Computer Graphics Assignment Report

Assignment:	**Assignment #2 Curves and surface generation**
Name:	**游忠敏**
ID#:	**11619370216**
Date of Experiment:	**2019.4.27**
Date of Report:	**2019.4.29**
Submission:	**2019.4.30**

Ⅰ OBJECTIVES

Using OpenGL to generate Bézier surface and B Spline surface. Input 16 control points to render the 3 order Bézier surface and B Spline surface. The knots could be uniform or nonuniform.

Ⅱ THEORETICAL BACKGROUND

Background of Bezier Curve

Given n + 1 points, P_0 (x_0, y_0),... P_n (x_n, y_n) generates a curve that matches the shape described by these points. If the curve is required to pass through all data points, it belongs to the “Interpolation”. If only the curve is required to approach these data points, it belongs to the “approximation”.

Approximation is mainly used in computer graphics to design beautiful or aesthetic curves. In order to solve this problem, it is necessary to find a method of constructing curves with small parts, namely curve segments, to meet the design standards. When curve segments are used to fit curve f(x), the curve can be expressed as the sum of many small segments φ i(x), in which φ i(x)is called a base (mixed) function.

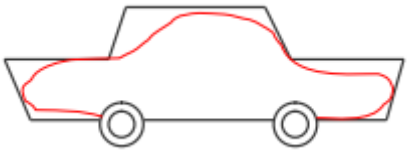
$$f(x) = \sum_{i=0}^{n} a_i f_i(x)$$

These base (mixed) functions are used for calculation and display. Therefore, polynomials are often chosen as basis (mixed) functions, and Bezier constructs a parametric curve and surface design method based on approximation.

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

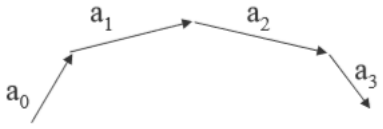
This idea is based on the idea that when designing a car's contour, the outline of the car's contour is first drawn by a broken line segment, and then the polygon is approached by a smooth parametric curve. This polygon is called characteristic polygon. The curve approaching the characteristic polygon is called Bezier curve. Bezier method combines function approximation with geometric representation, which makes Bezier curve as handy as drawing tool on computer and widely used in many graphic and image software, such as Flash,

Illstrator, CoralDRAW and Photoshop.



Bezier expressed the parametric n-th curve as follows:

$$p(t) = \sum_{i=0}^{n} a_i f_{i,n}(t) \qquad 0 \le t \le 1$$



The coefficient vector Ai (i = 0,1,... n) The order is head-to-tail. The polygons formed from the end of A0 to the end of an are called control polygons or Bezier polygons.

$$p(t) = \sum_{i=0}^{n} a_i f_{i,n}(t) \qquad 0 \le t \le 1$$

$$f_{i,n}(t) = \begin{cases} 1, & i = 0, \\ \dfrac{(-t)^i}{(i-1)!} \dfrac{d^{i-1}}{dt^{i-1}} \left(\dfrac{(1-t)^{n-1} - 1}{t} \right) \end{cases}$$

In 1972, Forest proved that the basis function of Bezier curve can be simplified to Bernstein basis function. A continuous function y=f(x) can always find a polynomial and approximate the function sufficiently if a $\xi > 0$ is allowed. Bernstein has a theory of approximation in the form of:

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} \qquad (i = 0, 1, \ldots n)$$

For Bezier curves, the position vector Pi (i = 0, 1, 2,... The parametric equation of Bezier curve segment is expressed as follows:

$$p(t) = \sum_{i=0}^{n} P_i B_{i,n}(t) \qquad t \in [0,1]$$

Where Pi (xi, yi, zi), i = 0, 1, 2... n is the n + 1 vertex of the control polygon, that is, the characteristic polygon of the curve; Bi, n (t) is Bernstein basis function, which has the following forms:

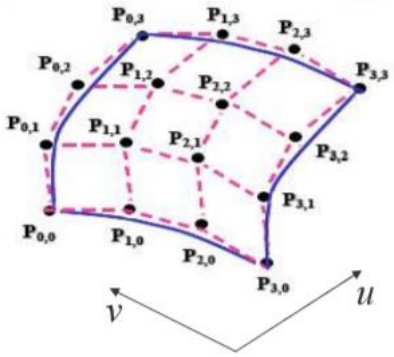
$$B_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} = C_n^i t^i (1-t)^{n-i} \qquad (i = 0,1,\ldots. n)$$

