CSC4140 Assignment VII

Computer Graphics

May 1, 2022

Ray Trace part II

This assignment is 8% of the total mark.

Strict Due Date: 11:59PM, May 1^{st}, 2022

Student ID: 118010202

Student Name: Zhixuan LIU

This assignment represents my own work in accordance with University regulations.

Signature: Zhixuan Liu

1 Overview

In this project, I finished the Part 1, 2 (task 1, 2, and 3), 3 and part 4. In the first two tasks, I extend the capabilities of ray-tracer for different materials with different BSDF properties. To be specific, right now my ray tracer can implement tracing glass, mirror, and objects with different materials. In the part 3, I simulate how environment light affects the object rendering. Lastly, in part 4, we consider the Depth and Field of ray tracing, make the ray tracing and rendering much more similar to humanbeings and real-world camera effects.

2 Part 1: Mirror and Glass Materials

To simulate the mirror and glass materials in the real world, I followed the knowledge of physics in the reflection and refraction. For an incoming ray, we want to get the refection and refraction ray using the corresponding functions. For the $BSDF :: reflect()$ function, the coming ray's x and y component will be changed in sign while the z component wouldn't change; and for the $BSDF :: refract()$ function, the Snell's Law is used for the refraction simulation.

Then, I implement the $RefractionBSDF :: sample_f()$ function for mirror objects. Mirror materials rely purely on reflection, so after implementing a function to help us reflect about the object normal, we simply called it in the sampling function, set the pdf to 1, and returned the reflectance divided by $abs\_cos\_theta(*wi)$.

I also implement the $GlassBSDF :: sample_f()$ funtion for the glass objects. And I use the Schlick's approximation to decide the ratio of the reflection energy to the refraction energy.

Now Let's see the results of the rotation:

Figure 1: CBsphere.dae with m=0.

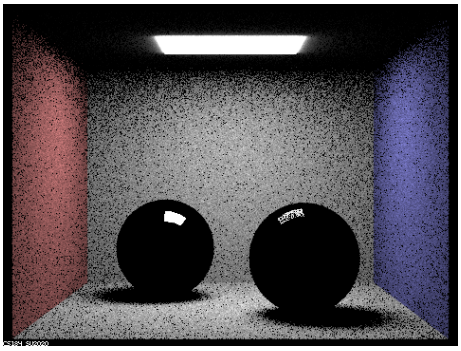

Figure 2: CBsphere.dae with m=1.

Here I want to mention some huge changes. When m=0, it equals zero-bounce radiance. The only rays that goes into the camera and show on the screen come from the light sources.

When m=1, which means we can see the light the have one reflection effect, so we can see the left mirror ball's showing the source light, and since the right ball is a glass ball, it also has the

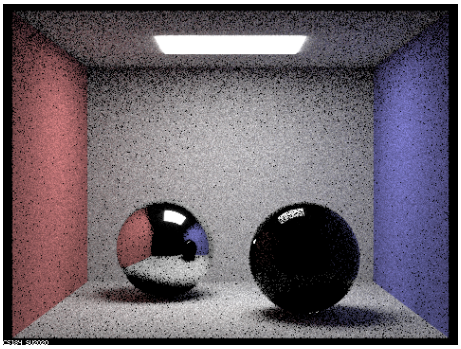

Figure 3: CBsphere.dae with m=2.

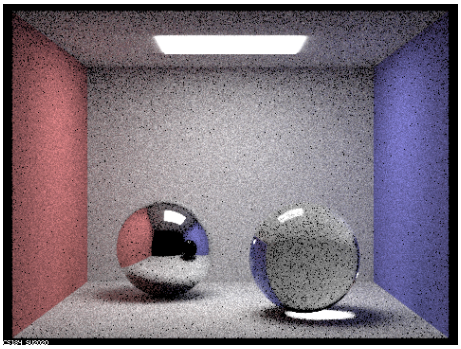

Figure 4: CBsphere.dae with m=3.

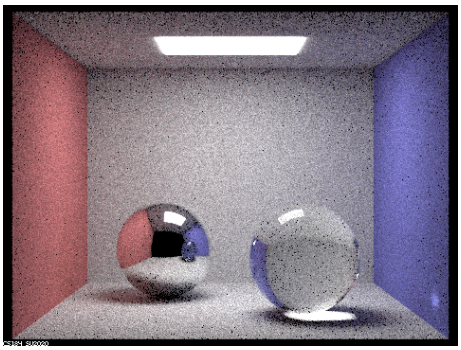

Figure 5: CBsphere.dae with m=4.

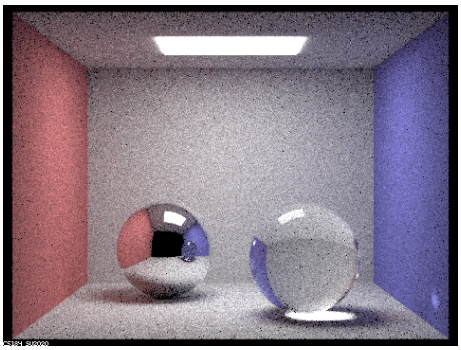

Figure 6: CBsphere.dae with m=5.

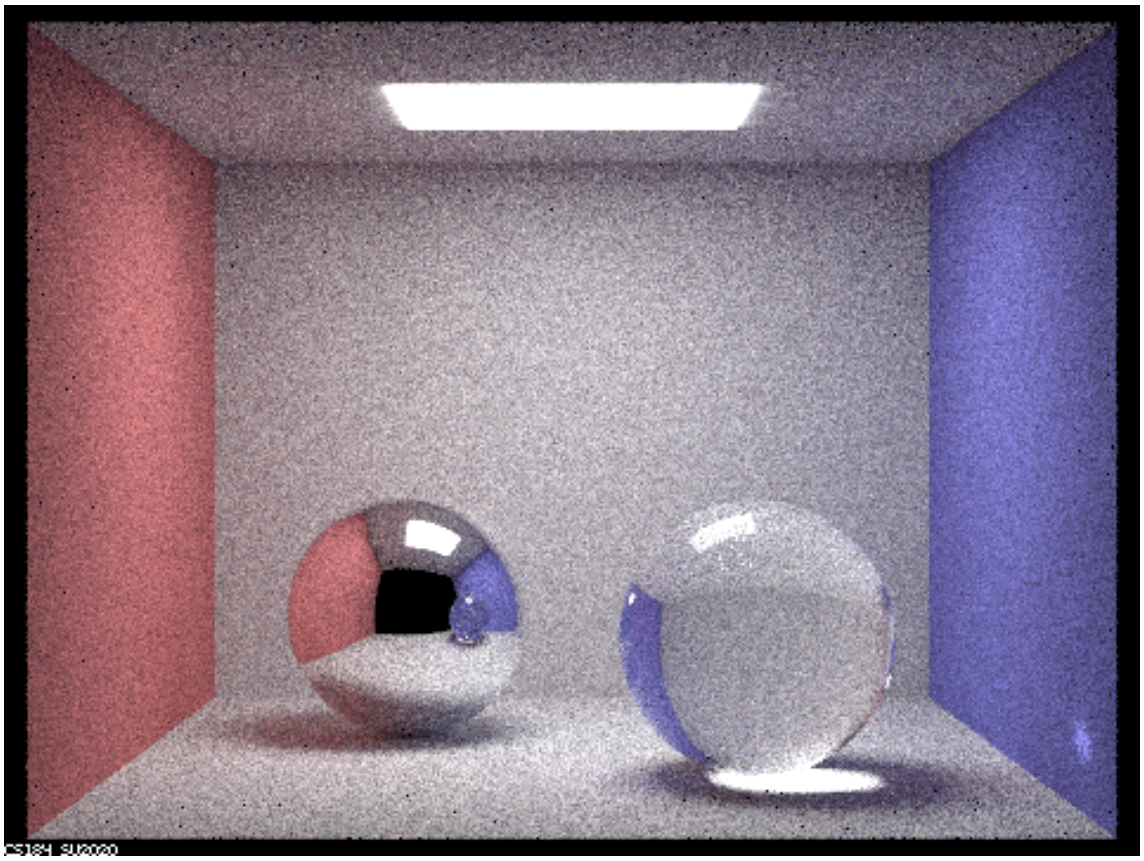

Figure 7: CBsphere.dae with m=100.

property of reflection, therefore, we can also see the reflected source light from the right ball but the effect is not as strong as the right one.

When m=2, we can see the reflection of the environment on the left mirror ball, since the rays first touch the wall, then touch ball, and next to the camera. However, the right ball is still dark because the refraction need 3 steps to get out of the ball.

So we can see that when m=3 and larger, we can see the effect of the refraction and are able to see the right ball.

When m=4 and larger, we can see the light that go though the right ball to the purple wall. This is because the bounce number allow the light to go into and out of the right glass ball and then to the right purple wall.

3 Part 2: Microfacet Materials

I only finished the first three tasks of this part, because in part 4 and 5, I need to render microfacet objects, therefore, I do this to enable this feature. In this part, I implemented a BSDF for objects of Microfacet Materials.

The microfacets is set to be perfectly specular reflectors. I also follow the Beckmann Distribution in order to define the distribution of half-vectors. This distribution contains a tunable roughness parameter which spreads out the distribution as it approaches 1 and narrows the distribution as it approaches 0. In the following figures, I change the α in the .dae file of the dragon_au. One thing to notice is that, the sample rate I choose is very small for convenient, therefore, the results are not super good. And I also didn't do the importance samping.

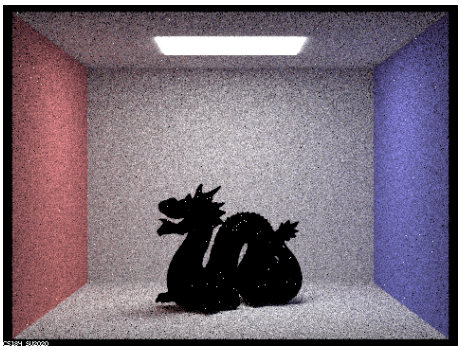

Figure 8: Dragon_au alpha=0.005

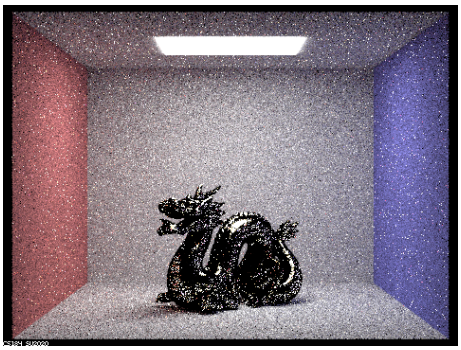

Figure 9: Dragon_au alpha=0.05

4 Part 3: Environment Light

In this part, I implement the environment light effect. Unlike the ray comes from the light sources, the light ray in the environment light is from infinitely far away and supplies incident

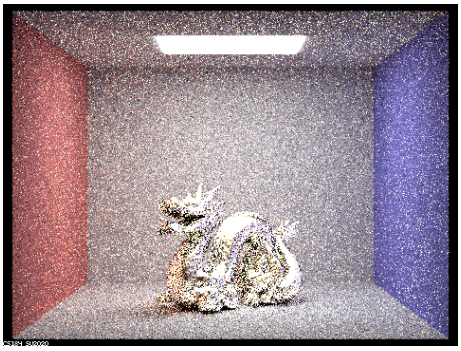

Figure 10: Dragon_au alpha=0.25

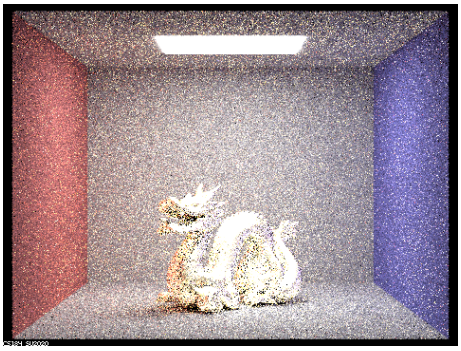

Figure 11: Dragon_au alpha=0.5

radiance from all directions on the sphere. Therefore, it looks much more real than the things we rendered before.

In this section, I follow the instruction and use the pre-stored texture maps , from which I extract the light intensity and do the sampling. There are two ways to do the sampling, the first is the uniform samping: a random orientation is generated on the sphere, and the bi-linear interpolation is implemented on the texture map. I also do the importance sampling. It is a common sense that the brighter light resource affect more of the whole rendering process, therefore, the sample should be more important and should take into more consideration for the much brighter objects.So the rendering results can be improved by doing in this way.

Figure 12: field.exr file

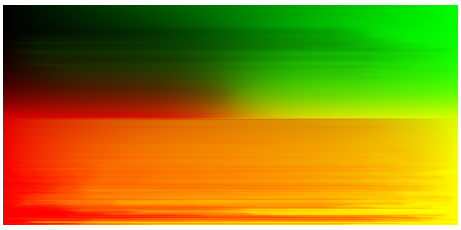

Figure 13: probability_debug_field

Figure 14: ennis.exr file

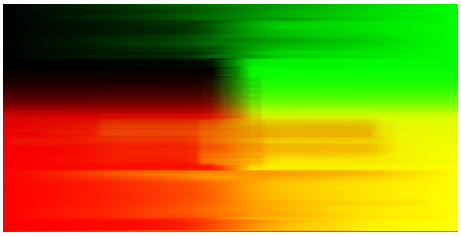

Figure 15: probability_debug_ennis

And here I show two comparison results using Uniform (Bi-linear) sampling and the Importance sampling. As we can see the results, the quality of the importance sampling is much better than the Uniform sampling, especially we can tell from the first roll. And the left, using the uniform sampling are much more noisy than the right figure.

And for The difference is relatively subtle, but there is a fair bit more noise in the image

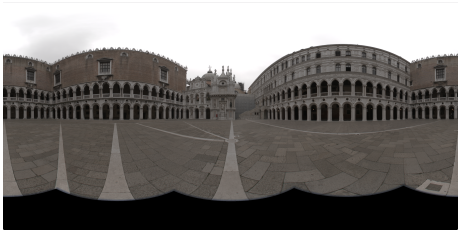

Figure 16: doge.exr file

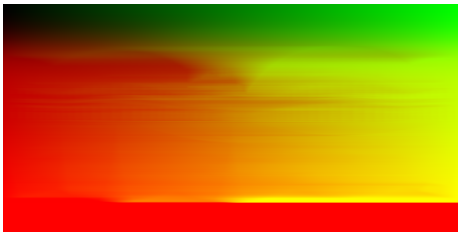

Figure 17: probability_debug_doge

Figure 18: bunny_unlit uniform sampling

Figure 19: bunny_unlit importance sampling

rendered using uniform sampling not present in the importance sampled image. Consequently, there is a slight obscuring of some of the finer details in the bunny in the uniform sampled image when compared to the image of the bunny rendered using the importance sampling.

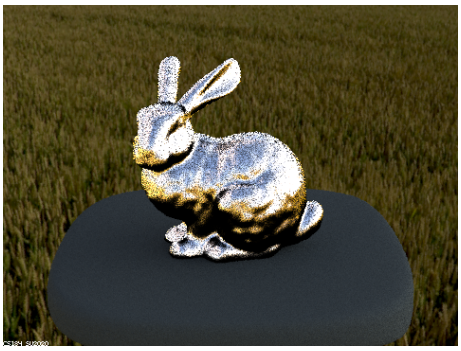

Figure 20: bunny_microfacet_cu_unlit uniform sampling

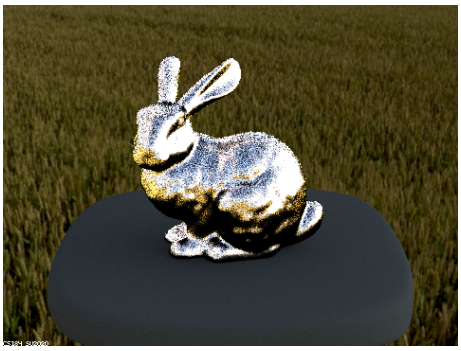

Figure 21: bunny_microfacet_cu_unlit importance sampling

5 Part 4: Depth of Field

Previously, we were using ideal pin-hole cameras. With the pin-hole model, everything is in focus. However, real cameras, including human eyes, are actually modeled as lenses with finite apertures defined by *lensRadius*. With the lens model, objects are in focus only if they're within a plane that is at $focalDistance$ from the lens.

As opposed to the pin-hole model, where any point on the film receives radiance from only one point in the scene, with a thin lens model, any given point on the film can receive radiance from

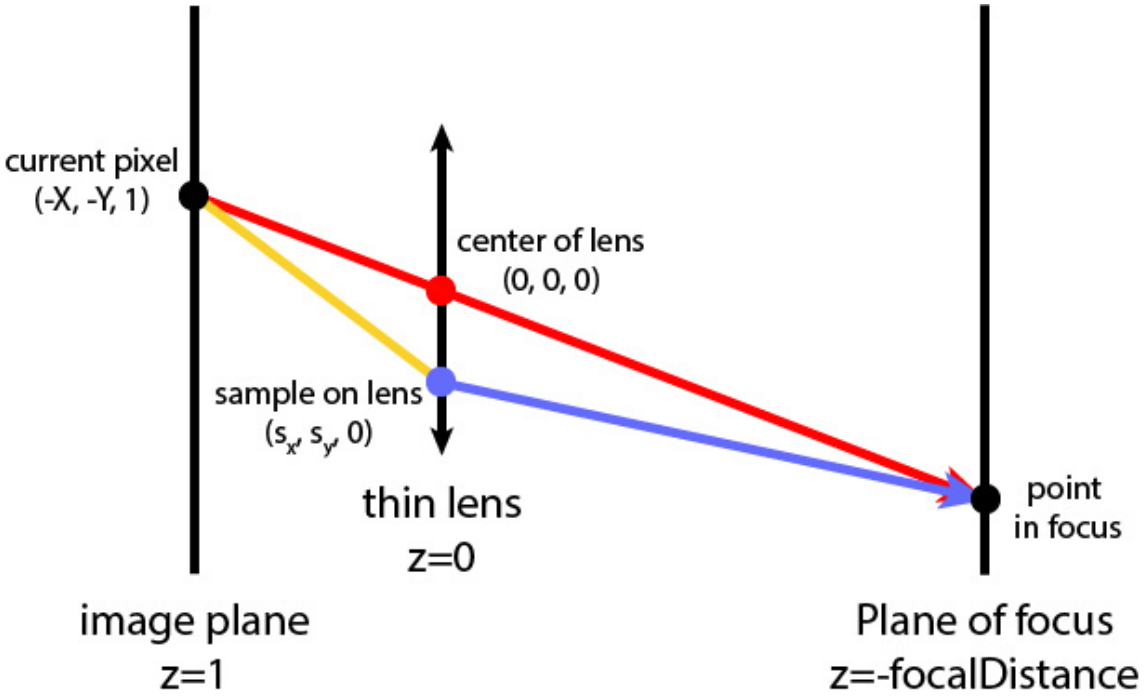

Figure 22: The Depth and Field I simulated.

any point on the thin lens. Therefore, we adjust our camera ray generation to sample the thin lens accordingly.

I followed the instructions to re-write the function of generate ray in the last assignment, instead I generate rays for a thin lens closely followed the instructions.

Here some results will be leased:

5.1 Change focal distance while apertures $lensRadius = 0.6$

In the results shown below, I set the apertures property, which is the *lensRadius* to be 0.6 while I change the focal distance. We can see that, with the increase of focal Distance, the camera focus is moving from near to far. For example, in the first figurem with focalDistance set to be 3.5. the focus is on the pink wall, which is closer to the camera.

