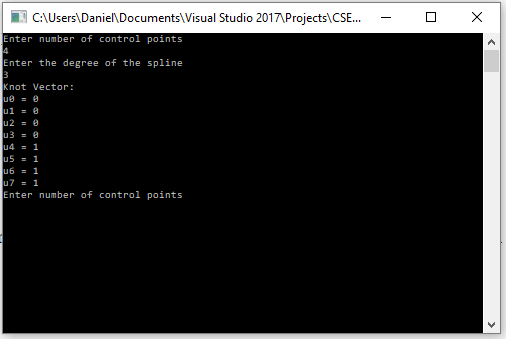
Daniel Meyer

CSE 520

Tong Yu

**Homework 2 Report**

**Part 1 (success):**



#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(0.0, 0.0, 0.0, 1.0);

glShadeModel(GL\_FLAT);

glEnable(GL\_MAP1\_VERTEX\_3);

}

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();

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-6.0, 6.0, -6.0\*(GLfloat)h / (GLfloat)w,

6.0\*(GLfloat)h / (GLfloat)w, -6.0, 6.0);

else

glOrtho(-6.0\*(GLfloat)w / (GLfloat)h,

6.0\*(GLfloat)w / (GLfloat)h, -6.0, 6.0, -6.0, 6.0);

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(screenWidth, screenHeight);

glutInitWindowPosition(100, 100);

glutCreateWindow(argv[0]);

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

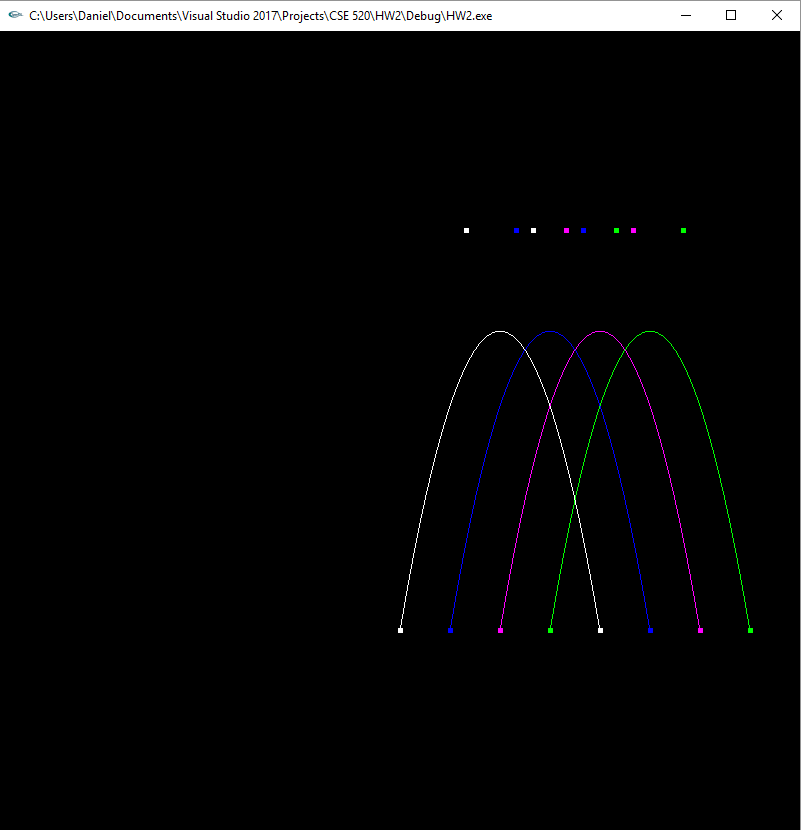
glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**Part 2 (success):**



#include <stdlib.h>

#include <iostream>

#include <GL/glut.h>

using namespace std;

const int screenHeight = 800;

const int screenWidth = 800;

GLfloat ctrlpoints1[4][3] = {

{-3, -3, 0.0}, {-2, 3.0, 0.0},

{-1, 3.0, 0.0}, {0, -3.0, 0.0}

};

