Assignment 3:
Ray Tracing Fundamentals

NAME: WENHAOZHOU
STUDENT ID: 2022533140
EMAIL: ZHOUWH2022@SHANGHAITECH.EDU.CN

1 INTRODUCTION

This assignment implements the following components:

[must] Compile the source code and configure the language server environment. [5%]

[must] Implement ray-triangle intersection functionality. [10%]

[must] Implement ray-AABB intersection functionality. [10%]

[must] Implement the BVH (Bounding Volume Hierarchy) construction. [25%]

[must] Implement the IntersectionTestIntegrator and PerfectRefraction material for basic ray tracing validation, handing refractive and solid surface interactions [25%]

[must] Implement the DirectLightingIntegrator for direct lighting with diffuse BRDF and shadow testing. [20%]

[must] Implement anti-aliasing via multi-ray sampling per pixel within a sub-pixel aperture. [5%]

2 IMPLEMENTATION DETAILS

2.1 Ray-Triangle Intersection

The ray-triangle intersection is implemented in the `TriangleIntersect` function in `src/accel.cpp`. This function uses geometric methods to compute the intersection point between a ray and a triangle mesh.

```
InternalVecType P = Cast<InternalScalarType>(ray.origin);
InternalScalarType u, v, t;
InternalVecType B_A = v1 - v0;
InternalVecType C_A = v2 - v0;
InternalVecType normal = Cross(B_A, C_A);

InternalScalarType normal_length_sq
                        = Dot(normal, normal);
if (normal_length_sq <
std::numeric_limits<InternalScalarType>::epsilon()) {
  return false;
}
InternalVecType n = Normalize(normal);
InternalScalarType d = Dot(n, v0);

//t= (d-nůP) /(nůdir)
InternalScalarType n_dot_dir = Dot(n, dir);
if (std::abs(n_dot_dir) <
std::numeric_limits<InternalScalarType>::epsilon()) {
  return false; //pingxing
}
InternalScalarType n_dot_P = Dot(n, P);
t = (d - n_dot_P) / n_dot_dir;
if (t < static_cast<InternalScalarType>(ray.t_min) ||
  t > static_cast<InternalScalarType>(ray.t_max)) {
  return false;
}
InternalVecType Q = P + t*dir;

//inside?
InternalVecType Q_A = Q - v0;
InternalVecType Q_B = Q - v1;
InternalVecType Q_C = Q - v2;
InternalVecType cross1 = Cross(B_A, Q_A);
InternalScalarType test1 = Dot(cross1, n);
if (Dot(Cross(B_A, Q_A), n)< InternalScalarType(0)) {
  return false;
}
InternalVecType C_B = v2 - v1;  // C-B
InternalVecType cross2 = Cross(C_B, Q_B);
InternalScalarType test2 = Dot(cross2, n);
if (test2 < InternalScalarType(0)) {
  return false;
}
InternalVecType A_C = v0 - v2;  // A-C
InternalVecType cross3 = Cross(A_C, Q_C);
InternalScalarType test3 = Dot(cross3, n);
if (test3 < InternalScalarType(0)) {
  return false;
}
//barycentric
u = Dot(cross3, n) / Dot(Cross(B_A, C_A), n);
v = Dot(cross1, n) / Dot(Cross(B_A, C_A), n);
if (u < InternalScalarType(0) ||
    v < InternalScalarType(0) ||
  u + v > InternalScalarType(1)) {
  return false;
}
```

First, compute the intersection with the triangle's plane, then verify the intersection point lies within the triangle.

2.2 Ray-AABB Intersection

```
const Vec3f &inverse_dir =
              ray.safe_inverse_direction;
const Vec3f &t_min = Min( (low_bnd - ray.origin)
      * inverse_dir, (upper_bnd - ray.origin)
         * inverse_dir);
const Vec3f &t_max = Max( (low_bnd - ray.origin)
        * inverse_dir, (upper_bnd - ray.origin)
       * inverse_dir);;
*t_in = max(ReduceMax(t_min), ray.t_min);
*t_out = min(ReduceMin(t_max), ray.t_max);
if (*t_out < 0) return false; //reverse
return *t_out >= *t_in;
```

2.3 BVH Construction

```
// Stop criteria for leaf nodes
if (depth >= CUTOFF_DEPTH
      || span_right - span_left == 1) {

}


split = span_left + count / 2;
std::nth_element(
  nodes.begin() + span_left,
  nodes.begin() + split,
  nodes.begin() + span_right,
  [dim](const NodeType &a, const NodeType &b) {
    return a.getAABB().getCenter()[dim]
  < b.getAABB().getCenter()[dim];
  });
```

2.4 Implement a Direct Illumination Integrator

```
//Cast multiple rays per pixel with small offsets
const Vec2f &pixel_sample = sampler.getPixelSample
auto ray = camera->generateDifferentialRay(pixel_

//Cast a shadow ray from the intersection point
//toward the light source to determine visibility
SurfaceInteraction test;
Ray shadow_ray(interaction.p, light_dir,
               RAY_DEFAULT_MIN, dist_to_light);
if (scene->intersect(shadow_ray, test)) {
  return color;
}
//For each visible intersection,
//compute direct illumination from the light source
if (bsdf != nullptr && is_ideal_diffuse) {
    Float cos_theta =
        std::max(Dot(light_dir, interaction.normal), 0.
// one-sided
    color = bsdf->evaluate(interaction) * point_light_f
          * cos_theta / (dist_to_light * dist_to_light)
}
```

2.5 Integrate with Refractive Materials

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

// Compute diffuse lighting
Vec3f refracted_dir;
  if (Refract(interaction.wo, normal, eta_corrected, re
    interaction.wi = refracted_dir;
  } else {
    interaction.wi = Reflect(interaction.wo, normal);
  }
```

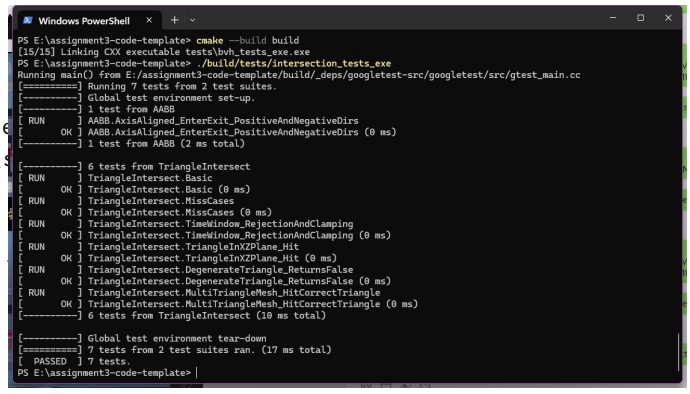
2.6 Anti-aliasing via Multi-ray Sampling

3 RESULTS

3.1 Intersection Tests

The ray-triangle and ray-AABB intersection implementations were validated using the provided test suite:

```
cmake -B build
cmake --build build
./build/tests/intersection_tests
```



3.2 BVH Construction Tests

The BVH construction was tested with the following command:

```
cmake -B build
cmake --build build
./build/tests/bvh_tests
```

```
PS E:\assignment3-code-template>  ./build/tests/bvh_tests_exe
Running main() from E:/assignment3-code-template/build/_deps/googletest-src/googletest/src/gtest_main.cc
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from BVH
[ RUN      ] BVH.BasicConstruction
[       OK ] BVH.BasicConstruction (0 ms)
[ RUN      ] BVH.SingleObject
[       OK ] BVH.SingleObject (0 ms)
[ RUN      ] BVH.EmptyTree
[       OK ] BVH.EmptyTree (0 ms)
[----------] 3 tests from BVH (2 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (5 ms total)
[  PASSED  ] 3 tests.
```

3.3 Visual Results

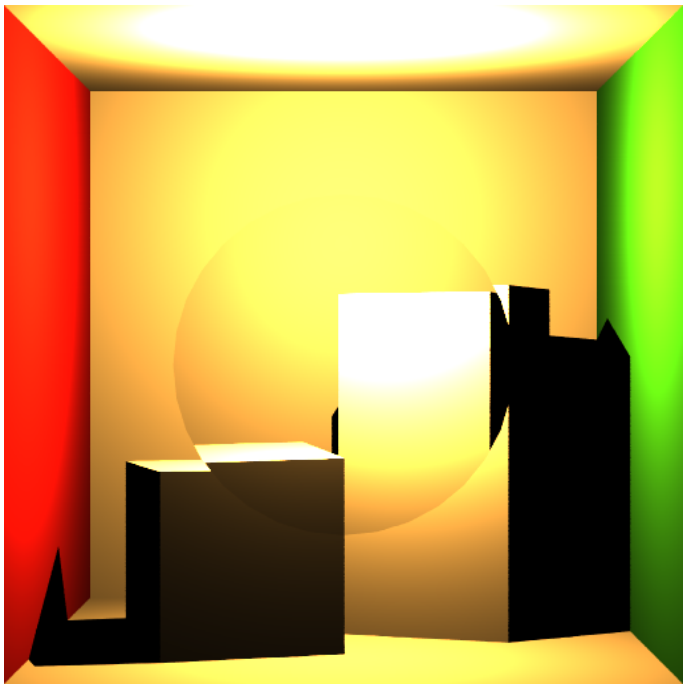


Fig. 2. Cbox No Light Refract

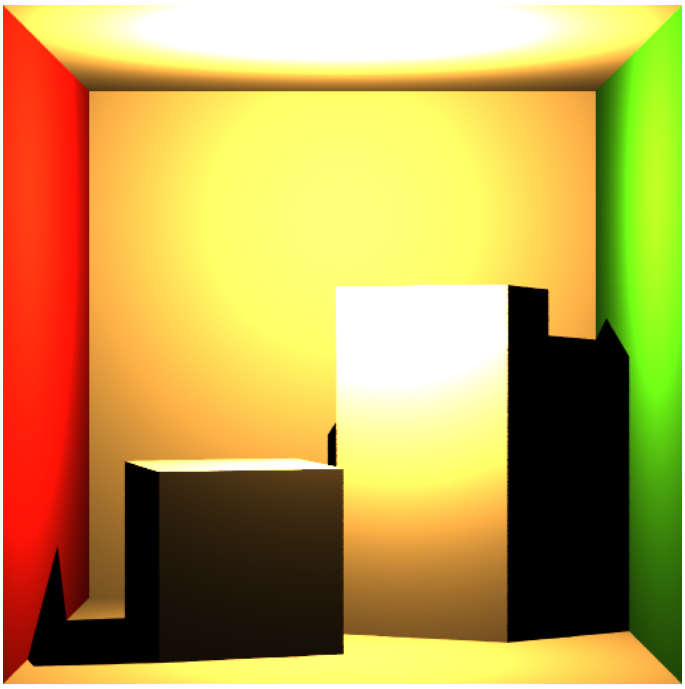


Fig. 1. Cbox No Light Refract