**Aim:** To implement curve plotting algorithms.

**Theory:**

-What is a Bezier Curve?

A Bezier curve is a mathematical curve that is defined by a set of control points. It's widely used in computer graphics and animation to create smooth and visually appealing curves. The shape of the Bezier curve is determined by the position of its control points. There are different degrees of Bezier curves, but I'll explain the quadratic Bezier curve for simplicity, which is defined by three control points.

A quadratic Bezier curve is defined by the following formula:

\[B(t) = (1-t)^2 \* P0 + 2 \* (1-t) \* t \* P1 + t^2 \* P2\]

Where:

- \(B(t)\) is the point on the curve at parameter \(t\).

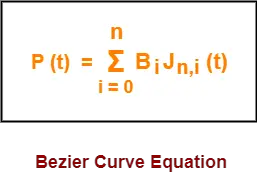
- \(t\) varies from 0 to 1.

- \(P0\) is the starting control point.

- \(P1\) is the control point that influences the direction of the curve.

- \(P2\) is the ending control point.

A bezier curve is parametrically represented by-



Here,

* t is any parameter where 0 <= t <= 1
* P(t) = Any point lying on the bezier curve
* Bi = ith control point of the bezier curve
* n = degree of the curve
* Jn,i(t) = Blending function = C(n,i)ti(1-t)n-i where C(n,i) = n! / i!(n-i)!

Algorithm for bezier curve :-

1.**Set up the Control Points:**

1.0*P*0 is the starting point of the curve.

2.1*P*1 and 2*P*2 are the control points that influence the direction and shape of the curve.

3.3*P*3 is the ending point of the curve.

2.**Compute the Curve Points:**

1.For each value of *t* ranging from 0 to 1, compute the corresponding point *B*(*t*) on the curve using the parametric equation.

3.**Draw the Curve:**

1.Connect consecutive points on the curve to visualize the Bézier curve.

•Let's go through a simple example with specific control points:

•0=(0,0)*P*0=(0,0)

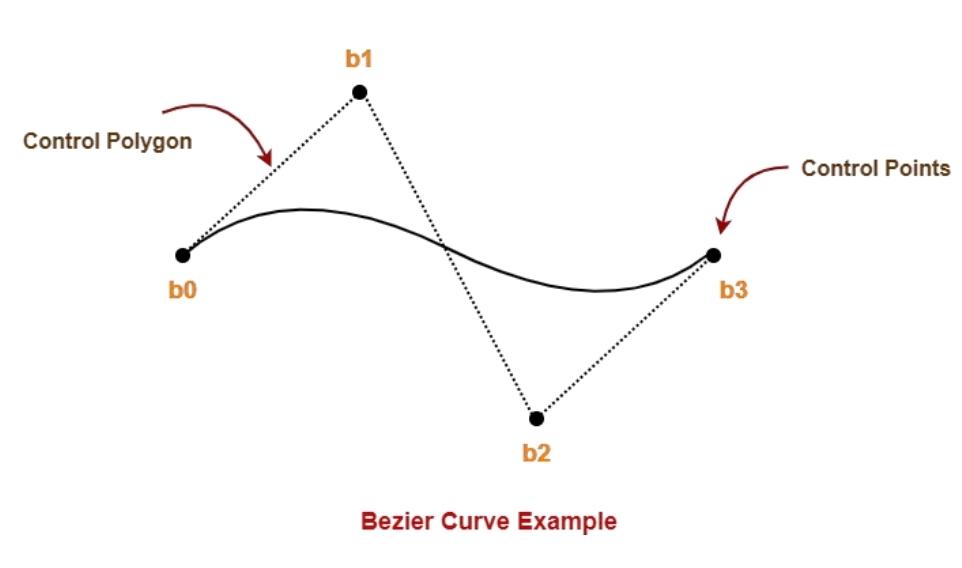
•1=(50,100)*P*1=(50,100)

•2=(150,−50)*P*2=(150,−50)

•3=(200,100)*P*3=(200,100)

•Now, compute points on the curve for various values of *t* (e.g,t=0,0.1,0.2,...,1*t*=0,0.1,0.2,...,1) and draw lines connecting these points. The resulting curve will smoothly interpolate between the control points.

The following curve is an example of a bezier curve-



Here,

* This bezier curve is defined by a set of control points b0, b1, b2 and b3.
* Points b0 and b3 are ends of the curve.
* Points b1 and b2 determine the shape of the curve.

-What is a B-Spline Curve?

A B-spline (Basis spline) is a piecewise-defined curve that is commonly used in computer graphics and computer-aided design. B-spline curves are defined by a set of control points and a degree value. The degree determines the number of control points that influence the shape of the curve. Unlike Bezier curves, B-spline curves offer more flexibility because they are defined by a local control mechanism.

Example:

|  |  |
| --- | --- |
| B-spline curve shape before changing the position of control point P1 – | B-spline curve shape after changing the position of control point P1 – |
|  |  |

You can see in the above figure that only the segment-1st shape as we have only changed the control point P1, and the shape of segment-2nd remains intact.

B-spline Curve Equation : The equation of the spline-curve is as follows –

Where

· {pi: i=0, 1, 2….n} are the control points

· k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree k – 1.

