Assignment 3:
Basic Ray Tracing

NAME:YANG YEXUAN
STUDENT NUMBER:2023533148
EMAIL: YANGYX2023@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, I implemented a basic ray-tracing renderer with several core components. The renderer is built on a CPU-based offline rendering framework and does not rely on GPU acceleration or OpenGL.

The key features implemented include:

- Ray-primitive intersection testing for triangles and AABBs
- Bounding Volume Hierarchy (BVH) for efficient scene traversal
- Direct lighting with diffuse BRDF and shadow ray testing
- Perfect refraction material for transparent objects
- Anti-aliasing through stratified sampling (32 samples per pixel)
- Rectangular area light sources for soft shadow effects.
- Apply new texture to the objects, walls and floor.

The final renderer successfully renders the Cornell Box scene with realistic lighting, shadows, and refraction effects. The output images have a resolution of 512×512 pixels with 32 samples per pixel, ensuring high-quality anti-aliased results.

2 IMPLEMENTATION DETAILS

2.1 Ray-Triangle Intersection

The ray-triangle intersection is implemented in the **accel.cpp**, which is very important. About the algorithm, I use the **Möller-Trumbore** algorithm, which has the main equation:

$$\mathbf{O} + t\mathbf{D} = (1 - u - v)\mathbf{V}_0 + u\mathbf{V}_1 + v\mathbf{V}_2 \tag{1}$$

2.2 Ray-AABB Intersection

2.2.1 Algorithm. About the AABB intersection, we consider it in the following ways.

- For each axis, compute t values where ray intersects the two planes.
- t_{near} = max(all entering times)
- t_{far} = min(all exiting times)
- Intersection exists if t_{near} <= t_{far} and t_{far} >= 0.

2.3 BVH Construction

The BVH is constructed in the `TriangleMesh` constructor:

```
accel = make_ref<BVHAccel>();
accel->setTriangleMesh(mesh.get());
accel->build();
```

The Surface Area Heuristic minimizes:

$$\text{Cost} = C_{\text{trav}} + \frac{A_L}{A_N}C_L + \frac{A_R}{A_N}C_R \tag{2}$$

2.4 IntersectionTestIntegrator

Thie part is the main ray tracing loop:

```
Vec3f IntersectionTestIntegrator::Li(
ref<Scene> scene, DifferentialRay &ray,
Sampler &sampler) const {

for (int i = 0; i < max_depth; ++i) {
bool intersected =
scene->intersect(ray, interaction);

if (is_perfect_refraction) {
interaction.bsdf->sample(interaction, sampler, &pdf);
ray = interaction.spawnRay(interaction.wi);
continue;
}
if (is_ideal_diffuse) {
return directLighting(scene, interaction, sampler);
    }
}
}
```

The `Li()` function traces a ray through the scene, handling specular surfaces (like glass) by refraction, and computing direct lighting when a diffuse surface is encountered.

Snell's law:

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t \tag{3}$$

2.5 Direct Lighting

Rendering equation:

$$L_o = \int_\Omega f_r L_i(\omega_i \cdot \mathbf{n})\, d\omega_i \tag{4}$$

where:

- f_r: BRDF (Bidirectional Reflectance Distribution Function)
- $\boldsymbol{\omega}_L = \frac{\mathbf{p}_L - \mathbf{p}}{\|\mathbf{p}_L - \mathbf{p}\|}$: normalized direction to light
- $r = \|\mathbf{p}_L - \mathbf{p}\|$: distance to light
- $\frac{\Phi}{4\pi r^2}$: irradiance from point light (**inverse square law**)
- $\max(\boldsymbol{\omega}_L \cdot \mathbf{n}, 0)$: **Lambert's cosine law**
- $V(\mathbf{p}, \mathbf{p}_L)$: visibility (1 if visible, 0 if occluded)

student number:2023533148
email: yangyx2023@shanghaitech.edu.cn

Implementation with shadow testing, the most important part is BRDF.

Lambertian BRDF:

$$f_r = \frac{\rho}{\pi} \tag{5}$$

2.6 Anti-aliasing

Stratified sampling with 32 spp:

```
for (int sample = 0; sample < spp; sample++) {
const Vec2f &pixel_sample =
      sampler.getPixelSample();
auto ray = camera->generateDifferentialRay(
    pixel_sample.x, pixel_sample.y);
const Vec3f &L = Li(scene, ray, sampler);
camera->getFilm()->commitSample(pixel_sample, L);
}
```

Final pixel color:

$$C = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{r}_i) \tag{6}$$

2.7 Rectangular Area Lights

Rectangle primitive implementation:

```
class Rectangle : public Shape {
private:
    Vec3f center;       // Rectangle center
    Vec3f edge1;        // First edge vector
    Vec3f edge2;        // Second edge vector
    Vec3f normal;       // Surface normal
    Float area_value;   // Precomputed area
};
```

Some formulas are as follows.

