Chapter 4

Lighting

4.1 Introduction

Illumination is a key component to almost every application of computer graphics. Even in situations where no attempts are made to mimic reality, a virtual lighting with at least some basic similarities to real world lights is useful to convey information about the relative position and distances of virtual objects. In the many applications where realism is desirable, the lighting is crucial to success. A scene with ever so detailed geometric models and highly accurate materials will still look bland if it's not properly lit.

In the real world, lighting is usually a quite complicated matter even for seemingly simple scenes. Every illuminated object reflects light not only towards the camera, but also onto its surroundings, acting as a secondary light source. The general problem of computing a physically accurate simulation of light transport in a computer graphics rendering is called *global illumination*. In situations where speed is more important than accuracy, simpler models are used to compute *local illumination*, where the main focus is on direct light incident from light sources onto surfaces, while secondary reflections and other interactions between objects are either simplified or not modelled at all. This chapter will focus on local illumination. Most of the discussion around global illumination will be saved for the chapter on off-line rendering.

4.2 Light types

4.2.1 Directional light

Throughout computer graphics history, some very simple models for light sources have been used with good results. Discounting overly simplified generalizations like the "ambient light" of the Phong reflection model, one of the simplest kinds of lighting is a *directional light source*, where the light is described by its intensity and color and a single directional vector. The light is assumed to shine from the same direction everywhere in the scene, and its intensity does not decay with distance. A real world example of such a light source is the Sun, as seen from far away like on the Earth's surface. Because the Earth is so far away from the Sun, all rays from the Sun that hit the Earth come from more or less the same direction, and sunlight in a scene can be modelled as parallel rays. In mathematical terms, what is needed to model a directional light is its intensity I (typically a three-component RGB value) and its direction $\mathbf{L}$ (typically a normalized vector).

4.2.2 Point light

We also need to be able to model lights that are placed near the scene, or even in the scene. Instead of describing the light by a constant directional vector, $\mathbf{L}$, we can set the position of the light source $\mathbf{p}_L$ and compute the light direction as the vector from the current surface point $\mathbf{p}$ to the light: $\mathbf{L} = \mathbf{p}_L - \mathbf{p}$, or, because we prefer the vector to be of unit length, $\mathbf{L} = \frac{\mathbf{p}_L - \mathbf{p}}{|\mathbf{p}_L - \mathbf{p}|}$

4.2.3 Spotlight

Lights that shine uniformly in all directions are not terribly common in the real world. Most light sources have some kind of lamp shade or armature that focuses and/or blocks the light to restrict the illumination only to certain directions. The intensity will also typically be different for different directions, either on purpose or because of physical limitations and other restrictions in the design of the light. Point lights that shine predominantly along one direction are often called *spotlights*. A point light has no orientation and can be described by its position only, but a spotlight needs a direction and preferably also a sense of what is "up", unless the "light cone" emanating from the spotlight has a perfectly circular cross-section. In computer graphics, the position of a spotlight is most conveniently specified in the same manner as the position of a camera: in the form of a *matrix* that transforms the world space into a local coordinate space for the light. The most common choice is to place the light at the origin and make the main direction of the light one of the coordinate axes, for example the negative z direction to make light matrices compatible with camera matrices in OpenGL. Using the transformation matrix of a light source as the camera matrix will then make that camera "see" what the light illuminates, which is convenient when setting up the lighting for a scene.

The directionally dependent intensity variation of a spotlight can be expressed either as a falloff with the angle to the main direction of illumination, which is easily computed by means of a scalar product, or as a texture mapped 2-D pattern where the texture coordinates are computed by projective mapping, just like a perspective camera projection maps a point in space to a position in the rendered image. The texture can be either image-based for full generality (creating a *projector spotlight*), or the pattern can be expressed as a function in (u, v) texture space.

Real world spotlights project a different pattern of light in close-up illumination than at a distance. This is called *near field effects*, and some computer graphics applications choose to model at least some of that effect as well. The perspective projection of a projector spotlight can still be used, but the projected pattern should depend also on distance, which means it should be a 3-D function defined in (u, v, w) texture space.

4.2.4 Area lights

Real world lights are not infinitely small points. Some lights are small compared to the scene and may be closely approximated by points, but many real world lighting designs strive for a "soft light" where light is emitted from larger areas, making the illumination smooth and the shadows fuzzy. The illumination properties of *area lights* can be computed analytically, but for most situations, in off-line rendering as well as in real time rendering, area lights are simulated by a number of point lights distributed over an area, where each point light contributes a fraction of the total intensity. Computing the influence of many simple point lights, maybe 100 or even more of them, can actually involve less work than accurately computing the influence of a single area light. Representing area lights with a set of point lights also allows for an arbitrary shape of the area light and an arbitrary variation of intensity across its surface, both of which are difficult to incorporate in an exact analytical model. Shadows, another important property that will be treated below, are also often prohibitively difficult to compute analytically for area light sources.

4.2.5 Skylight

A special kind of very useful area light is a *skylight*, which tries to model the influence of a bright environment on objects in the scene. Applications are not limited to light from the sky – the influence of an indoor environment with white walls would be well approximated by a skylight. In most real world situations, the light from the environment is *indirect light*: energy which does not emanate directly from light sources but which has has bounced off at least one diffuse reflective surface. A skylight is a lot better at modelling "ambient light" than the crude ambient term in the Phong model, but it can also be used as the main illumination for a scene. Using only a skylight for illumination will create a very soft light, but that is sometimes desirable.

Ambient occlusion

The problems of computing the influence of a skylight stems from the fact that it shines from all directions, and simulating it with an area light source would require a large number of sample points on an entire hemisphere. To speed up computations, various tricks are employed. One of these tricks is *ambient occlusion*, a method where the geometry of the scene is pre-processed to determine how much of the environment is visible from each surface point. The pre-processing is often limited to looking only within a short distance around each point, and often kept local to the object itself. The visual impact of ambient occlusion is that convex corners and ridges on the surface will have low occlusion and a higher amount of illumination from ambient light, while narrow valleys, small holes and concave corners will have a high occlusion and a lower amount of ambient light. Local ambient occlusion can also be performed in screen space using the z buffer, which allows an approximate version of the effect to be computed in real time. This *screen-space ambient occlusion* ("SSAO") trick is currently very popular in games.

4.2.6 Decay

The length of the computed vector from a surface to a point light or a spotlight is the distance to the light source, which makes it possible to compute a *decay with distance.* Real world light sources have an intensity that is proportional to the inverse of the square of the distance, $I \propto 1/|\mathbf{L}|^2$, provided that the light source is small compared to its distance. This, however, creates a strong dependence on the absolute distance to the light, and in computer graphics the decay is often eliminated or reduced to a weaker variation, like an inverse linear decay $I \propto 1/|\mathbf{L}|$ or, more commonly, a polynomial with both linear, quadratic and possibly constant terms, $I \propto 1/(a + b|\mathbf{L}| + c|\mathbf{L}|^2)$.

There are several motivations for using something else than the physically correct inverse square decay. First and foremost, the contrast of a typical real world scene is considerably larger than what you want in a computer graphics rendering, and making the decay with distance less pronounced than in the real world will reduce the contrast of the rendered image, making it easier to keep the output pixel values within a reasonable and predictable range. This motivation has become less relevant with the introduction of *high dynamic range (HDR) rendering*, where output pixels are first computed as floating point values with high precision and wide range and the rendered image is then *tone mapped* or "auto exposed" for a display or printout where the contrast range is restricted, but HDR rendering is still fighting against a long and strong tradition of rendering to an 8-bit buffer with very limited precision and a hard limit on the maximum intensity. A lot of the methods that have been developed for 8-bit rendering need to be modified for HDR rendering, and right now (2019) we are only half-way through that transition.

There are other reasons as well for not using a strict inverse square decay. Near an ideal point light, the intensity increases very rapidly, with a singularity yielding infinite intensity at zero distance. A real light source is not infinitely small, the intensity at the surface of the light source is certainly not infinite, and the decay with distance is less pronounced close to the light source than far away. Doing the math for a light source that is a sphere or a circle rather than a point, the decay with distance looks very much like a polynomial with a constant term, a linear term and a quadratic term, like the equation above. The illumination starts out at a high but not infinite intensity at the surface of the light, remains fairly constant close to the light source, starts to decay gradually and slowly at medium distances, and changes to a square decay only further away. (Actually doing this computation is a useful exercise in geometry, but it is beyond the scope of this presentation.)

4.2.7 Useful non-realism

Finally, there are some situations where realism is not the most desirable property of a computer graphics light source. In real world lighting for stage and film, people would kill for a light that has the same intensity everywhere in the scene and does *not* decay with distance, and that is perfectly possible to achieve with virtual lights in computer graphics. Other useful but unrealistic properties of virtual lights could be to have a decay that starts only after a while, a light that illuminates things only within a certain range of distances, illuminates only certain objects or penetrates a scene without having intermediate objects block the light and cast a shadow.

4.3 Shadows

Shadows are formally a global illumination property, because an object casting a shadow is influencing the illumination of other objects. Strict local illumination models are only concerned with light sources and surfaces directly illuminated by those light sources, and because of this they can't model shadows. However, shadows are a very important visual property of most real world scenes, and computer graphics wouldn't do well without them. Over the years, local illumination models have been complemented with various pre-processing methods to introduce fake shadows into the rendering. One such method that deserves special mention, most notably because it is still in widespread use, is *shadow mapping*. (Other useful methods exist for fake shadows, but they are more specialized and are left out of this presentation.)

4.3.1 Shadow mapping

Shadow mapping is an extension of the concept of a *depth buffer*, where a separate channel of the rendered image is used to store the distance from the camera to the current fragment. The depth buffer resolves in an elegant and quick way some rather tricky problems with computing per-pixel visibility, and a depth buffer is almost always used in modern real time rendering.

To extend this to light sources, we make the observation that the parts that are in shadow from a light source are the surfaces that are not visible from the vantage point of the light source. To determine whether a light source "sees" a certain point on a surface, we can use a *shadow map*, a precomputed texture with depth information, rendered from the vantage point of the light source. During rendering, the rendered point is transformed to the coordinate system of the shadow map, and a texture lookup is used to determine whether that particular point was visible from the light source or not. If the depth value stored in the shadow map is the same, or almost the same, as the depth value of the transformed point, the point is visible from the light source and should be illuminated by it. If, on the other hand, the stored depth is smaller than the depth of the transformed point, that point is behind some other surface that blocks the light, which means that the point is in shadow. Figure 4.1 and its accompanying caption explains this in more detail.

Shadow mapping is comparably easy to explain in principle, but in practice there are several problems that need to be addressed. One is that the comparison between the depth values in two different coordinate systems involves several interpolations and a matrix transformation with limited precision. Hence, a point should be determined to be in shadow only if its depth value differs by some safe margin from the depth stored in the shadow map. Failing to allow for limited precision results in so-called "shadow acne", where surfaces exhibit a spotted pattern of self-shadowing, because the comparison incorrectly claims that the surface is behind itself. On the other hand, allowing the margin of error to be too large could make light leak through to the other side of thin objects and make so-called *contact shadows*, shadows from objects that are placed directly on top of other objects, detach from the base of objects and appear to "float". This safe margin is called the *shadow bias*, and getting it right for all points in general scene requires extra care and some tricks which we won't explain further here.

There are also several problems related to the sampling of a shadow map. Because the depth values from the shadow map are used in comparisons, points are determined to be either completely in shadow or completely illuminated, and the edges of the shadows become crisp and pixelated. To create shadows with soft edges, other strategies than a binary "on or off" comparison need to be employed. These strategies are not explained here either.

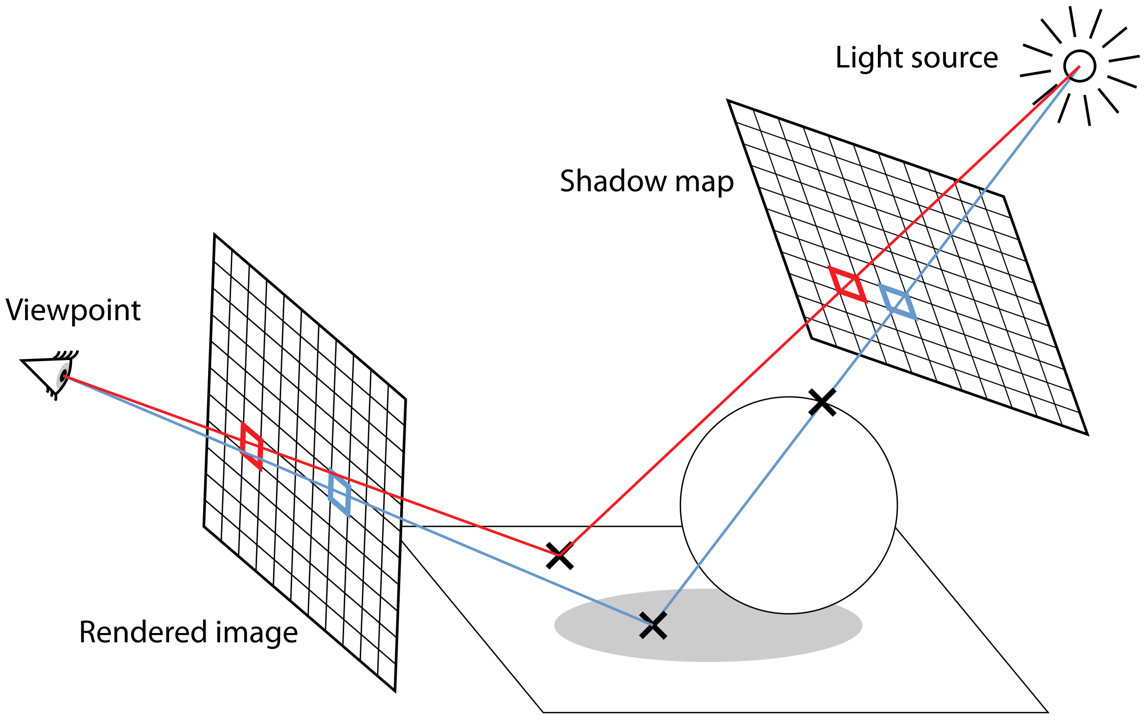

Figure 4.1: Shadow mapping requires a two-step rendering process. First, a *shadow map* is computed. This is a depth buffer image as seen from the light source. Second, when the final image is rendered, the scene point is transformed to the coordinate space of the shadow map, and its distance to the light source is compared to the pre-computed depth in the shadow map. If the point is farther away than the z value stored in the shadow map (blue rays in the figure), the point is in shadow and should not be illuminated by the corresponding light source. If, however, the point is at the same distance (red rays in the figure), it is illuminated.

Suffice it to say that shadow maps are a simple enough concept to explain, but to implement them correctly requires considerable care. Improving the speed and quality of shadow map computations for various situations is still an active area of research. Shadow mapping is a method which is several decades old, but new variations are still being developed, and for quite some time yet, shadow mapping will remain a very useful method in real time rendering.