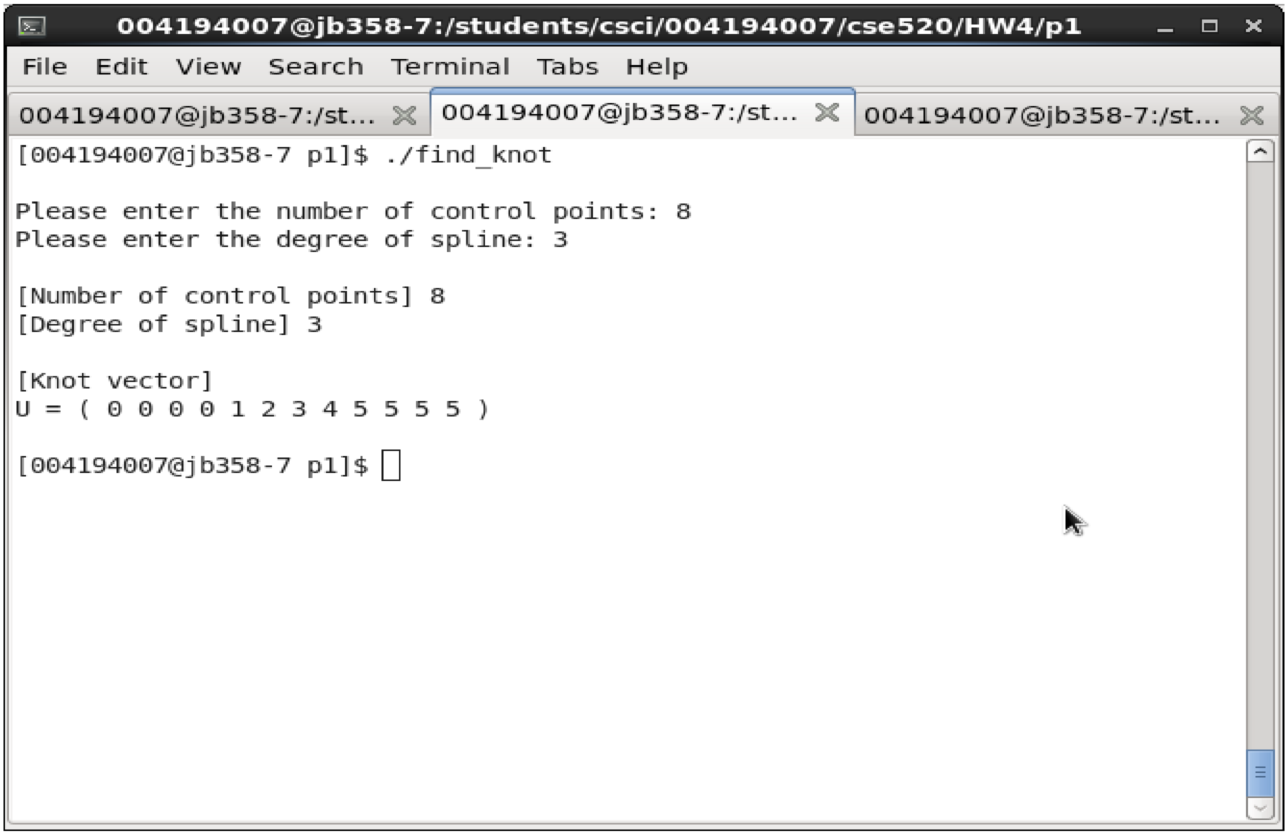
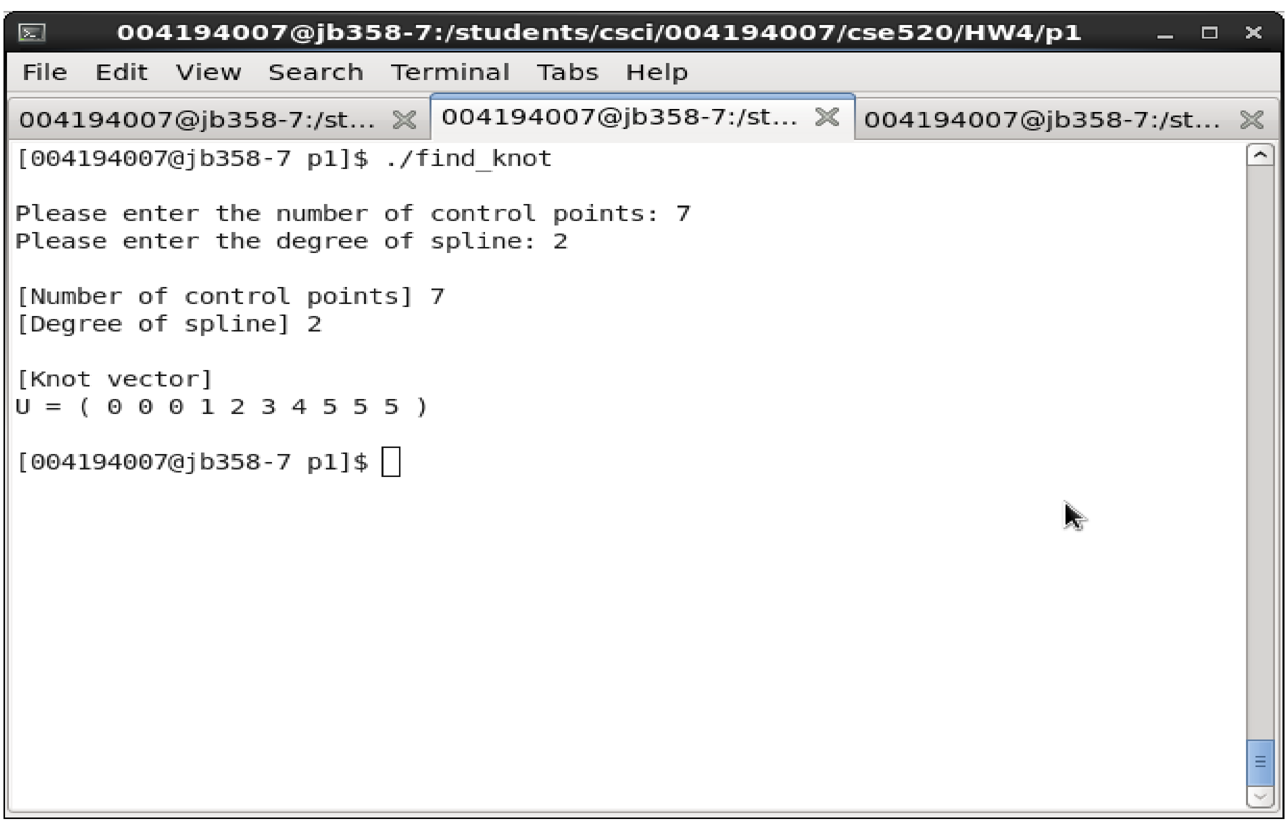