Area computation:

$$A = \|\mathbf{e}_1 \times \mathbf{e}_2\| \tag{7}$$

Uniform sampling:

$$p_A(\mathbf{x}) = \frac{1}{A} \tag{8}$$

Soft shadow via Monte Carlo:

$$L = \frac{1}{N}\sum_{i=1}^{N} V(\mathbf{x}_i) \cdot f_r \cdot L_e \cdot G \tag{9}$$

2.8 texture mapping

I simply find some images on line, then I apply the texture by creating a new json-file. And the result is below.

2.9 Conclusion

This assignment successfully implements a complete ray-tracing renderer with all required features and additional optional components. The implementation demonstrates:

(1) Solid understanding of ray-primitive intersection algorithms
(2) Efficient scene traversal using BVH acceleration
(3) Physically-based lighting with BRDF evaluation
(4) Advanced features including soft shadows and refraction

3 RESULTS

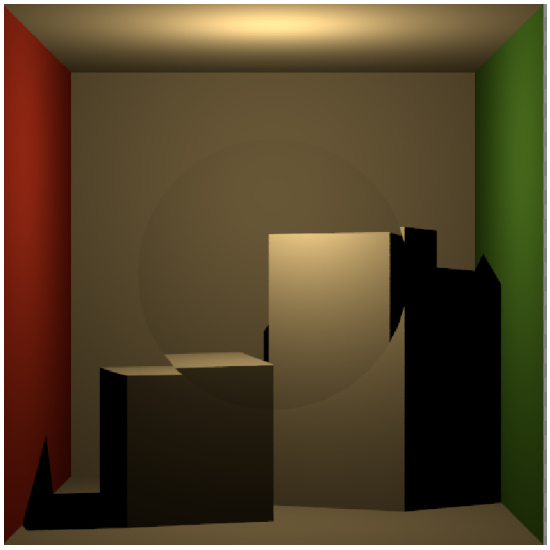

Fig. 1. render 1

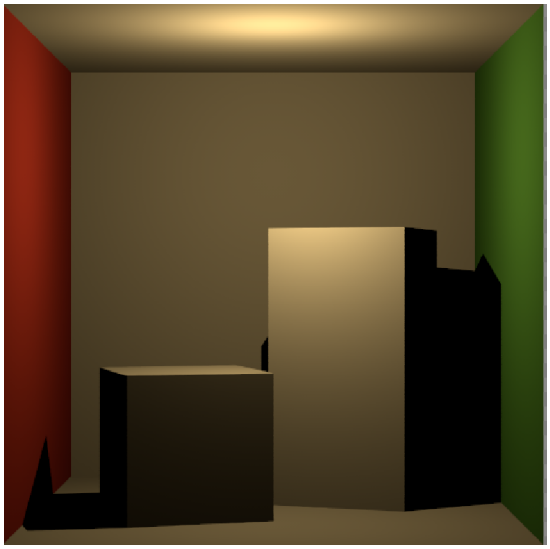

Fig. 2. render without refraction

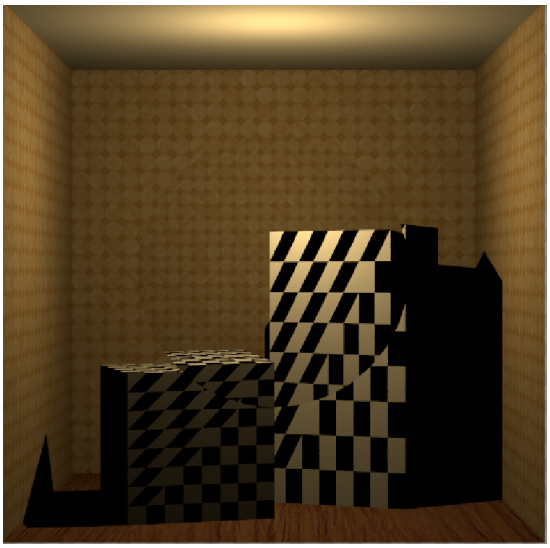

Fig. 3. render with texture

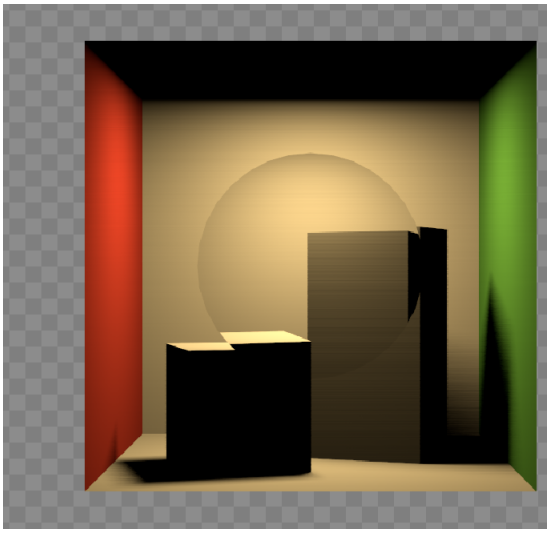

Fig. 4. render with rectangle area lights