Assignment 3: Basic Ray Tracing

NAME: YE HAOXU
STUDENT NUMBER: 2023533137
EMAIL:

1 INTRODUCTION

This assignment implements a basic ray tracing renderer in C++ for CS171. The renderer supports essential features including bounding volume hierarchies (BVH) for acceleration, ray-triangle and ray-AABB intersections, direct lighting with shadow rays, and refraction through transparent materials using Snell's law. The implementation is based on the provided framework and includes an IntersectionTestIntegrator for rendering scenes with diffuse and refractive objects. Scenes are defined in JSON format, and the renderer outputs EXR images for visualization.

2 IMPLEMENTATION DETAILS

2.1 Bounding Volume Hierarchy (BVH)

The BVH is implemented in `include/rdr/bvh_tree.h` and `src/bvh_accel.cpp`. The `BVHTree::build` function constructs the tree using a median heuristic for splitting along the longest axis. The stop criteria are a maximum depth of 20 and a minimum of 4 primitives per leaf. The build process recursively partitions primitives into left and right subtrees.

2.2 Ray-Primitive Intersections

Ray-triangle intersections are implemented using the Möller-Trumbore algorithm in `src/accel.cpp`, function `TriangleIntersect`. It computes barycentric coordinates and checks for valid intersections within the time window.

Ray-AABB intersections use the slab method in `AABB::intersect`, also in `accel.cpp`. It finds the entry and exit times along each axis and determines if the ray intersects the box.

2.3 Integrator

The `IntersectionTestIntegrator` in `src/integrator.cpp` handles direct lighting and refraction. The `Li` function traces rays, handling refraction through transparent objects and computing direct lighting from point lights with shadow rays.

For refraction, it uses the `PerfectRefraction` BSDF, which implements Snell's law. The `sample` function computes the refracted direction based on the incident direction, normal, and refractive indices. It handles entering and exiting media correctly.

Direct lighting computes the contribution from point lights, checking for occlusion with shadow rays.

2.4 Scene and Rendering

Scenes are loaded from JSON files in `data/`. The renderer uses OpenMP for parallel rendering and outputs EXR images. Anti-aliasing is implemented by supersampling.

3 RESULTS

3.1 Tests

All mandatory tests pass, including `intersection_tests` and `bvh_tests`. The BVH correctly accelerates ray tracing, and intersections handle time windows properly. Refraction is implemented with correct Snell's law application, avoiding total internal reflection issues.

3.2 Rendering

The renderer produces correct EXR outputs for various scenes:

- `cbox_no_light.exr`: Cornell box with diffuse surfaces and direct lighting. - `cbox_no_light_refract.exr`: Includes refraction through glass spheres. - Other scenes like `glass_sphere.exr` demonstrate refraction and lighting.

Renders show proper shadows, reflections, and refraction. Lighting intensities were adjusted in JSON files (e.g., `point_light_flux` set to [8.5, 6.0, 2.5]) for realistic appearance.

The implementation successfully completes all HW3 requirements.