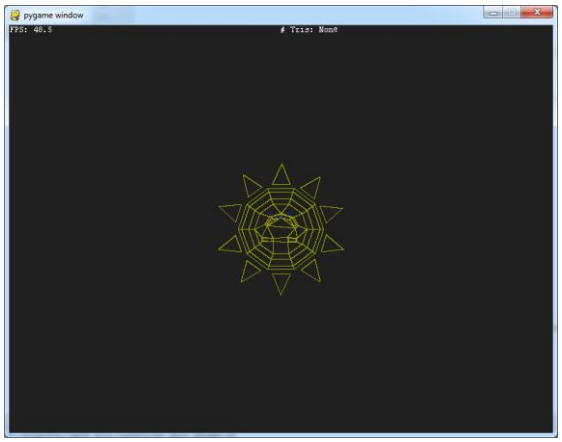
ETGG1803	Lab 9: **Simple Rasterizer** (last one!)	Points: **75**
Assigned: **4/17/2017**	Due: **5/1/2017 @ 8am** (Monday of finals week)	
	NO LATE WORK WILL BE ACCEPTED!	

Tasks:

1. Get a working copy of vector.py and matrix.py. You might also want to reference:
    a. objects3d from lab6 or lab7.
    b. the standard lighting equation code from lab7.
2. Get the .obj files (and .mtl) files
3. (**10 points**) **Phase I**: Transform functions
    a. Make functions in matrix.py that create 4x4 matrices for the following (include a parameter to control whether these are in a left-handed or right-handed system):
    b. Translate
    c. RotateX, RotateY, RotateZ
    d. Scale
4. (**20 points**) **Phase II**: Modify obj file-parsing.
    a. You can start with your code or my code from lab7.
    b. Remove the triangle-only requirement we had in lab7.
    c. We'll need to store all positions as Vector4's (with w=1) and normals as Vector4's (with w=0)
    d. Modify the parsing to pay attention to the **usemtl** portion of the obj file.
        i. Indicates the material to use for any future "f" lines
        ii. (until another usemtl directive is used)
    e. The name referenced on the usemtl line refers to a portion of a companion .mtl file. In that mtl file, one or more materials are defined by:
        i. Ns is shininess (or hardness)
        ii. Ka is ambient color; Kd is diffuse color
        iii. You can ignore everything else.
    f. Calculate the normal for each face and the center of that face.
        i. In a left-handed system, if you are looking at the front side of a face, the vertices will appear to be in clockwise order.
    g. Store a **single** transformation matrix as an attribute.
5. (**20 points**) **Phase III**: Rasterizer framework.
    a. Create a main program or Rasterizer class that can load multiple obj-file meshes.
    b. Use a left-handed coordinate system (which means in pygame, +z would be coming towards you)
    c. You don't actually need a camera (unless attempting Phase IV), but if there were one, it would be somewhere like (400, 300, 500) in an 800x600 window and it would have a cameraZ of (0, 0, -1). Do you see why?
    d. Draw the all the front-facing faces in all meshes using their appropriate color (from the mtl file) [for now, do an outline]
    e. Use the transformation matrix to transform all vertices before drawing (you may want to do this on a copy of the vertices).
    f. Sample results: This is sun.obj with Scale(50, 50, 50) * RotateX(90 degrees) * Translate(400, 300, 0) as the transformation matrix.



6. (**30 points**) Scene Graph
    a. Allow "Dummy" Meshes (not based on a file, just used for their transformation matrix)
    b. Allow a mesh to have 1 or more child mesh objects. When you draw the parent, draw the children as well.
    c. As you draw children, pass them an "accumulated" transformation matrix to combine with their own (as we discussed in lecture)
    d. Don't do any special-purpose modifying of position – everything has to be through transform matrices and parent/child relationships.
    e. Set up a scene graph which behaves as in this video: **https://youtu.be/B5BYITbDayU**
7. (**15 points**) Simple-filled polygons:
    a. Use filled polygons instead of outline.
    b. Sort (based on face centers) all polygons so you draw the farther-away polygons first.
    c. Implement the standard lighting equation when determining polygon color.
    d. You'll need to store lights in your scene….