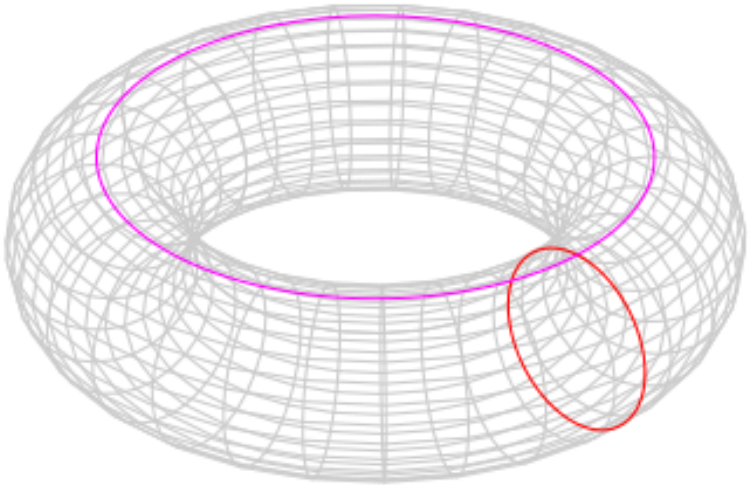
Programming Assignment #2

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 start from the new support code, which includes shaders and shader-loading code as well as a variety of functionality. You will then build a parametric torus object based on parameters *r* and *R*, and also controlling the resolution (number of triangles) of your approximation.

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

First follow these steps:

1) Download **PA2_support_code.zip** and **glew.zip** (which is required for a lot of the new OpenGL commands we will be using) from CatCourses and unzip them in the **same folder that has freeglut and glm**. It is important that they are in the same folder, as the support code solution is linked with relative paths.

2) Double-clicking on OpenGLProject.sln should open Visual Studio. If you press F5 or Ctrl+F5 you should see the project compile and run without any issues. However, a variety of problems may occur: be sure to have the latest drivers for your graphics card, and if you are running a different version of Visual Studio you may need to retarget the project. Your TA will be helping with these steps. Go to lab as that is the best way to get help with this! *(Note that the included version of glew will only work on Release build.)*

3) When you are able to run the application, you will see three rgb lines along the x, y, and z axes. Clicking and dragging will rotate the scene and you can use the mouse wheel to "zoom" in and out, which should help you to visualize what you create.

The support code includes new files to help with shader creation, loading, and use, and two basic shaders. There is already code to create the model, view, and projection matrices and send them to the perspective shader. It also shows how to create the necessary buffers and arrays of vertex information to render the lines for the x, y, and z axes, which serves as a useful example for your torus.

Take some time to familiarize yourself with the application and the new code and try to understand everything that is going on. If there is anything you do not understand, refer to the lecture slides or any one of the other OpenGL resources suggested, or ask your TA for help.

4) 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, just "push" them to an array to be sent to OpenGL.

Follow what is already implemented in the support code. The example implementation already shows how to organize the coordinates (and colors) of vertices in buffers and how to

make OpenGL calls to send the data to the GPU and for drawing the triangles encoded in the buffers. You will need to do the same, with a new VAO and VBOs for your torus.

Note 1: 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.

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 future assignments. Press ‘w’ to 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)

Consider that you do not need to generate a new VAO and VBOs for your torus when it changes, but merely send the new data to the GPU.

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!