CSC418/2504: Fall 2013 Assignment 3 Project

Due: Nov. 27, 2013. (The Wooden Monkey Competition) (worth 25% of final grade) [40 marks +50 marks+10 marks+10 BONUS marks]
You may work in teams of two students. Both students will receive the same mark.

[40 marks] Your first task is to implement a basic ray-tracer and render a simple scene using ray casting and local shading. The starter code sets up a scene comprising of an ellipsoid and a plane, being illuminated by a point light source. Your job is to render the scene by implementing code fragments required for object intersections and Phong shading.

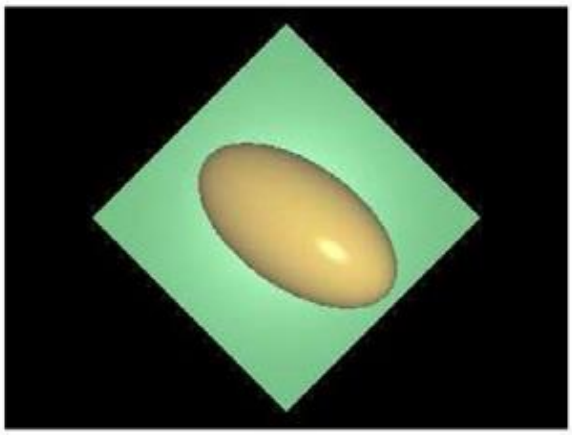
You will have to implement the following code fragments:

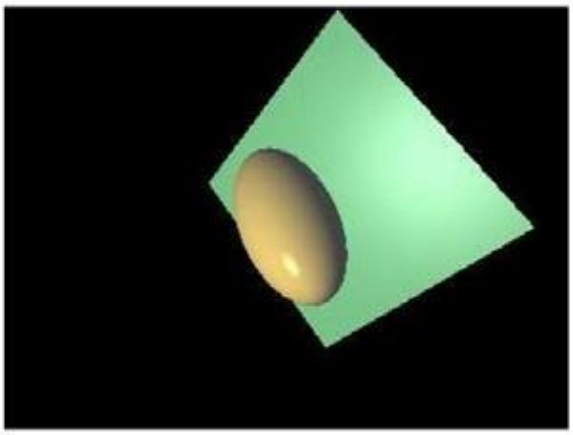
a. [10 marks] Ray casting.
b. [10 marks] Intersection code for ray-sphere intersection (it should also work for affinely deformed spheres).
c. [10 marks] Intersection code for ray-square intersection (it should also work for affinely deformed squares).
d. [10 marks] Phong shading for a point light source (including ambient, diffuse, and specular components).

To demonstrate the working of your program, you should generate three different types of renderings:

1. a *scene signature* where each pixel shows a unique color identifier for the first object hit (or background). This gives you an impression of the relative positions of the camera and the objects.
2. a rendered scene with only the diffuse and ambient components of the Phong model.
3. a rendered scene with all three terms of the Phong model.

In addition to specifying your scene, the starter code has been written to generate two distinct views (images) every time you run it. Thus, for each type of rendering above you will generate two views, comprising a total of six images. In your electronic submission, please call these image files `sig1`, `sig2`, `diffuse1`, `diffuse2`, `phong1` and `phong2`. Below are two images of what the Phong rendered scene should look like. Finally, ray-tracing is compute-intensive, so debug your code on small images, e.g., 200x200, and debug ray intersections using the scene signature. Also, when writing your code, keep in mind that your second task will build upon this work so make sure your implementation is modular and re-usable. Make sure you read the helper code carefully and only implement the functions that are necessary, i.e. try to avoid creating unnecessary structures or classes.





Helper Code To get started, we have created a simple demo for you. The demo sets up the scene and basic class structures required. Note that the helper code is written in C++ and does not use OpenGL. It will also provide you with a template Makefile for compilation and linking of your programs. To unpack and compile the helper code on `CDF`, download the file `a3.tgz` and use the following commands

- tar xvfz a3.tgz
- cd a3/raytracerLinux
- make
- raytracer

[50 marks] Your second task is free-form. you may build upon the raytracer or OpenGL to produce an animation, special ray-trace effects or an interactive game. Use your imagination and any resources you want (**as long as you cite them in your report**). As a guideline to the scope of what is expected:

Animation or interactive game: create a hierarchical scene with several control variables (degrees of freedom). You may use simple primitives like cubes, cylinders etc. like in assignment 1 and 2 or more complex polygonal geometry that you model or import into your program. The scene should have some articulation (jointed figures). Animate the scene by any combination of techniques to control your control variables.... key-frames, a physical simulation, motion capture data or user interaction. If you use key-frames interpolate between key-frames (use cubic interpolation for smoothness). Lights and cameras in your scene should be animatable as well. Render the scene using your ray tracer or OpenGL. Collate the image frames into a movie file.

or

Advanced ray-tracing: Extend your ray-tracer to enable recursive ray tracing to show secondary reflection with specularities (so you can see one object reflected in another) and shadows. Then implement at least 5 of the features below:

- Handling a non-trivial compound object, containing quadratic surfaces (e.g., a cone or cylinder). Spheres, ellipses, planes and patches thereof don't count.

- Handling arbitrary surface mesh geometry.
- Anti-aliasing
- Glossy reflections
- Depth of field
- Extended light sources, in order to to produce soft shadows
- Texture-mapping (an interesting procedural texture is also acceptable).
- Adding environment mapping to item 7.
- Motion Blur
- Refraction (*e.g.* glass spheres)

[10 marks] Write a clear one page *REPORT* describing:

- your overall submission.
- program, and other submitted file structure.
- what you have implemented and what external resources you have used.
- the role of each member on the project.

[10 BONUS marks] overall wow factor (check out past winning wooden monkey entries).

Turning in your Solution:

All your code should remain in the directory a3/raytracer. In addition to your code, you must complete the files *CHECKLIST* and *REPORT* contained in that directory. *Failure to complete these files will result in zero marks on your assignment.*

Your code should be well commented if you want to receive full (or even partial) credit for it. To pack and submit your solution, execute the following commands from the directory containing your code (i.e., `a3/raytracer`):

- cd ../../
- tar cvfz a3-solution.tgz a3
- submit -c csc418h -a A3a a3-solution.tgz (if registered for CSC418)
- submit -c csc2504h -a A3a a3-solution.tgz (if registered for CSC2504)

Compatibility

This assignment will be demo'ed to the TA's during an alloted time. You are welcome to work on any platform but must be able to run your demo in front of the TA. Also your submitted files must either run on `CDF` (Linux), or provide a self-contained windows compilation environment (Do not expect the TA's to fiddle with the project settings or search for 3rd party libraries to compile your code).