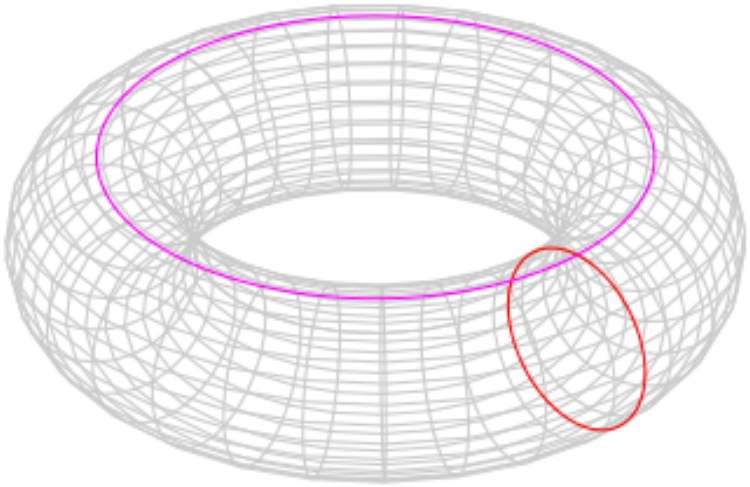
Programming Assignment #2

In this assignment you will create a 3D object from its parametric equation. The object will be a torus. In “line rendering” a torus looks like the following:



Read the text in https://en.wikipedia.org/wiki/Torus (the image above came from this link) and study the parametric equation of the surface of the torus:

$$x(\theta, \varphi) = (R + r\cos\theta)\cos\varphi$$
$$y(\theta, \varphi) = (R + r\cos\theta)\sin\varphi$$
$$z(\theta, \varphi) = r\sin\theta$$

With the above equation it is straightforward to generate triangles around the surface of the torus: you will simply vary the parameters of the equation by constant intervals in order to get points in the surface and connect them forming triangles. You will build a parametric torus object based on parameters *r* and *R*, and also controlling the resolution (number of triangles) of your approximation.

IMPORTANT: You will be using your torus in the next assignment(s) as well, so make sure to implement it well!

First follow these steps:

1) You can start with your code from PA1 or start from scratch with PA1's support code again.

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

Requirements:

Requirement 1 (60%): Parameterized torus using triangle primitives.

You will use the torus parametric equation to build a torus based on 3 parameters: *r*, *R*, and *n*, where *n* controls the number of triangles that are generated.

The number of triangles does not need to be parameterized exactly. Parameter *n* just needs to control the resolution in a way to properly control a finer or coarser subdivision of the surface in triangles. You will typically have two nested for loops to vary the 2 parameters (*r* and *R*) of the parametric equation, and *n* will control how large are the increments applied to the two parameters of the for loops. As triangles are created, "push" them to an array to be sent to OpenGL. You will likely want to use a dynamically sized structure like a vector.

Note 1: Remember, for the built-in OpenGL "back-face culling" to work properly, make sure the order of the vertices in each triangle is always counter-clockwise, when seen from outside the torus. This is almost always an important thing to get right in Computer Graphics.

Note 2: It will probably be better to inspect your torus in wireframe since at this point your object will not be shaded. We will cover shading and illumination in a future assignment. By default in the support code, pressing 'w' will toggle wireframe (or "line rendering") mode.

Requirement 2 (25%): Interaction.

In this requirement the previously described parameters are controlled with keys such that you can test your implementation and generate shapes with different properties. Use the callback functions to change the values of your parameters in response to key presses, and then regenerate the torus with the new parameters. Please use the following keys:

- '**q**' : increment the number of triangles (*n*)
- '**a**' : decrement the number of triangles (*n*)
- '**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)

Requirement 3 (15%): Overall quality.

Everything counts here: if requirements are well implemented, creativity, source code organization, etc. There is no need to do anything complex, just make sure your project looks good and you will get the full points here!

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!