HOMEWORK #6

SURFACES OF REVOLUTION FROM BÉZIER CURVES

This assignment may be done in teams of two. Due Dec. 9, 2009 by 11:59PM. The turnin name is cs433_hw6.

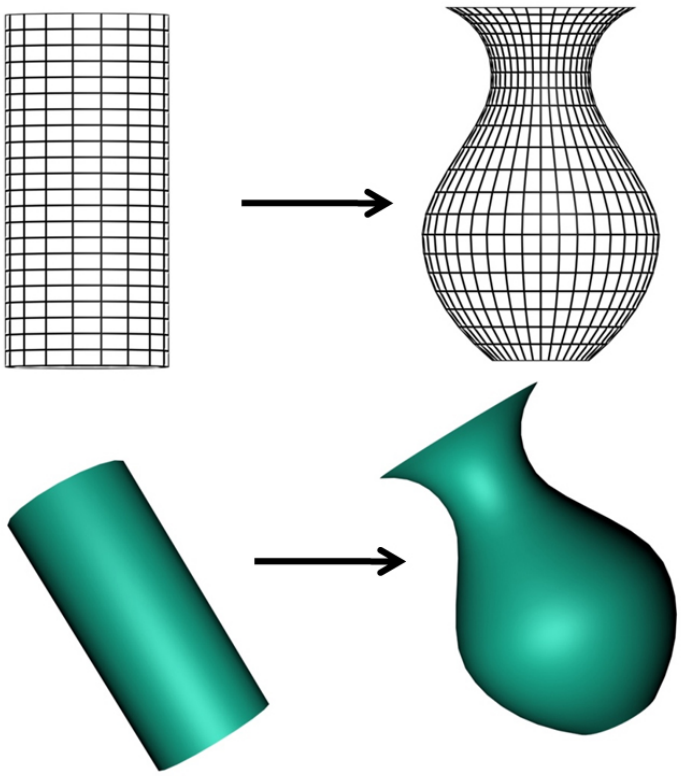

Figure 1. Applying a Bézier curve to morph a cylindrical mesh.

In this assignment you'll define a surface of revolution using a Bézier curve and implement a shader program that applies this curve to deform a mesh interactively. Such techniques have practical applications in real time graphics for surface morphing. You should begin by extending the viewing framework and shader programs that you built for Homeworks 4 and 5. In the bottom viewport, instead of showing a polyline, you'll show a Bézier curve, whose control points can be manipulated interactively by the user. In the top viewport, instead of showing a mobile, you'll present a surface of revolution that is formed by rotating this Bézier curve about an axis. The requirements of each component are outlined further below:

(1) **Bézier curve:** (30 pts) Let the bottom window visualize a third-order curve as defined by the Bézier basis functions. Your curve should have 4 control points, and you should visualize both the control points and the curve in the lower viewport, as before.

Your program should allow the user to manipulate the control points interactively. This is similar to the previous assignment, but with extension. The user now requires 2 degrees of translational freedom to move the control points, so you'll need to modify your key mapping from the previous assignment as follows: let the arrow keys be used to move each control point in space, with the up and down arrow keys corresponding to vertical axis (y) movement, and the left and right arrow keys corresponding to like motion of the control points along the horizontal axis (x). Allow switching between control points using the $<$ and $>$ keys.

(2) **Surface of revolution:** (40 pts) Let the top viewport visualize a 3D surface of revolution that is generated from the Bézier curve displayed in the bottom viewport. Implement a *vertex shader* to displace the vertices of a mesh according to the Bézier curve.

Conceptually, you're generating a surface of revolution by rotating m copies of the curve about a central axis of revolution. You can define a triangulation to generate a mesh from m curves as follows: Let $v_1 \dots v_n$ be the vertices of the ith copy of the curve, and $v'_1 \dots v'_n$ be the vertices of the next copy. Connect v_i to v'_i and connect v_i to $v'_i + 1$.

Practically speaking, a vertex shader cannot be used directly to add vertices to a mesh or to change vertex connectivity within that mesh. Vertex shaders can only modify vertex attributes, such as position, normals, and color. Thus you'll need to create a simple mesh that you can then modify in the vertex shader by displacing its vertices according to the parameterized Bézier curve. Hint: consider first creating a simple mesh that approximates a cylinder, and deform it accordingly. See Figure 1. A few additional notes that you should consider:

- You should specify your mesh with sufficient vertex density that it is an aesthetically pleasing approximation to the curve. Consider using at least 100 vertices along the length of the mesh for each Bézier and duplicating the curve every 5 degrees (corresponding to $m = 72$ above). Feel free to experiment.
- Animate the mesh to rotate about the viewing axis clockwise and counterclockwise, using the + and - keys to increase and decrease the rate of rotation.
- You should make an effort to *prevent* your surface of revolution from self-intersecting.
- Note: there is no bump mapping in this assignment; the Bézier curve is now being used to alter the actual surface geometry itself. Similarly, you may discard all the keyboard functionality that was previously associated with mobile manipulation.

(3) **Per-pixel shading:** (30 pts) You should implement a fragment shader to compute the shading of your surface at each pixel. Minimally, your shader should consider the diffuse and specular components of the standard Phong or Phong-Blinn local illumination model. You may assume that the light is at the viewpoint, as before. Grads: your functionality to move the light in a plane should still be in effect.

You should be able to reuse most of your fragment shader code from HW5, with minor modification. Note that texture and bump mapping are *not* required; you can simply assume your surface is one uniform color and apply shading accordingly. Please choose some pleasing color for your surface - nothing bright pastel and, obviously, nothing very dark!

For correct shading, you will need to compute the normal for each vertex in your mesh, based on the current geometry of your surface of revolution as defined by the Bézier curve. Also remember that the surface is being deformed dynamically by the user and is rotating in space.

BONUS POINTS: You may implement the following features for up to 50 points bonus. These points will then supplement your composite homework score:

(1) **Output in standard file format** (Bonus 20 pts): When the user presses the 's' key, your program should save the current mesh to a file. Your file must be readable by Geomview (www.geomview.org), a popular 3D mesh viewing program that provides many useful interface and viewing capabilities.

Your program must save your 3D mesh (and relevant information information, such as color and normals) in a standardized file format, such as VRML 2.0. The specific choice of file format is entirely up to you, but it *must* be one supported by Geomview. You should consult external sources for details on the particular format you choose. Please state in your readme file the format you use and how to run it; if the grader has difficulty reading the file, he may contact you for a demo.

(2) **Generate texture coordinates** (Bonus 30 pts): Your program should map an image onto the deformed surface, synonymous to how a physical "label" is affixed to a bottle. To do this, you'll need to dynamically generate the appropriate texture mapping (UV) coordinates.

You should adjust the UV mapping each time the surface is changed by the user. Your mapping solution should attempt to preserve the area of your "label" image as the surface is deformed. This will require computing relative distances along the surface; your solution may use a *linear approximation* of the curve rather than finding lengths on the Bézier curve directly.

Finally, you should consider the texture color at the current pixel in your lighting equation. This is exactly to how you implemented shading in HW5, where the current texel color is simply multiplied by the diffuse term in the lighting equation.