Daniel Meyer

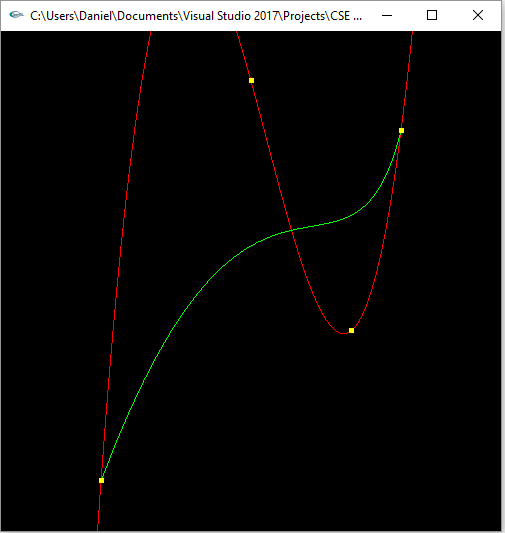
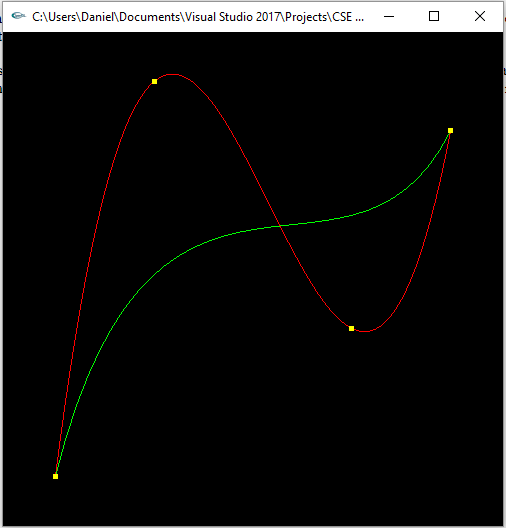
CSE 520

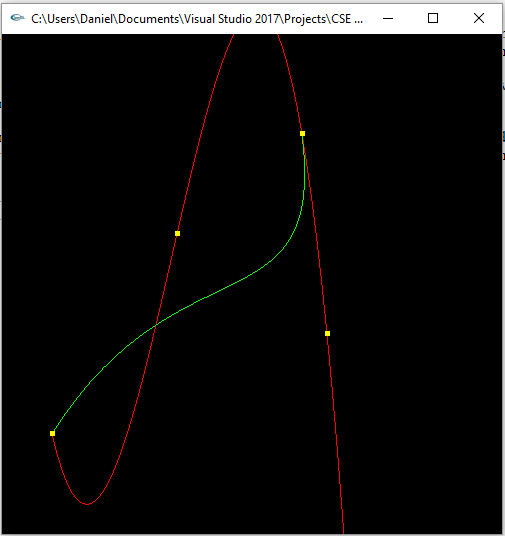
Tong Yu

Curves and Surface Rendering

**Lab 5 Report**

**Part 1 (success):**



*Poly Int = red; Bezier = green*

**polyVbezcurve.cpp**

/\* polyVbezcurve.cpp

\* \* This program demonstrates using polynomial interpretation to draw a curve

\* \* using Lagrange's method.

\* \*/

#include <stdlib.h>

#include <stdio.h>

#include <GL/glut.h>

using namespace std;

GLfloat ctrlpoints0[4][3] = {

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

{2.0, -1.0, 0.0}, {4.0, 3.0, 0.0} };

GLfloat ctrlpoints1[4][3] = {

{ -3.0, -4.0, 0.0}, { 0.0, 4.0, 0.0},

{2.0, -1.0, 0.0}, {3.0, 3.0, 0.0} };

GLfloat ctrlpoints2[4][3] = {

{ -4.0, -3.0, 0.0}, { -1.5, 1.0, 0.0},

{1.5, -1.0, 0.0}, {1.0, 3.0, 0.0} };

// {2.0, 3.0, 0.0}, {4.0, 4.0, 0.0}};

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_FLAT);

/\*

\* GL\_MAP1\_VERTEX\_3 -- specifies that 3-dimensional control points are

\* provided and 3-D vertices should be produced

\* 0.0 -- low value of parmeter u

\* 1.0 -- high value of parmeter u

\* 3 -- number of floating-point values to advance in the data between two

\* consecutive control points

\* 4 -- order of the spline ( = degree + 1 )

\* \*/

glMap1f(GL\_MAP1\_VERTEX\_3, 0.0, 1.0, 3, 4, &ctrlpoints2[0][0]);

glEnable(GL\_MAP1\_VERTEX\_3);

}

//polynomial interpretation for N points

float polyint(float points[][3], float x, int N)

{

float y;

float num = 1.0, den = 1.0;

float sum = 0.0;

for (int i = 0; i < N; ++i) {

num = den = 1.0;

for (int j = 0; j < N; ++j) {

if (j == i) continue;

num = num \* (x - points[j][0]); //x - xj

}

for (int j = 0; j < N; ++j) {

if (j == i) continue;

den = den \* (points[i][0] - points[j][0]); //xi - xj

}

sum += num / den \* points[i][1];