GLfloat ctrlpoints0[4][3] = {

{0, 0, 0.0}, {1, 0.5, 0.0},

{2.0, 0.5, 0.0}, {3.0, 0.0, 0.0}

};

/\*

\* 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(0.0, 0.0, 0.0, 1.0);

glShadeModel(GL\_FLAT);

glEnable(GL\_MAP1\_VERTEX\_3);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0);

const int n = 4, m = 4;

float time[n + m - 1] = { 0, 1, 2, 3, 4, 5, 6 };

float blendFunc[4];

float t = time[0];

t = 0;

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

{

//t = time[i];

// the t value inverted

float it = 1.0f - t;

// calculate blending functions

float b0 = it \* it\*it;

float b1 = 3 \* t\*it\*it;

float b2 = 3 \* t\*t\*it;

float b3 = t \* t\*t;

blendFunc[i] = b0 \* ctrlpoints0[0][0] + b1 \* ctrlpoints0[1][0] + b2 \* ctrlpoints0[2][0] + b3 \* ctrlpoints0[3][0];

t += 0.25;

}

ctrlpoints1[0][0] = blendFunc[0] + ctrlpoints0[0][0];

ctrlpoints1[1][0] = blendFunc[0] + ctrlpoints0[1][0];

ctrlpoints1[2][0] = blendFunc[0] + ctrlpoints0[2][0];

ctrlpoints1[3][0] = blendFunc[0] + ctrlpoints0[3][0];

glMap1f(GL\_MAP1\_VERTEX\_3, 0.0, 1.0, 3, 4, &ctrlpoints1[0][0]); //t0 -> t3

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINE\_STRIP);

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

glEvalCoord1f((GLfloat)i / 30.0);

glEnd();

glPointSize(5.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POINTS);

for (int i = 0; i < n; i++)

glVertex3fv(&ctrlpoints1[i][0]);

glEnd();

ctrlpoints1[0][0] = blendFunc[1] + ctrlpoints0[0][0];

ctrlpoints1[1][0] = blendFunc[1] + ctrlpoints0[1][0];

ctrlpoints1[2][0] = blendFunc[1] + ctrlpoints0[2][0];

ctrlpoints1[3][0] = blendFunc[1] + ctrlpoints0[3][0];

glMap1f(GL\_MAP1\_VERTEX\_3, 0.0, 1.0, 3, 4, &ctrlpoints1[0][0]); //t1 -> t4

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_STRIP);

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

glEvalCoord1f((GLfloat)i / 30.0);

glEnd();

glPointSize(5.0);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_POINTS);

for (int i = 0; i < n; i++)

glVertex3fv(&ctrlpoints1[i][0]);

glEnd();

ctrlpoints1[0][0] = blendFunc[2] + ctrlpoints0[0][0];

ctrlpoints1[1][0] = blendFunc[2] + ctrlpoints0[1][0];

ctrlpoints1[2][0] = blendFunc[2] + ctrlpoints0[2][0];

ctrlpoints1[3][0] = blendFunc[2] + ctrlpoints0[3][0];

glMap1f(GL\_MAP1\_VERTEX\_3, 0.0, 1.0, 3, 4, &ctrlpoints1[0][0]); //t2 -> t5

glColor3f(1.0, 0.0, 1.0);

glBegin(GL\_LINE\_STRIP);

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

glEvalCoord1f((GLfloat)i / 30.0);

glEnd();

glPointSize(5.0);

glColor3f(1.0, 0.0, 1.0);

glBegin(GL\_POINTS);

for (int i = 0; i < n; i++)

glVertex3fv(&ctrlpoints1[i][0]);

glEnd();

ctrlpoints1[0][0] = blendFunc[3] + ctrlpoints0[0][0];

ctrlpoints1[1][0] = blendFunc[3] + ctrlpoints0[1][0];

ctrlpoints1[2][0] = blendFunc[3] + ctrlpoints0[2][0];

ctrlpoints1[3][0] = blendFunc[3] + ctrlpoints0[3][0];

glMap1f(GL\_MAP1\_VERTEX\_3, 0.0, 1.0, 3, 4, &ctrlpoints1[0][0]); //t3 -> t6

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_LINE\_STRIP);

