Assignment 3: Basic Ray Tracing

NAME:
STUDENT NUMBER:
EMAIL:

ACM Reference Format:
Name: , student number:, email: . 2025. Assignment 3: Basic Ray Tracing. 1, 1 (November 2025), 1 page. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

This assignment walks through every must-have component of a basic ray tracer: geometric intersections, BVH construction, direct lighting with shadow testing, anti-aliasing, and perfect refraction. The project is configured with CMake; running `cmake -B build` followed by `cmake –build build` produces all executables and unit tests. The development environment is backed by `clangd`, ensuring the "compile and configure language server" requirement is satisfied.

2 Implementation Details

2.1 Ray–Geometry Intersections

Ray–AABB: In `src/accel.cpp` I replaced the `UNIMPLEMENTED` stub inside `AABB::intersect`. The final code: (1) subtracts the ray origin from the bounding-box bounds, (2) multiplies by `ray.safe_inverse_direction` to obtain parametric times on each axis, (3) uses `Min/Max` to gather the entry/exit time vectors, (4) folds them with `std::max/std::min` to get scalar $t_{\text{in}}/t_{\text{out}}$, and (5) clamps them against `ray.t_min/t_max`. The function writes both times through the provided pointers and returns whether $t_{\text{out}} \geq t_{\text{in}}$.

Ray–Triangle: The `TriangleIntersect` routine now performs a double-precision Möller–Trumbore solve. I compute edges (`edge1`, `edge2`), the cross products, determinant, and inverse determinant. The barycentric coordinates u and v are rejected when they fall outside $[0, 1]$ or $u + v > 1$. The hit distance t is compared against the ray time interval via `ray.withinTimeRange`. On success, I call `CalculateTriangleDifferentials`, assert the hit point matches `ray(t)`, and shrink `ray.t_max` to the intersection distance so that later triangles cannot overwrite a closer hit.

2.2 BVH Construction

The recursive helper `BVHTree::build` inside `include/rdr/bvh_tree.h` was also a stub. I added: (1) a leaf criterion based on span size ≤ 2 or depth $\geq$ `CUTOFF_DEPTH`, (2) axis selection using `ArgMax(prebuilt_aabb.getExte` (3) median partitioning with `std::nth_element` so that elements in $[L, \text{split})$ have smaller centroids on that axis, and (4) recursive construction of child indices followed by storing the precomputed AABB in the current node. The final tree shares the same node array as the primitives, which matches the framework's expectations.

2.3 IntersectionTestIntegrator and Anti-Aliasing

In `IntersectionTestIntegrator::render` I inserted the missing sampling loop: for each pixel I call `Sampler::getPixelSample()`, pass the jittered coordinates to `Camera::generateDifferentialRay`, evaluate radiance via `Li`, and feed the sample to the film through `commitSample`. This realizes Monte Carlo anti-aliasing with `spp` samples per pixel. Within `Li`, whenever the hit material is `PerfectRefraction`, I call `interaction.bsdf->sample()` to obtain the refracted (or reflected) direction and set `ray` to the spawned ray returned by `interaction.spawnRay`. The loop continues until the path hits a diffuse surface or escapes the scene.

2.4 Direct Lighting and Perfect Refraction

The direct-lighting function now spawns a shadow ray using `interaction.spawnR` and calls `scene->intersect`; any hit immediately returns zero contribution. Otherwise I set `interaction.wi` to the normalized light direction, evaluate the diffuse BSDF (`bsdf->evaluate`), multiply by the cosine term `Dot(light_dir, normal)` and by the inverse-square distance scaled with the point-light flux. In `src/bsdf.cpp`, `PerfectRefraction::sample` determines whether the ray is entering or leaving, adjusts η, calls `Refract` to compute the transmitted direction, and falls back to `Reflect` when total internal reflection occurs; the resulting `interaction.wi` is normalized before returning to the integrator.

3 Results

I rendered the Cornell-box configuration using `./build/src/renderer data/cbox_no_light_refract.json -o cbox_no_light_refract.exr`. The output EXR captures the expected shadows and the glass pane refraction; it can be viewed directly with an EXR viewer or converted to PNG for inclusion in the final write-up.

Author's Contact Information: Name:
student number:
email:

ACM XXXX-XXXX/2025/11-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn