ETGG1803	Lab6: **Raytracer, part 1 of 2**	Score: **55**
Assigned: **3/13/2017 (MW)**	Due: **3/24/2017 at noon**	

Tasks:

1. **Phase I** (**15 points**)**:** Define the camera and camera-space
   a. Make a **Camera** class (I put mine in objects3d)
      i. Constructor should take these values as parameters:
         1. **pos**: a 3d position of the camera's world-space position
         2. **coi**: a 3d position indicating what the camera is pointed at
         3. **up**: a *general* indication of the camera's up direction (usually the opposite direction gravity would point)
         4. **fov**: the number of degrees made by the view frustum if looking at the camera from camera-x direction
         5. **near**: the number of world units the virtual view plane sits in front of the camera.
         6. **surf**: the pygame surface we'll eventually render to.
      ii. Calculate the following (and store them away for later use):
         1. the **aspect ratio** of the render surface
         2. the camera's local axes (**xAxis**, **yAxis**, **zAxis**)
         3. the virtual **view-plane dimensions** (in world units)
         4. the 3d position of the upper-left corner (as viewed from the camera's perspective) of the virtual view plane [the **view-plane origin**]
   b. Test Case #1:

```
camera position:        <Vector3: 0.0, 0.0, -20.0>
camera coi:             <Vector3: 0.0, 0.0, 0.0>
camera up:              <Vector3: 0.0, 1.0, 0.0>
camera x-axis:          <Vector3: 1.0, 0.0, 0.0>
camera y-axis:          <Vector3: 0.0, 1.0, 0.0>
camera z-axis:          <Vector3: 0.0, 0.0, 1.0>
screen dimensions:      700 x 150 pixels
aspect ratio:           4.666666666666667
camera fov:             45.0 degrees
camera near:            3.2 world units
viewplane dimensions:   12.371178396209773 x 2.6509667991878083 world units
viewplane origin:       <Vector3: -6.1855891981048865, 1.3254833995939042, -16.8>
```

   c. Test Case #2

```
camera position:        <Vector3: 5.0, 7.0, -20.0>
camera coi:             <Vector3: 2.0, 5.0, 3.0>
camera up:              <Vector3: 0.09950371902099892, 0.9950371902099892, 0.0>
camera x-axis:          <Vector3: 0.9878161351194072, -0.09878161351194072, 0.12025587731888435>
camera y-axis:          <Vector3: 0.08725890869187214, 0.9913940281397905, 0.09758977314587813>
camera z-axis:          <Vector3: -0.12886103387626732, -0.08590735591751154, 0.9879345930513829>
screen dimensions:      800 x 600 pixels
aspect ratio:           1.3333333333333333
camera fov:             60.0 degrees
camera near:            1.5 world units
viewplane dimensions:   2.309401076758503 x 1.7320508075688772 world units
viewplane origin:       <Vector3: 3.7416450577771765, 7.843774561957185, -18.57244241401242>
```

   d. Test Case #3

```
camera position:        <Vector3: -5.0, 7.0, -30.0>
camera coi:             <Vector3: 2.0, 5.0, 3.0>
camera up:              <Vector3: 0.0, 1.0, 0.0>
camera x-axis:          <Vector3: 0.9782341251024412, 0.0, -0.20750420835506328>
camera y-axis:          <Vector3: 0.012280720653632045, 0.9982471502738048, 0.05789482593855107>
camera z-axis:          <Vector3: 0.20714048466026375, -0.05918299561721821, 0.9765194276841005>
screen dimensions:      300 x 200 pixels
aspect ratio:           1.5
camera fov:             60.0 degrees
camera near:            1.5 world units
viewplane dimensions:   2.598076211353316 x 1.7320508075688772 world units
viewplane origin:       <Vector3: -5.949417261728117, 7.7757328979667095, -28.21552659472189>
```

