Assignment 3: Basic Ray Tracing

NAME: ZONGRU LIU
STUDENT NUMBER:2023533123
EMAIL: LIUZR2023@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, I finished all the must task and first 4 optional task.For the last task, it's difficult to set up the environment while using gcc and nvcc to run bvh parallel algorithm on the GPU.

2 IMPLEMENTATION DETAILS

2.1 Ray-Triangle Intersection

I follows the algorithm of the first link, I first calculate the normal of the triangle use the cross product, and if the normal is 0 we should return false which means that the in put is not a triangle. then we should check the dot product of ray_dir and the normal of triangle, if it equals to 0, we should return false, which means they are parallel.We use this formula to calculate the position of insertion point.

$$t = \frac{d - \mathbf{n} \cdot P}{\mathbf{n} \cdot \mathbf{d}}$$

and use this formula to calculate the barycentric coordinates.

$$u = \frac{(C - B) \times (Q - B) \cdot \mathbf{n}}{(B - A) \times (C - A) \cdot \mathbf{n}}$$

and verify the t u v whether satisfies the requirements.

2.2 Ray-AABB Intersection

I calculate `txmax, tymax, tzmax, txmin, tymin, tzmin` individually

```
if (invDir.x >= 0)
{
txmin = (low_bnd.x-ray.origin.x)*invDir.x;
txmax = (upper_bnd.x-ray.origin.x)*invDir.x;
}
else {
txmax = (low_bnd.x-ray.origin.x)*invDir.x;
txmin = (upper_bnd.x-ray.origin.x)*invDir.x;
}
```

then get the intersection of 3 dimensional of t, if the intersection is not exist this function returns false otherwise it returns true.

2.3 Implement the BVH construction

the stop criteria is
`depth >= CUTOFF_DEPTH || span_right - span_left == 1`
and I use this function to sort the nodes

```
std::nth_element(nodes.begin() + span_left,
nodes.begin() + split,
nodes.begin() + span_right,
[dim](const auto x, const auto y)
{ return x.getAABB().getCenter()[dim]
    < y.getAABB().getCenter()[dim];});
```

I haven't implement the hprofile == EHeuristicProfile::ESurfaceAreaHeuristic part because it marked as BONUS.

2.4 Implement a Direct Illumination Integrator

I use these function to generate #spp rays for each pixel.

```
const Vec2f &pixel_sample=sampler.getPixelSample();
auto ray =
    camera->generateDifferentialRay(pixel_sample.x,
        pixel_sample.y);
```

In the function directLighting I use this function to test if there are objects between the intersection point and point light. If there are objects between I return (0,0,0)

```
SurfaceInteraction newSurface;
bool intersected = scene->
    intersect(test_ray, newSurface);
Vec3f interactPoint = newSurface.p;
if (linalg::distance(interactPoint, test_ray.origin)
    < linalg::distance(point_light_position,
        test_ray.origin) - 1e-4
&& intersected)
{
  return Vec3f(0.0,0.0,0.0)
}
```

then I calculate the albedo and radiance using

```
Vec3f albedo = bsdf->evaluate(interaction)
    * cos_theta
Vec3f radiance = point_light_flux /
    (4 * PI * dist_to_light * dist_to_light)
```

At last I assign color to albedo * radiance.

2.5 Integrate with Refractive Materials

if the bsdf is perfect refraction I use this function to update the ray

```
interaction.bsdf->sample
    (interaction, sampler, nullptr);
ray = interaction.spawnRay(interaction.wi);
```

student number:2023533123
email: liuzr2023@shanghaitech.edu.cn
In the function bsdf::sample if the ray can be refracted set the interaction.wi to the direction after refract, else set the interaction.wi to reflected direction.

```
Vec3f refracted;
if (Refract(interaction.wo, normal,
    eta_corrected, refracted))
{
    interaction.wi = refracted;
}
else
{
    interaction.wi = Reflect
        (interaction.wo, normal);
}
```

