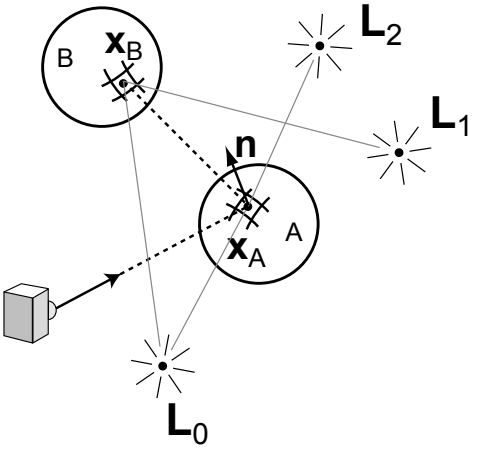
Chapter 17

Raytracing

17.1 Reflections



Now we need to look at another kind of illumination that can affect the shade of a surface we are rendering. This is called indirect or *global illumination*. Direct or *local illumination* comes directly from a light source. Indirect global illumination comes from light inter-reflections among objects in the scene. Examine the figure to the left. The camera is examining point $\mathbf{x}_A$ on sphere A. The point $\mathbf{x}_A$ is directly illuminated by lights L_0 and L_2. It is in shadow with respect to light L_1, which is behind sphere A. However, if we follow a light reflection ray from the surface of A (see vector notes on mirror reflection), this ray intersects sphere B at $\mathbf{x}_B$. Now, point $\mathbf{x}_B$ is directly illuminated by light L_0 and L_1 but is in shadow with respect to light L_2. Some of the illumination reflecting from $\mathbf{x}_B$ will reach $\mathbf{x}_A$ and be seen by the camera.

This fact is exploited in a raytracer to develop a recursive shading routine that accounts for the light in the scene that is reflected in the mirror reflection

direction from the objects in the scene. This turns out to be a very effective way of simulating the lighting of shiny specular reflectors, i.e. objects that have strong mirror reflections. It is less effective in capturing the global illumination in a scene with diffuse reflectors.

A ray shot from the camera and hitting an object is followed via mirror reflection, and the light traveling along this ray is accounted for in the shading calculation. This is usually called *recursive raytracing* because it can be implemented recursively in the shade routine. This recursive shade routine would look like this in pseudocode:

```
color shade(objectid o, position x, vector u_r, int level){
  if(level > max_recursion_level)
    return black;
  else{
    u_m = reflection_vector(x, o, u_r);
    (o_m, t) = shoot(x, u_m);
    if(t == INFINITY)
      color = background_color;
    else{
      m_color = shade(o_m, x + t * u_m, u_m, level + 1);
      color = directshade(o, x, u_r, m_color, -u_m);
    }
    for(each light i in the scene)
      if(light i is visible from x)
        color += directshade(o, x, u_r, light[i].color,
                             light[i].direction);
    return color;
  }
}
```

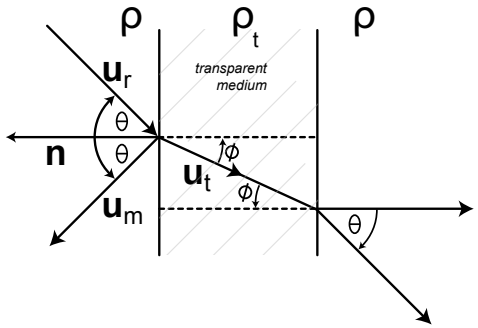
The `shade()` routine works as follows. Its parameters include o, the object that has been hit at spatial position $\mathbf{x}$ by a ray whose direction vector is $\mathbf{u}_r$. The `level` parameter is initialized to 0 upon the first call to `shade()`, and is used to cut off recursion if the number of bounces specified by the user defined constant `max_recursion_level` is reached. If we have not exceeded the maximum number of bounces, then we compute $\mathbf{u}_m$, which is the mirror reflection of vector $\mathbf{u}_r$ from the surface of object o at position $\mathbf{x}$. We then probe the scene by shooting a "bounce" ray from the original hit-point $\mathbf{x}$ in the direction $\mathbf{u}_m$. If this ray hits nothing, then we assign the background color to the probe. However, if this ray does hit an object o_m, then we recursively call the `shade()` routine from the hit point on object o_m, using the reflection vector $\mathbf{u}_m$ as the new ray direction, and incrementing the level parameter to indicate an extra level of recursion. This will return the shaded color `m_color` of this new hit-point as the color of the probe. We treat this as if it were a light shining in the direction $-\mathbf{u}_m$ with color `m_color`, and pass this to the routine `directshade()`, which does the computation of the ambient, diffuse, and specular reflection contributed by

this light source. Finally, we add in illumination contributions from each light that is visible from the original hit point **x**, again using `directshade()`. Using only lights visible from **x** assures that this point will be in shadow with respect to any light that is not visible.

The big advantage to recursive raytracing is that the global illumination calculation directly provides reflections (for shiny objects) and shadows, which are not possible in a local renderer without special code.

