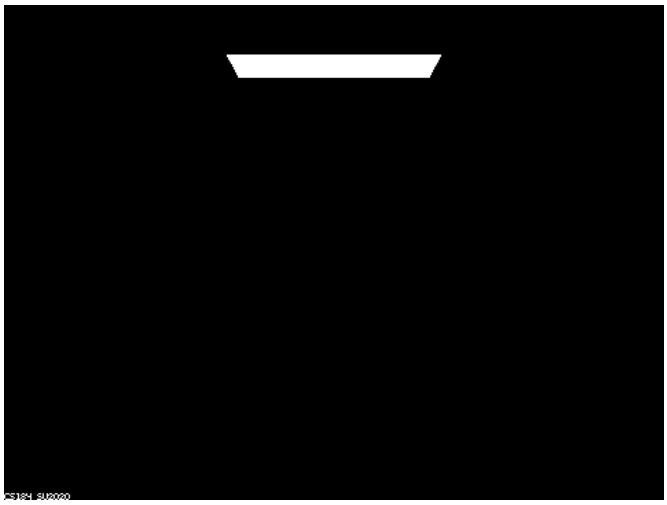
Assignment 3: PathTracer

Jacob Hsiung

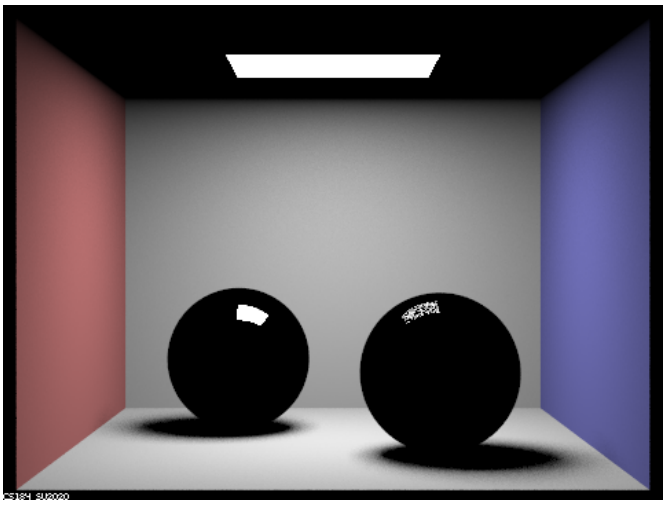
This project adds more feature to pathtracer, so we can render image with different material and simulate cameras with different focal distance and aperture size. In the first part of the project, I implemented the reflection and refraction function for mirror and glass material. Using these two function and Schlick's approximation, we can implement the bsdf of these two materials, so we can get the radiance and angle of the ray being reflected or refracted off the surface of different material. Secondly, to simulate the visual effect or real-world camera, we have to implement a function to calculate the ray when using thin lens model.

Part 1: Mirror and Glass Material

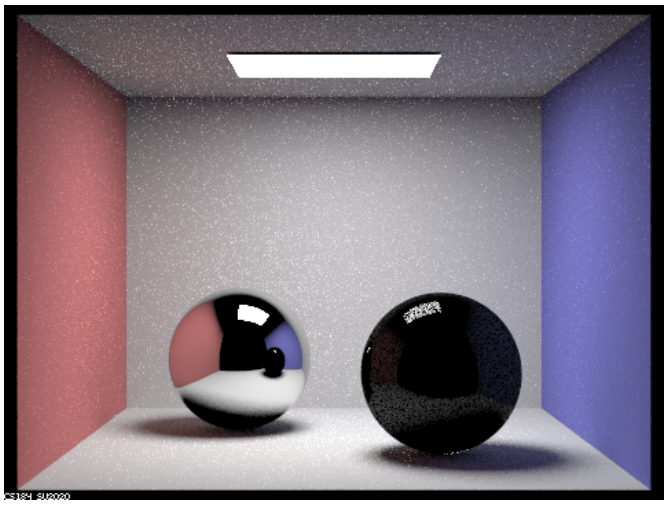
In this part of the project, we will implement the BSDF of different material. For mirror material, the only BSDF is reflection, and we can calculate the value of wi by switching the sign of x,y components of wo and copy the value of z components. Afterwards, we can set the pdf to 1. However, things are more complicated when it comes to Glass material because it has not only reflection but also refraction. To begin with, we can keep using the reflect function implemented earlier and add on a refract function. In the refract function, we have to first confirm that if the light is entering the glass or exiting the glass material. If it is the former case, then the eta value will be 1.0/ior and ior otherwise. With the help of snells law and algebra, the x components of wo is -eta*wo.x, the y components of wo is -eta*wo.y, and the z components of wo is +/- sqrt(1 - eta^2(1 - wo.z^2)). The plus minus indicates that wi.z has to be opposite sign of wo.z. Furthermore, if 1 - eta^2(1 - wo.z^2) is smaller than 0, then this indicates that we have a total internal reflection, and in that case we would return false for this function. In the RefractionBSDF::sample_f function, we call refract and if there is no total internal reflection then transmittance / abs_cos_theta(*wi) / eta^2 will be returned. Last but not least, to render the visual effect of glass material, we have to find a way to combine the reflection and refraction, and Schlick's approximation does exactly that. For GlassBSDF::sample_f, we start off by checking if there is total internal reflection, if so then we set the pdf to 1 and return reflectance / abs_cos_theta(*wi). Otherwise, we calculate the Schlick's refelction coefficient R = R0 + (1.0 - R0) * (1 - abs_cos_theta(*wi))^5, where R0 = (1 - ior)^2/(1 + ior)^2. We then pass in R as a parameter into coin_flip(p), which return true with probability p and false with probability 1-p. If coin_flip(R) then we treat this incident beam as refelction, so we return R * reflectance / abs_cos_theta(*wi) and set pdf = R. Otherwise, we treat the ligh as refraction and return (1.0 - R)* transmittance / abs_cos_theta(*wi) / eta^2 and set pdf = 1-R.



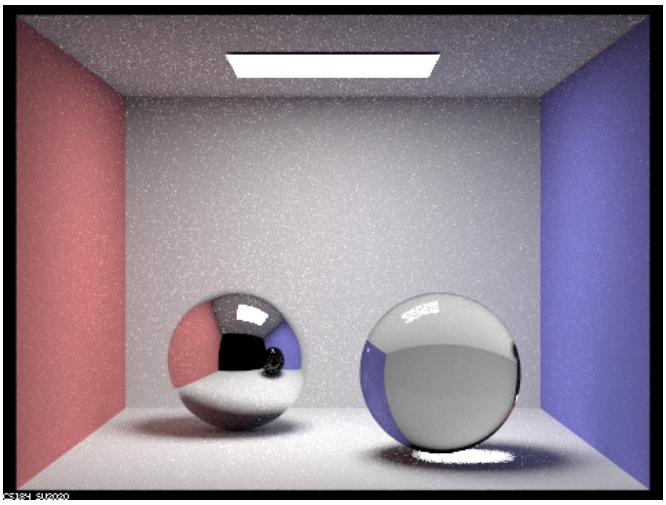
64 sapmle per pixel, 4 sample per light, m = 0



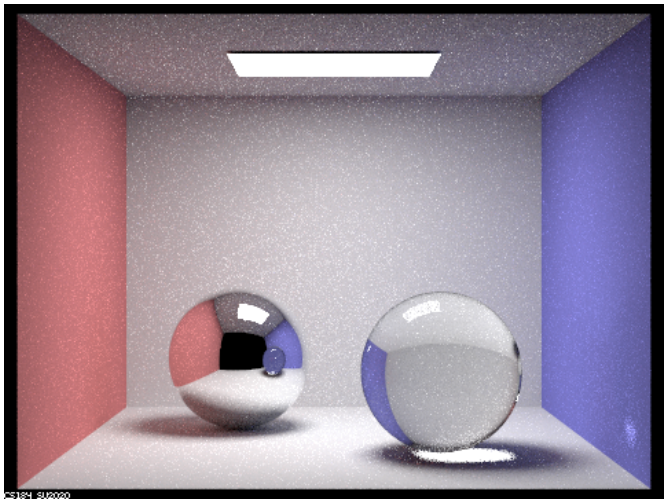
64 sapmle per pixel, 4 sample per light, m = 1



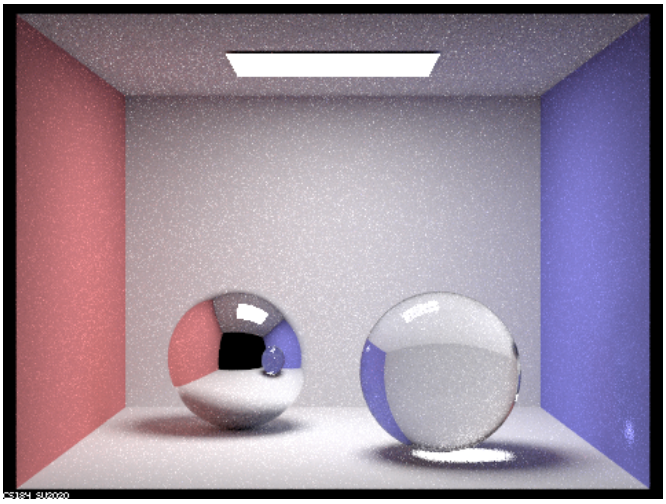
64 sapmle per pixel, 4 sample per light, m = 2



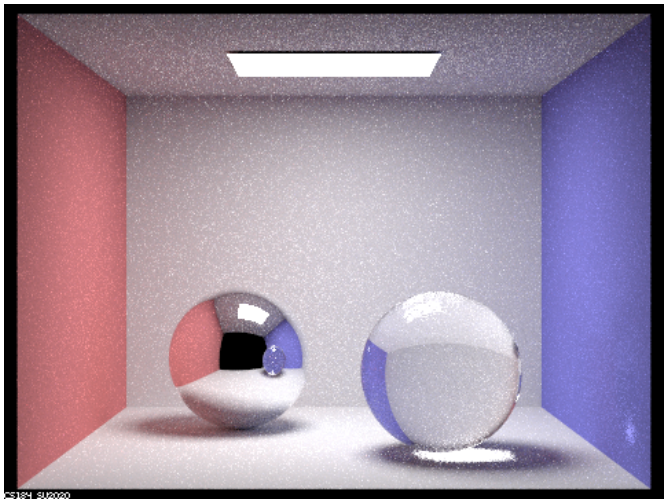
64 sapmle per pixel, 4 sample per light, m = 3



64 sapmle per pixel, 4 sample per light, m = 4



64 sapmle per pixel, 4 sample per light, m = 5

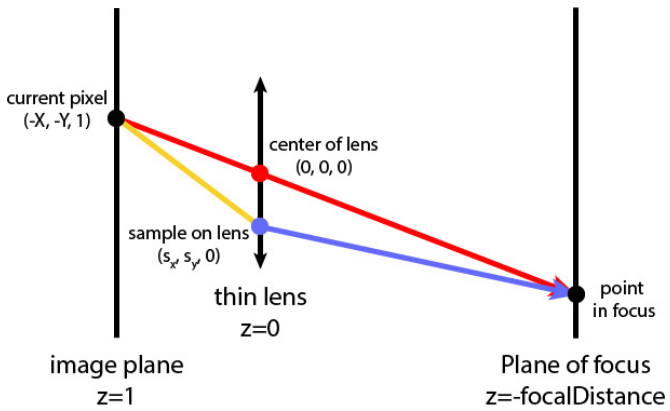


64 sapmle per pixel, 4 sample per light, m = 100

To conclude, when m = 0, we can only see light coming from light source. From m = 0 to m = 1, we can see that walls, floor, and top of the spheres are light up because of light directly being reflected off that surface. Since refraction is also consider as one bounce of light, so we cannot see refraction until m = 3. Also, the bright spot on the floor appear, and this is the result of light going in and out of the glass sphere then hit the floor. When m = 4, we can see the reflection of the from sphere onto the back sphere, and the bright spot on the wall shows up.

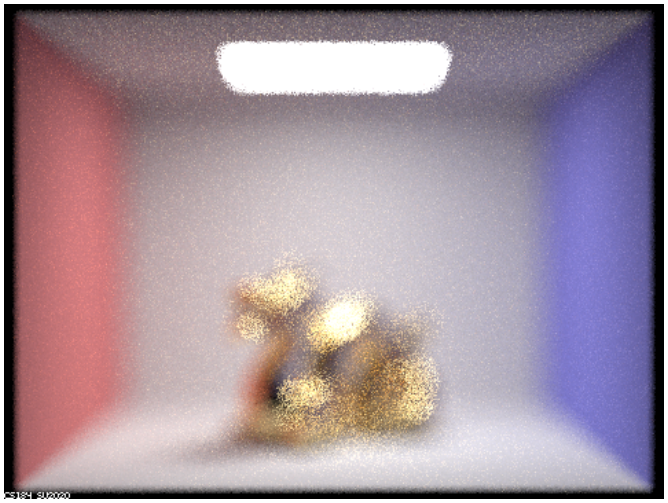
Part 4: Depth of Field

Before implementing this part of the project, we've been using the pinhole model which is a simple model without a lens. The light passes through a tiny aperture and form a opposite image on the other side. However, in real-world application, the light passes through a thing lens and and focus on the image plane. The difference is that every lens have their own focus distance, so not all the part of the object will be in focus. To implement the thin lens model, we start off by doing the exact same calculation as pinhole model. We have to calculate the X, Y value of the pixel on the image plane in the camera space. The top right corner of the image plane is (tan(radians(hFov) * 0.5), tan(radians(vFov) * 0.5), -1), and the bottom left is (-tan(radians(hFov) * 0.5), -tan(radians(vFov) * 0.5), -1). We can then interpolate the X, Y value using the x, y value passed into the function.

