# Assignment 3: Basic Ray Tracing

NAME: CHEN YILE
STUDENT NUMBER: 2023533174
EMAIL: CHENYL2023@SHANGHAITECH.EDU.CN

## 1 INTRODUCTION

In this assignment, I complete all the must requirements and 2 optional requirements including implementing rectangular area lights with soft shadow generation and implementing environment lighting via environment maps.

## 2 IMPLEMENTATION DETAILS

### 2.1 Implement ray-triangle intersection functionality

To judge whether a ray intersects with a triangle, I need to ensure the following matrix equation has solution:

$$\begin{bmatrix} r_{1,0} & r_{2,0} & -d_0 \\ r_{1,1} & r_{2,1} & -d_1 \\ r_{1,2} & r_{2,2} & -d_2 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ t \end{bmatrix} = o - p$$

where p0, p1 and p2 are the three vertices on the triangle, o is the origin of the ray and d is the normalized direction of the ray. r1 = p1 - p0, r2 = p2 - p0.
To compute this equation, I can use Cramer's rule. By rewritting the equation as:

$$\begin{bmatrix} a_1, a_2, a_3 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ t \end{bmatrix} = b$$

I have the solution: $\begin{cases} u &= \frac{b\cdot(a_2\times a_3)}{a_1\cdot(a_2\times a_3)} \\ v &= \frac{a_1\cdot(b\times a_3)}{a_1\cdot(a_2\times a_3)} \\ t &= \frac{a_1\cdot(a_2\times b)}{a_1\cdot(a_2\times a_3)} \end{cases}$

(If $a_1\cdot(a_2\times a_3) = 0$ the equation doesn't have a solution.) And it must be ensured that $u \geq 0, v \geq 0, u + v \leq 1, and, ray.t_{min} \leq t \leq ray.t_{max}$

### 2.2 Implement ray-AABB intersection functionality.

To judge whether a ray intersects with a AABB, I should compute the intersection of the ray and each planes of the box by groups (the planes which are parallel with x-y plane, x-z plane and y-z plane are divided into 3 groups respectivtely). And for each group, I compute the range of $t$ for the ray $o + t \cdot d$ is betIen the 2 planes. And if the ranges of 3 groups's intersection set is not empty, the ray is intersecting with the AABB.

### 2.3 Implement the BVH (Bounding Volume Hierarchy) construction.

In this assignment, BVH tree is a data structure that used to store the information of the primitives of each mesh faces. Each leaf node stores the primitive triangle and its AABB information, and a normal node stores the information of a AABB that exactly including the AABBs of all the children nodes. To build a such a BVH tree, before the stop criteria is satisfied (the left space exactly 1 node or the depth of the tree reach its maximum number (a constant)), I divided the remaining nodes into left and right children trees averagely . And because the nodes is exactly stored in a linear array and the tree just stores their indices, I need to sort the nodes by one dimension of the center of their AABBs, and the node whose that dimension of the center of its AABB is the median one is the boundary of 2 children trees.

### 2.4 Implement the IntersectionTestIntegrator and PerfectRefraction material for basic ray tracing validation, handing refractive and solid surface interaction.

In this assignment, to implement perfect refraction, I should set the 'interaction.wi' to the direction of the "in-coming light" after refraction or reflection. When the dot product of the direction of 'out-going light' and the normal of the surface is positive, which means the light intersected the surface in the side that can refract lights, I should implement refract. Otherwise I should implement reflact.
And after the refraction , I should set the ray's direction of the direction of the "in-coming light" too.

### 2.5 Implement a direct lighting function with diffuse BRDF and shadow testing.

In this assignment, for the sample surface interaction, I generate a test ray whose origin is the position of the interaction and direction is set as the direction from the position of the interaction to the position of the point light.
By using the BVH tree above, I can check whether the test ray is intersecting with some primitives (which means the light is occluded.). If so, the conrtibution of the light for the sample surface interaction is nothing, represented by Vec(0,0,0).
If the test ray is not occluded, then I can compute the contribution of the interaction, this assignment only required us to accomplish a simple phong-shading-like model and use the approximation of albedo. I set the color as the product of the albedo and the given flux of the point light divided by $4\pi$ multipling the square of the distance between the 2 points.

student number: 2023533174
email: `chenyl2023@shanghaitech.edu.cn`

## 2.6 Implement anti-aliasing via multi-ray sampling per pixel within a sub-pixel aperture.

To achieve the requirement, I use the given sampler to get pixel sample and create the diffrential ray based on the coordinate of the pixel sample.

## 2.7 Implement rectangular area lights with soft shadow generation.

For each area light, I sample N points and treat every point as a point light, which can result in soft shadow generation, and color contributions are set to be the product of the albedo and the given flux of the point light divided by light pdf and sample number. What's more, for the area light object itself, when the ray of camera intersects with it, the color will be set to the radiance of the area light divided by the maximum of its 3 dimensions.

## 2.8 Implement environment lighting via environment maps.

When implement environment, I need to sample at the direct lighting part, but the thing I sample isn't a position, but the directions because the environment map isn't a object and its depth is infinity. For environment light, I also need to generate a test ray whose origin is the position of interaction and direction is the negative direction of sample direction, if the test ray intersects with some primitives, then the environment light is occluded and has no contribution. What's more, if the camera light doesn't intersect with anything, its color will be set as the color of the material of environment map.

## 3 RESULTS

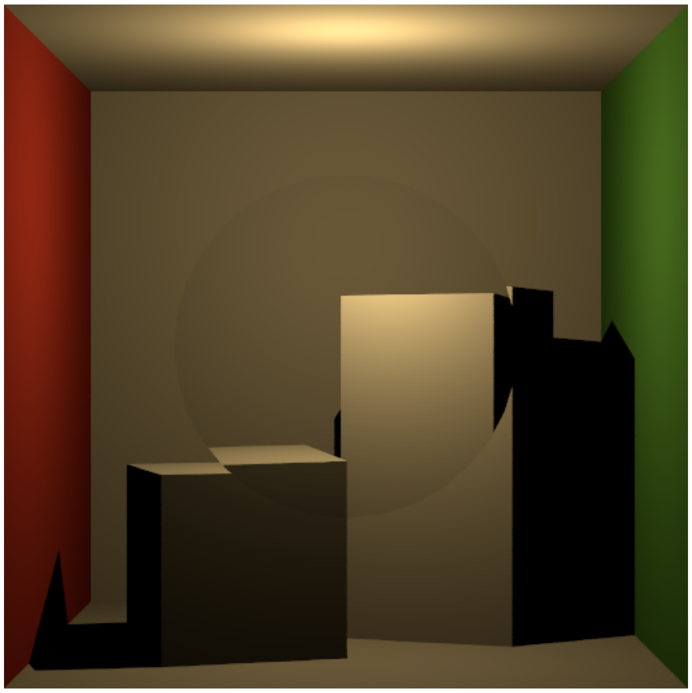
Fig. 1. cbox_no_light_refraction.json no light, perfect refract, spp = 32, 512*512

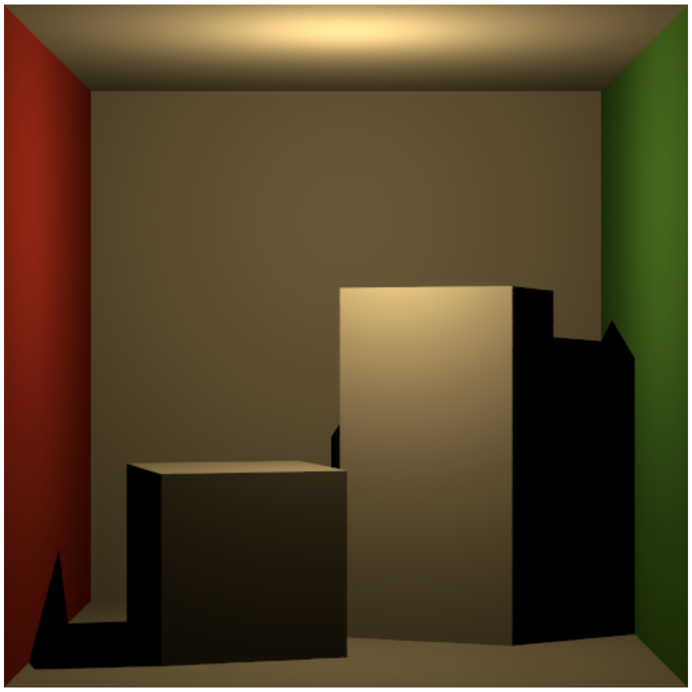
Fig. 2. cbox_no_light.json no light, spp = 32, 512*512

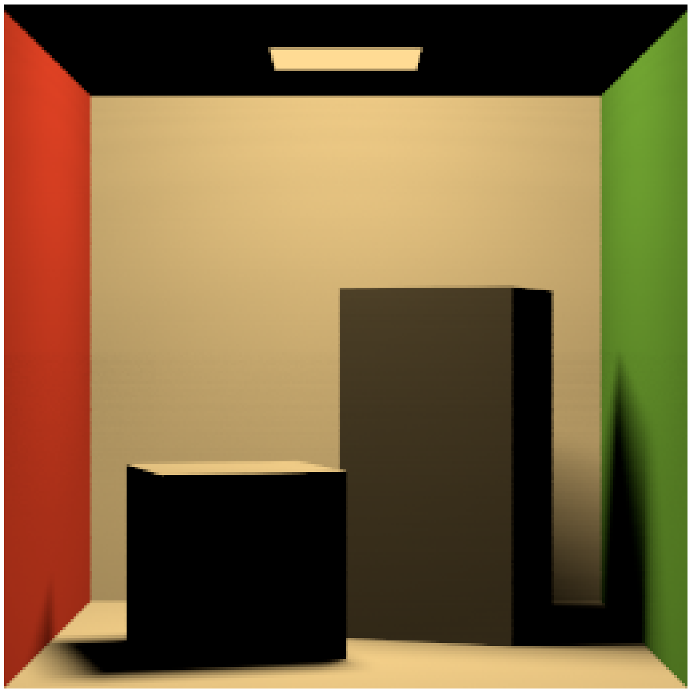

Fig. 3. cbox_area_light.json rectangule area light, spp = 128, 256*256

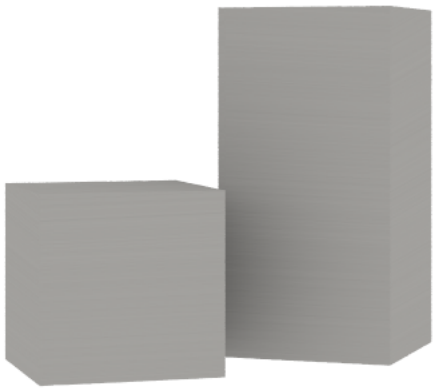

Fig. 4. cbox_env_light.json environment light (white background), spp = 128, 512*512