Assignment 1a: Getting Started with Ray Casting   Due Date: Mon Sept. 23, 2019

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 provided as a command line argument; 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.

You are asked to run your program on a variety of different input files, featuring a wide range of different input parameter settings. The goal of this second requirement is to help you develop an intuitive understanding of how the various viewing parameters impact the appearance of the rendered scene. Please be sure to specifically consider these questions:

- how the direction of the 'up' vector affects the apparent rotation of the scene with respect to the viewpoint;
- how changes in the field of view settings affect the appearance of the scene in your rendered image;
- how various modifications of the viewing parameters affect the apparent amount of perspective distortion in your image

Finally, you have the opportunity to earn 7% extra credit by enabling your program to render not only spheres but also ellipsoids, and/or 3% extra credit by enabling your program to define a varying background color, depending on the direction of the ray that returns the background color.

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 vertical field of view (in degrees, please)

  `vfov` *fov_v*

- 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 only 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 radius.

  `sphere` *c_x c_y c_z r*

- If you would like to attempt the extra credit portion of this assignment, please use this format to define an ellipsoid:

  `ellipsoid` *c_x c_y c_z r_x r_y r_z*

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:
   - 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 return this value to be stored at the appropriate location in your image array
     - If no ray/object intersection is found, return the background color, or (if you want to attempt the extra credit portion of this assignment) a variation of the background color

5. 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.

To explore the effects of various scene parameters, you are advised to vary the values of just one parameter at a time, such as: the eye position, the viewing direction, the direction of the 'up' vector, the vertical field of view, the image size, or image aspect ratio. Once you understand the effect of each parameter in isolation, you can experiment with combinations of changes, such as co-varying the size of the field of view and the proximity of the eye to distant objects in the scene (e.g. displacements of the view origin along the viewing direction). Your ability to appreciate the various effects of different view settings will be improved if you use a relatively complex scene, with some asymmetric features and not too much empty space.

<u>What you should turn in:</u>

- A single .zip file (not rar or tar or anything else) that contains all of your source code, clearly commented, and a Makefile or CMake file that the TA can use to compile your code
- One "showcase" image produced by your program, that I can share with the rest of the class
- A 1-3 page writeup (including pictures) in which you discuss your observations on how the key viewing parameters affect the appearance of the rendered scene. Please incorporate multiple different images produced by your ray casting program to illustrate and explain your findings. In your writeup, be sure to specifically address each of these three points:
    - how does the apparent rotation of the scene with respect to the viewpoint change with changes in the direction of the 'up' vector? [please experiment with using multiple different directions to define the up vector]
    - how do changes in the field of view settings affect the appearance of the scene in your rendered image?
    - how can the viewing parameters (e.g. the camera location, field of view settings, …) be adjusted to achieve a less exaggerated vs more exaggerated amount of apparent perspective distortion in your image?

CSci 5607, Fall 2019
Assignment 1a: Getting Started with Ray Casting
Due: Monday Sept 23rd

Name ______________________________

Score (out of 100) _______________

____ The materials were submitted as a single .zip file, and a Makefile or CMake file was included. The submitted code is clearly documented, is platform independent, and can be easily compiled for grading. (5 pts)

____ The program is capable of correctly reading a valid scene description from a file. The program accepts the name of the input file as a command line argument. (10 pts)

____ Given valid input viewing parameters (view origin, view direction, view 'up' vector, vertical field of view angle, and output image dimensions) the program correctly determines the corresponding 3D viewing coordinate system [(*u, v, n*), where the vectors *u* and *v* define the horizontal (left-to-right) and vertical (down-to-up) directions of a viewing window in 3D world coordinate space and *n* = –*view_dir*] that defines how the scene contents will be imaged. (10 pts)

____ The program correctly uses the derived viewing coordinate system to obtain the (*x*, *y*, *z*) coordinates of the four corners of a viewing window that, in conjunction with the view origin, delimits the volume of space whose contents will be rendered into the ray-traced image. (5 pts)

____ The program allocates an appropriately-sized 2D image array to store the (*r*, *g*, *b*) colors of each pixel that will make up the final image. The program defines an appropriate one-to-one mapping between pixels in this 2D image array and 3D points in the viewing window to enable an even sampling of the 3D scene. For each pixel, a viewing ray is correctly defined and cast into the scene to determine the color values to be stored at that point in the image. (15 pts)

____ The program correctly determines when and where a viewing ray intersects a single sphere. This capability is demonstrated via an image that shows a color difference between pixels where a ray/sphere intersection occurs and pixels where there is no ray/sphere intersection. The colors in the rendered image appropriately correspond to the specified colors in the input file. (15 pts)

____ The program correctly renders models containing more than one sphere, and correctly renders multiple spheres of different colors. (10 pts)

____ The program correctly outputs the final computed image in a valid PPM format. (5 pts)

____ The program is robust. It operates on arbitrarily-named input files, robustly handles reasonably formatted input, and gracefully exits with an appropriate error message if "invalid" scene description data is encountered, such as: zero length or non-unit-length input vectors, parallel up and viewing directions, fields of view greater than or equal to 180 degrees, zero or negative image width, or any other invalid or incomplete scene or object definitions. Specifically, no assumptions should be made about the input that would cause the program to "crash" if those assumptions are violated. (10 pts)

____ The student has turned in a 1-3 page writeup that appropriately answers each of the three questions specified in the project description. The report is illustrated with multiple different images produced by the student's own program. (15 pts)

____ The program is capable of correctly handling ellipsoids as well as spheres. (7 pts extra credit)

____ The program is capable of rendering a gently varying background color. (3 pts extra credit)