C481 B581 Computer Graphics
Dana Vrajitoru

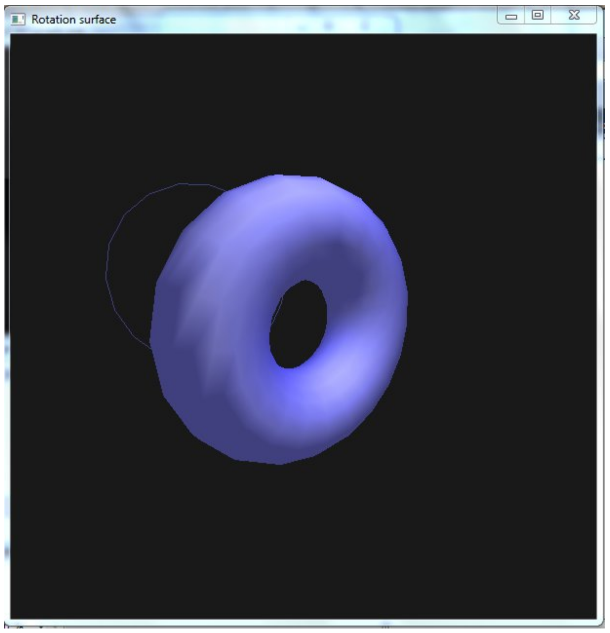
C481/B581 Homework 8

Due date: Monday, March 27.

In this surface we'll be generating rotation surfaces together with its normals and render them using material properties and a light source.

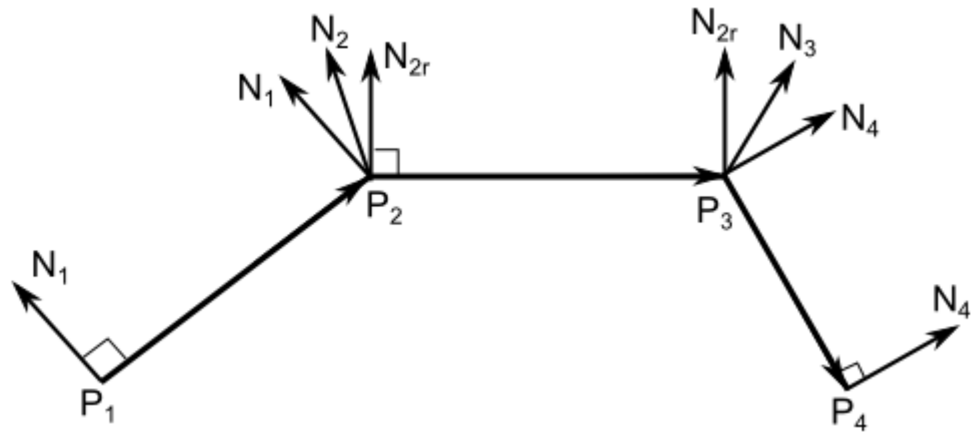
Ex. 1 Download the following files in a dedicated folder.
Point3f.h
Point3f.cc
interface.h
interface.cc
main.cc
surface.cc
surface.h
trackball.h
trackball.cc
glheader.h -> new version!
Makefile

Compile the program with the command `make` and run it with the command `surf`. For now, this program draws a circle and a torus with some material effects. Here is a snapshot of the application window:



a. Implement the function `computeNormalsXy` in `surface.cc`. This function must compute the normals for a curve in the xy plane. You can use the function `orthogonalXy` from the `Point3f` class (see the reference).

In addition to calling this function for each individual segment (made of two points in the sequence), for each 2 adjacent line segments, the normal should be the average of the two normals computed on each segment. For example, in the image below, `N1` is obtained by calling `orthogonalXy` on the vector `P2 - P1`. Then `N2r` is obtained in a similar way. Finally, `N2 = 0.5*(N1+N2r)`.



If the first and last point are the same, then the normal in the first and last points should be averaged between the first and the last segment.

You can assume in this function that the vector `normals` is initially empty, and use the function `push_back` to add values to it.

b. Implement the function `rotationSurfaceY` which takes as input a curve with normals in the xy plane and builds a rotation surface by rotating this curve around the Oy axis. The first two parameters represent the input, and the last two the output. If the curve is defined by $(x_1(s), y_1(s))$ (corresponding to (`curvePt[i][0], curvePt[i][1]`) in the code), then the surface is defined by

$x(s, t) = x_1(s) \cos(t)$
$y(s, t) = y_1(s)$
$z(s, t) = x_1(s) \sin(t)$

The normals are computed by applying the same rotation to the corresponding normal vector to the curve (`curveNm[i]`) in the given point $(x_1(s), y_1(s))$. For a closed surface, the parameter `t` goes from 0 to 2pi in increments similar to the function `circleXy`. Note that the math functions `sin` and `cos` take an angle in radians as parameter.

You can assume in this function that the initial output parameters are empty, and use the function `push_back` to add values to them. For a `vector3f`, adding a value is a simple call to this function. For a `matrix3f`, though, you have to add an empty `vector3f` for each new row, then you can add elements to that row.

c. Implement the function `drawSurface` that takes in a grid of points and normals and must draw the surface they define as a sequence of triangle strips (see class notes). You must also use the normals in this function so that the surface looks smooth. For every coordinate in cell `[i, j]` that you call the vertex function for, you must call the corresponding normal just before, like

```
glNormal3v(surfaceNm[i][j]);
glVertex3v(surfacePt[i][j]);
```

2. Replace the circle and `glut` torus in the display (look at the `createDisplayList` function in `interface.cc`) with at least two objects created as the rotation surfaces, one of which must be a torus, and the other something else (whatever you want). You can use the existing circle as the cross section curve to create a torus.

For the second object, you would start by defining an array of coordinates from the curve, and store it in a `vector3f` variable. You can use an equation for this, or explicit values. Then you have to call the function `computeNormalsXy` to generate the normals array for this surface. After that you have to call the function `rotationSurfaceY` to generate the surface.

Add another material function with a different color and shininess, and apply different material properties to the two objects.

Upload to Canvas: all the files that you modified/added.