CS 465 Homework 2b

(revised September 26, 2003)

out: Monday 22 September 2003
due: Monday 29 September 2003

In this assignment, you will explore the idea of parametric surfaces in 3D by implementing several such surfaces. The framework for this assignment draws the surface you have defined on the screen for you.

This assignment is to be done individually.

Principle of operation

A parametric surface is defined as the image of a domain D under a function f. For 2D surfaces in 3D space, $f : \mathbb{R}^2 \to \mathbb{R}^3$, and for the purposes this programming assignment $D = [0, 1] \times [0, 1]$; that is, the domain is the unit square. Your mission is to define several different functions f that result in particular 3D shapes.

For each shape, you implement a function that takes a point $(u, v) \in D$ as input and returns a 3D point $\mathbf{x} = f(u, v)$ (on the surface) and a 3D vector $\mathbf{n} = f^{\perp}(u, v)$ (the normal to the surface at that point) as output. You will use various trigonometric and other functions to create the mapping f, and you should compute the normal by taking the cross product of the two partial derivatives of f:

$$\mathbf{n} = f'_u(u, v) \times f'_v(u, v).$$

Here I am using the notation f'_u to denote the derivative of the (vector valued) function f with respect to the variable u. Writing this out by components, if we let

$$f(u, v) = \begin{bmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{bmatrix}$$

then

$$f'_u(u, v) = \begin{bmatrix} \frac{\partial x}{\partial u}(u, v) \\ \frac{\partial y}{\partial u}(u, v) \\ \frac{\partial z}{\partial u}(u, v) \end{bmatrix} \qquad f'_v(u, v) = \begin{bmatrix} \frac{\partial x}{\partial v}(u, v) \\ \frac{\partial y}{\partial v}(u, v) \\ \frac{\partial z}{\partial v}(u, v) \end{bmatrix}$$

Since the cross product follows the right-hand rule, this means that if we are looking at the front (outside) of a surface and the u coordinate is increasing to the right, then the v coordinate is increasing as you go up and the normal is pointing towards the viewer. Getting the normals facing the right way is one of the subtler parts of this assignment.

Framework code

For this assignment we are providing a framework that draws a 3D view of the surfaces that you define and provides some handy tools for understanding the behavior of your mapping. The framework code is mainly concerned with setting up the user interface and drawing the 3D view of the surface. It does this by calling your function many times to find out the 3D locations of points in the domain and the normals that go with them. The surface itself is actually drawn using a few thousand 3D quadrilaterals, and a grid is drawn over that to help you visualize the parameterization.

The main class of the program, `Param`, maintains a list of objects of class `Parametrization` – one for each type of parameterization that's been implemented. Each object has a set of parameters associated with it, and they are displayed for the user to edit when the "$f(u, v)$" tab is selected.

Your mission, then, is to create several classes that extend `Parametrization`. The main thing is to override the method `Parametrization.evaluate`, which takes a 2D point `vin`, which is (u, v) in the discussion above, and returns a 3D point `vout`, which is $\mathbf{x}$, and and a 3D vector `nout`, which is $\mathbf{n}$ (note that it returns these values by modifying the parameters `vout` and `nout` rather than by using a return value).

Each class also needs to define a constructor that sets up the parameter list for the map. As an example, we've provided the class `SquareMap`, which defines the simple mapping

$$f(u, v) = \begin{bmatrix} s_x * (u - 0.5) \\ 0 \\ -s_z * (v - 0.5) \end{bmatrix}.$$

The parameters s_x and s_z are set up in the constructor with the names `xscale` and `zscale`. Based on the simple example given by this class it should be easy to set up your own classes with as many parameters as needed.

The one other thing you need to do is change the method `Param.registerMaps`, which builds the list of maps that is presented to the user in the pop-up menu. You just need to duplicate the call to `mapRegistry.add` and edit it to reflect the name and class of your mapping.

Requirements

Your program must implement the following:

1. A spheroid. This is simply a sphere that is stretched by s_x, s_y, and s_z along the x, y, and z axes respectively. Use a parameterization similar to the latitude-longitude parameterization we discussed in class but arrange things so that $v = 0$ is the $-y$ pole, $v = 1$ is the $+y$ pole, and $v = 0.5$ is the equator. The point $(u, v) = (0, 0.5)$ maps to $(s_x, 0, 0)$. The orientation conventions described above determine which way u should wrap around the equator.