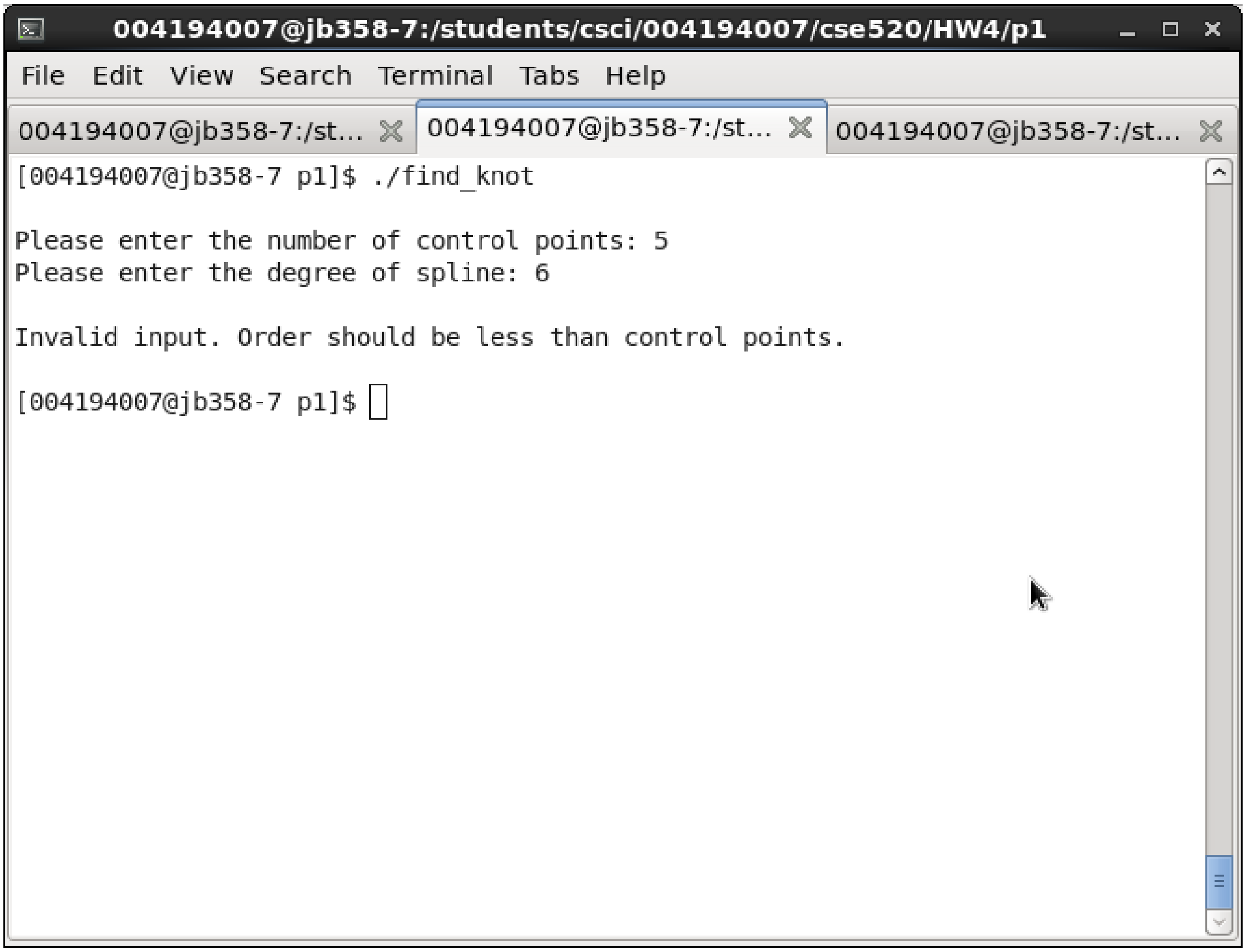
Yazhuo Liu
Homework 4

