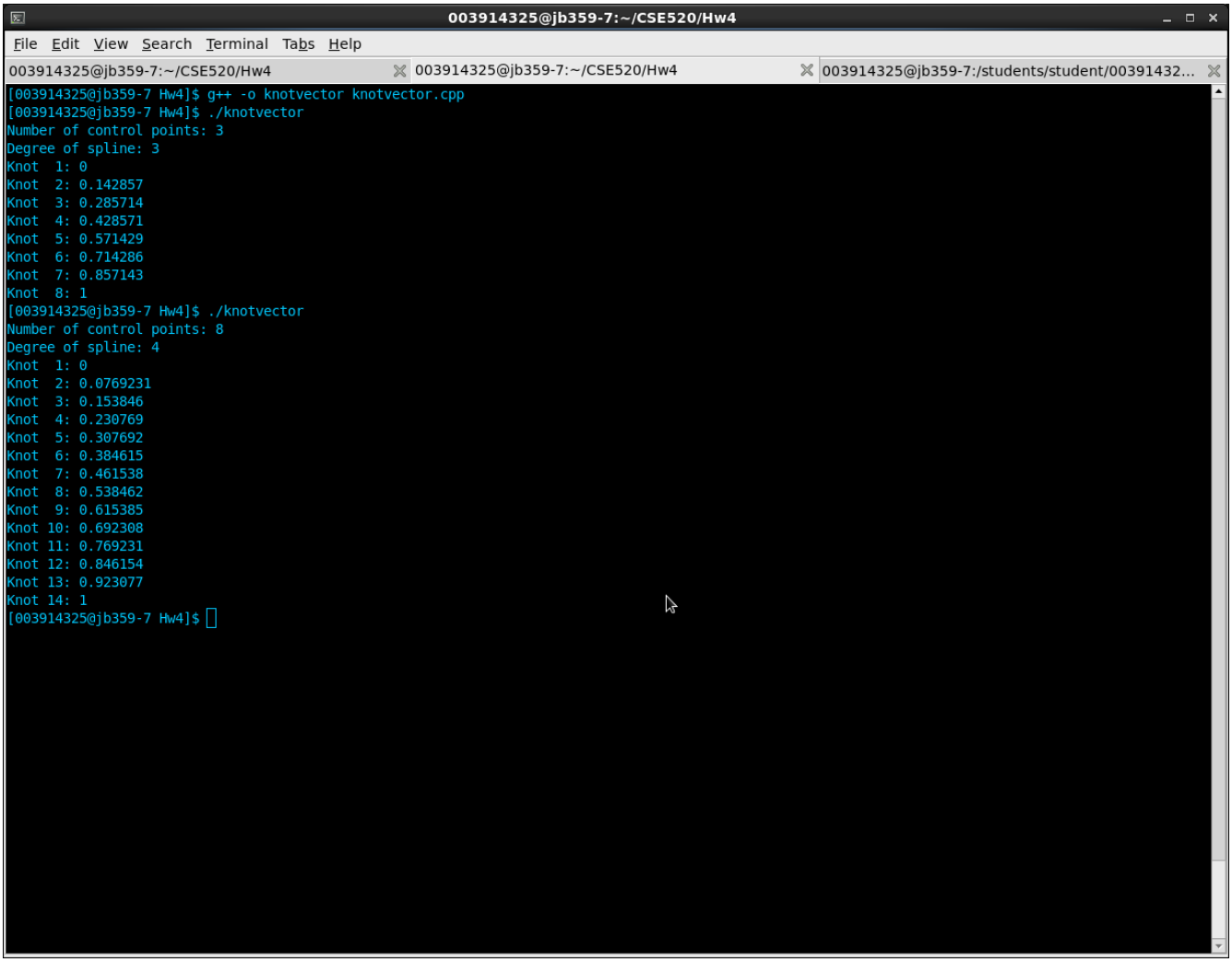
CSE520
Samuel Marrujo
Professor Yu
Homework 4