Based on the discussion of Bezier curves, the definition and properties of Bezier surfaces can be given. Some algorithms of Bezier curves can also be easily extended to the case of Bezier surfaces.

$$P(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} B_{i,m}(u) B_{j,n}(v) \qquad u,v \in [0,1]$$

$$B_{i,m}(u) = C_m^i u^i (1-u)^{m-i}$$

$$B_{i,n}(v) = C_n^j v^j (1-v)^{n-j}$$



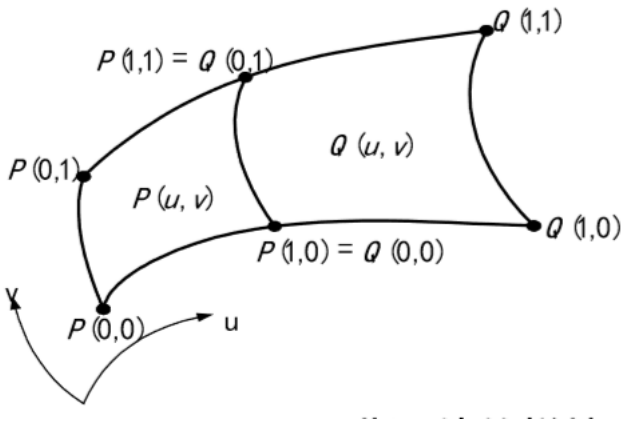
The matrix representation of Bezier surface is:

$$P(u,v) = \left[B_{0,n}(u), B_{1,n}(u), \mathbf{L}, B_{m,n}(u)\right] \begin{bmatrix} P_{00} & P_{01} & \mathbf{L} & P_{0m} \\ P_{10} & P_{11} & \mathbf{L} & P_{1m} \\ \mathbf{L} & \mathbf{L} & \mathbf{L} & \mathbf{L} \\ P_{n0} & P_{n1} & \mathbf{L} & P_{nm} \end{bmatrix} \begin{bmatrix} B_{0,m}(v) \\ B_{1,m}(v) \\ \mathbf{L} \\ B_{n,m}(v) \end{bmatrix}$$

Mosaic of Bezier Surface Patches:

$$P(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} B_{i,m}(u) B_{j,n}(v)$$

$$Q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} Q_{ij} B_{i,m}(u) B_{j,n}(v) \qquad u, v \in [0,1]$$



$$P(1,v) = Q(0,v)$$

$$P_{ni} = Q_{0i}, \quad (i = 0,1,\mathbf{L},m)$$

Background of B-spline

Bezier curves and surfaces have many advantages. They can change the shape of curves by dragging control vertices with the mouse. It is very intuitive and gives designers a lot of freedom. Bezier curves and surfaces are the main methods and tools for geometric modeling.

But Bezier curve has some shortages:

1. Once the number of vertices of the characteristic polygon (n + 1) is determined, the order of the curve (n times) is determined.

2. The stitching of Bezier curves or surfaces is more complicated

3. Bezier curve or surface can not be modified locally. In order to correct deviation, when moving part of the curve, the other part will be forced to move away from the desired curve.

The interval whose value is not zero is usually called its support interval, because every Bernstein polynomial is supported on the whole interval [0, 1], and the curve is a mixture of these functions, so each control vertex has an effect on the curve at the T value between 0 and 1.

In order to overcome the weakness of Bezier method, while retaining all the advantages of Bezier method, researchers proposed spline, piecewise continuous polynomial.

The whole curve is expressed in a complete form, but the intrinsic quantity is a series of

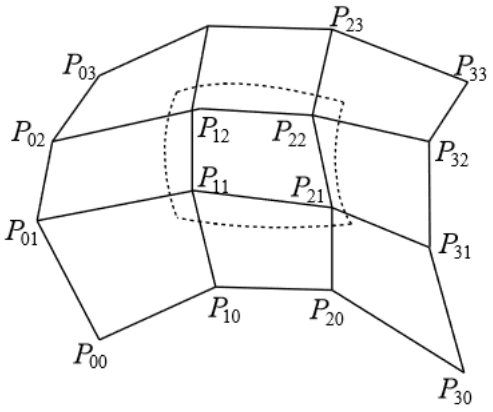
sections, such as a pile of three-degree curves pieced together, two consecutive times between the two, which not only overcomes the fluctuation phenomenon, but also the curve is low. There are both unified expressions and unified algorithms.

