Assignment 1a: Getting Started with Ray Casting **Due Date: Sept. 23, 2015**

In this assignment, you are asked to begin writing a basic ray casting program. This program should: 1) read a simple scene description from a file; 2) define an array of pixels that will hold a computer-generated image of the scene; 3) use ray casting to determine the appropriate color to store in each pixel of the output image; and 4) write the final image to an output file in ascii PPM format.

In subsequent assignments you will be asked to extend the code you write for this assignment to incorporate illumination, shadows, specular reflections and transparency, with these latter effects being achieved by recursive ray tracing. Be sure that your code is structured with these extensions in mind. In particular, you are advised to write simple, modular code following the example described in class.

Detailed instructions:

1. Your program should read the following information from an input scene description file (the syntax is shown below each item; entries in italics are variables, entries in bold are keywords):

- The view origin, also variously referred to as the 'eye position', 'camera position' or 'center of projection' (a 3D point in space)

  eye eye_x eye_y eye_z
- The viewing direction (a 3D vector)

  viewdir $vdir_x$ $vdir_y$ $vdir_z$
- The 'up' direction (a 3D vector)

  updir up_x up_y up_z
- The horizontal field of view (in degrees, please)

  fovh fov_h
- The size of the output image (in pixel units)

  imsize *width* *height*
- The 'background' color (in terms of r, g, b components ranging from 0-1)

  bkgcolor *r* *g* *b*
- A 'material' color (in terms of r, g, b components ranging from 0-1). The material color should be treated as a state variable, meaning that all subsequently-defined objects should use the immediately-preceding material color

  mtlcolor *r* *g* *b*

- A sequence of one or more objects. For this assignment, your program simply needs to be able to handle spheres. In subsequent assignments you will be asked to extend your code to handle triangles, so you will want to write your code with this future extension in mind. A sphere should be defined by the coordinates of its center point (a 3D point in space) and a radius.

  sphere c_x c_y c_z r

A sample scene description file is provided [here].

2. Define an array of sufficient size to store the color values of your image. Based on the size of the output image, determine the aspect ratio you need to use for the viewing frustum.

3. Based on the view origin, viewing direction, up direction, and aspect ratio of the viewing frustum, you can define an arbitrary "viewing window" in world coordinate space, and then define a 1-1 mapping between points within this viewing window and pixel locations in your output image.

4. For each pixel in the output image:
   - initialize its color to the background color for the scene
   - define the equation of the ray that begins at the view origin and passes through the corresponding 3D point in the viewing window
   - for each object in the scene:
     - determine whether the current viewing ray intersects that object, and if so at what point or points
     - if there are one or more ray/object intersection points that are 'in front of' the view origin with respect to the positive viewing direction, determine which of them is closest to the view origin; ray/object intersection points that are 'behind' the view origin, with respect to the viewing direction, should be ignored
     - determine the color of the object at the closest valid ray/object intersection point, if there is one, and store this value (instead of the background color) at the appropriate location in your image array

6. After all of the rays have been cast and a color has been determined for each pixel in the output image, write the final image to an output file, using the ascii PPM format.

You are strongly encouraged to begin with a very simple scene description, consisting for example of a single sphere located directly in front of the viewer. You are also advised to begin by using a very small image size, to expedite the debugging process.

What you should turn in:

- all of your source code, clearly commented, and any instructions we might need to compile your code
- two different text files containing two different scene descriptions that are handled by your program. Please try to be creative!
- two ascii PPM images that show the result of running your program using the provided scene description files as input
- a readme file that describes your input file format and explains how to compile your code