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

glEvalCoord1f((GLfloat)i / 30.0);

glEnd();

glPointSize(5.0);

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_POINTS);

for (int i = 0; i < n; i++)

glVertex3fv(&ctrlpoints1[i][0]);

glEnd();

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();

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-6.0, 6.0, -6.0\*(GLfloat)h / (GLfloat)w,

6.0\*(GLfloat)h / (GLfloat)w, -6.0, 6.0);

else

glOrtho(-6.0\*(GLfloat)w / (GLfloat)h,

6.0\*(GLfloat)w / (GLfloat)h, -6.0, 6.0, -6.0, 6.0);

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(screenWidth, screenHeight);

glutInitWindowPosition(100, 100);

glutCreateWindow(argv[0]);

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**Part 3 (success):**

P(0) = (0,0,0)

P(1/3) = (1,2,2)

P(2/3) = (2,3,4)

P(1) = (4,5,8)

Find P(0.8)

P(u) = B0(u)P0 + B1(u)P1 + B2(u)P2 + B3(u)P3

B0 = 0.008

B1 = 0.096

B2 = 0.384

B3 = 0.512

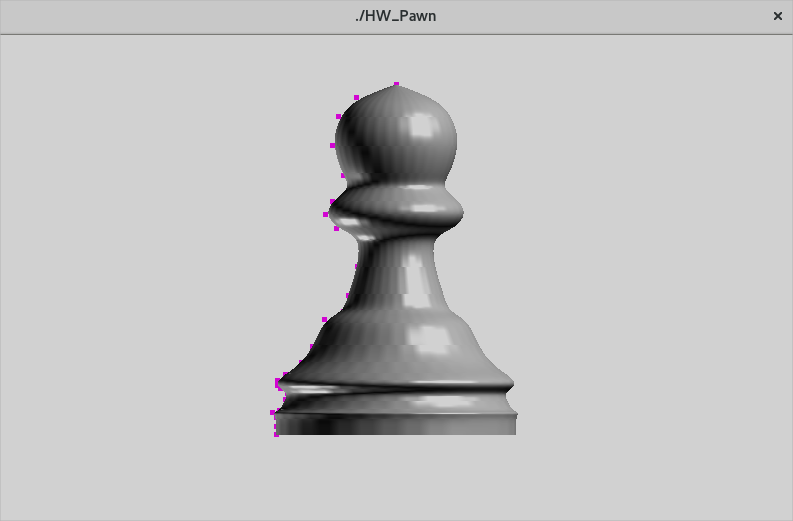
u = 0.8

P(0.8) = (0.008)(0.8)(0,0,0) + (0.096)(0.8)(1,2,2) + (0.384)(0.8)(2,3,4) + (0.512)(0.8)(4,5,8)

= (0,0,0) + (0.0768, 0.1536, 0.1536) + (0.6144, 0.9216, 1.2288) + (1.6384, 2.048, 3.2768)

**P(0.8) = (2.3296, 3.1232, 4.6592)**

**Part 4 (success):**



#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include "lib3ds\_vector.h"

#include <GL/glut.h>

using namespace std;

const double PI = 3.14159265389;

const int Npoints = 27;

const int m\_order = 4;

int anglex = 0, angley = 0, anglez = 0; //rotation angles

int window;

GLuint thePawn;

void pawn(int nx, int ntheta, float startx, float endx);