B Spline surface:

$$U = [u_0, u_1, \mathbf{L}, u_{m+p}]$$

$$V = [v_0, v_1, \mathbf{L}, v_{n+q}]$$

$$P(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{i,j} N_{i,p}(u) N_{j,q}(v)$$



Ⅲ PROCEDURES

In order to form a surface, first we need 16 uniform points to define the edges of surface.

```
GLfloat UniformPoints[4][4][3] =
{
    {
        { -1.0f, -0.4f, 0.5f }, { -0.4f, -0.8f, 0.2f }, { 0.2f, -0.65f, 0.3f }, { 0.7f, -0.7f, 0.2f }
    },
    {
        { -0.9f, -0.2f, 0.3f }, { -0.3f, -0.4f, 0.2f }, { 0.3f, -0.2f, 0.4f }, { 0.75f, -0.2f, 0.3f },
    },
    {
        { -0.9f, 0.3f, 0.3f }, { -0.3f, 0.2f, 0.5f }, { 0.25f, 0.25f, 0.6f }, { 0.8f, 0.3f, 0.3f },
    },
    {
        { -0.8f, 0.8f, 0.1f }, { -0.2f, 0.8f, 0.2f }, { 0.2f, 0.85f, 0.1f }, { 0.7f, 0.8f, 0.1f },
    }
};
```

According to Bezier fuction, we can calculate the 3 order function as follow:

$$B_{0,3}(t) = \frac{3!}{0!\times 3!} t^0 (1-t)^3 = (1-t)^3$$
$$B_{1,3} = \frac{3!}{1!\times 2!} t(1-t)^2 = 3t(1-t)^2$$
$$B_{2,3}(t) = \frac{3!}{2!\times 1!} t^2 (1-t) = 3t^2(1-t)$$
$$B_{3,3}(t) = \frac{3!}{3!\times 0!} t^3 (1-t)^0 = t^3$$

I need to construct coefficient as follow:

```
double a1 = pow((1 - t), 3);
double a2 = pow((1 - t), 2) * 3 * t;
double a3 = 3 * t*t*(1 - t);
double a4 = pow(t, 3);//三次Bezier的系数
```

For the construction curve, first construct from X direction, then construct from y direction, the code is as follows:

```
GLfloat calculate1[11][4][3];//先一个方向求bezier曲线，再求另一个方向
for (int j = 0; j < 4; j++) {
    GLint i = 0;
    for (double t = 0.0; t <= 1.0; t += 0.1)
    {

        double a1 = pow((1 - t), 3);
        double a2 = pow((1 - t), 2) * 3 * t;
        double a3 = 3 * t*t*(1 - t);
        double a4 = pow(t, 3);//三次Bezier的系数
        calculate1[i][j][0] = a1 * UniformPoints[0][j][0] + a2 * UniformPoints[1][j][0] + a3 * UniformPoints[2][j][0] + a4 * UniformPoints[3][j][0];
        calculate1[i][j][1] = a1 * UniformPoints[0][j][1] + a2 * UniformPoints[1][j][1] + a3 * UniformPoints[2][j][1] + a4 * UniformPoints[3][j][1];
        calculate1[i][j][2] = a1 * UniformPoints[0][j][2] + a2 * UniformPoints[1][j][2] + a3 * UniformPoints[2][j][2] + a4 * UniformPoints[3][j][2];
        i = i + 1;
    }
}
```

```
GLfloat calculate2[11][11][3];
for (int j = 0; j < 11; j++) {
    GLint i = 0;
    for (double t = 0.0; t <= 1.0; t += 0.1)
    {
        double a1 = pow((1 - t), 3);
        double a2 = pow((1 - t), 2) * 3 * t;
        double a3 = 3 * t*t*(1 - t);
        double a4 = pow(t, 3);
        calculate2[j][i][0] = a1 * calculate1[j][0][0] + a2 * calculate1[j][1][0] + a3 * calculate1[j][2][0] + a4 * calculate1[j][3][0];
        calculate2[j][i][1] = a1 * calculate1[j][0][1] + a2 * calculate1[j][1][1] + a3 * calculate1[j][2][1] + a4 * calculate1[j][3][1];
        calculate2[j][i][2] = a1 * calculate1[j][0][2] + a2 * calculate1[j][1][2] + a3 * calculate1[j][2][2] + a4 * calculate1[j][3][2];
        i = i + 1;
    }
}
```