   This surface has three parameters: s_x, s_y, and s_z, all of which should default to 1.0. The front of the surface should face out.

2. A torus. This is a doughnut or bagel shape that is formed by sweeping a circle around an axis in the plane of the circle but outside the circle. It is like a piece of tubing that is bent into a circle and joined seamlessly to itself. The symmetry axis should be the y axis. The u coordinate should run along the tube, increasing in the counterclockwise direction to a viewer positioned on the $+y$ axis. The v coordinate should run around the tube in the appropriate direction to maintain orientation. The point where $u = v = 0$ should be the extreme point on the x axis.

   The parameters are the outer radius r_o (the distance from the y axis to the center of the tube) and the inner radius r_i (the radius of the tube). The front of the surface should face out.

3. The graph of the function

   $$y = s_y \cos(s_r r) e^{-r^2/(2\sigma^2)}$$

   Where r is the distance from the center of the unit square in (u, v) space. The surface should be a height field above the x-z plane, and it looks like ripples on the surface of a pond. The coordinate x should depend on u and z should depend on v, and you should arrange for the front of the surface to face $+y$ and for the range of x and z to be controlled by the scale factors s_x and s_z and to be centered on zero. That is, if $s_x = 5$ then x runs from -2.5 to 2.5.

   The parameters are s_x, s_y, s_z, s_r, and σ.

4. One of the two following surfaces (or both for extra credit):

   (a) A helicoid. This is a surface that you get by starting with a helix and connecting each point to the axis of the helix by a straight line.

   The central axis should be the y axis, and the radius of the helix should be specified by a parameter r. The extent of the surface along the y axis should be s_y, and the helicoid should make n complete turns in that space (where s_y and n are parameters), starting and ending aligned with the x axis if n is an integer.

The front of the surface should face $+y$ when $n > 0$. You should arrange for v to increase along the sprial in the same direction as y and for u to increase from $u = 0$ at the axis to $u = 1$ at the outer edge.

(b) A rippled torus. Continuing the pastry analogy, this might be known as a "cruller" surface. This is just like the torus except that the cross section takes the form $r_i(1 + a\cos n\theta)$ (in polar coordinates centered on the tube axis) and this cross section rotates m times as it sweeps along the torus.

The parameters are r_i and r_o from the basic torus, plus the acircularity parameter a, the number-of-ridges parameter n, and the number-of-twists parameter m. The parameters n and m are supposed to be integers; your code should not break if they are non-integer, but it is OK to produce a surface with cracks in that case. The front of the surface should face out.

Be sure to pay attention to the orientation of the surface. You can tell if you are looking at the front or the back because the front renders as a shiny blue material and the back is a duller gray. Also, if your normals are inconsistent with the orientation of u and v then the surface will appear lit improperly.

Discussion questions

Answer the following questions in a comment in `Param.java` or on paper, to be turned in in class the same day the program is due:

- What is the normal to the sphere at each of the poles? What is the normal that's computed by the method we recommend?
- If you implemented the helicoid, how does the magnitude of your normal compare between points near the axis and points near the edge? Why is this?
- If you implemented the rippled torus, do the u parameter lines twist along with the torus? Why is this? How could you have implemented it to make this answer come out the other way?

Handing in

You will hand in your source code using CMS (the Course Management System), linked on the CS465 web site. The file you hand in should be a ZIP archive containing all the `.java` files and images that are needed to compile and run the program (that is, the framework plus the `.java` files for your new classes). Please don't include extra files, for instance `.class` files or extra source files that don't compile. With the class path set up to include GL4Java and the vecmath library, one should be able to compile and run your program from the command line using

```
javac *.java
java Param
```

Unless you are doing extra credit you should not modify the framework.

Extra credit

If you are enthusiastic about the project, we always encourage you to implement extra features for extra credit, subject to the ground rules described on the course web page. For this program appropriate extra credit might be extra parameterizations that produce nifty shapes that are at least as complicated to implement as the most difficult required ones.

One idea would be a mapping that takes as its parameter a scalar-valued formula in x and y, parses it (allowing arithmetic, exponents, and basic trig functions), and produces the graph of the function as a surface.

We recommend talking to us about your proposed extra credit first so we can steer you towards interesting mappings and make sure we agree that it would be worth extra credit.

Let us re-emphasize that, as always, extra credit is only for programs that correctly implement the basic requirements.