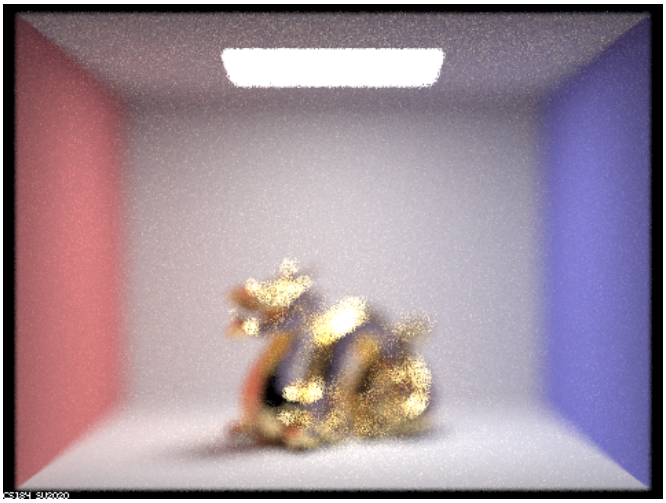


Visualization of thin lens model

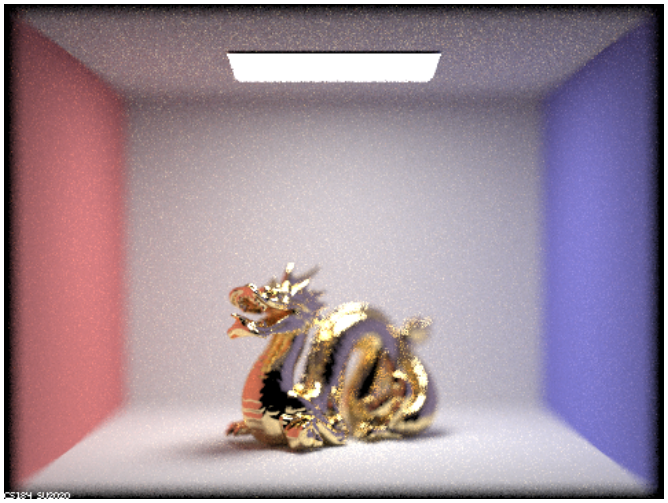
The position of the sample on lens can be sample using parameters: rndR and rndTheta. The next step we have to do is to calculate the position of focus point. We can get the value simply by -current_pixel * focalDistance. Last but not least, we subtact the postion of the sample on lens from focus and get the direction of the ray. Now, we have all the data we need to generate the ray for thin lens model, but we have to make sure to normalize the direction and transform it into the world spalce coordinate.



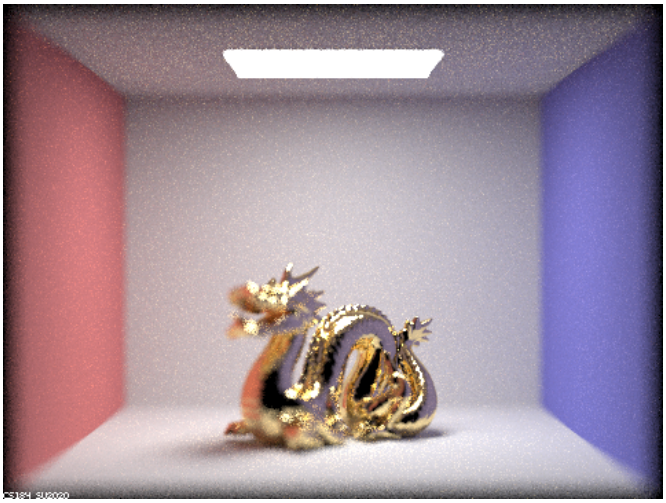
64 sapmle per pixel, 4 sample per light, m = 6, b = 0.18, d = 3.5



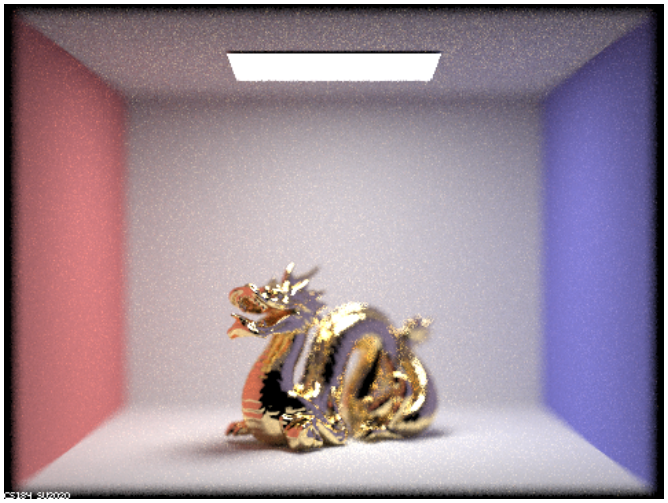
64 sapmle per pixel, 4 sample per light, m = 6, b = 0.18, d = 4



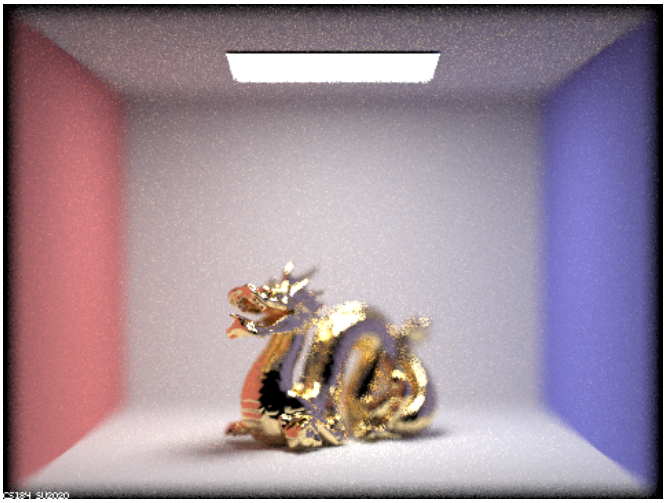
64 sapmle per pixel, 4 sample per light, m = 6, b = 0.18, d = 4.5



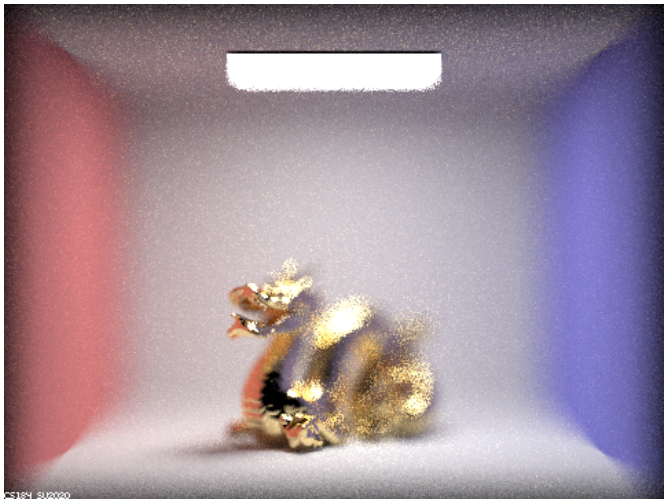
64 sapmle per pixel, 4 sample per light, m = 6, b = 0.18, d = 5



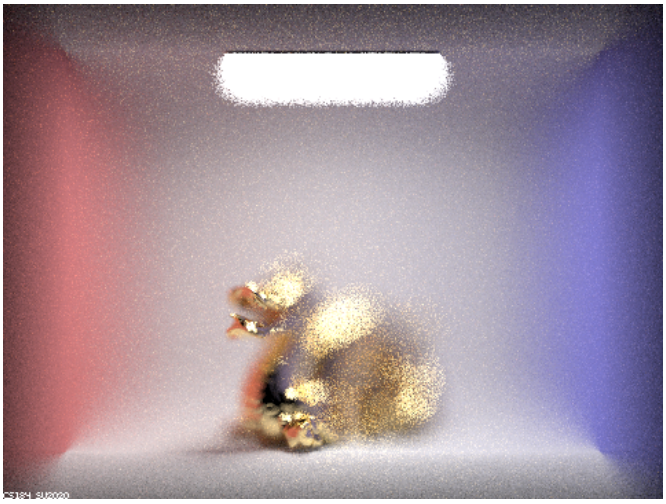
64 sapmle per pixel, 4 sample per light, m = 6, b = 0.15, d = 4.5



64 sapmle per pixel, 4 sample per light, m = 6, b = 0.2, d = 4.5



64 sapmle per pixel, 4 sample per light, m = 6, b = 0.5, d = 4.5



64 sapmle per pixel, 4 sample per light, m = 6, b = 1, d = 4.5