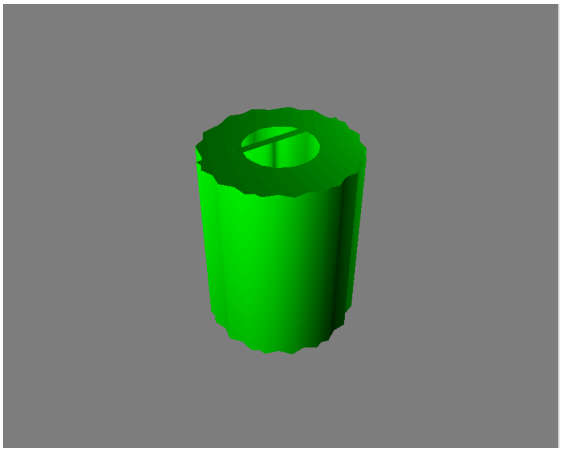
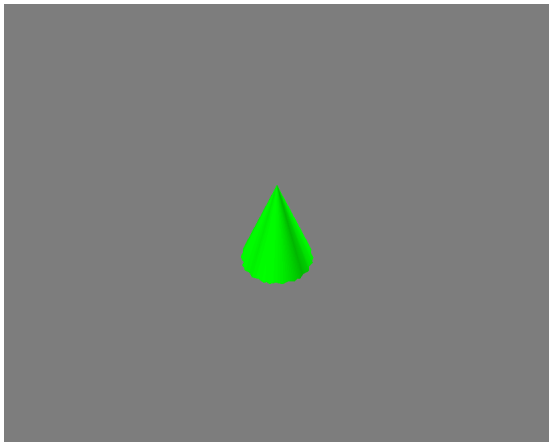
Extra credit part:
Completed 5.1 Additional polygon primitives - Cone & Pipe.





Part of 5.2 Texture mapping in OpenGL
I've completed the creation of a VBO for UVs in the Drawable class and its generation in the create() function of the geometries. I also created a texture, bind the texture, give the texture to OpenGL, and configure it. Yet haven't done the fragment shader / vertex shader part and UV coordinates for each vertex.

Questions:

1. (7 pts) In C++, what is a virtual function and how does it relate to object polymorphism? Say I have a class Base and a class Derived that inherits from Base. Both classes implement a non-virtual function called func(). If func() is invoked by dereferencing a pointer to a Base that actually points to an instance of Derived, how does the compiler know which implementation of func() to call?

Answer:
A virtual function is a member function that we can redefine in our derived class, while preserving its calling properties through references.
Its relationship with object polymorphism can be concluded as:

a. A class that declares or inherits a virtual function is called a polymorphic class.
b. If a member function has been declared as virtual in the base class then we can redefine the function in derived classes. Non-virtual members can also be redefined in derived classes, but non-virtual members of derived classes cannot be accessed through a reference of the base class

The operation can be illustrated as follows:

```
Base * base = &derived;
base->func();
```

Since both classes implement a Non-virtual function called func(), it will call the func() of base. If it's a virtual function, then it will call func() of derived.

2. (4 pts) The ShaderProgram class has several member variables of type int, such as attrPos and unifModel. What do these variables represent? How are they given values in the first place?

Answer:
These variables are numeric handle value(corresponding ID) to the matrix / vector in the vertex shader.
If you do not explicitly assign them to a location then OpenGL will assign them arbitrarily.

3. (4 pts) In the OpenGL Shading Language (GLSL), what is a uniform variable? What is an "in" variable? What is an "out" variable? How does a vertex shader pass data to a fragment shader?

Answer:
In OpenGL, Uniform variables are used to communicate with your vertex or fragment shader from "outside". In your shader you use the uniform qualifier to declare the variable, while in variable stands for linkage into shader from previous stage and out stands for linkage out of a shader to next stage.

The out variables declared in vertex shader will be the in variables in fragment shader, in this way data is passes from vertex shader to fragment shader.