

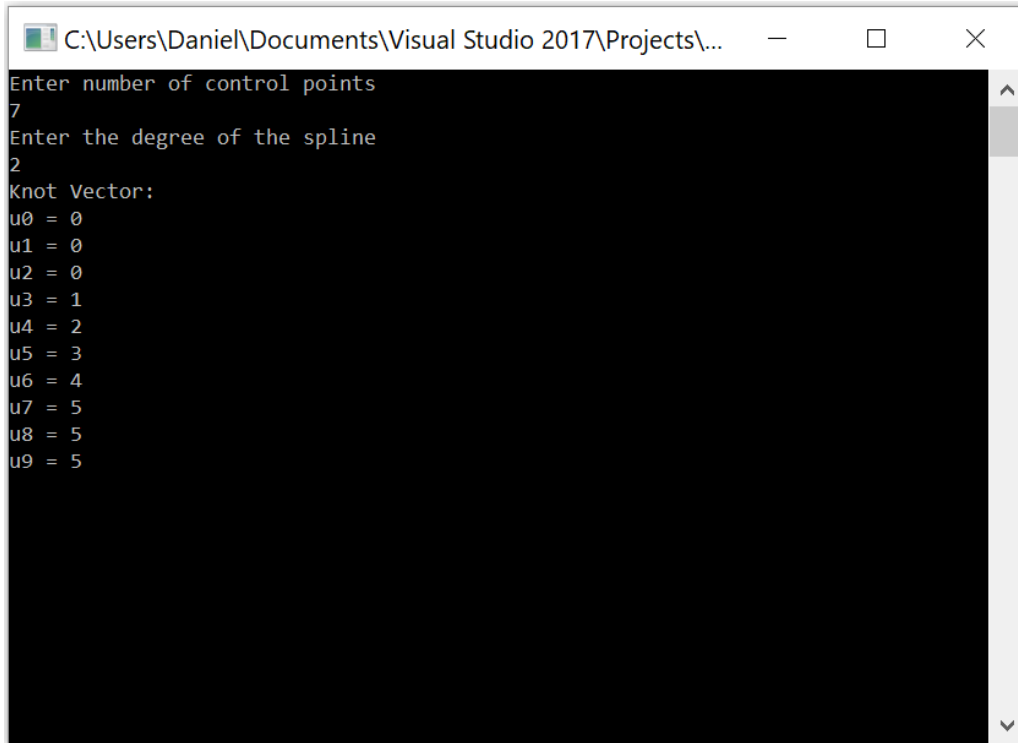
Daniel Meyer

CSE 520

Tong Yu

Lab 6 Report

Part 1 (success):



```
C:\Users\Daniel\Documents\Visual Studio 2017\Projects\...
Enter number of control points
7
Enter the degree of the spline
2
Knot Vector:
u0 = 0
u1 = 0
u2 = 0
u3 = 1
u4 = 2
u5 = 3
u6 = 4
u7 = 5
u8 = 5
u9 = 5
```

Template.cpp

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

using namespace std;

const int screenHeight = 800;
const int screenWidth = 800;

/*
 * Build standard knot vector for n control points
 * and B-splines of order m
 */
void buildKnots(int m, int n, float knot[])
{
```

```

    if (n < m) return;           //not enough control points
    for (int i = 0; i < n + m; ++i) {
        if (i < m) knot[i] = 0.0;
        else if (i < n) knot[i] = i - m + 1;           //i is at least
m here
        else knot[i] = n - m + 1;
    }
}

void init(void)
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glShadeModel(GL_FLAT);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    int numControlPoints = 0;
    int splineDegree = 0;

    cout << "Enter number of control points" << endl;
    cin >> numControlPoints;

    cout << "Enter the degree of the spline" << endl;
    cin >> splineDegree;

    splineDegree += 1; //degree + 1 = order

    if (splineDegree > numControlPoints)
    {
        cout << "Invalid order" << endl;
    }
    else
    {
        float *knot = new float[numControlPoints + splineDegree];
        buildKnots(splineDegree, numControlPoints, knot);

        cout << "Knot Vector: " << endl;
        for (int i = 0; i < numControlPoints + splineDegree; i++)
        {
            cout << "u" << i << " = " << knot[i] << endl;
        }
    }
}

```

