UNIVERSITÄT BERN

Introduction to Computer Graphics
Assignment 4 – Raytracer Extensions and OpenGL "Hello World"

Handout date: 18.10.2019

Submission deadline: 25.10.2019 12:00

This assignment gives you an opportunity to play around with and extend your raytracer. It is also intended to ensure that you can build and run the OpenGL framework code we'll be using for the upcoming assignments. This assignment will not be graded, but we will select a few of the best submissions to showcase in an upcoming exercise session. Please do make sure the OpenGL code is working on your computer during this assignment; in later weeks, we will not provide assistance in setting up the code.

Here is a list of potential ideas for using or extending your raytracer:

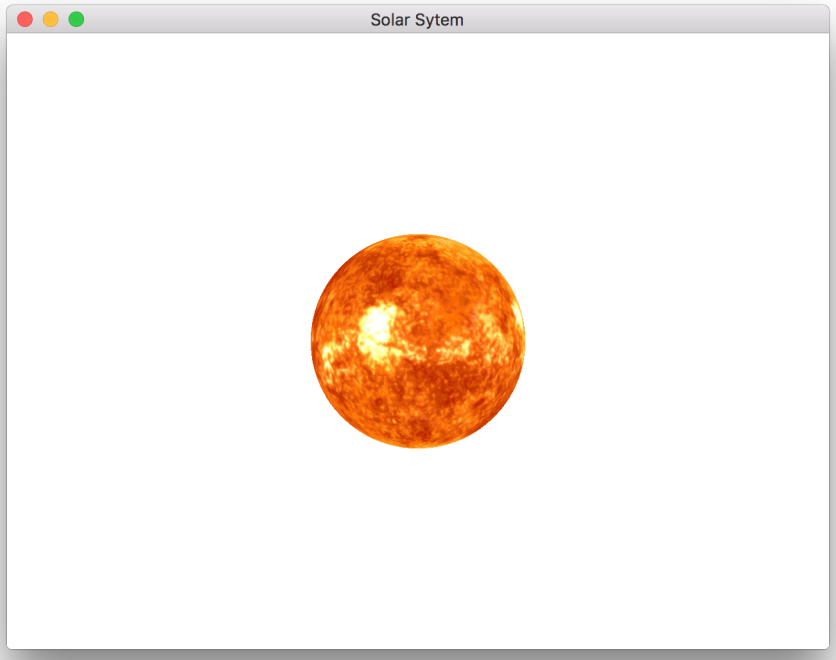
- Set up an interesting scene. Create a new scene file/collection of meshes that produces a nice image. You might consider downloading some free triangle meshes from an online service (e.g., sketchfab, cgtrader, or turbosquid) or creating your own in Blender. Note that whatever mesh you use must be converted to `OFF` format to be loaded into a scene. You can do this conversion with Blender.
- Render glass objects by implementing refraction. Refraction works similarly to reflection, though you'll need new formulas for computing the angle of the refracted ray (this can be done with Snell's law given the index of refraction of air and glass) and determining how to mix the reflected and refracted colors (this can be done with the Fresnel equations). Implementation details can be found in this article.
- Add a torus primitive and implement ray-torus intersections. You can take the same approach to deriving the intersection equation as you did for cylinder intersections. However you will end up with a quartic equation, not a quadratic one. You'll likely want to use a numerical method to solve for the smallest positive root of this equation instead of evaluating the rather complicated closed-form root expressions (see Wikipedia).
- Generate a movie. An example of how to do this is provided in `scenes/movie` of last week's assignment code. On Linux and macOS, you should be able to open a terminal in this scene directory and run `bash gen_movie.sh` (after installing `ImageMagick` and `FFmpeg`). This script assumes that you built your raytracer binary in a `build` directory inside the main project directory. The script feeds 90 different scene files into the raytracer to render each frame of animation and then stitches the output frame images into a movie.
- Speed up the rendering of complex meshes by implementing a spatial acceleration structure as presented in the lecture, e.g., a bounding box hierarchy or BSP. Make your own or utilize existing implementation, maybe from CGAL. Benchmark with a complex mesh and see how fast you can make it.
- Implement a more complex surface lighting model, e.g., Oren-Nayar for diffuse light from rough surfaces, or Cook-Torrance specular reflection.

- Implement textures – image-based or procedural. For this, modify the `intersect()` signature to also compute texture coordinates. From this you can either compute a color (e.g. checkerboard pattern) or sample an image texture. For loading images, see the next suggestion:
- Implement image backgrounds: Instead of taking a fixed background color for rays that do not intersect with anything, take a value from an image based on the *direction* vector. You can find nice free image textures for this at `https://hdrihaven.com`. To read image files, you can copy over the *LodePNG* library from the SolarSystem project (let us know if you need support getting this to compile; you'll have to edit the `CMakeLists.txt`).
- *(Advanced)*: Extend the ray tracer to a path tracer for more realistic results. A good resource is the online edition of the book Physically Based Rendering.

Note that if you add new `.cpp` files for your implementation, you will need to edit `src/CMakeLists.txt` to include them.

OpenGL Framework

Please download and unpack `assignment_4.zip` and see its `README.md` for instructions on building the framework code. If all goes well, you should see the following window when you run the `SolarSystem` binary:



Grading

This assignment is ungraded.

What to hand in

A compressed .zip file with the following contents:

- The source files that you added or changed in the raytracer.
- The results of your work, e.g., images or movie you created.
- A screenshot of the running OpenGL assignment code.
- A `readme.txt` file containing the full names of each team member, describing what you added to or did with the raytracer this week. Please also tell us any difficulties you encountered in getting the OpenGL framework running.