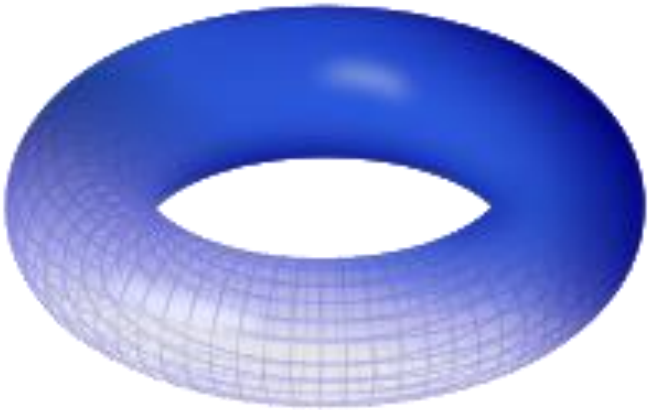
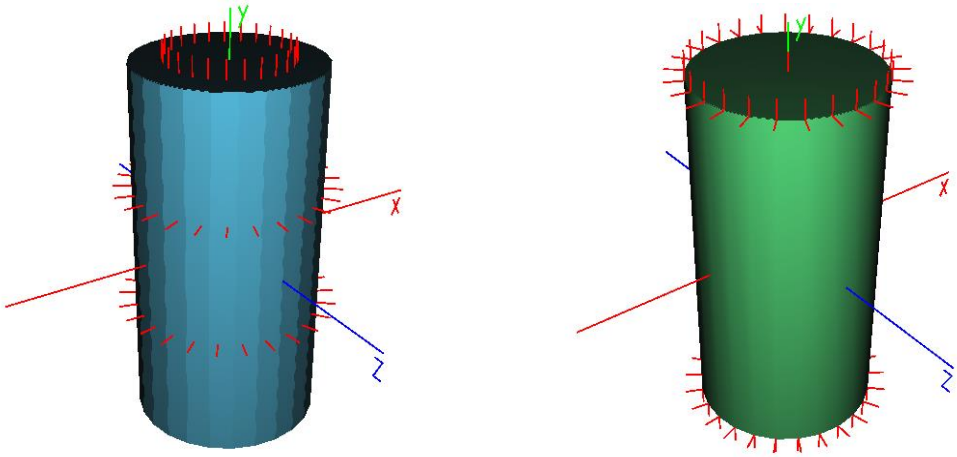
## Programming Assignment #3

In this assignment you will generate normal vectors for your torus primitive object in order to render it shaded:



You will display your torus in two modes: smooth and flat shading, and will display the normal vectors as well. As an example, see below how to display the normal vectors for a cylinder primitive (but do not forget you will be working on a torus primitive instead):




**First follow these steps:**

1) Download **PA3_support_code.zip**, which contains just the two shaders (*persplight.vert* and *persplight.frag*) you will be using to render with lighting. Place both into the shaders folder inside your project. You will have to write code to create and initialize them in the same way as the existing shaders.

2) You may inspect the shaders, particularly the fragment shader, to see how they implement the Phong illumination model that we saw in class. To keep things simple, most of the parameters are hard-coded, such as light position and intensity, material properties, and

shininess. The only new information you will have to supply these shaders is an array of normals.

3) When you are ready, implement the requirements below.

## Requirements:

### Requirement 1 (30%): Calculate normals and display your shaded torus object.

You will re-use your code from the previous assignment where you created a torus primitive, but you will now display the torus in both smooth and flat shading. As before with vertex coordinates and colors, you will need to create an additional array to store normals. Remember to create the necessary VBO and transfer the array of normals to the GPU after it is created, and to activate the new shader when rendering your torus.

We are keeping things simple by making all arrays the same size. This means we provide 1 set of coordinates, color, and normal for every vertex of the object, and they are all vec4's (or arrays with 4 elements).

**Flat Shading:**

For flat shading, for each triangle, all 3 vertices have the same normal vector, which is the normal vector of the face. Remember that you can calculate a perpendicular vector using the cross product (see `glm::cross()`), and the two vectors you use for the cross product can be calculated from the 3 vertices of the triangle. The order matters: if you use the wrong order, your normal will point inside of the object.

The left (blue) cylinder in the first page is an example of flat shading.

**Smooth Shading:**

For smooth shading, each vertex has its own normal vector, which is the normal vector to the torus surface at that point **v**. It is straightforward to compute the normal vector from the parameters of the torus: it will simply be the normalized vector from the circular axis of the torus to **v**. In other words, for any particular "slice" of the torus, given its center point **c**, the normal is simply **v** - **c**.

The right (green) cylinder in the first page is an example of smooth shading.

### Requirement 2 (30%): Visualization of normals.

You will also need to draw line segments representing each normal vector that you calculate. As you will be drawing line segments here instead of triangles, you will need to create new structures to store everything, instead of adding on to your torus structures.

You should draw the normal vectors in the following way:

- Smooth shading: segments will originate from each vertex and point outwards,
- Flat shading: segments will originate from the center of each triangle and point outwards. (See figures at the beginning of this document for examples.)

For flat shading, the reason for the normal vectors to originate from the center of each triangle is to indicate that all vertices of a triangle use the same normal vector, while for smooth shading each vertex uses a different normal vector.

Visualizing the normals is a good way to be sure you are calculating them correctly. If the object's shading looks wrong, the problem is likely your normals.

**Requirement 3 (30%): Controls.**

In order to demonstrate that your object is correct, include the usual controls for changing the resolution of your torus (10%):

- '**q**' : increment the number of triangles
- '**a**' : decrement the number of triangles
- '**w**' : increment the *r* radius (by a small value)
- '**s**' : decrement the *r* radius (by a small value)
- '**e**' : increment the *R* radius (by a small value)
- '**d**' : decrement the *R* radius (by a small value)

Then, add the following controls (20%):

- '**z**' : to enable Flat Shading
- '**x**' : to enable Smooth Shading
- '**c**' : show/do not show normal vectors (toggle)

**Requirement 4 (10%): Overall quality.**

Everything counts here and, once again, there is no need to do anything complex. Just make sure your project works and looks good and you will get full points.

**Submission:**

Please follow the instructions in parules.txt (uploaded to CatCourses). In particular: please do not include any third-party support code and do not forget to Clean Solution before preparing your project for submission! Also, check for hidden folders (such as .vs) which can sometimes balloon to hundreds of megabytes!