```

        glFlush();
    }

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 800, 0, 800);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

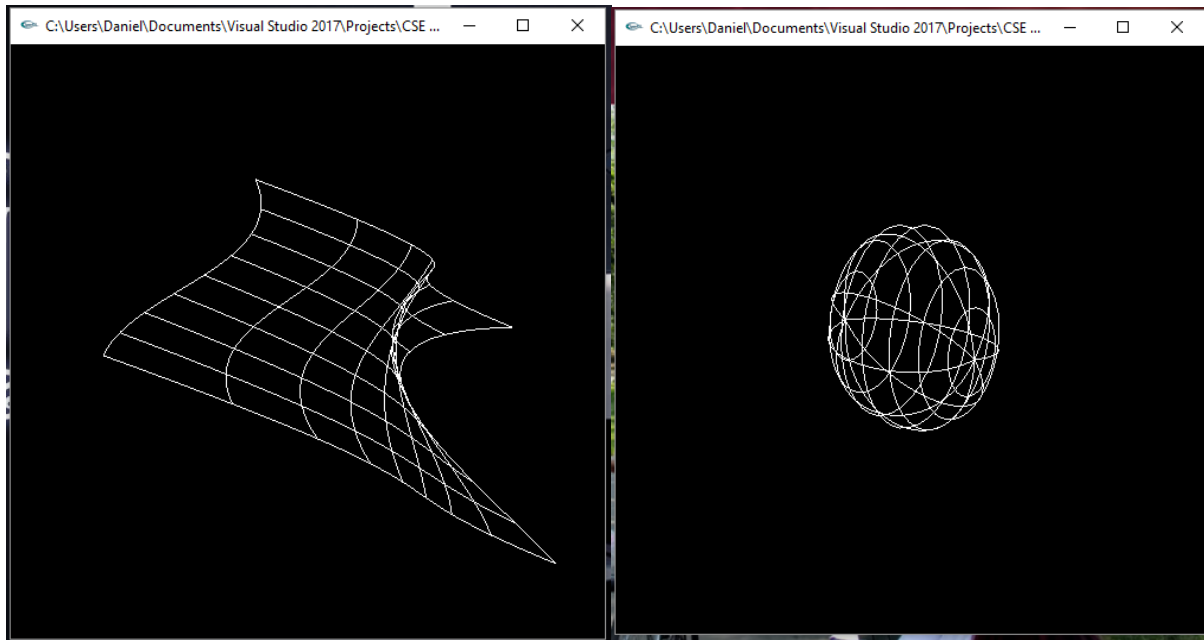
void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
        case 27:
            exit(0);
            break;
    }
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(screenWidth, screenHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);

    glutMainLoop();
    return 0;
}

```

Part 2/3 (success):



Bezsurf.cpp

```
/* bezsurf.c
 * This program renders a wireframe Bezier surface,
 * using two-dimensional evaluators.
 */
```

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

```
/*
GLfloat ctrlpoints[4][4][3] = {
    { {-1.0, 0.0, 1.5}, //bottom right
      {-1.5, 0.0, 0.5},
      {-1.5, 0.0, -0.5},
      {-1.0, 0.0, -1.5}}, //upper right

    { {-0.5, 0.0, 2.5},
      {-0.5, 1.3, 0.5},
      {-0.5, 1.3, -0.5},
      {-0.5, 0.0, -2.5}},

    { {0.5, 0.0, 2.5},
      {0.5, 1.3, 0.5},
      {0.5, 1.3, -0.5},
      {0.5, 0.0, -2.5}},
```

```

        {0.5, 0.0, -2.5}},

    { {1.0, 0.0, 1.5}, //bottom left
      {1.5, 0.0, 0.5},
      {1.5, 0.0, -0.5},
      {1.0, 0.0, -1.5}} //upper left
};
*/

GLfloat ctrlpoints[8][4][3] = {
    { {0.0, 0.0, 1.0}, //bottom right
      {2.0, 0.0, 1.0},
      {2.0, 0.0, -1.0},
      {0.0, 0.0, -1.0}}, //upper right

