Introduction to Raytracing
Nicolas Bonneel

Lecture # 3

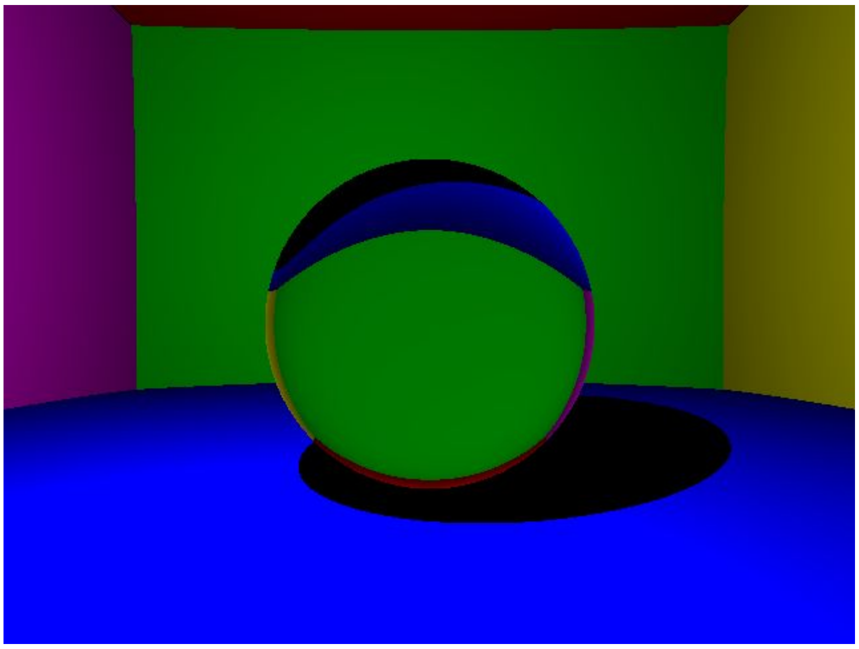
We will see how to manage the indirect lighting, and possibly meshes. But first, a note on the camera sensor and displays.

Gamma Correction

The perceived screen brightness does not vary linearly with the values we encode in the images : a pixel intensity of 127 does not seem halfway between pure black (0) and white (255). In fact, displays have a gamma factor : a power function like   brigthness = intensity ^ gamma, where gamma is often taken as 2.2.

To compensate for this transformation, we will encode images by applying the function x ^ (1./2.2) to each of the red, green and blue components.

Following is the same image as before, with gamma correction.



Indirect lighting

Handling indirect lighting is very similar to handling specular surfaces. The idea is, again, to recursively bounce light rays across the scene. But while mirror surfaces reflect only in the "mirror" direction, diffuse surfaces reflect rays randomly.

But first, a note on BRDFs, the rendering equation and Monte Carlo integration.

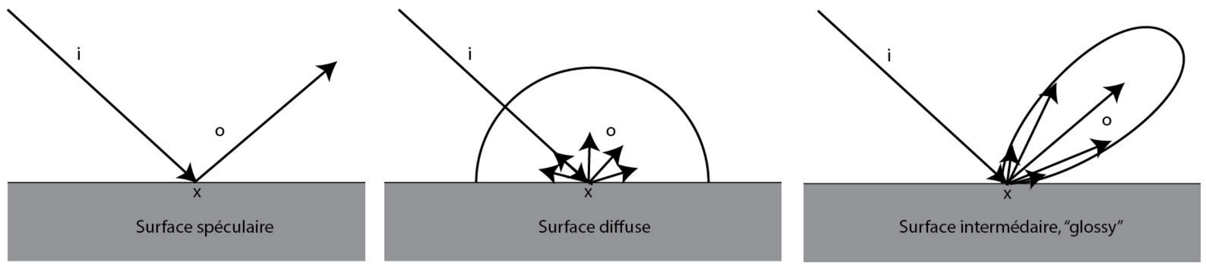
BRDFs

BRDFs or "Bidirectional Reflectance Distribution Functions", are functions characterizing materials. They describe the probability (up to a cosine factor and absorption) for a photon arriving at the surface of an object with an incident direction $\vec{i}$ to be reflect off in the direction $\vec{o}$. This probability is denoted $f(\vec{i},\ \vec{o})$ (by convention, we assume the incident vector as a vector pointing outwards the surface, i.e., the vector $-\vec{i}$ compared to previous figures).

