Part1 Shadow-ray

For the first part of the project, we applied "The Ray Tracing Algorithm" to obtain "Shadows" for objects that will be displayed after the "Rendering" operation. The usage of the ray tracing algorithm for shadows can be divided into 2 parts: First, a "ray" (which is the continuation of rays from the camera that have intersected with the object) is shot from all of the pixels of the object in the scene to the light source second, if these rays have intersections with any of the object in the scene than these pixels are considered as "black".

Mentioned stages are coded as:

- Finding the direction vector from the intersected object to the light source call it light vector:

$$light\_vec = light.location - hit\_loc$$

- Normalizing that vector because the ray tracing algorithm in terms of "shadows" suggests that:

$$light\_dir = light\_vec.normalized()$$

- After finding the light vector it was crucial to prevent "Spurious Self-Occlusion" because in some rare cases, the ray can be re-intersected with the same object. To do this the light vector position must be shifted with respect to the "epsilon" value. So, we "shift the light vector from the hitting location". In the given code it has a value "1e-3". Obtained this functionality with the code below:

$$new\_orig = hit\_loc + eps * light\_dir$$

- Finally, to test if the ray shot to the light source intersected with any of the objects in the scene the "ray_cast()" function has been used. The code can be observed below:

$$has\_light\_hit, \_, \_, \_, \_ = ray\_cast(scene, new\_orig, light\_dir)$$

- After rendering the scene, we obtained Figure 1.

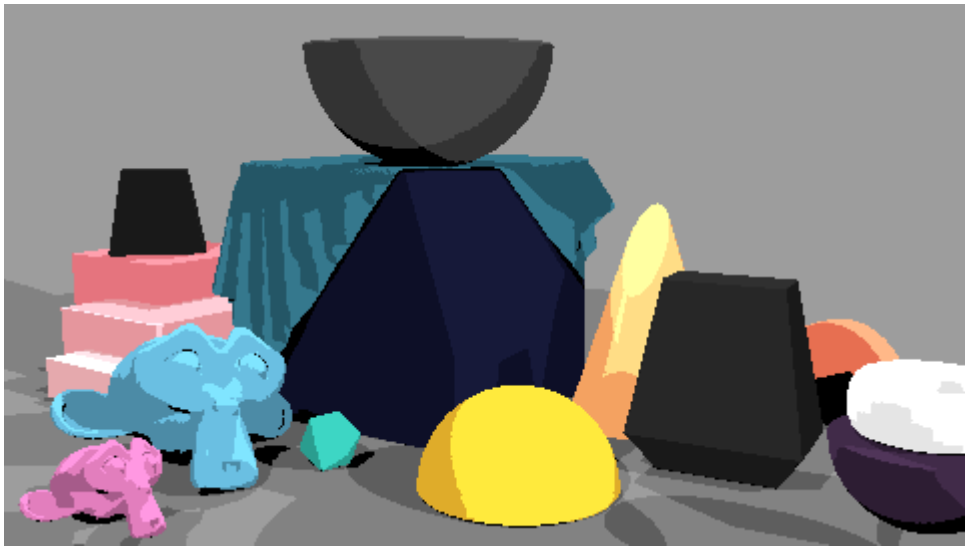

Figure 1.

Part2 Blinn Phong Shading

In this implementation, the goal is to calculate both diffuse and specular lighting at a point of intersection, determining whether each light source contributes to the final pixel color based on its position and the object's surface properties.

Diffuse Component Calculation:

The diffuse component simulates light scattering equally across a rough surface:

```
I_light = light_color / light_vec.length_squared  # Light attenuation

diffuse_component = np.array(diffuse_color) * I_light * np.dot(light_dir, hit_norm)

color += diffuse_component  # Adds the diffuse effect to final color
```

- **Light Attenuation:** I_light is scaled inversely by distance, capturing the natural falloff.
- **Diffuse Calculation:** The diffuse intensity is proportional to np.dot(light_dir, hit_norm), where hit_norm is the surface normal. This dot product determines how much light falls on the surface directly, with perpendicular angles producing maximum intensity.

Specular Component Calculation:

```
half_vector = (light_dir + (-ray_dir)) / np.linalg.norm(light_dir + (-ray_dir))

specular_component = np.array(specular_color) * I_light * (np.dot(hit_norm, half_vector) ** power)

color += specular_component
```

- **Half Vector Calculation:** The half_vector represents the direction halfway between the light direction and viewer direction. This midpoint allows for smoother, more accurate highlights.
- **Highlight Intensity:** The highlight's sharpness depends on specular_hardness, with higher values producing sharper highlights. The final specular component is scaled by I_light, specular_color, and (np.dot(hit_norm, half_vector) ** power).

Explanation of the Blinn-Phong Effect:

The Blinn-Phong model is ideal for simulating shiny surfaces in a realistic manner. The combination of diffuse and specular components allows for varied textures: matte surfaces can have high diffuse and low specular values, while shiny surfaces have lower diffuse and higher specular values. By capturing these surface variations, Blinn-Phong shading adds depth and realism to objects under multiple light sources.

- After rendering the scene, we obtained Figure 2(diffuse shading).
- After rendering the scene, we obtained Figure 3(specular shading).

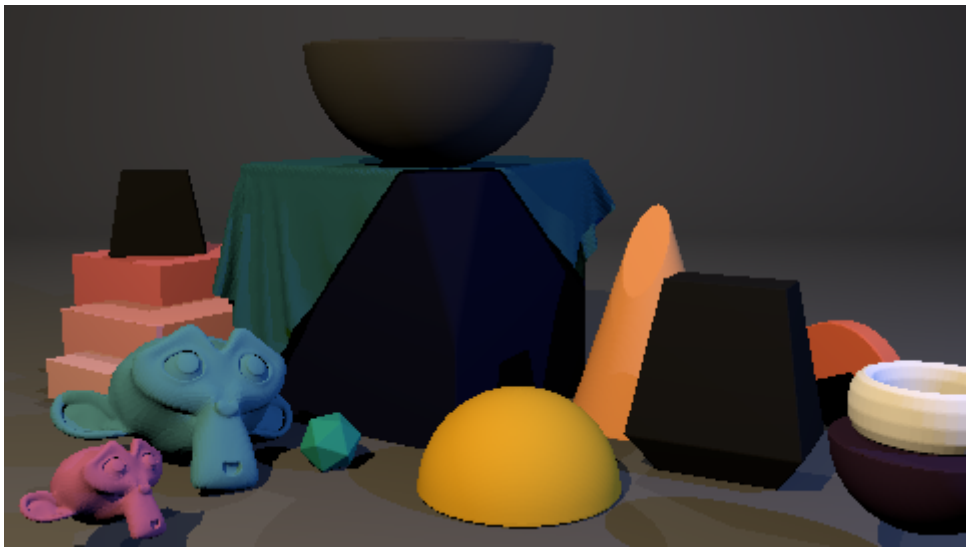

Figure 2(diffuse shading).

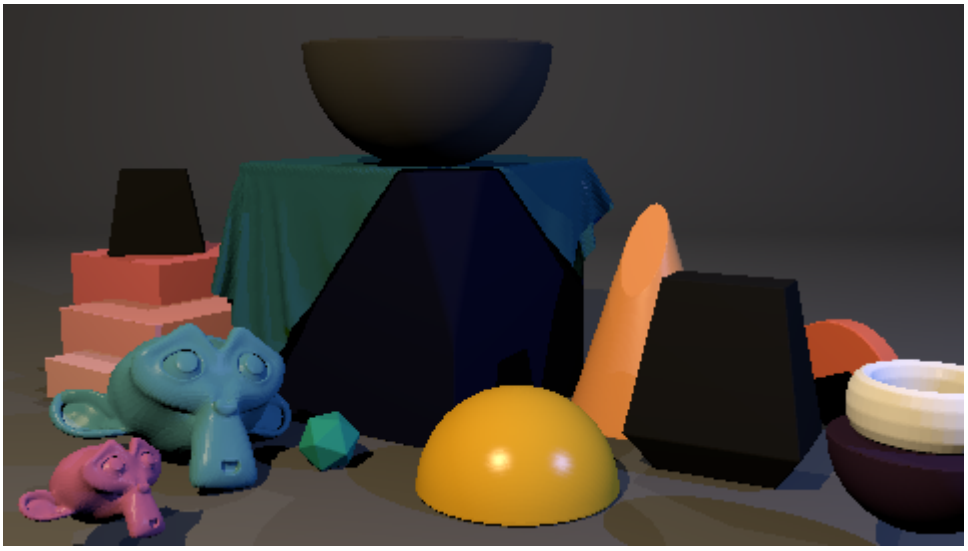

Figure 3(specular shading).

Part3 Ambient Light

With respect to models combined Lambertian and Blinn-Phong Shadings we expect to see black pixels if there is intersection between the shadow rays and any of the objects that stand middle of the ray between the object and the light source (meaningly shadow). But in real life there are indirect lighting that illuminates these shadows not completely but in recognizable extend. So, we might not be using these indirect lightings, but we add some constant to our models to make these shadows less black. This is called as “Ambient Lightning”.

This term is: $k_a\ I_a$ where k_a: surface ambient coefficient and I_a is ambient light intensity. So, we applied this formula with a code:

- Calculate Ambient lightning:

$$I\_ambient = diffuse\_color * Vector(ambient\_color)$$

- Add this calculated term to the global color variable:

$$color \mathrel{+}= I\_ambient$$

The rendered output of this complete model is Figure 4. As can be observed some of the shadows are less dense (for example shadows that monkey created).

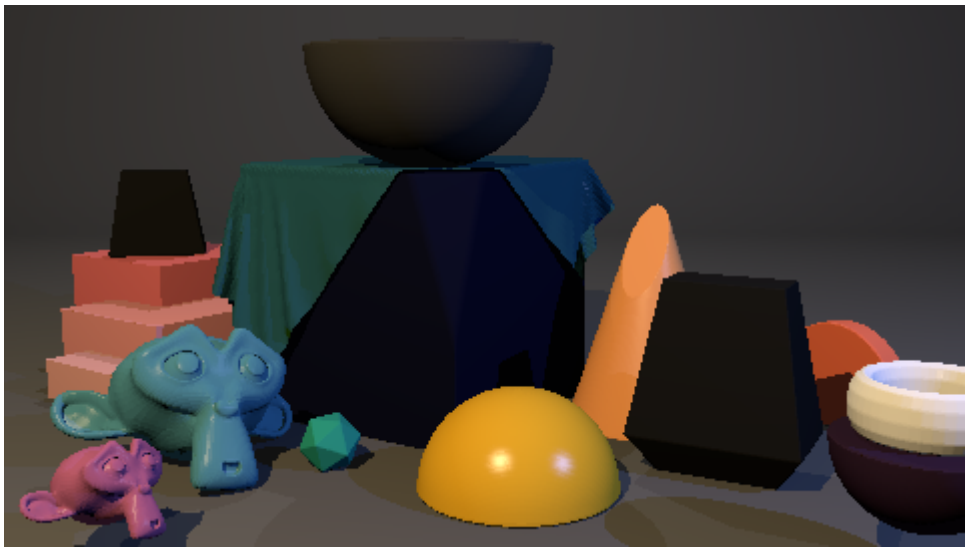

Figure 4.