8. Develop a menu driven program to animate a flag using Bezier curve algorithm.

Lets understand Bézier Curves first

Bézier curves are parametric curves that are generated with the control points. It is widely used in computer graphics and other related industry, as they appear reasonably smooth at all scales. Bézier curves was name after french engineer Pierre Bézier, who discovered it. Mathematically Bézier curves is represented as –

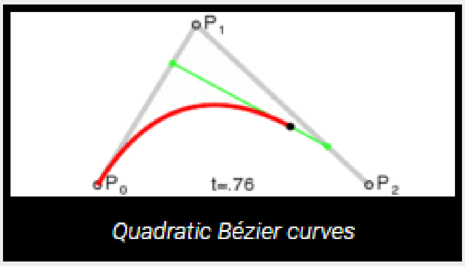
$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where p_i is the set of points and $B_i^n(t)$ represents the Bernstein polynomials which are given by –

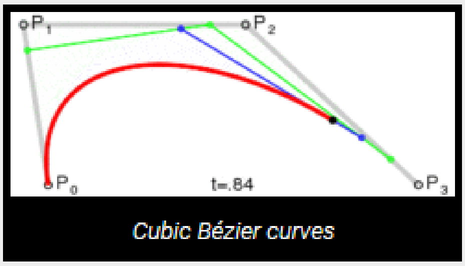
$$B_i^n(t) = \binom{n}{i}(1-t)^{n-t}t^i$$

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

Bézier curves are of different degree - linear curves, quadratic curve, cubic curve and high order curve.



Quadratic Bézier curves



Cubic Bézier curves

So basically we need to calculate

Bezier curve =	Berstein Polynomial		*	For every point
Bezier curve =	(nCr) *	$(1-t)^{n-t}t^i$	*	For every point

Where n = (number_of_control_points - 1)
= 4 - 1
n = 3

t ranges from 0 to 1

THE BASIC FLOW OF THIS CALCULATION IS:

Step 1: computeNcR
Step 2: bernstein_polynomial
Step 3: For every point

Finally – Multiply all

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
float theta = 0;

struct point
{
        GLfloat x, y, z;
};

int factorial (int n)
{
      if (n<=1)
        return (1);
      else
        n = n * factorial ( n-1 );
      return n;
}

void computeNcR (int n, int *hold_ncr_values)
{
      int r;
      for (r=0; r<=n; r++) //start from nC0, then nC1, nC2, nC3 till nCn
      {
        hold_ncr_values [r] = factorial (n) / ( factorial (n-r) * factorial (r) );
      }
}

void computeBezierPoints (float t, point *actual_bezier_point, int number_of_control_points,
                                 point *control_points_array, int *hold_ncr_values)  // 5 parameters
{
      int i, n = number_of_control_points - 1;

      float bernstein_polynomial;

      actual_bezier_point -> x = 0;
      actual_bezier_point -> y = 0;
      actual_bezier_point -> z = 0;

      for ( i=0; i<number_of_control_points; i++ )
       {
              bernstein_polynomial = hold_ncr_values [i]  * pow(t, i) * pow( 1-t, n-i);

              actual_bezier_point->x += bernstein_polynomial * control_points_array [i].x;
              actual_bezier_point->y += bernstein_polynomial * control_points_array [i].y;
              actual_bezier_point->z += bernstein_polynomial * control_points_array [i].z;
        }
}
```

See the above explanation to understand this

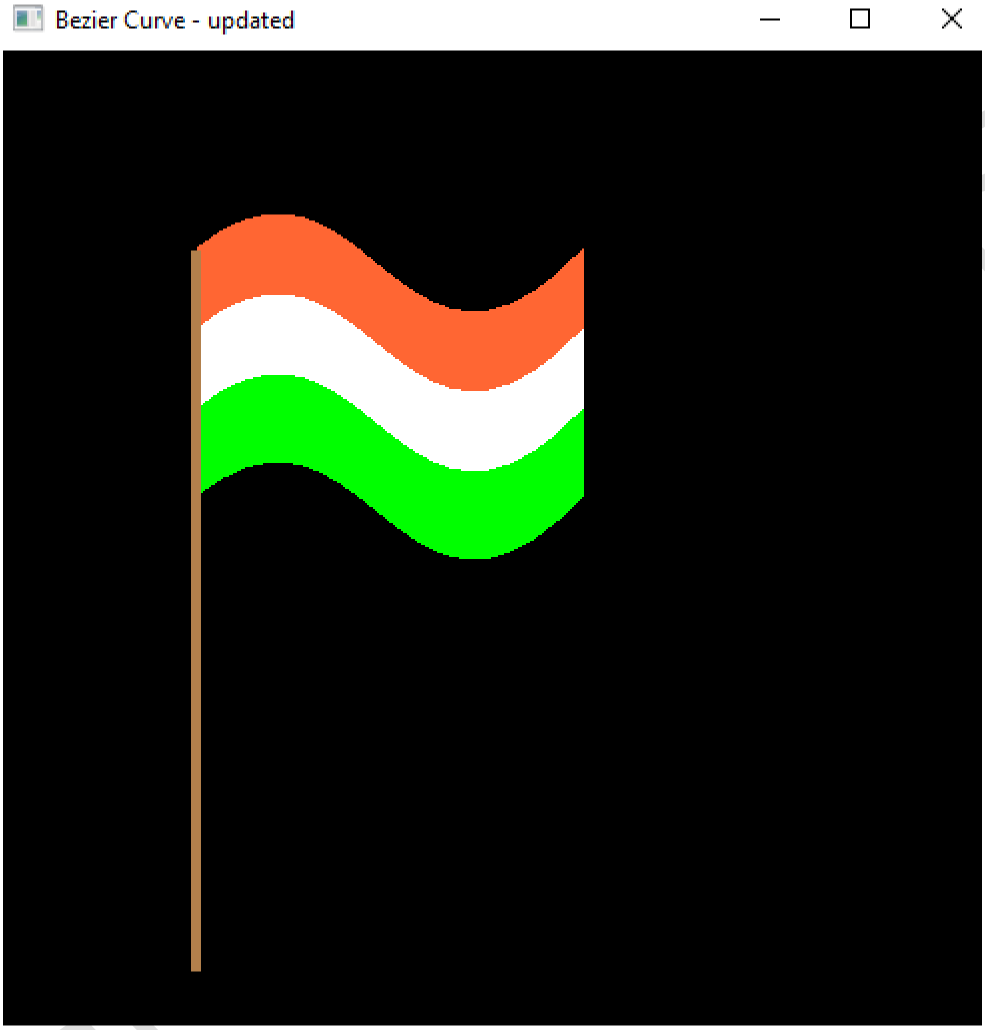
```
void Bezier (point *control_points_array, int number_of_control_points, int number_of_bezier_points)
{
        point actual_bezier_point;
        float t;
        int *hold_ncr_values, i;

        hold_ncr_values = new int [number_of_control_points]; // to hold the nCr values

        computeNcR (number_of_control_points - 1, hold_ncr_values); // calculate nCr values

        glBegin (GL_LINE_STRIP);
                for(i=0; i<=number_of_bezier_points; i++)
                 {
                    t=float (i) / float (number_of_bezier_points);

                    computeBezierPoints ( t, &actual_bezier_point, number_of_control_points,
                                                control_points_array, hold_ncr_values );// 5 parameters

                    glVertex2f (actual_bezier_point.x, actual_bezier_point.y);
                }
        glEnd ();

        delete [] hold_ncr_values;
}
```

See the above explanation to understand this

```
void display()
{
        glClear (GL_COLOR_BUFFER_BIT);
        int number_of_control_points= 4, number_of_bezier_points= 20;

        point control_points_array[4]= {{100, 400, 0}, {150, 450, 0}, {250, 350, 0},{300, 400, 0}};

        control_points_array[1].x += 50 * sin (theta * PI/180.0);      // for animating the flag
        control_points_array[1].y += 25 * sin (theta * PI/180.0);

        control_points_array[2].x -= 50 * sin ((theta+30) * PI/180.0)
        control_points_array[2].y -= 50 * sin ((theta+30) * PI/180.0)

        control_points_array[3].x -= 25 * sin ((theta-30) * PI/180.0)
        control_points_array[3].y += sin ((theta-30) * PI/180.0);

        theta += 2;                    //animating speed

        glPushMatrix ();

        glPointSize (5);               // for plotting the point
        glColor3f (1, 0.4, 0.2); //Indian flag: Saffron color code
        for (int i=0; i<50; i++)
        {
                glTranslatef(0, -0.8, 0 );
                bezier(control_points_array, number_of_control_points, number_of_bezier_points);
        }

        glColor3f(1, 1, 1); //Indian flag: white color code
        for(int i=0; i<50; i++)
        {
                glTranslatef(0, -0.8, 0);
                bezier(control_points_array, number_of_control_points, number_of_bezier_points);
        }

        glColor3f(0, 1, 0); //Indian flag: green color code
        for(int i=0; i<50; i++)
        {
                glTranslatef(0, -0.8, 0);
                bezier(control_points_array, number_of_control_points, number_of_bezier_points);
        }

        glPopMatrix();

        glLineWidth(5);

        glColor3f(0.7, 0.5,0.3); //pole colour

        glBegin(GL_LINES);
                glVertex2f(100,400);
                glVertex2f(100,40);
        glEnd();

        glutPostRedisplay();            // call display again
        glutSwapBuffers();              // show the output
}

void init ()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,500,0,500);
}

int main(int argc, char ** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowPosition(0, 0);
        glutInitWindowSize(500,500);
        glutCreateWindow ("Bezier Curve - updated");

        init ();

        glutDisplayFunc (display);

        glutMainLoop ();
}
```

****

OUTPUT


Bezier Curve - updated