In previous lectures, we saw two BRDFs characterizing two different materials:

- Specular BRDF. This gives a probability of 1 for a photon to be reflected in the "mirror" direction and a probability of 0 for all other directions. Its BRDF is given by $f(\vec{i},\ \vec{o}) = \frac{1}{\cos\theta_i}\,\delta(2 < \vec{i}, \vec{n} > \vec{n} - \vec{i} - \vec{o})$ where delta is a Dirac distribution, and the cosine factor will become clearer later.
- Diffuse BRDF. This gives a uniform probability for an incident ray to be reflected in any direction (on a full hemisphere). Its BRDF is therefore given by $f(\vec{i},\ \vec{o}) = \frac{constant}{\pi}$ (which is constant, and the pi factor is derived from the condition that $\int f(\vec{i},\ \vec{o}) \cos(\theta)\, d\vec{i} \leq 1$ where the integral runs over a hemisphere and “constant” is between 0 and 1). If the material absorbs some light, this constant is smaller than 1. It can also be different for each color (as you had previously programmed).

But there are many BRDFs in-between these two extreme cases: for example, similar to a Gaussian centered around the mirror direction. We will not see them in this course, but the following figure illustrates the concept. In general, for a function $f(\vec{i},\ \vec{o})$ to be a BRDF, it needs to respect the Helmholtz principle of reciprocity $f(\vec{i},\ \vec{o}) = f(\vec{o},\ \vec{i})$ and energy conservation $\int f(\vec{i},\ \vec{o})\ \cos\theta_i\ d\vec{i}\ <\ 1$ (see below). We also saw the case of transparent materials: these are not strictly speaking BRDFs ("R" comes from reflection and not transmission), but it is a very similar concept.

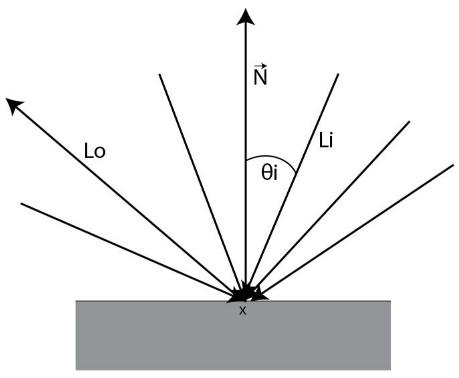


The rendering equation

Ultimately, it is the equation that gives the color of a pixel, but is expressed locally fairly easily.

$$L_o(x, \vec{o}) = E(x, \vec{o}) + \int f(\vec{i}, \vec{o})\, L_i(x, \vec{i})\, \cos\theta_i \, d\vec{i}$$

Here, $E(x, \vec{o})$ is the emissivity of the surface (we had considered only point light sources), $f(\vec{i}, \vec{o})$ is the BRDF, $L_i(x, \vec{i})$ the light intensity arriving at the intersection point x, $L_o(x, \vec{o})$ the intensity of the outgoing light, and $\cos\theta_i$ is the cosine of the angle between the incident ray $\vec{i}$ and the surface normal.



The factor $\cos\theta_i$ is because the surface will locally collect less light from a light ray at grazing angle than from a ray closer to normal incidence.

This equation simply says that the light outgoing in a particular direction at a point is the sum of all incident light (which includes indirect light sources such as walls) multiplied by the BRDF (and cosine).
In fact, you have already solved this equation for specular surfaces: remember that the BRDF of a mirror surface is $f(\vec{i},\ \vec{o})\ =\ \frac{1}{\cos\theta_i}\ \delta(2<\vec{i},\vec{n}>\vec{n}-\vec{i}-\vec{o})$. In the above integral, the cosine factor vanishes, and, assuming a non-emissive area (E = 0), the equation is:

$$L_o(x,\ \vec{o})\ =\ L_i(x,\ 2<\vec{o},\vec{n}>\vec{n}-\vec{o})$$

i.e., the sum is done only on a single incident direction: the one that corresponds to the reflection of the outgoing ray around the normal, and the result just corresponds to getting the reflected color by sending a new ray.

The rendering equation allows us to understand the condition $\int f(\vec{i},\ \vec{o})\ \cos\theta_i\ d\vec{i}\ <\ 1$ for a function to be a BRDF: this just says that a surface is not supposed to return more light than that the total light it receives!

The Monte Carlo method

This is a stochastic method for calculating integrals. We can show in general that computing an integral $\int f(x)\ dx$ can be done by sampling f randomly.

Let $x_0,\ x_1,\ ...,\ x_n$ be random numbers following a probability distribution $p(x)$. One can show that

$$\int f(x)\ dx\ =\ \frac{1}{n}\sum_{i=0}^{n} f(x_i)/p(x_i)\ +\ O(1/\sqrt{n})$$

In practice, the (proportionally) closer p is to f, the more accurate the method will be.

To compute the integral of the rendering equation, we will generate random directions following a probability distribution as close as possible to the term $f(\vec{i},\ \vec{o})\ \cos\theta_i$ up to a normalization factor (other strategies are possible!). The integral thus will be approximated by:

$$\int f(\vec{i},\ \vec{o})\ L_i(x,\ \vec{i})\ \cos\theta_i\ d\vec{i}\ \approx\ \frac{1}{n}\sum_{i=0}^{n}\ f(\vec{i}_i,\ \vec{o})\ L_i(x,\ \vec{i}_i)\ \cos\theta_i\ /\ p(\vec{i}_i)$$

In general, for any BRDF, generating a random direction with probability "similar to the BRDF" can be very difficult. Fortunately, for diffuse materials, it suffices to sample the factor $\cos\theta_i$ (since the BRDF itself is constant), and a closed form is known:

For diffuse surfaces, sampling a vector on the sphere proportionally to $\cos\theta_i$ corresponds to generating x, y, z coordinates by:

$$\begin{aligned} x &= \cos(2\pi r_1)\sqrt{1-r_2} \\ y &= \sin(2\pi r_1)\sqrt{1-r_2} \\ z &= \sqrt{r_2} \end{aligned}$$

where r_1 and r_2 are uniform random numbers between 0 and 1. You will obviously need to orient the ray in the frame oriented such that z is aligned with the normal at the point of intersection. We construct a frame by generating 2 vectors perpendicular to the normal and the normal itself, and apply a change of frame formula.
The probability $p(\vec{i})$ is written $p(\vec{i}) = \cos\theta_i / \pi$ where the pi factor allows this probability law to integrate to 1 over the hemisphere, and θ_i is the angle between $\vec{i}$ and the normal. Since we managed to exactly sample the BRDF not just a vague approximation, the full rendering integral for diffuse surfaces sampled with the above method is then:

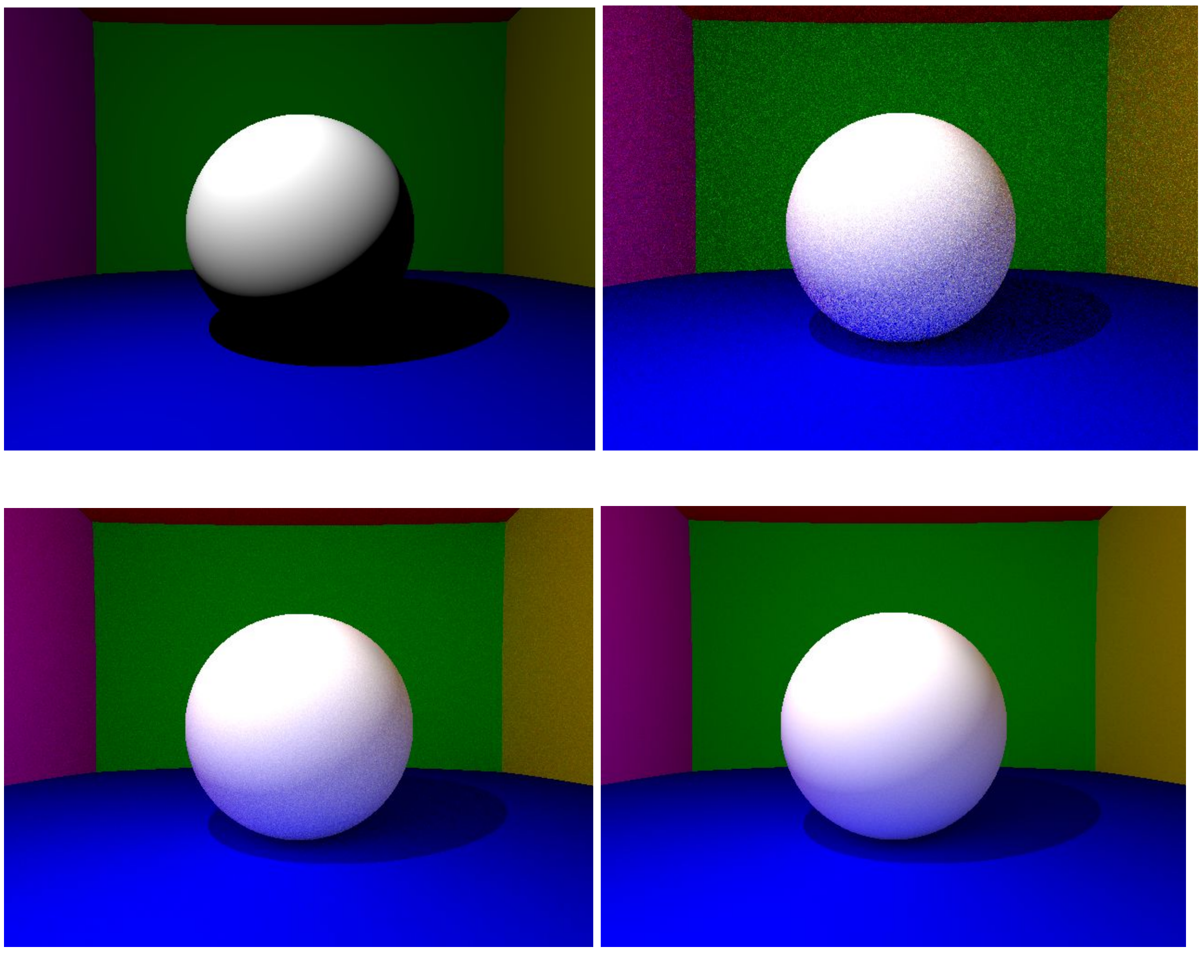
$$\int f(\vec{i}, \vec{o})\, L_i(x, \vec{i})\, \cos\theta_i \, d\vec{i} \approx \frac{1}{n}\sum_{i=0}^{n} \rho_d L_i(x, \vec{i_i})$$

where ρ_d is the diffuse coefficient between 0 and 1.
The other terms cancelling each other (also note the factor pi disappearing; see Note 2 below). One then use the same strategy as for specular surfaces: recursively launch rays to query the light intensities, but by drawing rays randomly instead of deterministically.

In general, many formulas to sample particular BRDFs are known, and are compiled in the Global Illumination Compendium by Philip Dutré.

This method however converges fairly slowly (in the square root of n), and expects to take a lot of samples to obtain a sharp image. Ideally, every intersection, we should draw n rays in different directions, then at each of the intersections of secondary rays we would draw n other rays etc. This would lead us to consider n^r rays, with r the number of bounces in the scene.
Fortunately another strategy is possible, and give the same result. Just send only one ray at each intersection randomly. This leads to a noisy image, obviously. But the whole process can be repeated to obtain a sharp image.
The strategy is therefore to draw n rays per pixel, instead of n rays per intersection. This corresponds to seeing the rendering integral as a large (even infinite) dimensional integral instead of recursively integrating integrals over the sphere.

In the above scene with a diffuse sphere (and a less bright illumination), with n = 0, 10, 100, 1000, we obtain:



<u>Note 1</u>: the computing time will increase significantly, especially if you have not paid attention to performance (eg, always prefer calculations with squared norms rather than norms that require costly calculations of roots) . Furthermore, the calculation of each of the pixels of the image is independent. It will be beneficial to use the parallelism of your machine to significantly reduce the computation time.
With OpenMP, simply add the following to your for loop that iterates through each line of the image:

```
#pragma omp parallel for schedule(dynamic,1)
```

(In CodeBlocks, use -lgomp; Visual Studio, enable it in the property "Language").

<u>Note 2</u>: the use of the rand () function of C ++ to generate random numbers is not recommended. It is preferred, if possible (i.e. if C ++ 11 is supported), the following:

```
#include <random>
std::default_random_engine engine;
std::uniform_real_distribution<double> distrib (0.1);
```