Name:Harshada Gopal Rayate

Roll No:19 SEDA

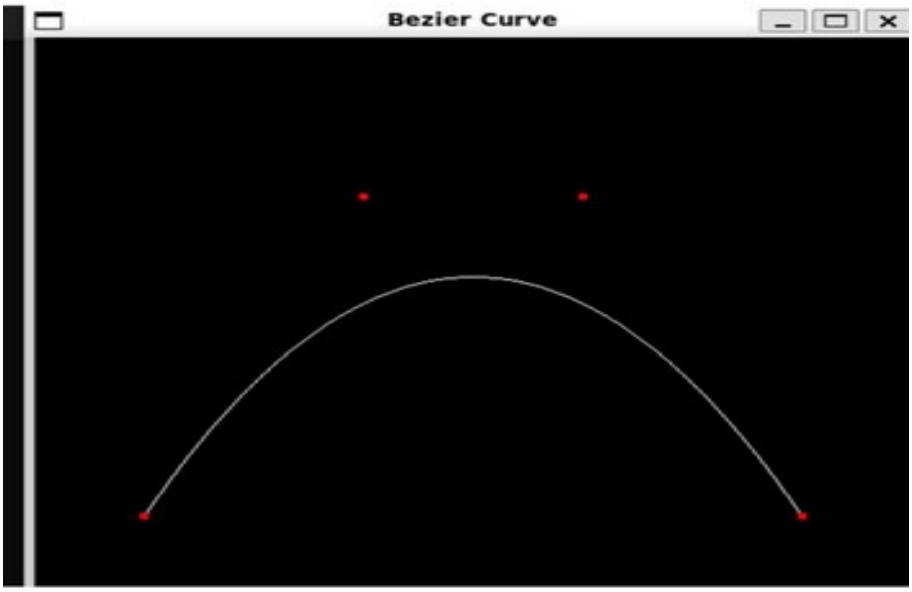
Subject :CGAVR

Curve Assignment

```
#include <GL/glut.h>
#include <iostream>

// Control points for the Bezier curve
GLfloat controlPoints[3][3] = {
    {-0.8f, -0.8f, 0.0f},  // Point 1
    {0.0f, 0.8f, 0.0f},    // Point 2
    {0.8f, -0.8f, 0.0f}    // Point 3
};

// Function to compute the Bezier curve points
void computeBezier(GLfloat t) {
    GLfloat x = (1 - t) * (1 - t) * controlPoints[0][0] + 2 * (1 - t) * t * controlPoints[1][0] + t * t * controlPoints[2][0];
    GLfloat y = (1 - t) * (1 - t) * controlPoints[0][1] + 2 * (1 - t) * t * controlPoints[1][1] + t * t * controlPoints[2][1];

    glVertex2f(x, y);
}

// Function to draw the Bezier curve
void drawBezierCurve() {
    glBegin(GL_LINE_STRIP);  // Draw a line strip to represent the Bezier curve
```

```
    for (float t = 0.0f; t <= 1.0f; t += 0.01f) {
        computeBezier(t);
    }
    glEnd();
}

// Function to display the scene
void display() {
    glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen

    glColor3f(1.0f, 0.0f, 0.0f);  // Set color to red
    drawBezierCurve();            // Draw the Bezier curve

    glColor3f(0.0f, 0.0f, 1.0f);  // Set color to blue
    glPointSize(5.0f);            // Set point size
    glBegin(GL_POINTS);           // Draw the control points
    for (int i = 0; i < 3; i++) {
        glVertex2fv(controlPoints[i]);
    }
    glEnd();

    glutSwapBuffers();  // Swap buffers for double buffering
}

// Function to initialize OpenGL
void initGL() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);  // Set background color to white
```

```
    glMatrixMode(GL_PROJECTION);          // Set projection matrix
    glLoadIdentity();                     // Reset projection matrix
    gluOrtho2D(-1.0f, 1.0f, -1.0f, 1.0f);  // Set orthographic view
}

// Main function
int main(int argc, char** argv) {
    glutInit(&argc, argv);                // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);  // Set display mode
    glutInitWindowSize(600, 600);         // Set window size
    glutCreateWindow("Quadratic Bezier Curve");  // Create window

    initGL();                             // Initialize OpenGL settings
    glutDisplayFunc(display);             // Register display function

    glutMainLoop();                       // Enter GLUT main loop
    return 0;
}
```



B-spline Curve

```
#include <GL/glut.h>
#include <vector>
#include <iostream>
#include <cmath>
using namespace std;

// Define a point structure
struct Point {
    double x, y;
};

// Function to calculate basis function (Blend function)
double blend(vector<double> &uVec, double u, int k, int d) {
    if (d == 1) {
        if (uVec[k] <= u && u < uVec[k + 1])
            return 1.0;
        return 0.0;
    }
    double left = (u - uVec[k]) / (uVec[k + d - 1] - uVec[k]);
    double right = (uVec[k + d] - u) / (uVec[k + d] - uVec[k + 1]);

    // Handle cases where denominator might be zero
```

```
    if (uVec[k + d - 1] == uVec[k]) left = 0;
    if (uVec[k + d] == uVec[k + 1]) right = 0;

    return left * blend(uVec, u, k, d - 1) + right * blend(uVec, u, k + 1, d - 1);
}

// Function to draw B-Spline curve
void drawBSplineCurve(vector<Point> poly) {
    int n = poly.size();
    int degree;
    cout << "Enter degree of curve: ";
    cin >> degree;

    // Generate uniform knot vector
    vector<double> uVec;
    for (int i = 0; i < n + degree; i++) {
        uVec.push_back(static_cast<double>(i) / (n + degree - 1));
    }

    // Draw the B-Spline curve
    glColor3f(1.0, 0.0, 0.0); // Red color for the curve
    glBegin(GL_LINE_STRIP);
    for (double u = 0; u <= 1; u += 0.001) {
        double x = 0, y = 0;
        for (int i = 0; i < poly.size(); i++) {
            double basis = blend(uVec, u, i, degree);
```

```
            x += basis * poly[i].x;
            y += basis * poly[i].y;
        }
        glVertex2f(x, y);
    }
    glEnd();

    // Draw control points
    glColor3f(0.0, 0.0, 1.0); // Blue for control points
    glPointSize(5.0);
    glBegin(GL_POINTS);
    for (const auto &p : poly) {
        glVertex2f(p.x, p.y);
    }
    glEnd();

    // Draw control polygon
    glColor3f(0.0, 1.0, 0.0); // Green for control polygon
    glBegin(GL_LINE_STRIP);
    for (const auto &p : poly) {
        glVertex2f(p.x, p.y);
    }
    glEnd();

    glFlush();
}
```

```
// Callback function for display
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Define control points
    vector<Point> controlPoints = {
        {50, 200}, {150, 400}, {250, 100},
        {350, 300}, {450, 200}, {550, 400}
    };

    drawBSplineCurve(controlPoints);
}

// Main function
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("B-Spline Curve");

    glClearColor(1.0, 1.0, 1.0, 0.0); // White background
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0); // Set 2D viewing area
```

```
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```