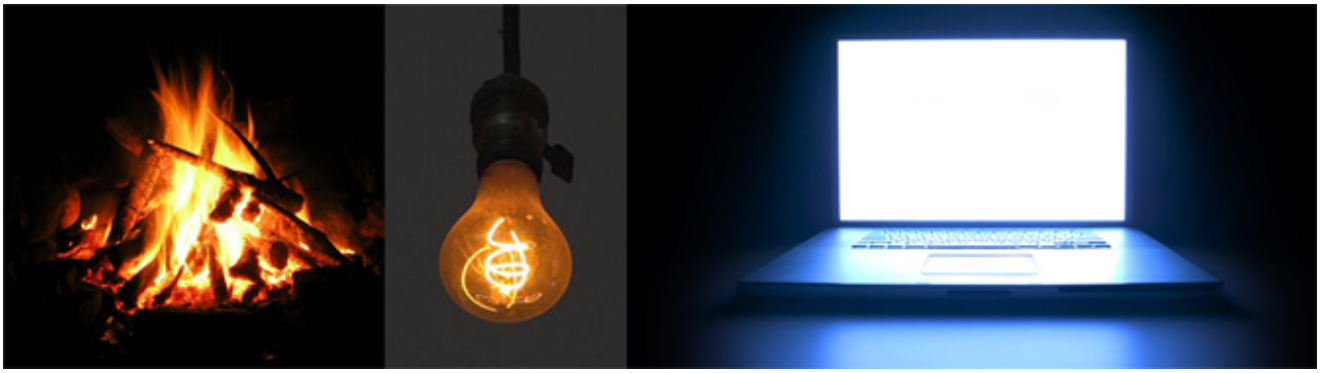
Introduction to Shading

Contents

Lights

Before we look into producing our first shaded image, we first need to introduce the concept of light. A scene as we mentioned in previous lessons is composed of a camera, objects and lights. Lights are special "entities" whose only function is to indicate where is light emitted from in the scene. As mentioned before, the only reason why we see objects is because light emitted by light sources, bounces off of the surface of objects. If you don't create a light in the scene, the scene should be rendered black.



In the real world, every light source has a physical body. Light sources are objects which have the property to emit light. But a light source is nothing else than a standard object: it has a shape and a size. Though it has a property that other objects don't have: it emits light. Because light sources are objects in their own right, they can also be directly seen by the eyes. Though generally, never looked a light sources directly with naked eyes as it can be harmful. Looking at the sun with naked yes, will burn your eyes. Though, we can contemplate other sources of light safely such as for example the flames of a campfire or the light bulb of a torch. If you are reading these lines, you are even looking at one just now: the screen of your phone or computer. The problem is that in CG, simulating light in a physically accurate way, that is by representing them as objects with their own shape and size, is computationally expensive. Such lights are called **area lights** or more generally geometric area lights. For this reason, lights in CG have for a very long time only been represented as idealised objects, that is, as entities with no physical size. In opposition to area lights such lights are also sometimes called **delta lights** (from the term delta function which is a special function in mathematics that was created to sort of define a function that can't exist in nature but does exist in the abstract world of mathematics. Although abstract it is super useful to solve all sort of interesting and practical problems). We can differentiate essentially two types of delta lights: directional or **distant lights** and **spherical lights** or **point light sources**. The former are generally a subset of the latter, but we will explain this in a moment.

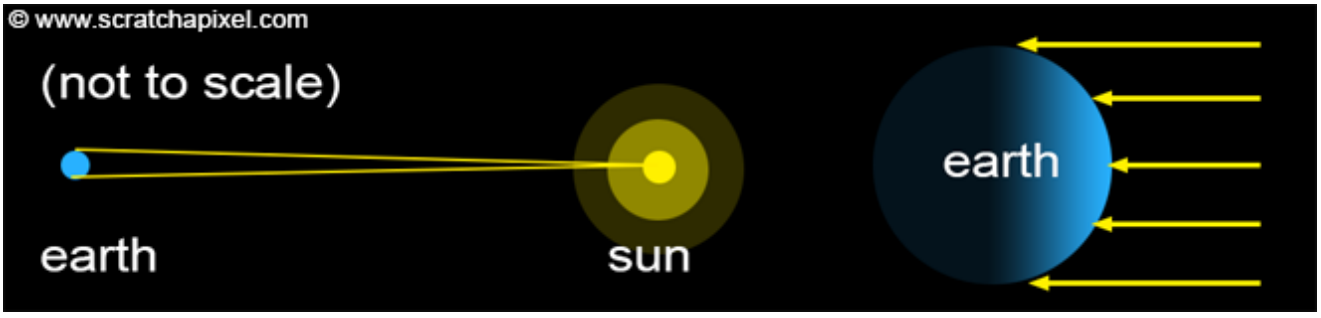
> You need to know: delta lights were used in the early days of computer graphics for practical reasons because simulating area lights was for a very long time too expensive. As computers became faster in the late 2000s, it also slowly became practically possible to switch to area lights. Using delta lights, especially today should be avoided as much as possible. Why? Because not representing light as area lights causes a lot of problems. For example, we know that objects reflects other objects and of course light sources since they are objects in their own right should also be reflected by glossy surfaces or mirrors for instance. Though the size of the reflection of an object by a glossy surface, depends on the reflected object's size and distance to the glossy surface (these two