//control points

GLfloat ctrlpoints[Npoints][3] = {

{4.38,0.00, 0},{4.22,0.50, 0},{3.98,0.72, 0},{3.62,0.80, 0},{3.24,0.66, 0},

{3.10,0.56, 0},{2.92,0.80, 0},{2.76,0.88, 0},{2.58,0.74, 0},{2.48,0.44, 0},

{2.10,0.48, 0},{1.74,0.60, 0},{1.54,0.66, 0},{1.44,0.90, 0},{1.10,1.04, 0},

{0.90,1.18, 0},{0.76,1.38, 0},{0.68,1.48, 0},{0.62,1.48, 0},{0.58,1.44, 0},

{0.52,1.36, 0},{0.44,1.38, 0},{0.30,1.46, 0},{0.28,1.54, 0},{0.22,1.50, 0},

{0.10,1.50, 0},{0.00,1.50, 0}

};

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

glPolygonMode(GL\_FRONT, GL\_FILL);

thePawn = glGenLists(1);

glNewList(thePawn, GL\_COMPILE);

pawn(32, 64, 0, 3.5);

glEndList();

glShadeModel(GL\_SMOOTH);

//lighting

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

GLfloat mat\_shininess[] = { 50.0 };

GLfloat light[] = { 1.0, 1.0, 1.0 };

GLfloat light1[] = { 1.0, 1.0, 1.0 };

GLfloat light\_position[] = { 1.0, 1.0, 1.0, 0.0 };

GLfloat light\_position1[] = { -1.0, -1.0, -1.0, 0.0 };

GLfloat lmodel\_ambient[] = { 0.1, 0.1, 0.1, 1.0 };

glClearColor(1.0, 1.0, 1.0, 0.0);

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

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

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glLightfv(GL\_LIGHT1, GL\_POSITION, light\_position1);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, light1);

glLightfv(GL\_LIGHT1, GL\_SPECULAR, light1);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, lmodel\_ambient);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHT1);

glEnable(GL\_DEPTH\_TEST);

}

/\*

\* 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;

}

}

//evaluate blending functions recurvsively

float bSpline(int k, int m, float u, float knot[])

{

float d1, d2, sum = 0.0;

if (m == 1)

return (knot[k] < u && u <= knot[k + 1]); //1 or 0

//m larger than 1, so recurse

d1 = knot[k + m - 1] - knot[k];

if (d1 != 0)

sum = (u - knot[k]) \* bSpline(k, m - 1, u, knot) / d1;

d2 = knot[k + m] - knot[k + 1];

if (d2 != 0)

sum += (knot[k + m] - u) \* bSpline(k + 1, m - 1, u, knot) / d2;

return sum;

}

//non uniform rational B-splines, n control points, order m, p[] is the output point

void nurb(float control\_points[][3], float u, float knot[], float p[])

{

// sum the control points mulitplied by their respective blending functions

for (int i = 0; i < 3; ++i) { //x, y, z components

p[i] = 0;

for (int k = 0; k < Npoints; ++k)

p[i] += bSpline(k, m\_order, u, knot) \* control\_points[k][i];

}

}

//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;

}

float aLine(float x)

{

return x + 2.5;

}

//cubic B-spline, a special case of NURB

void bspline(float points[][3], float t, float out[])

{

// the t value inverted

float it = 1.0f - t;

// calculate blending functions

float b0 = it \* it\*it;

float b1 = 3 \* t\*it\*it;

float b2 = 3 \* t\*t\*it;

float b3 = t \* t\*t;

// sum the control points mulitplied by their respective blending functions

out[0] = b0 \* points[0][0] + b1 \* points[1][0] + b2 \* points[2][0] + b3 \* points[3][0]; //x

out[1] = b0 \* points[0][1] + b1 \* points[1][1] + b2 \* points[2][1] + b3 \* points[3][1]; //y

out[2] = b0 \* points[0][2] + b1 \* points[1][2] + b2 \* points[2][2] + b3 \* points[3][2]; //z

}

void testing()

