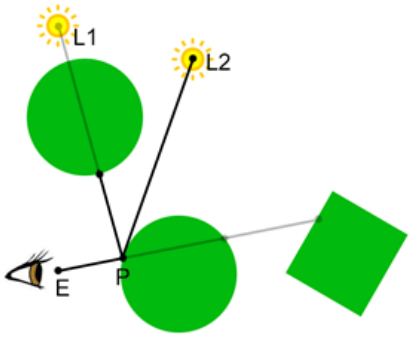
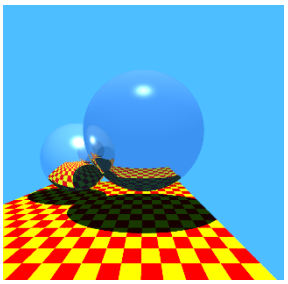
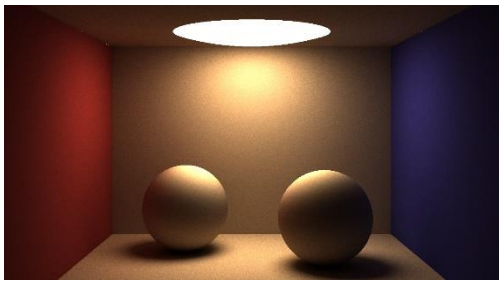
Advanced Graphics 2021/2022 – Assignment 1







Introduction

First of all: this assignment is a preparation for Assignment 2. Without a successful assignment 1, you will still be able to do assignment 2, but you will not have the benefit of a comfy testbed that you know like the back of your hand (or, in Dutch, *like your pants pocket*).

We will soon move to the construction of a real-time renderer. For Assignment 1, you will be laying the groundwork for this, in the form of a test bed for ray tracing and path tracing. This test bed consists of the ingredients commonly required for ray tracing, plus a Whitted-style ray tracer for validation.

Right from the start you have three main options to pick from, as discussed in the first lecture. You may implement this testbed either ***(1)*** as a 'render core' for the Lighthouse 2 renderer, or ***(2)*** as a stand-alone application, e.g. built on top of the C/C++ template, or ***(3)*** as a stand-alone application but using the voxel template for ray/scene intersections. If you chose the first option, the Lighthouse 2 framework takes care of a lot of mundane tasks, which allows you to focus on the essence of the assignment, at the cost of having to 'learn' Lighthouse 2. The second option lets you skip learning how Lighthouse 2 works, but this forces you to do more low-level work yourself. The third option saves you some work on writing intersection code but limits you to voxel scenes. All three options should yield a rewarding experience, so pick the one that fits you best. And finally: code you write for option 2 or 3 should be mostly interchangeable, so there's some room to change your mind later.

Architecture

In a nutshell, a ray tracer renders an image based on a scene description, given a camera and a screen plane, using rays originating from the camera and extending through pixels on the screen plane, which then find intersections with the geometry, and occlusions between this geometry and the light sources.

For subsequent assignments, we will extend and optimize this functionality to produce a real-time physically based renderer.

Ray Tracing

To demonstrate the suitability of your framework, implement a **Whitted-style ray tracer** (also referred to as a **recursive ray tracer** in literature). The Whitted-style ray tracer must support shadows, reflections and dielectrics. Dielectrics must support Beer's law. Support of nested

dielectrics is optional. You may also ignore scenarios where the camera starts in a medium other than air. Note that a Whitted-style ray tracer does **not** support area lights and glossy reflections. Illumination from area lights is thus not supported.

Practical Details

The deadline for this assignment is Wednesday December 8th, **17:00** (note the unusual time: this is for your health). An extended deadline is also available: in exchange for a 1 point grade reduction you can hand in materials until Thursday December 9th, **17:00**.

The materials to submit are:

- your project, including sources and build instructions (if these are not obvious);
- a brief report, detailing implemented functionality, division of work, references and other information relevant to grading your submission.

Please **clean** your project before handing it in. For the provided templates, this is easily achieved by executing `clean.bat` in the template directory or in the root of the Lighthouse 2 project folder. Note that your project must not be open in Visual Studio when you run this file. For sane file sizes: **please exclude .svn and .git folders** from your submission. PROTIP: if the final package is more than 50MB, you did it wrong. I'll not punish you, but it's wrong.

You may work on this assignment **alone, or with one other student**. I recommend working together: you learn more, and it's probably healthier as well, especially if the course should go online at some point in time. Note that you are not required to work in the same team for all three assignments. Do be gentle if you split up of course.

You may implement your platform in C++ or C# or any other programming language that you may prefer. Do note that support in working colleges may be limited if you chose an exotic programming language (don't let that discourage you!). You may also target other operating systems than Windows, but again, support may be limited and I have to assume that you can support yourself.