concepts, size of the object and distance of the reflected object to the reflective surface can be combined into the concept of solid angle which we won't talk about much in this lesson. For more information on this topic, please check the lessons from the advanced sections). When lights have no size, then it is simple impossible to decide how big their reflection by a glossy surface should be! This has caused a lot of problems in the field of rendering for many years. People had to use hacks to control the size of these reflections by adjusting the roughness of the reflective surface as well as artificially controlling that roughness by adjusting a "roughness" parameter on the light itself. Doing so would help you cheat the size of the light on a per light basis, but this could certainly never lead to producing physically accurate images. Hopefully such practices have almost entirely disappeared these days, mostly due to the fact that simulating area lights as just mentioned, is now affordable (though simulating area lights is still computationally expensive and requires careful optimisations).

Using area lights is a condition to **physically based rendering** or **PBR** which is a term you may be already familiar with. This topic is covered in great details in the lessons from the advanced sections [links]. You can also find more information about it in the last lesson of this section.

Distant Lights

Distant lights are lights which are considered to be so far away from us, that the light they emit is only reaching us in the form of light rays parallel to each other. With such light sources, all we really care about is thus the direction of these light rays. An example of distant light is our sun. The sun is a sphere, thus you may rightly think that it is a spherical light. At the scale of the solar system or even at a larger scale, yes, the sun acts as a spherical light. Though, as shown in the image below, the Earth is small compared to the sun and more importantly so far away from the star, that sunlight that reaches the Earth surface is contained within a super small cone of directions.



In fact the solid angle of this cone for the readers familiar with the concept of solid angle is only roughly 0.0000687 steradians. In CG, when we render a scene, the scene only covers a small area of the Earth's surface. Consequently we can safely assume that sun rays illuminating that scene are all parallel to each other. There might be a small variation but it is so negligible that we can ignore it all together (we will probably never have enough numerical precision to represent this variation in a computer anyway). To say it differently: we don't really care. Sun rays hitting the Earth's surface are parallel to each other. Period. Consequently, all we care about in this particular case, is the light rays direction, hence the term **directional light**. Or to say it differently the position of the light source with regards to the rendered scene is irrelevant because again, the only reason why rays are parallel to each other is because the source is super far away from the scene, and thus rays illuminating the scene are contained in a tiny cone of directions. Hence the term distant light. In essence, all we need in CG to simulate the sun or any other distant light is a direction and nothing else.

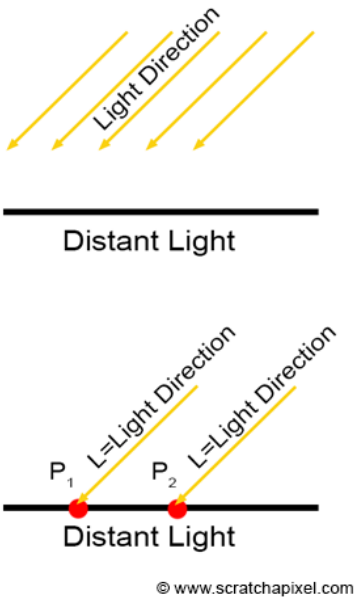


Figure 1: distant lights only require a direction.

Though, note the sun is very far away but it has a size in the sky nonetheless. In fact, as surprising as it may seem, the sun is bigger in the sky than the moon.

In this lesson, we will first render diffuse objects and learn how to cast shadows using distant lights. We will then learn how to simulate spherical lights.

Spherical Light Sources

Spherical light sources are the most common type of light sources found in nature. To some extent even lights which are not spherical can be somehow approximated as a collection of spherical light sources. To the difference of distant light sources, the position of spherical lights matters. In fact, for spherical light sources, this is the only thing that matters. We need to know where they are in space. If we call P a point on the surface of an object that we want to shade, then finding the direction of the ray that a spherical light source emits in the direction of P is really simple: it is simply P minus the spherical light position (let's call it L_P) as shown in figure 2.

$$\text{Light Direction} = P - L_P.$$

In the next chapters, we will show that the distance between P and the light source also matters (for a spherical light but not for a distant light, the amount of light illuminating an object depends on the distance between the light and the object). We will need this distance, which we can then use to normalize the light direction vector. This will be explained later in the lesson.