· the Ni,k(t) are the “normalized B-spline blending functions”. They are described by the order k and by a non-decreasing sequence of real numbers normally called the “knot sequence”.

**Ti : i=0,...n+K**

The Ni, k functions are described as follows –

Some cases of basis function:

## · Properties of B-spline Curve:

B-spline curves have the following properties −

· The sum of the B-spline basis functions for any parameter value is 1.

· Each basis function is positive or zero for all parameter values.

· Each basis function has precisely one maximum value, except for k=1.

· The maximum order of the curve is equal to the number of vertices of the defining polygon.

· The degree of the B-spline polynomial is independent on the number of vertices of a defining polygon.

· B-spline allows local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.

· The curve exhibits the variation diminishing property.

· The curve generally follows the shape of defining polygon.

· Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.

The curve line within the convex hull of its defining polygon

**Pseudo Code:**

function basisFunction(i, k, u, knots):

// Recursive definition of B-spline basis functions

if k == 1:

if knots[i] <= u < knots[i + 1]:

return 1

else:

return 0

else:

firstTerm = 0

secondTerm = 0

if knots[i + k - 1] != knots[i]:

firstTerm = (u - knots[i]) / (knots[i + k - 1] - knots[i]) \* basisFunction(i, k - 1, u, knots)

if knots[i + k] != knots[i + 1]:

secondTerm = (knots[i + k] - u) / (knots[i + k] - knots[i + 1]) \* basisFunction(i + 1, k - 1, u, knots)

return firstTerm + secondTerm

function bSplineCurve(controlPoints, knots, degree, u):

n = length(controlPoints) – 1

result = [0, 0] // Initialize result with 0 for both x and y coordinates

for i from 0 to n:

basis = basisFunction(i, degree + 1, u, knots)

result[0] += basis \* controlPoints[i][0] // x-coordinate

result[1] += basis \* controlPoints[i][1] // y-coordinate

return result

**Output:**

public struct point

{

public int x;

public int y;

public void setxy(int i,int j)

{

x=i;

y=j;

}

public void clearxy()

{

x=0;

y=0;

}

}

public void BSPLINE(point p1,point p2,point p3,point p4,int divisions)

{

double []a=new double[4];

double []b=new double[4];

a[0] = ( -p1.x + 3\*p2.x - 3\*p3.x + p4.x)/6.0;

a[1] = ( 3\*p1.x - 6\*p2.x + 3\*p3.x )/6.0;

a[2] = (-3\*p1.x + 3\*p3.x )/6.0;

a[3] = ( p1.x + 4\*p2.x + p3.x )/6.0;

b[0] = ( -p1.y + 3\*p2.y - 3\*p3.y + p4.y)/6.0;

b[1] = ( 3\*p1.y - 6\*p2.y + 3\*p3.y )/6.0;

b[2] = (-3\*p1.y + 3\*p3.y )/6.0;

b[3] = ( p1.y + 4\*p2.y + p3.y )/6.0;

spline\_out\_x[0] = a[3];

spline\_out\_y[0] = b[3];

for (int i = 1 ; i < divisions ; i++)

{

float t;

t = ((float) i) / ((float) divisions);

spline\_out\_x[i] = a[3] + t\*(a[2] + t\*(a[1] + t\*a[0]));

spline\_out\_y[i] = b[3] + t\*(b[2] + t\*(b[1] + t\*b[0]));

}

}

public void plus\_draw(int x,int y,int pen\_width,Color cl )

{

Graphics g = Graphics.FromHwnd(this.Handle);

g.DrawLine(new Pen(cl,pen\_width) ,x-3,y,x+3,y);

g.DrawLine(new Pen(cl,pen\_width) ,x,y-3,x,y+3);

}

public void Form\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

if (Movement\_Click>3)

{

pt[0]=pt[1];

pt[1]=pt[2];

pt[2]=pt[3];

pt[3].setxy(e.X,e.Y);

int no\_of\_interpolated\_points=(int)(Math.Sqrt(Math.Pow((pt[2].x-pt[1].x),2)+Math.Pow((pt[2].y-pt[1].y),2)));

BSPLINE(pt[0],pt[1],pt[2],pt[3],no\_of\_interpolated\_points);

for (int i=0;i<no\_of\_interpolated\_points;i++)

plus\_draw((int )spline\_out\_x[i],(int)spline\_out\_y[i],2,Color.Blue);

}

else

{

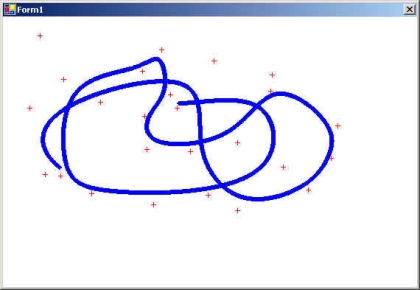
pt[Movement\_Click].setxy(e.X,e.Y);

}

Movement\_Click=Movement\_Click+1;

plus\_draw(e.X,e.Y,1,Color.Red);

}



**Conclusion:** I have understood how to implement curve plotting algorithms in CG.