CSC 473
Program 2 – shading + shadows and camera – 9% total

Due Sunday April 17th, at 11:55pm

Overview:
Throughout this quarter you will implement the basic functions of a distributed ray tracer. *This software will be enhanced throughout the quarter, thus this second assignment builds on your prior code and will serve as the base for following assignments*. Since you will be adding and reusing your code, it is advised that you write your code in a clean, structured object-oriented fashion. All code must be written in C++.

Goal for assignment 1: In this assignment you will implement the direct illumination shading and shadow casting (and drawing) for your ray tracer along with a free camera, supporting only spheres and planes in terms of intersections. You code must include implementation of camera transforms and multiple objects. Specific technologies that must be supported are computation of:

- the shading for opaque non-reflective objects using two different BRDFs (you will need to implement the Blinn-Phong model and your choice of an alternative shading model (e.g. Cook-Torrance)). Please see: https://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function Your shading must use the light and material properties specified in the pov file. In order to switch between the two different BRDF models use a command line option (see "program execution below)
- shadow feeler rays for all intersections that are used for computing shadows for all objects. Please experiment with appropriate epsilon offsets to avoid shadow pimples.
- correct images for camera with arbitrary world view, i.e not only looking down the negative z axis (e.g. simp_cam.pov files)

Software engineering considerations:
You will once again need to support specific unit tests throughout the quarter for specific rays (relative to the camera – i.e. listed in pixel space, e.g. [Xi, Yi]). For this assignment, expect to have plane intersections tested, along with shading values for specific rays. At this time, please be sure that you are supporting testing via two different mains – i.e. build your default ray tracer that renders all rays and include an alternative main that only traces (and prints) values for specified rays.

Program execution:
Your program should have the following syntax:

```
raytrace <width> <height> <input_filename> <BRDF>
```

where the options are:
  width = the image width
  height = the image height
  input_filename = the name of the povray file to read and render
  BRDF: value of 0 or null uses Blinn-Phong, values of 1 use your alternative BRDF
Thus:
raytrace 640 480 sample.pov
or
raytrace 640 480 sample.pov 0

will render a 640x480 image file, "sample.tga" consisting of the scene defined in "sample.pov" using the Blinn-Phong BRDF for shading.

Sample input files and images are given on the class webpage. Some of these pictures may be generated by Povray and will not look identical to your output (due to differences in the shading model, etc.).

What you should hand in via polylearn:

- Your code, include all files necessary to compile and run your ray tracer, including a Makefil or cmakeLists.txt
- A README.txt file with any information about what is working or not working with your implementation to assist the grader in determining what is causing potential errors in your output and help in assigning partial credit.
- renders of specific files (all simp_cam files)

You need to handin your code and images generated using poly learn. Look for the assignment directory.

Grading breakdown:
40 points two different shading models working with all files
30 points working shadows with all files
30 points working camera with arbitrary views with all files