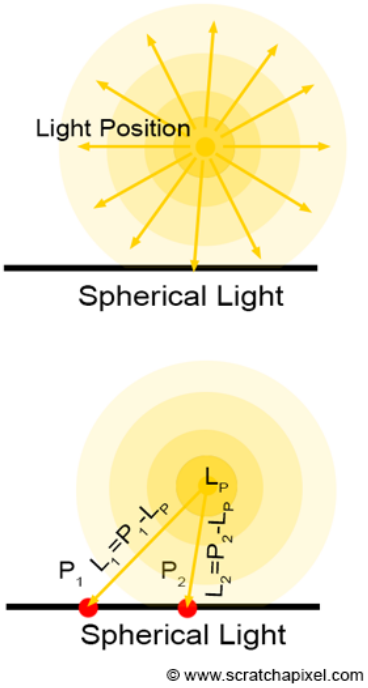


Figure 2: spherical lights only require a position.

Light Intensity (and Color)

Despite the light position (if it is a spherical or point light source) or the light direction (if it is a distant or directional light) what else do we need to define a light source? Well, light sources emits light which in CG, we can simply represent as a color. It is generally best to define the light color as a combination of a color and an intensity (a real number). The color of the light can be kept in the range [0,1] and the light intensity can take any value between 0 and infinity (or in computer code, the maximum value we can represent with a float for example). The final amount of emitted light is obtained by multiplying the light color by its intensity:

$$\text{light amount} = \text{light color} * \text{light intensity}.$$

Implementation

In code, we will differentiate lights from geometry by creating a special `Light` class. We will add to this base class the following member variables:

- `lightToWorld`: lights too can be transformed by 4x4 matrices. In fact we will use this matrix to compute the position of spherical lights and the direction of directional lights.
- `color`: the RGB color of the light (with values in the range [0,1]).
- `intensity`: the light intensity.

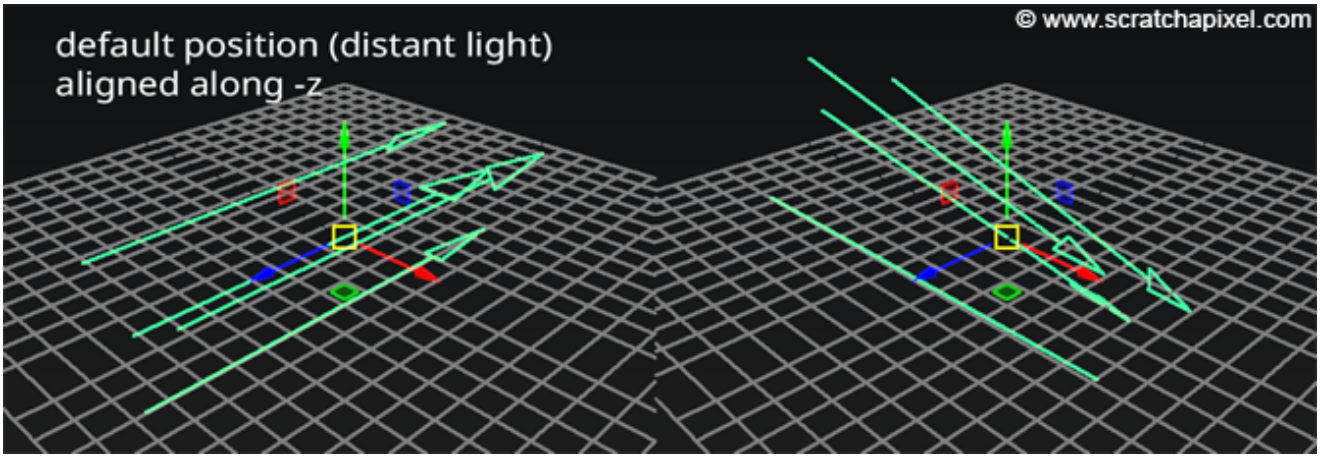
Let's look at the code:

```
class Light
{
public:
    Light(const Matrix44f &l2w) : lightToWorld(l2w) {}
    virtual ~Light() {}
    Matrix44f lightToWorld;
    Vec3f color;
    float intensity;
};
```

By default, we will assume that points light are created at the origin of the world. We will use the light-to-world matrix to transform the light to its position in world space. Note that lights are unaffected by scale. Point lights are also unaffected by rotation (but distant lights are). Distant lights are unaffected by translation. Let's now look at one possible implementation of a point light source:

```
class PointLight
{
public:
    PointLight(const Matrix44f &l2w, const Vec3f &c = 1, const float &i = 1) : Light(l2w)
    {
        this->color = c;
        this->intensity = i;
        l2w.multVecMatrix(Vec3f(0), pos);
    }
    Vec3f pos;
};
```

Similarly to point light source, we will assume that by default the direction of a distant light source is pointing along the negative z-axis (if you have access to Maya create a distant light source and check its default orientation). In other words, the default light direction is (0,0,-1).



Again, to change or control the light direction, we will change the light-to-world transformation matrix:

```
class DistantLight
{
public:
    DistantLight(const Matrix44f &l2w, const Vec3f &c = 1, const float &i = 1) : Light(l2w)
    {
        this->color = c;
        this->intensity = i;
        l2w.multDirMatrix(Vec3f(0, 0, -1), dir);
        dir.normalize();
    }
    Vec3f dir;
};
```

Now that we know how to create lights in our system, let's learn about simulating the appearance of diffuse objects.