2. **Phase II** (**15 points**): Define 3d position "twins" (on the virtual view plane) of pygame pixels (on our screen)
    a. Add a new method to the camera class that will calculate the 3d position (in our virtual world) for any pygame pixel in the render surface.
    b. Using Camera Test Case #1 (from the last page)

```
pixel (0, 0) = world-space <Vector3: -6.1855891981048865, 1.3254833995939042, -16.8>
pixel (699, 149) = world-space <Vector3: 6.1855891981048865, -1.3254833995939042, -16.8>
pixel (350, 75) = world-space <Vector3: 0.008849197708304501, -0.008895861742240996, -16.8>
pixel (113, 23) = world-space <Vector3: -4.185670516028056, 0.9162737594508197, -16.8>
pixel (623, 83) = world-space <Vector3: 4.840511146442593, -0.15122964961809648, -16.8>
```

    c. Using Camera Test Case #2 (from the last page)

```
pixel (0, 0) = world-space <Vector3: 3.7416450577771765, 7.843774561957185, -18.57244241401242>
pixel (799, 599) = world-space <Vector3: 5.871771840594022, 5.89850337029028, -18.463753806833434>
pixel (400, 300) = world-space <Vector3: 4.808009865783406, 6.869562865771644, -18.51806541263644>
pixel (113, 542) = world-space <Vector3: 3.927521946207569, 6.25776765065206, -18.686111246939337>
pixel (723, 11) = world-space <Vector3: 5.803141950102814, 7.605813781493841, -18.324243756792846>
```

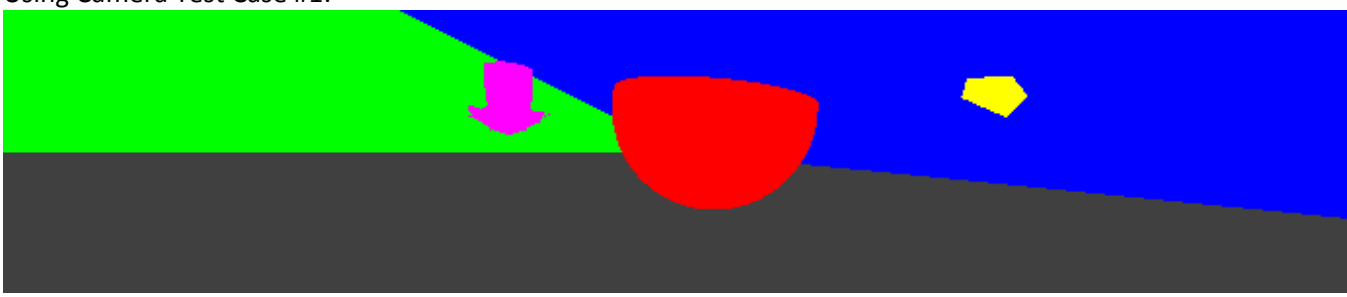
    d. Using Camera Test Case #3 (from the last page)

```
pixel (0, 0) = world-space <Vector3: -5.949417261728117, 7.7757328979667095, -28.21552659472189>
pixel (299, 199) = world-space <Vector3: -3.4291612842910917, 6.046718115181635, -28.85491512222581>
pixel (150, 100) = world-space <Vector3: -4.685092672479383, 6.906881248325969, -28.536374334843 79>
pixel (113, 542) = world-space <Vector3: -5.046840752862432, 3.066556956913894, -28.69238682030266>
pixel (723, 11) = world-space <Vector3: 0.194971790768277 03, 7.680159216506228, -29.52467419261267>
```

3. **Phase III** (**25 points**): Integrate objects3d and make a Solid-color rendering
    a. We'll create the main program together in class (maybe along with a little skeleton code for the raytracer class).
    b. The main interesting function is renderOneLine (feel free to add other helper methods too). Here, you'll
        i. Create a perspective ray for each pixel on that line.
        ii. Do a hit-test with each of the primitives in the scene. The closest one's color should determine the color you set that pixel to in pygame (set it to a background color if no objects are hit)
    c. Here is the scene I used:

        i. Sphere: center=(2,3,5), radius=7, color=(1,0,0)
        ii. Plane1: normal=(0,1,0), point: (0, 5, 0), color=(0,1,0)
        iii. Plane2: normal=(0.1,1,0), point: (0, 4, 0), color=(0,0,1)
        iv. Box: origin=(15, 12, -5), hextents=(9, 5, 2), color=(1,1,0)
            1. Rotated 45 degrees around z, then 45 around y, then 15 around x.
        v. Polymesh: fname="sword.obj", offset=(-10,8,0), color=(1, 0, 1)
            1. sword.obj should be on ssugames (ask if it's not).

    d. Using Camera Test Case #1:



e. Using Camera Test Case #2:

f. Using Camera Test Case #3: