Programming Project #3
Toroidal Spirals

CS 442/542

Due Wednesday, October 29, 2008 by Midnight

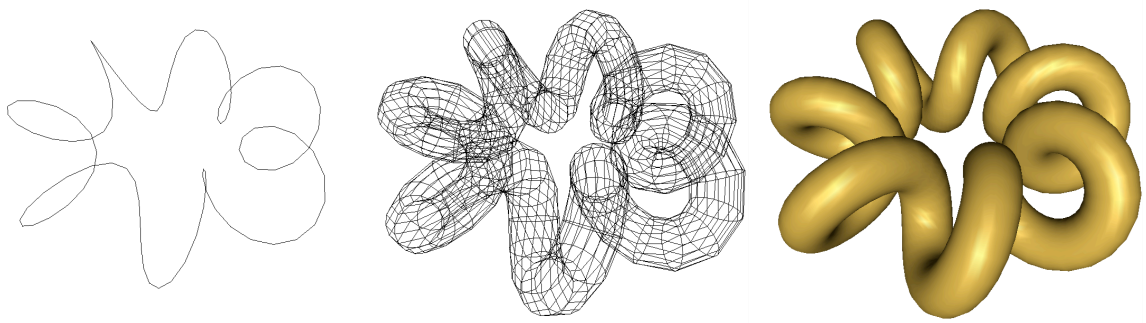


Figure 1: Spine curve of toroidal spiral (left). Mesh formed from extruded circle (center). Shaded tube (right).

1 Overview

This programming project is designed to give you some experience with the following:

- creating a 3D polygonal mesh,
- viewing and projecting,
- visible surface determination using a depth buffer, and
- lighting and shading.

You will create a mesh that approximates a surface defined by extruding a circle along the path of a toroidal spiral. The project consists of three phases. For the first phase, you will write an OpenGL program that simply draws the spine curve as shown on the left of Figure 1. In the second phase, you will build and render a wire frame mesh as shown in the center of Figure 1. Finally, you will render a smooth shaded version of the visible portion of the surface.

2 The Spine Curve

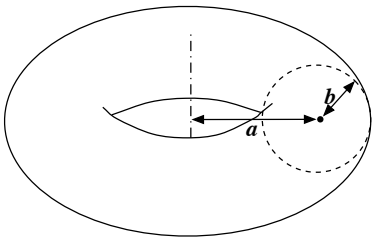
The first phase of this project is to create a program that draws the *spine curve*

$$\mathbf{C}(t) = (x(t), y(t), z(t)), \quad 0 \le t < 2\pi \tag{1}$$

of your toroidal spiral, where

$$x(t) = (a + b\cos qt)\cos pt, \tag{2}$$
$$y(t) = (a + b\cos qt)\sin pt, \tag{3}$$
$$z(t) = b\sin qt. \tag{4}$$

This curve wraps around the surface of a torus (i.e., donut) centered at the origin where a and b are the large and small radii of the torus as shown in the following figure:



The value q controls the number of times the curve coils around the torus, and p determines how many times the curve winds around the origin.

2.1 Drawing the Spine

Draw the spine curve as a single closed polyline (i.e., via `GL_LINE_LOOP`) with N vertices $\{\mathbf{C}(t_i)\}_{i=0}^{N-1}$ where $t_i = 2\pi i/N$.

2.2 Viewing the Spine

Place the eye at some convenient location (typically some point outside of the curve's bounding box is a nice choice). The "look-at point" should be the origin, and $(0, 0, 1)$ is the "up direction" (as long as this is not parallel to the view direction). Use a "reasonable" perspective projection (e.g., field of view 40°).

3 The Tube Mesh

The second phase of this project is to create a mesh for a "circular tube" whose centerline follows the previously defined spine. At each sample point $\mathbf{C}(t_i)$ along the spine we need to form the *Frenet*

frame in which we draw the tube's circular cross section. First we find the curve's tangent vector $\mathbf{T}(t_i)$ at each sample which can be computed from the "velocity vector"

$$\mathbf{T}(t) = \dot{\mathbf{C}}(t) = (\dot{x}(t), \dot{y}(t), \dot{z}(t)). \tag{5}$$

The corresponding derivatives are

$$\begin{aligned} \dot{x}(t) &= -p(a + b\cos qt)\sin pt - bq\sin qt\cos pt \\ &= -p\ y(t) - bq\sin qt\cos pt, & (6) \\ \dot{y}(t) &= p(a + b\cos qt)\cos pt - bq\sin qt\sin pt \\ &= p\ x(t) - bq\sin qt\sin pt, & (7) \\ \dot{z}(t) &= bq\cos qt. & (8) \end{aligned}$$

Now we need to find another vector that is not parallel to $\mathbf{T}(t_i)$. Where the spine is curved, the "acceleration vector" $\mathbf{A}(t)$ will be close to perpendicular to $\mathbf{T}(t_i)$:

$$\mathbf{A}(t) = \ddot{\mathbf{C}}(t) = (\ddot{x}(t), \ddot{y}(t), \ddot{z}(t)). \tag{9}$$