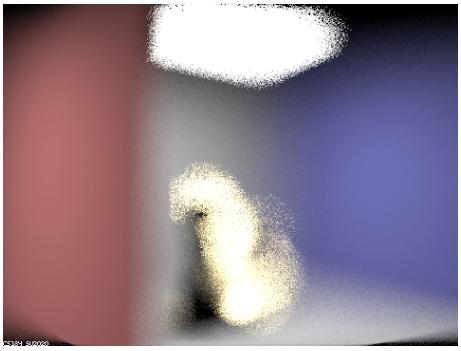

Figure 23: focalDistance = 3.5, lensRadius = 0.6

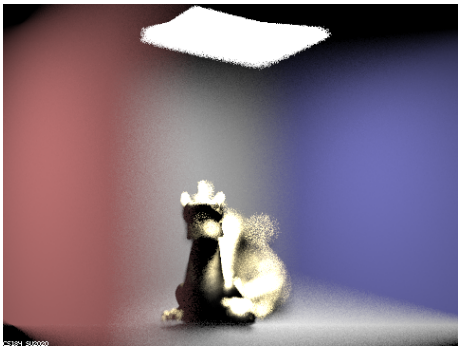

Figure 24: focalDistance = 4.1, lensRadius = 0.6

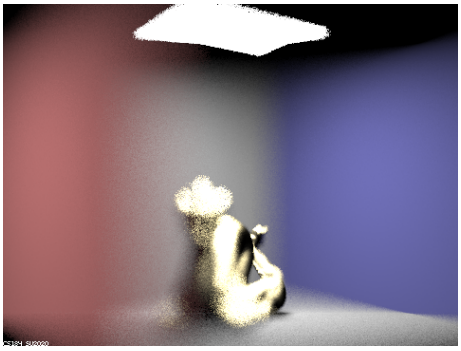

Figure 25: focalDistance = 4.6, lensRadius = 0.6

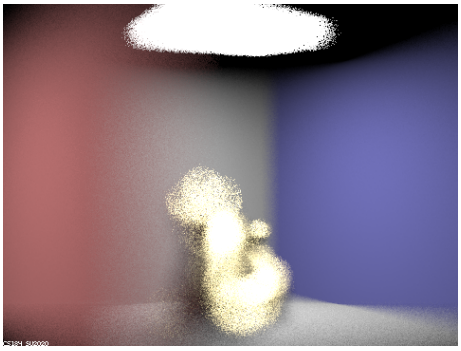

Figure 26: focalDistance = 5.1, lensRadius = 0.6

5.2 Change apertures *lensRadius* while $focalDistance = 4.2$

Since we keep the focal Distance to be 4.2, which is the head and front body of the dragon, with the increment of *lensRadius*, only object within the plane can retain the sharpness and be perfectly rendered as usual, while other parts of the dragon will be blurred.

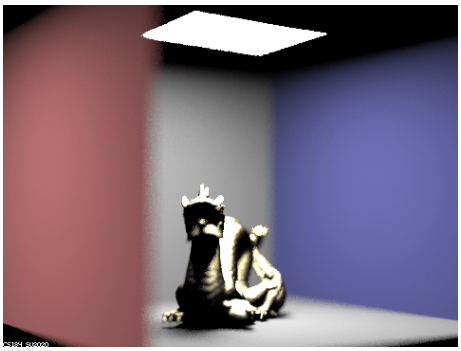

Figure 27: focalDistance = 4.2, lensRadius = 0.15

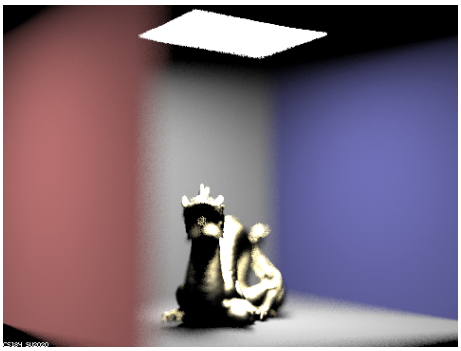

Figure 28: focalDistance = 4.2, lensRadius = 0.25

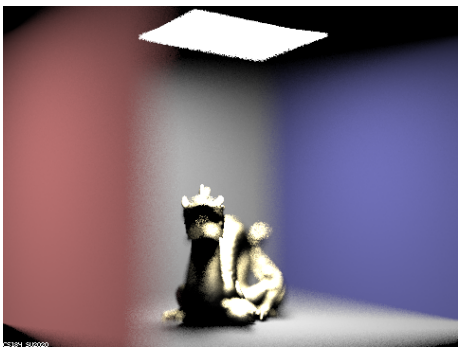

Figure 29: focalDistance = 4.2, lensRadius = 0.35

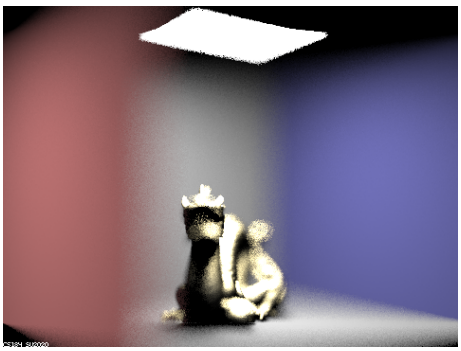

Figure 30: focalDistance = 4.2, lensRadius = 0.45