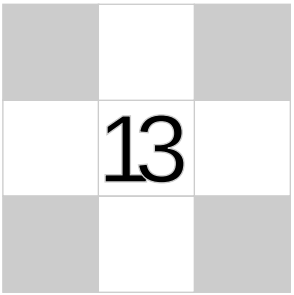

13 More Ray Tracing

A ray tracer is a great substrate on which to build all kinds of advanced rendering effects. Many effects that take significant work to fit into the object-order rasterization framework, including basics like the shadows and reflections already presented in Chapter 4, are simple and elegant in a ray tracer. In this chapter we discuss some fancier techniques that can be used to ray-trace a wider variety of scenes and to include a wider variety of effects. Some extensions allow more general geometry: instancing and constructive solid geometry (CSG) are two ways to make models more complex with minimal complexity added to the program. Other extensions add to the range of materials we can handle: refraction through transparent materials, like glass and water, and glossy reflections on a variety of surfaces are essential for realism in many scenes.

This chapter also discusses the general framework of *distribution ray tracing* (Cook et al., 1984), a powerful extension to the basic ray-tracing idea in which multiple random rays are sent through each pixel in an image to produce images with smooth edges and to simply and elegantly (if slowly) produce a wide range of effects from soft shadows to camera depth-of-field.

The price of the elegance of ray tracing is exacted in terms of computer time: most of these extensions will trace a very large number of rays for any non-trivial scene. Because of this, it's crucial to use the methods described in Chapter 12 to accelerate the tracing of rays.

If you start with a brute-force ray intersection loop, you'll have ample time to implement an acceleration structure while you wait for images to render.

13.1 Transparency and Refraction

In Chapter 4 we discussed the use of recursive ray tracing to compute specular, or mirror, reflection from surfaces. Another type of specular object is a *dielectric*—a transparent material that refracts light. Diamonds, glass, water, and air are dielectrics. Dielectrics also filter light; some glass filters out more red and blue light than green light, so the glass takes on a green tint. When a ray travels from a medium with refractive index n into one with a refractive index n_t, some of the light is transmitted, and it bends. This is shown for $n_t > n$ in Figure 13.1. Snell's law tells us that

$$n \sin\theta = n_t \sin\phi.$$

Example values of *n*:
air: 1.00;
water: 1.33–1.34;
window glass: 1.51;
optical glass: 1.49–1.92;
diamond: 2.42.

Computing the sine of an angle between two vectors is usually not as convenient as computing the cosine, which is a simple dot product for the unit vectors such as we have here. Using the trigonometric identity $\sin^2\theta + \cos^2\theta = 1$, we can derive a refraction relationship for cosines:

$$\cos^2\phi = 1 - \frac{n^2\left(1-\cos^2\theta\right)}{n_t^2}.$$

Note that if n and n_t are reversed, then so are θ and ϕ as shown on the right of Figure 13.1.

To convert $\sin\phi$ and $\cos\phi$ into a 3D vector, we can set up a 2D orthonormal basis in the plane of the surface normal, $\mathbf{n}$, and the ray direction, $\mathbf{d}$.

From Figure 13.2, we can see that $\mathbf{n}$ and $\mathbf{b}$ form an orthonormal basis for the plane of refraction. By definition, we can describe the direction of the transformed

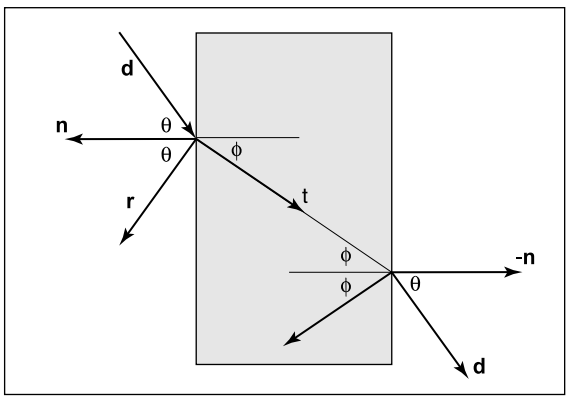

Figure 13.1. Snell's Law describes how the angle ϕ depends on the angle θ and the refractive indices of the object and the surrounding medium.

ray, $\mathbf{t}$, in terms of this basis:

$$\mathbf{t} = \sin\phi\mathbf{b} - \cos\phi\mathbf{n}.$$

Since we can describe $\mathbf{d}$ in the same basis, and $\mathbf{d}$ is known, we can solve for $\mathbf{b}$:

$$\mathbf{d} = \sin\theta\mathbf{b} - \cos\theta\mathbf{n},$$
$$\mathbf{b} = \frac{\mathbf{d} + \mathbf{n}\cos\theta}{\sin\theta}.$$

This means that we can solve for $\mathbf{t}$ with known variables:

$$\begin{aligned}\mathbf{t} &= \frac{n\left(\mathbf{d} + \mathbf{n}\cos\theta\right)}{n_t} - \mathbf{n}\cos\phi \\ &= \frac{n\left(\mathbf{d} - \mathbf{n}(\mathbf{d}\cdot\mathbf{n})\right)}{n_t} - \mathbf{n}\sqrt{1 - \frac{n^2\left(1 - (\mathbf{d}\cdot\mathbf{n})^2\right)}{n_t^2}}.\end{aligned}$$

Note that this equation works regardless of which of n and n_t is larger. An immediate question is, "What should you do if the number under the square root is negative?" In this case, there is no refracted ray and all of the energy is reflected. This is known as *total internal reflection*, and it is responsible for much of the rich appearance of glass objects.

The reflectivity of a dielectric varies with the incident angle according to the *Fresnel equations*. A nice way to implement something close to the Fresnel equations is to use the *Schlick approximation* (Schlick, 1994a),

$$R(\theta) = R_0 + (1 - R_0)\left(1 - \cos\theta\right)^5,$$

where R_0 is the reflectance at normal incidence:

$$R_0 = \left(\frac{n_t - 1}{n_t + 1}\right)^2.$$

Note that the $\cos\theta$ terms above are always for the angle in air (the larger of the internal and external angles relative to the normal).

For homogeneous impurities, as is found in typical colored glass, a light-carrying ray's intensity will be attenuated according to *Beer's Law*. As the ray travels through the medium it loses intensity according to $dI = -CI\,dx$, where dx is distance. Thus, $dI/dx = -CI$. We can solve this equation and get the exponential $I = k\exp(-Cx) + k'$. The degree of attenuation is described by the RGB attenuation constant a, which is the amount of attenuation after one unit of distance. Putting in boundary conditions, we know that $I(0) = I_0$, and

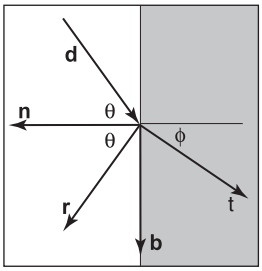

Figure 13.2. The vectors **n** and **b** form a 2D orthonormal basis that is parallel to the transmission vector **t**.

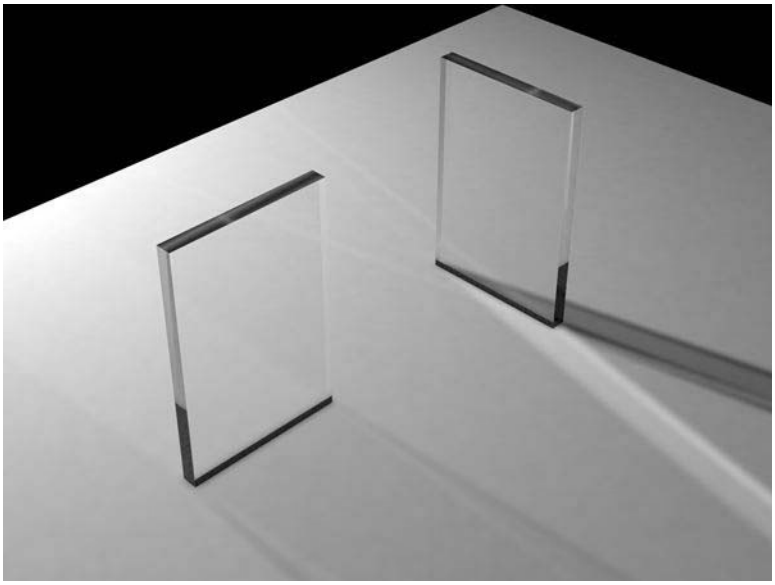

Figure 13.3. The color of the glass is affected by total internal reflection and Beer's Law. The amount of light transmitted and reflected is determined by the Fresnel equations. The complex lighting on the ground plane was computed using particle tracing as described in Chapter 24. (See also Plate IV.)