The corresponding second derivatives are

$$\begin{aligned} \ddot{x}(t) &= -p\ \dot{y}(t) + bq(p\sin qt\sin pt - q\cos qt\cos pt), & (10) \\ \ddot{y}(t) &= p\ \dot{x}(t) - bq(p\sin qt\cos pt + q\cos qt\sin pt), & (11) \\ \ddot{z}(t) &= -q^2 b\sin qt. & (12) \end{aligned}$$

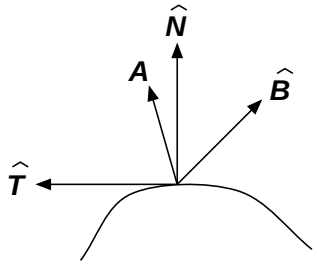
Now we can compute the "binormal vector" $\mathbf{B}(t)$, which will be perpendicular to $\mathbf{T}(t)$, via the cross product

$$\mathbf{B}(t) = \mathbf{T}(t) \times \mathbf{A}(t). \tag{13}$$

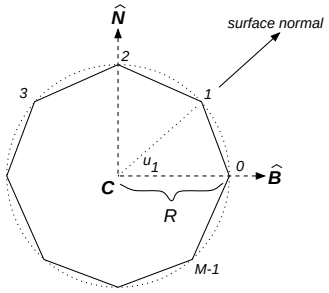
Finally, we compute the "normal vector" $\mathbf{N}(t)$ which is perpendicular to both $\mathbf{B}(t)$ and $\mathbf{T}(t)$ via the cross product

$$\mathbf{N}(t) = \mathbf{B}(t) \times \mathbf{T}(t). \tag{14}$$

Our three coordinate vectors defining our Frenet frame at t_i, after vector normalization, are $(\hat{\mathbf{N}}(t_i), \hat{\mathbf{B}}(t_i), \hat{\mathbf{T}}(t_i))$.



The following figure shows the cross section of our tube with radius R :



We approximate the circle with a polyline of M points

$$\{\mathbf{C} + R(\cos u_j \hat{\mathbf{B}} + \sin u_j \hat{\mathbf{N}})\}_{j=0}^{M-1}$$

where $u_j = 2\pi j/M$. We also note that the corresponding surface normals (which we will need later) are

$$\{\cos u_j \hat{\mathbf{B}} + \sin u_j \hat{\mathbf{N}}\}_{j=0}^{M-1}$$

(note these are already have unit magnitude). We can store our mesh in an array containing $N \times M$ 3D points

```
GLdouble tube[TUBE_N][TUBE_M][3];
GLdouble tubeNormals[TUBE_N][TUBE_M][3];
```

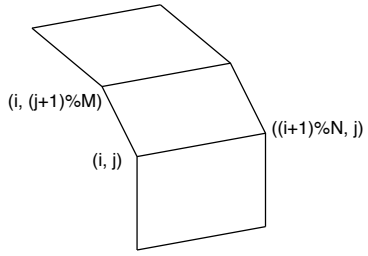
Therefore we can create our mesh as follows:

```
dt = 2*M_PI/TUBE_N;
du = 2*M_PI/TUBE_M;
for (i = 0, t = 0.0; i < TUBE_N; i++, t += dt) {
  GLdouble C[3], T[3], A[3], B[3], N[3];
  C[0] = x(t), C[1] = y(t), C[2] = z(t);
  T[0] = dx(t), T[1] = dy(t), T[2] = dz(t);
  A[0] = ddx(t), A[1] = ddy(t), A[2] = ddz(t);
  cross(T,A,B);
  normalize(T);
  normalize(B);
  cross(B,T,N);
  for (j = 0, u = 0.0; j < TUBE_M; j++, u += du)
    for (k = 0; k < 3; k++) {
      tubeNormals[i][j][k] = cos(u)*B[k] + sin(u)*N[k];
      tube[i][j][k] = C[k] + R*tubeNormals[i][j][k];
    }
}
```

The functions `dx(t)` and `ddx(t)` compute $\dot{x}(t)$ and $\ddot{x}(t)$ respectively.

3.1 Drawing a Wire Frame Mesh

Draw each quadrilateral boundary in the mesh using `GL_LINES` or `GL_LINE_STRIP` with the appropriate vertex connections. For efficiency sake, try to construct a scheme that draws each edge exactly once. The following figure shows a quadrilateral connecting cross section i with cross section $(i+1)\%N$.



4 Shading the mesh

Draw the mesh as N strips of quadrilaterals using `GL_QUAD_STRIP`. Define each quadrilateral so the its vertices are specified in counter-clockwise order from their visible side. Remember to specify normals (via `glNormal3()`) for each vertex.

