PART1 REFLECTION

In this step, we're implementing reflection using recursive ray tracing. This technique allows rays to "bounce" off reflective surfaces, creating realistic mirror effects. By calculating the direction of the reflected ray and tracing it recursively, we can simulate how light interacts with reflective materials, adding depth and realism to the scene. Here's how to set it up:

Reflection Direction Calculation: D_reflect = ray_dir - 2 * np.dot(ray_dir, hit_norm) * hit_norm

- This formula calculates the direction of the reflection ray based on the incoming ray (ray_dir) and the surface normal at the intersection (hit_norm). It uses the reflection formula: Dreflect=D−2(D·N) ·N

Recursive Ray Tracing for Reflected Ray: reflect_color = RT_trace_ray(scene, hit_loc + eps * D_reflect, D_reflect, lights, depth - 1)

- This line calls RT_trace_ray recursively with the new reflected ray (D_reflect), reducing the depth by 1. The recursion depth controls how many reflections are calculated, preventing infinite recursion.

Combine Reflection with Current Color: color += reflectivity * reflect_color

- Adds the reflected color (L_reflect) to the original color, scaled by the reflectivity factor (k_r or reflectivity), simulating how reflective the material is.

- After rendering the scene, we obtained Figure 1.

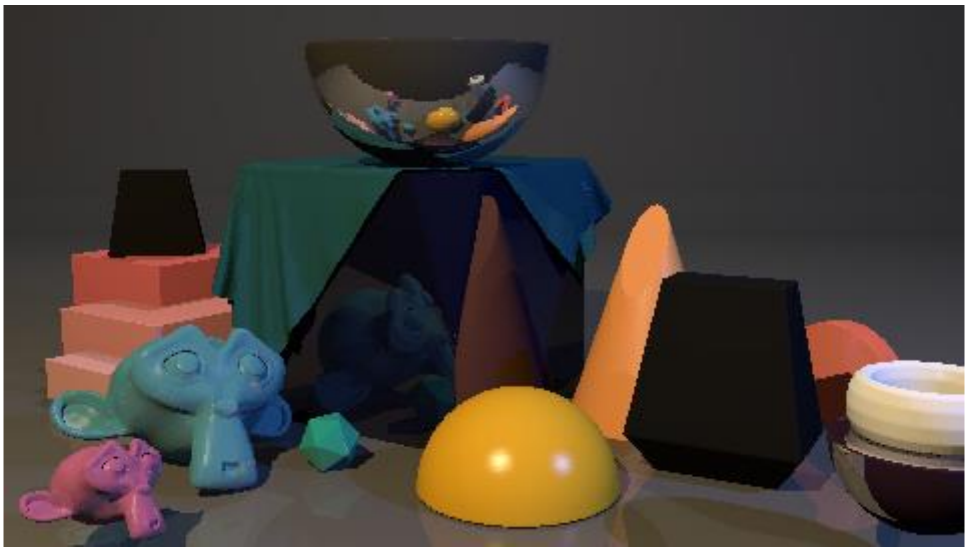

Figure 1.

PART 2 FRESNEL

To implement the Fresnel effect, we'll adjust the reflectivity based on the viewing angle and material properties. This adjustment allows reflective surfaces to exhibit more realistic reflection behaviors, where reflectivity increases as the viewing angle becomes more grazing (using Schlick's approximation). Here's the process:

- **R_0 Calculation:** This represents the reflectivity at a perpendicular angle, based on the Index of Refraction (IOR). R_0 = ((1 - mat.ior) / (1 + mat.ior)) ** 2
- **Reflectivity Calculation:** Using Schlick's approximation, we adjust k_r based on the angle theta between the ray and the surface normal. As the angle increases, the reflectivity rises, simulating real-world behavior.

```
theta = abs(ray_dir.dot(hit_norm))

reflectivity = R_0 + (1 - R_0) * (1 - theta) ** 5
```

- After rendering the scene, we obtained Figure 2.

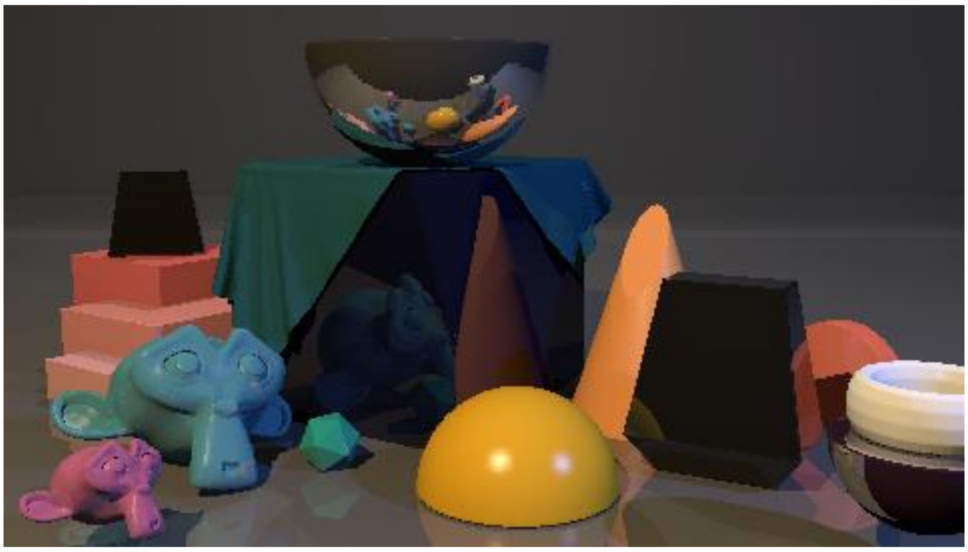

Figure 2.

PART 2 TRANSMISSION

To implement transmission, we simulate how light passes through transparent objects like glass or water. This involves calculating the transmitted ray using the refractive indices of the media and the material's transmission factor. Here's the step-by-step process:

The first step involves distinguishing between light transitioning into or out of an object:

- **Refractive Indices Assignment:** The refractive index of air is set to 1, while mat.ior is the object's refractive index. The direction of light refraction varies depending on whether it is entering or exiting the object, with n1 and n2 dynamically assigned based on ray_inside_object.

```
n1, n2 = (mat.ior, 1) if ray_inside_object else (1, mat.ior)
n_ratio = n1 / n2
```

- **Calculating Incident Angle:** The angle between the incident ray and the surface normal is given by cos_theta_i, which helps derive cos_theta_t2 and ensures the transmitted ray can be calculated only if the conditions for refraction are met.

```
cos_theta_i = -hit_norm.dot(ray_dir)

cos_theta_t2 = 1 - n_ratio ** 2 * (1 - cos_theta_i ** 2)
```

Using Snell's Law, the transmitted ray direction is calculated if the transmission condition is met:

- **Total Internal Reflection Check:** If cos_theta_t2 < 0, total internal reflection occurs, which prevents transmission. When cos_theta_t2 ≥ 0, the transmitted direction, D_transmit, is calculated using Snell's Law.
- **Direction of Transmitted Ray:** This computation adjusts the transmitted ray direction based on the angle and refractive index ratio, producing a vector that represents the refracted ray in the scene.

```
if cos_theta_t2 >= 0:
  D_transmit = n_ratio * ray_dir + (n_ratio * cos_theta_i - sqrt(cos_theta_t2)) * hit_norm
```

The transmitted ray is recursively traced to determine the color contribution from refraction:

- **Color Blending:** The transmitted ray's contribution to the final pixel color is scaled by (1 - k_r) * mat.transmission, ensuring that both reflection and transmission are visible in transparent materials. Here, k_r represents the reflectivity factor, balancing reflection and transmission.

```
transmit_color = RT_trace_ray(scene, hit_loc + eps * D_transmit, D_transmit, lights, depth - 1)

color += (1 - reflectivity) * mat.transmission * transmit_color
```

- After rendering the scene, we obtained Figure 3.

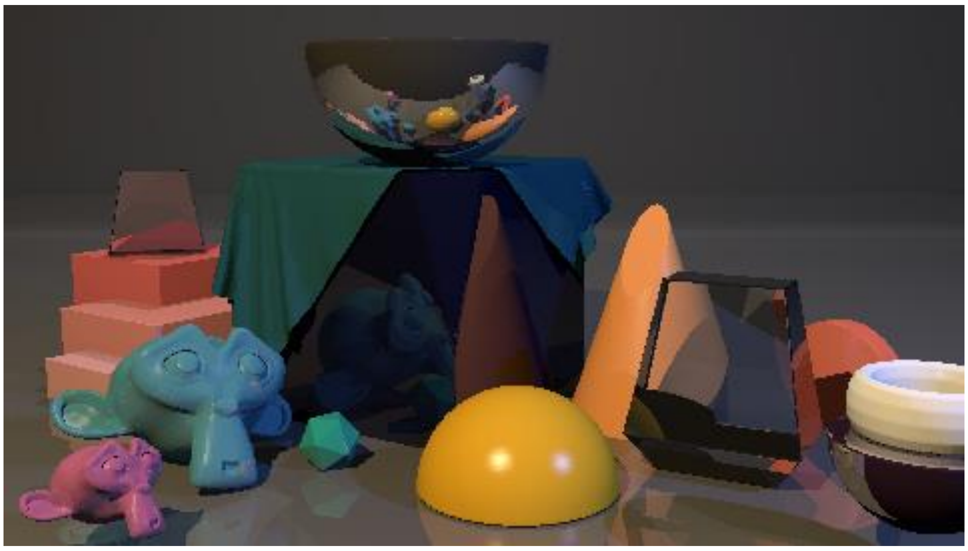
Figure 3.