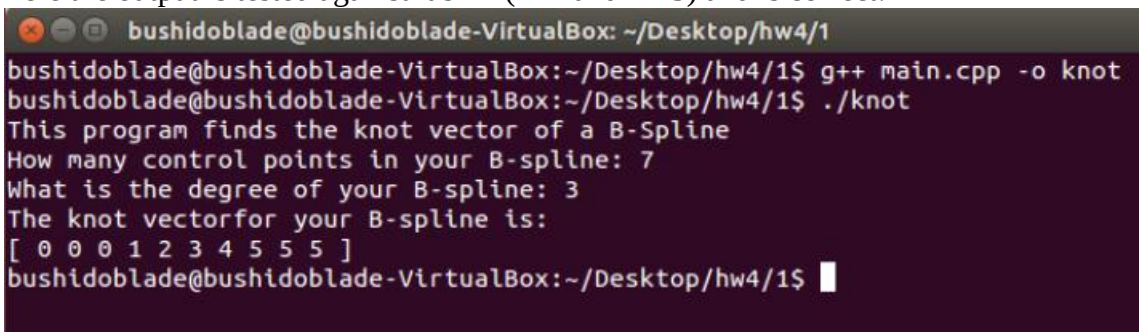
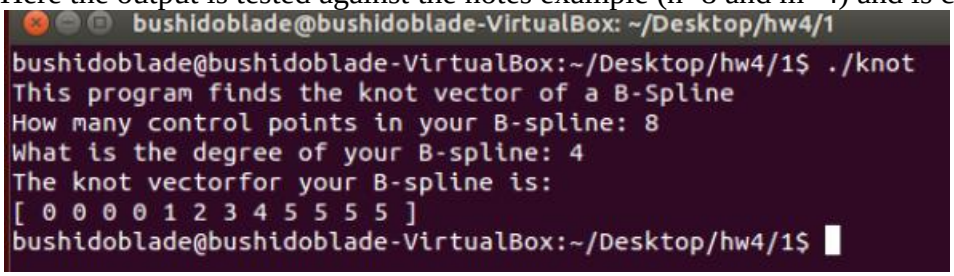
Andrew Yenalavitch
Homework 4
CSE 520 - Winter 2015

1. (10 points) 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.

Here the output is tested against lab 12 (n=7 and m=3) and is correct:



Here the output is tested against the notes example (n=8 and m=4) and is correct:



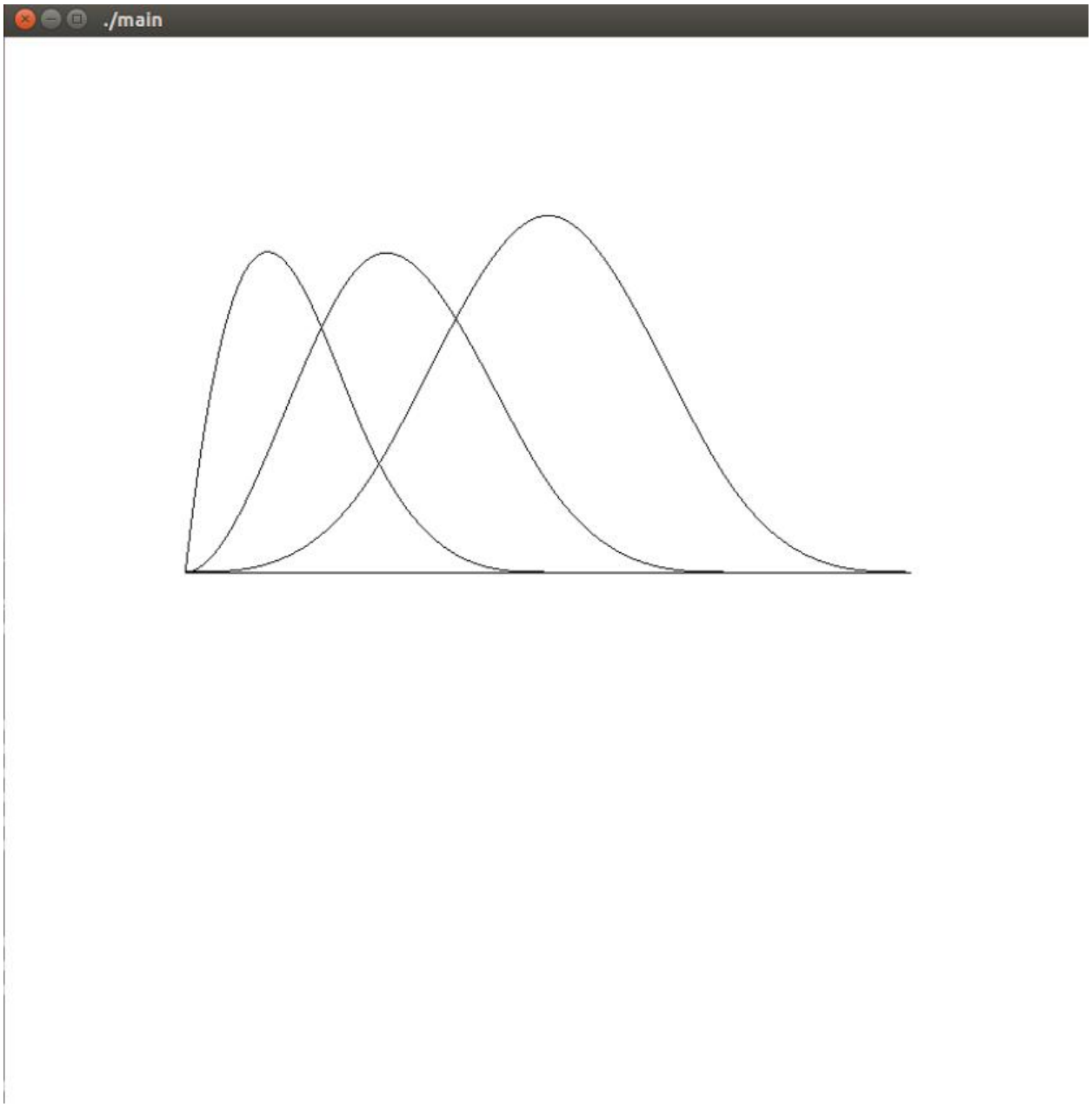
Source code:

```
#include <iostream>


using namespace std;

int main()
{
  int n = 0;
  int m = 0;

  cout << "This program finds the knot vector of a B-Spline" << endl;

  cout << "How many control points in your B-spline: ";
  cin >> n;

  cout << "What is the degree of your B-spline: ";
  cin >> m;

  if( m < 3 || n < m )
  {
```

```
    cout << "Unable to process. Check input values." << endl;
    return 0;
  }

  int last_knot = n - m + 1;
  int total_knots = n + m;
  int number = 1;

  int knot_vector[total_knots];


  for(int i = 0; i < m; ++i)
  {
    // first m terms are 0
    knot_vector[i] = 0;
  }

  for(int i = m; i < n; ++i)
  {
    // next n-m terms are from 1 to n-m
    knot_vector[i] = number;
    ++number;
  }

  for(int i = n; i < total_knots; ++i)
  {
    // last terms are equal to n - m + 1
    knot_vector[i] = last_knot;
  }

  //display knot vector
  cout << "The knot vectorfor your B-spline is:" << endl;
  cout << "[ ";
  for(int i = 0; i < total_knots; ++i){
    cout << knot_vector[i] << " ";
  }
  cout << "]" << endl;

  return 0;
}
```

2. (10 points) Write a program that plots all the blending functions of degree 3 (m = 4) on the same screen.

