Lights and Materials

Now that we have the theoretical machinery of the BRDFs, we are now ready to begin discussing how to extend our ray tracers in order to perform shading. In simple terms, our previous algorithm looked something like this:

```
for (pixel : image)
{
    Ray r = computeRay();
    ShadeRec rc;
    for (object : geometries)
    {
        if (object.intersect(r, rc))
        {
            pixel = rc.colour;
        }
    }
}
```

What we are now going to add to this are the following two elements:

1. A `Material` which will act as the container for the BRDF(s) that are required to model that particular object and
2. The light sources.

In terms of the code, the body of the if-statement will be changed to invoke a shading function in the material. What this will return will be the final radiance, expressed as an RGB colour. The process will involve:

- Compute the ambient term from the corresponding BRDF.
- For every light in the scene, compute ω_i. If the direction of the light is not parallel to the scene (if $\cos\theta \neq 0$), then compute the BRDF at the point of intersection, using ω_o as the negative origin of the ray.
- Add up the contribution of all the lights to the ambient value we computed earlier.
- Return the final value.

Illumination and Reflection

The *illumination* in a scene is a general term for the light that comes from light sources and arrives at the surfaces of objects. We distinguish two types of illumination:

- *Direct Illumination:* light that arrives to the surface of the object by travelling directly from the light source.
- *Indirect Illumination:* light that arrives to the surface by having been reflect from at least one other surface.

Another naming convention is to call direct illumination *local illumination*, and indirect illumination *global illumination*.

The way light is reflected of objects can be quite complex depending on the properties of the surface. In computer graphics we tend to simplify this and express light reflection in terms of specular and diffuse reflection. At one extreme, there's *perfect specular reflection* in which all light is reflected in a single direction of mirror reflection. On the other extreme is perfect diffuse reflection, in which light is scattered equally in all directions. Halfway between these two is *glossy specular reflection* where light is concentrated around the direction of mirror reflection but it is also scattered.

Perfect specular reflection is used to render mirrors and transparent material such as glass, with indirect illumination. Mirrors can be modeled by tracing a single reflected ray at each hit point. In contrast, glossy specular reflection is used for both direct and indirect illumination. Perfect diffuse reflection is also used for direct and indirect illumination where it's a direct-reflection component of matte materials. Both glossy specular and perfect diffuse reflections require many ray sin order to produce accurate results.

Lights

We will be defining several types of lights here: ambient, point, and directional. In general, lights should be defined in terms of their power (radiant flux), but that's difficult to define for directional lights and point lights (which means these two are not physically accurate). What we will use instead is the colour of the light and a radiance scaling factor. This is appropriate since lights deliver radiance proportional to the product of the scaling factor and the colour along rays to the hit points. We can use the scaling factor as a convenience to control the brightness of the light.

Ambient Light

Due to the complexities of rendering indirect illumination, a common assumption is that this illumination is constant throughout the scene, where it's called *ambient illumination*. Although this does not even approximately correct for real scenes, it's better than nothing as it provides illumination to those parts of the scene that receive no direct illumination. Without it, parts of the scene would be rendered black, which isn't realistic as there is always some light coming from a different direction.

Ambient illumination corresponds to the first term of the rendering equation L_e. This is now a volume light that is providing all surfaces with the same amount of incident radiance. The ambient light provides part of the reflected radiance for all materials that contain a diffuse-reflection component. Images rendered with just ambient illumination show all surfaces with constant colours, but each material can have a different diffuse colour and reflect a different fraction of the ambient illumination.

The incident radiance $L_i(p, \omega_i)$ from the ambient light is

$$L_i = l_s c_i \tag{1}$$

which is independent of p and ω_i. The reflected ambient radiance is

$$L_o(p, \omega_o) = p_{hh}(p) \star l_s c_i$$

where $\star$ is the component-wise multiplication and ρ_{hh} is what we defined in equation 18 from the previous notes.

Directional Lights

Light rays from a directional light are parallel and they therefore come from a single direction. In addition, their incidence radiance doesn't vary with position. The Sun, or any other potent light source that is sufficiently far away are good approximations of a directional light source because their rays will hit the surface being essentially parallel. It is worth noting though that directional lights are mathematical abstractions as no ray of light is truly parallel.

A direction light is specified by the direction l from which the light is coming. This is *opposite* to the direction of the incoming light. Direction illumination at a point p from a directional light is represented by the integral:

$$L_o(p, \omega_o) = \int_{2\pi^+} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) \cos\theta_i d\omega_i \tag{2}$$

Where the incident radiance comes from a single direction $\omega_i = l$ and is independent of p. The reflected radiance therefore is:

$$L_o(p, \omega_o) = f_r(p, l, \omega_o) \star l_s c_l \cos\theta_l \tag{3}$$

For the sake of argument, lets discuss how we would design $L_i(p, \omega_i)$ by working backwards from L_o. What we want is a distribution that is zero for all incoming directions except l, where it has to be infinite. Therefore, L_i has to be represented using a *delta function* as follows:

$$L_i(p, \omega_i) = l_s c_l \delta(\cos\theta_i - \cos\theta_l)\delta(\phi_i - \phi_l) \tag{4}$$

Where θ, ϕ come from the expression of ω_i in spherical coordinates in relation to the normal at p. Here, eq. 4 demonstrates how illumination from a directional light is represented in the rendering equation.

