Project 3-2 Write-up

Yaofu Zuo and Bohan Yu

Link to our webpage:
https://cal-cs184-student.github.io/sp22-project-webpages-yfz3357/proj3-2/index.html

An overview of the project

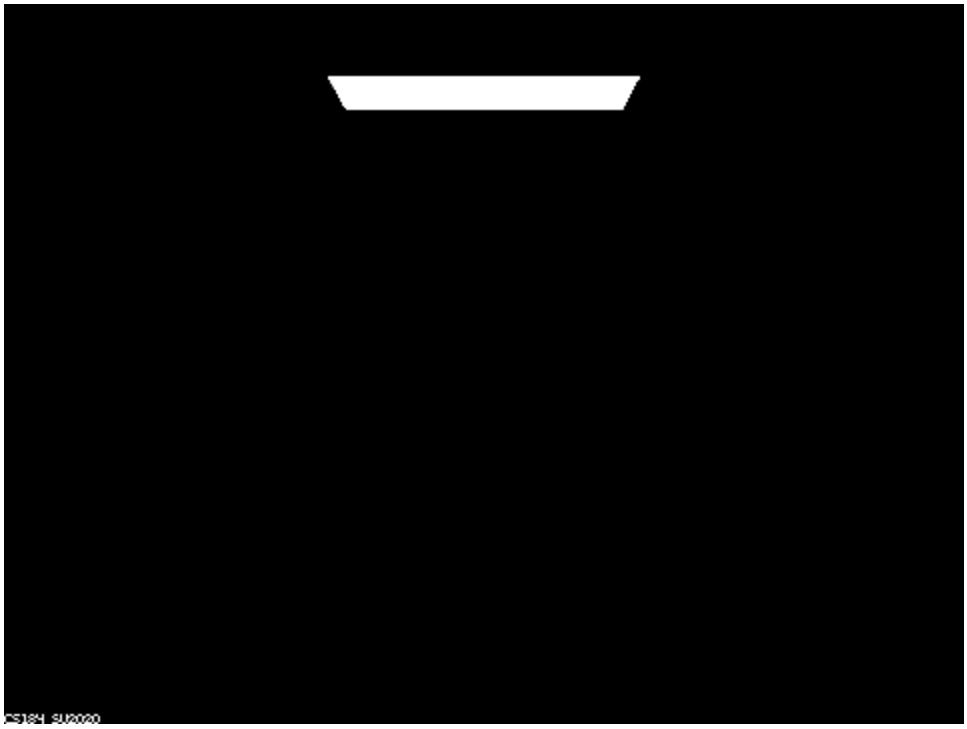
To “render pretty pictures”(as per checkpoint quiz), we implemented reflection and refraction for both mirror BSDF and microfacet BSDF.

Part 1:

Implementation:

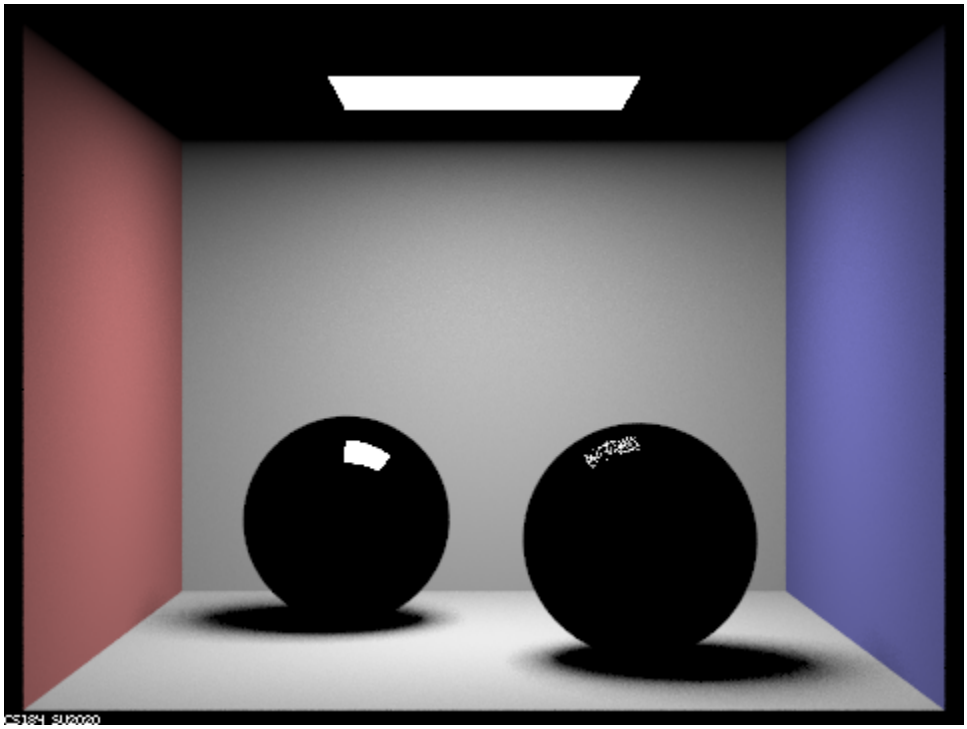
To simulate the effect of the reflecting and refracting, we strictly followed the formula and pseudo-code mentioned in the lecture and the spec. There aren't many places for us to make changes, so this part’s implementation is quite simple. Our implementation for Part 2 is described in more details as they are more complex.

M = 0:



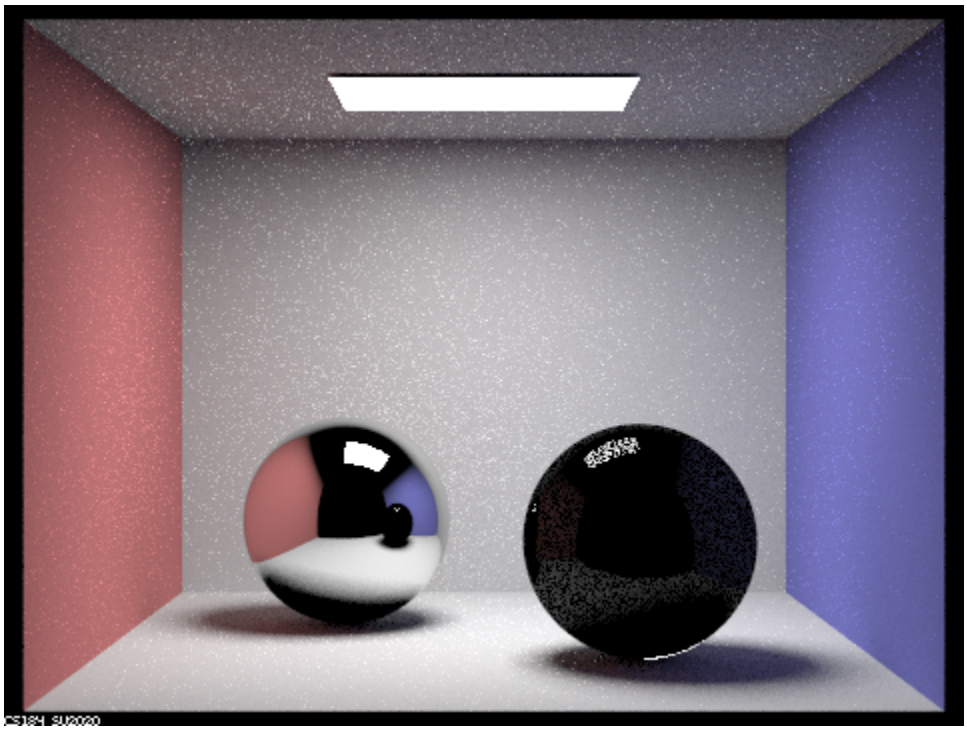
At M=0, no bounces are occurring, so we are only able to see the light source in the image, which is zero-bounce light.

M = 1:



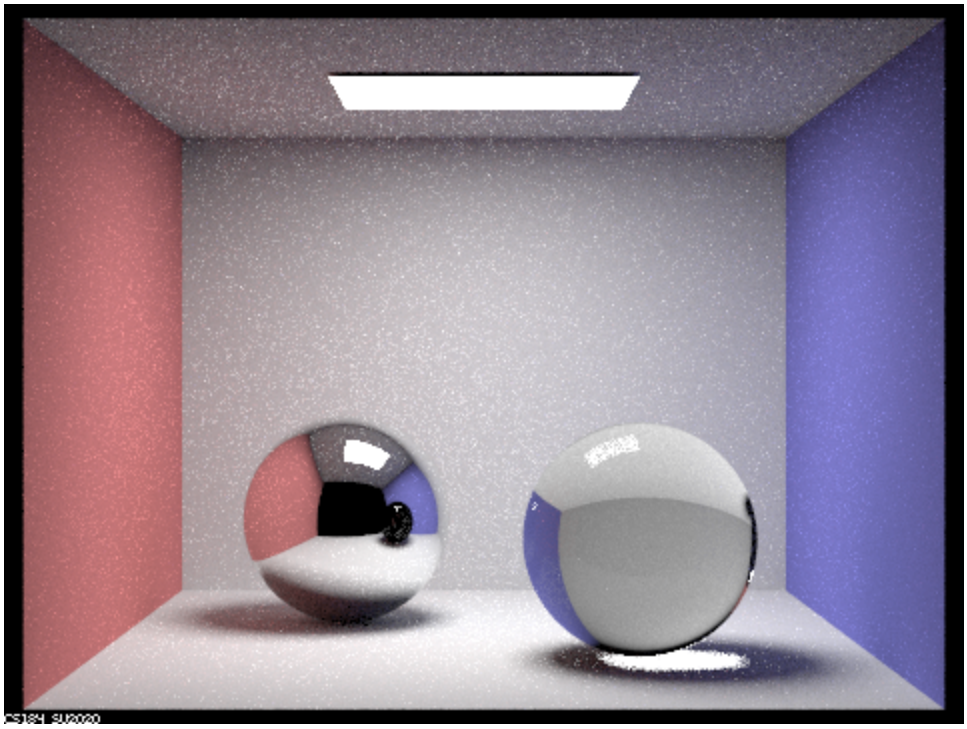
At M=1, only one-bounce rays from the light source are shown (i.e. direct illumination). Therefore, only the walls and a few pixels of the spheres that are directly under the light source are not black, as these places only take one bounce for the light source to bounce into the camera.

M = 2:



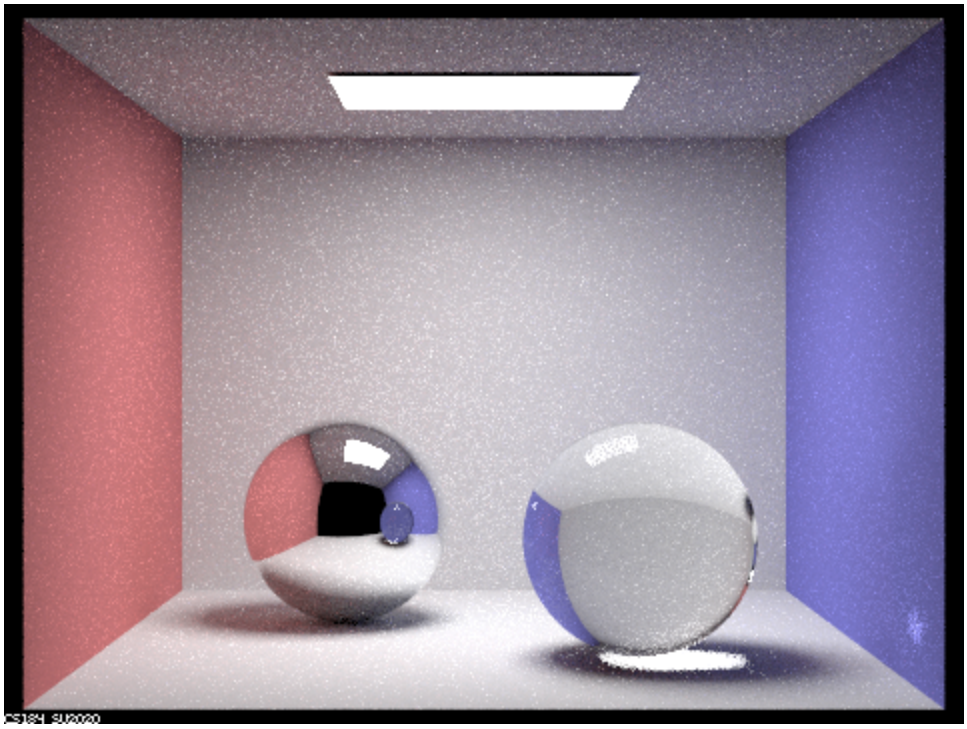
At M=2, we can additionally see lights that bounce twice before being received by the camera. Therefore, we are able to see a mirror image of the walls, the light source, and a sphere in the left sphere because it takes two bounces for these lights to arrive at the camera (e.g. first bounce off the wall, and then bounce off the left sphere). There's also some slight changes in the right sphere, where some two-bounces lights allow us to blurrily see a mirrored gray wall and the light source. The area of the mirrored light source on the right sphere actually increases as two-bounce light rays increase the solid angle from which the camera can see the light source.

M = 3:



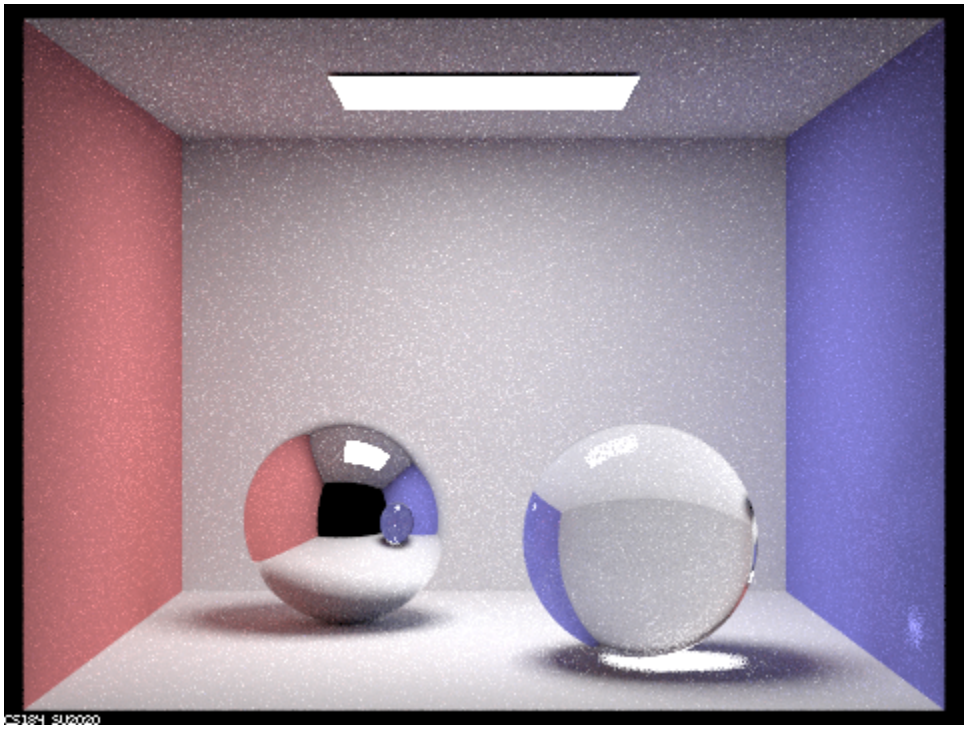
At M=3, we are able to see the refraction effect in the glass sphere that takes a total of three bounces: bounce off the walls → enters glass sphere → exits the glass sphere. The whole scene is also brighter as more lights are bounced off and received by the camera. For example, the top of the left sphere used to be black when M = 2, but now it's mirroring the roof.

M = 4:



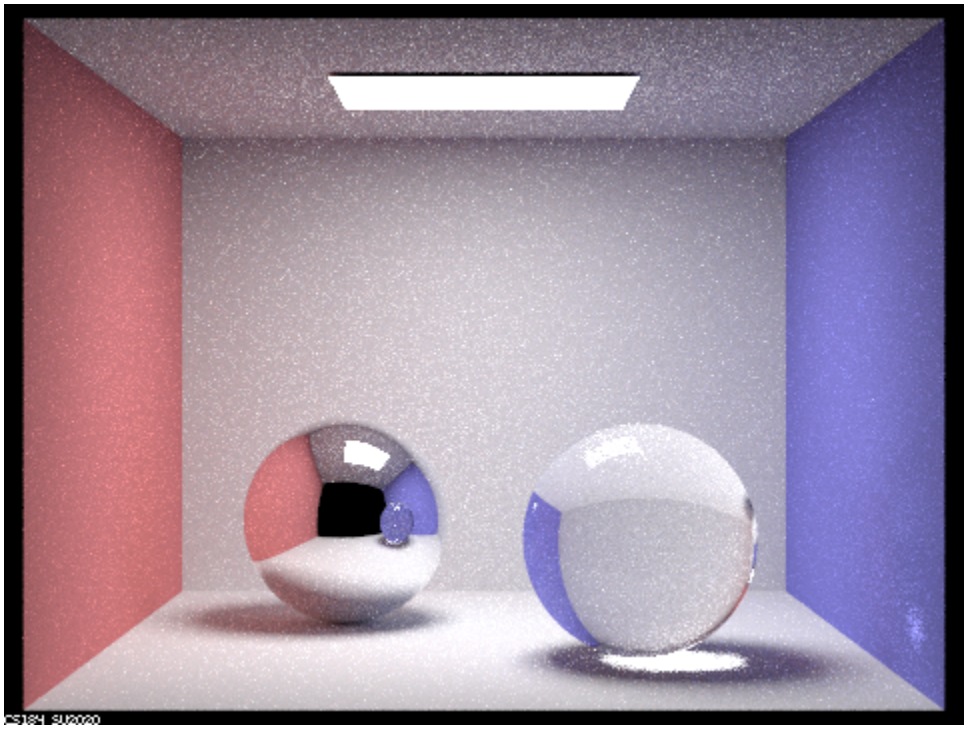
At M=4, we can clearly see that in the left sphere, it mirrors a blue right sphere. That means the light takes the following four bounces: bounce off the blue wall on the right, enter the back of the right sphere, exit the back of the right sphere, and finally bounce off the left sphere.

M = 5:



The difference between M=4 and M=5 isn't huge. We can see less noise when M = 5, and a slightly brighter scene overall. One of the reasons for not observing a huge difference is because the intensity of the light rays that bounced 5 times are small compared to those with less bounces. It's also because in this scene, M = 4 can capture most of the lights, either reflected or refracted.

M = 100:



As we increase M to 100, we see a slight increase in the overall lighting of the scene. We can't observe much difference between M = 100 and M = 5, since lights that take many bounces to reach the camera are either pre-terminated by the Russian roulette and adaptive sampling, or the change in intensity caused by one more bounce is too small to be observed.

The rate map for
M=5:

M=100:

This is due to the implementation of adaptive sampling. Because most light rays that bounced 5 times are weak, the illuminance of such ray sample might be too low so that it doesn't pass the threshold we defined in Part 5 in Proj 3-1:

$$I \leq maxTolerance \cdot \mu,$$

Thus, the ray tracing most likely had terminated early before reaching depth of 100, so our image rendered at M=5 is almost identical to image rendered at M=100.

Part 2:

Implementation:

1. MicrofacetBSDF::f()

Before we apply the formula in the lecture to simulate MicrofacetBSDF, we check if the denominator of the formula is larger than zero, as invalid $wo,\ w_i$ will result in a zero dot product, in which case we simply return a zero vector. If both $wo,\ w_i$ are valid, we then calculate the f according to the following formula:

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

, where both $F(w_i)\ and\ D(h)$ are implemented as the following (in the next two parts), and $G(wo,\ w_i\ )$ is already given in the code.

2. MicrofacetBSDF::D()

We need to first calculate $\theta_h\ =\ arcosine\ (\ dot\ (n,\ h)\ /\ (|n|\ *\ |h|))$. Since both h and n are unit vector, we know that θ_h can be simplified to h.z ,the z component of h. We also know that α is given in the code as the “roughness” factor. So can calculate $D(h)$ according to the following formula and return the final result.

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

3. MicrofacetBSDF::F()

With similar simplification above, we know that θ_h is simply the z component of h_i. Since both η and k are given, we can calculate the final F according to the formula below:

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

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

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

4. MicrofacetBSDF::sample_f()

We first randomly sampled two floats r1 and r2 within [0, 1). Then we applied the formulae below to get the two sampled angles θ_h and ϕ_h:

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

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

With sampled angles in spherical coordinates, we can calculate the following:
$h.x = (sin(\theta_h) * cos(\phi_h),\ h.y = sin(\theta_h) * sin(\phi_h),\ h.z = cos(\theta_h))$
Then we calculate the sampled w_i according to the following formula:
$w_i = 2 * dot(wo,\ h) * h - wo$
Lastly, we calculate $p_w(h)$ according to the following formula, where we also calculated p_theta(h) and p_phi(h):

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

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

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

$$\begin{aligned} x &= r\cos\varphi\ \sin\theta, \\ y &= r\sin\varphi\ \sin\theta, \\ z &= r\cos\theta. \end{aligned}$$

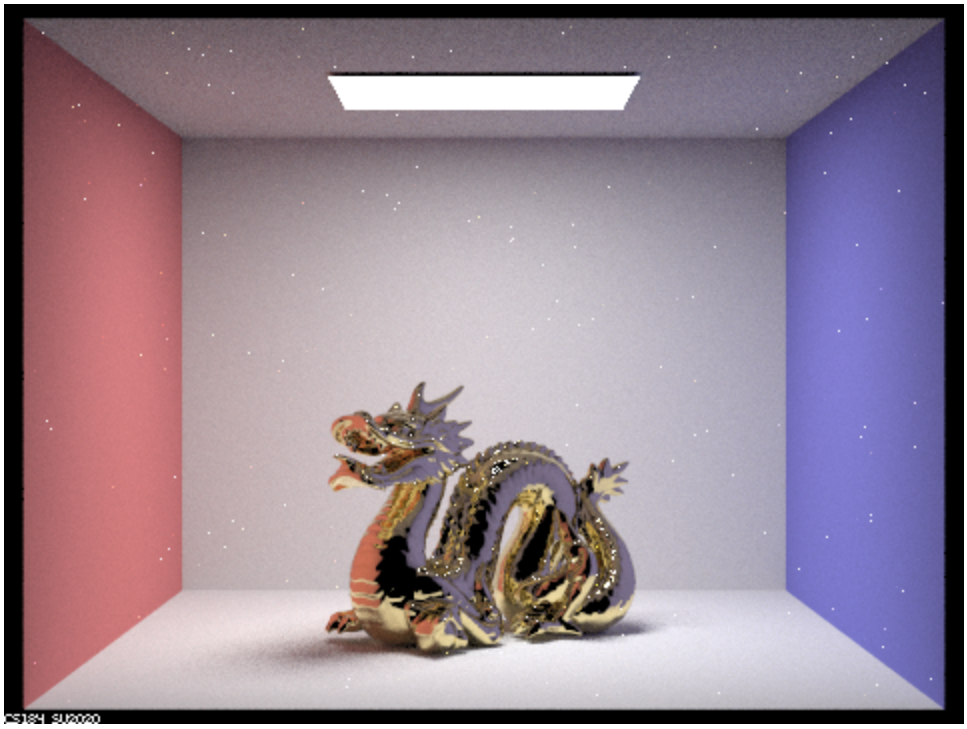
To complete the pdf, we calculated
Using w_i, we calculated $p_w(w_i)$ and assigned it to pdf:

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

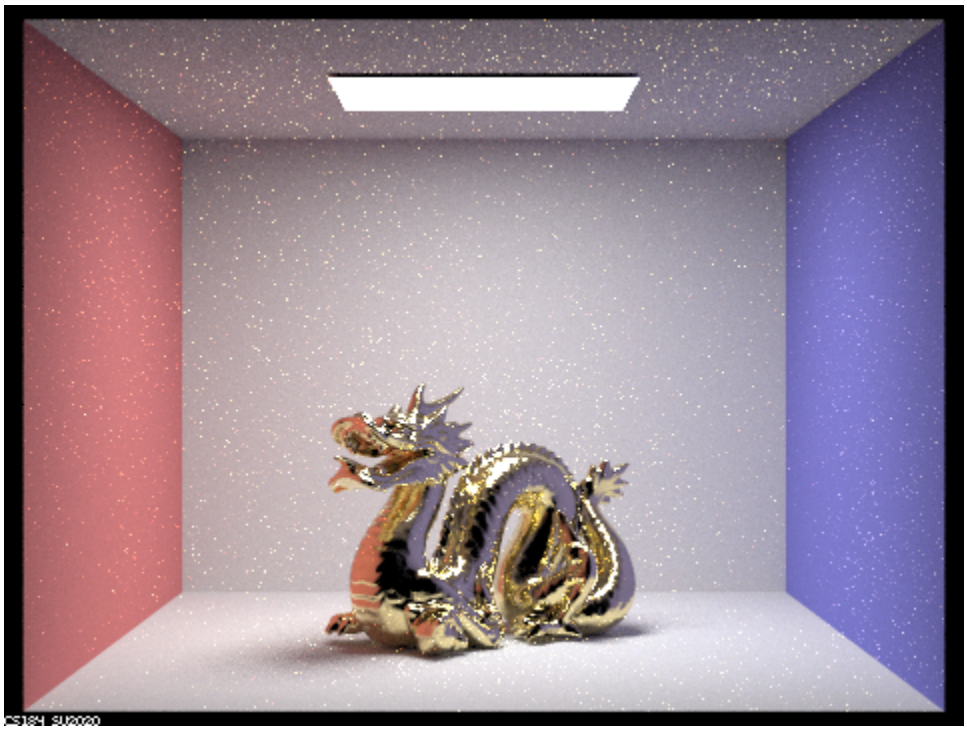
At last, we returned MicrofacetBSDF::f(wo, *wi).

The below images are rendered with 128 samples per pixel, 1 sample per area light, max ray depth of 5, and resolution of 480x360.

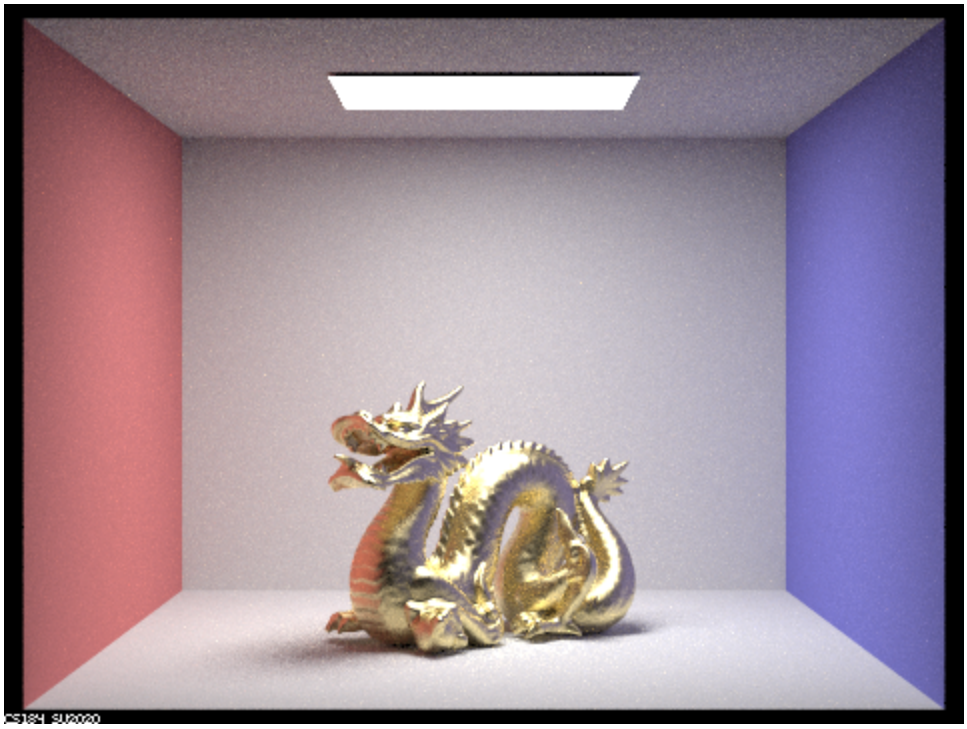
α = 0.005:



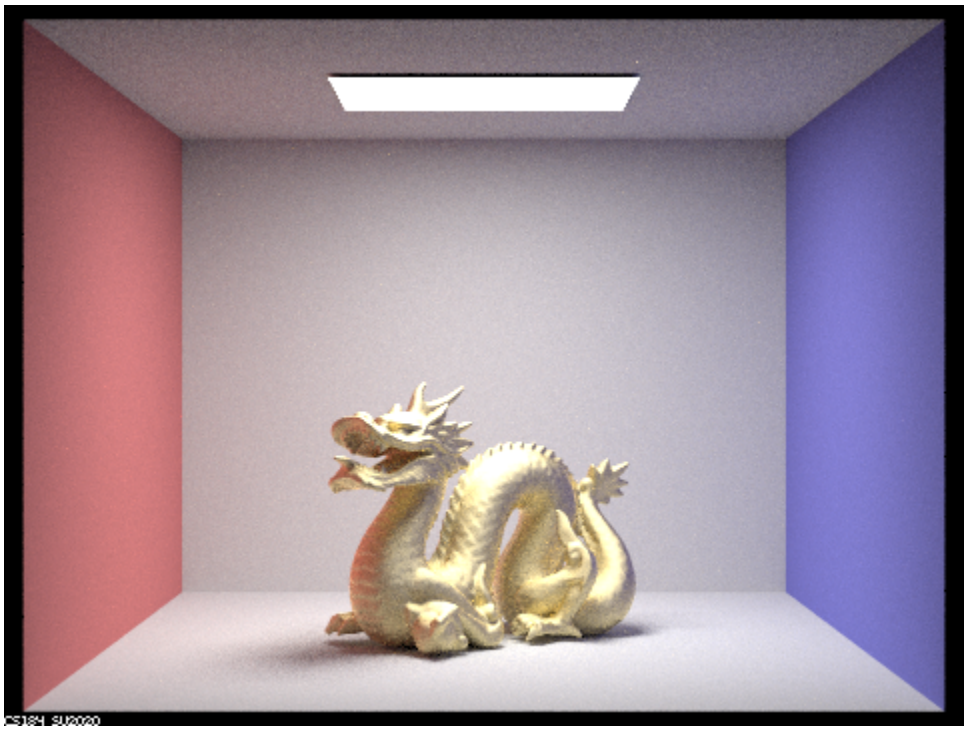
α = 0.05:



α = 0.25:



α = 0.5:



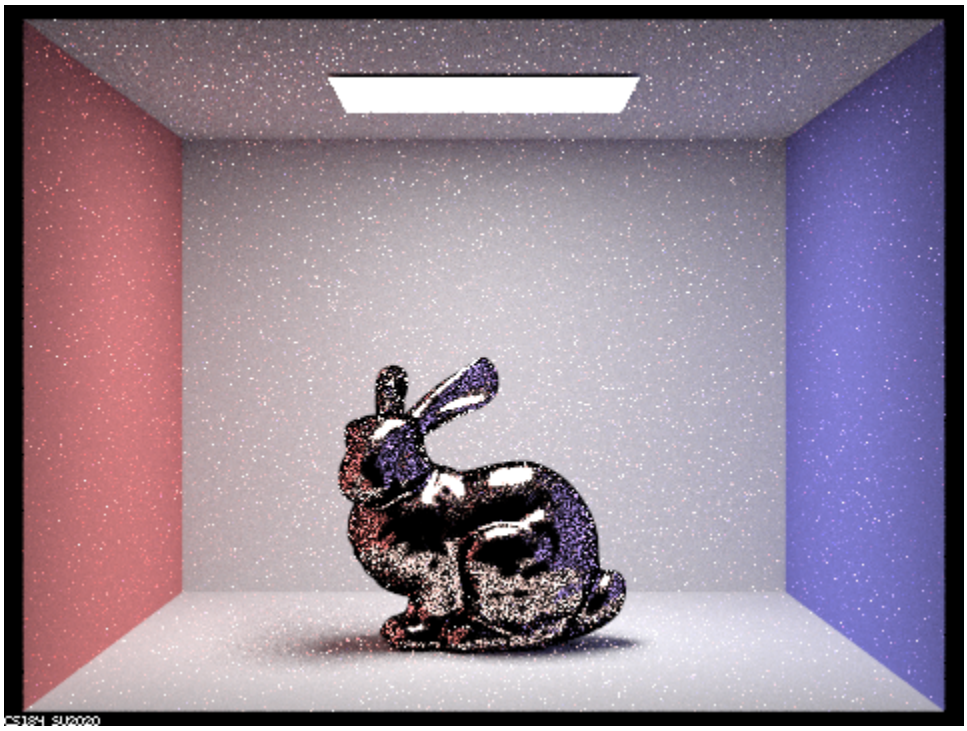
As α increases, the surface of the dragon seems to have a brighter colour. Also, the noise(random white pixels) decreases as α increases. Both of the above circumstances described above are most likely due to an increase of light reflection on the surface of the

dragon, as more reflected light results a brighter dragon, and more reflected light means the more likely that a ray direction sampled in a pixel actually points towards the light source(so less noise).

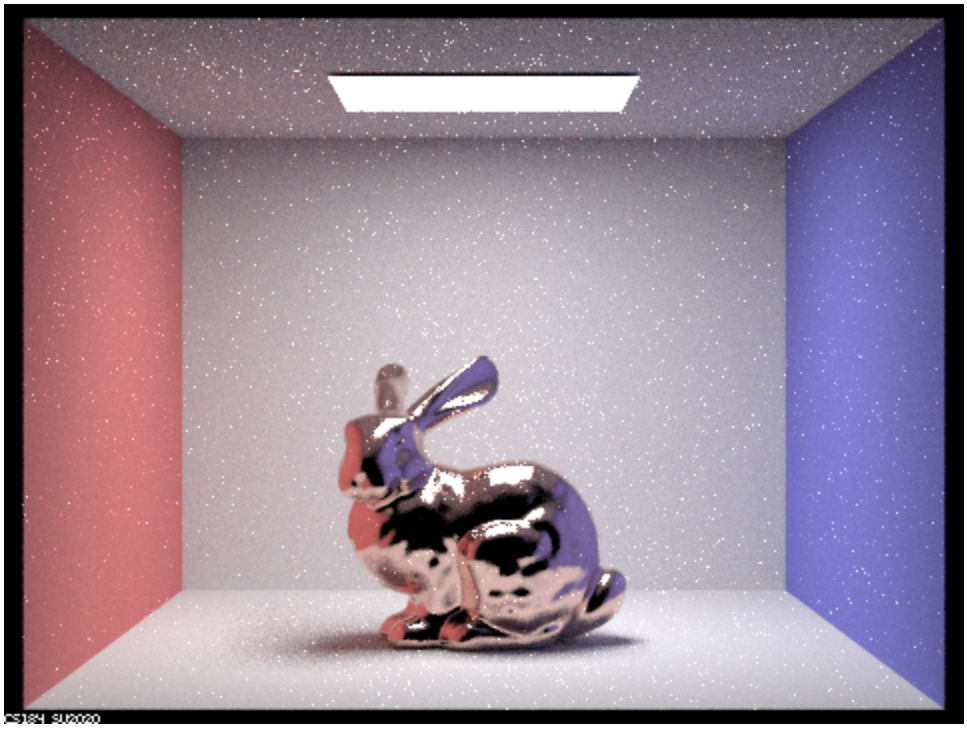
- Show two images of scene `CBbunny_microfacet_cu.dae` rendered using cosine hemisphere sampling (default) and your importance sampling. The sampling rate should be fixed at 64 samples per pixel and 1 samples per light. The number of bounces should be at least 5. Briefly discuss their difference.

The images below are rendered at 64 samples per pixel, 1 samples per light and 5 bounces:

Cosine Hemisphere:



Importance Sampling:

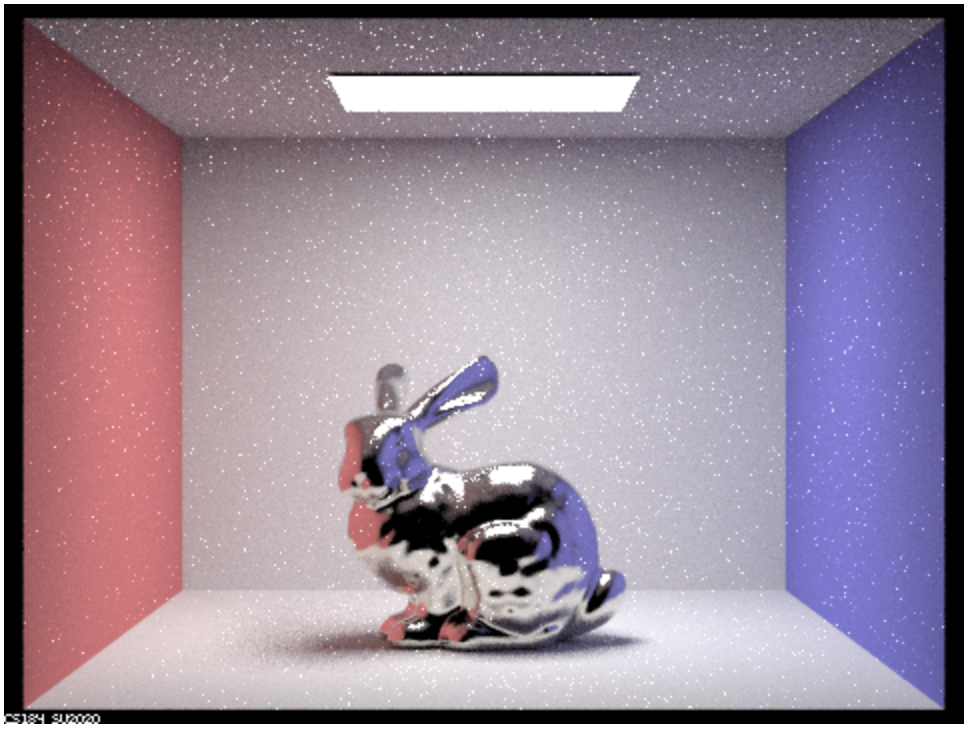


There is a notable decrease of noise(random white pixels) by using importance sampling instead of cosine hemisphere sampling. This is, most likely, due to that importance sampling would more likely sample a direction that points towards light source, so the more likely the pixel's color is correctly rendered.

- Show at least one image with some other conductor material, replacing `eta` and `k`. Note that you should look up values for real data rather than modifying them arbitrarily. Tell us what kind of material your parameters correspond to

We used the material Calcium(Ca), and using the website provided by the specs, we got eta = <0.29294, 0.28568, 0.28733>, k = <2.7181, 2.3479, 1.8333>.

The images below are rendered at 64 samples per pixel, 1 samples per light and 5 bounces:



- At the end, if you worked with a partner, please write a short paragraph together for your final report that describes how you collaborated, how it went, and what you learned.

We collaborated by using pair programming, not just because one of us uses M1 Chip but also because it's easier to debug programs that require scrutiny—never mis-type the formula! We switched our roles of driver and navigator after completing one task. The navigator is the one who is always thinking a few steps ahead, providing macroscopic instructions and microscopic improvements and corrections that the driver misses. For example, the navigator is responsible for tracking the current progress, knowing what's the next step, and discovering any bugs that the driver implemented. The driver is the one who actually writes out the code. With this type of collaboration, this project went pretty well, where we only spent a few minutes debugging and most of the bugs were discovered by the navigator right away. Together we learnt how different material works computationally, and we learnt how to efficiently and smoothly write a project.