|  |  |  |
| --- | --- | --- |
|  | **Rochester Institute of Technology**  **Golisano College of Computing and Information Sciences**  **School of Interactive Games and Media**  **2145 Golisano Hall – (585) 475-7680** |  |

**Data Structures & Algorithms for Games & Simulation II**

**IGME 309**

**A02 & A03 - 3D Primitives pt1 and 2**

Background:

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 5 (in total, for part one and part two) different methods/functions that create buffers and containers able to hold and display data for different 3D shapes.

For part one you have to make these methods / functions that implement the following signatures:

1. Cone(float radius, float height, subdivisions, vector3 color);
   1. radius of the base of the cone
   2. height of the cone.
   3. subdivisions (refer to the cylinder’s subdivisions)
   4. color of the shape



1. Cylinder(float radius, float height, int subdivisions, vector3 color);
   1. radius of the base of the cylinder.
   2. height of the cylinder.
   3. 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.
   4. color of the shape



1. Tube(float outerRadius, float innerRadius, float height, float subdivisions, vector3 color)
   1. outherRadius is the radius of the outermost part of the tube
   2. innerRadius is the radius of the innermost part of the tube (the whole if you will)
   3. height of the tube
   4. subdivisions (refer to the sylinder’s subdivisions)
   5. color of the shape



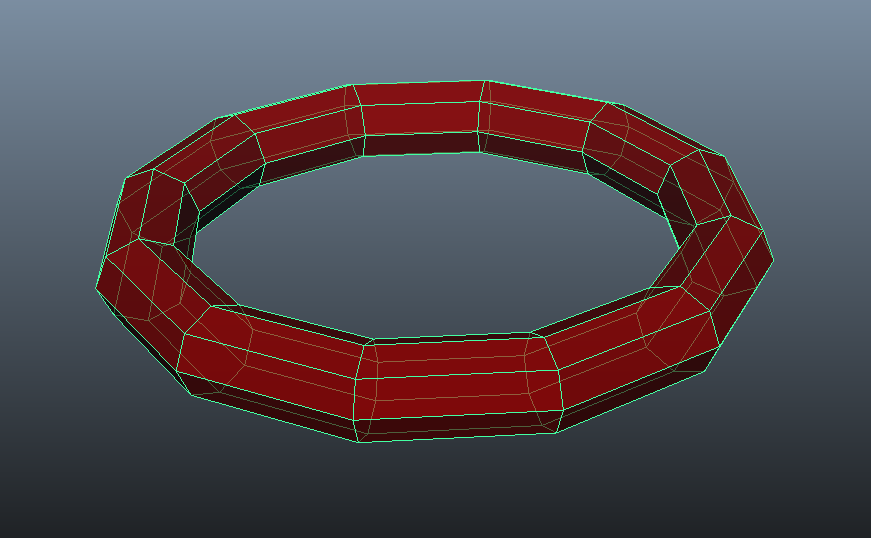
For part two you have to implement these methods using the following signatures:

1. Sphere(float radius, int subdivisions, vector3 color)
   1. radius, how big your sphere is.
   2. subdivisions
   3. 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.

1. Torus(float innerRadius, float outerRadius, int subdivisions, vector3 color);
2. innerRadius is the size of the radius going from the center to the innermost part of the donut.
3. outerRadius is the size of the radius going from the center to the outermost part of the donut.
4. Subdivisions is the number of subdivisions that goes around the torus.
5. Color of the shape.



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

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

* 1. size, is the length of each edge in your cube.
  2. color, the color of this shape



Startup code is available at our repository under “A02 - 3D Primitives pt1” and “A03 - 3D Primitives pt2”.

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

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.

**Tips:**

Tube is similar to Cylinder but requires extra points to calculate.

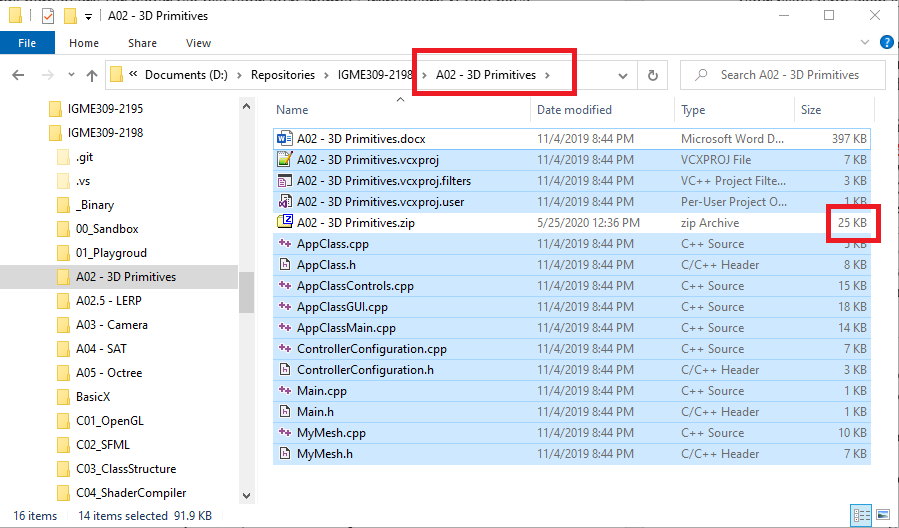
Sphere is the most complex shape in the set and there are various ways to implement it, there are no minimum requirements for it aside that the resulting shape should look like a sphere, if you want to implement a dodecahedron for it that is acceptable, but think about this, if you create a new point 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. Also, for Sphere subdivision means different thing for different methods, I will be very flexible with this one.

**Grading:**

Each shape is worth the same number of points. It is not necessary to have all the shapes working by the end of each assignment, but you will receive partial credit for whatever you do, but it’s a requirement to at least attempt them, partial credit will be given for uncompleted methods and 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 within the first 5 seconds of run time you will not receive any credit. It’s better to comment un-compiling code than don’t receive a grade for it.

As any other submission from this point forward you only need to submit the a .zip file of your project folder ***not your entire solution***. This means your submission is going to be ***less than 500kb***. Submissions will not be accepted if they don’t comply with this requirement, even if they are at a 100% level. Let me elaborate, Visual Studio creates hidden folders that are megs and megs in size to store compiler data which is useless to any other person as another instance of VS will recreate these files for that person’s machine, I also already have all of the libraries which on their own are 20megs in size so in reality anything other than what I’m asking you slows down the grading process. A project folder will look like this:



Notice how without the instructions your whole homework will fit inside of a 25kb .zip file.

You will also need to push to your own private fork of your repository, and I would need access to it. While having it uploaded there is not in the rubric nor you would lose any points for not having it, it is a great help to you in case of a grade dispute.

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 pointers. Memory leaks will subtract points from your final grade.

There are some controls already implemented in this start up code:

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

F4 to F10 will let you change your primitives.

***Submit to the dropbox labeled: A2 - 3D Primitives pt1 and A3 – 3D Primitives pt2***