As a last point, if we have n directional lights, we can compute the final reflected radiance by simply adding them all up.

Point Lights

Light from a point light radiates from a specified point in world coordinates. If the radiant intensity of the light is I, its flux Φ is obtained from integrating I over the surface of a unit sphere centred at the light:

$$\Phi = \int_{4\pi} l d\omega = 4\pi I \tag{5}$$

Now consider another sphere of radius r that surrounds the light. The flux through this sphere must be the same, but because the surface area is $4\pi r^2$, the radiance at every point on the surface is

$$E = \frac{\Phi}{4\pi r^2} = \frac{I}{r^2}$$

What this equation expresses is the fact that the radiance of a point light decreases as the inverse square of the distance from the light. In physics, this is called *inverse square law*; while in computer graphics it is called *distance attenuation.*

Point lights are also mathematical abstraction, since the light is generated from a point, which has no area. In real life, light sources have a finite area. That being said, point lights are still very useful for rendering scenes because of the simple expression for their irradiance.

If we let $I = l_s c_l$, then the reflected radiance at a surface point is:

$$L_o(p, \omega_o) = f_r(p, l(p), \omega_o) \star \frac{l_s c_l}{r^2} \cos\theta_l \tag{6}$$

Where $r = |p_i - p|$ is the distance between p and the location of the light. In contrast to a directional light, the light direction l does vary with p because it always points towards the light location. Similarly to eq. 4, the point light has the following equation:

$$L_i(p, \omega_i) = \frac{l_s c_l}{r^2} \delta(\cos\theta_i - \cos\theta_l)\delta(\phi_i - \phi_l)$$

Likewise, if we have n point lights, we can compute the final radiance by adding them up. In fact, if we have a mix of both, we can simply add up the contribution of each light to compute the final radiance. For the sake of simplicity we will ignore the distance attenuation for point lights, though keep in mind that it is possible to add it in (and in fact isn't that hard to implement).

Diffuse Shading

Basic Expressions

The reflected radiance from a material that implements diffuse reflection will generally have an ambient and a direct illumination component. The ambient component is given by the following:

$$L_o(p, \omega_o) = \rho_{hh} \star L_i(p, \omega_i) = k_a c_d \star (l_s c_l) \tag{7}$$

Since we are ignoring distance attenuation for point lights, we can therefore combine directional and point lights together in a single equation (again note that in code it doesn't matter nor does it change the computation). The reflected radiance is therefore the sum of the product of the BRDF for a Lambertian BRDF and the L_o of the lights themselves. What we end up with is the following:

$$L_o(p, \omega_o) = k_a c_d \star (l_s c_l) + \sum_{j=1}^{n} \frac{k_d c_d}{\pi} \star (l_{s,j} c_{l,j})(n \cdot l_j) \tag{8}$$

For the total reflected radiance at p, where $n \cdot l_j = \cos\theta_j$.

The question now becomes: why is $n \cdot l_j$ in that expression? The reason for it is that we need to have a way of shading surface of objects depending on their *orientation* in relation to the light. Specifically, we can express this as the angle between the normal of the surface and the light direction l. Points on the surface that satisfy the relationship $n \cdot l \geq 0$ will be shaded, while points that are less than 0 will not as they will not be visible to the light. Note that for cases where $n \cdot l > 0$, the light could still be blocked by other objects, but this is what shadows are for.

Out-of-Gamut Colours

Previously, we had defined our RGB colours as a triplet of floats in the range 0 to 1. The *gamut* of a display device is the set of all colours that it can display. For our purposes, the gamut of the device will be the RGB colours we defined previously. Any colour that goes beyond this is called *out-of-gamut* as the colours do not map to displayable colours (at least not without artifacting). Colours are also said to *overflow* when they go out of gamut. As it is perfectly common for colours to overflow during the process of computing the radiance, we have to do something about them. What we will do is to first compute the colour for the pixel regardless of whether they are in gamut or not. This is called *high dynamic range*. Once we have the colour, we can now apply a *local tone-mapping operator* to bring the colours back into gamut.

There are a number of different ways of implementing a local tone-mapping operator, but here we will discuss two simple approaches:

1. *Max-to-one*: this is a global tone-mapping operator that works by checking if any of the RGB values are out of gamut. If they are, then the largest one is chosen and all three values are divided by this value. The advantage of this approach is that it preserves hue and leaves the maximum component as 1.
2. *Clamp*: this function will map all out of gamut colours to a specified colour. This has the advantage of being able to display any regions where the colours go beyond the gamut, which may be useful in certain applications such as photography.