{

int n = Npoints, m = m\_order;

float knot[n + m];

buildKnots(m, n, knot);

for (int i = 0; i < n + m; i++)

printf("%4.2f,", knot[i]);

printf("\n");

}

void pawn(int nx, int ntheta, float startx, float endx)

{

const int n = Npoints, m = m\_order; //n control points, degree m NURB

float knot[n + m];

const float dx = (endx - startx) / nx; //x step size

const float dtheta = 2 \* PI / ntheta; //angular step size

float theta = PI / 2.0; //from pi/2 to3pi/2

buildKnots(m, n, knot);

int i, j;

float x, y, z, r; //current coordinates

float x1, y1, z1, r1; //next coordinates

float t, v[3];

float va[3], vb[3], vc[3], normal[3];

int nturn = 0;

x = startx;

nurb(ctrlpoints, 0, knot, v);

x = v[0];

r = v[1];

bool first\_point = true;

for (int k = m - 1; k < n; ++k) { //step through the knots

float dknot = knot[k + 1] - knot[k];

if (dknot == 0) continue;

theta = 0; //PI / 2.0;

int start = 0, nn = 60, end = nn;

// if ( k == n - 1 ) end = nn;

for (i = start; i <= end; i++) {

t = knot[k] + dknot \* (float)i / nn;

nurb(ctrlpoints, t, knot, v);

if (first\_point) {

v[0] = ctrlpoints[0][0];

v[1] = ctrlpoints[0][1];

first\_point = false;

}

x1 = v[0];

r1 = v[1];

//draw the surface composed of quadrilaterals by sweeping theta

glBegin(GL\_QUAD\_STRIP);

for (j = 0; j <= ntheta; ++j) {

theta += dtheta;

double cosa = cos(theta);

double sina = sin(theta);

y = r \* cosa; y1 = r1 \* cosa; //current and next y

z = r \* sina; z1 = r1 \* sina; //current and next z

if (nturn == 0) {

va[0] = x; va[1] = y; va[2] = z;

vb[0] = x1; vb[1] = y1; vb[2] = z1;

nturn++;

}

else {

nturn = 0;

vc[0] = x; vc[1] = y; vc[2] = z;

//vector\_normal(normal, va, vb, vc);

vector\_normal()

glNormal3f(normal[0], normal[1], normal[2]);

}

//edge from point at x to point at next x

glVertex3f(x, y, z);

glVertex3f(x1, y1, z1);

//forms quad with next pair of points with incremented theta value

}

glEnd();

x = x1;

r = r1;

}

} //for k

}

//revolve about y-axis

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 1.0, 1.0);

const float startx = 0, endx = 3.5;

const int nx = 32; //number of slices along x-direction

const int ntheta = 64; //number of angular slices

glPushMatrix();

glRotatef(anglex, 1.0, 0.0, 0.0); //rotate the object about x-axis

glRotatef(angley, 0.0, 1.0, 0.0); //rotate about y-axis

glRotatef(anglez, 0.0, 0.0, 1.0); //rotate about z-axis

glEnable(GL\_LIGHTING);

glCallList(thePawn);

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

glDisable(GL\_LIGHTING);

glPointSize(5.0);

glColor3f(1.0, 0.0, 1.0);

glBegin(GL\_POINTS);

for (int i = 0; i < Npoints; i++)

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

glEnd();

glPopMatrix();

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 'x':

anglex = (anglex + 3) % 360;

break;

case 'X':

anglex = (anglex - 3) % 360;

break;

case 'y':

angley = (angley + 3) % 360;

break;

case 'Y':

angley = (angley - 3) % 360;

break;

case 'z':

anglez = (anglez + 3) % 360;

break;

case 'Z':

anglez = (anglez - 3) % 360;

break;

case 'r': //reset

anglez = angley = anglex = 0;

glLoadIdentity();

break;

case 27: /\* escape \*/

glutDestroyWindow(window);

exit(0);