According to B-Spline fuction, we can calculate the 3 order function as follow:

$$B_0(t) = (1-t)^3/6$$
$$B_1(t) = (3t^3 - 6t^2 + 4)/6$$
$$B_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6$$
$$B_3(t) = t^3/6\ ,\quad t \in [0,1]$$

I need to construct coefficient as follow:

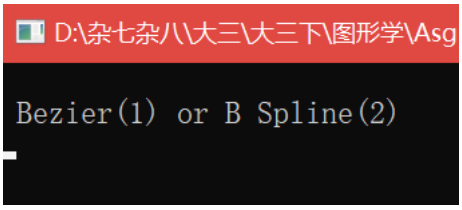
```
double a1 = (pow((1 - t), 3)) / 6;
double a2 = (3 * pow(t, 3) - 6 * pow(t, 2) + 4) / 6;
double a3 = (3 * (pow(t, 2) + t - pow(t, 3)) + 1) / 6;
double a4 = (pow(t, 3)) / 6;//三次B样条的系数
```

For the construction curve, first construct from X direction, then construct from y direction, the code is as follows:

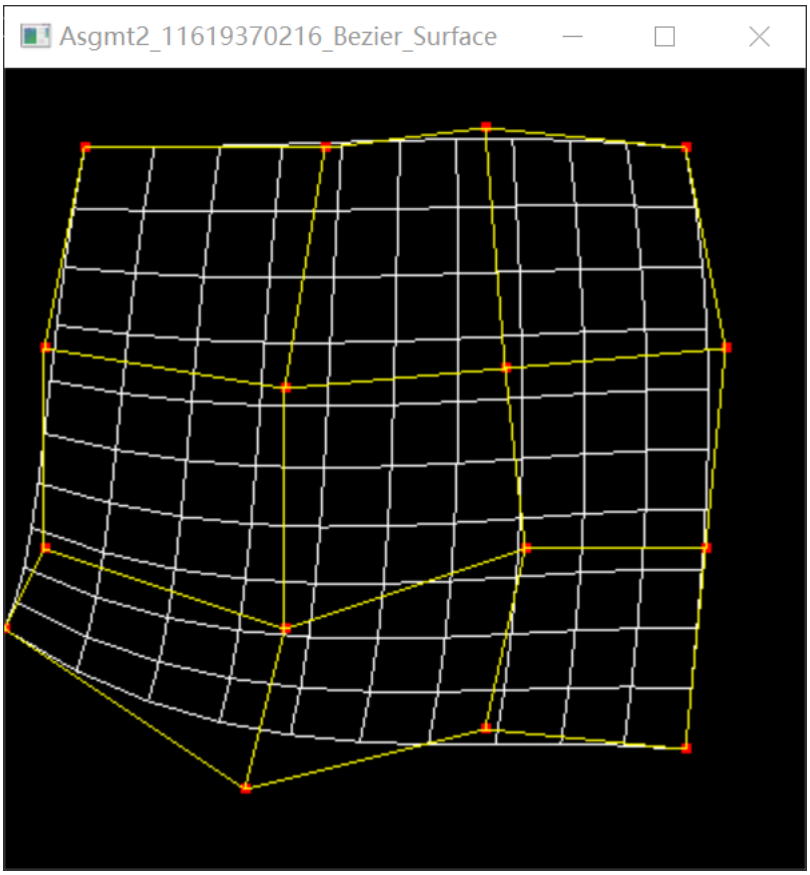
```
GLfloat calculate1[11][4][3];//先一个方向求B样条曲线，再求另一个方向
for (int j = 0; j < 4; j++) {
    GLint i = 0;
    for (double t = 0.0; t <= 1.0; t += 0.1)
    {
        double a1 = (pow((1 - t), 3)) / 6;
        double a2 = (3 * pow(t, 3) - 6 * pow(t, 2) + 4) / 6;
        double a3 = (3 * (pow(t, 2) + t - pow(t, 3)) + 1) / 6;
        double a4 = (pow(t, 3)) / 6;//三次B样条的系数
        calculate1[i][j][0] = a1 * UniformPoints[0][j][0] + a2 * UniformPoints[1][j][0] + a3 * UniformPoints[2][j][0] + a4 * UniformPoints[3][j][0];
        calculate1[i][j][1] = a1 * UniformPoints[0][j][1] + a2 * UniformPoints[1][j][1] + a3 * UniformPoints[2][j][1] + a4 * UniformPoints[3][j][1];
        calculate1[i][j][2] = a1 * UniformPoints[0][j][2] + a2 * UniformPoints[1][j][2] + a3 * UniformPoints[2][j][2] + a4 * UniformPoints[3][j][2];
        i = i + 1;
    }
}
```