2.6 Implement multiple light sources

I change the construction function of IntersectionTestIntegrator to get multiple point light position.

```
IntersectionTestIntegrator(const Properties &props)
: Integrator(props) {
for (int i = 0;;i++)
{
    std::string pos_name =
        format("point_light_position_{}", i);
    std::string flux_name =
        format("point_light_flux_{}", i);
    if (props.hasProperty(pos_name)
        && props.hasProperty(flux_name))
    {
        point_light_positions.push_back
           (props.getProperty<Vec3f>
            (pos_name));
        point_light_fluxs.push_back
           (props.getProperty<Vec3f>
            (flux_name));
    }
    else
    {
        break;
    }
}
if (props.hasProperty("point_light_position")
    && props.hasProperty("point_light_flux"))
{
    point_light_positions.push_back
        (props.getProperty<Vec3f>
            ("point_light_position"));
    point_light_fluxs.push_back
        (props.getProperty<Vec3f>
            ("point_light_flux"));
}
max_depth = props.getProperty<int>("max_depth", 16);
spp       = props.getProperty<int>("spp", 8);
}
```

for the function directLighting I iterate the point lights in point_light_positions, and add all the result up to get the correct result.

2.7 Implement rectangular area light

I add a new class of Integrator named AreaLightTestIntegrator I copy the render and Le function in IntersectionTestIntegrator, for correct creating class from json file I modified the register factory macro to let it can create the new type of Integrator I defined.

```
RDR_REGISTER_FACTORY(Integrator,
  [](const Properties &props) -> Integrator * {
  auto type = props.getProperty<std::string>
      ("type", "path");
  if (type == "intersection_test") {
    return Memory::alloc<IntersectionTestIntegrator>
        (props);
  }
  else if (type == "area_light_test"){
    return Memory::alloc<AreaLightTestIntegrator>
        (props);
  }
  else {
    Exception_("Integrator␣type␣{}␣not␣found", type);
  }

  return nullptr;
})
```

To implement the directLighting function I iterate all light in scene->getLights() and add the color up.

For each area light I sample spp times and store the sampled position in light_interaction and check the occlusion using the same method with the point light. For the raidiance part we need to use this formula to calculate.

```
    scene->getLights()[j]->
        Le(light_interaction, -light_dir)
        * Dot(-light_dir, light_interaction.normal)/
        (light_pdf * dist_to_light * dist_to_light);
```

after iteration I take the average of #spp colors.

2.8 Implement environment lighting

I collect a environment map and put it in "data\assets\". For sample in the sphere I use sphere coordinate to sample it. For the environment light, I also implement it in AreaLightTestIntegrator, and I use dynamic_cast<InfiniteAreaLight *>(scene->getLights()[j].get()) != nullptr to verify whether it's an environment light or not. Since the sampled environment light points are set at the infinite point so when we check the occlusion, we don't need to consider the self-occlusion problem, so I implement it as follows.

```
SurfaceInteraction newSurface;
bool intersected =
    scene->intersect(test_ray, newSurface);
if (intersected)
{
  colors.push_back(color);
  continue;
}
```

And for computing the color, I ignore the distance between the interaction point and the sampled light point.

So I set the radiance to be

```
Vec3f radiance =
    scene->getLights()[j]->
        Le(light_interaction, -light_dir)
    * Dot(-light_dir, light_interaction.normal)
    / (light_pdf);
```

For the rays doesn't hit the triangle meshes I use this to compute its color.

`scene->getInfiniteLight()->Le(interaction, -ray.direction)`

2.9 Implement texture mapping

I collected a texture and put it in "data\assets\" After modify the .json file, I found it perform quit well in mapping the texture.

3 RESULTS

The results are listing as followed.

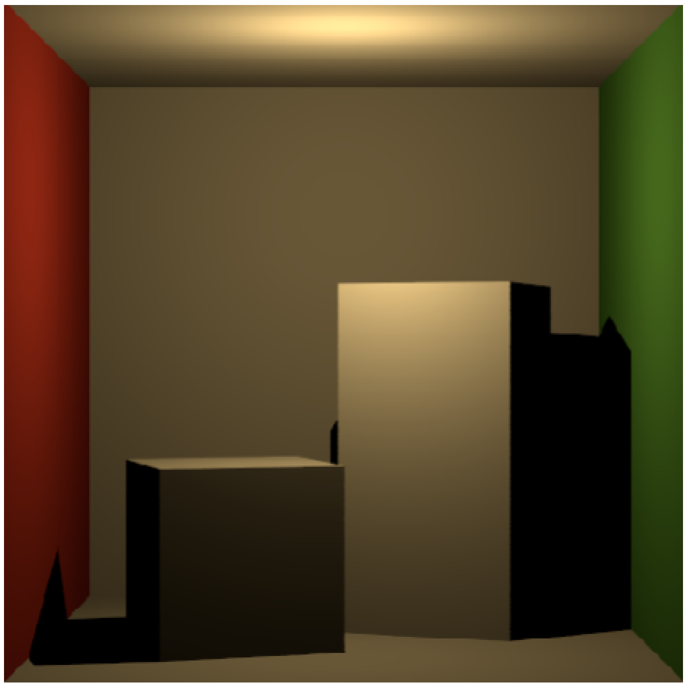

Fig. 1. result of cbox_no_light.json

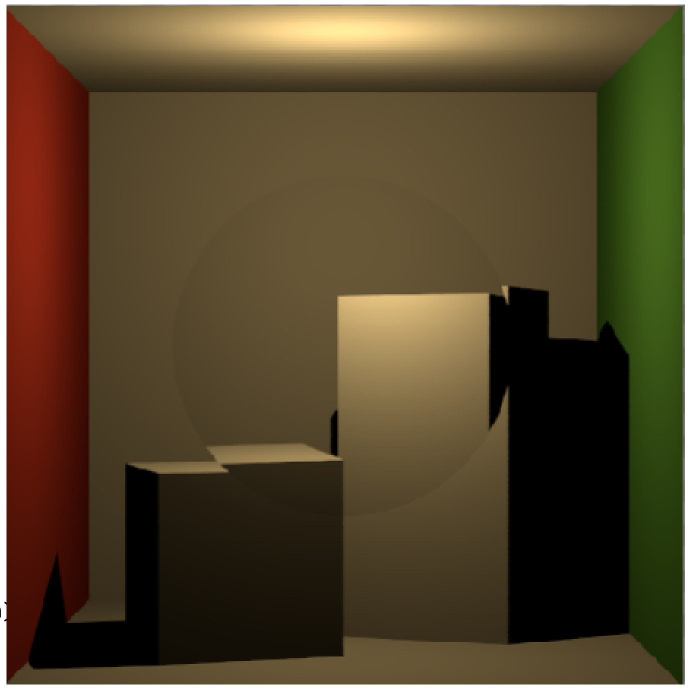

Fig. 2. result of cbox_no_light_refract.json

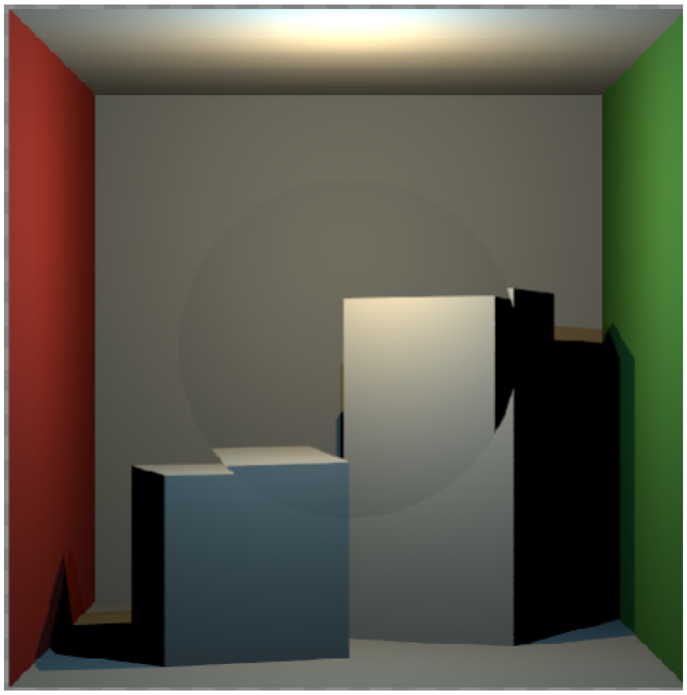

Fig. 3. result of cbox_multi_point_light_refract.json

student number:2023533123
email: liuzr2023@shanghaitech.edu.cn

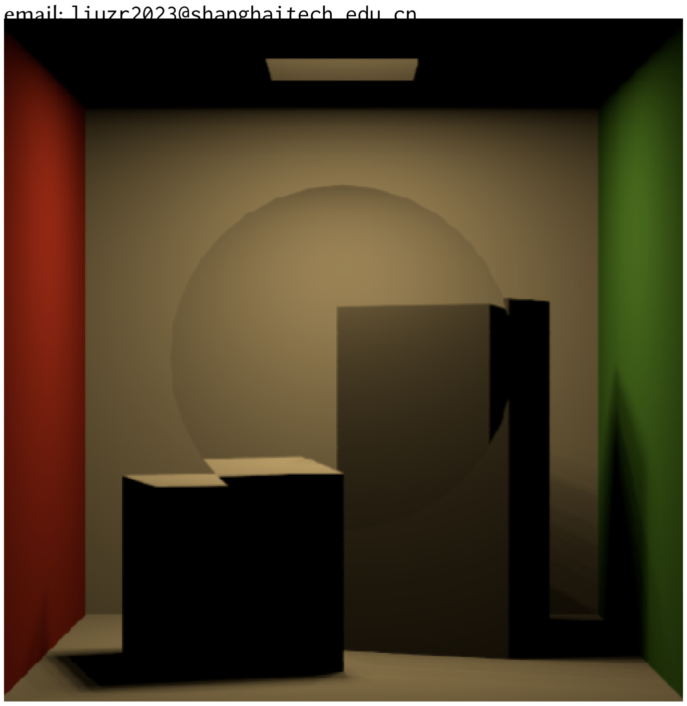
Fig. 4. result of cbox_area_light_refract.json

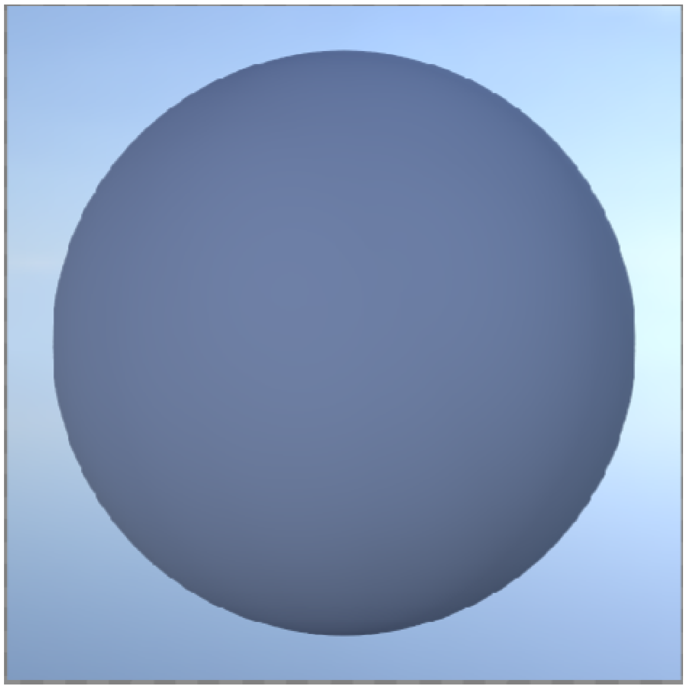
Fig. 5. result of cbox_environment_map.json

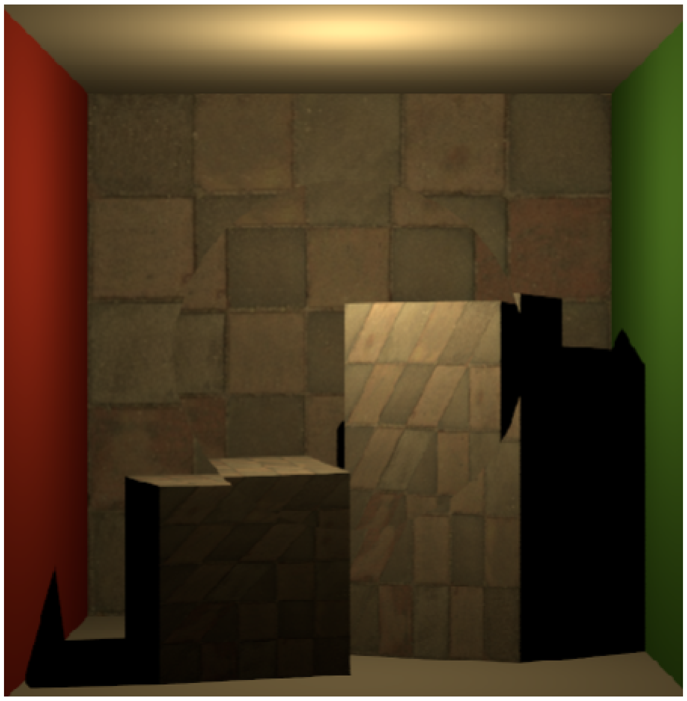
Fig. 6. result of cbox_no_light_refract_texture.json