UNIT V
CURVED SURFACES

In OpenGL, working with Bezier and B-Spline curves typically requires understanding how to represent and render them. OpenGL itself does not provide direct functions for drawing these curves, so you generally implement them using control points and associated mathematical formulas. Below, I'll outline a basic approach for both Bezier and B-Spline curves using OpenGL.

1. Bezier Curves

Bezier curves are commonly represented by control points, and the curve is evaluated using a polynomial formula.

Bezier Curve Formula:

A cubic Bezier curve (commonly used in graphics) with 4 control points P_0, P_1, P_2, P_3 can be calculated as:

$$B(t) = (1-t)^3 \cdot P_0 + 3 \cdot (1-t)^2 \cdot t \cdot P_1 + 3 \cdot (1-t) \cdot t^2 \cdot P_2 + t^3 \cdot P_3$$

Where t is the parameter between 0 and 1.

OpenGL Code for Bezier Curve:

```
#include <GL/glut.h>
struct Point {
   float x, y;
};

Point P0 = {100, 100};
Point P1 = {150, 250};
Point P2 = {250, 250};
Point P3 = {300, 100};

Point bezier(float t, Point P0, Point P1, Point P2, Point P3) {
   Point result;
   result.x = (1-t)*(1-t)*(1-t)*P0.x + 3*(1-t)*(1-t)*t*P1.x + 3*(1-t)*t*t*P2.x + t*t*t*P3.x;
   result.y = (1-t)*(1-t)*(1-t)*P0.y + 3*(1-t)*(1-t)*t*P1.y + 3*(1-t)*t*t*P2.y + t*t*t*P3.y;
   return result;
}
```

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINE_STRIP);
    for (float t = 0; t <= 1.0; t += 0.01) {
        Point p = bezier(t, P0, P1, P2, P3);
        glVertex2f(p.x, p.y);
    }
    glEnd();

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Bezier Curve");
    glutDisplayFunc(display);
    gluOrtho2D(0.0, 400.0, 0.0, 400.0);
    glutMainLoop();
    return 0;
}
```

This code defines a cubic Bezier curve and displays it by evaluating the curve for multiple values of **t**.

2. B-Spline Curves

B-Splines (Basis Splines) are more general than Bezier curves and can represent curves with more flexibility. They are typically defined by a set of control points and a set of basis functions. In the case of a **uniform B-Spline**, the basis functions are calculated using a recursive formula, and the B-spline curve is evaluated by summing over the basis functions and control points.

For simplicity, let's assume we're working with a **quadratic B-Spline** with a set of control points.

B-Spline Curve Formula (Quadratic):

Given control points $P_0, P_1, P_2, P_3, \ldots$, the B-spline curve can be evaluated using the following formula for a quadratic B-Spline:

$$S(t) = \sum_{i=0}^{n} N_{i,2}(t) \cdot P_i$$

Where $N_{i,2}(t)$ are the quadratic B-spline basis functions, and t is the parameter that varies from 0 to 1.

OpenGL Code for B-Spline Curve:

```
#include <GL/glut.h>
struct Point {
    float x, y;
};
Point controlPoints[] = {{100, 100},{150, 250},{250, 250},{300, 100}};
float basisFunction(float t, int i, int degree) {
    if (degree == 2) {
        // Quadratic B-Spline basis function
        if (t >= i && t < i + 1) return t - i;
        if (t >= i + 1 && t < i + 2) return 2 - (t - i);
        return 0;
    }
    return 0;
}
Point bSpline(float t) {
    Point result = {0, 0};
    for (int i = 0; i < 3; i++) {  // Only 3 segments for this example
        float basis = basisFunction(t, i, 2);
        result.x += basis * controlPoints[i].x;
        result.y += basis * controlPoints[i].y;
    }
    return result;
}
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINE_STRIP);
    for (float t = 0; t <= 2.0; t += 0.01) {  // t from 0 to 2 for quadratic
        Point p = bSpline(t);
        glVertex2f(p.x, p.y);
    }
    glEnd();

    glFlush();
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("B-Spline Curve");
    glutDisplayFunc(display);
    gluOrtho2D(0.0, 400.0, 0.0, 400.0);
    glutMainLoop();
    return 0;
}
```

This example shows how to implement a basic quadratic B-Spline curve. It uses a simple basis function that is not a full general B-Spline implementation, but it gives you the idea of how to evaluate the curve using the basis functions and control points.

Conclusion

- **Bezier curves** are evaluated using polynomial formulas based on the number of control points (e.g., cubic for 4 points).
- **B-Spline curves** are more flexible and involve evaluating a set of basis functions. OpenGL does not directly support these types of curves, so you need to implement them yourself using the appropriate mathematical formulas.

For more complex B-Splines (like **Non-Uniform Rational B-Splines (NURBS)**), there are libraries like **OpenNURBS** that provide advanced support for NURBS curves and surfaces, which you can integrate with OpenGL.