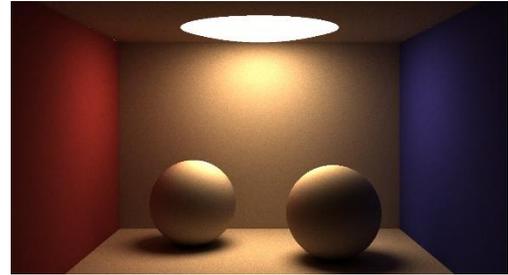
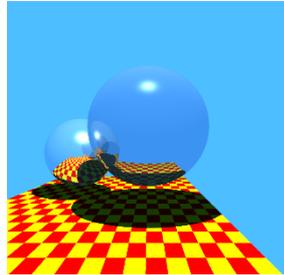
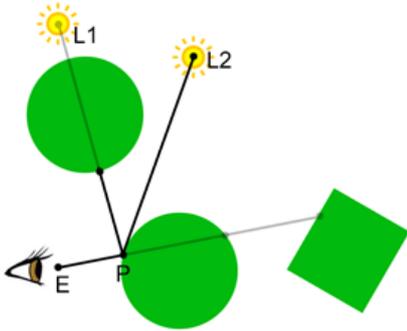


# Advanced Graphics 2020/2021 – Assignment 1



## 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 basic Whitted-style ray tracer and a Kajiya-style path tracer for validation.

Right from the start you have two main options to pick from. 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](#). 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. The second option lets you skip learning how Lighthouse 2 works, but this forces you to do more low-level work yourself. Either option is a rewarding experience.

## Architecture

In a nutshell, a ray tracer (as well as a path 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/Path Tracing

To demonstrate the suitability of your framework, implement a **Whitted-style ray tracer**. The Whitted-style ray tracer must support shadows, reflections and dielectrics. Implementation of Beer's law is recommended, but, for now, optional. Note that a Whitted-style ray tracer does **not** support area lights and glossy reflections. Illumination from area lights is thus not supported.

Besides the Whitted-style ray tracer, implement a basic **path tracer**. The path tracer uses a random walk to find light sources. Since the probability of encountering a point light this way is zero, the path tracer **only** supports area lights (for now).

### ***Practical Details***

The deadline for this assignment is Wednesday December 2<sup>nd</sup>, 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 December 3<sup>rd</sup>, 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. 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.

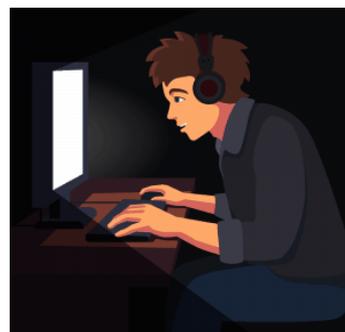
You may work on this assignment alone, or with one other student. I recommend working together, especially in these trying times: you learn more, and it's probably healthier as well. Note that you are not required to work in the same team for all three assignments.

You may implement your platform in C++ or C# or any other programming language that you may prefer. Do note however 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.

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'.



IV



## **Tasks & Grading**

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;
- a Kajiya-style path tracing renderer, running on the CPU, supporting shadows from area lights, reflections and refraction.

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 (0.25 pts).
3. Dielectrics with absorption using Beer's law (0.5 pts).
4. Anti-aliasing (0.25 pts).
5. Barrel distortion and a fish eye lens (0.5 pts).
6. Image postprocessing: gamma correction, vignetting and chromatic aberration (0.5 pts).

If you decide to write your own platform from scratch, 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;
- 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;
- a Whitted-style ray tracing renderer, running on the CPU, supporting shadows, reflections and refraction.

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

1. Support for triangle meshes, using 'obj' or 'fbx' files to import scenes (max 0.5pt).
2. Support for complex primitives, complex being a torus or better (max 0.5pt).
3. Texturing on all supported primitives, where the texture is a generic bitmap (max 0.5pt).
4. Spots and directional lights (0.25).
5. Dielectrics with absorption using Beer's law (0.5 pts).
6. Anti-aliasing (0.25 pts).
7. Barrel distortion and a fish eye lens (0.5 pts).
8. Image postprocessing: gamma correction, vignetting and chromatic aberration (0.5 pts).

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

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

Your grade will be capped at 10.

### ***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.