    { {0.0, 0.0, 1.0},
      {2.0, 4.0, 1.0},
      {2.0, 4.0, -1.0},
      {0.0, 0.0, -1.0}},

    { {0.0, 0.0, 1.0},
      {-2.0, 4.0, 1.0},
      {-2.0, 4.0, -1.0},
      {0.0, 0.0, -1.0}},

    { {0.0, 0.0, 1.0}, //bottom left
      {-2.0, 0.0, 1.0},
      {-2.0, 0.0, -1.0},
      {0.0, 0.0, -1.0}}, //upper left

    { {0.0, 0.0, 1.0}, //bottom right
      {-2.0, 0.0, 1.0},
      {-2.0, 0.0, -1.0},
      {0.0, 0.0, -1.0} }, //upper right

    { {0.0, 0.0, 1.0},
      {-2.0, -4.0, 1.0},
      {-2.0, -4.0, -1.0},
      {0.0, 0.0, -1.0} },

    { {0.0, 0.0, 1.0},
      {2.0, -4.0, 1.0},
      {2.0, -4.0, -1.0},
      {0.0, 0.0, -1.0} },

    { {0.0, 0.0, 1.0}, //bottom left

```

```

    {2.0, 0.0, 1.0},
    {2.0, 0.0, -1.0},
    {0.0, 0.0, -1.0} }, //upper left
};

void display(void)
{
    int i, j;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glPushMatrix();
    glRotatef(85.0, 1.0, 1.0, 1.0);
    for (j = 0; j <= 8; j++) {
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 30; i++)
            glEvalCoord2f((GLfloat)i / 30.0, (GLfloat)j / 8.0);
        glEnd();
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 30; i++)
            glEvalCoord2f((GLfloat)j / 8.0, (GLfloat)i / 30.0);
        glEnd();
    }
    //glEvalMesh2( GL_LINE, 0, 20, 0, 20 );
    glPopMatrix();
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    /*
    GL_MAP1_VERTEX_3 -- specifies that 3-dimensional control points
are
        provided and 3-D vertices should be produced
    0   -- min u
    1   -- max u
    3   -- ustride, number of values to advance in the data between
two
        consecutive control points ( see diagram below )
    4   -- order of u ( = degree + 1 )
    0   -- min v
    1   -- max v
    12  -- vstride ( see diagram below )
    4   -- order of v ( = degree + 1 )
    */

```

```

glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
        0, 1, 12, 8, &ctrlpoints[0][0][0]);
glEnable(GL_MAP2_VERTEX_3);
/*
    glMapGrid2f() -- defines a grid going from u1 to u2; v1 to v2 in
                    evenly-spaced steps
    20 -- number of u steps
    0.0 -- u1 ( starting u )
    1.0 -- u2 ( ending u )
    20 -- number of v steps
    0.0 -- v1 ( starting v )
    1.0 -- v2 ( ending v )

    use with glEvalMesh2()
*/
//glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
glEnable(GL_DEPTH_TEST);
glShadeModel(GL_FLAT);
}

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-4.0, 4.0, -4.0*(GLfloat)h / (GLfloat)w,
                4.0*(GLfloat)h / (GLfloat)w, -4.0, 4.0);
    else
        glOrtho(-4.0*(GLfloat)w / (GLfloat)h,
                4.0*(GLfloat)w / (GLfloat)h, -4.0, 4.0, -4.0, 4.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
        case 27:
            exit(0);
            break;
    }
}

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

```

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);
glutCreateWindow(argv[0]);
init();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMainLoop();
return 0;
}
```

Summary:

For this assignment we are focusing on Bezier Splines and knot vectors. For the first part of the assignment I reused the code that the professor provided us in the lecture notes which I had implemented for the 1st part of our homework assignment. The code worked, and I double checked the knot vector result by working out the math on a spare sheet of paper. The next part of the assignment was to run the bezsurf.c program provided in the lecture notes and then modify it to show a sphere. I had done this by modifying the control points to take in 32 points instead of the normal 16 and then plotted out all of the points. Overall, the program compiled and ran successfully and thus believe I have earned the full 20 points for the assignment.