}

glutPostRedisplay();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 800);

glutInitWindowPosition(100, 100);

window = glutCreateWindow(argv[0]);

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**Part 4 (success):**

#include <stdlib.h>

#include <string.h>

#include <iostream>

#include <vector>

#include <algorithm>

#include "LinearR4.h"

#include <SDL/SDL.h>

#include <GL/glut.h>

using namespace std;

int anglex = 0, angley = 0, anglez = 0; //rotation angles

int window;

float a = 1;

float b = 0.5;

float p = 1;

float q = 7;

//float dC[4] = (-p \* (q + b \* cos(q\*t)) \* sin(p\*t) - b \* q \* sin(q\*t) \* cos(p\*t), p \* (a + b \* cos(q\*t)) \* cos(p\*t) - b \* q \* sin(q\*t) \* sin(p\*t), b \* q \* cos(q\*t), 0);

//float dC[3];

//float ddC[3];

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glEnable(GL\_DEPTH\_TEST);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-3.0, 3.0, -3.0, 3.0, 0.1, 100);

glMatrixMode(GL\_MODELVIEW); // position and aim the camera

glLoadIdentity();

gluLookAt(0, 0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

}

//v1[], v2[] are two vectors

//out[] holds the crossproduct v1 x v2

void crossprod(float v1[3], float v2[3], float out[3])

{

out[0] = v1[1] \* v2[2] - v1[2] \* v2[1];

out[1] = v1[2] \* v2[0] - v1[0] \* v2[2];

out[2] = v1[0] \* v2[1] - v1[1] \* v2[0];

}

//helix curve

void get\_C(float C[4], float t, float b)

{

C[0] = (a + b \* cos(q \* t)) \* cos(p \* t);

C[1] = (a + b \* cos(q \* t)) \* sin(p \* t);

C[2] = b \* sin(q \* t);

C[3] = 1;

}

//Matrix for transforming to Frenet frame

//void setM(LinearMapR4 &M, float t, float b)

void setM(LinearMapR4 &M, float t)

{

float c = 1.0 / sqrt(1 + b \* b);

M.SetColumn1((7 \* sin(t) \* sin(7 \* t)) + cos(t) \* (-25 \* cos(7 \* t) - 1),

(-25.5 \* sin(t) \* cos(7 \* t)) - (10.5 \* sin(7 \* t) \* cos(t)),

-24.5 \* sin(7 \* t),

0); //Normal N(t)

M.SetColumn2((-89.25 \* pow(cos(7\*t),2) \* sin(t)) + (24.5 \* cos(t) \* sin(7\*t)) - (24.5 \* cos(t) \* cos(7\*t) \* sin(7\*t)) - (85.75 \* sin(t) \* pow(sin(7\*t), 2))\*c,

(3.5 \* cos(t) \* cos(7\*t)) + (87.5 \* cos(t) \* pow(cos(7\*t), 2)) + (24.5 \* sin(t) \* sin(7\*t)) - (12.25 \* cos(7\*t) \* sin(t) \* sin(7\*t)) + (85.75 \* cos(t) \* pow(sin(7\*t), 2)) \*c,

(-pow(cos(t), 2)) - (25.5 \* pow(cos(t), 2) \* cos(7\*t)) - (12.5 \* pow(cos(t), 2) \* pow(cos(7\*t), 2)) - (25.5 \* cos(7\*t) \* pow(sin(t), 2)) - (12.75 \* pow(cos(7\*t), 2) \* pow(sin(t), 2)) - (3.5 \* cos(t) \* cos(7\*t) \* sin(t) \* sin(7\*t)) - (36.75 \* pow(cos(t), 2) \* pow(sin(7\*t), 2)) - (24.5 \* pow(sin(t), 2) \* pow(sin(7\*t), 2)),

0); //Binormal B(t)

M.SetColumn3(sin(t) \* (-0.5 \* Cos(7 \* t) - 1) - (3.5 \* sin(7 \* t) \* cos(t)) \* c,

