Point Clouds & 3D Modeling
Practical Assignment

Rendering

The objective of this assignment is to write a Python program which renders a shaded image using a simple *normal* image. The following packages are recommended: PIL, math, numpy.

I. Image loading and display

For this assignment, we will make use of the file normal.png (see the course website). This file is an image of scanned point cloud, where each pixel stores a local estimate of the normal vector at the pixel location. Based on this normal vector, we will compute a per-pixel color response latter. We consider that the X and Y coordinates in image space correspond to the X and Y coordinate in 3D space, and that the camera is located along the Z axis.

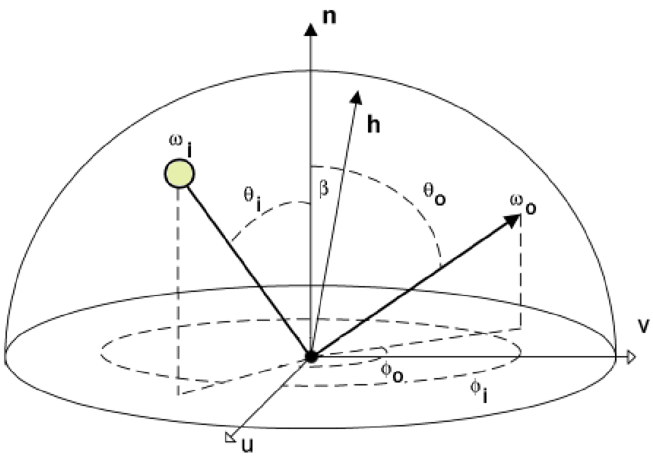
a. Load normal.png in an Image object (normalImage)
b. Display normalImage (using Python)
c. Store the image in a file called render.png

II. Diffuse shading

We recall the rendering equation:

$$L_o(\omega_o) = L_e(\omega_o) + \int_0^{2\pi} \int_0^{\pi} L_i(\omega_i) f(\omega_i, \omega_o) \cos\theta_i d\theta_i d\phi_i$$

This equation can be visually interpreted as follow:



Here:

- ω_i is the incoming light direction,
- ω_o is the outgoing direction e.g., the final sensor (camera) direction for simple direct lighting scenarios,
- n is the surface normal vector,
- L stands for the radiance measure,
- h is the *halfvector* i.e., the spherical average of ω_i and ω_o: $h = \frac{\omega_i + \omega_o}{\|\omega_i + \omega_o\|}$

For each pixel of the image, we now want to replace its value by the diffuse color response, using the Lambert BRDF:

$$f^d(\omega_i, \omega_o) = \frac{k^d \cdot albedo}{\pi}$$

a. Define a simple Material model (using a set of global variables or an object) in the form of an *albedo* RGB value and a *diffuse coefficient* k^d.

b. Define a lighting environment composed of a set of point light sources (again using a set of global variables or encapsulating the LightSource concept in an object), each point light source being defined by a 3D *position*, an RGB *color* value and an *intensity*. Start with a single point light source located at coordinates [0, 1, 1].

c. Implement a function def shade (normalImage) which renders an image using your Material and LightSource models, together with the scene's geometry defined solely by the per-pixel normal input normalImage.

III. Specular Materials

We now want to enrich our shading function with a more evolved reflection model, able to reproduce specular light reflections.

a. To do so, implement the Blinn-Phong BRDF:

$$f^s(\omega_i, \omega_o) = k^s(n.\omega_h)^S$$

With:

- k^s the *specular coefficient*,
- S the *shininess coefficient*.

Use both f^d and f^s by summing their contribution, for each light source.

b. Finally, we want to replace our specular BRDF by a *physically-based microfacet* model taking the general form a the Cook-Torrance BRDF:

$$f^s(\omega_i, \omega_o) = \frac{D(\omega_h)F(\omega_i, \omega_h)G(\omega_i, \omega_o, \omega_h)}{4(n \cdot \omega_i)(n \cdot \omega_o)}$$

This BRDF should be controlled by a *roughness* value α and a (theoretically binary) *metallic* property β characterizing the conductor/insulator nature of the material, together with a specular RGB color (derived from the Fresnel index). You will use the *GGX normal distribution function* as D, the *Schlick* approximation to the *Smith* model for G and spherical guassian variant of the *Schlick Fresnel* approximation for F. you can refer to the Unreal Engine Real Time Shading reference white paper by Karis (page 2 and 3) for details.