SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

50.017 Graphics and Visualization

Assignment 2 – Surface Modeler

Handout date: 2020.06.03

Submission deadline: 2020.06.10, 11:59 pm

Late submissions are not accepted

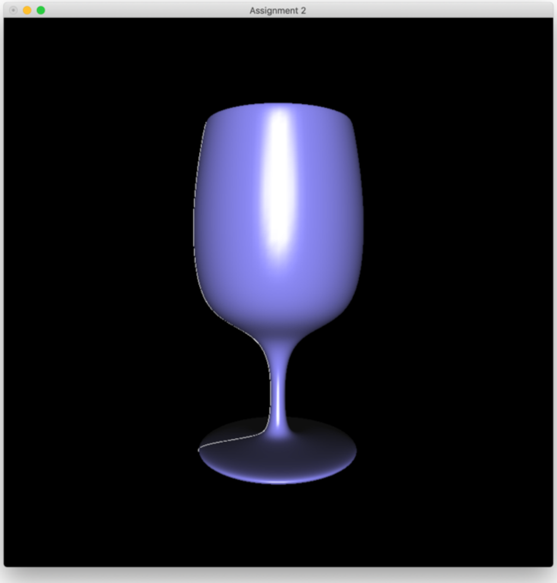

Figure 1. Expected program output by constructing surface of revolution from wineglass.swp.

In this assignment, you will be implementing swept surfaces to model interesting shapes. The primary goal of this assignment is to introduce you to how to model swept surfaces (in particular, surfaces of revolution) based on given profile curves. “TODO” comments have been inserted in surf.cpp to indicate where you need to add your implementations.

1. Overview of Starter Code

1.1 File Input

Your program must read in a specific file format (SWP) that allows users to specify curves (Bezier or B-Spline). Four SWP files are provided in the Data subdirectory. The specification of the file format is given in the header file of the provided parser (parse.h).

Reading a SWP file has been fully implemented in the starter code (main.cpp):

int LoadInput_SWP()

1.2 Initialize Curve

After reading a SWP file, the starter code will initialize a curve (Bezier or B-Spline). A Curve is represented as a vector of CurvePoint. And each CurvePoint has four variables:

V: vertex position

T: unit tangent vector, a unit vector that is tangent to the curve at point **V**.

B: unit binormal vector, a vector orthogonal to **T**, and we arbitrarily select it to be in pointing in the positive z-direction.

N: unit normal vector, **N** = **B** x **T**.

Please refer to curve.h and curve.cpp for the Curve data structure and how a curve is initialized from the data stored in a SWP file.

1.3 User Interface

After running the starter code, it should prompt you to enter filename .swp. Once you input an SWP file name such as wineglass.swp, the starter code should show a window that renders the curve like the one in Figure 2.

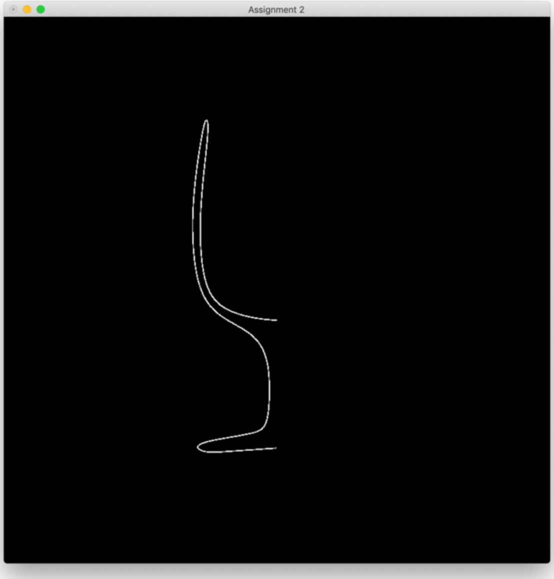

Figure 2. Curve of wineglass.swp

You should be able to interact with the curve in a similar way to Assignment 1:

- Left mouse drag will rotate the object around a mapped axis based on the mouse motion.
- Left mouse drag + holding shift key will scale the object to make it either smaller or bigger depending on the mouse moving direction.
- You can reset view of the model by pressing the r key.
- You can show or hide the curve by pressing the c key.

2. Modeling Surfaces of Revolution

In this assignment, you will be using the curve loaded from a SWP file to create swept surfaces, specifically, surfaces of revolution. Figure 1 and 2 show an example surface of revolution and the corresponding profile curve. Figure 2 is the profile curve on the xy-plane, and Figure 1 is the result of sweeping the surface about the y-axis.

2.1 Construct Surface

For this assignment, we define a surface of revolution as the product of sweeping a curve on the xy-plane counter-clockwise around the positive y-axis. The specific direction of the revolution will be important.

Surface Vertices. Suppose that you have already initialized the profile curve. Then, you can generate the vertices of the surface by simply duplicating the initialized curve points at evenly sampled values of rotation. This should be your first task during the implementation process.

However, vertices alone do not define a surface. As we saw from the previous assignment, we also need to define normals and faces. This is where things get a little more challenging.

Surface Normals. Well, we already have normal vectors **N** from the initialization of the curve. So, we can just rotate these normal vectors using the same transformation as we used for the vertices, right? Yes, and no. It turns out that, if we transform a vertex by a homogeneous transformation matrix **M**, its normal should be transformed by the inverse transpose of the top-left 3 × 3 submatrix of **M**. A discussion of why this is the case appears in this wiki page: https://en.wikipedia.org/wiki/Normal_(geometry) (see Transforming Normals section). You can take comfort in the fact that the inverse transpose of a rotation matrix is itself (since rotation is a rigid transformation).

Another thing that you'll need to worry about is the orientation of the normals. For OpenGL to perform proper lighting calculations, the normals need to be facing out of the surface. So, you can't just blindly rotate the normals of any curve and expect things to work.

Surface Faces. So far, we have ignored the faces. Your task will be to generate triangles that connect each repetition of the profile curve, as shown in Figure 3. The basic strategy is to zigzag back and forth between adjacent repetitions to build the triangles.

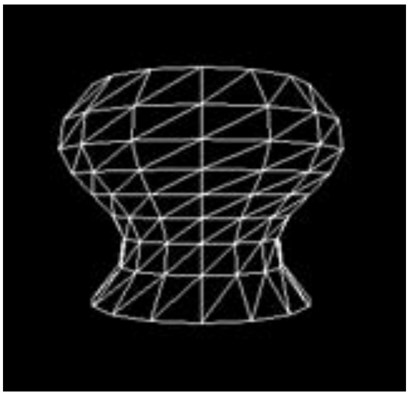

Figure 3. Generate triangular faces from the rotated profile curves.

In OpenGL, you are required to specify the vertices in a specific order. They must form a triangle in counter-clockwise order (assuming that you are looking at the front of the triangle). If you generate your triangle faces backwards, your triangles will have incorrect lighting calculations, or even worse, not appear at all. So, when you are generating these triangle meshes, keep this in mind. In particular, this is where our assumption of counter-clockwise rotation about the y-axis comes in handy.

Important: Constructing surface should be implemented in the following function (in surf.cpp):

Surface makeSurfRev(const Curve &profile, unsigned steps)

2.2 Draw Surface

To demonstrate that the surface is correctly modelled in your program, you should render the surface with OpenGL. First, you should render the surface with smooth shading like Figure 1. Second, you should render the surface together with the vertex normals drawn pointing outwards from the surface like Figure 4.

In the program, you can show or hide the smooth surface by pressing the s key. You can also show or hide the vertex normals by pressing the n key. Note that this keyboard interaction has been implemented in the starter code.

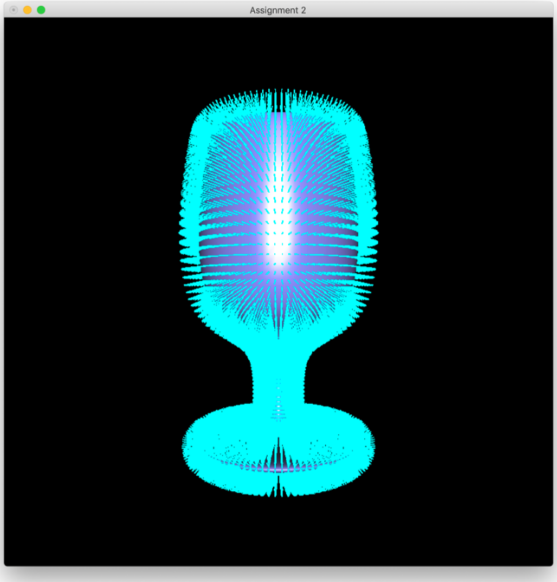

Figure 4. Render surface of revolution of wineglass.swp together with vertex normals.

Important: Rendering smooth surface and vertex normals should be implemented in the following two functions respectively (in surf.cpp):

void drawSurface(const Surface &surface)

void drawNormals(const Surface &surface, float len)

3. Grading

Each part of this assignment is weighted as follows:

- Construct Surface: 70% in total
  - Surface vertices: 20%
  - Surface normals: 30%
  - Surface faces: 20%
- Draw Surface: 30% in total
  - Draw smooth surface: 15%
  - Draw vertex normals: 15%

4. Submission

A .zip compressed file renamed to AssignmentN_Name_I.zip, where N is the number of the current assignment, Name is your first name, and I is the number of your student ID. It should contain only:

- The **source code** project folder (the entire thing).
- A **readme.txt file** containing a description of how you solved each part of the assignment (use the same titles) and whatever problems you encountered. If you know there are bugs in your code, please provide a list of them, and describe what do you think caused it if possible. This is very important as we can give you partial credit if you help us understand your code better.
- A couple of **screenshots** clearly showing that you can display the constructed surfaces of revolution with and without vertex normals.

Upload your zipped assignment to e-dimension. Late submissions receive 0 points!