(-3.5 \* sin(t) \* sin(7 \* t)) + (0.5 \* cos(7 \*t) \* cos(t)) + cos(t) \* c,

(3.5 \* cos(7 \* t)) \* c,

0); //Tangent T(t)

M.SetColumn4((1 + 0.5 \* cos(7 \* t)) \* cos(t),

(1 + 0.5 \* cos(7 \* t)) \* sin(t),

0.5 \* sin(7 \* t),

1); //The curve C(t)

float tangent[3];

tangent[0] = (-p \* (q + b \* cos(q\*t)) \* sin(p\*t) - b \* q \* sin(q\*t) \* cos(p\*t); //dx

tangent[1] = p \* (a + b \* cos(q\*t)) \* cos(p\*t) - b \* q \* sin(q\*t) \* sin(p\*t); //dy

tangent[2] = b \* q \* cos(q\*t); //dz

float deriveTan[3];

deriveTan[0] = (-p \* p \* (a + b \* cos(q\*t)) \* cos(p\*t) - b \* q \* sin(q\*t) \* sin(p\*t)) + b \* q \* (p \* sin(q\*t) \* sin(p\*t) - q \* cos(q\*t) \* cos(p\*t)); //ddx

deriveTan[1] = (p \* (-p \* (q + b \* cos(q\*t)) \* sin(p\*t) - b \* q \* sin(q\*t) \* cos(p\*t)) - b \* q \* (p \* sin(q\*t) \* cos(p\*t) + q \* cos(q\*t) \* sin(p\*t)); //ddy

deriveTan[2] = (-q \* -q) \* b \* sin(q\*t); //ddz

float binormal[3];

crossprod(tangent, deriveTan, binormal);

float normal[3];

crossprod(binormal, tangent, normal);

M.SetColumn1(normal[0], normal[1], normal[2], 0); //Normal N(t)

M.SetColumn2(binormal[0], binormal[1], binormal[2], 0); //Binormal B(t)

M.SetColumn3(-p \* (q + b \* cos(q\*t)) \* sin(p\*t) - b \* q \* sin(q\*t) \* cos(p\*t), p \* (a + b \* cos(q\*t)) \* cos(p\*t) - b \* q \* sin(q\*t) \* sin(p\*t), b \* q \* cos(q\*t), 0); //Tangent T(t)

M.SetColumn4((a + b \* cos(q \* t)) \* cos(p \* t), (a + b \* cos(q \* t)) \* sin(p \* t), b \* sin(q \* t), 1); //The curve C(t)

}

void print\_M(LinearMapR4 &M)

{

cout << "(" << M.m11 << ",\t" << M.m12 << ",\t" << M.m13 << ",\t" << M.m14 << ")" << endl;

cout << "(" << M.m21 << ",\t" << M.m22 << ",\t" << M.m23 << ",\t" << M.m24 << ")" << endl;

cout << "(" << M.m31 << ",\t" << M.m32 << ",\t" << M.m33 << ",\t" << M.m34 << ")" << endl;

cout << "(" << M.m41 << ",\t" << M.m42 << ",\t" << M.m43 << ",\t" << M.m44 << ")" << endl;

}

class Cfloat3 { //Note: array is copyable; e.g. int a[8],b[8]; "a = b;" won't work

public:

float p3[3];

};

void display(void)

{

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

const float b = 0.1; //constant of Helix curve

double H = 6.0;

LinearMapR4 M; //Transformation matrix

const int N = 4; //number of vertices in base

vector<Cfloat3>vp0(N), vp1(N);

VectorR4 p\_1; //transformed point

//4 vertices of a quad

//float p[4][3]= { {-0.2,-0.2,0}, {0.2,-0.2,0}, {0.2,0.2, 0},{-0.2,0.2,0} };