Write a program that finds the knot vector (u_0, ..., u_{n-1}) of a B-spline. It asks for 'number of control points' and 'degree of spline' as inputs and prints out the knot vector.







Code:

```
/*****************************************************************************

        knot.cpp
           This program finds the knot vector ( u0, u1, ..., un-1 )
        of a B-spline. The program finds the knot vecotr by using the
        steps provided in our lecture notes, then store each knot in
        a vector.

*****************************************************************************/

#include <iostream>
#include <vector>

using namespace std;

int main()
{
   int cpoints, degree, i;
```

```
    cout << endl << "Please enter the number of control points: ";
    cin >> cpoints;
    cout << "Please enter the degree of spline: ";
    cin >> degree;

    int order = degree + 1;              //order is 1 less than degree
    int value = 1;
    int size = cpoints + order;          //number of knots = control points + order

    if (order > cpoints) {               //knot vector only exists when order <= control points
        cout << endl
        << "Invalid input. Order should be less than control points. \n" << endl;
        return 0;
    }

    vector<int> kv;                      //store each knot value

    //First m knots, u0, ..., um-1 all have value 0
    for (i = 0; i < order; i++)
        kv.push_back(0);

    //Knots um, ..., un-1 increases in increments of value 1, from 1 to n - m
    for (i = order; i < cpoints; i++)
        kv.push_back(value++);

    //The final m knots, un, ..., un+m-1 are all equal to n - m + 1
    for (i = cpoints; i < size; i++)
        kv.push_back(value);

    cout << endl << "[Number of control points] " << cpoints;
    cout << endl << "[Degree of spline] " << degree;

    //print out the knot vector
    cout << endl << endl << "[Knot vector] \nU = ( ";
    for (i = 0; i < kv.size(); i++)
        cout << kv[i] << " ";
    cout << ")" << endl;

    cout << endl;

}
```

Write a program that plots all the blending functions of degree 3 (m = 4) on the same screen.

Cubic interpolating polynomial is used to find a point for a certain value of the parameter u. Suppose the points at u = 0, 1/3, 2/3, 1 are:

P(0) = (0, 0, 0)
P(1/3) = (1, 2, 2)
P(2/3) = (2, 3, 4)
P(1) = (4, 5, 8)

Find the point at u = 0.8.

Formula used: $P = AC$ $C = A^{-1}P$

For ease of use, I used a matrix calculator online to get the results.

First, I calculated the inverse of A.

A^{-1}:

№	A1	A2	A3	A4
1	1	0	0	0
2	-115/34	63/17	-9/34	-1/17
3	9/17	-45/34	18/17	-9/34
4	63/34	-81/34	-27/34	45/34

Then I used $C = A^{-1}P$ to calculate C:

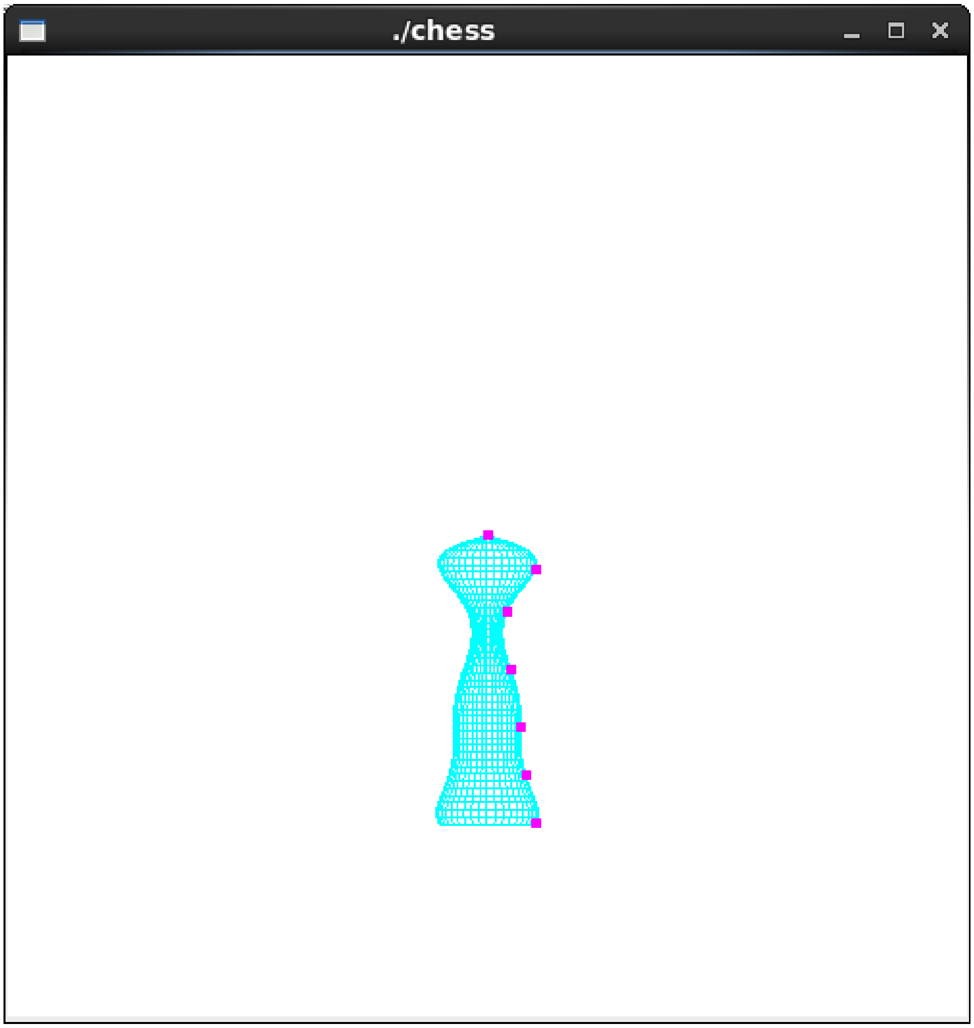
№	C1	C2	C3
1	0	0	0
2	50/17	215/34	100/17
3	-9/34	-27/34	-9/17
4	45/34	-9/17	45/17

Then I multiplied the results and A, which I got from the question, to get the final results:

№	C1	C2	C3
1	1216/425	9094/2125	2432/425

Therefore, when u = 0.8, P(0.8) = (2.86, 4.28, 5.72)

Write a program that uses B-splines and some control points to generate a profile and then use the profile and surface of revolution to generate a graphic chess piece like the one shown in class notes.



I added some control points to draw the chess piece. I chose to draw a pawn, although it does not look quite like the actual piece, this is the best I could do. I tried to use 14 control points instead of 7, but somehow that didn't work.

Code:

```
//chess.cpp

...

GLfloat ctrlpoints[7][3] = {

        { 0.0, 0.0, 0.0 }, { 0.35, 0.5, 0.0 },

        { 0.8, 0.2, 0.0 }, { 1.4, 0.25, 0.0 },

        { 2.0, 0.35, 0.0 }, { 2.5, 0.4, 0.0 },

        { 3.0, 0.5, 0.0 }//, { 1.6, 0.25, 0.0 },

};

...
```

```
void display(void)
{
  int i, j;
  float x, y, z, r;                              //current coordinates
  float x1, y1, z1, r1;                  //next coordinates
  float theta;

  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 1.0, 1.0);
  const float startx = 0, endx = 3;
  const int nx = 40;                             //number of slices along x-direction
  const int ntheta = 40;                         //number of angular slices
  const float dx = (endx - startx) / nx;         //x step size
  const float dtheta = 2*PI / ntheta;            //angular step size

  x = startx;
  //r = aLine ( x );
  r = polyint( ctrlpoints, x, 7);
  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

  for ( i = 0; i < nx; ++i ) {                   //step through x
    theta = 0;
    x1 = x + dx;                                 //next x
    //r1 = aLine ( x1 );                         //next f(x)
    r1 = polyint( ctrlpoints, x1, 7);            //next f(x)
    //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

        //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 i

  /* The following code displays the control points as dots. */
  glPointSize(5.0);
  glColor3f(1.0, 0.0, 1.0);
  glBegin(GL_POINTS);
    for (i = 0; i < 7; i++)
      glVertex3fv(&ctrlpoints[i][0]);
  glEnd();
  glPopMatrix();
  glFlush();
}
```

Report:

I successfully finished problem 1, 3, and 4 of the homework.