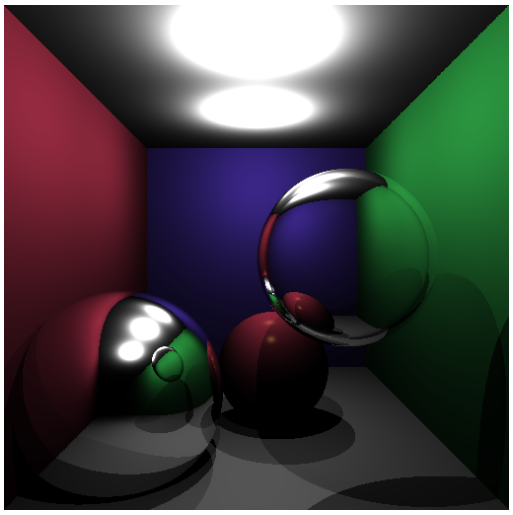
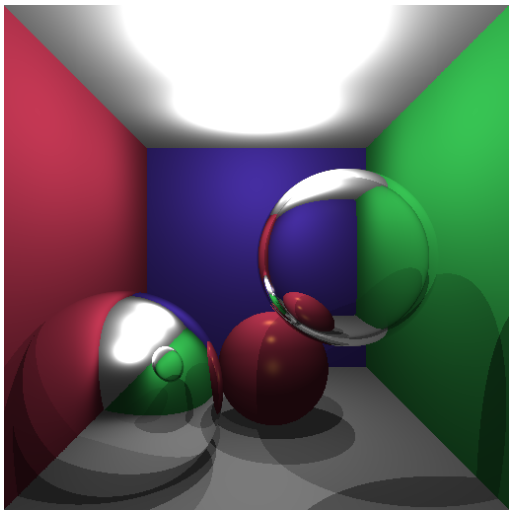
Imatge Sintètica - Assignment 5

Ricardo Marques

2017/2018



(a) Direct Illumination



(b) Global Illumination using an ambient term

Figure 1: Direct illumination VS Global illumination simulated using an ambient term.

1 Introduction

Assignment 4 consisted in enhancing the direct shader implemented in assignment 3 with new lighting effects: perfect specular reflections and transmissions. Moreover, new shapes such as infinite planes and triangles have been added to the ray tracer. In this assignment, we will focus on improving the light propagation simulation of our ray tracer by accounting for indirect illumination, i.e., by accounting for the light that arrives at a surface point after one or more bounces in other objects of the scene.

2 Simulating Indirect Illumination with an Ambient Term

2.1 Context

Implementing global illumination is not a trivial task. Therefore, we will start by implementing a new shader, called *GlobalShader*, which, instead of explicitly computing the indirect illumination at the shading point, uses an ambient term to produce a rough approximation.

The ambient term a_t is a color (*Vector3D*) which accounts for the small amount of light that is scattered about the entire scene. Using an ambient term implies assuming that the same small amount of indirect light a_t arrives at all points of the scene. Note that, in our case, we only have to compute the indirect illumination at the surface points which have a Phong material. This is because the ambient term aims at accounting for the indirect light arriving at the shading point from all directions. Consequently, it only makes sense to compute it for materials which are able to reflect light arriving from any direction (in our case, only the Phong material fills this requirement).

Adding an ambient term to the Phong reflection model is typically implemented by summing the product $k_d\, a_t$ to the direct illumination computed at the shading point. The light $L_o(\mathbf{p}, \omega_o)$ reflected at a point $\mathbf{p}$ in the outgoing direction $\boldsymbol{\omega}_o$ is then given by:

$$L_o(\mathbf{p}, \boldsymbol{\omega}_o) = L_o^{ind}(\mathbf{p}, \boldsymbol{\omega}_o) + L_o^{dir}(\mathbf{p}, \boldsymbol{\omega}_o) \tag{1}$$

where:

$$L_o^{ind}(\mathbf{p}, \boldsymbol{\omega}_o) = a_t\, k_d \tag{2}$$

$$L_o^{dir}(\mathbf{p}, \boldsymbol{\omega}_o) = \sum_{s=1}^{nL} L_i^s(\mathbf{p})\, r(\boldsymbol{\omega}_i^s, \boldsymbol{\omega}_o)\, V^s(\mathbf{p}) \tag{3}$$

An example of an image computed using this type of approximation is depicted in Fig. 1. Note that, as opposed to the image computed with direct illumination, all the surface of the roof appears now illuminated.

2.2 Your Task

Your task in this exercise is to implement a new shader, called GlobalShader which implements global illumination according to Eq. (1). To do so you should follow this list of steps:

- Based on the DirectShader, create a new shader called GlobalShader simply by copying the DirectShader implementation
- Add a new method called *getDiffuseCoefficient()* to the Phong material which returns the value of k_d required to evaluate Eq. (2). Such a method should be declared and implemented as a virtual method in the parent class *Material*, and reimplemented in the *Phong* class. The implementation of the method in the Material class should look as follows:

```
Vector3D Material::getDiffuseCoefficient() const
{
  std::cout << "Warning!"
            << "Calling \"Material::getDiffuseCoefficient()\""
            << "for a non-diffuse or non-glossy material"
            << std::endl;
  return Vector3D(-1);
}
```

- Change the function *Vector3D GlobalShader::computeColor()* so that it accounts for the ambient term according to Eq. (1). Note that the term L_o^{dir} is simply the direct illumination term that you already know how to compute since assignment 3. This means that all you have to do is to implement the computation of the L_o^{ind} term.

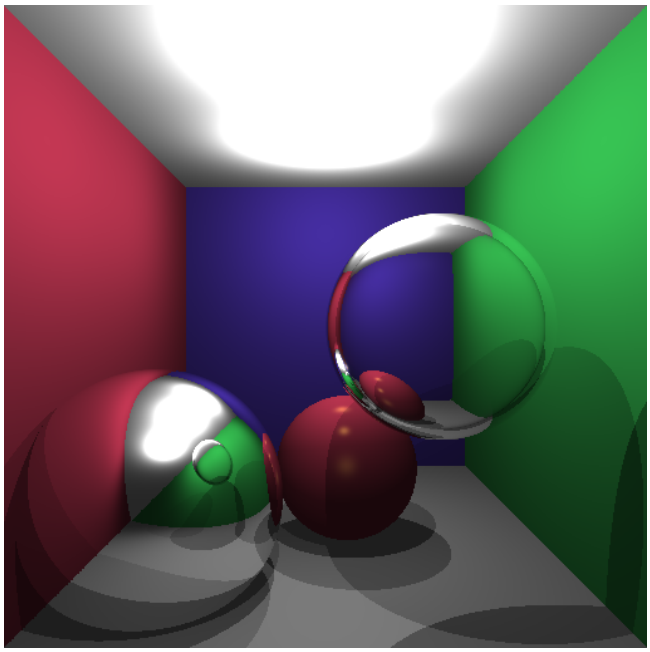
3 Explicit Computation of Indirect Illumination

3.1 Context

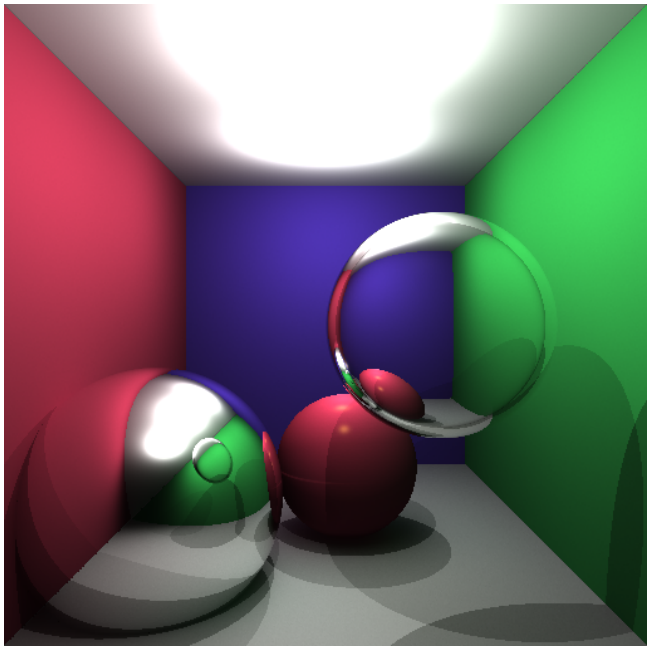
Using an ambient term to approximate the direct illumination is a very rough approximation which suffers fro two main drawbacks:

1. The indirect illumination component is assumed to be constant across all the scene. This is clearly not realistic.
2. Effects such as color bleeding (objects or surfaces are colored by reflection of colored light from nearby surfaces) is not simulated. For example, the left part of the top plan of the Cornell box should appear slightly red due to the light reflected by the left plan.

To cope with this problems we will implement an explicit computation of the indirect illumination component, which should allow you to produce images similar to Fig. 2(b).



(a) Global Illumination using an ambient term



(b) Explicit computation of Global Illumination

Figure 2: Global illumination using ambient term (a) and an explicit computation (b).

3.2 Your Task 1 (2-bounces Indirect Illumination)

The goal of this exercise is to implement an explicit computation of the indirect light arriving at a shading point. To do so, you need to make use of the *ray.depth*

attribute to control the number of bounces at Phong surfaces as follows:

- for camera rays, the attribute *ray.depth* must be initialized with value 0 (meaning that no intersection with Phong materials has been found so far)
- When an intersection with a perfect specular material is detected, the reflected/refracted ray has the same depth of the incident ray.
- when an intersection with a Phong material is detected:
  1. If $ray.depth == 0$ then we are in the first intersection with a Phong material detected so far. To compute that incident indirect illumination we will send $nSamples$ rays through the hemisphere centered in the normal at the surface. The indirect light component L_o^{ind} is then given by:

$$L_o^{ind}(\mathbf{p}, \boldsymbol{\omega}_o) = \frac{1}{2\pi n} \sum_{j=1}^{n} L_i(\mathbf{p}, \boldsymbol{\omega}_j)\, r(\boldsymbol{\omega}_j, \boldsymbol{\omega}_o) \tag{4}$$

     where $\boldsymbol{\omega}_i$ is a random direction on the hemisphere at $\mathbf{p}$ which can be obtained by calling the function *sampler.getSample(normal)*. The secondary rays sent from the shading point must have depth given by:

$$secondaryRay.depth = ray.depth + 1$$

  2. If $ray.depth > 0$ then the indirect illumination is approximated using the ambient term as in Eq. (2).

3.3 Your Task 2 (n-bounces Indirect Illumination)

The goal of this exercise is to make the GlobalShader support n-bounces of light across the scene (as opposed to the 2 which it currently supports). To this end, you should compute the indirect illumination as follows:

- If $ray.depth == 0$ then send $nSamples$ rays through the hemisphere centered in the normal at the surface (similar to the previous exercise)
- If $ray.depth == maxDepth$ then the indirect illumination is approximated using the ambient term as in Eq. (2). (similar to the previous exercise)
- Otherwise $L_o^{ind}(\mathbf{p}, \boldsymbol{\omega}_o)$ is given by applying Eq. (4) but fixing the sampled directions to the normal at $\mathbf{p}$ (denoted by $\boldsymbol{\omega}_n$) and to the perfect reflection direction $\boldsymbol{\omega}_r$, yielding:

$$L_o^{ind}(\mathbf{p}, \boldsymbol{\omega}_o) = \frac{1}{4\pi} \left(L_i(\mathbf{p}, \boldsymbol{\omega}_n)\, r(\boldsymbol{\omega}_n, \boldsymbol{\omega}_o) + L_i(\mathbf{p}, \boldsymbol{\omega}_r)\, r(\boldsymbol{\omega}_r, \boldsymbol{\omega}_o) \right)$$