$I(1) = aI(0)$. The former implies $I(x) = I_0 \exp(-Cx)$. The latter implies $I_0 a = I_0 \exp(-C)$, so $-C = \ln(a)$. Thus, the final formula is

$$I(s) = I(0)e^{-\ln(a)s},$$

where $I(s)$ is the intensity of the beam at distance s from the interface. In practice, we reverse-engineer a by eye, because such data is rarely easy to find. The effect of Beer's Law can be seen in Figure 13.3, where the glass takes on a green tint.

To add transparent materials to our code, we need a way to determine when a ray is going "into" an object. The simplest way to do this is to assume that all objects are embedded in air with refractive index very close to 1.0, and that surface normals point "out" (toward the air). The code segment for rays and dielectrics with these assumptions is:

if ($\mathbf{p}$ is on a dielectric) **then**
  $\mathbf{r}$ = reflect($\mathbf{d}$, $\mathbf{n}$)
  if ($\mathbf{d} \cdot \mathbf{n} < 0$) **then**
    refract($\mathbf{d}, \mathbf{n}, n, \mathbf{t}$)
    $c = -\mathbf{d} \cdot \mathbf{n}$
    $k_r = k_g = k_b = 1$

else
$k_r = \exp(-a_r t)$
$k_g = \exp(-a_g t)$
$k_b = \exp(-a_b t)$
if refract$(\mathbf{d}, -\mathbf{n}, 1/n, \mathbf{t})$ **then**
$c = \mathbf{t} \cdot \mathbf{n}$
else
return $k *$ color$(\mathbf{p} + t\mathbf{r})$
$R_0 = (n-1)^2/(n+1)^2$
$R = R_0 + (1 - R_0)(1 - c)^5$
return $k(R$ color$(\mathbf{p} + t\mathbf{r})$ + $(1 - R)$ color$(\mathbf{p} + t\mathbf{t}))$

The code above assumes that the natural log has been folded into the constants (a_r, a_g, a_b). The *refract* function returns false if there is total internal reflection, and otherwise it fills in the last argument of the argument list.

13.2 Instancing

An elegant property of ray tracing is that it allows very natural *instancing*. The basic idea of instancing is to distort all points on an object by a transformation matrix before the object is displayed. For example, if we transform the unit circle (in 2D) by a scale factor $(2, 1)$ in x and y, respectively, then rotate it by $45°$, and move one unit in the x-direction, the result is an ellipse with an eccentricity of 2 and a long axis along the $(x = -y)$-direction centered at $(0, 1)$ (Figure 13.4). The key thing that makes that entity an "instance" is that we store the circle and the composite transform matrix. Thus, the explicit construction of the ellipse is left as a future operation at render time.

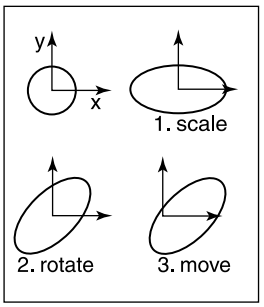

Figure 13.4. An instance of a circle with a series of three transforms is an ellipse.

The advantage of instancing in ray tracing is that we can choose the space in which to do intersection. If the base object is composed of a set of points, one of which is $\mathbf{p}$, then the transformed object is composed of that set of points transformed by matrix $\mathbf{M}$, where the example point is transformed to $\mathbf{Mp}$. If we have a ray $\mathbf{a} + t\mathbf{b}$ that we want to intersect with the transformed object, we can instead intersect an *inverse-transformed ray* with the untransformed object (Figure 13.5). There are two potential advantages to computing in the untransformed space (i.e., the right-hand side of Figure 13.5):

1. the untransformed object may have a simpler intersection routine, e.g., a sphere versus an ellipsoid;