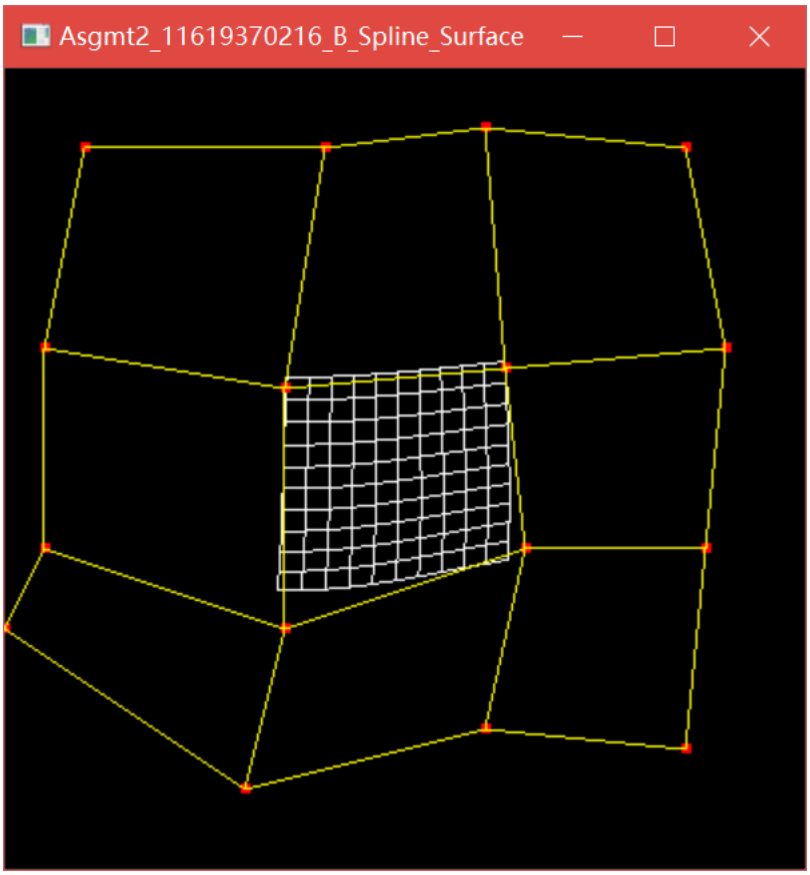
```
GLfloat calculate2[11][11][3];
for (int j = 0; j < 11; j++) {
    GLint i = 0;
    for (double t = 0.0; t <= 1.0; t += 0.1)
    {
        double a1 = (pow((1 - t), 3)) / 6;
        double a2 = (3 * pow(t, 3) - 6 * pow(t, 2) + 4) / 6;
        double a3 = (3 * (pow(t, 2) + t - pow(t, 3)) + 1) / 6;
        double a4 = (pow(t, 3)) / 6;
        calculate2[j][i][0] = a1 * calculate1[j][0][0] + a2 * calculate1[j][1][0] + a3 * calculate1[j][2][0] + a4 * calculate1[j][3][0];
        calculate2[j][i][1] = a1 * calculate1[j][0][1] + a2 * calculate1[j][1][1] + a3 * calculate1[j][2][1] + a4 * calculate1[j][3][1];
        calculate2[j][i][2] = a1 * calculate1[j][0][2] + a2 * calculate1[j][1][2] + a3 * calculate1[j][2][2] + a4 * calculate1[j][3][2];
        i = i + 1;
    }
```

IV RESULTS



Press 1, the result is Bezier surface, while press 2, the result is B-Spline surface:





V DISCUSSION

Bézier curves are widely used in computer graphics to model smooth curves. As the curve is completely contained in the convex hull of its control points, the points can be graphically displayed and used to manipulate the curve intuitively. Affine transformations such as translation and rotation can be applied on the curve by applying the respective transform on the control points of the curve.

B-spline function is a combination of flexible bands that passes through the number of points that are called control points and creates smooth curves. These functions enable the creation and management of complex shapes and surfaces using a number of points. B-spline function and Bézier functions are applied extensively in shape optimization methods.

VI CONCLUSIONS

The respective surfaces of Bezier curve and B-spline curve make me fully understand the algorithm of image smoothing and deepen my understanding of graphics.

APPENDIX: C/JAVA/Python code

```
#include <GL/glut.h>
#include <stdio.h>
#include <Windows.h>
#include <stdlib.h>
#include <math.h>

GLfloat UniformPoints[4][4][3] =
{
    {
        { -1.0f, -0.4f, 0.5f }, { -0.4f, -0.8f, 0.2f }, { 0.2f, -0.65f, 0.3f }, { 0.7f, -0.7f, 0.2f }
    },
    {
        { -0.9f, -0.2f, 0.3f }, { -0.3f, -0.4f, 0.2f }, { 0.3f, -0.2f, 0.4f }, { 0.75f, -0.2f,
0.3f },
    },
    {
        { -0.9f, 0.3f, 0.3f }, { -0.3f, 0.2f, 0.5f }, { 0.25f, 0.25f, 0.6f },  { 0.8f, 0.3f,
0.3f },
    },
    {
        { -0.8f, 0.8f, 0.1f }, { -0.2f, 0.8f, 0.2f }, { 0.2f, 0.85f, 0.1f },   { 0.7f, 0.8f,
0.1f },
    }
};
```

```
void Bezierrr(void)
{

    glClear(GL_COLOR_BUFFER_BIT);
    GLfloat calculate1[11][4][3];//先一个方向求bezier曲线，再求另一个方向
    for (int j = 0; j < 4; j++) {
        GLint i = 0;
        for (double t = 0.0; t <= 1.0; t += 0.1)
        {

            double a1 = pow((1 - t), 3);
            double a2 = pow((1 - t), 2) * 3 * t;
            double a3 = 3 * t*t*(1 - t);
            double a4 = pow(t, 3);//三次Bezier的系数
            calculate1[i][j][0] = a1 * UniformPoints[0][j][0] + a2 * UniformPoints[1][j][0] + a3
* UniformPoints[2][j][0] + a4 * UniformPoints[3][j][0];
            calculate1[i][j][1] = a1 * UniformPoints[0][j][1] + a2 * UniformPoints[1][j][1] + a3
* UniformPoints[2][j][1] + a4 * UniformPoints[3][j][1];
            calculate1[i][j][2] = a1 * UniformPoints[0][j][2] + a2 * UniformPoints[1][j][2] + a3
* UniformPoints[2][j][2] + a4 * UniformPoints[3][j][2];
            i = i + 1;
        }
    }



    GLfloat calculate2[11][11][3];
    for (int j = 0; j < 11; j++) {
        GLint i = 0;
        for (double t = 0.0; t <= 1.0; t += 0.1)
        {
            double a1 = pow((1 - t), 3);
            double a2 = pow((1 - t), 2) * 3 * t;
            double a3 = 3 * t*t*(1 - t);
            double a4 = pow(t, 3);
            calculate2[j][i][0] = a1 * calculate1[j][0][0] + a2 * calculate1[j][1][0] + a3 *
calculate1[j][2][0] + a4 * calculate1[j][3][0];
            calculate2[j][i][1] = a1 * calculate1[j][0][1] + a2 * calculate1[j][1][1] + a3 *
calculate1[j][2][1] + a4 * calculate1[j][3][1];
            calculate2[j][i][2] = a1 * calculate1[j][0][2] + a2 * calculate1[j][1][2] + a3 *
calculate1[j][2][2] + a4 * calculate1[j][3][2];
            i = i + 1;
```

```
        }
    }


    glColor3f(1.0, 1.0, 1.0);
    for (int i = 0; i < 11; i++) {
        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 11; j++)
            glVertex3fv(&calculate2[i][j][0]);
        glEnd();

        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 11; j++)
            glVertex3fv(&calculate2[j][i][0]);
        glEnd();
    }


    glPointSize(5.0);
    glColor3f(1.0, 0.0, 0.0);

    for (int i = 0; i < 4; i++) {
        glBegin(GL_POINTS);
        for (int j = 0; j < 4; j++)
            glVertex3fv(&UniformPoints[i][j][0]);
        glEnd();
    }//绘制点


    glColor3f(1.0, 1.0, 0.0);
    for (int i = 0; i < 4; i++) {
        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 4; j++)
            glVertex3fv(&UniformPoints[i][j][0]);
        glEnd();
        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 4; j++)