//homogeneous coordinates of the four vertices of a quad

VectorR4 points[4]; //define four points

points[0] = VectorR4(-0.1, -0.1, 0, 1); //x, y, z, w

points[1] = VectorR4(0.1, -0.1, 0, 1); //x, y, z, w

points[2] = VectorR4(0.1, 0.1, 0, 1); //x, y, z, w

points[3] = VectorR4(-0.1, 0.1, 0, 1); //x, y, z, w

glColor3f(0.1, 1.0, 0);

glPushMatrix();

glRotatef(anglex, 1.0, 0.0, 0.0); //rotate the object about x-axis

glRotatef(angley, 0.0, 1.0, 0.0); //rotate about y-axis

glRotatef(anglez, 0.0, 0.0, 1.0); //rotate about z-axis

float C[4];

glLineWidth(3);

glPolygonMode(GL\_FRONT, GL\_LINE);

glPolygonMode(GL\_BACK, GL\_LINE);

glPolygonMode(GL\_FRONT, GL\_FILL);

glPolygonMode(GL\_BACK, GL\_FILL);

//The curve

glBegin(GL\_LINE\_STRIP);

for (float t = 0; t <= 26; t += 0.2) {

get\_C(C, t, b);

glVertex4fv(C);

}

glColor3f(1.0, 0.1, 0);

glEnd();

float p3[3]; //3-D point, (x, y, z)

//starting

//setM(M, 0, b); //t = 0

setM(M, 0);

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

p\_1 = M \* points[i]; //transform the point

p\_1.Dump(vp0[i].p3); //put (x, y, z) in vp0[i].p3[]

}

glBegin(GL\_QUADS); //a side has four points

for (float t = 0.2; t <= 26; t += 0.2) {

//setM(M, t, b);

setM(M, t);

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

p\_1 = M \* points[i]; //transform the point

p\_1.Dump(vp1[i].p3); //put (x, y, z) in vp1[i].p3[]

}

for (int i = 0; i < N; ++i) { //draw the N sides of tube between 'base' and 'cap'

int j = (i + 1) % N;

glVertex3fv(vp0[i].p3);

glVertex3fv(vp0[j].p3);

glVertex3fv(vp1[j].p3);

glVertex3fv(vp1[i].p3);

}

copy(vp1.begin(), vp1.end(), vp0.begin()); //copy vp1 to vp0

} //for t

glEnd();

glPopMatrix();

glFlush();

}

void keyboard(unsigned char key, int x, int y)

{

switch (key) {

case 27:

glutDestroyWindow(window);

exit(0);

case 'x':

anglex = (anglex + 3) % 360;

break;

case 'X':

anglex = (anglex - 3) % 360;

break;

case 'y':

angley = (angley + 3) % 360;

break;

case 'Y':

angley = (angley - 3) % 360;

break;

case 'z':

anglez = (anglez + 3) % 360;

break;

case 'Z':

anglez = (anglez - 3) % 360;

break;

case 'r': //reset

anglez = angley = anglex = 0;

break;

}

glutPostRedisplay();

}

int main(int argc, char \*argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

window = glutCreateWindow("Mesh ");

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glClearColor(1.0f, 1.0f, 1.0f, 0.0f); //white background

glViewport(0, 0, 500, 500);

init();

glutMainLoop();

return 0;

}

**Summary:**

For this assignment we had multiple tasks. The first task was to write a program to print out the standard knot vector user provided number of control points and degree of the spline. It then prints out the knot vector to the screen. The next task was to make a program that plots the blending functions for degree 3. Next was to find the point at u=0.8 using the given four control points. The fourth assignment was to use a B-Spline generated by a set of given control points and then use surface of revolution to generate a chess piece. Finally, the last part was to make tube using the Frenet frame of a toroidal spiral with the given x, y, z functions. Overall each program compiled and ran successfully per the requirements of each task, however I am missing the screenshot for the last part of the assignment due to being unable to get to the computer lab to grab a screenshot of the program before the due date and I believe I earned 65 points for the assignment.