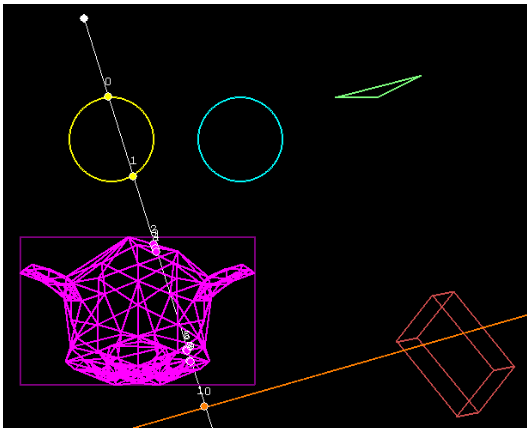
ETGG1803 Lab5: Geometric Primitives Points: 70 (81 possible)
Assigned: 2/20/2017 Due: 3/3/2017 @ noon



1. Get a working copy of vector.py from lab4 (even if using your own, you want to glance at mine – especially a tidbit I added to __init__)
2. Create a new module called **objects3d.py**.
   - a. (**5 points**) Create a **BaseObject** class
     - i. Takes a color (Vector3, where each element is 0.0 – 1.0) and a position (or origin), which should also be a Vector3.
       1. The 0.0 – 1.0 thing will make lab 7 a bit easier.
     - ii. Do type-checking and raise exceptions if the wrong type of data is passed.
     - iii. For all other shapes, use inheritance to derive from this class (make sure to call the super method)
   - b. (**5 points**) Create a **RayCollision** class
     - i. This is where we'll store all information about a ray-object collision
     - ii. The constructor should take (and create attributes) for: The ray involved, the other object involved, the distance along the ray to the collision point, and the point at which the ray hit.
     - iii. Define an __lt__ method that will compare two RayCollision objects based on the distance along the ray (so we can sort collisions in increasing ray distance)
   - c. *Each of the following object should:*
     - i. *Be derived from BaseObject (don't forget to use the super function)*
     - ii. *Define a* ***pygameDraw*** *method (which takes a surface to draw two)*
       1. *Don't forget to convert from our color to pygame color!*
     - iii. *Define a* ***rayTest*** *method (which should return a sorted list of RayCollision objects), or an empty list if there are no collisions.*
   - d. (**6 points**) A **Ray** class (defined by an origin point and a direction vector)
     - i. Additionally, provide a **getPoint** method (which takes a positive distance and returns a point that far along the ray)
   - e. (**7 points**) A **Plane** class (defined by a point on the plane and a normal direction)
   - f. (**7 points**) A **Sphere** class (defined by a center and radius)
   - g. (**8 points**) A **Triangle** class (defined by 3 points)
   - h. (**14 points**) A **Box** class (defined by an origin and a vector representing the half-extents)
     - i. Provide a method to rotate the box around the world x, y, or z axis (you may want to store a set of three axes as attributes, which are modified by this method) – ask for the algorithm…
   - i. (**14 points**) A **TriangleMesh** class (defined by an obj file name exported from blender)
     - i. I'll give you some hints on deciphering the obj file format if you ask.
     - ii. Do an optimization to calculate a bounding box. Only do the full intersection if we hit this.
3. (**15 points**) Create a main program which:
   - a. Creates instances of each shape.
   - b. When the user left-clicks, adjust the origin of the ray
   - c. When the user right-clicks, make the ray point towards the mouse (make sure you handle the case where you right-click on the ray's origin)
   - d. Draw all the shapes
   - e. Calculate, sort, and display the intersection points (if any) at which the ray intersects a shape.
     - i. Also show the sorting order of each point (0 should be closest to ray origin)
4. Here's a demo of my program: `https://youtu.be/IogRMYAjedY`