Feel free to discuss practical details on Teams or elsewhere. You are not supposed to share complete ray tracers or significant chunks of crucial code, but if everyone uses the same optimal ray/triangle test, that would be considered 'research' and 'smart'.

$\geq$

Tasks & Grading

OPTION 1 – LIGHTHOUSE 2

If you decide to develop a render core for **Lighthouse 2**, a passing grade (6) for this assignment requires:

- correct handling of the 'ViewPyramid' handed to you by the RenderSystem;
- support for triangles, as handed to you via SetGeometry;
- basic handling of materials passed via SetMaterials;
- a Whitted-style ray tracing renderer, running on the CPU, supporting shadows from point lights, reflections and refraction with absorption (Beer's law).

To obtain additional points, you may work on the following:

1. Handling of the textures and materials that the RenderSystem passes to the core (max 1pt).
2. Correct handling of Lighthouse 2 spot- and directional lights (max 0.5 pts).
3. Anti-aliasing (max 0.5 pts).
4. Barrel distortion and a fish eye lens (max 0.5 pts).
5. Image postprocessing: gamma correction, vignetting and chromatic aberration (max 0.5 pts).
6. A multi-threaded rendering system that yields at least a 4x speedup (max 1pt).

OPTION 2 – FROM SCRATCH

If you decide to write your own **platform from scratch** (e.g., based on the advgrtmp8), a passing grade requires:

- implementing a generic and extendible architecture for a ray tracer;
- a 'free camera' with configurable position, orientation, FOV and aspect ratio (configurable: from code, or at runtime using some interface);
- a basic UI or controls to control the camera at run-time;
- support for at least planes and spheres;
- a basic but extendible material class;
- a basic scene consisting of a small set of these primitives;
- a Whitted-style ray tracing renderer, running on the CPU, supporting shadows, reflections and refraction with absorption (Beer's law).

To obtain additional points, you may work on the following:

1. Support for triangle meshes, using 'obj', 'gltf' or 'fbx' files to import scenes (max 1pt).
2. Support for complex primitives, complex being a torus or better (max 1pt).
3. Texturing on all supported primitives, where the texture is a generic bitmap (max 1pt).
4. Spots and directional lights (max 0.5 pts).
5. Anti-aliasing (max 0.5 pts).
6. Barrel distortion and a fish eye lens (max 0.5 pts).
7. Image postprocessing: gamma correction, vignetting and chromatic aberration (max 0.5 pts).
8. A multi-threaded rendering system that yields at least a 4x speedup (max 1pt).

OPTION 3 – VOXEL TEMPLATE

And finally, if you decide to use the **voxel template** as your starting point, a passing grade requires:

- implementing a generic and extendible architecture for a ray tracer;
- a 'free camera' with configurable position, orientation, FOV and aspect ratio (configurable: from code, or at runtime using some interface);
- a basic UI or controls to control the camera at run-time;
- a way to handle materials in the voxel engine;
- a basic but extendible material class;
- a basic scene consisting of a small set of these primitives;
- a Whitted-style ray tracing renderer, running on the CPU, supporting shadows, reflections and refraction with absorption (Beer's law).

To obtain additional points, you may work on the following:

1. Support for primitives 'on top of' the voxel world: a sphere (0.5pt) **or** complex primitives (complex being a torus or better, max 1pt).
2. Textured voxels (Minecraft style), with at least 8 unique textured blocks (max 1pt).
3. Spots and directional lights (max 0.5 pts).
4. Anti-aliasing (max 0.5 pts).
5. Barrel distortion and a fish eye lens (max 0.5 pts).
6. Image postprocessing: gamma correction, vignetting and chromatic aberration (max 0.5 pts).
7. A multi-threaded rendering system that yields at least a 4x speedup (max 1pt).

Regardless of the chosen option:

Additional bonuses may apply; please discus with me to be sure.

IMPORTANT: if you fail the minimum requirements, you are not eligible for additional points.

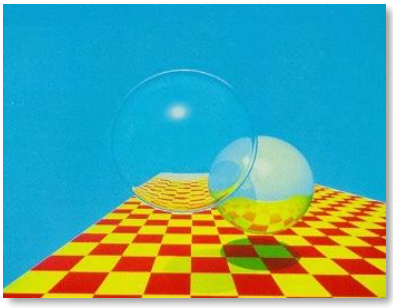
Your grade will be clamped to the range 1..10. Not handing in anything yields 0 points.

Academic Integrity

The work you hand in must be your own work. If you use materials from others (source code, libraries) please state this clearly in your report. If your work is mostly a copy of online sources, it cannot be graded.

Purpose

We will use the result of this assignment in the second assignment. The code you produce should therefore be reusable.



May the Light be with you,

- Jacco.