Homework 4

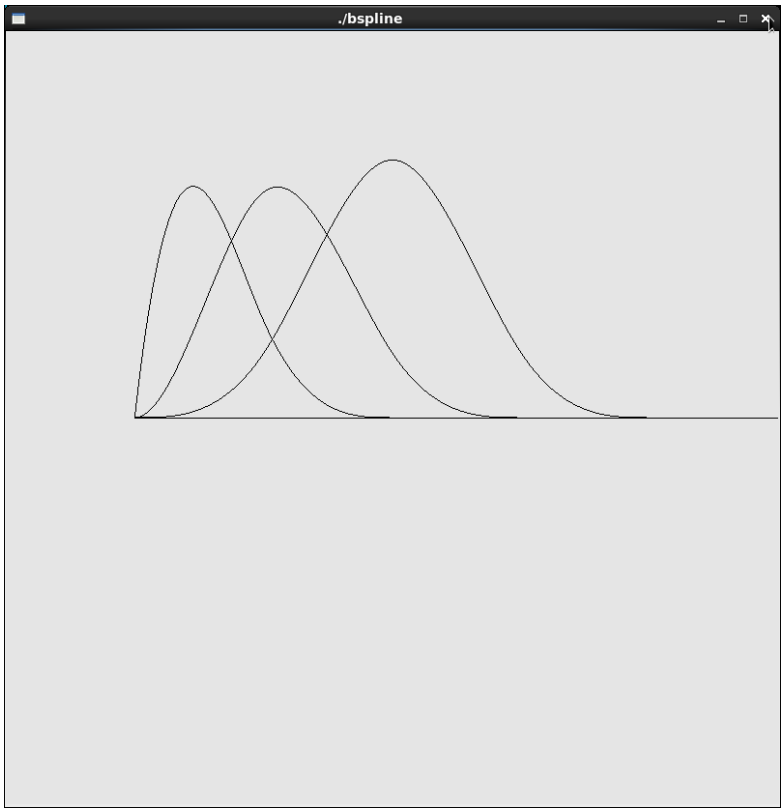
In this homework, we are to finish 5 different problems. I have successfully completed this task to the best of my ability, and here are pictures of the following:

Each picture will be on a separate page as shown below.

#1:


003914325@jb359-7:~/CSE520/Hw4
File Edit View Search Terminal Tabs Help
003914325@jb359-7:~/CSE520/Hw4
003914325@jb359-7:~/CSE520/Hw4
003914325@jb359-7:/students/student/00391432...
[003914325@jb359-7 Hw4]$ g++ -o knotvector knotvector.cpp
[003914325@jb359-7 Hw4]$ ./knotvector
Number of control points: 3
Degree of spline: 3
Knot 1: 0
Knot 2: 0.142857
Knot 3: 0.285714
Knot 4: 0.428571
Knot 5: 0.571429
Knot 6: 0.714286
Knot 7: 0.857143
Knot 8: 1
[003914325@jb359-7 Hw4]$ ./knotvector
Number of control points: 8
Degree of spline: 4
Knot 1: 0
Knot 2: 0.0769231
Knot 3: 0.153846
Knot 4: 0.230769
Knot 5: 0.307692
Knot 6: 0.384615
Knot 7: 0.461538
Knot 8: 0.538462
Knot 9: 0.615385
Knot 10: 0.692308
Knot 11: 0.769231
Knot 12: 0.846154
Knot 13: 0.923077
Knot 14: 1
[003914325@jb359-7 Hw4]$

#2:



#3
 This was done by hand. The solution is as follows:

[2.624   3.568   5.248]

How this was approached was done by the following method:
First, consider that we can express a cubic interpolating polynomial in a matrix form:

$$P = AC$$

Where P is the given matrix for values P(0), P(1/3), P(2/3), P(1).
      C is a variable matrix and is a 3x4 matrix for each 0,..,3, by x,y,z.
      A is the matrix given in the lecture notes

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1/3 & (1/3)^2 & (1/3)^3 \\ 1 & 2/3 & (2/3)^2 & (2/3)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Then, from lab 14,