I used the buildKnots and bSpline functions from the lecture notes and plotted the points produced in a fashion similar to the sin wave assignment from 420. Here is the output:



Source code:

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

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;
  }
}

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;
}

void display()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  float knots[11];           //12
  buildKnots(4, 8, knots);  //12

  glBegin(GL_LINE_STRIP);
  for (int i = 1; i < 4; i++)
  {
    for (float x = 0.0; x < 4.0; x += 0.01)
    {
      float y = bSpline(i, 4, x, knots);
      glVertex2f(x, y);
    }
  }
  glEnd();
  glFlush();
}

void setWindow(double left,double right, double bottom, double top)
{
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(left, right, bottom, top);
}

void init()
{
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  glPointSize(4.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-1.0, 5.0, -1.0, 1.0);
}

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(800, 800);
  glutInitWindowPosition(100, 150);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

3. (10 points) 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.

From the lecture notes, P = AC.

$$P = \begin{vmatrix} 0 & 0 & 0 \\ 1 & 2 & 2 \\ 2 & 3 & 4 \\ 4 & 5 & 8 \end{vmatrix} \quad A = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & 1/3 & 1/9 & 1/27 \\ 1 & 2/3 & 4/9 & 8/27 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

from lab 12, we know the inverse of A is:

$$A^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{vmatrix}$$

$C = A^{-1}P$

$$C = \begin{vmatrix} 0 & 0 & 0 \\ 4 & 9.5 & 8 \\ -4.5 & -13.5 & -9 \\ 4.5 & 9 & 9 \end{vmatrix}$$

$$P(0.8) = \begin{vmatrix} 1 & 0.8 & 0.8^2 & 0.8^3 \end{vmatrix} * \begin{vmatrix} 0 & 0 & 0 \\ 4 & 9.5 & 8 \\ -4.5 & -13.5 & -9 \\ 4.5 & 9 & 9 \end{vmatrix}$$

P(0.8) = | 1 0.8 0.64 0.512 | * | 0 0 0 |
| 4 9.5 8 |
| -4.5 -13.5 -9 |
| 4.5 9 9 |

P(0.8) = | 2.624 3.568 5.248 |

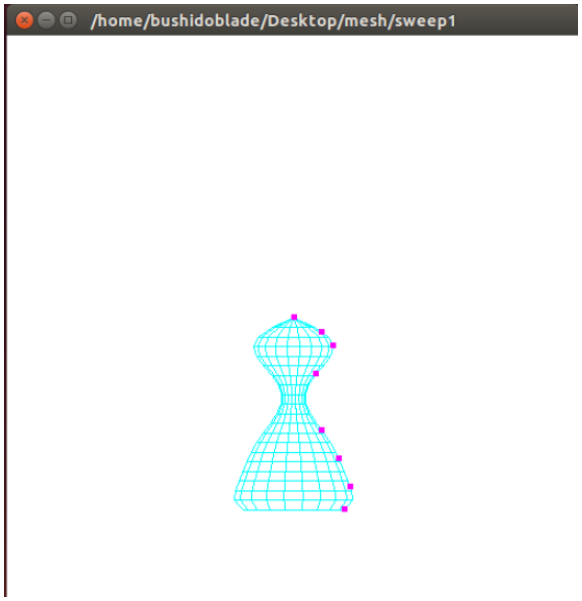
4. (20 points) Write a program that uses B-splines and some control points to generate a profile and then use the profile and surface of revolution to generate a graphic chess piece like the one shown in class notes. (You can choose any chess piece. You can gain extra credit by doing more than one piece.)

I started with sweep1.cpp in the mesh folder and modified the control points to resemble a pawn:

Here are the control points:

```
GLfloat ctrlpoints[8][3] = {
  {0.0, 0.0, 0.0},
  {0.25, 0.5, 0.0},
  {0.5, 0.7, 0.0},
  {1.0, 0.4, 0.0},
  {2.0, 0.5, 0.0},
  {2.5, 0.8, 0.0},
  {3.0, 1.0, 0.0},
  {3.4, 0.9, 0.0}
};
```

And here is the pawn:



I would liked to have used more control points, but there is an issue with sweep1.cpp that breaks the mesh with a higher number of control points.

I also modified the end point:

```
const float startx = 0.0, endx = 3.4;
```

and increased the number of slices:

```
const int nx = 20; //number of slices along x-direction
const int ntheta = 20; //number of angular slices
```

and added a control variable, "num_points" to make it easier to modify the number of control points:

```
int num_points = 8;
```

Source code:

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

using namespace std;
const double PI = 3.14159265389;
int num_points = 8;
int anglex= 0, angley = 0, anglez = 0; //rotation angles
int window;

//control points
GLfloat ctrlpoints[8][3] = {
  {0.0, 0.0, 0.0},
  {0.25, 0.5, 0.0},
  {0.5, 0.7, 0.0},
  {1.0, 0.4, 0.0},
  {2.0, 0.5, 0.0},
  {2.5, 0.8, 0.0},
  {3.0, 1.0, 0.0},
  {3.4, 0.9, 0.0}
};

void init(void) {
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glPolygonMode( GL_FRONT, GL_LINE ) ;
  glPolygonMode( GL_BACK, GL_LINE ) ;
  glShadeModel(GL_FLAT);
}

//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()
{
  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 = 0.0, endx = 3.4;
  const int nx = 20; //number of slices along x-direction
  const int ntheta = 20; //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 = polyint( ctrlpoints, x, num_points);
  glPushMatrix();
  glRotatef( anglex, 1.0, 0.0, 0.0); //rotate 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 = polyint( ctrlpoints, x1, num_points); //next f(x)
    //draw the surface composed of quads 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 incr. theta
      }
    glEnd();
    x = x1;
    r = r1;
  } //for i

  glPointSize(5.0);
  glColor3f(1.0, 0.0, 1.0);
  glBegin(GL_POINTS);
```

```
    for (i = 0; i < num_points; 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 reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 30.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt ( 0, 0, 15, 0, 0, 0, 0, 1, 0 );
}
*/

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 (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc (keyboard);
   glutMainLoop();
   return 0;
}
```

5. (20 points) Find a Frenet frame for the toroidal spiral given by

$$x(t) = [1 + 0.5 * \cos (7t)] \cos (t)$$
$$y(t) = [1 + 0.5 * \cos (7t)] \sin (t)$$
$$z(t) = 0.5 * \sin (7t)$$

Write a program that generates a tube using the above curve.

I started with the code I used for lab 15 and modified it to match the above formula

Here is C(t):

C(t) = (x(t), y(t), z(t))

x(t) = (1 + b*cos (7 * t)) * cos (t)

y(t) = (1 + b*cos (7 * t)) * sin (t)

z(t) = b*sin (7 * t)

b = 0.5

Here is the 1st derivative of C(t):

x'(t) = -sin(t) * (1 + b * cos(7 * t)) - 7 * b * sin(7 * t) * cos(t)

y'(t) = cos(t) * (1 + b * cos(7 * t)) - 7 * b * sin(7 * t) * sin(t)

z'(t) = b * 7 * cos(7 * t)

Here is the 2nd derivative of C(t):

x"(t) = b * 14 * sin(t) * sin(7 * t) - b * 49 * cos(t) * cos(7 * t) - cos(t) * (1 + b * cos(7 * t))

y"(t) = - b * 14 * cos(t) * sin(7 * t) - b * 49 * sin(t) * cos(7 * t) - sin (t) * (1 + b * cos(7 * t))

z"(t) = - b * 49 * sin(7 * t)

We now have enough to make the code work. I use the same technique as in lab15; using the first and second derivatives and various VectorR4 cross products to find the Tangent, Normal and Binormal vectors. Here are the changes to the setM function from my lab15.:

```
//Matrix for transforming to Frenet frame
void setM( LinearMapR4 &M, float t, float b )
{

  VectorR4 C,T,B,N,A;
  // C = curve
  C.x = (1 + b * cos(7 * t)) * cos (t);
  C.y = (1 + b * cos(7 * t)) * sin (t);
  C.z = b * sin(7 * t);
  C.w = 1;

  // T = normalized 1st derivative
  // 1st derivative
  T.x = -sin(t) * (1 + b * cos(7 * t)) - 7 * b * sin(7 * t) * cos(t);
  T.y = cos(t) * (1 + b * cos(7 * t)) - 7 * b * sin(7 * t) * sin(t);
  T.z = b * 7 * cos(7 * t);
  T.w = 0;
  // normalize T
  T.Normalize();

  //2nd derivative
  A.x = b * 14 * sin(t) * sin(7 * t) - b * 49 * cos(t) * cos(7 * t) - cos(t) * (1 + b
* cos(7 * t));
  A.y = -b * 14 * cos(t) * sin(7 * t) - b * 49 * sin(t) * cos(7 * t) - sin(t) * (1 + b
* cos(7 * t)) ;
  A.z = -b * 49 * sin(7 * t);
  A.w = 0;

  //N = T X A (cross product)
  N.x = T.y * A.z - T.z * A.y;
  N.y = T.z * A.x - T.x * A.z;
  N.z = T.x * A.y - T.y * A.x;
  N.w = 0;
```

```
  // normalize N
  N.Normalize();

  // B = T X N (cross product)
  B.x = T.y * N.z - T.z * N.y;
  B.y = T.z * N.x - T.x * N.z;
  B.z = T.x * N.y - T.y * N.x;
  B.w = 0;

  //Normal    N(t)
  M.SetColumn1( N.x, N.y, N.z, N.w );
  //Binormal  B(t)
  M.SetColumn2( B.x, B.y, B.z, B.w );
  //Tangent   T(t)
  M.SetColumn3( T.x, T.y, T.z, T.w );
  //The curve C(t)
  M.SetColumn4( C.x, C.y, C.z, C.w );
}
```

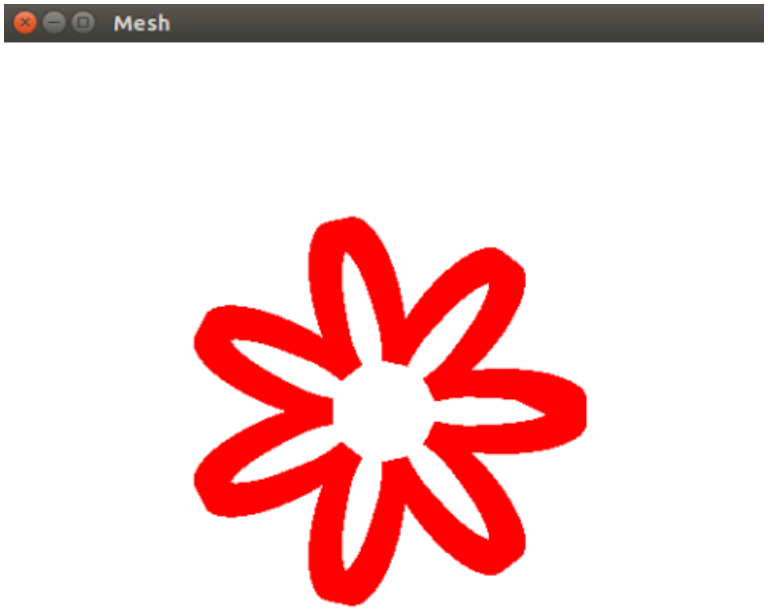
And I set the b value in the display function to 0.5 to match the formula:

```
const float b = 0.5;
```

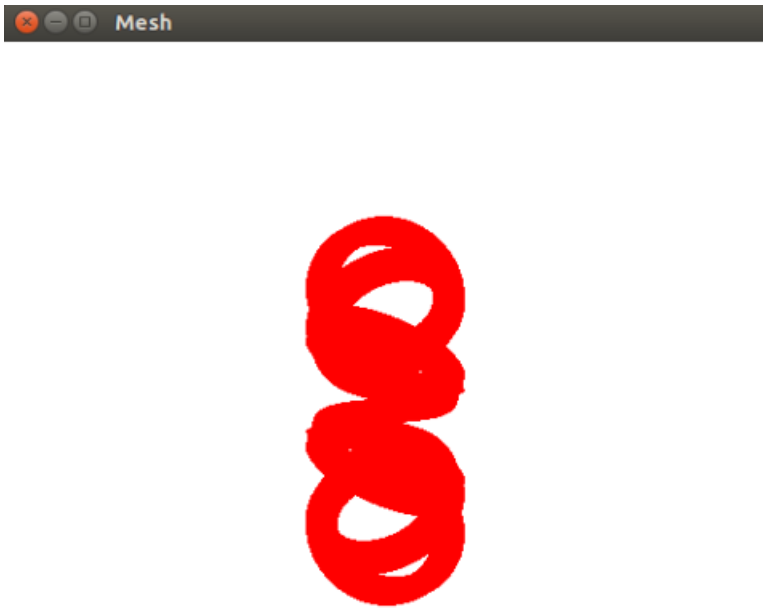
And lastly I changed the color to red:

```
glColor3f ( 1.0, 0.0, 0 );
```

Default output:



Rotated 90 degrees along the y-axis:



Source code:

```
#include <SDL/SDL.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include "LinearR4.h"

using namespace std;

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

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);
```

```
}

//toroidal spiral
void get_C ( float C[4], float t, float b )
{
  C[0] = (1 + b * cos( 7 * t ) ) * cos ( t );
  C[1] = (1 + b * cos( 7 * t ) ) * sin ( t );
  C[2] = b * sin( 7 * t );
  C[3] = 1;
}


//Matrix for transforming to Frenet frame
void setM( LinearMapR4 &M, float t, float b )
{

  VectorR4 C,T,B,N,A;
  // C = curve
  C.x = (1 + b * cos(7 * t)) * cos (t);
  C.y = (1 + b * cos(7 * t)) * sin (t);
  C.z = b * sin(7 * t);
  C.w = 1;

  // T = normalized 1st derivative
  // 1st derivative
  T.x = -sin(t) * (1 + b * cos(7 * t)) - 7 * b * sin(7 * t) * cos(t);
  T.y = cos(t) * (1 + b * cos(7 * t)) - 7 * b * sin(7 * t) * sin(t);
  T.z = b * 7 * cos(7 * t);
  T.w = 0;
  // normalize T
  T.Normalize();

  //2nd derivative
  A.x = b * 14 * sin(t) * sin(7 * t) - b * 49 * cos(t) * cos(7 * t) - cos(t) * (1 + b
* cos(7 * t));
  A.y = -b * 14 * cos(t) * sin(7 * t) - b * 49 * sin(t) * cos(7 * t) - sin(t) * (1 + b
* cos(7 * t)) ;
  A.z = -b * 49 * sin(7 * t);
  A.w = 0;

  //N = T X A (cross product)
  N.x = T.y * A.z - T.z * A.y;
  N.y = T.z * A.x - T.x * A.z;
  N.z = T.x * A.y - T.y * A.x;
  N.w = 0;
  // normalize N
  N.Normalize();

  // B = T X N (cross product)
  B.x = T.y * N.z - T.z * N.y;
  B.y = T.z * N.x - T.x * N.z;
  B.z = T.x * N.y - T.y * N.x;
  B.w = 0;

  //Normal    N(t)
  M.SetColumn1( N.x, N.y, N.z, N.w );
  //Binormal  B(t)
  M.SetColumn2( B.x, B.y, B.z, B.w );
  //Tangent   T(t)
  M.SetColumn3( T.x, T.y, T.z, T.w );
  //The curve C(t)
  M.SetColumn4( C.x, C.y, C.z, C.w );
}
```

```
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 = 0.5;           //constant of toroidal spiral
  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 );
  }
  glEnd();
*/
  glColor3f ( 1.0, 0.0, 0 );
```

```
  float p3[3];             //3-D point, (x, y, z)
  //starting
  setM ( M, 0, b ); //t = 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 += 0.01 ) { // decreased
    setM ( M, t, b );
    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;
}
```

Evaluation: I successfully completed all parts of the homework and am giving myself 70 points.