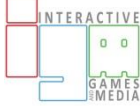
	<p style="text-align: center;"> Rochester Institute of Technology Golisano College of Computing and Information Sciences School of Interactive Games and Media 2145 Golisano Hall – (585) 475-7680 </p>	
---	--	---

Data Structures & Algorithms for Games & Simulation II
IGME 309, 2015 Fall
A4: 3D Shapes

Due: September 13th 2015 at 23:59Hrs.

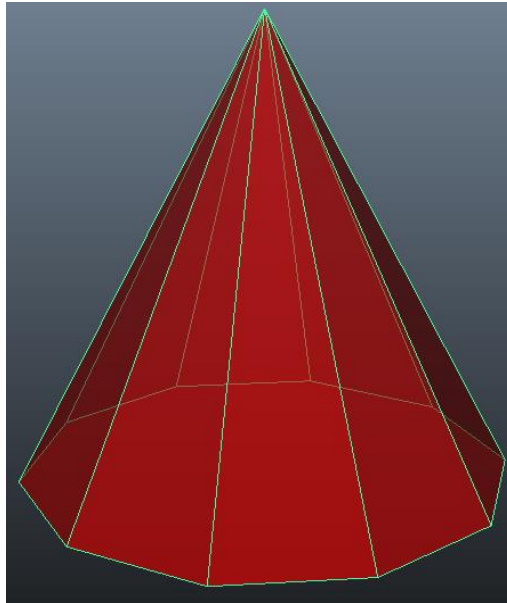
This is not a team-based homework.

Through the semester I will be using a basic engine for rendering shapes and objects called ReEngine (Rendering Engine) to prove data structures and algorithms concepts without the hassle of reinventing the wheel for each lesson. You are not required to use said engine for your assignments as you can use your own existing framework, but it will be your responsibility to translate the provided startup code on your own; most of the time (depending on your framework) it would be as easy as adding the necessary cpp and headers and if necessary change your shader functionality.

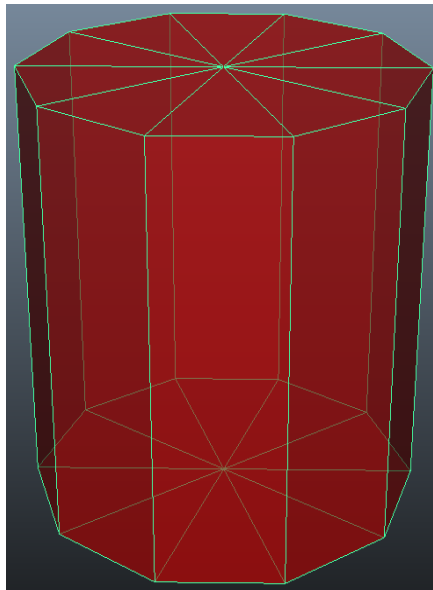
Glut for OpenGL 2.0 had methods that implemented primitive 3D Shapes asking the programmer only to provide the necessary values for the arguments. These shapes where more than often used as placeholders for different assets on the scene. While OpenGL 2.0 is not a standard in the industry anymore (and more than often frowned upon) the idea of generating basic shapes with a single line of code is really useful for debugging purposes.

Your goal for this homework assignment is to implement 4 different methods/functions that create buffers and containers able to hold and display data for different 3D shapes, these methods/functions need to fulfill/implement the following signatures:

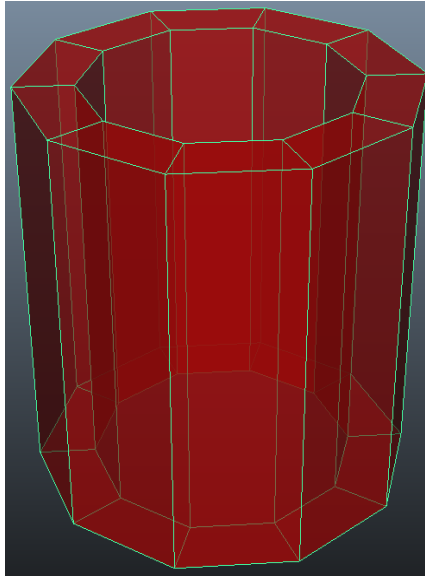
1. Cone(float radius, float height, subdivisions, vector3 color);
 - a. radius of the base of the cone
 - b. height of the cone.
 - c. subdivisions (refer to the cylinder's subdivisions)
 - d. color of the shape



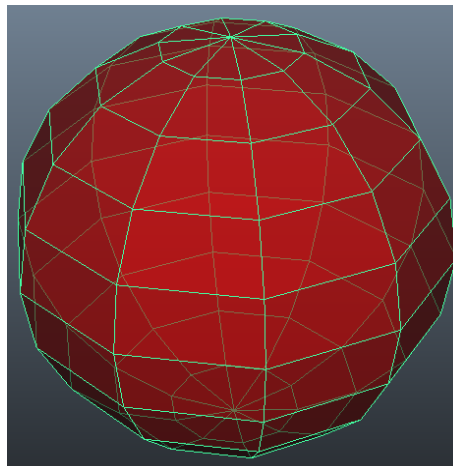
2. Cylinder(float radius, float height, int subdivisions, vector3 color);
 - a. radius of the base of the cylinder.
 - b. height of the cylinder.
 - c. subdivisions is how many sides the base have, if this is 3 the cylinder has a triangular base, if it has 360 the base is a circle, etc.
 - d. color of the shape



3. Tube(float outerRadius, float innerRadius, float height, float subdivisions, vector3 color)
- a. outerRadius is the radius of the outermost part of the tube
 - b. innerRadius is the radius of the innermost part of the tube (the whole if you will)
 - c. height of the tube
 - d. subdivisions (refer to the cylinder's subdivisions)
 - e. color of the shape



4. Sphere(float radius, int subdivisions, vector3 color)
- a. radius, how big your sphere is.
 - b. subdivisions
 - c. color of the shape



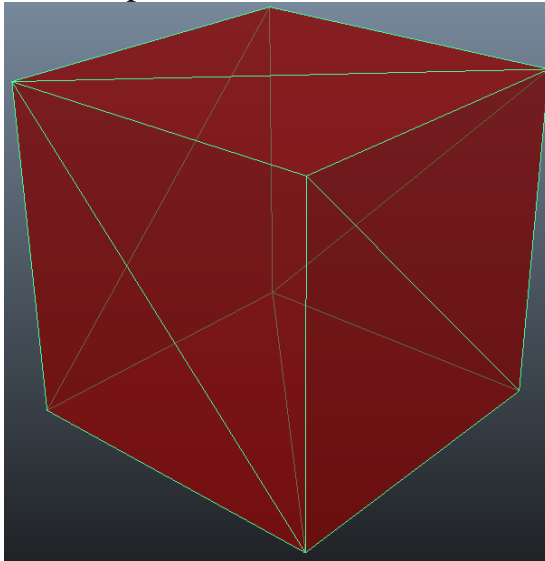
*One of the many ways your sphere can look like, doing it like this is more complicated than the proposed solution below.

In order to make this assignment easier to understand I will provide the functionality of another 3D Shape, the Cube:

```
Cube(float size, vector3 color); // Example code
```

d. size, is the length of each edge in your cube.

e. color, the color of this shape



Startup code is available at: <https://github.com/labigm/ReEngApplication> under the A04_3DShapes project.

Inside the _Binary folder you will find a binary DEMO with an example of the solution.

All shapes need to be:

Centered in the origin so if you were to move them to a different location you can simply provide a model matrix at render time.

Procedural, meaning that even if your shape is generated like you normally expect with the provided example signature but it fails to change the number of sides or size at demand, your solution is wrong.

You may base your solution on the same solution in ReEngine or create a blank solution for this homework (or better yet use your own engine if you have one). I recommend you go for the first option as you would not need to create your own framework, meaning you can skip the programming involved in camera, input, buffer binding, etc. If desired, you may use your own solution, the only condition is that you need to use the ICE1 way (or a similar one) of dividing your code/solution in folders and having all

the intermediary files out of the final submission and that your project is VS2013 compliant.

If you decide to use MyEngine you only need to provide the vertex position for each one of the vertices after calculating it. All your work should be done in MyPrimitiveShapes.cpp, and that should be the only file you need to modify for creating the shapes, on the other hand for displaying them all your work should be done in the AppClass.cpp file as in the provided example. You may have a single shape on the screen or use multiple shapes at the same time (that will require extra variables of course). Before zipping your code please remove all other projects from the solution (from both within visual studio and the folders in file explorer) to make the grader's life easier.

If you decide to use your own solution, you are only allowed to use a single fragment and vertex shader for this homework. Should you need to change a value in the shaders you would do it through uniforms to make the grader's life easier.

Tips:

Cone and Cylinder are the easiest to implement, I recommend you start with them. Tube is similar to Cylinder but requires extra points to calculate. Sphere is the most complex shape in the set and there are various ways to implement it, there are no minimum requirements for it aside that the resulting shape should look like a sphere, if you want to implement a dodecahedron for it that is acceptable, but think about this, if you create a new point at the middle of each pentagon and move that point radius units from the center in the vector that is created from the center to its current location, the shape would look "rounder" and this action can be repeated one more time to make it look more smooth.

Grading:

Each shape is worth 25% of the grade. It is not necessary to have all the shapes working by the end of the assignment, you will receive partial credit for whatever you do, but it's a requirement to at least attempt them, no partial credit will be given for uncompleted methods but it's in your best benefit to explain what do you suspect is your issue with each shape if it is not working.

If your code doesn't compile or compiles and then crashes you will not receive any credit. Test your code in the lab computers as we are going to be using them for grading.

As any other submission your code needs to be in the format specified in the ICE1 with a clean solution (remove all intermediary files and the sdf file). Submissions will not be accepted if they don't comply with this requirement.

All Generate* methods in the provided code will clean up the memory and compile the object for you, there is no need to worry about the memory allocation for this homework assignment unless you are using your own engine/framework. Memory leaks will subtract points from your final grade.

There are some controls already implemented in this solution:

Arrow keys will let you move the primitive around the screen in its local coordinate system.

WASD will let you move in your view vector

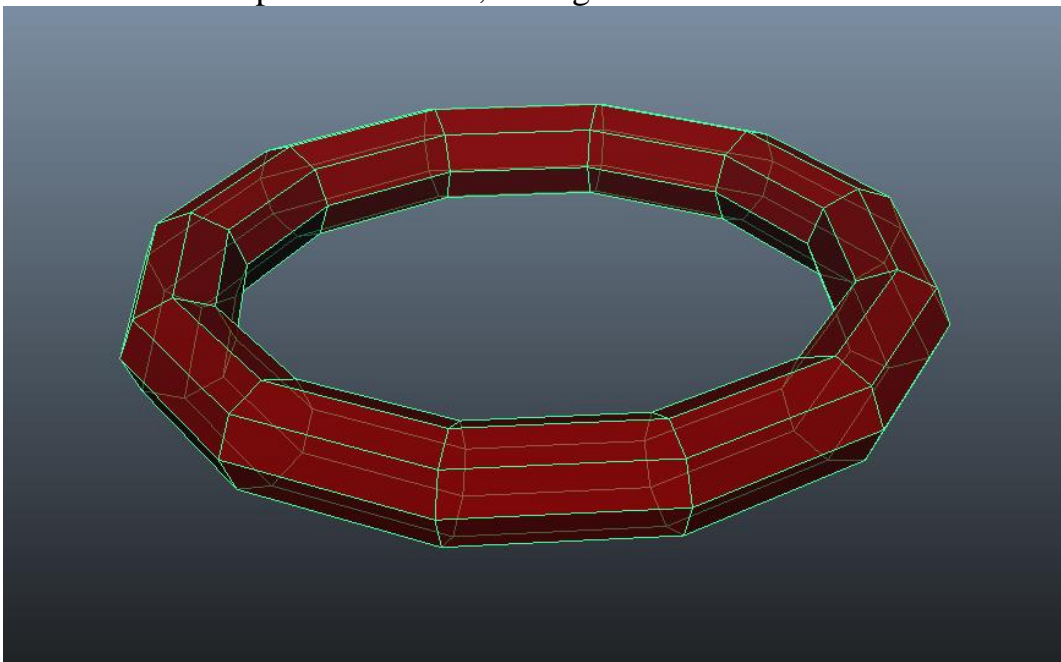
Right click (and hold) will change the angle of the view vector

Middle click and drag will let you rotate the object in the global coordinate system

F1 to F5 will let you change your primitives.

Extra:

For extra 25% credit implement a torus, the signature should be as follows:



Torus(float innerRadius, float outerRadius, int subdivisions, vector3 color);

- innerRadius is the size of the radius going from the center to the innermost part of the donut.
- outerRadius is the size of the radius going from the center to the outermost part of the donut.
- Subdivisions is the number of subdivisions that goes around the torus.
- Color of the shape.

Submit to the dropbox labeled A4 3D Shapes