}

y = sum;

return y;

}

void display(void)

{

int i;

float x, y;

glClear(GL\_COLOR\_BUFFER\_BIT);

//Polynomial Interpolation = Red

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINE\_STRIP);

for (i = -40; i <= 40; i++) {

x = (float)i / 10.0;

y = polyint(ctrlpoints2, x, 4);

glVertex2f(x, y);

}

glEnd();

//Bezier Curve = Green

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_LINE\_STRIP);

for (i = 0; i <= 30; i++)

glEvalCoord1f((GLfloat)i / 30.0);

glEnd();

/\* The following code displays the control points as dots. \*/

glPointSize(5.0);

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_POINTS);

for (i = 0; i < 4; i++)

glVertex3fv(&ctrlpoints2[i][0]);

glEnd();

glFlush();

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-5.0, 5.0, -5.0\*(GLfloat)h / (GLfloat)w,

5.0\*(GLfloat)h / (GLfloat)w, -5.0, 5.0);

else

glOrtho(-5.0\*(GLfloat)w / (GLfloat)h,

5.0\*(GLfloat)w / (GLfloat)h, -5.0, 5.0, -5.0, 5.0);

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);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow(argv[0]);

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

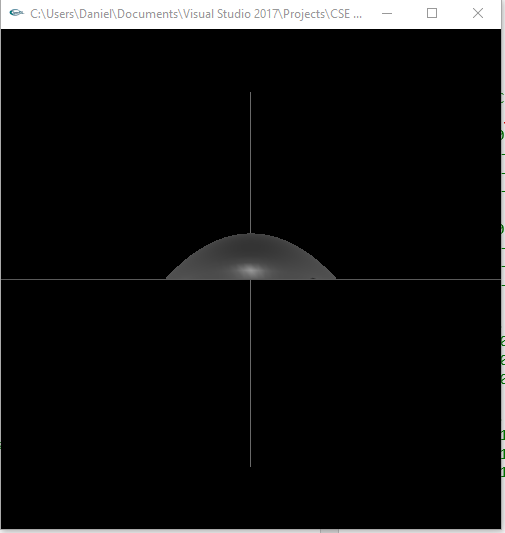
glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**Part 2 (success):**



**Bezmesh.cpp**

/\* bezmesh.c

\* This program renders a lighted, filled 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}},

{ {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

};

int rx = 0;

int ry = 0;

void initlights(void)

{

GLfloat ambient[] = { 0.2, 0.2, 0.2, 1.0 };

GLfloat position[] = { 0.0, 0.0, 2.0, 1.0 };

GLfloat mat\_diffuse[] = { 0.6, 0.6, 0.6, 1.0 };

GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat mat\_shininess[] = { 150.0 };

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambient);

glLightfv(GL\_LIGHT0, GL\_POSITION, position);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

//glRotatef(0.0, 1.0, 1.0, 1.0);

//glRotatef(45.0, 1.0, 1.0, 1.0);

//glRotatef(90.0, 1.0, 1.0, 1.0);

glEvalMesh2(GL\_FILL, 0, 20, 0, 20);

glPopMatrix();

//glRotatef(85.0, 1.0, 1.0, 1.0);

glColor3f(1, 0, 0);

glBegin(GL\_LINES);

glVertex3f(-4, 0, 0);

glVertex3f(4, 0, 0);

glEnd();

glColor3f(0, 1, 0);

glBegin(GL\_LINES);

glVertex3f(0, -3, 0);

glVertex3f(0, 3, 0);

glEnd();

/\*

glColor3f(0, 1, 0);

glBegin(GL\_LINES);

glVertex3f(0, 0, -4);

glVertex3f(0, 0, 4);

glEnd();

\*/

glFlush();

}

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glEnable(GL\_DEPTH\_TEST);

glMap2f(GL\_MAP2\_VERTEX\_3, 0, 1, 3, 4,

0, 1, 12, 4, &ctrlpoints[0][0][0]);

glEnable(GL\_MAP2\_VERTEX\_3);

glEnable(GL\_AUTO\_NORMAL);

glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);

initlights(); /\* for lighted version only \*/

}

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();

glutReshapeFunc(reshape);

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**Summary:**

For this assignment we had two tasks using interpolation to create various curves and a surface. The first was to implement a program that showed both a Bezier and polynomial interpolation curve using the same control points to compare the difference in results. Both had to use 4 control points and meaning the Bezier curve was degree 3. I performed this on 3 different sets of control points and found the Bezier curve was smaller and more direct whereas the polynomial interpolation was far more extreme, hitting each control point. The next part of the assignment was to create the upper-half of a sphere using 16 points and Bezier curves. Overall the program compiled and ran successfully and believe I have earned the full 20 points.