CS 174A: Introduction to Computer Graphics: Assignment 3

Weight: 15 %
Maximum points: 40
Collaboration: **None**. If you discuss this assignment with others you should submit their names along with the assignment material.
Start working on this assignment early. You will not have time to do it well at the last minute.

Instructions: Submit your assignment on CCLE in an archive named <*uid*>.*zip*, where <*uid*> denotes your 9 digit bruin UID. This should include ALL of the code necessary to compile and run the program, and should not contain any functionality beyond what is requested below. The `main` function should be in a file named raytrace.cpp. Include a README file that indicates which of the tests the output of your software passes and which ones if any it has not passed.

Your solution should not contain any OpenGL API calls. However, you may use OpenGL to display your results during debugging. In the materials associated with this assignment on the CCLE course website, you will find two pieces of code: One inverts a 4x4 matrix, and the other writes a char buffer to a ppm image, which is the expected output of this program.

For this assignment, you will be implementing a Ray Tracer. Your system need only handle the rendering of spheres, with a camera situated at the origin of a right-handed coordinate system, looking down the negative z axis. Local illumination, reflections, and shadows must be implemented. The following will be taken as inputs:

a. The near plane[2], left[2], right[2], top[2], and bottom[2]
b. The resolution of the image nColumns[1] x nRows[1]
c. The position[2] and scaling[2] (non-uniform), color[3], K_a[3], K_d[3], K_s[3], K_r[3] and the specular exponent n[2] for up to five spheres
d. The position[2] and intensity[2] for up to five positional, point light sources
e. The background color[2]
f. The scene's ambient intensity[2]
g. The output file name

[1] int [2] float [3] float in the range [0..1]

Your program should take a single filename as a command-line argument. The input file describes the scene in the following format:

NEAR <n>
LEFT <l>
RIGHT <r>
BOTTOM <b>
TOP <t>
RES <x> <y>
SPHERE <name> <pos x> <pos y> <pos z> <scl x> <scl y> <scl z> <r> <g> <b> <K_a> <K_d> <K_s> <K_r> <n>
LIGHT <name> <pos x> <pos y> <pos z> <I_r> <I_g> <I_b>
BACK <r> <g > <b>
AMBIENT <I_r> <I_g> <I_b>
OUTPUT <name>

All fields (NEAR, LEFT, RIGHT, etc.) are separated by one or more carriage returns. Note that the SPHERE and LIGHT fields may appear up to five times each. Also, the fields may appear in any order, and blank lines may occur between them.

All parameters (<name>, <posx>, <posy>, etc.) are separated by one or more white space characters such as spaces or tabs. There will be no angle brackets in the input file; the ones above merely indicate the parameters. Also, names should be limited to 20 characters with no spaces.

(cont'd)

Grading scheme:

[5 points] Parsing of input files
[5 points] Coding Style (i.e., well designed, clean, commented code)
[10 points] Ability to cast a ray and display the spheres properly
[10 points] Local illumination
[10 points] Shadows and Reflection

Notes:

- The code that inverts a 4x4 matrix expects two 4x4 matrices as arguments. The first matrix will be inverted and the result will be stored in the second matrix. Both matrices are row-order, so you have M[row][column].
- Make sure that your parser routine does not crash based on where the EOF is.
- Given a reasonable resolution (400x400), your program should take no more than a couple of seconds to run when compiled in "release" mode.
- When intersecting a ray with an object, the textbook specifies to choose the minimum hit time greater than 0. Given their parametric ray formulation, this is incorrect. The closest object is the one with minimum hit time greater than 1. A hit time between 0 and 1 falls between the eye and the near plane, and hence is not a part of the view volume. Note that reflected rays do not have this problem.
- When creating shadow rays from the closest hit point to the lights, you are looking for any intersections with hit time between 0.0001 and 1. To deal with numerical error, you should not consider an intersection at time 0 to be blocking the light from the object.
- The template code for saving your image to disk uses the ppm image format. If you do not already have a program that can read images of this type, you can download IrfanView from the web (www.irfanview.com). This is an excellent, free program for viewing images, and can read many image formats, including ppm files.
- The "NEAR" value is an absolute value and represents the distance along the negative z-axis.
- Your code should handle hollow spheres which are "cut" by the near-plane.
- You are encouraged to use the STL string and vector classes.
- You will be using the following local illumination model:
    - PIXEL_COLOR[c] = $K_a*I_a[c]*O[c]$ +
      for each point light source (p) { $K_d*I_p[c]*(N.L)*O[c]+K_s*I_p[c]*(R.V)^n$ } +
      K_r*(Color returned from reflection ray)
        - O is the object color
        - [c] means that the variable has three different color components, so the value may vary depending on whether the red, green, or blue pixel color is being calculated
        - You should calculate the reflection color up to a depth of three (i.e., you should not recursively spawn more than three reflection rays for each pixel)
        - The other components of this equation are explained in the lecture notes
- When summing contributions over all the lights, it is possible for the PIXEL_COLOR value to exceed 1. In this case, the simplest solution is to clamp the value to 1. Do not forget to scale by 255 before creating the ppm image using the given save_image function.