Set up a single light source with added ambient light. Placing the light source at our near the eye usually works well.

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light0_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Pick some nice surface properties and use smooth shading to render the tube.

```
glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
glMaterialfv(GL_FRONT, GL_SHININESS, &shininess);
glShadeModel(GL_SMOOTH);
```

4.1 Visible Surface Determination

Create a *depth buffer* and enable it.

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
...
glEnable(GL_DEPTH_TEST);
```

If you intend to animate your surface (or drag it with the mouse) use double buffering (i.e., use `GLUT_DOUBLE` instead of `GLUT_SINGLE`). Clear the both the depth buffer and the frame buffer before rendering each frame:

```
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ...
}
```

Once the above seems to be working properly, you can enable *back face culling* for efficiency. This depends on the fact that you specified the vertices of each quadrilateral in counter-clockwise order from its visible side.

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

5 Implementation

5.1 Picking tube parameters

The following table lists the parameters that define the spine's path and the size of the tube.

N	number of spine samples
M	number of samples in tube cross sections
a	"big radius" of torus
b	"little radius" of torus
R	tube radius
p	number of winds around the origin
q	number of coils

Figure 2 shows three tubes using several choices of p and q. I used $a = 100$, $b = 40$, and $R = 20$ for these surfaces. You are free to choose any parameters you like, as long as an "interesting surface" is produced. Pick reasonable values for N and M that yield the appropriate coarseness for your mesh.

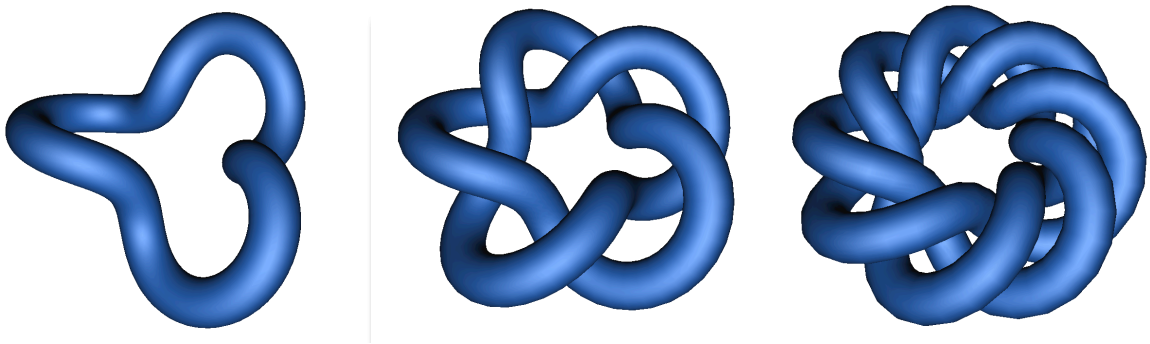


Figure 2: Tubes for $(p, q) = (1, 3), (2, 5), (3, 8)$ from left to right.

5.2 Bonus

Feel free to animate your surface or allow the user to rotate the surface by dragging the mouse. I store the eye location in spherical coordinates (i.e., (ρ, θ, ϕ) where ρ is the eye's distance from the origin, and θ and ϕ are the view angles) and create a new view transformation each time the user drags the mouse. Rotating the eye in spherical coordinates is easy. I convert (ρ, θ, ϕ) to cartesian coordinates (x, y, z) and use the `gluLookAt()` function to initialize the model-view matrix. The values `eyeRho, eyeTheta,` and `eyePhi` are initialized elsewhere and are updated using the `mouseMotion()` callback based on how many pixels the mouse has been dragged over.

```
int mousex, mousey;

void mouse(int button, int state, int x, int y) {
  mousex = x;
  mousey = y;
}

void mouseMotion(int x, int y) {
  int dx = x - mousex, dy = y - mousey;
  double eyex, eyey, eyez;
  eyeTheta -= dx*RADIANS_PER_PIXEL;
  eyePhi -= dy*RADIANS_PER_PIXEL;
  mousex = x, mousey = y;
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  sphericalToCartesian(eyeRho, eyeTheta, eyePhi,
                       &eyex, &eyey, &eyez);
  gluLookAt(eyex, eyey, eyez, 0,0,0, 0,0,1);
  glutPostRedisplay();
}
```

I usually add some extra code to make sure $\epsilon < \phi < \pi - \epsilon$ for some small constant ϵ.

5.3 Submission

You will write three OpenGL programs, one for each phase of the project. Make sure your code is portable (at least it should compile without alteration under Linux, Win32, and Mac OS X). All of your source code, documentation, executables, etc... will be archived in to a single compressed archive file and submitted electronically. With every project submission you are to include a text file named `README` that tells me who you are with the appropriate contact (email address and student ID).

Follow the instructions at the following site to submit your solution:

`http://ezekiel.vancouver.wsu.edu/~cs442/submit/submit.html`

Your project is due at midnight on the due date. Have fun!