CS171 Assignment 3: Ray Tracing with Direct Lighting

Introduction

In this assignment, you are required to implement the basic ray-tracing with direct illumination using the Phong lighting model. In the following, we will give you the specifics about what you need to accomplish, as well as some related guidelines in order to assist your programming.

Programming Requirements

- **[must]** You are required to implement a pin-hole camera model, which is able to shoot a set of optical rays. And there should be at least one ray per pixel. [30%]
- **[must]** You are required to implement the algorithm for the ray-triangle intersection (without the acceleration structure). [20%]
- **[must]** You are required to implement the algorithm for the ray-cube intersection based on the ray-triangle intersection (without the acceleration structure). [20%]
- **[must]** You are required to implement anti-aliasing for ray-tracing by using super-sampling with the rotated grid. [30%]
- **[optional]** You may implement texturing. [10%]
- **[optional]** You may implement a mipmapping for texturing in ray tracing. [15%]
- **[optional]** You may implement normal mapping. [10%]
- **[optional]** You may implement environment lighting with importance sampling. [10%]

Notes

- Please notice that you are **NOT** allowed to use OpenGL in this assignment and **NOT** allowed to use third-party libraries except for the libraries we give in the skeleton code.
- As the computation in this assignment is quite heavy, we encourage you to use OpenMP for acceleration. You may apply for an account on the SIST HPC cluster, if necessary.
- We will offer a code skeleton for you. Please understand the functionality of each declared class and method in the code skeleton before you start.
- To verify your implemented algorithms, you can use various camera/light settings.

Submission

You are required to submit the following things through the GitHub repository:

- Project scripts and an executable program in the Coding folder.
- A PDF-formatted report which describes what you have done in the Report folder.

Submission deadline: **22:00, Nov 16, 2021**

Grading Rules

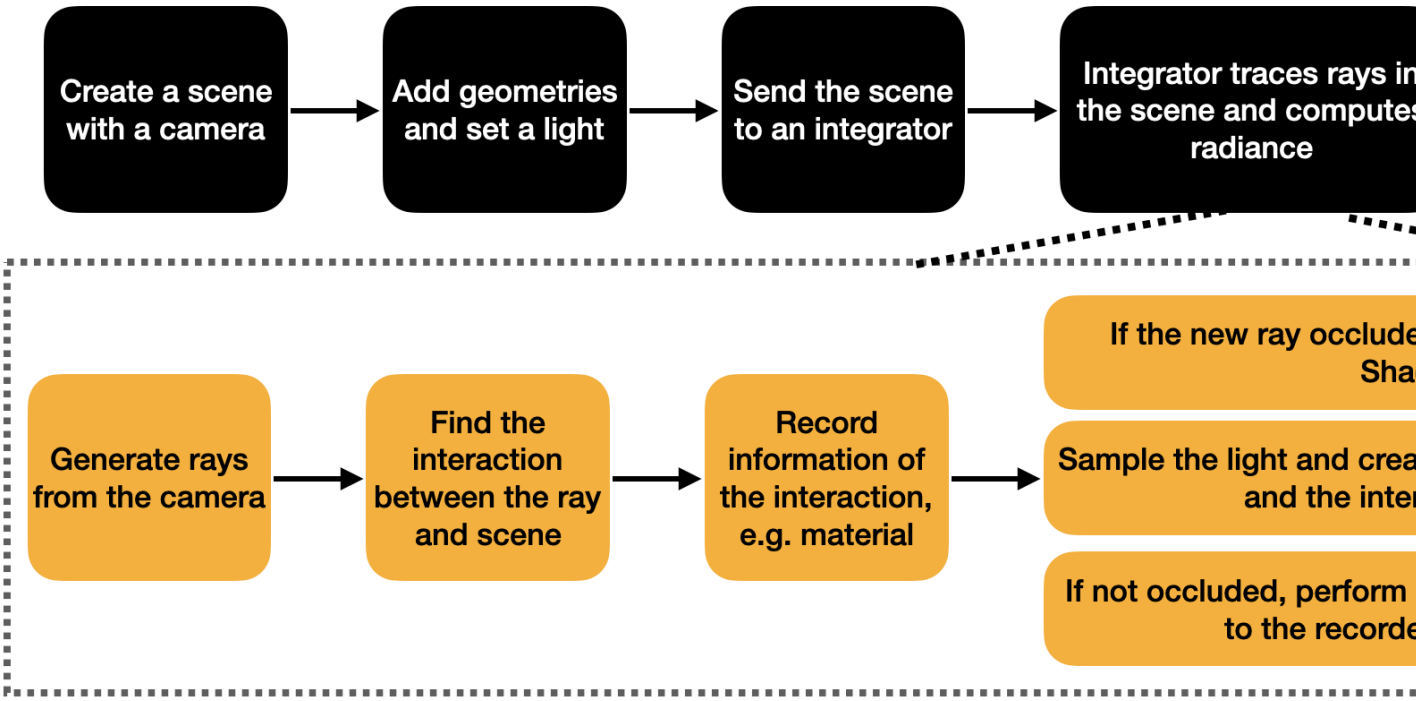
- You can choose to do the **[optional]** item, and if you choose to do it, you will get additional scores based on the additional work you have done. But the maximum additional score will not exceed 20% of the entire score of this assignment.
- **NO CHEATING!** If found, your score for the entire assignment is zero. You are required to work **INDEPENDENTLY**. We fully understand that implementations could be similar somewhere, but they cannot be identical. To avoid being evaluated inappropriately, please show your understanding of code to TAs.
- Late submission of your assignment will be subject to score deduction.

Skeleton Project/ Report Template

- The skeleton program and report template will be provided once you accept the assignment link of GitHub classroom which we published on the Piazza. If you accept the assignment through the link properly, a repository which contains the skeleton project and report template will be created under your GitHub account.
- Please **follow the template** to prepare your report.

Implementation Guide

We give you a brief bottom-up implementation guide, namely, the instructions below will first direct you to encapsulate some interfaces, and then combine them together to build a complete rendering pipeline. However, you can also follow your own understanding to complete each part in your own order. The figure below illustrates the basic working flow of building and rendering a scene.



Git Classroom

Accept the assignment in this link through Git classroom to start your assignment or you can download the zip package.

1. Pin-hole camera model

This part requires you to implement the `lookAt` and `generateRay` method of the `Camera` class. In the `lookAt` method, you need to build axes of camera according to the focal length and field of view (FOV). In `generateRay` method, you need to shoot camera rays for a specified pixel. The **site 1** provides a detailed description of shooting rays from cameras.

2. Ray-triangle intersection test

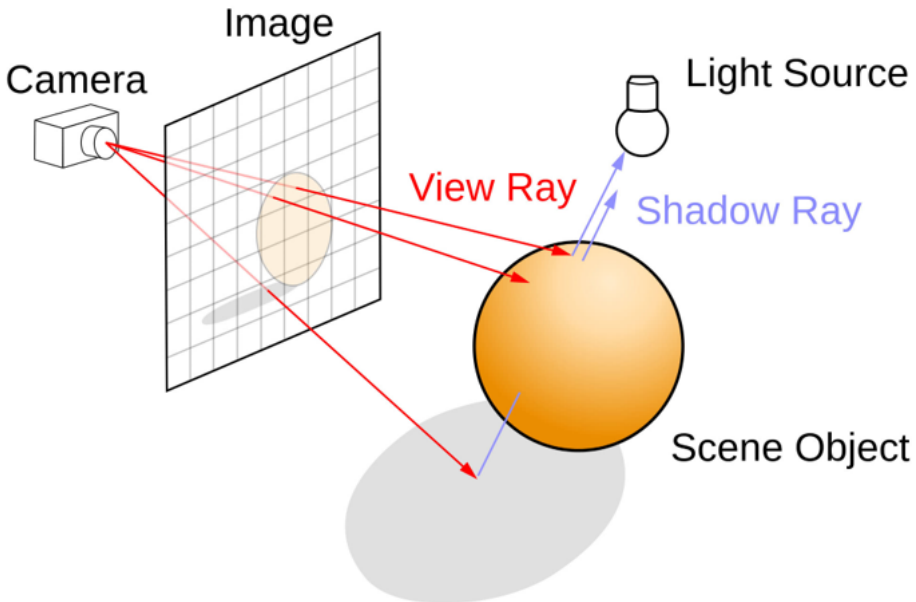
The ray intersection test with a triangle is a long-established technique. Here we will adopt Moller-Trumbore intersection algorithm, where the basic idea is to find a point satisfying both the equation of ray and equation of ray and the equation of triangle. The related method you need to fill in `Triangle::rayIntersection`. Please refer to **site 2** for the implementation details.

3. Area light

To enable area light, you need to complete the `AreaLight::samples` method. For simplicity, you can sample the area light uniformly in a grid pattern to form a collection of point light sources. For this purpose, the `samples` method is supposed to return a list of uniformly sampled point lights. Also note, if there are N samples on a uniform area light with total radiance of L, the radiance of each sample is L/N.

4. Phong lighting integrator

In this part, you are going to shoot rays for each pixel and compute the radiance brought back by those rays. The figure below concretely exhibits the direct lighting model. You can also follow the aforementioned working flow to write your integrator.



The code you need to complete is located in the `PhongLightingIntegrator::render` and `PhongLightingIntegrator::radiance` methods. Please recall the discrete case of rendering equation for direct lighting:

$$L_o(p, \omega_o) = \sum_{L_i} \text{brdf}(p) \cdot L_i \cos\theta_i \Delta A$$

where L_i's are sampled light sources. In this assignment, we approximate the BRDF term by using Phong model, i.e., we can compute the returned radiance at point p by L_i as $\text{diffusion}(L_i, p) + \text{specular}(L_i, p) + \text{ambient light}$.

For more details, please refer to **site 3**. You can also try Blinn-Phong lighting in this part.

5. Anti-aliasing with rotated grid pattern

Rotated grid: A rotated n×n grid layout for each pixel is used to avoid sample alignment on the axes to improve anti-aliasing quality. For an optimal pattern, the rotation angle is arctan(1/2) (about 26.6°)

For more details, please refer to **site 4**.

References

- Site 1: Scratch a pixel - Generating camera rays
- Site 2: Scratch a pixel - Ray-triangle intersection
- Site 3: Scratch a pixel - The Phong model and the concepts of Illumination models and BRDF
- Site 4: Computer Graphics I: Lecture 9 @ ShanghaiTech

Expected Results

We create a classic Cornell box with cubes inside to verify your implementation. Below we exhibit our expected results under different camera and light settings.