Assignment 3: PathTracer

Shenao Zhang

In this project, what we mainly do is about materials, environment light and thin-lens camera model. We have four parts: mirror and glass materials, microfacet material, environment light, depth of field. Let me introduce ecah part in detail.

Part 1: Mirror and Glass

In this part, what we mainly do is to implement mirror and glass models with both reflection and refraction.

Task 1:Reflection

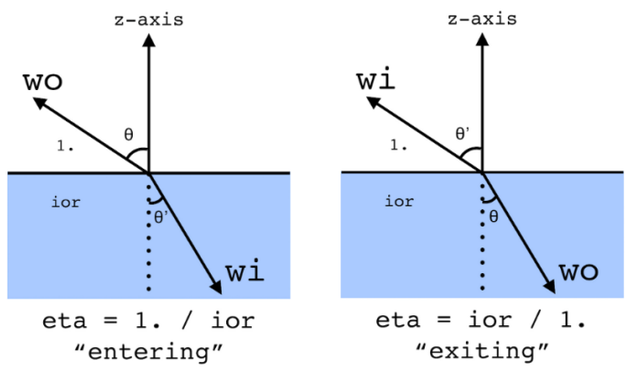
In this task, we calculate the **wi** direction given **wo** direction. It is straightforward: ***wi=Vector3D(-wo.x,-wo.y,wo.z)**, since the origin is the intersection point and the z axis lies along the normal vector.

Task 2:Mirror Material

In this task, the function **MirrorBSDF::sample_f** returns **reflectance / abs_cos_theta(*wi)** because we need to cancel out the cosine that the **at_least_one_bounce_radiance** function will multiply by.
And the maximum ray depth must be greater than 1 since if it's 0, which is only the light itself emit. When the maximum is set to 1, it represents the one bounce light, i.e. cannot represents the ray which light reflects.

Task 3:Refraction

The main idea here is to use Snell's equations. Since our BSDF calculations always take place in a canonical "object coordinate frame" where the z axis points along the surface normal, this allows us to take advantage of the spherical coordinate Snell's equations. So we get $\omega o.x=\sin\theta\cos\phi, \omega o.y=\sin\theta\sin\phi, \omega o.z=\pm\cos\theta$.

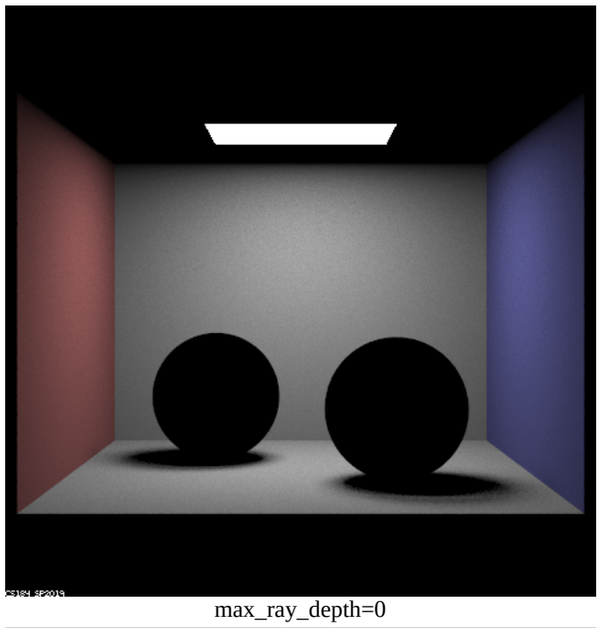


Example of refraction

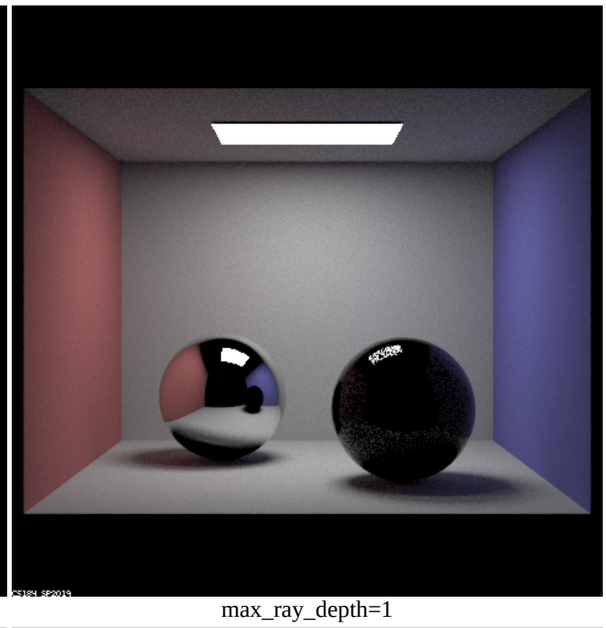
As shown in the picture above,we put a ± sign on the z coordinate because when wo starts out inside the object with index of refraction greater than 0, its z coordinate will be negative. The surface normal always points out into air. When z is positive, we say that we are entering the non-air material, else we are exiting. $\phi'=\phi+\pi, \sin\theta'=\eta\sin\theta$, so we can get that $\omega i.x=-\eta\omega o.x$, $\omega i.y=-\eta\omega o.y$, $\omega i.z=\pm\sqrt{1-\eta^2(1-\cos^2\theta)}$, where we are indicating that **ωi.z** has the opposite sign of **ωo.z**.

Task 4:Glass Material

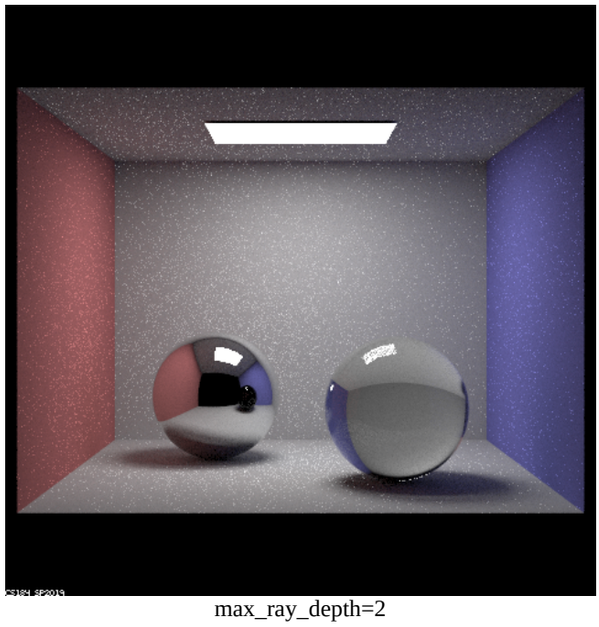
For glass material when wo has a valid refracted wi (so total internal reflection does not occur), both reflection and refraction will occur at this point. To simulate this, we have Fresnel equations to model the actual physics behind this phenomenon. Here, I use a simple version called Schlick's approximation to decide the ratio. Since **sample_f()** is only allowed to return one ray direction, we will use Schlick's approximation to give us a coin-flip probability of either reflecting or refracting.
To test the effects of multibounce, I use the **CBspheres.dae** with different **max_ray_depth** value. The other settings are 64 samples per pixel and 4 samples per light. The screenshots examples are as following:



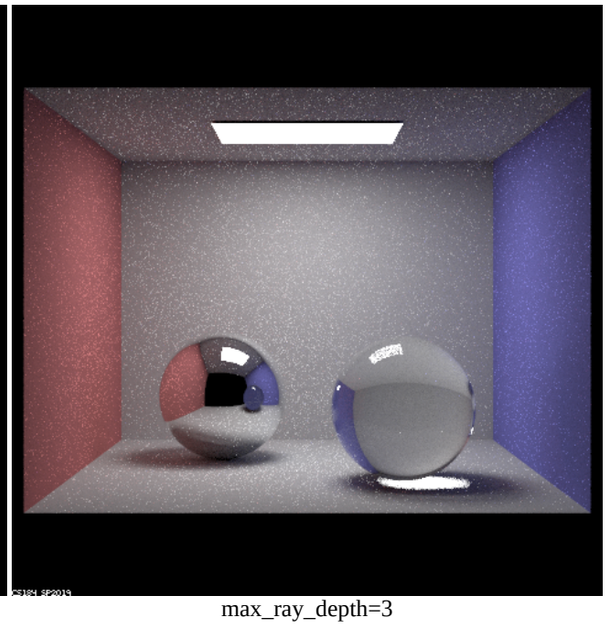
max_ray_depth=0



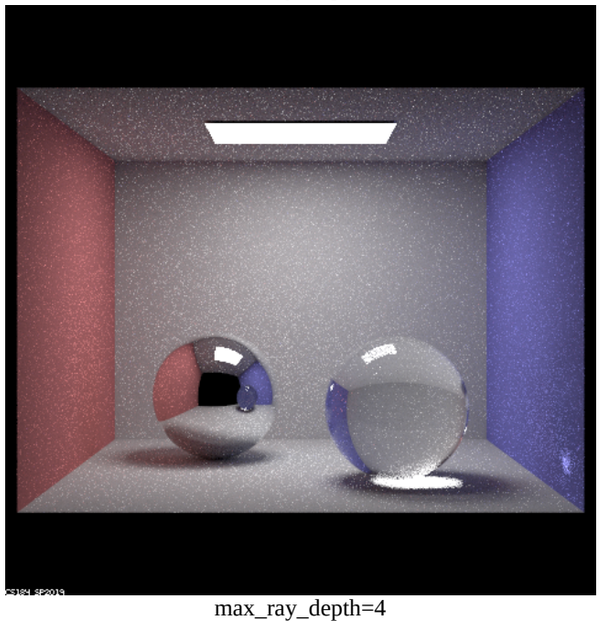
max_ray_depth=1



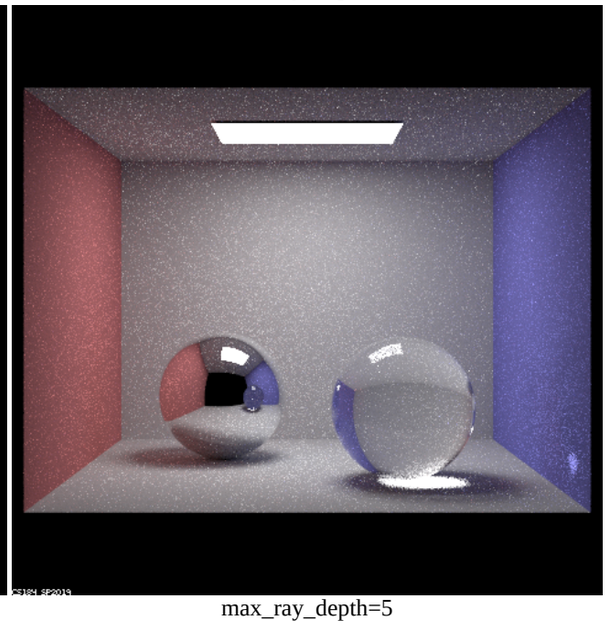
max_ray_depth=2



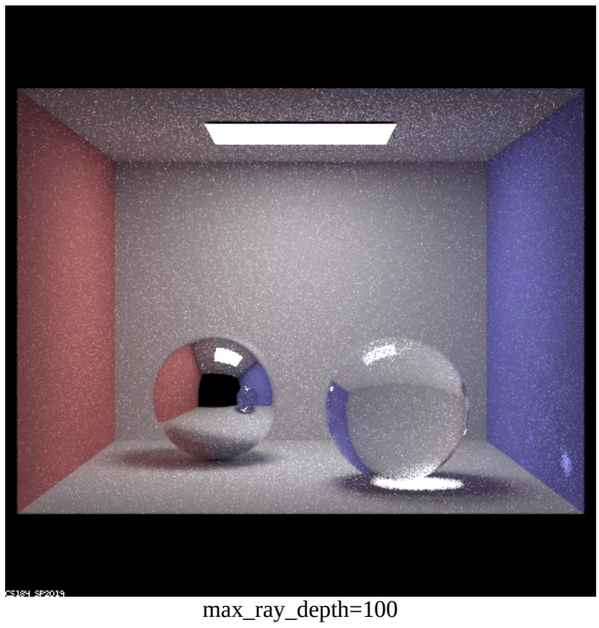
max_ray_depth=3



max_ray_depth=4

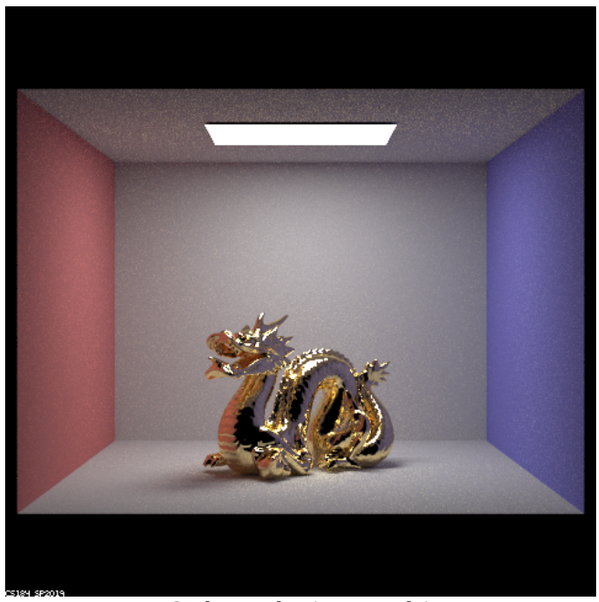


max_ray_depth=5

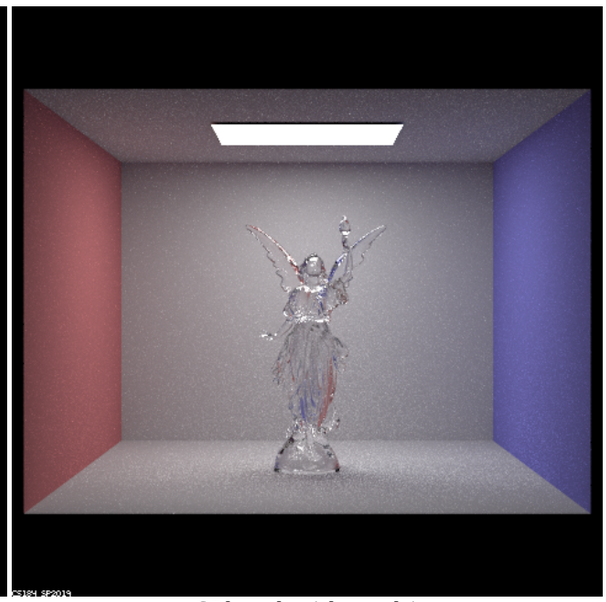


max_ray_depth=100

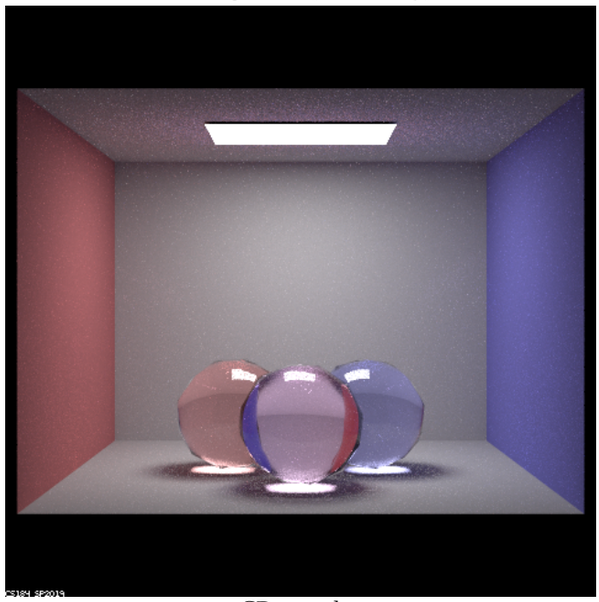
As we can see, when max_ray_depth=0, which means what is in the image is only the light source. When max_ray_depth=1, we can see that ray get reflected once on the mirror and glass sphere, and the delta sphere only reflects the light from the light source. When max_ray_depth=2, we can see that both spheres are with shadows. When max_ray_depth=3, we can see that upder the delta sphere is bright caused by multi-bounces. When the max_ray_depth is bigger, we can see that there is a light spot on the right wall, also caused by 4-bounces.
Here are some other examples of screenhots:



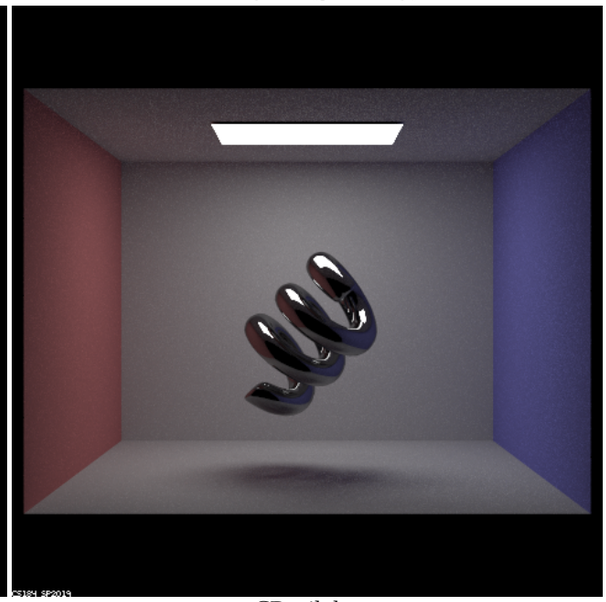
CBdragon.dae (mirror only)



CBlucy.dae (glass only)



CBgems.dae



CBcoil.dae

Part 2: Microfacet Materials

In this part, what we need to do is implementing the microfacet model on isotropic rough conductors.

Task 1:Microfacet BRDF

First,we have BRDF evaluation function:

$$f = \frac{F(\omega_i) * G(\omega_o, \omega_i) * D(h)}{4 * (n \cdot \omega_o) * (n \cdot \omega_i)}$$

F is the Fresnel term, G is the shadowing-masking term, and D is the normal distribution function (NDF). n is the macro surface normal, which is always (0,0,1) in local coordinates. h is the half vector.
So all we need to do next is to implement these functions one by one.

Task 2:Normal Distribution Function (NDF)

The NDF defines how the microfacets' normals are distributed. Given the light incident and outgoing directions wi and wo, we know that only those microfacets whose normals are exactly along the half vector h are able to reflect wi to wo because the microfacets are assumed to be perfectly specular. The NDF can be defined in various kinds of distribution functions. Here we adopt Beckmann distribution, which is similar to a Gaussian distribution.

$$D(h) = \frac{e^{-\frac{\tan^2\theta_h}{\alpha^2}}}{\pi\alpha^2\cos^4\theta_h}$$

Here, α is the roughness of the macro surface, θh is the angle between h and the macro surface normal n, i.e. (0,0,1).

Task 3:Fresnel Term

In this part, Fresnel is different from part 1, since those are for air-dielectric interfaces but here we need air-conductor. The air-conductor Fresnel term is wavelength-dependent, which means that it contains color information, thus has type Spectrum here. We need to calculate the Fresnel term for every possible wavelength, then convert the sampled spectrum to an RGB color, but it will waste a lot of time. So we calculate the Fresnel terms for R, G, B channels respectively, assuming that each channel has a fixed wavelength, which can reduce computation. Here is the approximatio version:

$$F = \frac{R_s + R_p}{2},$$

$$R_s = \frac{(\eta^2 + k^2) - 2\eta\cos\theta_i + \cos^2\theta_i}{(\eta^2 + k^2) + 2\eta\cos\theta_i + \cos^2\theta_i},$$

$$R_p = \frac{(\eta^2 + k^2)\cos^2\theta_i - 2\eta\cos\theta_i + 1}{(\eta^2 + k^2)\cos^2\theta_i + 2\eta\cos\theta_i + 1}.$$

Here η and k are used together to represent indices of refraction for conductors. Both of them are Spectrum values, recording the scalar η and k values at wavelengths 614 nm (red), 549 nm (green) and 466 nm (blue).

Part 4: Importance Sampling

In this part, our goal is to implement importance sample the microfacet BRDF according to the shape of the Beckmann NDF. In task 2, we have the NDF D(h), so we can calculate the pdfs pθ and pϕ:

$$p_\theta(\theta_h) = \frac{2\sin\theta_h}{\alpha^2\cos^3\theta_h} e^{-\tan^2\theta_h/\alpha^2},$$

$$p_\phi(\phi_h) = \frac{1}{2\pi}.$$

Then we can calculate the corresponding CDF(integrating) and get the inverse of the pdfs. The results are as follows:

$$\theta_h = \arctan\sqrt{-\alpha^2\ln(1 - r_1)},$$

$$\phi_h = 2\pi r_2,$$

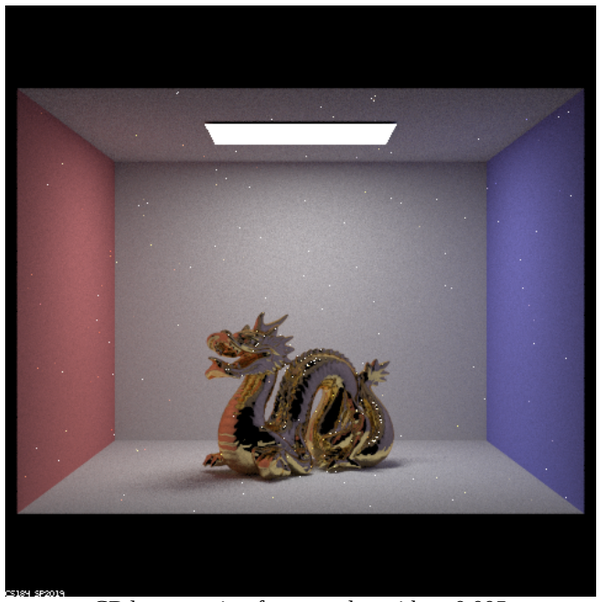
Now we need to get the pdf of sampling wi. We first calculate the pdf of sampling h with respect to solid angle:

$$p_\omega(h) = \frac{p_\theta(\theta_h) \cdot p_\phi(\phi_h)}{\sin(\theta_h)}.$$

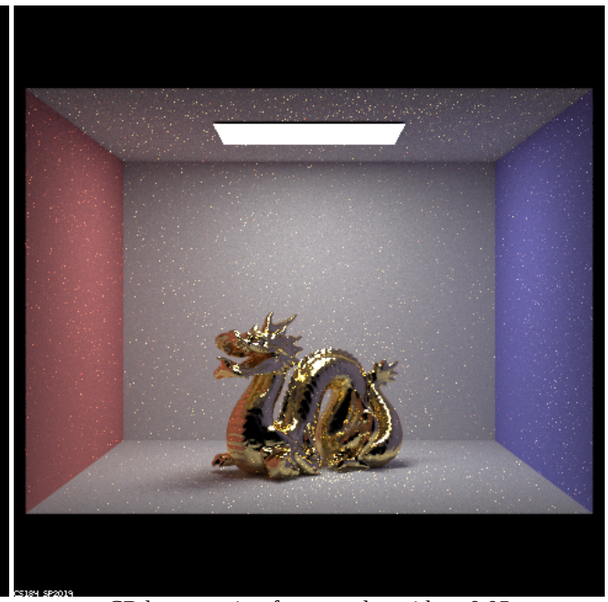
Secondly, we calculate the final pdf of sampling wi w.r.t. solid angle as:

$$p_\omega(\omega_i) = \frac{p_\omega(h)}{4(\omega_i \cdot h)}$$

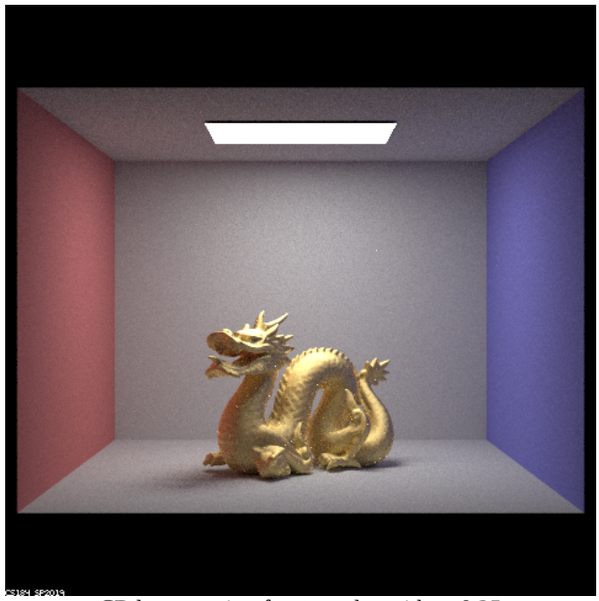
Now to test the effects of different α, i.e. different roughness of the macro surface, I set 128 samples per pixel and 1 samples per light and the number of bounces 5. Here are some results:



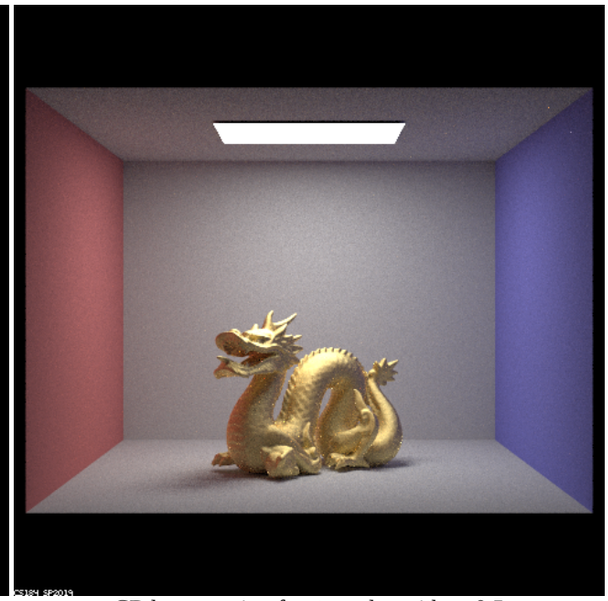
CBdragon_microfacet_au.dae with α=0.005



CBdragon_microfacet_au.dae with α=0.05



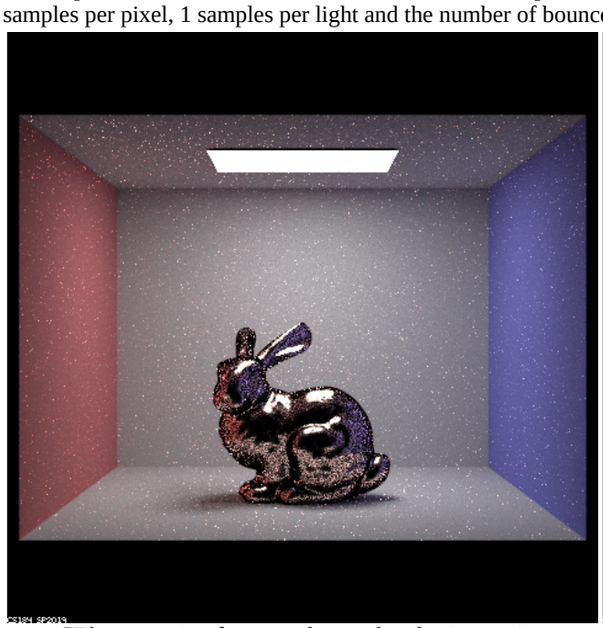
CBdragon_microfacet_au.dae with α=0.25



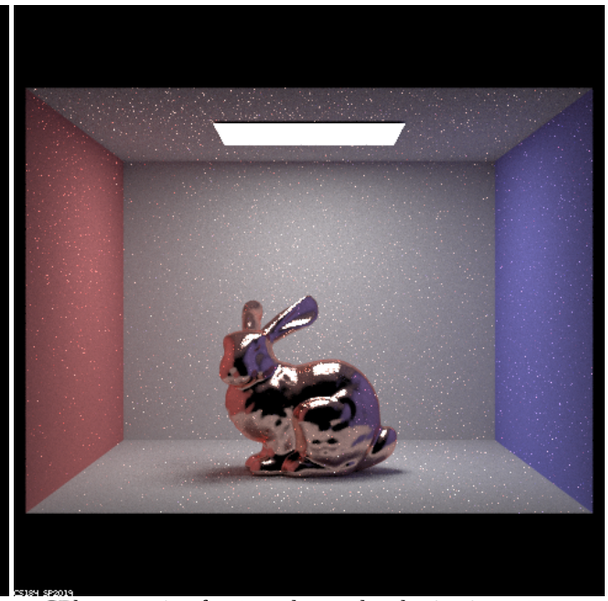
CBdragon_microfacet_au.dae with α=0.5

We can observe that with different alpha values, we have different roughness macro surface visual effects. The smaller alpha is, the smoother the surface is, i.e. more likely to be a mirror. The macro surface tends to be diffuse when α is large and glossy when α is small.

To compare the difference between default cosine hemisphere sampling and importance sampling, I set 128 samples per pixel, 1 samples per light and the number of bounces 5. The screenshots are as follows:



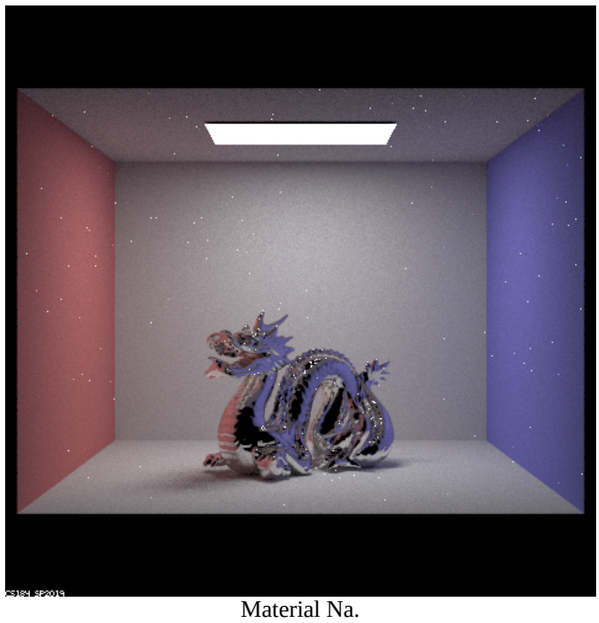
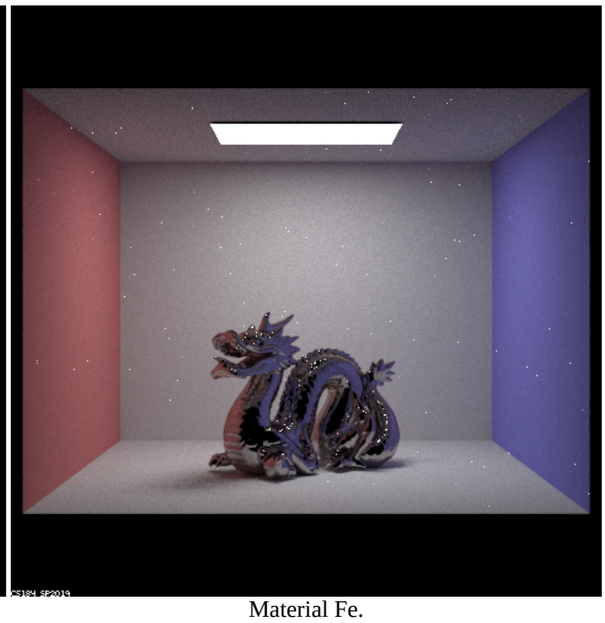
CBbunny_microfacet_cu.dae rendered using cosine hemisphere sampling.



CBbunny_microfacet_cu.dae rendered using importance sampling.

We can see that the bunny using default method is noisier than importance sampling. Because in default method, we sample **theta=2*PI*random_uniform()**. But since we need the NDF to represent how the microfacets' normals are distributed, this will not work. In importance sampling, we use inverse method to calculate the CDF, so it can represent the microfacets' normals, thus more realistic.

Now to test different conductor material, I choose two materials: Na(Sodium) and Fe(iron), find their eta and k values at wavelengths 614 nm (red), 549 nm (green) and 466 nm (blue). The eta and k values of Na are (0.051699,0.058306,0.066435), (2.5436,2.2476,1.8655) and the eta, k values of iron are (2.8851,2.95,2.65), (3.0449,2.93,2.8075). I set α=0.005, in which case the surface is smooth. The results are as follows:

Material Na. Material Fe.

As we can see, these two materials are like the real ones with the same materials.

Part 3:Environment Light

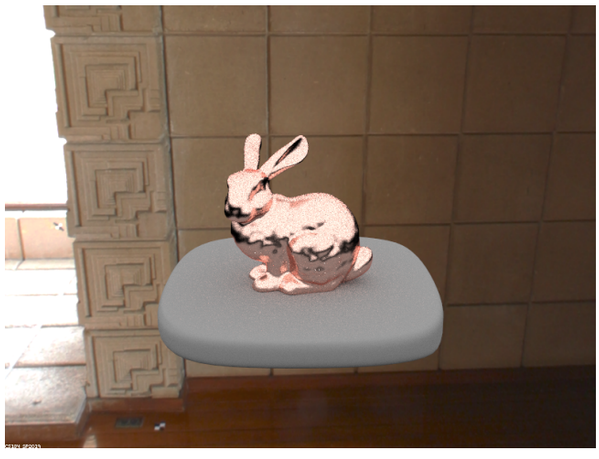
In the previous parts, we have many kind of light sources. But as for infinite environment light, which means the source is thought to be "infinitely far away" and is representative of realistic lighting environments in the real world, like the sun, we need to implement it in this part.

Task 1:EnvironmentLight::sample_dir()

In this section, we need to get coordinates that can be used to index into the envMap variable, then use bilinear interpolation to sample a Spectrum from envMap at that location. After implementing **sample_dir()** function, we can observe that background of bunny_unlit.dae is set to grace.exr and ennis.exr.


Task 2: Uniform sampling

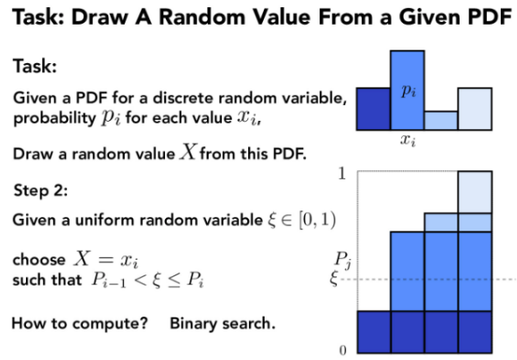
In this task, we mainly implement uniform sampling by first generating a random direction on the sphere, then like task 1, convert this direction to coordinates (ϕ,θ), then look up the appropriate radiance value in the texture map using bilinear interpolation.
To show the result, I render with the second background in the last task, i.e. ennis.exr. The settings are 4 samples/pixel, 256 samples/light, and max ray depth equal to 5.
The result is as follows:



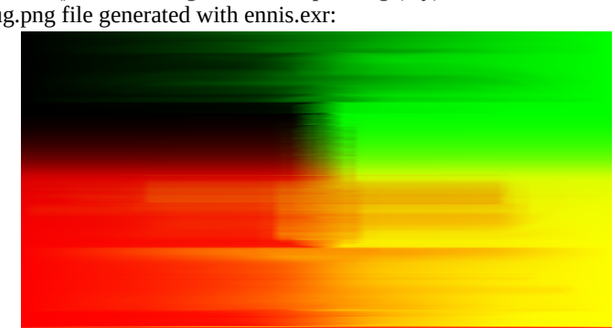
Task 3: Importance sampling

Since in the real world, the energy is more likely to concentrated in the directions toward bright light sources, we implement importance sampling in this part.
To implement this, we have several steps:
Sample a row of the environment map using the marginal distribution p(y);

Sample a pixel within that row using the condition distribution p(x|y);
Convert that (x,y) to a direction vector and return the appropriate radiance and pdf values.



The last step, i.e. updating sample_L to do importance sampling, it uses the same idea as the picture above.
We use **std::upper_bound()** function to get the corresponding (x,y).
The probability_debug.png file generated with ennis.exr:



To compare the difference between uniform sampling and importance sampling of bunny_unlit.dae, I set 4 samples per pixel and 64 samples per light in each. The results are as follows:

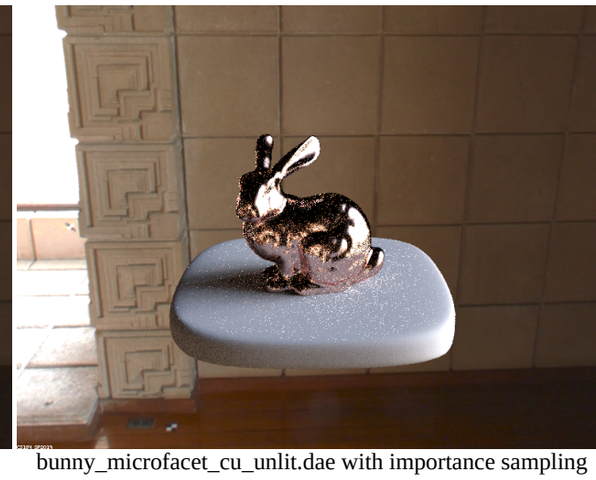

bunny_unlit.dae with uniform sampling


bunny_unlit.dae with importance sampling

As we can see, bunny_unlit.dae with uniform sampling is noisier than the one using importance sampling with the same other settings.

To compare the difference between uniform sampling and importance sampling of bunny_microfacet_cu_unlit.dae, I set 4 samples per pixel and 64 samples per light in each. The results are as follows:


bunny_microfacet_cu_unlit.dae with uniform sampling

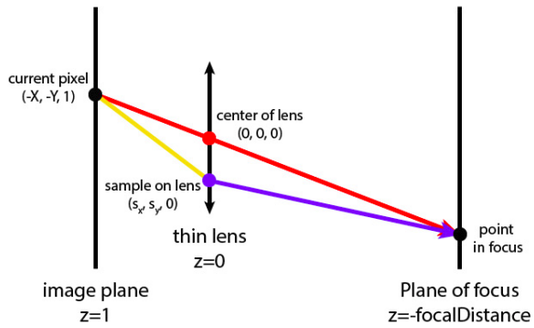

bunny_microfacet_cu_unlit.dae with importance sampling

As we can see, bunny_microfacet_cu_unlit.dae with uniform sampling is noisier than the one using importance sampling with the same other settings.

Part 4: Depth of Field

In the previous parts, we use ideal pin-hole model. With the pin-hole model, everything is in focus. But in the real world, like cameras and human eyes, it is like lenses with finite apertures instead of ideal pin-hole

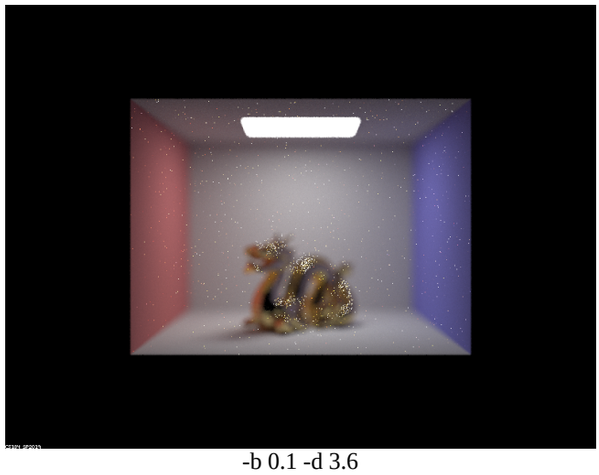
model. So in this part, what we do is to simulate lens.
To model lens, here is an example picture to make everything clear.

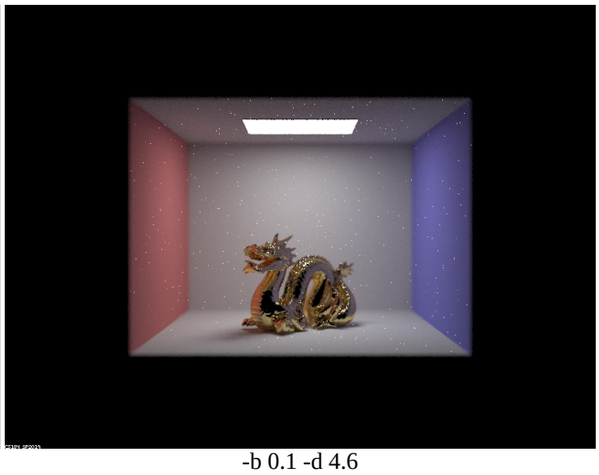


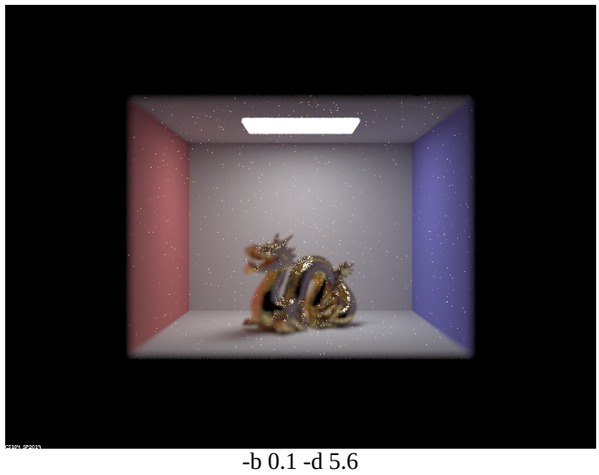
The main idea is first uniformly sample the lens, which is a disk of radius lensRadius, to get the sampled point pLens on the lens. Then calculate the direction of a ray from pLens towards the scene (blue segment in the figure).
To generat ray direction (red segment in the figure), we can use the same method in project3-1 to get the corresponding x, y in the camera coordinates, and z equals to -1. Then sample the disk representing the thin lens at:

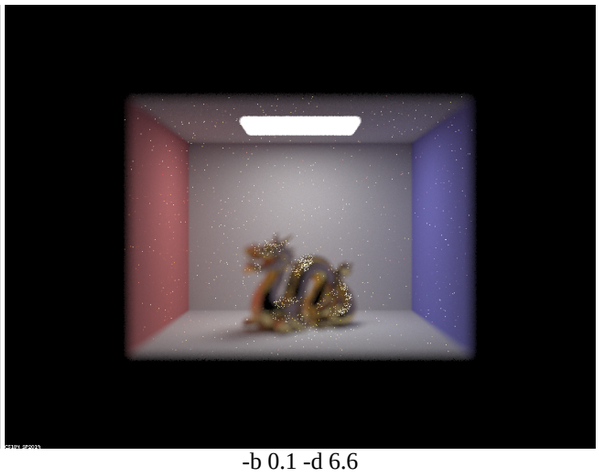
$$\text{pLens} = (\text{lensRadius}\ \sqrt{rndR} \cdot \cos(\text{rndTheta}), \text{lensRadius}\ \sqrt{rndR} \cdot \sin(\text{rndTheta}), 0)$$

Then calculate the ray that originates from pLens, and set its direction towards pFocus (blue segment in the figure).Normalize the direction of the ray, perform the camera-to-world conversion for both its origin and direction. Let me show some amazing results.
A "focus stack" (focus at 4 visibly different depths through a scene).(-b: lens radius, -d: focal distance)

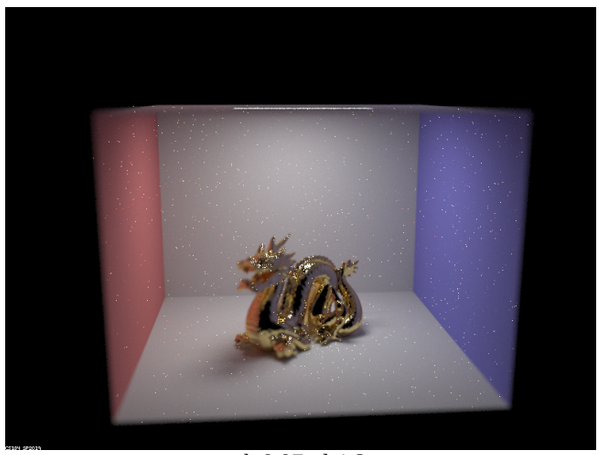

-b 0.1 -d 3.6

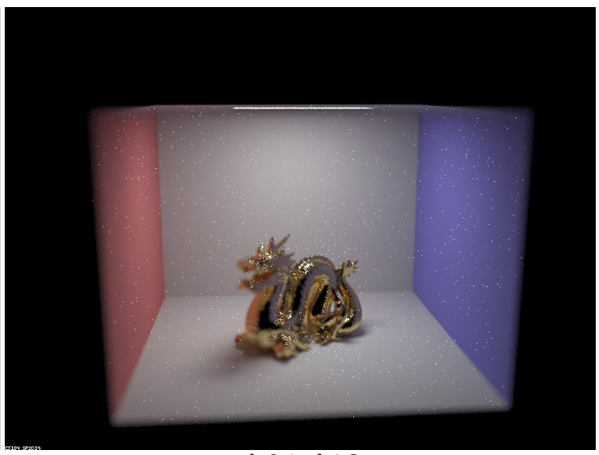

-b 0.1 -d 4.6

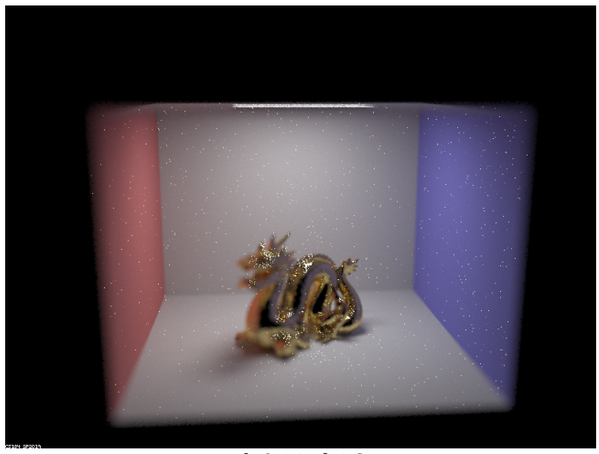

-b 0.1 -d 5.6


-b 0.1 -d 6.6

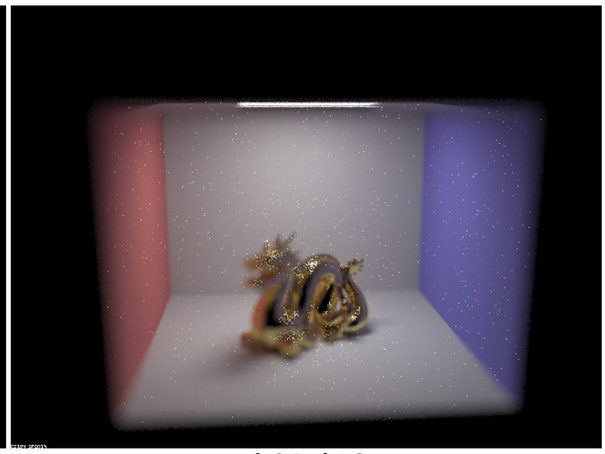
Then let me show a sequence of 4 pictures with visibly different aperture sizes, all focused at the same point in a scene.(-b: lens radius, -d: focal distance)


-b 0.07 -d 4.6


-b 0.1 -d 4.6



-b 0.14 -d 4.6



-b 0.2 -d 4.6

As we can see, with changes of lens radius and focal distance, the images can simulate the thin-lens camera.