            glVertex3fv(&UniformPoints[j][i][0]);
        glEnd();
    }//连线

    glFlush();
}
```

```
void B_Spline(void)
{

    glClear(GL_COLOR_BUFFER_BIT);


    GLfloat calculate1[11][4][3];//先一个方向求B样条曲线，再求另一个方向
    for (int j = 0; j < 4; j++) {
        GLint i = 0;
        for (double t = 0.0; t <= 1.0; t += 0.1)
        {
            double a1 = (pow((1 - t), 3)) / 6;
            double a2 = (3 * pow(t, 3) - 6 * pow(t, 2) + 4) / 6;
            double a3 = (3 * (pow(t, 2) + t - pow(t, 3)) + 1) / 6;
            double a4 = (pow(t, 3)) / 6;//三次B样条的系数
            calculate1[i][j][0] = a1 * UniformPoints[0][j][0] + a2 * UniformPoints[1][j][0] + a3
* UniformPoints[2][j][0] + a4 * UniformPoints[3][j][0];
            calculate1[i][j][1] = a1 * UniformPoints[0][j][1] + a2 * UniformPoints[1][j][1] + a3
* UniformPoints[2][j][1] + a4 * UniformPoints[3][j][1];
            calculate1[i][j][2] = a1 * UniformPoints[0][j][2] + a2 * UniformPoints[1][j][2] + a3
* UniformPoints[2][j][2] + a4 * UniformPoints[3][j][2];
            i = i + 1;
        }
    }


    GLfloat calculate2[11][11][3];
    for (int j = 0; j < 11; j++) {
        GLint i = 0;
        for (double t = 0.0; t <= 1.0; t += 0.1)
        {
            double a1 = (pow((1 - t), 3)) / 6;
            double a2 = (3 * pow(t, 3) - 6 * pow(t, 2) + 4) / 6;
            double a3 = (3 * (pow(t, 2) + t - pow(t, 3)) + 1) / 6;
            double a4 = (pow(t, 3)) / 6;
            calculate2[j][i][0] = a1 * calculate1[j][0][0] + a2 * calculate1[j][1][0] + a3 *
calculate1[j][2][0] + a4 * calculate1[j][3][0];
            calculate2[j][i][1] = a1 * calculate1[j][0][1] + a2 * calculate1[j][1][1] + a3 *
calculate1[j][2][1] + a4 * calculate1[j][3][1];
            calculate2[j][i][2] = a1 * calculate1[j][0][2] + a2 * calculate1[j][1][2] + a3 *
calculate1[j][2][2] + a4 * calculate1[j][3][2];
            i = i + 1;
        }
    }
```

```
    glColor3f(1.0, 1.0, 1.0);
    for (int i = 0; i < 11; i++) {
        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 11; j++)
            glVertex3fv(&calculate2[i][j][0]);
        glEnd();

        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 11; j++)
            glVertex3fv(&calculate2[j][i][0]);
        glEnd();
    }


    glPointSize(5.0);
    glColor3f(1.0, 0.0, 0.0);

    for (int i = 0; i < 4; i++) {
        glBegin(GL_POINTS);
        for (int j = 0; j < 4; j++)
            glVertex3fv(&UniformPoints[i][j][0]);
        glEnd();
    }//绘制点


    glColor3f(1.0, 1.0, 0.0);
    for (int i = 0; i < 4; i++) {
        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 4; j++)
            glVertex3fv(&UniformPoints[i][j][0]);
        glEnd();
        glBegin(GL_LINE_STRIP);
        for (int j = 0; j < 4; j++)
            glVertex3fv(&UniformPoints[j][i][0]);
        glEnd();
    }//连线

    glFlush();
}


int main(int argc, char** argv)
```

```
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(400, 400);

    int i;
    printf("\n Bezier(1) or B Spline(2)\n");
    scanf_s("%d", &i);
    switch (i)
    {
    case 1:
        glutCreateWindow("Asgmt2_11619370216_Bezier_Surface");
        glutDisplayFunc(Bezierrr);
        break;
    case 2:
        glutCreateWindow("Asgmt2_11619370216_B_Spline_Surface");
        glutDisplayFunc(B_Spline);
        break;
    default:
        printf("\n Error!!\n");
    }

    glutMainLoop();
    return 0;
}
```