Section One:

The goal of this project was to build a path tracer that could simulate different materials. We did this by setting up Monte Carlo sampling to fire rays, using a BRDF that we modeled after the Cook-Torrance model, and implementing area lights.

Monte Carlo:

For setting up our path tracer, we used Monte Carlo Sampling to randomly shoot bouncing rays through our scene. As touched upon in class we use Russian Roulette Termination for speed, where there is a termination probability *q* and a constant *c*. If you terminate this ray, return *c* otherwise, at the end, divide this ray's contribution by the survival probability as shown in the equation below [2].

$$F' = \begin{cases} \frac{F - qc}{1 - q} & \xi > q \\ c & \text{otherwise.} \end{cases}$$

To actually fire the bounces, we randomly fire on the hemisphere of the object as discussed in class by generating spherical coordinates and then converting that into world coordinates.

Cook-Torrance:

To get the actual color of the ray, we use the Cook-Torrance BRDF. This microfacet BRDF actually allows the artist to define a roughness and metallic parameter to their materials, where roughness is how many tiny microscopic gouges are on the surface. For our diffuse term, we simply used lambertian BRDF which is just the base color of the object divided by pi. This is basically just creating a uniform spread of the color in all directions (which is what a lambertian diffuse material is) [2].

For the specular term, the Cook-Torrance looks like the following: $\frac{D * G * F}{4 * (\vec{n} \cdot \vec{l}) * (\vec{n} \cdot \vec{v})}$ *D* represents the normal distribution function which is determined by how rough the surface is. *D* essentially describes the random microscopic gouges in the surface. *G* is the geometry function which, using the view and the light direction, determines a masking/shadowing effect for how the tiny facets block one another. *F* takes into account the angle of incidence, and reflects the amount of light based off of that.

For specifics on each distribution, we used a Schlick approximation for the Fresnel term, the GGX distribution for the NDF term, and the GGX Geometry function which we found to be the modern approximations [1].

Area Lights:

Area lights are a rendering technique used in computer graphics to realistically simulate the behavior of light in a scene. Unlike point or directional lights, area lights represent light sources with finite surface area, such as rectangular panels or spherical bulbs. In path tracing, area lights emit light uniformly from their surface, which means they can create soft shadows and more realistic lighting effects compared to point lights. When a ray of light intersects with an area light, the renderer calculates how much light is emitted from each point on the light's surface and how that light interacts with the surfaces in the scene. In theory, the renderer does this by integrating along the surface of the area light. However since that's not very feasible in a simple path tracer, we can also approximate this calculation by using uniform random sampling.

Section 2:

Path Tracing Implementation:

In our implementation, we essentially kept the backbone of the Whitted Ray Tracer intact and replaced traceRay with a new function called tracePath. In the beginning of this function, we take care of the Russian Roulette termination in the beginning then fire off the bounce ray using the Monte Carlo sampling method. After receiving the color for that ray, we divide by the PDF function used to sample the ray.

Next, we call a new shade function we created called shadeBRDF in the material class. This function takes in the indirect light direction, the view vector, and the indirect light contribution. This function first calculates the parts of the Cook-Torrance BRDF that are independent of the light source, such as the alpha value, the F0 term for the Fresnel reflectance, etc. With the Fresnel reflectance specifically, if the material is metallic, we mix the F0 calculation with the base color of the material based on how metallic it is. Next, we go through all of the direct light sources and calculate their diffuse contribution and specular contribution using the equations described in the last section and add them all up. Finally, we also do this for indirect light and return it.

After returning the color of this ray, we scale it by 0.9 (the survival probability) to finish the ray's contribution for Russian Roulette termination. When tracePixel returns, we average out the *N* rays fired through that pixel to get the pixel's contribution.

A known shortcoming we have is that we fire a reflection ray with the probability being based on how smooth the object is. This is to make the metallic object work. We tried to implement Importance Sampling but could not get the distribution to work for us.

Area Light Implementation:

In our path tracer we implemented rectangular area lights. A rectangular area light is defined by the corner of the area light, a width and height basis, and the corresponding length of the width and height basis (to form the finite rectangle). Whenever we want to randomly sample from the area light, we take a random u value fitting in the width, a random v value fitting in the height, and interpolate the width and height basis by u and v while adding the corner to obtain the point in the rectangle. So, whenever we need to calculate the illumination from an area light, we shoot N shadow arrays onto random sampled points into the area light. Once we calculate all of those illuminations, we average them all up to obtain the final illumination for that point. It's important to note that after the random sampling, we treat the resulting point as a point light, meaning it has its own shadow attenuation and distance attenuation.

References

1. YouTube. (2022, July 29). MICROFACET BRDF: Theory and implementation of basic PBR materials [shaders monthly #9]. YouTube. https://www.youtube.com/watch?v=gya7x9H3mV0
2. Pharr Matt, Jakob Wenzel, Humphreys Greg, (2023). *Physical Based Rendering: From Theory to Implementation* (4th ed., Sections 2.1-2.2, 9.2)