(U) = {}^nC_i (1-U)^{n-i} U^i$$

$$B_{0,3}(U) = (1-U)^3$$

$$B_{1,3}(U) = 3u(1-u)^2$$

$$B_{2,3}(U) = 3u^2(1-u)$$

$$B_{3,3}(U) = u^3.$$

* Properties of Bezier Curve.

1. It is an approximation curve.
2. Used in CAD, Type face, Drawing, etc.
3. Easy to implement.
4. Parametric curve.
5. CP affect the shape of the curve.

* Applications of Bezier curves.

1. In animations to outline.
2. Fonts which is done by quadratic Bezier s.
3. Rubobies to produce trajectors of an end effector.
4. Used in computer graphics to model smoother curves.

★ Conclusion:

We have studied the ~~implementation~~ implementation of Bezier curve for n points

Code: -

```
#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\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%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");
  /- draw polygon -/
  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)) /- 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(3);
      for (i = 0; i < k; i += 2) {
        line(xi[i], y, xi[i + 1] + 1, y);
        getch();
      }
    }
  }
```

OUPUT: