LAB 2: Full Path Tracer

PART 1:

Task 1: Improve Phong Material

For this task we have modified the *phong.cpp* to match the new formula given to us by the exercise. This time we respect the conservation of energy of the BRDF not only for diffuse materials but also for specular components. Then the new modified version of BRDF Phong follows:

$$f_r(\omega_i, \omega_o) = \frac{k_d}{\pi} + \frac{2\pi}{\alpha+1} k_s (\omega_r \cdot \omega_o)^\alpha$$

Task 2: Rendering Equation and Direct Illumination

4.2.1 Hemispherical Direct Illumination Shader

In this task, we implement the direct illumination part of the rendering equation, using a method called hemispherical sampling. This builds on what we did in the previous lab, where we handled transmissive and reflective materials. Now, we will calculate direct illumination by sampling directions within a hemisphere around each point where a ray hits a surface.

For this shader, we use 256 samples per pixel (spp). These samples are generated with the getSample function, which picks directions within the hemisphere formed by the normal of the surface we intersect. Each direction, labeled as wi, creates a new ray that starts from the intersection point and moves in the direction of wi.

We then trace this new ray to find where it hits next, which we call its_hemis. If its_hemis intersects an object that emits light, we take the light emitted at that point. Using the light from this new intersection, along with the BRDF of the original intersection and a weighting factor (based on the angle with the surface normal), we calculate each sample's effect on the final color. By averaging these contributions with Monte Carlo integration, we get a good estimation of the direct illumination in the scene.

This approach captures realistic lighting effects, like soft shadows, and the results can be seen in Figure 1 with 256 samples per pixel.

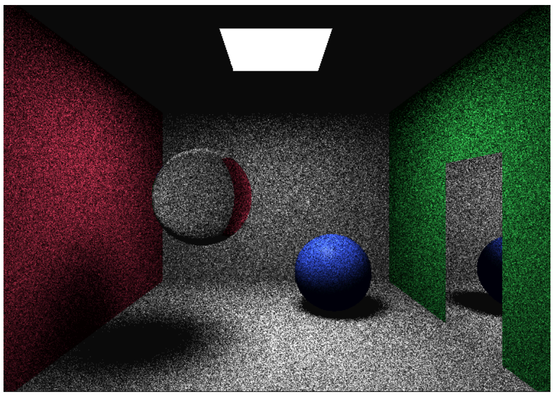

***Figure 1:** Hemispherical Sampling (256spp) with an Area light source*

4.2.2 Area Direct Illumination Shader

In this exercise, we take advantage of the area light source on the ceiling without using hemispherical samples. Instead, we loop through all the light sources in the scene (only one in this case), calculate each light's area, and then sample random points on the light's surface.

We use a second loop for the number of samples (in this case, 256 samples per pixel). We pick a random point on the light using sampleLightPosition and calculate the vector wi from the intersection point to this sampled point. We then compute the distance between these points.

As in the previous exercise, we create a new ray from the intersection point and check if there is any object blocking the light, setting a visibility flag. If there’s nothing in the way, we retrieve the light’s intensity (Le) and calculate the value G, which accounts for distance and angles between the light and the surface.

Finally, we use Monte Carlo integration to combine the reflectance and the light’s intensity for direct illumination. This gives us the final color result with 256 samples per pixel.

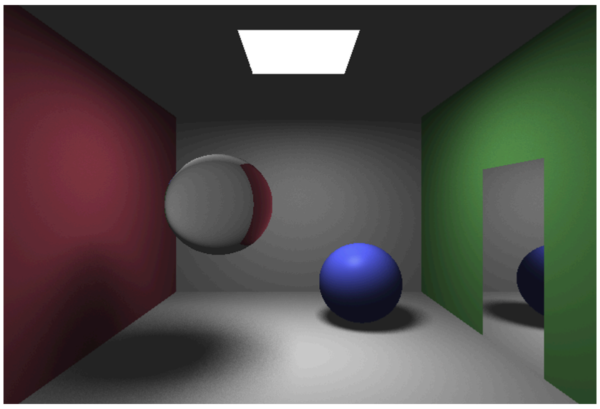

***Figure 2:** Area Sampling (256spp) with an Area light source*

PART 2:

Task 3: Adding Indirect Illumination

4.3.1 Pure Path Tracing

In this task, we add indirect illumination using a method called Pure Path Tracing, which calculates how light bounces around the scene. We use a similar structure to the hemispherical sample, but now we define a new function to compute radiance, with a set maximum depth for the light bounces.

First, we check if the current depth is less than the maximum depth. If so, we then check the material type at the intersection point. If it’s emissive, reflective, or transmissive, we handle it the same way as in computeColor.

For diffuse or glossy surfaces, we sample a random direction from the hemisphere above the surface using hemispherical sampling. We then trace a new ray in this sampled direction and compute the incoming radiance from further bounces. This incoming radiance is combined with the surface's BRDF and the cosine of the angle between the surface's normal and incoming direction, then averaged using Monte Carlo integration.

Each sample's contribution is averaged over 256 samples to reduce noise and achieve a smooth, realistic image.

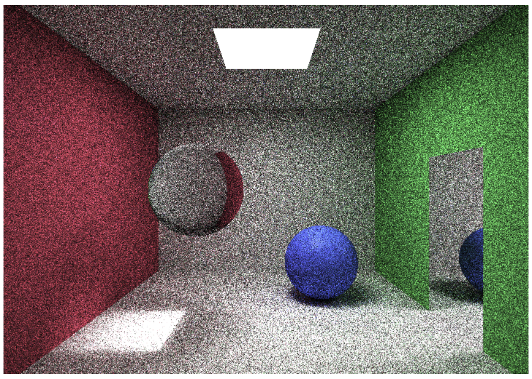

Figure 3: *Pure Path Tracing (256spp) with an area light source*

4.3.2 Next Event Estimation (Only Phong Materials)

In this task, we use Next Event Estimation to improve lighting accuracy for scenes with Phong materials. This approach builds upon the area light shader from exercise 4.2.2 but incorporates the ComputeRadiance() function to calculate light contribution more precisely.

With this method, we sample points directly on the area light source, as in direct lighting. However, now we adjust the PDF (probability density function) to account for the light's area, which gives a more accurate representation of how much light reaches each point.

Finally, we combine the ComputeRadiance() function with the area light calculations to handle Phong materials specifically, making the scene rendering more realistic by capturing direct lighting contributions from area lights accurately. This approach results in more precise shading effects and smoother lighting.

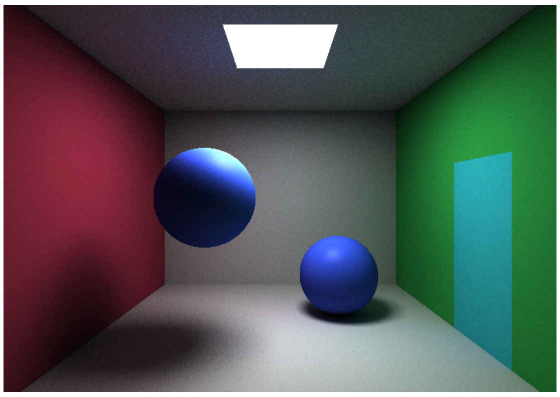

Figure 4: *NEE with only Phong materials (256 spp)*