# Rendering Translucent Object with BSSRDF

He Jiayu
Gao Jianshu
Zhao Donghao

June 27, 2018

**Abstract**

This paper introduces the framework of the bidirectional reflection distribution function (BSSRDF), The model enables efficient simulation of efficient simulation of effects that BRDF doesn't capture. We will demonstrate how do we construct and optimize the whole process. And there are pictures of a Stanford dragon with the material of jade.

## 1 Introduction

The surface scattering is completely different if the object is translucent and with small particles inside, while many light path will be generated with the light interacts with these small particles.

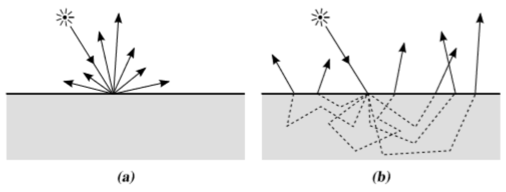


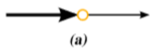
Figure2.1.:Light scattering model for (a) BRDF and (b) BSSRDF. Figure reproduced from Jensen et al. [JMLH01]

The biggest difference is that, in BRDF, the outgoing ray is generated from the same base point as the incoming ray, and in BSSRDF, base point is not the same. The light path under the surface in the medium enables us to see the inner of the object, then we will see a soft edge with tactile sensation. Objects like milk, marble, jade can not be represented by the BRDF, which means that the surface diffuse or specular or the transparent (the refraction ray) can't simulate the light process.

## 2 Theory

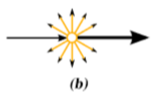
### 2.1 Volumetric Scattering

Absorption Part of the radiance is absorbed by the medium. Since all interactions conserve energy, the radiance doesn't vanish; instead, it is converted into different forms of energy, such as heat. For our purposes, we can simply consider the radiance to have vanished with absorption, since other forms of energy are invisible to the human eye. The absorbed fraction of the radiance is described by the absorption coefficient a. Putting it in differential form describes the change in radiance as:

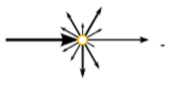


$$(\vec{\omega} \cdot \nabla) L(\vec{x}, \vec{\omega}) = -\sigma_a L(\vec{x}, \vec{\omega})$$

Emission In some cases, the medium itself may produce light. Chemical reactions, found in burning gases for example, may cause volumetric emission within the medium; in fact, the sunlight that we see daily is produced by self-emission within the sun, caused by fission reaction. Although not necessarily physically plausible, volumetric emission is also of interest in computer animation, leaving many possibilities to the artist for visual effects. In general, we can model emission within the medium using a source term Q( x,$\omega$ ),dependent on location and direction. In differential form:



Outscattering At each step, the radiance may also be reduced due to light being scattered into other directions than $\omega$ . The fraction of radiance being scattered is described by the scattering coefficient $\sigma$s. Similar to the absorption, we can formulate it in differential form as:



$$(\vec{\omega} \cdot \nabla) L(\vec{x}, \vec{\omega}) = -\sigma_s L(\vec{x}, \vec{\omega})$$

In scattering Finally, we also have to take into account the radiance arriving at x from other directions and being scattered towards direction$\omega$ ,increasing the radiance along the ray. Again, we use the scattering coefficient $\sigma$s to describe the fraction of radiance being scattered. Additionally,capturing all incident directions means integrating the incident radiance over the whole sphere of directions $\Omega$:

**(d)**

$$(\vec{\omega} \cdot \nabla) L(\vec{x}, \vec{\omega}) = \int_{\Omega} p(\vec{\omega}', \vec{\omega}) L(\vec{x}, \vec{\omega}') \mathrm{d}\vec{\omega}'$$

Here, we used the phase function p. The phase function describes the angular distribution of light intensity being scattered, similar to the BRDF. For isotropic scattering, light scatters uniformly in all directions and the phase function is constant. Most real materials exhibit dominant scattering directions however, leading to anisotropic scattering. A measure of the scattering anisotropy is the mean cosine g, computed as:

$$g = \int_{\Omega} (\vec{\omega} \cdot \vec{\omega}') p(\vec{\omega}', \vec{\omega}) \mathrm{d}\vec{\omega}'$$

Combining the four terms yields the radiative transfer equation (RTE):

$$\begin{aligned}(\vec{\omega} \cdot \vec{\nabla}) L(\vec{x}, \vec{\omega}) = & -(\sigma_a + \sigma_s) L(\vec{x}, \vec{\omega}) \\ & + Q(\vec{x}, \vec{\omega}) \\ & + \sigma_s \int_{\Omega} L(\vec{x}, \vec{\omega}) \rho(\vec{\omega}, \vec{\omega}') \mathrm{d}\vec{\omega}'\end{aligned}$$

The solution of the RTE provides the radiance L( x, $\omega$ ) at each location and direction. For the case of subsurface scattering, normally only the radiant exitance on the surface is important for rendering, whereas radiance within the medium is of little importance. This makes it convenient to reformulate the problem in terms of a bidirectional surface scattering reflection distribution function (BSSRDF), which, for any two rays hitting the surface of the medium, relates the incident flux $\varphi$i at surface location xi coming from direction w i to the reflected radiance Lr at surface location xo in direction w o:

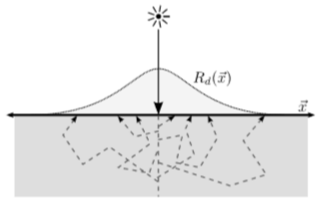


**Figure 2.3.:** *Outline of the searchlight problem*

$$S(\vec{x}_i, \vec{\omega}_i; \vec{x}_o, \vec{\omega}_o) = \frac{\mathrm{d}L_r(\vec{x}_o, \vec{\omega}_o)}{\mathrm{d}\Phi_i(\vec{x}_i, \vec{\omega}_i)}$$

The reduced radiance term S(0), representing unscattered radiance only affected by extinction; a single scattering term S(1), formed by radiance scattered exactly once; and a multiple scattering term Sd of radiance scattered more than once:

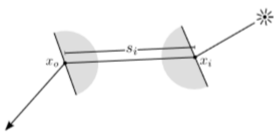
$$S = S^{(0)} + S^{(1)} + S_d$$

We can now obtain the total reflected radiance Lr at surface point xo in direction $\omega$ o due to the BSSRDF through integration:

$$L_r(\vec{x}_o, \vec{\omega}_o) = \int_A \int_{\Omega} S(\vec{x}_i, \vec{\omega}_i; \vec{x}_o, \vec{\omega}_o) L_i(\vec{x}_i, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) \mathrm{d}\vec{\omega}_i \mathrm{d}A(\vec{x}_i)$$

Our approximation:
1) As the there orders of the S0, we can obtain the following separated simulation

a) Reduced Radiance



(a)

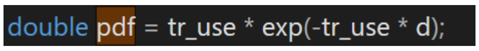
$$L_o^{(0)} = f_t(\vec{x_o}, \vec{\omega_o}) f_t(\vec{x_i}, \vec{\omega_i}) e^{-s_i \sigma_t} L_i(\vec{x_i}, \vec{\omega_i})$$

This means suppose that the light path doesn't hit a particle with scattering, which will only be influenced by the absorption and outscattering. So we only count the reduce and two times for refraction (Fresnel refraction)

$$L_o^{(0)} = f_t(\vec{x_o}, \vec{\omega_o}) f_t(\vec{x_i}, \vec{\omega_i}) e^{-s_i \sigma_t} L_i(\vec{x_i}, \vec{\omega_i})$$

b) Single scattering
1) We can obtain the refraction ray from the incoming ray's direction and the normal of the hitting surface by Fresnel law.
2) the location of the scattering event is fixed by choosing a distance so along the refracted ray. We can generate the pdf of the distance on ray by:

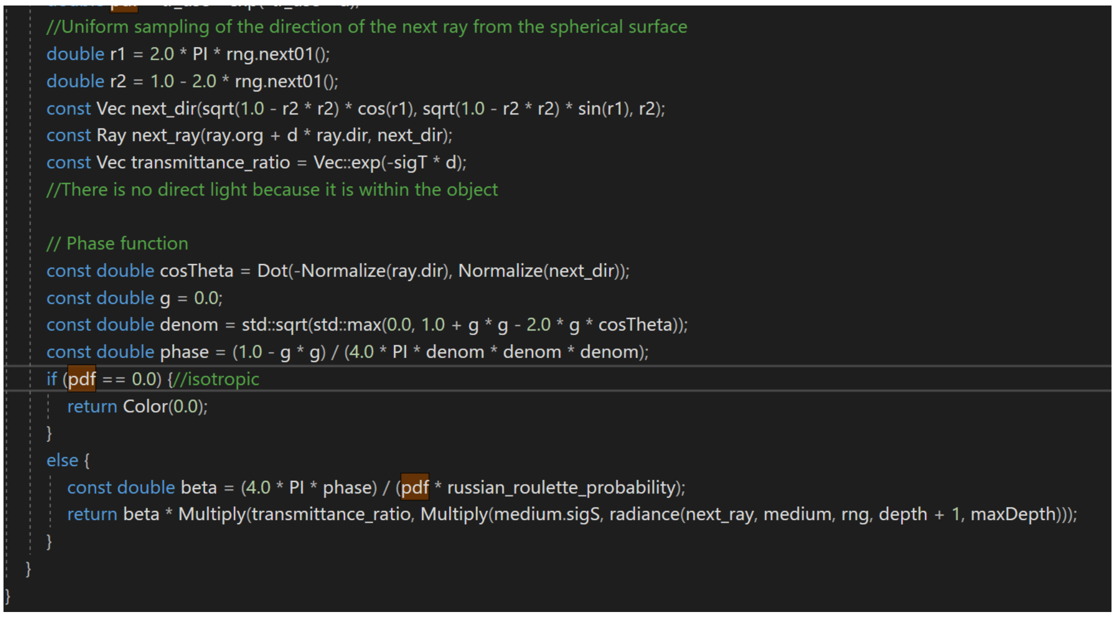
$$\sigma'_t e^{-\sigma'_t x}.$$



```
double pdf = tr_use * exp(-tr_use * d);
```

Which means there are only absorption and outscattering before the first hitting.
3) then with the direction, we can obtain the importance sampling phase function p.
If g = 0: then it is isotropic, generate a uniform sampling located in a ball.
Then outscattered radiance computed as follows.

$$L_o^{(1)}(\vec{x_o}, \vec{\omega_o}) = f_t(\vec{x_o}, \vec{\omega_o}) f_t(\vec{x_i}, \vec{\omega_i}) e^{-s_i \sigma_t} e^{-s_o \sigma_t} \alpha p(\vec{\omega_i} \cdot \vec{\omega_o}) L_i(\vec{x_i}, \vec{\omega_i})$$



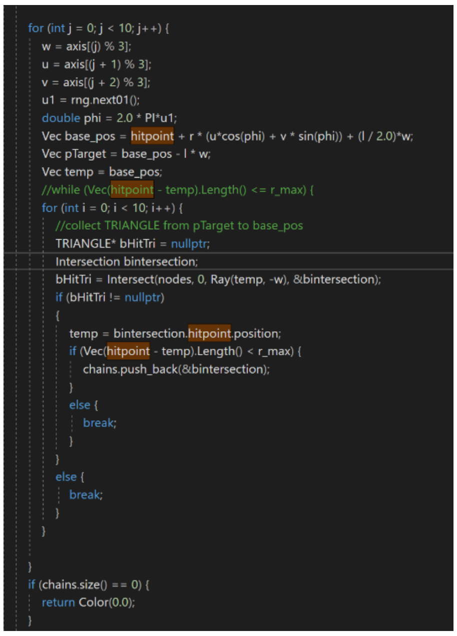
```
        //Uniform sampling of the direction of the next ray from the spherical surface
        double r1 = 2.0 * PI * rng.next01();
        double r2 = 1.0 - 2.0 * rng.next01();
        const Vec next_dir(sqrt(1.0 - r2 * r2) * cos(r1), sqrt(1.0 - r2 * r2) * sin(r1), r2);
        const Ray next_ray(ray.org + d * ray.dir, next_dir);
        const Vec transmittance_ratio = Vec::exp(-sigT * d);
        //There is no direct light because it is within the object

        // Phase function
        const double cosTheta = Dot(-Normalize(ray.dir), Normalize(next_dir));
        const double g = 0.0;
        const double denom = std::sqrt(std::max(0.0, 1.0 + g * g - 2.0 * g * cosTheta));
        const double phase = (1.0 - g * g) / (4.0 * PI * denom * denom * denom);
        if (pdf == 0.0) {//isotropic
            return Color(0.0);
        }
        else {
            const double beta = (4.0 * PI * phase) / (pdf * russian_roulette_probability);
            return beta * Multiply(transmittance_ratio, Multiply(medium.sigS, radiance(next_ray, medium, rng, depth + 1, maxDepth)));
        }
    }
}
```

c) Multiple scattering

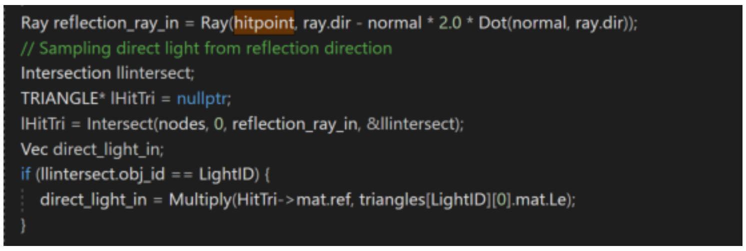
Multiple scattering can be regarded as the combination of the many single scattering, while the incoming light may not be the refraction ray.

We iterate for 10 times to check if the light path has an exit point away from the surface, we generate a chain to store the light path.

The incoming light's calculation:

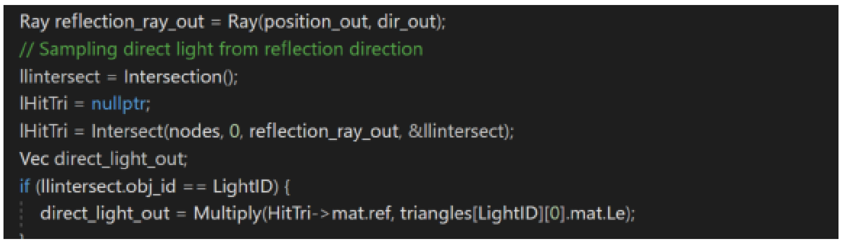


```
for (int j = 0; j < 10; j++) {
    w = axis[(j) % 3];
    u = axis[(j + 1) % 3];
    v = axis[(j + 2) % 3];
    u1 = rng.next01();
    double phi = 2.0 * PI*u1;
    Vec base_pos = hitpoint + r * (u*cos(phi) + v * sin(phi)) + (l / 2.0)*w;
    Vec pTarget = base_pos - l * w;
    Vec temp = base_pos;
    //while (Vec(hitpoint - temp).Length() <= r_max) {
    for (int i = 0; i < 10; i++) {
        //collect TRIANGLE from pTarget to base_pos
        TRIANGLE* bHitTri = nullptr;
        Intersection bintersection;
        bHitTri = Intersect(nodes, 0, Ray(temp, -w), &bintersection);
        if (bHitTri != nullptr)
        {
            temp = bintersection.hitpoint.position;
            if (Vec(hitpoint - temp).Length() < r_max) {
                chains.push_back(&bintersection);
            }
            else {
                break;
            }
        }
        else {
            break;
        }
    }


}
if (chains.size() == 0) {
    return Color(0.0);
}
```



```
Ray reflection_ray_in = Ray(hitpoint, ray.dir - normal * 2.0 * Dot(normal, ray.dir));
// Sampling direct light from reflection direction
Intersection llintersect;
TRIANGLE* lHitTri = nullptr;
lHitTri = Intersect(nodes, 0, reflection_ray_in, &llintersect);
Vec direct_light_in;
if (llintersect.obj_id == LightID) {
    direct_light_in = Multiply(HitTri->mat.ref, triangles[LightID][0].mat.Le);
}
```

The outgoing light's calculation:

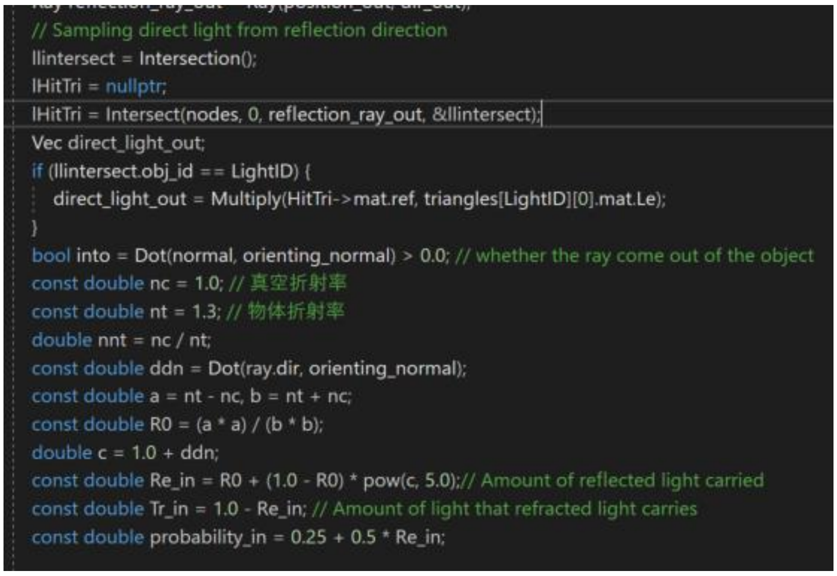


```
Ray reflection_ray_out = Ray(position_out, dir_out);
// Sampling direct light from reflection direction
llintersect = Intersection();
lHitTri = nullptr;
lHitTri = Intersect(nodes, 0, reflection_ray_out, &llintersect);
Vec direct_light_out;
if (llintersect.obj_id == LightID) {
    direct_light_out = Multiply(HitTri->mat.ref, triangles[LightID][0].mat.Le);
```

Then with the light path data we store in the chain vector, we can calculate the outgoing radiance by sample with the RTE.

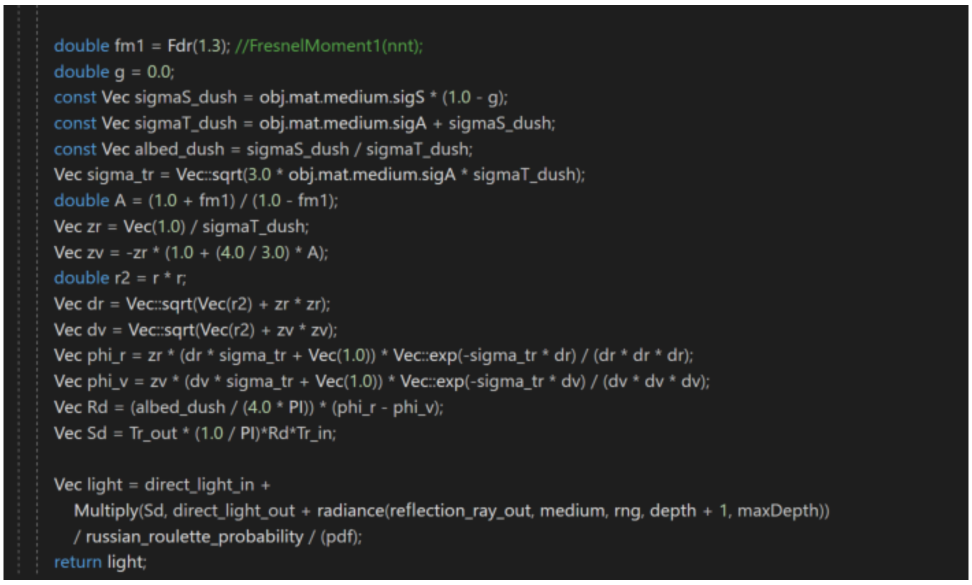
Since the it is bidirectional model, we can regard the subsurface scattering object as a light source in global, the calculation of out going influence should be combined with the sample of reflection.

The reflection calculation is based on the reflection law.



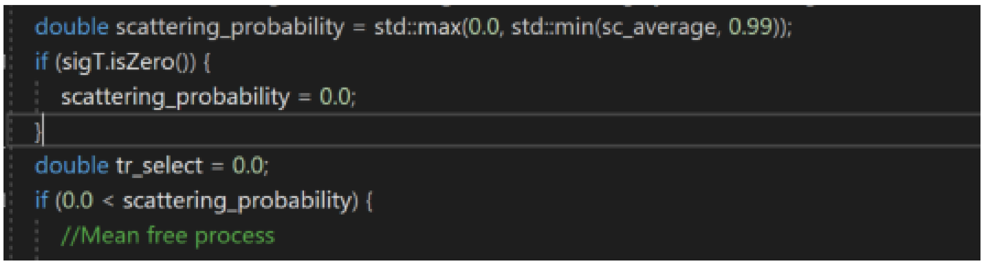
```
// Sampling direct light from reflection direction
lIntersect = Intersection();
lHitTri = nullptr;
lHitTri = Intersect(nodes, 0, reflection_ray_out, &lIntersect);
Vec direct_light_out;
if (lIntersect.obj_id == LightID) {
    direct_light_out = Multiply(HitTri->mat.ref, triangles[LightID][0].mat.Le);
}
bool into = Dot(normal, orienting_normal) > 0.0; // whether the ray come out of the object
const double nc = 1.0; // 真空折射率
const double nt = 1.3; // 物体折射率
double nnt = nc / nt;
const double ddn = Dot(ray.dir, orienting_normal);
const double a = nt - nc, b = nt + nc;
const double R0 = (a * a) / (b * b);
double c = 1.0 + ddn;
const double Re_in = R0 + (1.0 - R0) * pow(c, 5.0);// Amount of reflected light carried
const double Tr_in = 1.0 - Re_in; // Amount of light that refracted light carries
const double probability_in = 0.25 + 0.5 * Re_in;
```

And combined with the refraction ray, we get the surface color (the representation of the "emission")



```
double fm1 = Fdr(1.3); //FresnelMoment1(nnt);
double g = 0.0;
const Vec sigmaS_dush = obj.mat.medium.sigS * (1.0 - g);
const Vec sigmaT_dush = obj.mat.medium.sigA + sigmaS_dush;
const Vec albed_dush = sigmaS_dush / sigmaT_dush;
Vec sigma_tr = Vec::sqrt(3.0 * obj.mat.medium.sigA * sigmaT_dush);
double A = (1.0 + fm1) / (1.0 - fm1);
Vec zr = Vec(1.0) / sigmaT_dush;
Vec zv = -zr * (1.0 + (4.0 / 3.0) * A);
double r2 = r * r;
Vec dr = Vec::sqrt(Vec(r2) + zr * zr);
Vec dv = Vec::sqrt(Vec(r2) + zv * zv);
Vec phi_r = zr * (dr * sigma_tr + Vec(1.0)) * Vec::exp(-sigma_tr * dr) / (dr * dr * dr);
Vec phi_v = zv * (dv * sigma_tr + Vec(1.0)) * Vec::exp(-sigma_tr * dv) / (dv * dv * dv);
Vec Rd = (albed_dush / (4.0 * PI)) * (phi_r - phi_v);
Vec Sd = Tr_out * (1.0 / PI)*Rd*Tr_in;

Vec light = direct_light_in +
  Multiply(Sd, direct_light_out + radiance(reflection_ray_out, medium, rng, depth + 1, maxDepth))
  / russian_roulette_probability / (pdf);
return light;
```

2) Optimization
   a) Define a scattering probability:
      We notice that, if an object's medium is quite transparent (visible particles quantity or size is not obvious enough with light path), there is no need to use multiple scattering to measure it. Multiple scattering is costly and trivial. So we define a scattering probability to judge is the object needs to use multiple scattering.



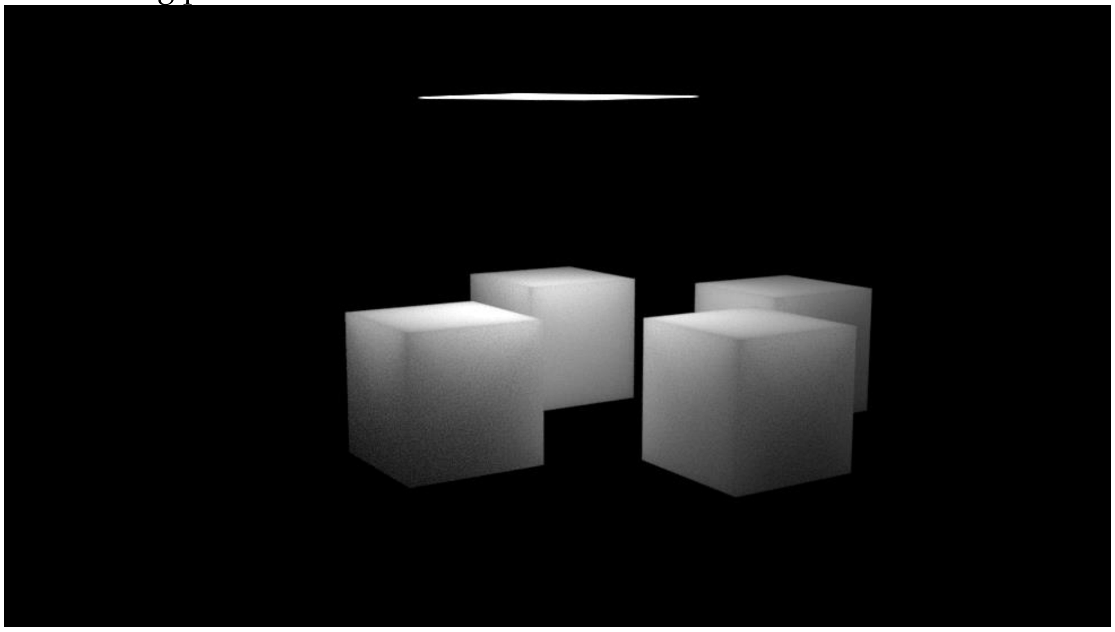
```
double scattering_probability = std::max(0.0, std::min(sc_average, 0.99));
if (sigT.isZero()) {
    scattering_probability = 0.0;
}
double tr_select = 0.0;
if (0.0 < scattering_probability) {
    //Mean free process
```

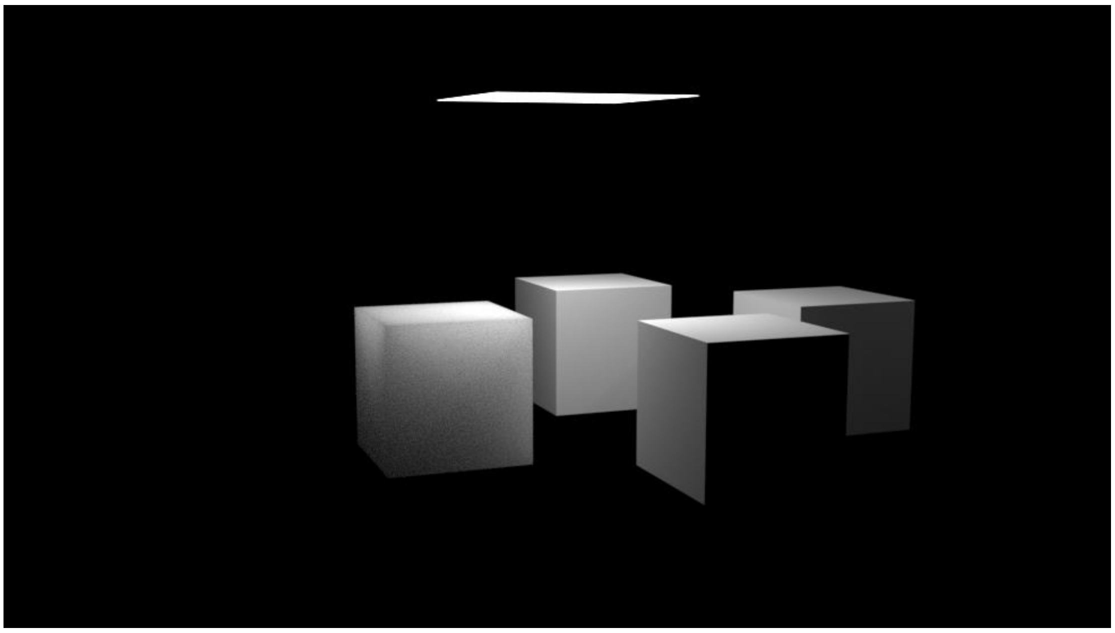
## 3 Result

We set the material as the silk milk

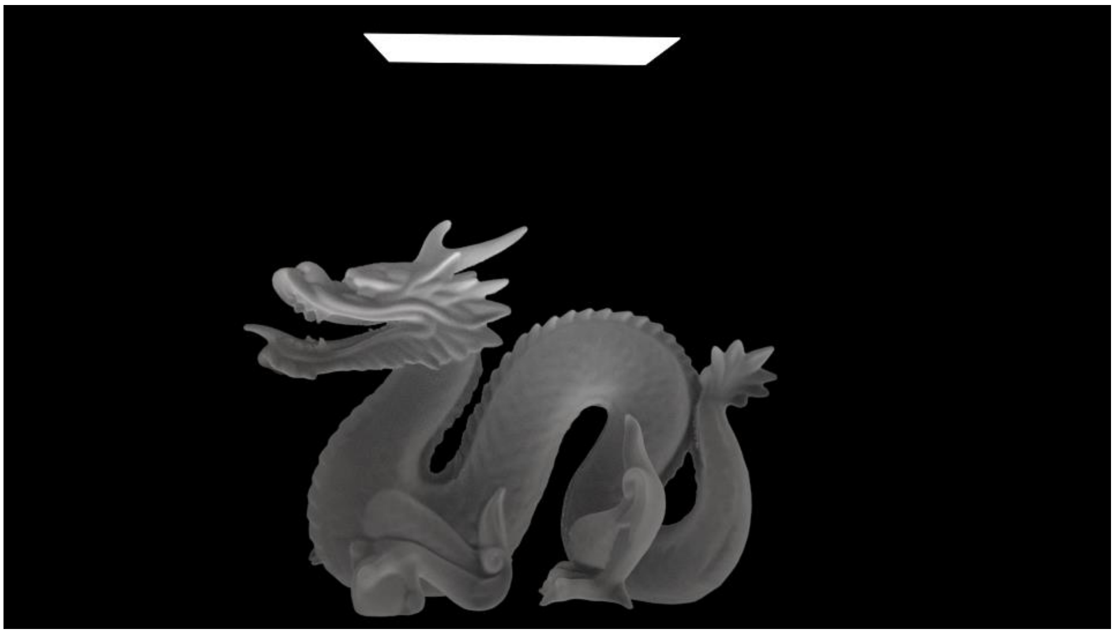
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Skimmilk | 0.70 | 1.22 | 1.90 | 0.0014 | 0.0025 | 0.0142 | 0.81 | 0.81 | 0.69 | 1.3 |

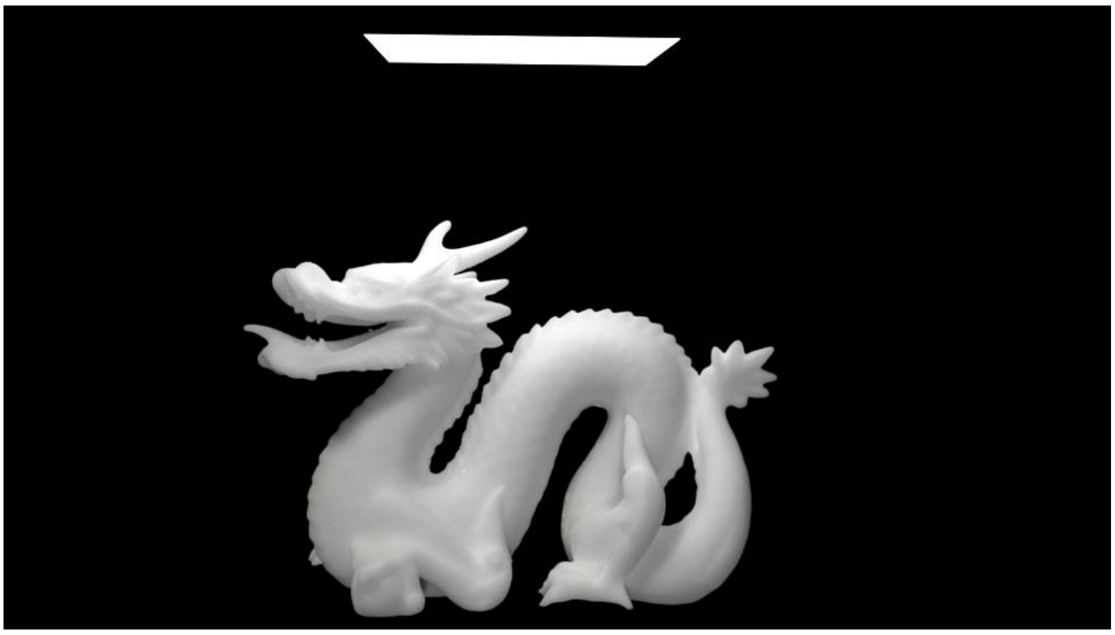
The following picture is the contrast with BRDF and BSSRDF





The following picture is the contrast of single scattering and multiple scattering





The last result