Assignment 1: Exploring OpenGL Programming

NAME: CHENGYANG LI
STUDENT NUMBER: 2022533154
EMAIL: LICHY22@SHANGHAITECH.EDU.CN

1 INTRODUCTION

[must] Compile the source code and configure the language server environment.

[must] Implement ray-triangle intersection functionality.

[must] Implement ray-AABB intersection functionality.

[must] Implement the BVH (Bounding Volume Hierarchy) construction.

[must] Implement the IntersectionTestIntegrator and PerfectRefraction material for basic ray tracing validation, handing refractive and solid surface interactions

[must] Implement a direct lighting function with diffuse BRDF and shadow testing.

[must] Implement anti-aliasing via multi-ray sampling per pixel within a sub-pixel aperture.

2 IMPLEMENTATION DETAILS

This section provides detailed descriptions of the implementation locations and core algorithms for each task.

2.1 Ray-Triangle Intersection (10%)

Implementation Location: `src/accel.cpp`, lines 74-123, function `TriangleIntersect`

Algorithm Description: Implements ray-triangle intersection using the Möller-Trumbore algorithm. This algorithm directly computes barycentric coordinates and intersection distance by solving a linear system.

Key Steps:

- Compute triangle edge vectors $e_1 = v_1 - v_0$ and $e_2 = v_2 - v_0$
- Compute auxiliary vector $p = \text{dir} \times e_2$ and determinant $\det = e_1 \cdot p$
- Compute barycentric coordinates $u = (t_{vec} \cdot p)/\det$ and $v = (\text{dir} \cdot q)/\det$
- Compute intersection distance $t = (e_2 \cdot q)/\det$
- Verify boundary conditions: $u \geq 0, v \geq 0, u + v \leq 1, t \in [t_{min}, t_{max}]$

2.2 Ray-AABB Intersection (10%)

Implementation Location: `src/accel.cpp`, lines 32-75, function `AABB::intersect`

Algorithm Description: Implements ray-AABB intersection using the Slab method. Treats the AABB as the intersection of three pairs of parallel planes and computes the ray intersection with each pair.

Key Steps:

- Compute entry time t_0 and exit time t_1 for each of the X, Y, Z axes
- Handle cases where ray direction is negative (swap t_0 and t_1)
- Compute overall entry time $t_{in} = \max(t_{0x}, t_{0y}, t_{0z})$
- Compute overall exit time $t_{out} = \min(t_{1x}, t_{1y}, t_{1z})$
- Check intersection condition: $t_{out} \geq t_{in}$ and $t_{out} \geq 0$

2.3 BVH Construction (25%)

Implementation Location: `include/rdr/bvh_tree.h`, lines 127-214, function `BVHTree::build`

Algorithm Description: Implements recursive BVH tree construction using median-based heuristics for space partitioning.

Key Steps:

- **Termination Condition (lines 136-155):** Create a leaf node when the number of nodes ≤ 1 or depth $\geq$ CUTOFF_DEPTH
- **Median Split (lines 178-186):**
  - Select the longest dimension of the bounding box as the split axis
  - Use `std::nth_element` to perform median split based on centroid positions
  - Divide the node array into two parts: [span_left, split) and [split, span_right)
- Recursively build left and right subtrees
- Merge the bounding boxes of subtrees as the parent node's bounding box

2.4 IntersectionTestIntegrator (25%)

This task contains three sub-components:

2.4.1 *Multi-ray Sampling and Anti-aliasing.*

Implementation Location: `src/integrator.cpp`, lines 41-61, function `IntersectionTestIntegr`

Implementation Description:

- Generate spp (samples per pixel) rays for each pixel
- Use `Sampler::getPixelSample()` to obtain sampling positions with sub-pixel random offsets
- Generate differential rays and compute radiance
- Submit results to the film, which automatically performs Monte Carlo integration averaging

2.4.2 *Refraction Ray Tracing.*

Implementation Location: `src/integrator.c` lines 92-109, function `IntersectionTestIntegrator::Li`

Implementation Description:

- Use a loop to trace rays until hitting a diffuse surface or exceeding maximum depth

- Detect surface type (ideal diffuse, perfect refraction, or other)
- For perfect refraction surfaces, call the BSDF sampling function to obtain the refraction direction
- Use `spawnRay` to generate new rays to avoid self-intersection
- Continue tracing until finding a diffuse surface

2.4.3 Direct Lighting Computation. **Implementation Location:** `src/integrator.cpp`, lines 131-183, function `IntersectionTestIntegra`

Implementation Description:

- **Shadow Testing (lines 136-151):**
  - Cast shadow rays from the intersection point towards the light source
  - Check for occluders
  - If an occluder is found before the light source, return black (no lighting)
- **Lighting Computation (lines 159-180):**
  - Use Lambert diffuse model: $L = \Phi \cdot \rho \cdot \cos\theta/(4\pi d^2)$
  - Where Φ is light flux, ρ is albedo, θ is incident angle, d is distance
  - Obtain albedo through BSDF's `evaluate` function

2.5 Perfect Refraction Material (20%)

Implementation Location: `src/bsdf.cpp`, lines 90-114, function `PerfectRefraction::sample`

Algorithm Description: Implements physical simulation of perfect refraction and total internal reflection.

Key Steps:

- Determine whether the ray is entering or leaving the medium based on the incident direction
- Adjust the normal direction to ensure it points to the correct hemisphere
- Use Snell's law to compute the refraction direction: $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$
- Call the `Refract` function to attempt refraction
- If total internal reflection occurs ($\sin\theta_2 > 1$), use the `Reflect` function to compute the reflection direction
- Set `interaction.wi` to the computed direction

2.6 Anti-Aliasing (5%)

Implementation Location: Integrated in the `IntersectionTestIntegrator::render` function

Implementation Description: Anti-aliasing is achieved through multiple random samples within each pixel. The sampler automatically generates random offsets within the pixel range $[x, x+1) \times [y, y+1)$, and the film automatically averages the results of multiple samples, effectively reducing aliasing artifacts.

3 RESULTS

This section presents the rendering results of the implemented ray tracer.

Fig. 1. Cornell box scene with a refractive glass sphere. The scene demonstrates ray tracing with refraction, direct lighting, and anti-aliasing.