Part (b): Advanced Raytracing

Assignment Description

This part of the assignment is open-ended. You need to add **THREE POINTS** worth of non-trivial advanced features to your raytracer from part (a). You're free to choose what you want to implement, but if it's not on the list below, check with me in advance to figure out how many points it counts for. Here are some suggestions. For some of these, you may need to write your own .scd files, extend the .scd format, or specify the input in some other clean way (avoid hardcoding it in the code if possible). You may make any changes to the code you need.

- **(1 point) Refraction:** Handle transparent objects. As a test, try viewing an object through a glass sphere, or even better, try to model a lens-shaped object. The latter will count for an extra point or two since you have to write a new primitive type, see below.
- **(1 point) Area lights:** Sample lots of points over the area and trace multiple shadow rays to them. The result should have nice soft shadows. Also figure out how to handle Phong shading in this situation.
- **(1 point) Triangles:** Complete the `Triangle` class (in Primitives.[ch]pp), and use it to render the "teapot" and "dunkit" scenes which involve triangle meshes. The code to load the mesh and populate the World object with its triangles is already present in the skeleton code. You can use MeshLab to preview the meshes.
- **(1 point each) Other primitives:** Implement quadrics, torii, superquadrics etc. See Sid's guide for solving equations of degree 3 and 4.
- **(2 points) Constructive solid geometry (CSG):** CSG models new objects by boolean operations on old ones. For instance, a convex lens is the intersection of two spheres. Raytracing provides a very clean way to handle CSG: just check which intersection points satisfy the condition.
- **(2 points) Acceleration structure:** Implement an acceleration structure, such as a bounding box (or bounding sphere) hierarchy, or an octree, or a kd-tree, and use it to speed up rendering of scenes like #3. Your structure should work for (at least) spheres and triangles, and be general enough to work on any scene with these primitives. *Suggestion:* add a function to the Primitive class to get the bounding box of the primitive, and specialize it in derived classes. When constructing the tree, use this bounding box (instead of the actual primitive, which can be trickier) to determine membership in a node.
- **(2 points) Camera lens with depth-of-field effect:** See midsem question 6. Instead of a pinhole camera model, simulate an actual lens placed in front of the camera, with associated background blur effects. Remember, the eye position is no longer important -- you need to cover the surface of the lens with lots of rays from each pixel on the image plane, and track them after refraction through the lens. If you have to implement a new primitive for this, the associated points add up. But you can also just hand-code the specific refraction formula for this special case (thin lens approximations are fine).
- **(2 points) Arbitrary BRDFs**
- **(3 points) Global illumination:** via photon mapping, or radiosity.

Grading Scheme

33.3% for every implemented point, clamped to a maximum of 150%.

Submission Instructions

Please submit a zip of your assignment directory, containing:

- All source code, which should build on our Unix systems by just typing `make` without our having to install anything extra.
- All .scd, .obj and any other data files in the `data` subfolder.
- An `images` subfolder, at the same level as `data`, containing all your new rendered images.
- A plain-text README file detailing:
    - A brief writeup on everything you implemented. One bullet per feature. Include all info you think is relevant, including and especially a list of the functions and classes created/modified to implement the feature.
    - The **EXACT** Bash command lines needed to reproduce every single image in `images`. We will run the command lines from the `data` folder. We should not need to rebuild your code to turn different features on/off -- make these command line options. List the full command lines one after another, e.g.

```
../trace  refraction_scene.scd  ../images/refraction_image.png 4
../trace  csg_scene1.scd  ../images/csg_image1.png 2
../trace  csg_scene2.scd  ../images/csg_image2.png 2
../trace  area_light_scene.scd  ../images/area_light_scene.png 3
../trace  scene3.scd  ../images/complex_scene.png --enable-octree 3
```

    - The approximate time taken to execute each of the command lines above on your system.

Do NOT submit object/executable files. Run `make clean` before zipping.