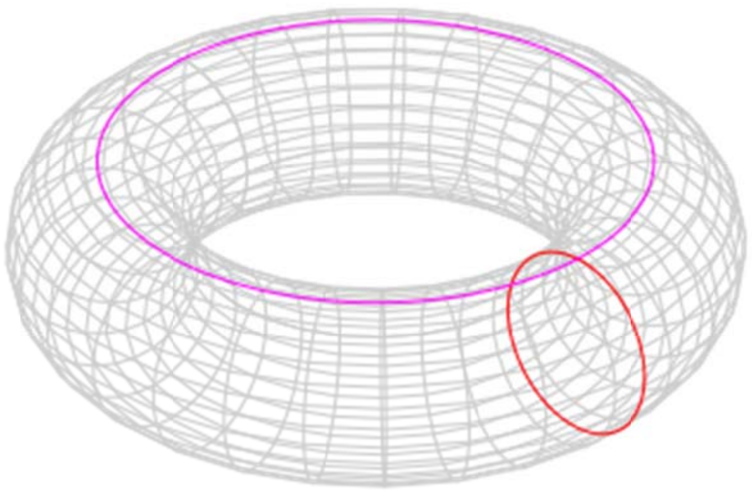
Programing Assignment #1

In this assignment you will create a 3D primitive 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:

$$\begin{aligned}x(\theta,\varphi) &= (R + r\cos\theta)\cos\varphi\\ y(\theta,\varphi) &= (R + r\cos\theta)\sin\varphi\\ z(\theta,\varphi) &= r\sin\theta\end{aligned}$$

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 Visual Studio project **sigmynode**. You will then build a parametric torus "scene node" based on parameters *r* and *R*, and also controlling the resolution (number of triangles) of your approximation.

Your scene node will be used in the next assignment so it will be useful that you rename your node class with something like SnMyTorus for later re-use (although this name is not a requirement).

First follow these steps:

1) Download the two packages **sig** and **sigmynode** from Cat Courses and un7zip them in a **same folder**. It is important that they are in a same folder because sig libraries are linked with relative paths. (Note: the packages in Cat Courses have projects for Visual Studio 2019 so that they should run fine in our computer labs, and the sig package also comes with precompiled libraries and executables.)

2) Enter in sigmynode/vs2019/ and double click on sigmynode.sln. Visual Studio should then start. If you press Ctrl+F5 you should then 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.

3) When you are able to run the application you will see a few rectangles in the screen. Take some time to familiarize yourself with the application: a shift+drag rotates the scene, ctrl+shift+a displays some info, and see also the right-click menu options.

4) Finally, you may open sigfull.sln from inside sig/vs2019/ to study the sig project. The libraries are already compiled but it will be useful to have the project open so that you can read the comments in the header files and see how the many examples provided are implemented. You may as well try to run some of the examples. Also make sure to read file doc/**sigdoc.pdf**.

5) Now return to the "sigmynode" project and study files sn_mynode.h/.cpp and glr_mynode.h/.cpp. These files implement one "scene graph node" in sig with the purpose of drawing triangles using OpenGL commands. The used OpenGL commands are in glr_mynode.cpp. Basically it organizes two triangles (forming a rectangle) in an array that is sent to the graphics card for triangle rasterization. You may use the webpage http://www.opengl.org/sdk/docs/ to read the full description of the used OpenGL commands. Focus on the descriptions of the OpenGL 4.5 version.

While in CSE-170 we will not go deep in using all the OpenGL functionality, if you would like to explore it further, you will be able to develop your final project in a topic of your choice and you may choose to explore OpenGl programming effects and techniques.

Requirement 1 (60%): Parameterized torus using triangle primitives in a scene node.

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 a 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.

Just follow what is already implemented in the support code. It already shows how to organize the coordinates of vertices in buffers and how to make OpenGL calls for drawing the triangles encoded in the buffers. Note that the triangles created in the support code depend on some member variables (init, width, height) declared in sn_mynode.h. Variables like these are needed in order to parameterize your node, and in your case your variables will be *r*, *R*, and *n*.

Scene node creation is organized in a way to separate shape definition (SnMyNode class) from drawing calls (GlrMyNode). In this way the shape is well defined regardless of the used renderer. Glr classes in SIG implement our OpenGL renderer.

Note 1: in order for the built-in OpenGL "back-face culling" to properly work, make sure the order of the vertices in each triangle is always counter-clockwise, when seen from outside the torus. We will explain this in class.

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.

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. Update virtual method `MyViewer::handle_keyboard(const GsEvent &e)` in my_viewer.cpp in order to change the values of your node parameters in response to key presses. You will have to keep a pointer to the node inserted in the scene so that you can change its parameters from the handle_keyboard() method. You should then be able to control and test your object by pressing some keys. Please use the following keys:

- '**q**' : increment the number of faces
- '**a**' : decrement the number of faces
- '**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)

Note: call mynode->touch() to force the node to redraw after a parameter is changed.

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 present and submit your PA according to parules.txt posted ad CatCourses.