17.2 Refraction

To complete the raytracing story, we need to deal not only with ray reflection but with ray refraction. A light ray $\mathbf{u}_r$ striking a transparent surface will essentially split into a mirror reflected ray $\mathbf{u}_m$ and a transmitted ray $\mathbf{u}_t$. If $\mathbf{n}$ is the surface normal where the ray strikes the surface, then both $\mathbf{u}_m$ and $\mathbf{u}_t$ will lie in the plane formed by $\mathbf{u}_r$ and $\mathbf{n}$. In this plane, we know that if the angle between the incoming ray $\mathbf{u}_r$ and the normal $\mathbf{n}$ is θ, then the reflected ray $\mathbf{u}_m$ will also be at angle θ to $\mathbf{n}$. In a previous chapter we showed that the mirror reflection ray is given by



$$\mathbf{u}_m = \mathbf{u}_r - 2(\mathbf{u_r} \cdot \mathbf{n})\mathbf{n}.$$

The transmitted ray $\mathbf{u}_t$, on the other hand, will obey Snell's Law, which relates the angle to the normal to the index of refraction of the material, ρ (note: the index of refraction is normally written as n, but we will use ρ to avoid confusion with the surface normal). The index of refraction in a vacuum is defined to be 1, with all the other transmitting media having an index of refraction greater than 1. Air has an index of refraction just slightly greater than 1, so it is typical to assume $\rho = 1$ for the environment of a typical raytraced scene. However, water has an index of refraction $\rho = 1.33$, glass has an index $\rho = 1.52$, and diamond $\rho = 2.42$.

Snells Law states that

$$\rho_r \sin\theta = \rho_t \sin\phi,$$

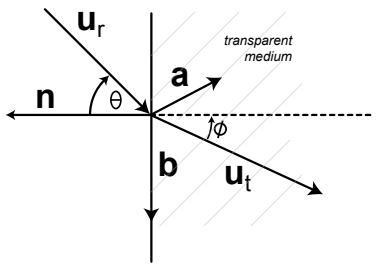
where ρ_r is the index of refraction in the external medium, ρ_t is the index in the

transmitting medium, and ϕ is the angle of the transmitted ray to the surface normal $\mathbf{n}$ at the ray surface intersection.

We can use Snell's Law to derive an equation for the transmitted ray $\mathbf{u}_t$. First, we construct a coordinate frame in the plane of $\mathbf{u}_r$ and $\mathbf{n}$

$$\begin{aligned} \mathbf{a} &= (\mathbf{n} \times \mathbf{u}_r)/\|\mathbf{n} \times \mathbf{u}_r\|, \\ \mathbf{b} &= \mathbf{a} \times \mathbf{n}, \end{aligned}$$

where $\mathbf{a}$ points out of the page in the diagram to the right, and $\mathbf{b}$ is tangent to the surface at the intersection point. Now we have:



$$\begin{aligned} \mathbf{u}_r \cdot \mathbf{n} &= -\cos\theta, \quad \mathbf{u}_t \cdot \mathbf{n} = -\cos\phi, \\ \mathbf{u}_r \cdot \mathbf{b} &= \sin\theta, \quad \mathbf{u}_t \cdot \mathbf{b} = \sin\phi. \end{aligned}$$

By Snells Law

$$\sin\phi = \frac{\rho_r}{\rho_t}\sin\theta = \frac{\rho_r}{\rho_t}\mathbf{u}_r \cdot \mathbf{b},$$

and by the Pythagorean theorem

$$\cos\phi = \sqrt{1 - \sin^2\phi}.$$

Thus, in $(\mathbf{n}, \mathbf{b})$ coordinates

$$\mathbf{u}_t = [-\cos\phi, \sin\phi]$$

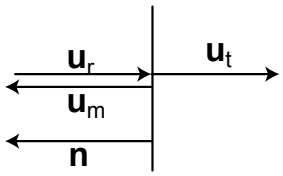
or in our original (x, y, z) coordinates

$$\mathbf{u}_t = -\mathbf{n}\cos\phi + \mathbf{b}\sin\phi.$$

Now we know that the mirror reflected ray $\mathbf{u}_m$ and the transmitted refracted ray $\mathbf{u}_t$, but we need to know the fraction of the original ray that is reflected and the fraction that is transmitted, so that in shading calculations we can weight reflection color and transmission color appropriately. These fractions are given by the the Fresnel equations

$$\begin{aligned} R_0 &= (\frac{\rho_t - 1}{\rho_t + 1})^2, \\ R(\theta) &= R_0 + (1 - R_0)(1 - \cos\theta)^5, \end{aligned}$$

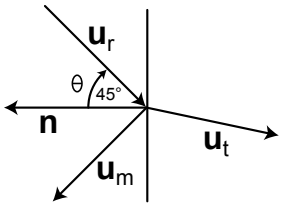
where R is the fraction of light reflected, and R_0 is the fraction reflected when the ray and normal are parallel. Below are some examples of the factor R:



In this case

$$R(\theta) = R(0) = R_0.$$

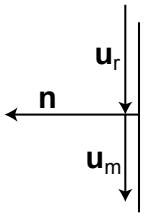
Thus, the reflection ray is weighted by R_0 and the transmitted ray by $(1 - R_0)$



In this case

$$R(\theta) = R(45^\circ) = R_0 + (1 - R_0)(1 - \frac{\sqrt{2}}{2})^5$$

Thus, the reflection ray is weighted by $R(45^\circ)$ and the transmitted ray by $[1 - R(45^\circ)]$.



In this case

$$R(\theta) = R(90^\circ) = 1.$$

Thus, there is no transmitted ray.