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

Then solve for C: C = Ainv*P =

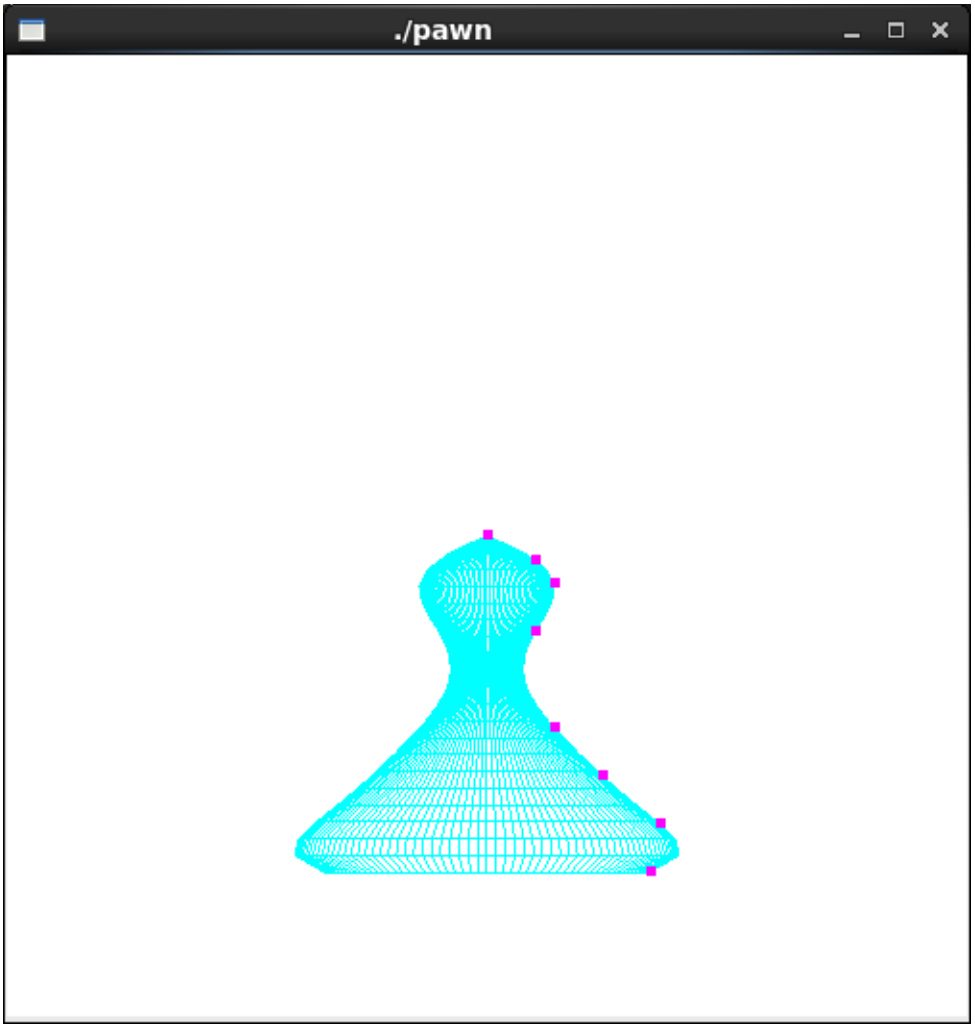
$$\begin{bmatrix} 0 & 0 & 0 \\ 4 & 9.5 & 8 \\ -4.5 & -13.5 & -9 \\ 4.5 & 9 & 9 \end{bmatrix}$$

Then, consider P(u) = [x(u), y(u), z(u)], and plug in 0.8 = u

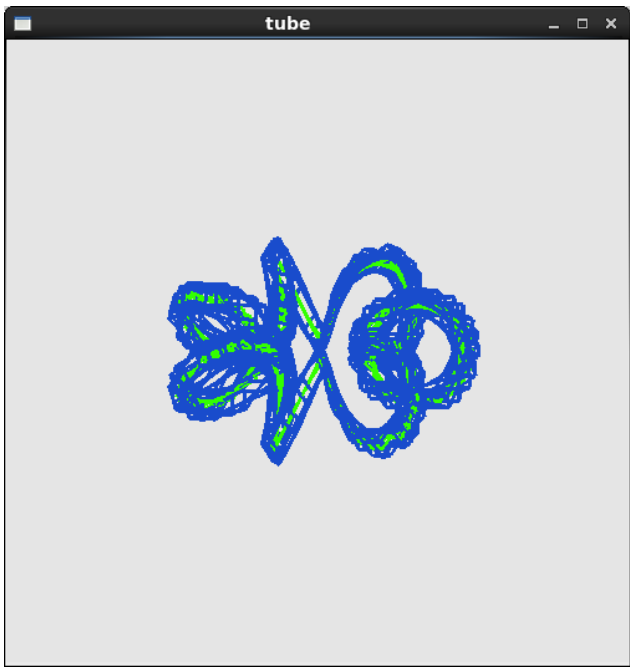
=>

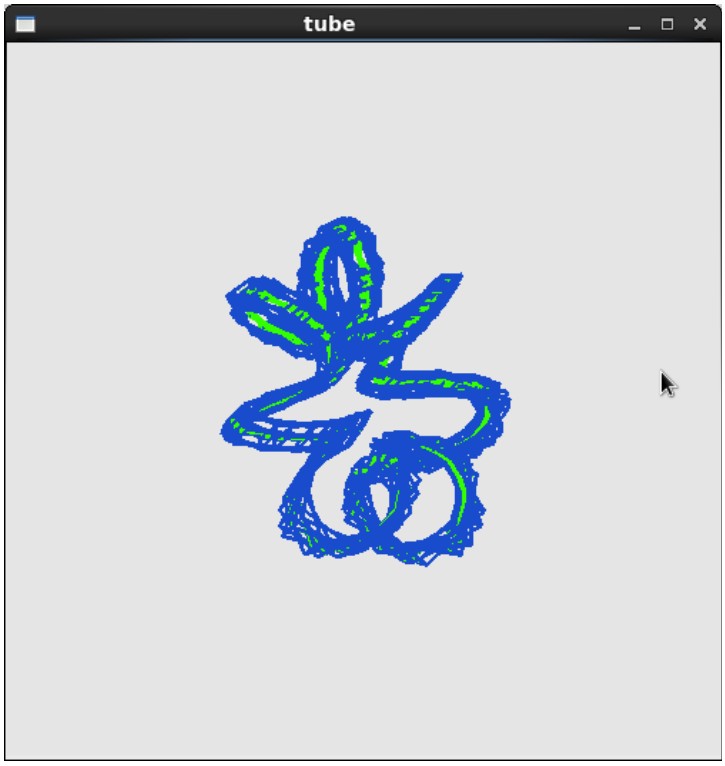
$$P(u) = P(0.8) = \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \end{bmatrix} * C = [2.624\ 3.568\ 5.248]$$

#4



#5





Code:
#1 is obvious. Created it's own program and used a formal way of defining points for a B-spline Curve. This way, it is used to define the control points as it should to it's exact values.

#2:
Used code similar from the lecture notes (buildknots)
//bspline.cpp

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

const int dx = 0.3;
const int maxL = 10;

float p[maxL][3];

using namespace std;

void build(int m, int n, float knt[]) {
	if (n < m)
		return; //this means there isn't enough control points, so it returns
	for (int i = 0; i < n + m; ++i) {
		if (i < m)
			knt[i] = 0.0;
		else if (i < n)
			knt[i] = i - m + 1;
		else
			knt[i] = n - m + 1;
	}
}

float bSpline(int k, int s, float u, float knt[]) {
	float a;
	float b;
	float sum = 0.0;
	if (s == 1)
		return (knt[k] <= u && u <= knt[k + 1]);
	a = knt[k + s - 1] - knt[k];
	if (a != 0)
		sum = (u - knt[k]) * bSpline(k, s - 1, u, knt) / a;
	b = knt[k + s] - knt[k + 1];
	if (b != 0)
		sum += (knt[k + s] - u) * bSpline(k + 1, s - 1, u, knt) / b;
	return sum;
}

void display(void) {
	float knt[11];
	build(4, 8, knt);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glPushMatrix();
	glBegin(GL_LINE_STRIP);
	for (int i = 1; i < 4; i++) {
		for (float x = 0.0; x < 5.0; x += 0.01) {
			float y = bSpline(i, 4, x, knt);
			glVertex2f(x, y);
```

```
                }
        }
        glEnd();
        glFlush(); //renderer
}

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

void init(void) {
        glClearColor(0.9, 0.9, 0.9, 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); //sets the window size on screen
        glutInitWindowPosition(100, 150); //sets the window position on screen
        glutCreateWindow(argv[0]);
        init();
        glutDisplayFunc(display); //references the display function
        glutKeyboardFunc(keyboard); //used to exit the program with 'ESC'
        glutMainLoop();
        return 0;
}
```

#3
See above
#4

Used sweep1.cpp to begin creating the pawn.

```
/*  sweep1.cpp
*  Construct surfaces of revolution using wireframe but have not considered
lighting.
*  Surface is generated by revolving a curve around x-axis.
*  A curve f(x) is generated by polynomial interpolation from some control points
*  or by some interested functions.
*
*  @Author: T.L. Yu, Fall 2008
*/
#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 Npoints = 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, 0.5, 0.0},
  {2.0, 0.7, 0.0}, {2.5, 1.2, 0.0}, {3.0, 1.8, 0.0}, {3.5, 1.7, 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;
}

float aLine ( float x )
{
  return x + 2.5;
}

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 = 0.0, endx = 3.5;
   const int nx = 20;       //number of slices along x-direction
   const int ntheta = 128;     //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, Npoints);
   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, Npoints);   //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

   glPointSize(5.0);
   glColor3f(1.0, 0.0, 1.0);
   glBegin(GL_POINTS);
      for (i = 0; i < Npoints; i++)
         glVertex3fv(&ctrlpoints[i][0]);
   glEnd();
   glPopMatrix();
   glFlush();
}
```

#5
This code started off from the helix-tube1.cpp code in the extrude folder of meshes in lab4.

This was suggested from the lecture notes to look in here.

```
int angle_x= 0, angle_y = 0, angle_z = 0;
int window;

void getC(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;
}

void setM( LinearMapR4 &Mat, float t, float b ) {

      float c = 1.0 / sqrt(1 + b*b);
      Mat.SetColumn1(-cos(t)*(1 -b * cos ( 7*t )), -sin(t)*(-b * cos(7*t)), 0,
0 );
      Mat.SetColumn2(c*b*sin(t)*(1 + b*cos(7*t)), -c*b*cos(t)*(1 - b*cos(7*t)), c,
0 );
      Mat.SetColumn3(-c*sin(t)*(1 + b*cos(7*t)), c*cos(t)*( 1 + b*cos(7*t)), b*c,
0);
      Mat.SetColumn4(cos(t)*(1 + b*cos(7*t)), sin(t)*(1 + b*cos(7*t)), b*sin(7*t),
1);

}

class Cfloat3 {
      public:
            float p3[3];
};

const float b = 0.5;
double H = 6.0;
LinearMapR4 Mat;
const int N = 4;

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

      vector<Cfloat3>vp0(N), vp1(N);
      VectorR4 p_1;
      VectorR4 points[4];

      points[0] = VectorR4 (-0.1, -0.1, 0, 1 );
      points[1] = VectorR4 ( 0.1, -0.1, 0, 1 );
      points[2] = VectorR4 ( 0.1, 0.1, 0, 1 );
      points[3] = VectorR4 ( -0.1, 0.1, 0, 1 );
      glColor3f ( 0.2, 1.0, 0 );
      glPushMatrix();
      glRotatef(angle_x, 1.0, 0.0, 0.0);
      glRotatef(angle_y, 0.0, 1.0, 0.0);
      glRotatef(angle_z, 0.0, 0.0, 1.0);

      float C[4];
```

```
	glLineWidth(3);
	glPolygonMode(GL_FRONT, GL_LINE);
	glPolygonMode(GL_BACK, GL_LINE);

	glBegin(GL_LINE_STRIP);
	for(float t = 0; t <= 26; t += 0.2) {
		getC(C, t, b);
		glVertex4fv(C);
	}
	glColor3f(0.1, 0.3, 0.8);
	glEnd();
	float p3[3];


	setM(Mat, 0, b);

	for (int i = 0; i < 4; ++i) {
		p_1 = Mat * points[i];
		p_1.Dump(vp0[i].p3);
	}
	glBegin(GL_QUADS);

	for (float t = 0.2; t <= 26; t += 0.2) {
		setM (Mat, t, b);
		for (int i = 0; i < N; ++i) {
			p_1 = Mat * points[i];

			p_1.Dump(vp1[i].p3);
		}
		for (int i = 0; i < N; ++i) {
			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());
	}
	glEnd();
	glPopMatrix();
	glFlush();
}

void keyboard ( unsigned char key, int x, int y ) {
	switch (key) {
		case 27:
			glutDestroyWindow(window);
			exit (0);
		case 'x':
			angle_x = (angle_x + 3) % 360;
			break;
		case 'X':
			angle_x = (angle_x - 3) % 360;
			break;
		case 'y':
			angle_y = (angle_y + 3) % 360;
			break;
		case 'Y':
```

```
                  angle_y = (angle_y - 3) % 360;
                  break;
            case 'z':
                  angle_z = (angle_z + 3) % 360;
                  break;
            case 'Z':
                  angle_z = (angle_z - 3) % 360;
                  break;
            case 'r':
            case 'R':
                  angle_z = angle_y = angle_x = 0; //resets the angles
                  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("tube");
      glutDisplayFunc(display);
      glutKeyboardFunc( keyboard );
      glClearColor( 0.9f, 0.9f, 0.9f, 0.0f ); //light gray background
      glViewport ( 0, 0, 500, 500 );
      init ();
      glutMainLoop();
      return 0;
}
```