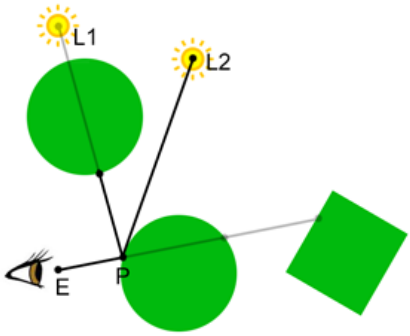
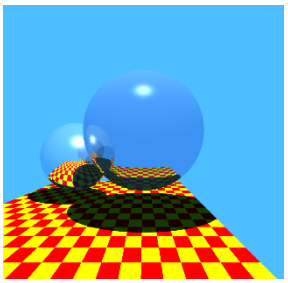
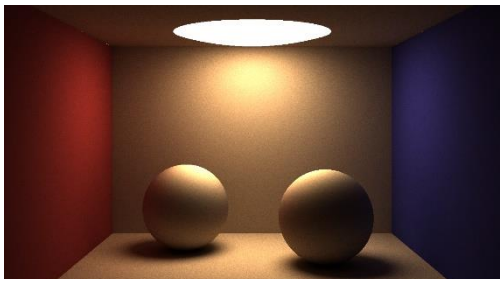
Advanced Graphics 2022/2023 – Assignment 1







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 two renderers for validation: a deterministic Whitted-style ray tracer, and a stochastic Kajiya-style path tracer.

Right from the start you have two main options to pick from, as discussed in the first lecture. You may implement this testbed either ***(1)*** using the provided ray tracing template, or ***(2)*** as a stand-alone application, e.g. built on top of an empty C/C++ template, or in a different language than C/C++. If you chose the first option, the framework takes care of a lot of mundane tasks, which allows you to focus on the essence of the assignment. The second option forces you to do more low-level work yourself - but may lead to a product that feels more like your own.

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. So far so good; you'll basically find just that in the ray tracing template.

Not in the template: a lot of needful things. For starters, you will need a *user-configurable camera*, perhaps one that also supports depth of field. The framework also needs to support a currently unavailable primitive: the *triangle*. Groups of triangles form *meshes*, support for which is needed for assignment 2. For basic light transport you'll need *light sources*, in various forms. *Materials* also come in handy; these encapsulate surface properties: reflection, color, a texture perhaps.

For light transport itself some standard ingredients are required: handling *reflection* and *transmission*, as well as *paths* that consist of ray segments.

If you're feeling adventurous it may be worthwhile to add *"Dear ImGui"* to your project, to control settings at run-time. And perhaps *GPGPU* is your thing: this would obviously speed up rendering significantly, especially if you have a decent GPU. This is all very optional, obviously.

Ray Tracing

To demonstrate the suitability and completeness 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 correctly implement 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, and neither is depth of field.

Path Tracing

With a working Whitted-style ray tracer, making the move to full path tracing should be straight-forward: the two renderers share a lot of functionality. For the path tracing renderer, add support for diffuse bounces and replace the point lights in your scene by area lights.

Practical Details

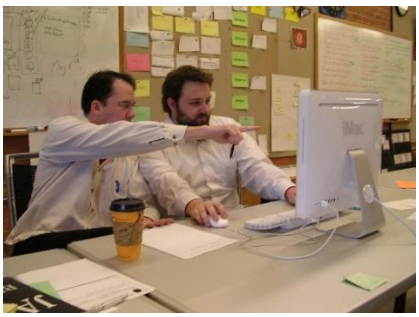
The deadline for this assignment is Wednesday December 7th, **17:00** (note the unusual time: this is for your health). An extended deadline is also available: in exchange for a 1 point grade reduction you can hand in materials until Thursday December 8th, **17:00**.

The materials to submit are:

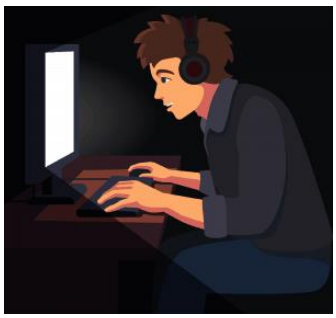
- your project, including sources and build instructions (if these are not obvious);
- a brief report, detailing implemented functionality, division of work, references and other information relevant to grading your submission.

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 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. If there's still a .vs folder, 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, it's more fun and probably healthier as well. Note that you are not required to work in the same team for all three assignments. Do be gentle if you split up - but don't stay in a non-functional team: both of you will be held equally accountable if things go south.



$\geq$



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'.

Tasks & Grading

Time for the formal details. A passing grade requires:

- Implementing a generic and extendible architecture for a ray tracer (mostly already there if you go for the ray tracing template).
- a 'free camera' with configurable position, orientation, FOV and aspect ratio. 'Configurable' means: adjustable from code, or at runtime using some interface.
- A basic user interface, or keys / mouse input handling to control the camera position and orientation at run-time.
- Support for (at least) planes, spheres and triangles.
- An extendible material class.
- A basic scene, consisting of a small set of these primitives. Use each primitive type at least once.
- A Whitted-style ray tracing renderer, supporting shadows, reflections and refraction with absorption (Beer's law).
- A Kajiya-style path tracing renderer, supporting diffuse interreflections and area lights.

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

1. Support for triangle meshes of arbitrary size, using 'obj', 'gltf' or 'fbx' files to import scenes (max 1pt). The .obj file format is highly recommended for this assignment.
2. Support for complex primitives - complex being a torus or better (max 1pt).
3. Texturing on all supported primitives. Texture must be a bitmap loaded from a file (max 1pt).
4. Besides the point lights: Spot lights and directional lights (max 0.5 pts).
5. Anti-aliasing (max 0.5 pts).
6. A skydome, with a texture loaded from a HDR file format (max 0.5 pts).
7. Barrel distortion, a Panini-projection and/or a fish eye lens (max 0.5 pts).
8. Image postprocessing: gamma correction, vignetting and chromatic aberration (max 0.5 pts).

IMPORTANT: The additional features will not be considered for grading if the requirements for a passing grade are not met. Also note that your grade will be capped at a 10.

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

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.