Assignment 4: Global Illumination

NAME: BINGNAN LI
STUDENT NUMBER: 2020533092
EMAIL: LIBN@SHANGHAITECH.EDU.CN

ACM Reference Format:
Name: Bingnan Li, student number: 2020533092, email: libn@shanghaitech.edu.cn. 2022. Assignment 4: Global Illumination. 1, 1 (December 2022), ?? pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

- Path-tracing with Monte Carlo integration with direct and indirect lighting.
- Ideal diffusion BRDF and area light source.
- Global BVH with Morton code + compressed and linearized BVH.
- Ideal specular BRDF.

2 IMPLEMENTATION DETAILS

(1) **Path-tracing with Monte Carlo integration with direct and indirect lighting:** We perform the Monte Carlo integration with the following pseudocode:

- Beta = 1, L = 0
- Generate a ray from the camera to a pixel on the image plane
- For i = 1 to max depth
- (0) Suppose the marching ray hits at P(i)
- (1) L += Beta * Direct lighting at P(i)
- (2) Sample ONE next ray according to P(i)'s BRDF and find the corresponding Pdf
- (3) Beta *= BRDF * cosΘ / Pdf
- (4) Spawn and march the new ray
- where L is the output radiance.

And here is the code:

```
Vec3f L(0, 0, 0);
Vec3f beta(1, 1, 1);
for (int i = 0; i < max_depth; ++i) {
  Interaction interaction{};
  if (!scene->intersect(ray, interaction
      )) break;
  interaction.wo = ray.direction;
  if (i == 0 && interaction.type ==
      Interaction::Type::LIGHT) {
    return scene->getLight()->emission(
        Vec3f(0, 0, 0), interaction.wo);
  }
  if (interaction.type == Interaction::
      Type::LIGHT) break;
  L += beta.cwiseProduct(directLighting(
      interaction, sampler));
  float pdf = interaction.material->
      sample(interaction, sampler);
  Vec3f BSDF = interaction.material->
      evaluate(interaction);
  float cosine = interaction.wi.dot(
      interaction.normal.normalized());
  beta = beta.cwiseProduct(BSDF * cosine
       / pdf);
  ray = Ray(interaction.pos, interaction
      .wi);
}
```

(2) **Ideal BRDF and area light:**
First, we need to specify the rendering equation:

$$L_o(x,\omega_o) = \int_{\Omega_+} L_i(x,\omega_i) f_r(x,\omega_o,\omega_i)\cos\theta d\omega_i$$

In order to get cosine-weighted sampling over hemisphere area, we have the following inference:

$$\begin{aligned}
&let\\
&\quad p(\omega) = c\cos\theta\\
&\quad \Rightarrow \int_\Omega p(\omega)d\omega = 1\\
&\quad \Rightarrow c = \frac{1}{\pi}\\
&\quad \Rightarrow p(\omega) = \frac{1}{\pi}\cos\theta\\
&since\\
&\quad p(\omega)d\omega = \frac{1}{\pi}\cos\theta\sin\theta d\theta d\phi = p(\theta,\phi)d\theta d\phi\\
&\quad \Rightarrow p(\theta,\phi) = \frac{\cos\theta\sin\theta}{\pi}
\end{aligned}$$

Author's address: Name: Bingnan Li
student number: 2020533092
email: libn@shanghaitech.edu.cn.

Thus

$$p(\theta) = \int_0^{2\pi} p(\theta,\phi)d\phi = 2\cos\theta\sin\theta$$
$$p(\phi|\theta) = \frac{p(\theta,\omega)}{p(\theta)} = \frac{1}{\pi}$$
$$\Rightarrow$$
$$P(\theta) = \int_0^{\theta} 2\cos\theta'\sin\theta' d\theta' = \frac{1-\cos 2\theta}{2} = 1 - \cos\theta^2$$
$$P(\phi|\theta) = \int_0^{\phi} \frac{1}{2\pi} d\phi' = \frac{\phi}{2\pi}$$

There, for any two uniformly sampled number ξ_1, ξ_2, we have

$$\theta = \cos^{-1}\sqrt{1-\xi_1}, \quad \phi = 2\pi\xi_2$$

Then, for each sample, we sample a new direction with $\theta = \cos^{-1}\sqrt{1-\xi_1}, \quad \phi = 2\pi\xi_2$ and pdf with $\frac{\cos\theta}{\pi}$.

(3) **Global BVH with Morton code + compressed and linearized BVH:**

First, we need to calculate Morton codes for each triangles by the following codes:

```
unsigned int expandBits(unsigned int v)
    {
  v = (v * 0x00010001u) & 0xFF0000FFu;
  v = (v * 0x00000101u) & 0x0F00F00Fu;
  v = (v * 0x00000011u) & 0xC30C30C3u;
  v = (v * 0x00000005u) & 0x49249249u;
  return v;
}

unsigned int morton3D(Vec3f v) {
  float x = v.x();
  float y = v.y();
  float z = v.z();
  x = std::min(std::max(x * 1024.0f, 0.0
     f), 1023.0f);
  y = std::min(std::max(y * 1024.0f, 0.0
     f), 1023.0f);
  z = std::min(std::max(z * 1024.0f, 0.0
     f), 1023.0f);
  unsigned int xx = expandBits((unsigned
       int) x);
  unsigned int yy = expandBits((unsigned
       int) y);
  unsigned int zz = expandBits((unsigned
       int) z);
  return (xx << 2) + (yy << 1) + zz;
}
```

Then, we sort triangles by Morton code. After that, we build bvh by recursively finding the split points and put two parts into left and right child node respectively.

After that, we need to linearize this bvh tree into an array. To do so, we defined a new structure named LBVHNode with the following members:

```
struct LBVHNode {
AABB aabb;
int triangle_begin_idx{-1};
union {int right_idx{-1};
  int triangle_end_idx;};
};
```

and we use dfs order to push each node into an array, say LBVH.

Finally, we traverse through LBVH to find interaction with triangles.

(4) **Ideal specular BRDF:**

Actually, we just need to calculate the reflection direction and set it to the new direction of new ray, then the work is done.

Here is how to get the reflection direction:

$$d_{reflection} = d_{incoming} + 2(d_{incoming} \cdot normal)normal$$

3 RESULTS

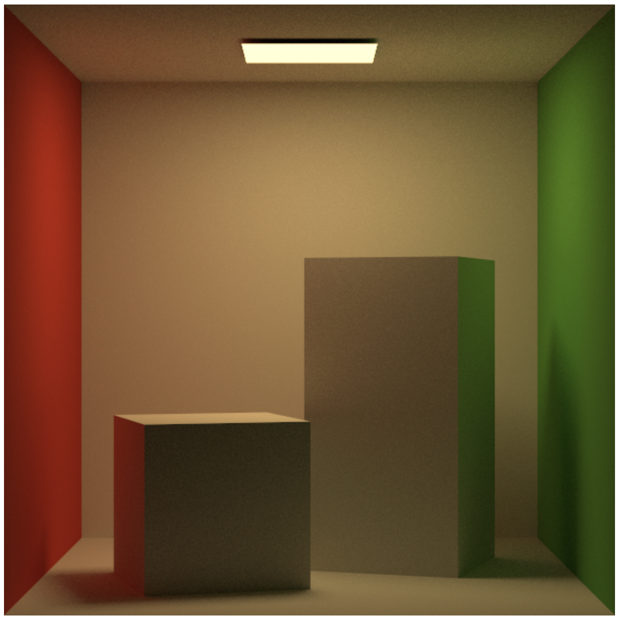

Fig. 1. simple box

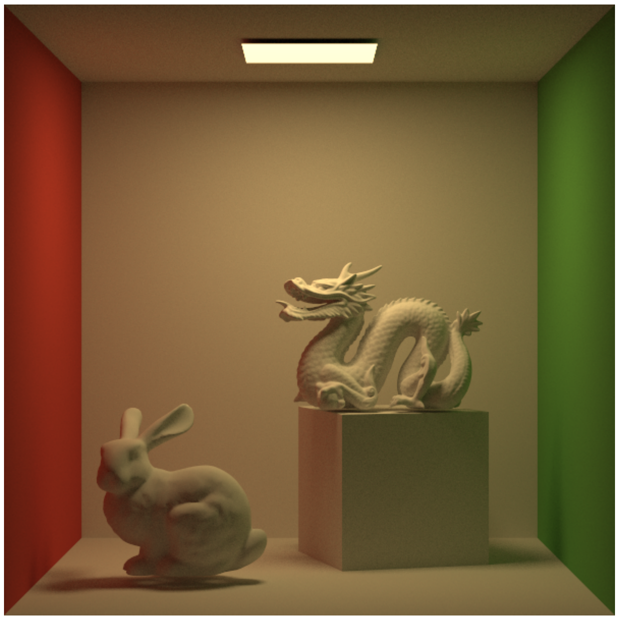
Fig. 2. large mesh

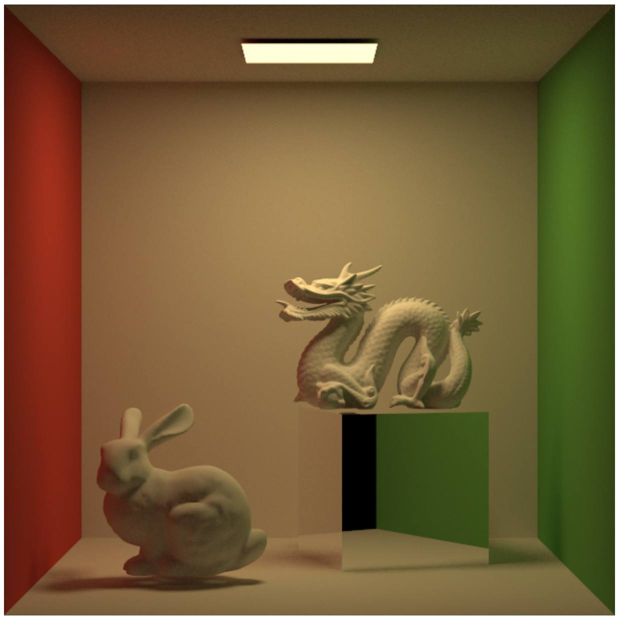
Fig. 4. large mesh with specular brdf

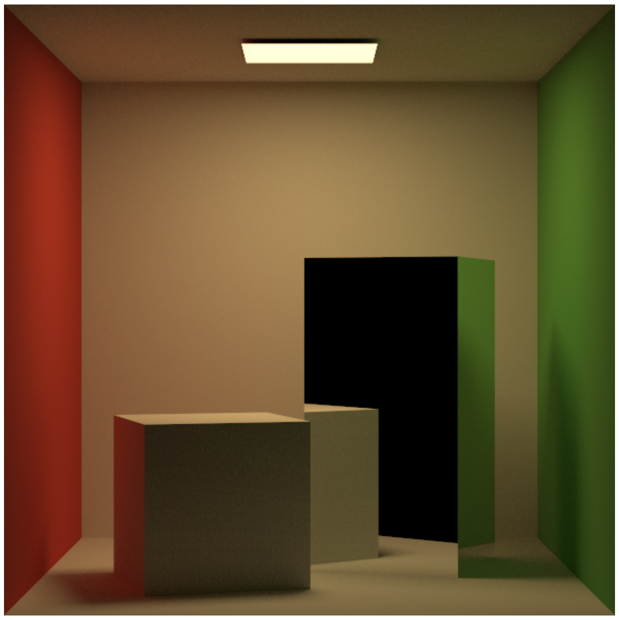
Fig. 3. Ideal specular