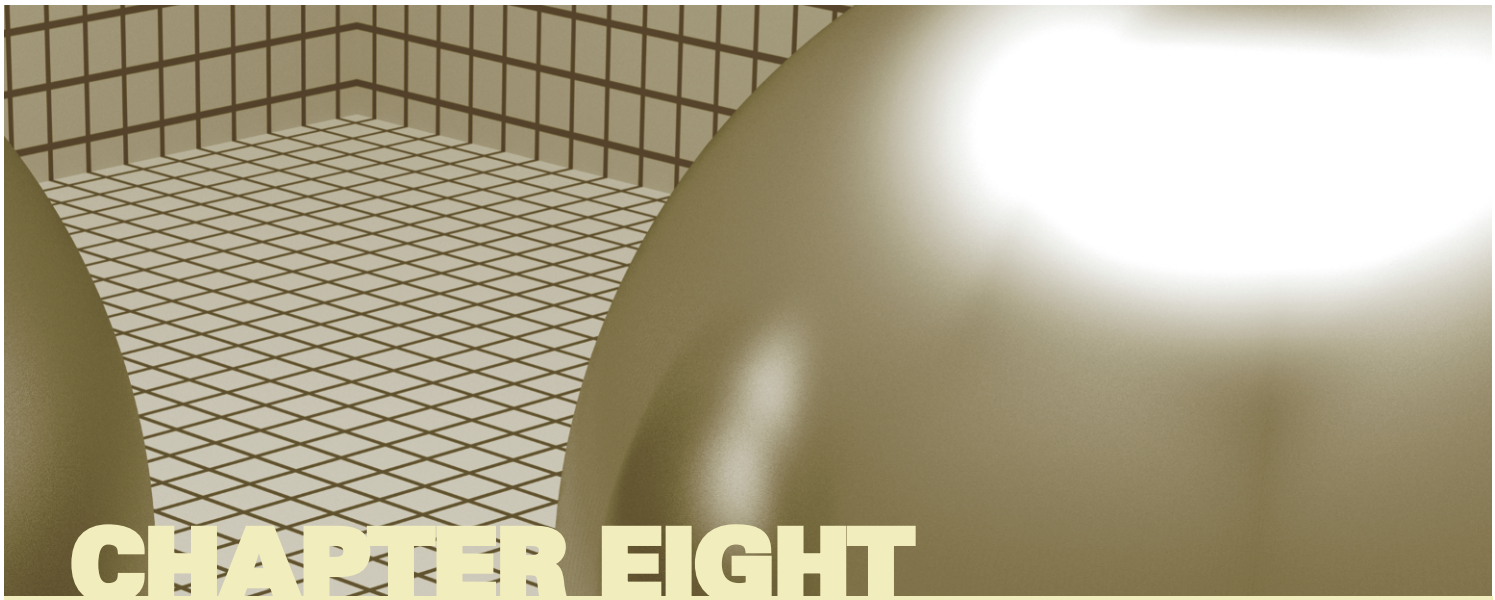


CHAPTER EIGHT

REFLECTION MODELS

This chapter defines a set of classes for describing the way that light scatters at surfaces. Recall that in Section 5.6.1 we introduced the bidirectional reflectance distribution function (BRDF) abstraction to describe light reflection at a surface, the BTDF to describe transmission at a surface, and the BSDF to encompass both of these effects. In this chapter, we will start by defining a generic interface to these surface reflection and transmission functions. Scattering from realistic surfaces is often best described as a mixture of multiple BRDFs and BTDFs; in Chapter 9, we will introduce a BSDF object that combines multiple BRDFs and BTDFs to represent overall scattering from the surface. The current chapter sidesteps the issue of reflection and transmission properties that vary over the surface; the texture classes of Chapter 10 will address that problem. Classes that represent subsurface scattering properties of objects will be introduced in Section 11.6, after some of the related theory is introduced in Chapter 11.

Surface reflection models come from a number of sources:

1. *Measured data:* Reflection distribution properties of many real-world surfaces have been measured in laboratories. This data may be used directly in tabular form or to compute coefficients for a set of basis functions.
2. *Phenomenological models:* Equations that attempt to describe the qualitative properties of real-world surfaces can be remarkably effective at mimicking them. These types of BSDFs can be particularly easy to use, since they tend to have intuitive parameters that modify their behavior (e.g., "roughness"). Many of the reflection functions used in computer graphics fall into this category.
3. *Simulation:* Sometimes, low-level information is known about the composition of a surface. For example, we might know that a paint is comprised of colored particles of some average size suspended in a medium, or that a particular fabric is comprised of two types of thread, each with known reflectance properties. In these

cases, light scattering from the microgeometry can be simulated to generate reflection data. This simulation can be done either during rendering or as a preprocess, after which it may be fit to a set of basis functions for use during rendering.

4. *Physical (wave) optics:* Some reflection models have been derived using a detailed model of light, treating it as a wave and computing the solution to Maxwell's equations to find how it scatters from a surface with known properties. These models tend to be computationally expensive, however, and usually aren't appreciably more accurate than models based on geometric optics.
5. *Geometric optics:* As with simulation approaches, if the surface's low-level scattering and geometric properties are known, then closed-form reflection models can sometimes be derived directly from these descriptions. Geometric optics makes modeling light's interaction with the surface more tractable, since complex wave effects like polarization are ignored.

In this chapter, we will describe implementations of reflection models based on measured data, phenomenological models, and geometric optics. The "Further Reading" section at the end of this chapter gives pointers to a variety of other models.

Before we define the relevant interfaces, a brief review of how they fit into the overall system is in order. In the common case where the `SamplerRenderer` is used, the renderer is first responsible for determining which surface is first visible along a ray. It then calls the integrator classes, defined in Chapter 15, which call the surface shader that is bound to the surface. The surface shader is a method in subclasses of the `Material` class and is responsible for deciding what the BSDF is at a particular point on the surface (see Chapter 9); it returns a `BSDF` object that holds BRDFs and BTDFs that it has allocated and initialized for that point. The integrator then uses the BSDF to compute the scattered light at the point, based on the incoming illumination from the light sources in the scene. (The process where the `MetropolisRenderer` is used rather than the `SamplerRenderer` is similar, but without the use of separate `Integrators`.)

Basic Terminology

In order to be able to compare the visual appearance of different reflection models, we will introduce some basic terminology for describing reflection from surfaces.

Reflection from surfaces can be split into four broad categories: *diffuse*, *glossy specular*, *perfect specular*, and *retro-reflective* (Figure 8.1). Most real surfaces exhibit reflection that is a mixture of these four types. Diffuse surfaces scatter light equally in all directions. Although a perfectly diffuse surface isn't physically realizable, examples of near-diffuse surfaces include dull chalkboards and matte paint. Glossy specular surfaces such as plastic or high-gloss paint scatter light preferentially in a set of reflected directions—they show blurry reflections of other objects. Perfect specular surfaces scatter incident light in a single outgoing direction. Mirrors and glass are examples of perfect specular surfaces. Finally, retro-reflective surfaces like velvet or the Earth's moon scatter light primarily back along the incident direction. Images throughout this chapter will show the differences between these various types of reflection when used in rendered scenes.

Given a particular category of reflection, the reflectance distribution function may be *isotropic* or *anisotropic*. Most objects are isotropic: if you choose a point on the surface and rotate it around its normal axis at that point, the amount of light reflected doesn't

BSDF 478
Integrator 740
Material 483
MetropolisRenderer 852
SamplerRenderer 25

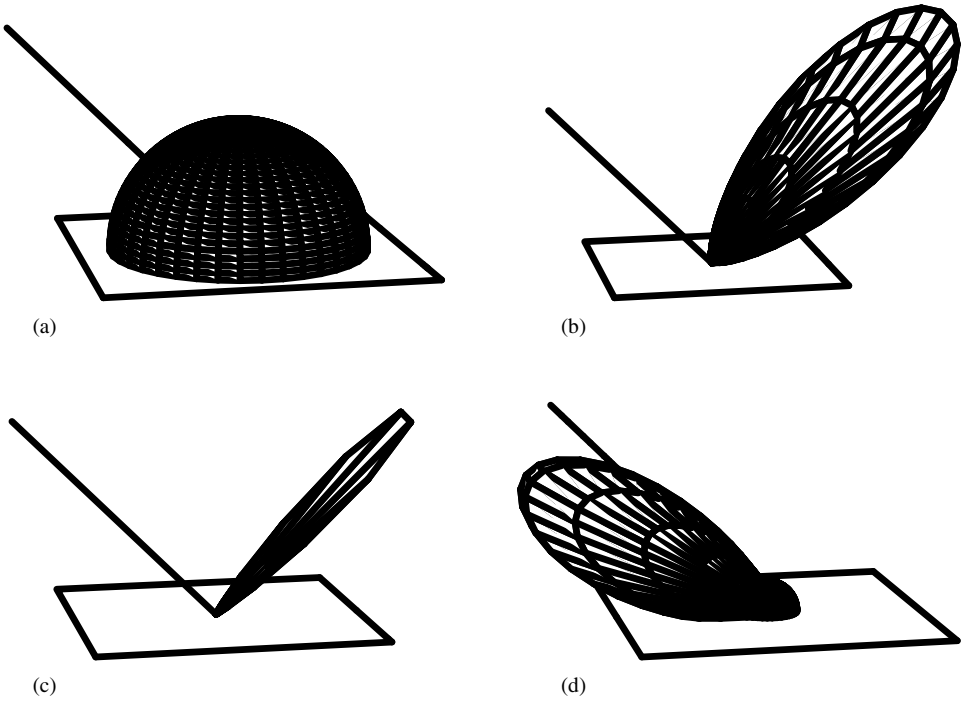


Figure 8.1: Reflection from a surface can be generally categorized by the distribution of reflected light from an incident direction (heavy lines): (a) diffuse, (b) glossy specular, (c) perfect specular, and (d) retro-reflective distributions.

change. In contrast, anisotropic materials reflect different amounts of light as you rotate them in this way. Examples of anisotropic surfaces include brushed metal, phonographic records, and compact disks.

Geometric Setting

Reflection computations in `pbrt` are evaluated in a reflection coordinate system where the two tangent vectors and the normal vector at the point being shaded are aligned with the x, y, and z axes, respectively (Figure 8.2). All direction vectors passed to and returned from the BRDF and BTDF routines will be defined with respect to this coordinate system. It is important to understand this coordinate system in order to understand the BRDF and BTDF implementations in this chapter.

The shading coordinate system also gives a frame for expressing directions in spherical coordinates (θ, ϕ); the angle θ is measured from the given direction to the z axis, and ϕ is the angle formed with the x axis after projection of the direction onto the xy plane. Given a direction vector ω in this coordinate system, it is easy to compute quantities like the cosine of the angle that it forms with the normal direction:

$$\cos\theta = (\mathbf{n} \cdot \omega) = ((0, 0, 1) \cdot \omega) = \omega_z.$$

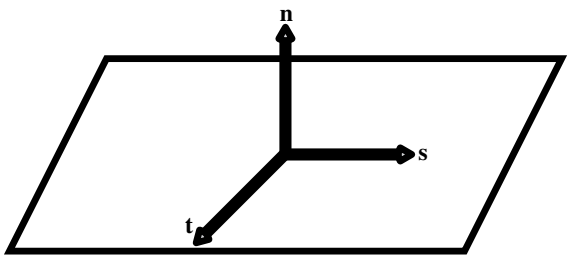


Figure 8.2: The Basic BSDF Interface Setting. The shading coordinate system is defined by the orthonormal basis vectors $(\mathbf{s}, \mathbf{t}, \mathbf{n})$. We will orient these vectors such that they lie along the x, y, and z axes in this coordinate system. Direction vectors ω in world space are transformed into the shading coordinate system before any of the BRDF or BTDF methods are called.

We will provide utility functions to compute this value and its absolute value; their use serves to make the expression of BRDF and BTDF implementations more clear.

⟨*BSDF Inline Functions*⟩ ≡
```
inline float CosTheta(const Vector &w) { return w.z; }
inline float AbsCosTheta(const Vector &w) { return fabsf(w.z); }
```

The value of $\sin^2\theta$ can be computed using the trigonometric identity $\sin^2\theta + \cos^2\theta = 1$.

⟨*BSDF Inline Functions*⟩ +≡
```
inline float SinTheta2(const Vector &w) {
    return max(0.f, 1.f - CosTheta(w)*CosTheta(w));
}
```

⟨*BSDF Inline Functions*⟩ +≡
```
inline float SinTheta(const Vector &w) {
    return sqrtf(SinTheta2(w));
}
```

We can similarly use the shading coordinate system to simplify the calculations for the sine and cosine of the ϕ angle (Figure 8.3). In the plane of the point being shaded, the vector ω has coordinates (x, y), which are given by $r\cos\phi$ and $r\sin\phi$, respectively. The radius r is $\sin\theta$, so

$$\cos\phi = \frac{x}{r} = \frac{x}{\sin\theta}$$
$$\sin\phi = \frac{y}{r} = \frac{y}{\sin\theta}.$$

CosTheta() 426
Vector 57

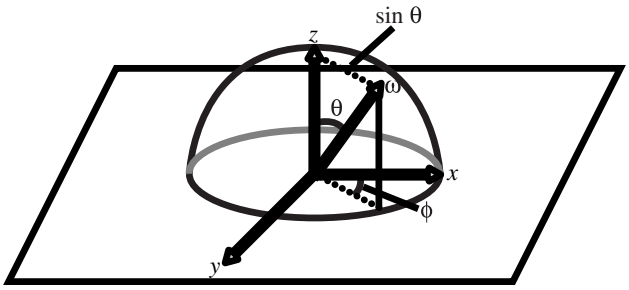


Figure 8.3: The value of $\sin\theta$ can be found by computing the length of the dotted line, which is the magnitude of the xy components of the vector. (Recall that the length of the vector ω is one.) The $\sin\phi$ and $\cos\phi$ values can be computed using the circular coordinate equations $x = r\cos\phi$ and $y = r\sin\phi$, where r, the length of the dashed line, is equal to $\sin\theta$.

⟨*BSDF Inline Functions*⟩ +≡

```
inline float CosPhi(const Vector &w) {
    float sintheta = SinTheta(w);
    if (sintheta == 0.f) return 1.f;
    return Clamp(w.x / sintheta, -1.f, 1.f);
}
inline float SinPhi(const Vector &w) {
    float sintheta = SinTheta(w);
    if (sintheta == 0.f) return 0.f;
    return Clamp(w.y / sintheta, -1.f, 1.f);
}
```

Another convention we will follow is that the incident light direction ω_i and the outgoing viewing direction ω_o will both be normalized and outward facing after being transformed into the local coordinate system at the surface. By convention, the surface normal $\mathbf{n}$ always points to the "outside" of the object, which makes it easy to determine if light is entering or exiting transmissive objects: if the incident light direction ω_i is in the same hemisphere as $\mathbf{n}$, then light is entering; otherwise, it is exiting.

Therefore, one detail to keep in mind is that the normal may be on the opposite side of the surface than one or both of the ω_i and ω_o direction vectors. Unlike many other renderers, `pbrt` does not flip the normal to lie on the same side as ω_o. Therefore, it is important that BRDFs and BTDFs be implemented so that they don't expect otherwise.

Furthermore, note that the local coordinate system used for shading may not be exactly the same as the coordinate system returned by the `Shape::Intersect()` routines from Chapter 3; they can be modified between intersection and shading to achieve effects like bump mapping. See Chapter 9 for examples of this kind of modification.

Clamp() 1000
Shape::Intersect() 111
SinTheta() 426
Vector 57

One final detail to be aware of when reading this chapter is that BRDF and BTDF implementations should not concern themselves with whether ω_i and ω_o lie in the same hemisphere. For example, although a reflective BRDF should in principle detect if the incident direction is above the surface and the outgoing direction is below and always return no reflection in this case, here we will expect the reflection function to instead compute and return the amount of light reflected using the appropriate formulas for their reflection model, ignoring the detail that they are not in the same hemisphere. Higher-level code in `pbrt` will ensure that only reflective or transmissive scattering routines are evaluated as appropriate. The value of this convention will be explained in Section 9.1.

8.1 BASIC INTERFACE

We will first define the interface for the individual BRDF and BTDF functions. BRDFs and BTDFs share a common base class, `BxDF`. Because both have the exact same interface, sharing the same base class reduces repeated code and allows some parts of the system to work with `BxDF`s generically without distinguishing between BRDFs and BTDFs.

⟨*BxDF Declarations*⟩ ≡
```
class BxDF {
public:
    ⟨BxDF Interface 429⟩
    ⟨BxDF Public Data 429⟩
};
```

The `BSDF` class, which will be introduced in Section 9.1, holds a collection of `BxDF` objects that together describe the scattering at a point on a surface. Although we are hiding the implementation details of the `BxDF` behind a common interface for reflective and transmissive materials, some of the light transport algorithms in Chapter 15 will need to distinguish between these two types. Therefore, all `BxDF`s have a `BxDF::type` member that holds flags from `BxDFType`. For each `BxDF`, the flags should have exactly one of `BSDF_REFLECTION` or `BSDF_TRANSMISSION` set, and exactly one of the diffuse, glossy, and specular flags. Note that there is no retro-reflective flag; retro-reflection is treated as glossy reflection in this categorization.

⟨*BSDF Declarations*⟩ ≡
```
enum BxDFType {
    BSDF_REFLECTION   = 1<<0,
    BSDF_TRANSMISSION = 1<<1,
    BSDF_DIFFUSE      = 1<<2,
    BSDF_GLOSSY       = 1<<3,
    BSDF_SPECULAR     = 1<<4,
    BSDF_ALL_TYPES        = BSDF_DIFFUSE |
                            BSDF_GLOSSY |
                            BSDF_SPECULAR,
    BSDF_ALL_REFLECTION   = BSDF_REFLECTION |
                            BSDF_ALL_TYPES,
    BSDF_ALL_TRANSMISSION = BSDF_TRANSMISSION |
                            BSDF_ALL_TYPES,
```

BSDF 478
BSDF_ALL_REFLECTION 428
BSDF_ALL_TRANSMISSION 428
BSDF_ALL_TYPES 428
BSDF_DIFFUSE 428
BSDF_GLOSSY 428
BSDF_REFLECTION 428
BSDF_SPECULAR 428
BSDF_TRANSMISSION 428
BxDF 428
BxDF::type 429
BxDFType 428

```
    BSDF_ALL              = BSDF_ALL_REFLECTION |
                            BSDF_ALL_TRANSMISSION
};
```

⟨*BxDF Public Data*⟩ ≡ 428
```
const BxDFType type;
```

⟨*BxDF Interface*⟩ ≡ 428
```
BxDF(BxDFType t) : type(t) { }
```

The `MatchesFlags()` utility method determines if the `BxDF` matches the user-supplied flags:

⟨*BxDF Interface*⟩ +≡ 428
```
bool MatchesFlags(BxDFType flags) const {
    return (type & flags) == type;
}
```

The key method that `BxDF`s provide is the `BxDF::f()` method. It returns the value of the distribution function for the given pair of directions. This interface implicitly assumes that light in different wavelengths is *decoupled*—energy at one wavelength will not be reflected at a different wavelength. By making this assumption, the effect of the reflection function can be represented directly with a `Spectrum`. To support fluorescent materials where this assumption is not true would require that this method return an $n \times n$ matrix that encoded the transfer of energy between spectral samples (where n is the number of samples in the `Spectrum` representation).

⟨*BxDF Interface*⟩ +≡ 428
```
virtual Spectrum f(const Vector &wo, const Vector &wi) const = 0;
```

Not all `BxDF`s can be evaluated with the `f()` method. For example, perfectly specular objects like a mirror, glass, or water only scatter light from a single incident direction into a single outgoing direction. Such `BxDF`s are best described with delta distributions that are zero except for the single direction where light is scattered.

These `BxDF`s need special handling in `pbrt`, so we will also provide the method `BxDF::Sample_f()`. This method is used both for handling scattering that is described by delta distributions as well as for randomly sampling directions from `BxDF`s that scatter light along multiple directions; this second application will be explained in the discussion of Monte Carlo sampling in Chapter 14. `BxDF::Sample_f()` computes the direction of incident light ω_i given an outgoing direction ω_o and returns the value of the `BxDF` for the given pair of directions. For delta distributions, it is necessary for the `BxDF` to choose the incident light direction in this way, since the caller has no chance of generating the appropriate ω_i direction.[1] The `u1`, `u2`, and `pdf` parameters aren't needed for delta distribution `BxDF`s, so they will be explained later, in Section 14.5, when we provide implementations of this method for nonspecular reflection functions.

`BxDF` 428
`BxDF::f()` 429
`BxDF::Sample_f()` 694
`BxDF::type` 429
`BxDFType` 428
`Spectrum` 263
`Vector` 57

1 Delta distributions in reflection functions have some additional subtle implications for light transport algorithms. Sections 14.5.4 and 15.2.5 describe the issues in detail.

⟨*BxDF Interface*⟩ +≡ 428
```
virtual Spectrum Sample_f(const Vector &wo, Vector *wi,
                          float u1, float u2, float *pdf) const;
```

8.1.1 REFLECTANCE

It can be useful to take the aggregate behavior of the 4D BRDF or BTDF, defined as a function over pairs of directions, and reduce it to a 2D function over a single direction, or even to a constant value that describes its overall scattering behavior.

The *hemispherical-directional reflectance* is a 2D function that gives the total reflection in a given direction due to constant illumination over the hemisphere, or, equivalently, total reflection over the hemisphere due to light from a given direction.[2] It is defined as

$$\rho_{\mathrm{hd}}(\omega_{\mathrm{o}}) = \int_{\mathcal{H}^2(\mathbf{n})} f_{\mathrm{r}}(\mathrm{p}, \omega_{\mathrm{o}}, \omega_{\mathrm{i}}) \, |\cos\theta_{\mathrm{i}}| \, \mathrm{d}\omega_{\mathrm{i}}.$$

The `BxDF::rho()` method computes the reflectance function ρ_{hd}. Some `BxDF`s can compute this value in closed form, although most use Monte Carlo integration to compute an approximation to it. For those `BxDF`s, the `nSamples` and `samples` parameters are used by the implementation of the Monte Carlo algorithm; they are explained in Section 14.5.5.

⟨*BxDF Interface*⟩ +≡ 428
```
virtual Spectrum rho(const Vector &wo, int nSamples,
                     const float *samples) const;
```

The *hemispherical-hemispherical reflectance* of a surface, denoted by ρ_{hh}, is a constant spectral value that gives the fraction of incident light reflected by a surface when the incident light is the same from all directions. It is

$$\rho_{\mathrm{hh}} = \frac{1}{\pi} \int_{\mathcal{H}^2(\mathbf{n})} \int_{\mathcal{H}^2(\mathbf{n})} f_{\mathrm{r}}(\mathrm{p}, \omega_{\mathrm{o}}, \omega_{\mathrm{i}}) \, |\cos\theta_{\mathrm{o}} \cos\theta_{\mathrm{i}}| \, \mathrm{d}\omega_{\mathrm{o}} \mathrm{d}\omega_{\mathrm{i}}.$$

The `BxDF::rho()` method computes ρ_{hh} if no direction ω_{o} is provided. The remaining parameters are again used when computing a Monte Carlo estimate, if needed.

⟨*BxDF Interface*⟩ +≡ 428
```
virtual Spectrum rho(int nSamples, const float *samples1,
                     const float *samples2) const;
```

8.1.2 BRDF → BTDF ADAPTER

It's handy to define an adapter class that makes it easy to reuse an already-defined BRDF class as a BTDF, especially for phenomenological models that may be equally plausible models of transmission. The `BRDFToBTDF` class takes a BRDF's pointer in the constructor and uses it to implement a BTDF. In particular, doing so involves forwarding method calls on to the BRDF and switching the ω_{i} direction to lie in the other hemisphere.

BRDFToBTDF 431
BxDF 428
BxDF::rho() 430
Spectrum 263
Vector 57

2 The fact that these two quantities are equal is due to the reciprocity of real-world reflection functions. If we had a nonphysically based BRDF that did not obey reciprocity, this assumption, along with many others in `pbrt`, would break down.

⟨*BxDF Declarations*⟩ +≡
```
class BRDFToBTDF : public BxDF {
public:
    ⟨BRDFToBTDF Public Methods 431⟩
private:
    BxDF *brdf;
};
```

The constructor for the adapter class is simple. It switches the reflection and transmission flags of the `BxDF::type` member.

⟨*BRDFToBTDF Public Methods*⟩ ≡ 431
```
BRDFToBTDF(BxDF *b)
    : BxDF(BxDFType(b->type ^ (BSDF_REFLECTION | BSDF_TRANSMISSION))) {
    brdf = b;
}
```

The adapter needs to convert an incoming vector to the corresponding vector in the opposite hemisphere. Fortunately, this is a simple calculation in the shading coordinate system, just requiring negation of the vector's z coordinate.

⟨*BRDFToBTDF Public Methods*⟩ +≡ 431
```
static Vector otherHemisphere(const Vector &w) {
    return Vector(w.x, w.y, -w.z);
}
```

The `BRDFToBTDF::otherHemisphere()` method is used to reflect a ray into the other hemisphere before calling the BRDF's `BxDF::f()`, `BxDF::rho()`, and `BxDF::Sample_f()` methods. We'll only include `f()` here since the others are analogous.

⟨*BxDF Method Definitions*⟩ ≡
```
Spectrum BRDFToBTDF::f(const Vector &wo, const Vector &wi) const {
    return brdf->f(wo, otherHemisphere(wi));
}
```

BRDFToBTDF 431
BRDFToBTDF::brdf 431
BRDFToBTDF::otherHemisphere() 431
BSDF 478
BSDF_REFLECTION 428
BSDF_TRANSMISSION 428
BxDF 428
BxDF::f() 429
BxDF::rho() 430
BxDF::Sample_f() 694
BxDF::type 429
BxDFType 428
MixMaterial 488
ScaledBxDF 431
Spectrum 263
Vector 57

8.1.3 BxDF SCALING ADAPTER

It is also useful to take a given `BxDF` and scale its contribution with a `Spectrum` value. The `ScaledBxDF` wrapper holds a `BxDF *` and a `Spectrum` and implements this functionality. This class is used by the `MixMaterial` (defined in Section 9.2.3), which creates `BSDF`s based on a weighted combination of two other materials.

⟨*BxDF Declarations*⟩ +≡
```
class ScaledBxDF : public BxDF {
public:
    ⟨ScaledBxDF Public Methods 432⟩
private:
    BxDF *bxdf;
    Spectrum s;
};
```

⟨*ScaledBxDF Public Methods*⟩ ≡ 431
```
ScaledBxDF(BxDF *b, const Spectrum &sc)
    : BxDF(BxDFType(b->type)), bxdf(b), s(sc) {
}
```

The implementations of the `ScaledBxDF` methods are straightforward; we'll only include `f()` here.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum ScaledBxDF::f(const Vector &wo, const Vector &wi) const {
    return s * bxdf->f(wo, wi);
}
```

8.2 SPECULAR REFLECTION AND TRANSMISSION

The behavior of light at perfectly smooth surfaces is relatively easy to characterize analytically using both the physical and geometric optics models. These surfaces exhibit perfect specular reflection and transmission of incident light; for a given ω_i direction, all light is scattered in a single outgoing direction ω_o. For specular reflection, this direction is the outgoing direction that makes the same angle with the normal as the incoming direction:

$$\theta_i = \theta_o.$$

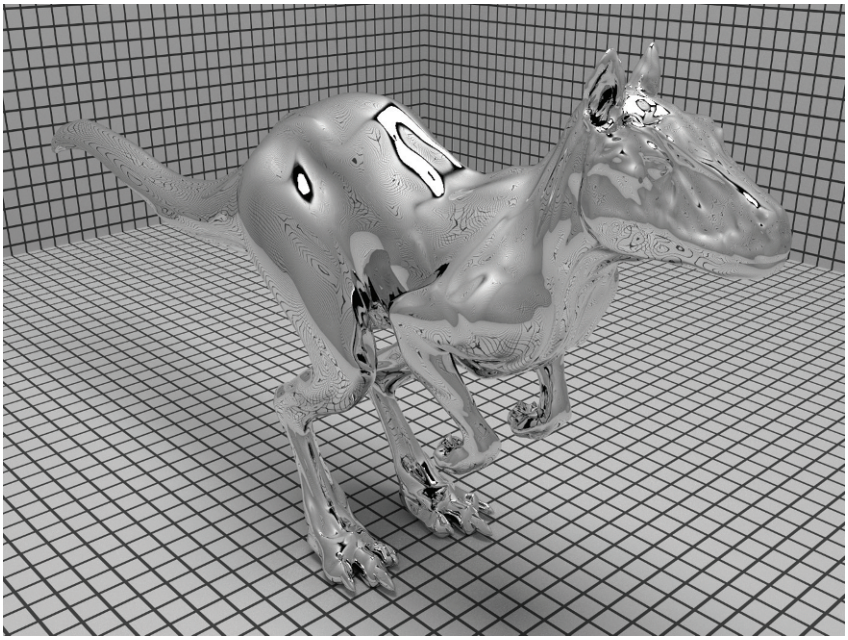
For transmission, the outgoing direction is given by *Snell's law*, which relates the angle θ_t between the transmitted direction and the surface normal $\mathbf{n}$ to the angle θ_i between the incident ray and the surface normal $\mathbf{n}$. (One of the exercises at the end of this chapter is to derive Snell's law using Fermat's principle from optics.) Snell's law is based on the *index of refraction* for the medium that the incident ray is in and the index of refraction for the medium it is entering. The index of refraction describes how much more slowly light travels in a particular medium than in a vacuum. We will use the Greek letter η, pronounced "eta," to denote the index of refraction. Snell's law is

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t.$$

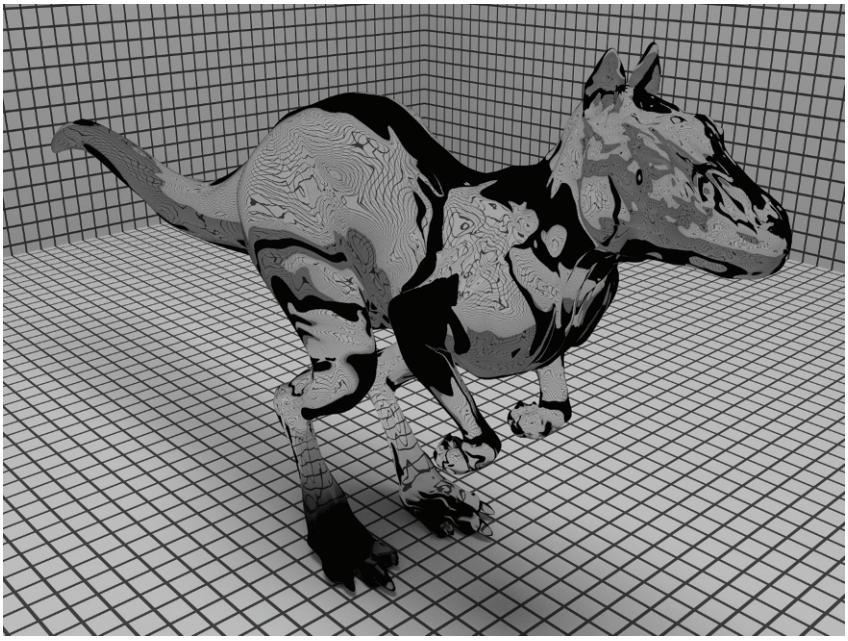
In general, the index of refraction varies with the wavelength of light. Thus, incident light generally scatters in multiple directions at the boundary between two different media, an effect known as *dispersion*. This effect can be seen when incident white light is split into spectral components by a prism. Common practice in graphics is to ignore this wavelength dependence, since this effect is generally not crucial for visual accuracy and ignoring it simplifies light transport calculations substantially. Alternatively, the paths of multiple beams of light (e.g., at a series of discrete wavelengths) can be tracked through the environment in which a dispersive object is found. The "Further Reading" section at the end of this chapter has pointers to more information on these issues.

Figure 8.4 shows the effect when the Killeroo model is rendered with a BRDF describing perfect specular reflection and a BTDF describing specular transmission. Note how refraction through the transmissive object distorts the scene behind it.

BxDF 428
BxDF::f() 429
BxDFType 428
ScaledBxDF 431
ScaledBxDF::bxdf 431
ScaledBxDF::s 431
Spectrum 263
Vector 57



(a)



(b)

Figure 8.4: Killeroo model rendered with (a) perfect specular reflection and (b) perfect specular refraction. *(Model courtesy of headus/Rezard.)*

Table 8.1: Indices of refraction for a variety of objects, giving the ratio of the speed of light in a vacuum to the speed of light in the medium. These are generally wavelength-dependent quantities; these values are averages over the visible wavelengths.

Medium	Index of refraction η
Vacuum	1.0
Air at sea level	1.00029
Ice	1.31
Water (20°C)	1.333
Fused quartz	1.46
Glass	1.5–1.6
Sapphire	1.77
Diamond	2.42

8.2.1 FRESNEL REFLECTANCE

In addition to the reflected and transmitted directions, it is also necessary to compute the fraction of incoming light that is reflected or transmitted. In simple ray tracers, these fractions are typically just given as "reflectivity" or "transmissiveness" values, which are uniform over the entire surface. For physical reflection or refraction, however, these terms are directionally dependent and cannot be captured by constant per-surface scaling amounts. The *Fresnel equations* describe the amount of light reflected from a surface; they are the solution to Maxwell's equations at smooth surfaces.

There are two sets of Fresnel equations: one for *dielectric media* (objects that don't conduct electricity, like glass) and one for *conductors* (like metals). For each of these cases, the Fresnel equations have two forms, depending on the polarization of the incident light. Properly accounting for polarization in rendering is a complex task, and so in `pbrt` we will make the common assumption that light is unpolarized; that is, it is randomly oriented with respect to the light wave. With this simplifying assumption, the Fresnel reflectance is the average of the squares of the parallel and perpendicular polarization terms.

To compute the Fresnel reflectance of a dielectric, we need to know the indices of refraction for the two media. Table 8.1 has the indices of refraction for a number of dielectric materials. A close approximation to the Fresnel reflectance formulae for dielectrics is

$$r_{\parallel} = \frac{\eta_t \cos\theta_i - \eta_i \cos\theta_t}{\eta_t \cos\theta_i + \eta_i \cos\theta_t}$$
$$r_{\perp} = \frac{\eta_i \cos\theta_i - \eta_t \cos\theta_t}{\eta_i \cos\theta_i + \eta_t \cos\theta_t},$$

where $r_{\parallel}$ is the Fresnel reflectance for parallel polarized light and $r_{\perp}$ is the reflectance for perpendicular polarized light. η_i and η_t are the indices of refraction for the incident and transmitted media, and ω_i and ω_t are the incident and transmitted directions, where ω_t was computed with Snell's law.

The cosine terms should all be greater than or equal to zero; for the purposes of computing these values, the geometric normal should be flipped to be on the same side as ω_i and then ω_t.

For unpolarized light, the Fresnel reflectance is

$$F_r = \frac{1}{2}(r_\parallel^2 + r_\perp^2).$$

The function FrDiel() computes the Fresnel reflection formula for dielectric materials and circularly polarized light. The quantities $\cos\theta_i$ and $\cos\theta_t$ are passed in with the variables cosi and cost.

⟨*BxDF Utility Functions*⟩ ≡
```
Spectrum FrDiel(float cosi, float cost, const Spectrum &etai,
                const Spectrum &etat) {
    Spectrum Rparl = ((etat * cosi) - (etai * cost)) /
                     ((etat * cosi) + (etai * cost));
    Spectrum Rperp = ((etai * cosi) - (etat * cost)) /
                     ((etai * cosi) + (etat * cost));
    return (Rparl*Rparl + Rperp*Rperp) / 2.f;
}
```

Due to conservation of energy, the energy transmitted by a dielectric is $1 - F_r$.

Unlike dielectrics, conductors don't transmit light, but some of the incident light is absorbed by the material and turned into heat. The Fresnel formula for conductors tells how much is reflected—the amount depends on the additional quantities η, the index of refraction of the conductor, and k, its *absorption coefficient*. Figure 8.5 shows a plot of the index of refraction and absorption coefficent for gold; both of these are wavelength-dependent quantities. The directory scenes/spds/metals in the pbrt distribution has wavelength-dependent data for η and k for a variety of metals. Figure 9.4 in the next chapter shows a model rendered with a metal material.

A widely used approximation to the Fresnel reflectance for conductors is

$$r_\parallel^2 = \frac{(\eta^2 + k^2)\cos\theta_i^2 - 2\eta\cos\theta_i + 1}{(\eta^2 + k^2)\cos\theta_i^2 + 2\eta\cos\theta_i + 1} \tag{8.1}$$

$$r_\perp^2 = \frac{(\eta^2 + k^2) - 2\eta\cos\theta_i + \cos\theta_i^2}{(\eta^2 + k^2) + 2\eta\cos\theta_i + \cos\theta_i^2}. \tag{8.2}$$

⟨*BxDF Utility Functions*⟩ +≡
```
Spectrum FrCond(float cosi, const Spectrum &eta, const Spectrum &k) {
    Spectrum tmp = (eta*eta + k*k) * cosi*cosi;
    Spectrum Rparl2 = (tmp - (2.f * eta * cosi) + 1) /
                      (tmp + (2.f * eta * cosi) + 1);
    Spectrum tmp_f = eta*eta + k*k;
    Spectrum Rperp2 =
        (tmp_f - (2.f * eta * cosi) + cosi*cosi) /
        (tmp_f + (2.f * eta * cosi) + cosi*cosi);
    return (Rparl2 + Rperp2) / 2.f;
}
```

FrDiel() 435
Spectrum 263

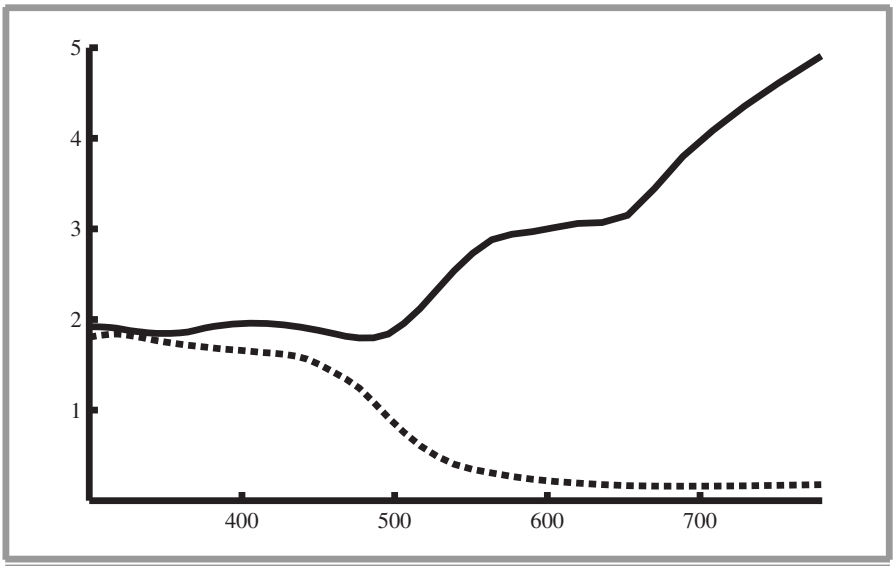


Figure 8.5: Absorption Coefficient and Index of Refraction of Gold. This plot shows the spectrally varying values of the absorption coefficient k (solid line) and the index of refraction η (dashed line) for gold, where the horizontal axis is wavelength in nm.

For convenience, we will define an abstract `Fresnel` class that provides an interface for computing Fresnel reflection coefficients. The `FresnelConductor` and `FresnelDielectric` implementations of this interface help simplify the implementation of subsequent BRDFs that may need to support both forms.

```
⟨BxDF Declarations⟩ +≡
    class Fresnel {
    public:
        ⟨Fresnel Interface 436⟩
    };
```

The only method provided by the `Fresnel` interface is `Fresnel::Evaluate()`. Given the cosine of the angle made by the incoming direction and the surface normal, it returns the amount of light reflected by the surface.

```
⟨Fresnel Interface⟩ ≡                                              436
    virtual Spectrum Evaluate(float cosi) const = 0;
```

Fresnel Conductors

`FresnelConductor` implements this interface for conductors.

```
⟨BxDF Declarations⟩ +≡
    class FresnelConductor : public Fresnel {
    public:
        ⟨FresnelConductor Public Methods 437⟩
```

Fresnel 436
Fresnel::Evaluate() 436
FresnelConductor 436
FresnelDielectric 437
Spectrum 263

```
private:
    Spectrum eta, k;
};
```

Its constructor stores the given index of refraction η and absorption coefficient k.

⟨*FresnelConductor Public Methods*⟩ ≡ 436
```
FresnelConductor(const Spectrum &e, const Spectrum &kk)
    : eta(e), k(kk) {
}
```

The evaluation routine for `FresnelConductor` is also simple; it just calls the `FrCond()` function defined earlier. Note that it takes the absolute value of `cosi` before calling `FrCond()`, since `FrCond()` expects that the cosine will be measured with respect to the normal on the same side of the surface as ω_i, or equivalently that the absolute value of $\cos\theta_i$ should be used.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum FresnelConductor::Evaluate(float cosi) const {
    return FrCond(fabsf(cosi), eta, k);
}
```

Fresnel Dielectrics

`FresnelDielectric` similarly implements the `Fresnel` interface for dielectric materials.

⟨*BxDF Declarations*⟩ +≡
```
class FresnelDielectric : public Fresnel {
public:
    ⟨FresnelDielectric Public Methods 437⟩
private:
    float eta_i, eta_t;
};
```

Its constructor stores the indices of refraction on the two sides of the surface, η_i and η_t:

⟨*FresnelDielectric Public Methods*⟩ ≡ 437
```
FresnelDielectric(float ei, float et) : eta_i(ei), eta_t(et) { }
```

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum FresnelDielectric::Evaluate(float cosi) const {
    ⟨Compute Fresnel reflectance for dielectric 438⟩
}
```

FrCond() 435
Fresnel 436
FresnelConductor 436
FresnelDielectric 437
FresnelDielectric::eta_i 437
FresnelDielectric::eta_t 437
Spectrum 263

Evaluating the Fresnel formula for dielectric media is a bit more complicated than for conductors. First, it is necessary to determine if the incident direction is on the outside of the medium or inside it, so that the two indices of refraction can be interpreted appropriately. Next, Snell's law is applied to compute the sine of the angle between the transmitted direction and the surface normal. Finally, the cosine of this angle is found using the identity $\sin^2\theta + \cos^2\theta = 1$.

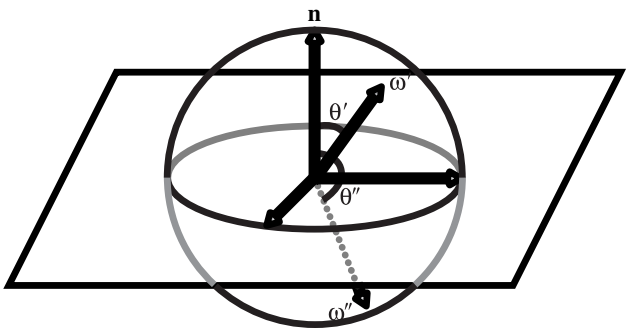


Figure 8.6: The cosine of the angle θ between a direction ω and the geometric surface normal indicates whether the direction is pointing outside the surface (in the same hemisphere as the normal) or inside the surface. In the standard reflection coordinate system, this test just requires checking the z component of the direction vector. Here, ω' is in the upper hemisphere, with a positive-valued cosine, while ω'' is in the lower hemisphere.

⟨*Compute Fresnel reflectance for dielectric*⟩ ≡ 437
```
cosi = Clamp(cosi, -1.f, 1.f);
⟨Compute indices of refraction for dielectric 438⟩
⟨Compute sint using Snell's law 438⟩
if (sint >= 1.) {
    ⟨Handle total internal reflection 439⟩
}
else {
    float cost = sqrtf(max(0.f, 1.f - sint*sint));
    return FrDiel(fabsf(cosi), cost, ei, et);
}
```

The sign of the cosine of the incident angle indicates on which side of the medium the incident ray lies (Figure 8.6). If the cosine is between 0 and 1, the ray is on the outside, and if the cosine is between −1 and 0, the ray is on the inside. The variables `ei` and `et` are set such that `ei` has the index of refraction of the incident medium.

⟨*Compute indices of refraction for dielectric*⟩ ≡ 438
```
bool entering = cosi > 0.;
float ei = eta_i, et = eta_t;
if (!entering)
    swap(ei, et);
```

Once the indices of refraction are set, we can compute $\sin\theta_t$ using Snell's law:

⟨*Compute* `sint` *using Snell's law*⟩ ≡ 438
```
float sint = ei/et * sqrtf(max(0.f, 1.f - cosi*cosi));
```

When light is traveling from one medium to another medium with a lower index of refraction, none of the light at incident angles near grazing passes into the other medium.

Clamp() 1000
FrDiel() 435

The largest angle at which this happens is called the *critical angle*; when θ_i is greater than the critical angle, *total internal reflection* occurs, and all of the light is reflected. That case is detected here by a value of $\sin \theta_t$ greater than one; in that case, the Fresnel equations are unnecessary.

⟨*Handle total internal reflection*⟩ ≡ 438
```
return 1.;
```

A Special Fresnel Interface

The `FresnelNoOp` implementation of the `Fresnel` interface returns 100% reflection for all incoming directions. Although this is physically implausible, it is a convenient capability to have available.

⟨*BxDF Declarations*⟩ +≡
```
class FresnelNoOp : public Fresnel {
public:
    Spectrum Evaluate(float) const { return Spectrum(1.); }
};
```

8.2.2 SPECULAR REFLECTION

We can now implement the `SpecularReflection` class, which describes physically plausible specular reflection, using the Fresnel interface to compute the fraction of light that is reflected. First, we will derive the BRDF that describes specular reflection. Since the Fresnel equations give the fraction of light reflected, $F_r(\omega_i)$, then we need a BRDF such that

$$L_o(\omega_o) = f_r(\omega_o, \omega_i) L_i(\omega_i) = F_r(\omega_i)\, L_i(\omega_i),$$

where ω_i is the reflection vector for ω_o about the surface normal. (Recall that $\theta_i = \theta_o$ for specular reflection, and therefore $F_r(\omega_o) = F_r(\omega_i)$.)

Such a BRDF can be constructed using the Dirac delta distribution. Recall from Section 7.1 that the delta distribution has the useful property that

$$\int f(x)\, \delta(x - x_0)\, \mathrm{d}x = f(x_0). \qquad [8.3]$$

The delta distribution requires special handling compared to standard functions. In particular, integrals with delta distributions must be evaluated by explicitly accounting for the delta distribution; their values cannot be properly computed without doing so. For example, consider the delta distribution in Equation (8.3): if we tried to evaluate it using the trapezoid rule or some other numerical integration technique, by definition of the delta distribution there would be zero probability that any of the evaluation points x_i would have a nonzero value of $\delta(x_i)$. Rather, we must allow the delta distribution to determine the evaluation point itself. We will see this issue in practice for specular `BxDF`s as well as for some of the light sources in Chapter 12.

Intuitively, we want the BRDF to be zero everywhere except at the perfect reflection direction, which suggests the use of the delta distribution. A first guess might be to use

BxDF 428
Fresnel 436
FresnelNoOp 439
Spectrum 263
SpecularReflection 440

delta functions to restrict the incident direction to the reflection angle ω_r. This would yield a BRDF of

$$f_r(p, \omega_o, \omega_i) = \delta(\omega_i - \omega_r) = \delta(\cos\theta_i - \cos\theta_r)\delta(\phi_i - \phi_r).$$

Although this seems appealing, plugging into the scattering equation, Equation (5.8), reveals a problem:

$$\begin{aligned} L_o\left(\theta_o, \phi_o\right) &= \int_{S^2} \delta(\cos\theta_i - \cos\theta_r)\delta(\phi_o - \phi_r \pm \pi) L_i\left(\theta_i, \phi_i\right) |\cos\theta_i| \, d\omega_i \\ &= L_i\left(\theta_r, \phi_r \pm \pi\right) |\cos\theta_i|. \end{aligned}$$

This is not correct because it contains an extra factor of $\cos\theta_i$. But we can divide out this factor to find the correct BRDF for perfect specular reflection:

$$f_r(p, \omega_o, \omega_i) = F_r(\omega_o) \frac{\delta(\omega_i - R(\omega_o, \mathbf{n}))}{|\cos\theta_i|},$$

where $R(\omega_o, \mathbf{n})$ is the specular reflection vector for ω_o reflected about the surface normal $\mathbf{n}$.

⟨*BxDF Declarations*⟩ +≡
```
class SpecularReflection : public BxDF {
public:
    ⟨SpecularReflection Public Methods 440⟩
private:
    ⟨SpecularReflection Private Data 440⟩
};
```

The `SpecularReflection` class takes a `Fresnel` object to describe dielectric or conductor Fresnel properties and an additional `Spectrum` object, which is used to scale the reflected color.

⟨*SpecularReflection Public Methods*⟩ ≡ 440
```
SpecularReflection(const Spectrum &r, Fresnel *f)
    : BxDF(BxDFType(BSDF_REFLECTION | BSDF_SPECULAR)),
      R(r), fresnel(f) {
}
```

⟨*SpecularReflection Private Data*⟩ ≡ 440
```
Spectrum R;
Fresnel *fresnel;
```

The rest of the implementation is straightforward. No scattering is returned from `SpecularReflection::f()`, since for an arbitrary pair of directions the delta function returns no scattering.[3]

3 If the caller happened to pass a vector and its perfect mirror direction, this function still returns zero. Although this might be a slightly confusing interface to these reflection functions, we still get the correct result in the end because reflection functions involving singularities with delta distributions receive special handling by the light transport routines (see Chapter 15).

BSDF_REFLECTION 428
BSDF_SPECULAR 428
BxDF 428
BxDFType 428
Fresnel 436
Spectrum 263
SpecularReflection 440
SpecularReflection::f() 441

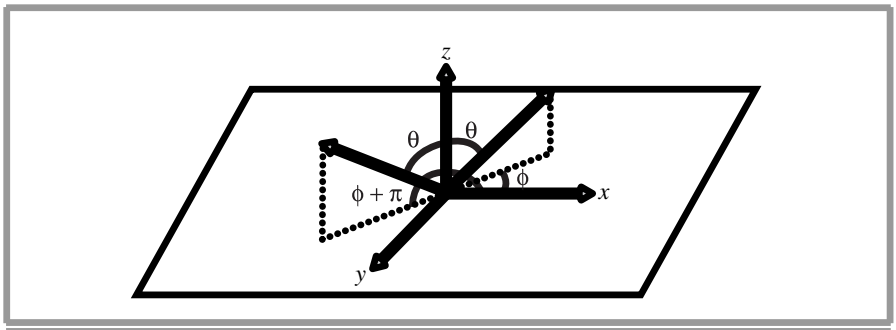


Figure 8.7: Given an incident direction that makes an angle θ with the surface normal and an angle ϕ with the x axis, the reflected ray about the normal makes an angle θ with the normal and $\phi + \pi$ with the x axis. The (x, y, z) coordinates of this direction can be found by scaling the incident direction by $(-1, -1, 1)$.

⟨*SpecularReflection Public Methods*⟩ +≡ 440
```
Spectrum f(const Vector &, const Vector &) const {
    return Spectrum(0.);
}
```

However, we do implement the `Sample_f()` method, which selects an appropriate direction according to the delta function. It sets the output variable `wi` to be the reflection of the supplied direction `wo` about the surface normal. The `*pdf` value is set to be one, which is the appropriate value for this case, where no Monte Carlo sampling is being done.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum SpecularReflection::Sample_f(const Vector &wo,
        Vector *wi, float u1, float u2, float *pdf) const {
    ⟨Compute perfect specular reflection direction 441⟩
    *pdf = 1.f;
    return fresnel->Evaluate(CosTheta(wo)) * R / AbsCosTheta(*wi);
}
```

The desired direction is the reflection of ω_o around the surface normal. Because all computations take place in a shading coordinate system where the surface normal is $(0, 0, 1)$, we just rotate ω_i by π radians about $\mathbf{n}$ (Figure 8.7). Recall the transformation matrix from Chapter 2 for a rotation around the z axis; if the angle of rotation is π radians, the matrix is

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

When a vector is multiplied by this matrix, the effect is just to negate the x and y components.

⟨*Compute perfect specular reflection direction*⟩ ≡ 441
```
*wi = Vector(-wo.x, -wo.y, wo.z);
```

CosTheta() 426
Fresnel::Evaluate() 436
Spectrum 263
SpecularReflection 440
SpecularReflection::R 440
Vector 57

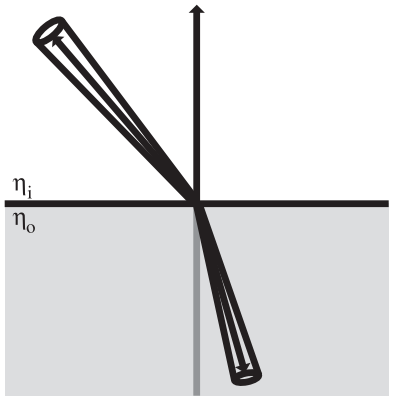


Figure 8.8: The amount of transmitted radiance at the boundary between media with different indices of refraction is scaled by the squared ratio of the two indices of refraction. Intuitively, this can be understood as the result of the radiance's differential solid angle being compressed or expanded as a result of transmission.

8.2.3 SPECULAR TRANSMISSION

We will now derive the BTDF for specular transmission. Snell's law is the basis of the derivation. Not only does it give the direction for the transmitted ray, but it can also be used to show that radiance along a ray changes as the ray goes between media with different indices of refraction.

Consider incident radiance arriving at the boundary between two media, with indices of refraction η_i and η_o for the incoming and outgoing media, respectively (Figure 8.8). We use τ to denote the fraction of incident energy that is transmitted to the outgoing direction, as given by the Fresnel equations, so $\tau = 1 - F_r(\omega_i)$. The amount of transmitted differential flux, then, is

$$d\Phi_o = \tau d\Phi_i.$$

If we use the definition of radiance, Equation (5.2), we have

$$(L_o \cos\theta_o \, dA \, d\omega_o) = \tau (L_i \cos\theta_i \, dA \, d\omega_i).$$

Expanding the solid angles to spherical angles, we have

$$(L_o \cos\theta_o \, dA \sin\theta_o \, d\theta_o \, d\phi_o) = \tau (L_i \cos\theta_i \, dA \sin\theta_i \, d\theta_i \, d\phi_i). \quad [8.4]$$

We can now differentiate Snell's law with respect to θ, which gives the relation

$$\eta_o \cos\theta_o \, d\theta_o = \eta_i \cos\theta_i \, d\theta_i.$$

Rearranging terms, we get

$$\frac{\cos\theta_o \, d\theta_o}{\cos\theta_i \, d\theta_i} = \frac{\eta_i}{\eta_o}.$$

Substituting this and Snell's law into Equation (8.4) and simplifying, we have

$$L_o \, \eta_i^2 \, d\phi_o = \tau L_i \, \eta_o^2 \, d\phi_i.$$

Because $\phi_i = \phi_o + \pi$ and therefore $d\phi_i = d\phi_o$, this gives the final relationship:

$$L_o = \tau L_i \frac{\eta_o^2}{\eta_i^2}. \tag{8.5}$$

The BTDF for specular transmission is thus

$$f_t(p, \omega_i, \omega_o) = \frac{\eta_o^2}{\eta_i^2}(1 - F_r(\omega_i)) \frac{\delta(\omega_i - T(\omega_o, \mathbf{n}))}{|\cos \theta_i|},$$

where $T(\omega_o, \mathbf{n})$ is the specular transmission vector for ω_o about the surface normal $\mathbf{n}$. The $1 - F_r(\omega_i)$ term in this equation corresponds to an easily observed effect: transmission is stronger at near-perpendicular angles. For example, if you look straight down into a clear lake, you can see far into the water, but at grazing angles most of the light is reflected as if from a mirror.

The `SpecularTransmission` class is almost exactly the same as `SpecularReflection` except that the sampled direction is respectively the direction for perfect specular transmission. Figure 8.9 shows an image of the Killeroo model using specular reflection and transmission BRDF and BTDF to model glass.

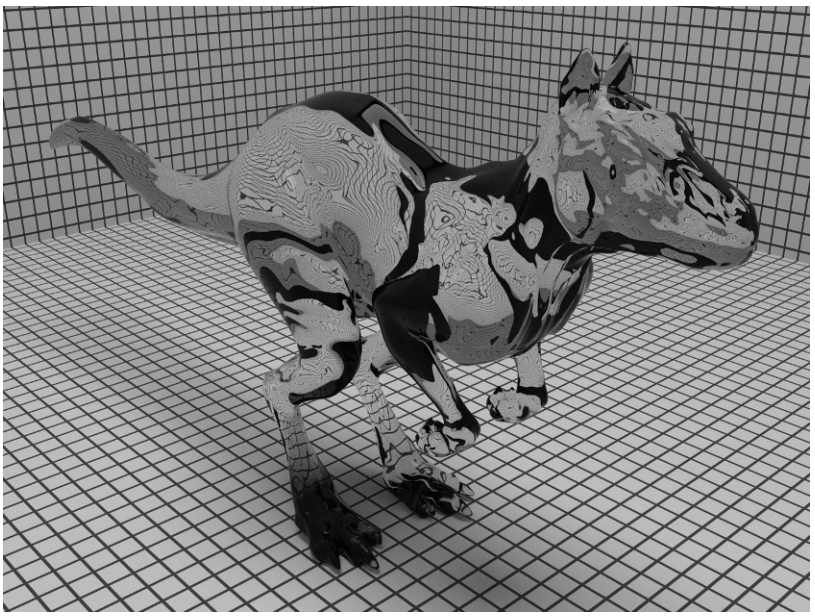


Figure 8.9: When the BRDF for specular reflection and the BTDF for specular transmission are modulated with the Fresnel formula for dielectrics, the realistic angle-dependent variation of the amount of reflection and transmission makes the result more visually convincing. *(Model courtesy of headus/Rezard.)*

SpecularReflection 440
SpecularTransmission 444

⟨*BxDF Declarations*⟩ +≡
```
class SpecularTransmission : public BxDF {
public:
    ⟨SpecularTransmission Public Methods 444⟩
private:
    ⟨SpecularTransmission Private Data 444⟩
};
```

The `SpecularTransmission` constructor stores the two indices of refraction on either side of the surface, as well as a transmission scale factor `T`.

⟨*SpecularTransmission Public Methods*⟩ ≡ 444
```
SpecularTransmission(const Spectrum &t, float ei, float et)
    : BxDF(BxDFType(BSDF_TRANSMISSION | BSDF_SPECULAR)),
      fresnel(ei, et) {
    T = t;
    etai = ei;
    etat = et;
}
```

Because conductors do not transmit light, a `FresnelDielectric` object is always used to do the Fresnel computations.

⟨*SpecularTransmission Private Data*⟩ ≡ 444
```
Spectrum T;
float etai, etat;
FresnelDielectric fresnel;
```

As with `SpecularReflection`, zero is always returned from `SpecularTransmission::f()`, since the BTDF is a scaled delta distribution.

⟨*SpecularTransmission Public Methods*⟩ +≡ 444
```
Spectrum f(const Vector &, const Vector &) const {
    return Spectrum(0.);
}
```

Figure 8.10 shows the geometry of specular transmission. The incident ray is refracted about the surface normal, with the angle θ_t given by Snell's law.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum SpecularTransmission::Sample_f(const Vector &wo,
        Vector *wi, float u1, float u2, float *pdf) const {
    ⟨Figure out which η is incident and which is transmitted 445⟩
    ⟨Compute transmitted ray direction 445⟩
    *pdf = 1.f;
    Spectrum F = fresnel.Evaluate(CosTheta(wo));
    return (et*et)/(ei*ei) * (Spectrum(1.)-F) * T /
        AbsCosTheta(*wi);
}
```

BSDF_SPECULAR 428
BSDF_TRANSMISSION 428
BxDF 428
BxDFType 428
CosTheta() 426
Fresnel::Evaluate() 436
FresnelDielectric 437
Spectrum 263
SpecularReflection 440
SpecularTransmission 444
SpecularTransmission::etai 444
SpecularTransmission::etat 444
SpecularTransmission::f() 444
SpecularTransmission::T 444
Vector 57

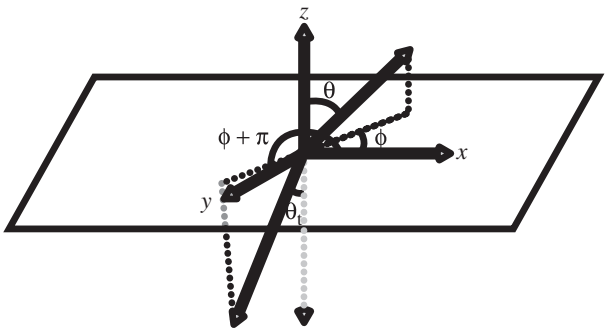


Figure 8.10: The Geometry of Specular Transmission. The specularly transmitted direction makes an angle θ_t with the surface normal. Like specular reflection, the angle it makes with the x axis is π greater than the incident ray's angle.

The method first determines whether the incident ray is entering or exiting the refractive medium. We use the convention that the surface normal, and thus the (0, 0, 1) direction in local reflection space, is oriented such that it points toward the outside of the object. Therefore, if the z component of the ω_o direction is greater than zero, the incident ray is coming from outside of the object.

⟨*Figure out which η is incident and which is transmitted*⟩ ≡ 444
```
bool entering = CosTheta(wo) > 0.;
float ei = etai, et = etat;
if (!entering)
    swap(ei, et);
```

Figure 8.11 shows the basic setting for computing the transmitted ray direction.

We next compute `sini2` and `sint2`, which are the squares of $\sin\theta_i$ and $\sin\theta_t$, respectively. In the reflection coordinate system, $\sin\theta_i$ is equal to the sum of the squares of the x and y components of ω_o; $\sin^2\theta_t$ can be computed directly from $(\sin\theta_i)^2$ using Snell's law. (Note the overloaded notation: these values are for the incident and transmitted media, though here the incident medium is the one the outgoing direction ω_o lies in.)

We then apply the trigonometric identity $\sin^2\theta + \cos^2\theta = 1$ to compute $\cos\theta_t$ from $\sin\theta_t$; this directly gives us the z component of the transmitted direction. To compute the x and y components, we first mirror ω_o about the normal, as we did for specular reflection, but then scale it by the ratio $\sin\theta_t / \sin\theta_i$ to give it the proper magnitude. From Snell's law, this ratio is just η_i/η_t, which we happen to have computed previously.

CosTheta() 426
SinTheta2() 426
SpecularTransmission::
etai 444
SpecularTransmission::
etat 444
Vector 57

⟨*Compute transmitted ray direction*⟩ ≡ 444
```
float sini2 = SinTheta2(wo);
float eta = ei / et;
float sint2 = eta * eta * sini2;
⟨Handle total internal reflection for transmission 446⟩
```

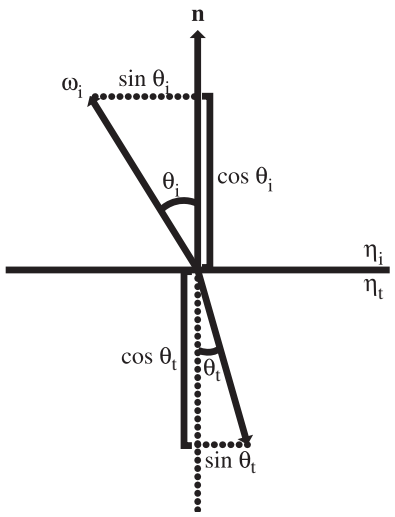


Figure 8.11: Geometry for computing the transmitted direction ω_t from the incident direction ω_i. The $\cos\theta$ terms are equal to the z components of the corresponding direction vectors, and the $\sin\theta$ terms are equal to the xy lengths of the direction vectors.

```
float cost = sqrtf(max(0.f, 1.f - sint2));
if (entering) cost = -cost;
float sintOverSini = eta;
*wi = Vector(sintOverSini * -wo.x, sintOverSini * -wo.y, cost);
```

We need to handle the case of total internal reflection here as well. If the squared value of $\sin\theta_t$ is greater than or equal to one, no transmission is possible, so black is returned.[4]

⟨*Handle total internal reflection for transmission*⟩ ≡ 445

```
if (sint2 >= 1.) return 0.;
```

8.3 LAMBERTIAN REFLECTION

One of the simplest BRDFs is the Lambertian model. It models a perfect diffuse surface that scatters incident illumination equally in all directions. Although this reflection model is not physically plausible, it is a good approximation to many real-world surfaces such as matte paint.

4 The first version of `pbrt` had a test > 1 rather than ≥ 1. Though the difference between the two may seem innocuous, this discrepancy led to not-a-number values occasionally being computed due to the z component of ω_i being zero (in the tangent plane of the surface) and thus to the $1/\cos\theta$ term being infinite.

⟨*BxDF Declarations*⟩ +≡
```
class Lambertian : public BxDF {
public:
    ⟨Lambertian Public Methods 447⟩
private:
    ⟨Lambertian Private Data 447⟩
};
```

The `Lambertian` constructor takes a reflectance spectrum R, which gives the fraction of incident light that is scattered.

⟨*Lambertian Public Methods*⟩ ≡ 447
```
Lambertian(const Spectrum &reflectance)
    : BxDF(BxDFType(BSDF_REFLECTION | BSDF_DIFFUSE)), R(reflectance) { }
```

⟨*Lambertian Private Data*⟩ ≡ 447
```
Spectrum R;
```

The reflection distribution function for `Lambertian` is quite straightforward, since its value is constant. However, the value R/π must be returned, rather than R: the constructor takes the BRDF's reflectance; equating this to the earlier ρ_{hd} integral and solving for the BRDF's value demonstrates why this is the correct value to return for the BRDF.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum Lambertian::f(const Vector &wo, const Vector &wi) const {
    return R * INV_PI;
}
```

The directional-hemispherical and hemispherical-hemispherical reflectance values for a Lambertian BRDF are trivial to compute analytically, so the derivations are omitted here.

⟨*Lambertian Public Methods*⟩ +≡ 447
```
Spectrum rho(const Vector &, int, const float *) const { return R; }
Spectrum rho(int, const float *, const float *) const { return R; }
```

8.4 MICROFACET MODELS

Many geometric-optics-based approaches to modeling surface reflection are based on the idea that rough surfaces can be modeled as a collection of small *microfacets*. A surface comprised of microfacets is essentially a heightfield, where the distribution of facets is described statistically. Figure 8.12 shows cross sections of a relatively rough surface and a much smoother microfacet surface.

Microfacet-based BRDF models work by statistically modeling the scattering of light from a large collection of microfacets. If we assume that the differential area being illuminated, $\mathrm{d}A$, is relatively large compared to the size of individual microfacets, then a large number of microfacets are illuminated, and so their aggregate behavior determines the scattering.

BSDF_DIFFUSE 428
BSDF_REFLECTION 428
BxDF 428
BxDFType 428
INV_PI 1002
Lambertian 447
Lambertian::R 447
Spectrum 263
Vector 57

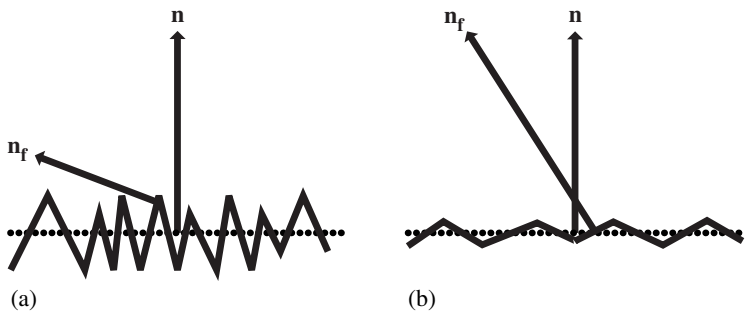


Figure 8.12: Microfacet surface models are often described by a function that gives the distribution of microfacet normals $\mathbf{n}_f$ with respect to the surface normal $\mathbf{n}$. (a) The greater the variation of microfacet normals, the rougher the surface is. (b) Smooth surfaces have relatively little variation of microfacet normals.

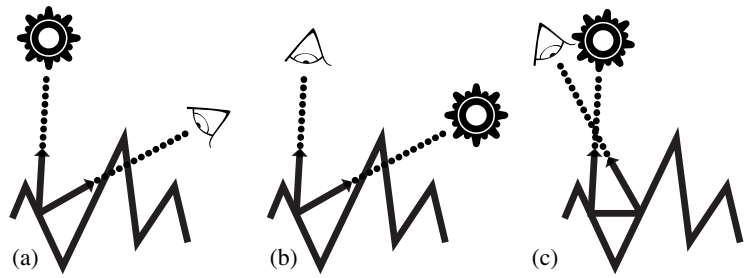


Figure 8.13: Three Important Geometric Effects to Consider with Microfacet Reflection Models. (a) *Masking*: the microfacet of interest isn't visible to the viewer due to occlusion by another microfacet. (b) *Shadowing*: analogously, light doesn't reach the microfacet. (c) *Interreflection*: light bounces among the microfacets before reaching the viewer.

The two main components of microfacet models are an expression for the distribution of facets and a BRDF that describes how light scatters from individual microfacets. Given these, the goal is to derive a closed-form expression giving the BRDF that describes scattering from such a surface. Perfect mirror reflection is typically used for the microfacet BRDF, although the Oren–Nayar model (described in the next section) treats them as Lambertian reflectors.

To compute reflection from such a model, local lighting effects at the microfacet level need to be considered (Figure 8.13). Microfacets may be occluded by another facet, may lie in the shadow of a neighboring microfacet, or interreflection may cause a microfacet to reflect more light than predicted by the amount of direct illumination and the low-level microfacet BRDF. A common simplification is to assume that all of the microfacets make up symmetric V-shaped grooves. If this assumption is made, then interreflection

with most of the other microfacets can be ignored and only the neighboring microfacet needs to be considered.

Particular microfacet-based BRDF models consider each of these effects with varying degrees of accuracy. The general approach is to make the best approximations possible, while still obtaining an easily evaluated expression.

8.4.1 OREN–NAYAR DIFFUSE REFLECTION

Oren and Nayar (1994) observed that real-world objects tend not to exhibit perfect Lambertian reflection. Specifically, rough surfaces generally appear brighter as the illumination direction approaches the viewing direction. They developed a reflection model that describes rough surfaces as a collection of symmetric V-shaped grooves in an effort to better model effects like these. They further assumed that each individual microfacet (groove face) exhibited perfect Lambertian reflection and derived a BRDF that models the aggregate reflection of the collection of grooves. The distribution of microfacets was modeled with a Gaussian distribution with a single parameter σ, the standard deviation of the orientation angle.

The resulting model, which accounts for shadowing, masking, and interreflection among the microfacets, does not have a closed-form solution, so they found the following approximation that fit it well:

$$f_r(\omega_i, \omega_o) = \frac{\rho}{\pi}\left(A + B\,\max(0, \cos(\phi_i - \phi_o))\sin\alpha\tan\beta\right),$$

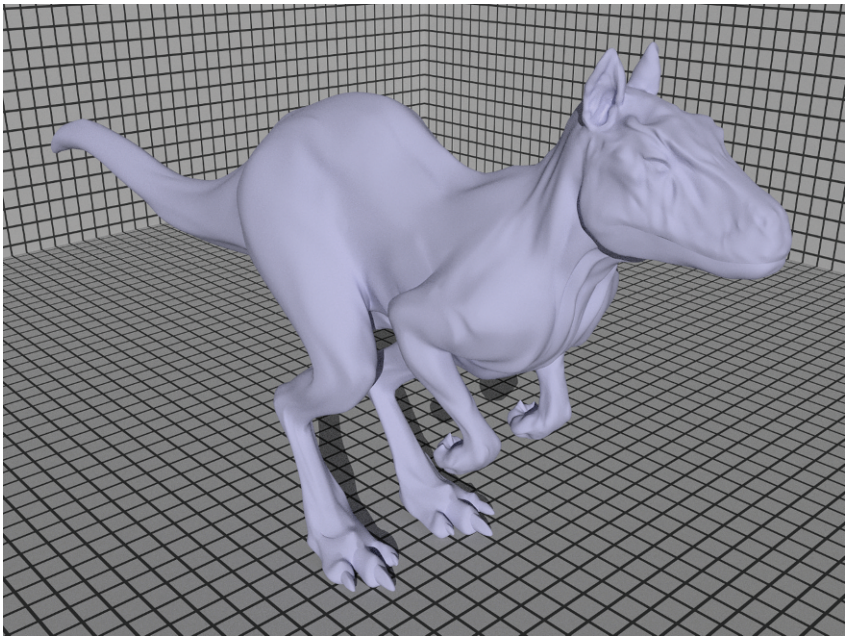
where if σ is in radians,

$$A = 1 - \frac{\sigma^2}{2(\sigma^2 + 0.33)}$$
$$B = \frac{0.45\sigma^2}{\sigma^2 + 0.09}$$
$$\alpha = \max(\theta_i, \theta_o)$$
$$\beta = \min(\theta_i, \theta_o).$$

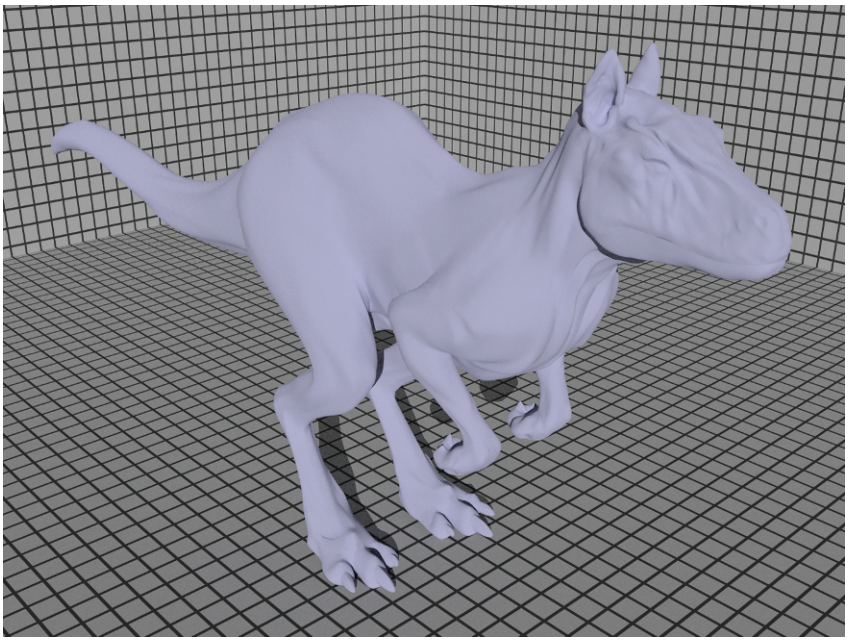
The implementation precomputes and stores the values of the A and B parameters in the constructor to save work in evaluating the BRDF later. Figure 8.14 compares the difference between rendering with ideal diffuse reflection and with the Oren–Nayar model.

BSDF_DIFFUSE 428
BSDF_REFLECTION 428
BxDF 428
BxDFType 428
OrenNayar::A 451
OrenNayar::B 451
OrenNayar::R 451
Spectrum 263

⟨*OrenNayar Public Methods*⟩ ≡
```
OrenNayar(const Spectrum &reflectance, float sig)
    : BxDF(BxDFType(BSDF_REFLECTION | BSDF_DIFFUSE)),
      R(reflectance) {
    float sigma = Radians(sig);
    float sigma2 = sigma*sigma;
    A = 1.f - (sigma2 / (2.f * (sigma2 + 0.33f)));
    B = 0.45f * sigma2 / (sigma2 + 0.09f);
}
```



(a)



(b)

Figure 8.14: Killeroo model rendered (a) with standard diffuse reflection from the `Lambertian` model and (b) with the `OrenNayar` model with a σ parameter of 20 degrees. Note the increase in reflection at the silhouette edges and the generally less-drawn-out transitions at light terminator edges with the Oren–Nayar model. *(Model courtesy of headus/Rezard.)*

⟨*OrenNayar Private Data*⟩ ≡
```
Spectrum R;
float A, B;
```

Application of trigonometry can substantially improve the efficiency of the evaluation routine compared to a naive implementation. The implementation starts by computing the values of $\sin\theta_i$ and $\sin\theta_o$.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum OrenNayar::f(const Vector &wo, const Vector &wi) const {
    float sinthetai = SinTheta(wi);
    float sinthetao = SinTheta(wo);
    ⟨Compute cosine term of Oren-Nayar model 451⟩
    ⟨Compute sine and tangent terms of Oren-Nayar model 451⟩
    return R * INV_PI * (A + B * maxcos * sinalpha * tanbeta);
}
```

It is now necessary to compute the $\max(0, \cos(\phi_i - \phi_o))$ term. We can apply the trigonometric identity

$$\cos(a - b) = \cos a \cos b + \sin a \sin b,$$

such that we just need to compute the sines and cosines of ϕ_i and ϕ_o.

⟨*Compute cosine term of Oren-Nayar model*⟩ ≡ 451
```
float maxcos = 0.f;
if (sinthetai > 1e-4 && sinthetao > 1e-4) {
    float sinphii = SinPhi(wi), cosphii = CosPhi(wi);
    float sinphio = SinPhi(wo), cosphio = CosPhi(wo);
    float dcos = cosphii * cosphio + sinphii * sinphio;
    maxcos = max(0.f, dcos);
}
```

Finally, the $\sin\alpha$ and $\tan\beta$ terms are found. Note that whichever of ω_i or ω_o has a larger value for $\cos\theta$ (i.e., a larger z component) has a *smaller* value for θ. We can set $\sin\alpha$ using the appropriate sine value computed at the beginning of the method. The tangent can then be computed using the identity $\tan a = \sin a / \cos a$.

AbsCosTheta() 426
CosPhi() 427
INV_PI 1002
OrenNayar 449
OrenNayar::A 451
OrenNayar::B 451
OrenNayar::R 451
SinPhi() 427
SinTheta() 426
Spectrum 263
Vector 57

⟨*Compute sine and tangent terms of Oren-Nayar model*⟩ ≡ 451
```
float sinalpha, tanbeta;
if (AbsCosTheta(wi) > AbsCosTheta(wo)) {
    sinalpha = sinthetao;
    tanbeta = sinthetai / AbsCosTheta(wi);
}
else {
    sinalpha = sinthetai;
    tanbeta = sinthetao / AbsCosTheta(wo);
}
```

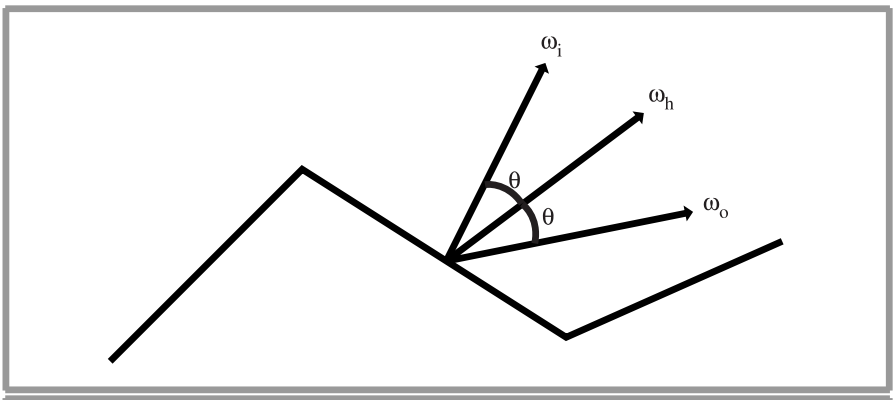


Figure 8.15: For perfectly specular microfacets and a given pair of directions ω_i and ω_o, only those microfacets with normal $\omega_h = \omega_i \widehat{+} \omega_o$ will reflect any light from ω_i to ω_o.

8.4.2 TORRANCE–SPARROW MODEL

One of the first microfacet models for computer graphics was developed by Torrance and Sparrow (1967) to model metallic surfaces. They modeled surfaces as collections of perfectly smooth mirrored microfacets. The surface is statistically described by a distribution function $D(\omega_h)$ that gives the probability that a microfacet has orientation ω_h (recall Figure 8.12, which shows how roughness and the microfacet normal distribution function are related).

Because the microfacets are perfectly specular, only those with a normal equal to the *half-angle vector*,

$$\omega_h = \omega_i \widehat{+} \omega_o,$$

cause perfect specular reflection from ω_i to ω_o (Figure 8.15).

The derivation of the Torrance–Sparrow model has a number of interesting steps; we'll go through it in detail here. Consider the differential flux $d\Phi_h$ incident on the microfacets oriented with half-angle ω_h for directions ω_i and ω_o. From the definition of radiance, Equation (5.2), it is

$$d\Phi_h = L_i(\omega_i)\, d\omega\, dA^{\perp}(\omega_h) = L_i(\omega_i)\, d\omega\, \cos\theta_h\, dA(\omega_h),$$

where we have written $dA(\omega_h)$ for the area measure of the microfacets with orientation ω_h and $\cos\theta_h$ for the cosine of the angle between ω_i and ω_h (Figure 8.16).

The differential area of microfacets with orientation ω_h is

$$dA(\omega_h) = D(\omega_h)\, d\omega_h\, dA.$$

The first two terms describe the differential area of facets per unit area that have the proper orientation, and the dA term converts this to differential area.

Therefore,

$$d\Phi_h = L_i(\omega_i)\, d\omega\, \cos\theta_h\, D(\omega_h)\, d\omega_h\, dA. \quad \text{(8.6)}$$

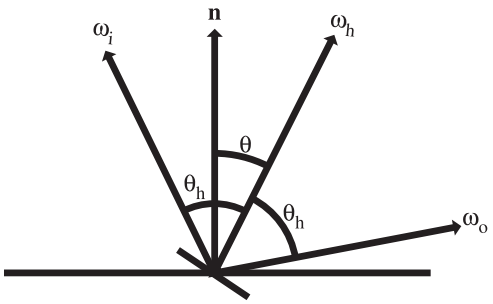


Figure 8.16: Setting for the Derivation of the Torrance–Sparrow Model. For directions ω_i and ω_o, only microfacets with normal ω_h reflect light. The angle between ω_h and $\mathbf{n}$ is denoted by θ, and the angle between ω_h and ω_o is denoted by θ_h. (The angle between ω_h and ω_i is also necessarily θ_h.)

If we assume that the microfacets individually reflect light according to Fresnel's law, the outgoing flux is

$$\mathrm{d}\Phi_o = F_r(\omega_o)\mathrm{d}\Phi_h. \tag{8.7}$$

Again using the definition of radiance, the reflected outgoing radiance is

$$L(\omega_o) = \frac{\mathrm{d}\Phi_o}{\mathrm{d}\omega_o \cos\theta_o \,\mathrm{d}A}.$$

If we substitute Equation (8.7) into this and then Equation (8.6) into the result, we have

$$L(\omega_o) = \frac{F_r(\omega_o)\, L_i(\omega_i)\,\mathrm{d}\omega_i\, D(\omega_h)\,\mathrm{d}\omega_h\,\mathrm{d}A\, \cos\theta_h}{\mathrm{d}\omega_o\,\mathrm{d}A\, \cos\theta_o}.$$

In Section 14.5.1, we will derive an important relationship between $\mathrm{d}\omega_h$ and $\mathrm{d}\omega_o$:

$$\mathrm{d}\omega_h = \frac{\mathrm{d}\omega_o}{4\cos\theta_h}.$$

We can substitute this into the previous equation and simplify, giving

$$L(\omega_o) = \frac{F_r(\omega_o)\, L_i(\omega_i)\, D(\omega_h)\,\mathrm{d}\omega_i}{4\,\cos\theta_o}.$$

We can now apply the definition of the BRDF, Equation (5.7), giving us the Torrance–Sparrow BRDF:

$$f_r(\mathrm{p}, \omega_o, \omega_i) = \frac{D(\omega_h)\, F_r(\omega_o)}{4\,\cos\theta_o\,\cos\theta_i}.$$

The Torrance–Sparrow model also includes a *geometric attenuation* term, which describes the fraction of microfacets that are masked or shadowed, given directions ω_i and ω_o. This

G term can just be included in the derivation as the Fresnel term was earlier. The full model, then, is

$$f_r(p, \omega_o, \omega_i) = \frac{D(\omega_h)\, G(\omega_o, \omega_i)\, F_r(\omega_o)}{4 \cos\theta_o \cos\theta_i}. \quad [8.8]$$

One of the nice things about the Torrance–Sparrow model is that the derivation doesn't depend on the particular microfacet distribution being used. Furthermore, it doesn't depend on a particular Fresnel function, so it can be used for both conductors and dielectrics. However, reflection functions other than perfect specular reflection cannot be easily substituted: the relationship between $d\omega_h$ and $d\omega_o$ used in the derivation depends on the specular reflection assumption.

We now use the Torrance–Sparrow model to implement a general microfacet-based BRDF. It takes a pointer to an abstract `MicrofacetDistribution` class, which provides a single method to compute the D term of the Torrance–Sparrow model. This function, `MicrofacetDistribution::D()`, gives the probability density for microfacets to be oriented with normal ω_h.

⟨*BxDF Declarations*⟩ +≡
```
class MicrofacetDistribution {
public:
    ⟨MicrofacetDistribution Interface 454⟩
};
```

⟨*MicrofacetDistribution Interface*⟩ ≡ 454
```
virtual float D(const Vector &wh) const = 0;
```

The `Microfacet` BRDF, then, just takes a reflectance, the Fresnel function, and a pointer to a distribution.

⟨*BxDF Declarations*⟩ +≡
```
class Microfacet : public BxDF {
public:
    ⟨Microfacet Public Methods 455⟩
private:
    ⟨Microfacet Private Data 454⟩
};
```

⟨*BxDF Method Definitions*⟩ +≡
```
Microfacet::Microfacet(const Spectrum &reflectance, Fresnel *f,
                       MicrofacetDistribution *d)
    : BxDF(BxDFType(BSDF_REFLECTION | BSDF_GLOSSY)),
      R(reflectance), distribution(d), fresnel(f) {
}
```

⟨*Microfacet Private Data*⟩ ≡ 454
```
Spectrum R;
MicrofacetDistribution *distribution;
Fresnel *fresnel;
```

BSDF_GLOSSY 428
BSDF_REFLECTION 428
BxDF 428
BxDFType 428
Fresnel 436
Microfacet 454
Microfacet::distribution 454
Microfacet::fresnel 454
Microfacet::R 454
MicrofacetDistribution 454
MicrofacetDistribution::D() 454
Spectrum 263
Vector 57

Evaluating the terms of the Torrance–Sparrow BRDF is straightforward. For the Fresnel term, recall that the angle θ_h is the same between ω_h and both ω_i and ω_o, so it doesn't matter which vector we use to compute the cosine of θ_h. We arbitrarily choose ω_i.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum Microfacet::f(const Vector &wo, const Vector &wi) const {
    float cosThetaO = AbsCosTheta(wo);
    float cosThetaI = AbsCosTheta(wi);
    if (cosThetaI == 0.f || cosThetaO == 0.f) return Spectrum(0.f);
    Vector wh = Normalize(wi + wo);
    float cosThetaH = Dot(wi, wh);
    Spectrum F = fresnel->Evaluate(cosThetaH);
    return R * distribution->D(wh) * G(wo, wi, wh) * F /
               (4.f * cosThetaI * cosThetaO);
}
```

The geometric attenuation term is derived by assuming that the microfacets are arranged along infinitely long V-shaped grooves. This assumption is more restrictive than the general term $D(\omega_h)$ used to model the microfacet distribution and does not account for the roughness of the surface, but yields a closed-form result, which is helpful for efficiency. It is also easy to evaluate and matches many real-world surfaces well. The attenuation term (omitting the derivation) is

$$G(\omega_o, \omega_i) = \min\left(1, \min\left(\frac{2(\mathbf{n}\cdot\omega_h)(\mathbf{n}\cdot\omega_o)}{\omega_o\cdot\omega_h}, \frac{2(\mathbf{n}\cdot\omega_h)(\mathbf{n}\cdot\omega_i)}{\omega_o\cdot\omega_h}\right)\right).$$

⟨*Microfacet Public Methods*⟩ ≡ 454
```
float G(const Vector &wo, const Vector &wi, const Vector &wh) const {
    float NdotWh = AbsCosTheta(wh);
    float NdotWo = AbsCosTheta(wo);
    float NdotWi = AbsCosTheta(wi);
    float WOdotWh = AbsDot(wo, wh);
    return min(1.f, min((2.f * NdotWh * NdotWo / WOdotWh),
                        (2.f * NdotWh * NdotWi / WOdotWh)));
}
```

AbsCosTheta() 426
AbsDot() 61
Dot() 60
Fresnel::Evaluate() 436
Microfacet 454
Microfacet::G() 455
Microfacet::R 454
MicrofacetDistribution::D() 454
Spectrum 263
Vector 57
Vector::Normalize() 63

8.4.3 BLINN MICROFACET DISTRIBUTION

Blinn (1977) proposed a model where the distribution of microfacet normals is approximated by an exponential falloff. The most likely microfacet orientation in this model is in the surface normal direction, falling off to no microfacets oriented perpendicular to the normal. For smooth surfaces, this falloff happens very quickly; for rough surfaces, it is more gradual.

The Blinn model raises the cosine of the angle between the half-vector and the normal to an exponent e:

$$D(\omega_h) \propto (\omega_h \cdot \mathbf{n})^e.$$

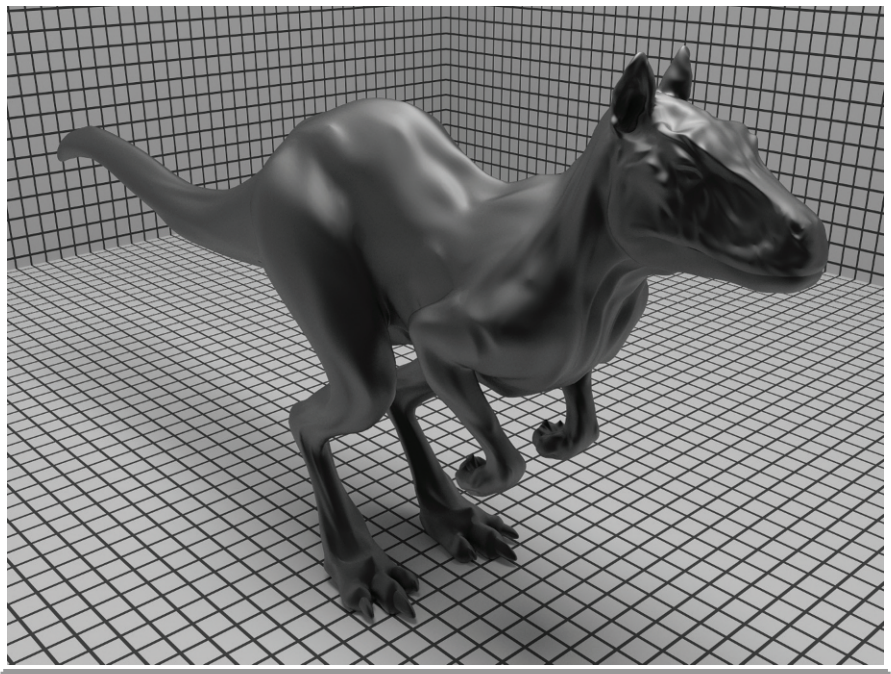


Figure 8.17: Killeroo model rendered with the Torrance–Sparrow microfacet model and Blinn microfacet distribution function. *(Model courtesy of headus/Rezard.)*

Figure 8.17 shows the Killeroo rendered with the Torrance–Sparrow model and the Blinn microfacet distribution.

⟨*BxDF Declarations*⟩ +≡
```
class Blinn : public MicrofacetDistribution {
public:
    Blinn(float e) { if (e > 10000.f || isnan(e)) e = 10000.f;
                     exponent = e; }
    ⟨Blinn Public Methods 457⟩
private:
    float exponent;
};
```

Microfacet distribution functions must be *normalized* to ensure that they are physically plausible. In the case of microfacets, there must exist some heightfield with the distribution of face normals $D(\omega_h)$. Another way to think about this restriction is that if we sum the projected area of all the microfacet faces over some area dA, the sum should equal dA. Mathematically, this means we must enforce

$$\int_{\mathcal{H}^2(\mathbf{n})} D(\omega_h) \cos\theta_h \, d\omega_h = 1.$$

Blinn 456
MicrofacetDistribution 454

A common error in normalizing microfacet distributions is to perform this integral over solid angle instead of projected solid angle (i.e., to leave out the $\cos\theta_h$ term), which does not guarantee the existence of a heightfield with the right distribution.

For the Blinn model, we know that $D(\omega_h) \propto (\omega_h \cdot \mathbf{n})^e$. The earlier normalization requirement gives

$$\begin{aligned}\int_{\mathcal{H}^2(\mathbf{n})} c(\omega_h \cdot \mathbf{n})^e \cos\theta_h \, d\omega_h &= \int_0^{2\pi} \int_0^{\frac{\pi}{2}} c(\cos\theta_h)^{e+1} \sin\theta_h \, d\theta \, d\phi \\ &= 2c\pi \int_0^1 u^{e+1} du \\ &= 2c\pi \left. \frac{u^{e+2}}{e+2} \right|_0^1 \\ &= \frac{2c\pi}{e+2} = 1.\end{aligned}$$

Therefore,

$$c = \frac{e+2}{2\pi},$$

and the properly normalized Blinn microfacet distribution is

$$D(\omega_h) = \frac{e+2}{2\pi}(\omega_h \cdot \mathbf{n})^e. \tag{8.9}$$

Figure 8.18 shows how the exponent affects the distribution. Figure 8.18(a) shows a plot of the Blinn model's distribution with outgoing direction shown and an exponent of 4. This corresponds to a fairly rough surface, so there is a high probability of microfacets being oriented in a direction far away from the normal. Figure 8.18(b) shows the distribution from an exponent of 20, corresponding to a smoother surface. For this case, there is a much lower probability that any microfacets will be oriented very far from the surface normal direction.

⟨*Blinn Public Methods*⟩ ≡ 456

```
float D(const Vector &wh) const {
    float costhetah = AbsCosTheta(wh);
    return (exponent+2) * INV_TWOPI * powf(costhetah, exponent);
}
```

8.4.4 ANISOTROPIC MICROFACET MODEL

Because the Blinn distribution described in the last section only depends on the angle between the half-angle and the surface normal, it is *radially symmetric* and yields an isotropic BRDF. Ashikhmin and Shirley (2000, 2002) developed a microfacet distribution function for modeling the appearance of anisotropic surfaces. Recall that an anisotropic BRDF is one where the reflection characteristics at a point vary as the surface is rotated about that point in the plane perpendicular to the surface normal. Brushed metals and some types of fabric exhibit anisotropy.

AbsCosTheta() 426
INV_TWOPI 1002
Vector 57

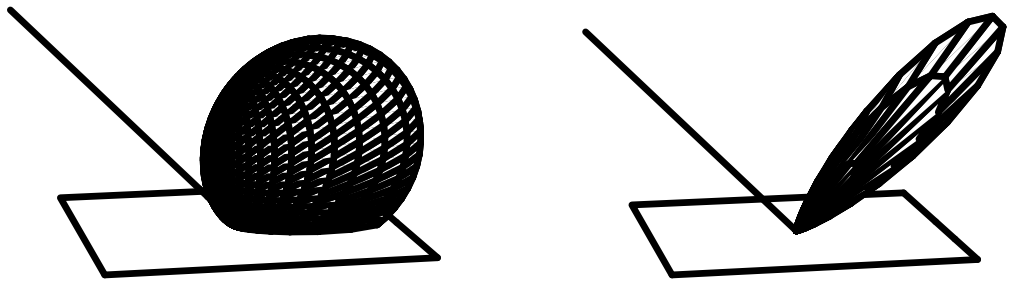


Figure 8.18: The Effect of Varying the Exponent for the Blinn Microfacet Distribution Model. (a) Distribution from exponent $e = 4$, (b) $e = 20$. The larger the exponent, the more likely it is that a microfacet will be oriented close to the surface normal, as would be the case for a smooth surface.

Ashikhmin and Shirley's model is physically based, has intuitive parameters, is efficient, and fits well into the Monte Carlo integration techniques that will be introduced in later chapters. We won't present the derivation here; the reader is referred to the "Further Reading" section for pointers to their original papers. Their model takes two parameters: e_x, which gives an exponent for the distribution function for half-angle vectors with an azimuthal angle that orients them exactly along the x axis, and e_y, an exponent for microfacets oriented along the y axis. Exponents for intermediate orientations are found by considering these two values as the lengths of the axes of an ellipse and finding the appropriate radius for the actual microfacet orientation (Figure 8.19).

The resulting microfacet distribution function is

$$D(\omega_h) = \frac{\sqrt{(e_x+2)(e_y+2)}}{2\pi}(\omega_h \cdot \mathbf{n})^{e_x \cos^2\phi + e_y \sin^2\phi}.$$

Figure 8.20 shows this model in use.

⟨*BxDF Declarations*⟩ +≡
```
class Anisotropic : public MicrofacetDistribution {
public:
    ⟨Anisotropic Public Methods 458⟩
private:
    float ex, ey;
};
```

⟨*Anisotropic Public Methods*⟩ ≡ 458
```
Anisotropic(float x, float y) {
    ex = x; ey = y;
    if (ex > 10000.f || isnan(ex)) ex = 10000.f;
    if (ey > 10000.f || isnan(ey)) ey = 10000.f;
}
```

Anisotropic 458
MicrofacetDistribution 454

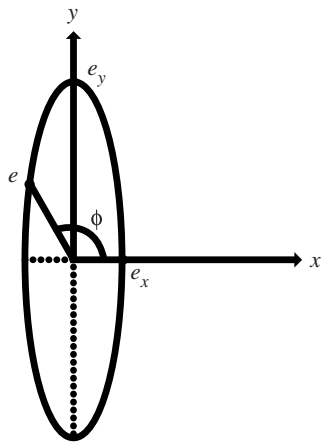


Figure 8.19: The two exponents e_x and e_y for the anisotropic microfacet distribution function give specular exponents for microfacets facing exactly along the x and y axes, respectively. For microfacets with other orientations, the exponent e is computed by finding the radius e of the ellipse for the actual orientation angle ϕ.

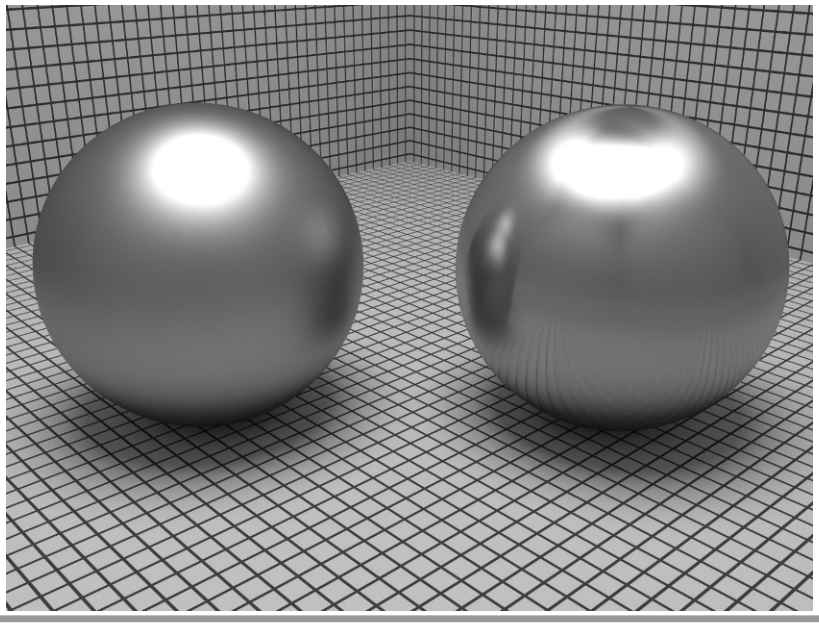


Figure 8.20: Spheres rendered with an isotropic microfacet distribution (left) and an anisotropic distribution (right). Note the different specular highlight shapes from the anisotropic model. We have used spheres here instead of the Killeroo, since anisotropic models like these depend on a globally consistent set of tangent vectors over the surface to orient the direction of anisotropy in a reasonable way.

The terms of the distribution function can be computed quite efficiently. Recall from the beginning of the chapter that $\cos\phi = x/\sin\theta$ and $\sin\phi = y/\sin\theta$. Since we want to compute $\cos^2\phi$ and $\sin^2\phi$, however, we can use the substitution $\sin^2\theta + \cos^2\theta = 1$, so that

$$\cos^2\phi = \frac{x^2}{1-z^2}$$

$$\sin^2\phi = \frac{y^2}{1-z^2}.$$

Thus, the implementation is

⟨*Anisotropic Public Methods*⟩ +≡ 458

```
float D(const Vector &wh) const {
    float costhetah = AbsCosTheta(wh);
    float d = 1.f - costhetah * costhetah;
    if (d == 0.f) return 0.f;
    float e = (ex * wh.x * wh.x + ey * wh.y * wh.y) / d;
    return sqrtf((ex+2.f) * (ey+2.f)) * INV_TWOPI * powf(costhetah, e);
}
```

8.5 FRESNEL INCIDENCE EFFECTS

Most BRDF models in graphics do not account for the fact that Fresnel reflection reduces the amount of light that reaches the bottom level of layered objects. Consider a polished wood table or a wall with glossy paint: if you look at their surfaces head-on, you primarily see the wood or the paint pigment color. As you move your viewpoint toward a glancing angle, you see less of the underlying color as it is overwhelmed by increasing glossy reflection due to Fresnel effects.

In this section, we will implement a BRDF model developed by Ashikhmin and Shirley (2000, 2002) that models a diffuse underlying surface with a glossy specular surface above it. The effect of reflection from the diffuse surface is modulated by the amount of energy left after Fresnel effects have been considered. Figure 8.21 shows this idea: When the incident direction is close to the normal, most light is transmitted to the diffuse layer and the diffuse term dominates. When the incident direction is close to glancing, glossy reflection is the primary mode of reflection. The car model in Figures 12.14 and 12.15 uses this BRDF for its paint.

⟨*BxDF Declarations*⟩ +≡

```
class FresnelBlend : public BxDF {
public:
    ⟨FresnelBlend Public Methods 462⟩
private:
    ⟨FresnelBlend Private Data 461⟩
};
```

The model takes two spectra, representing diffuse and specular reflectance, and a microfacet distribution for the glossy layer.

AbsCosTheta() 426
BxDF 428
INV_TWOPI 1002
Vector 57

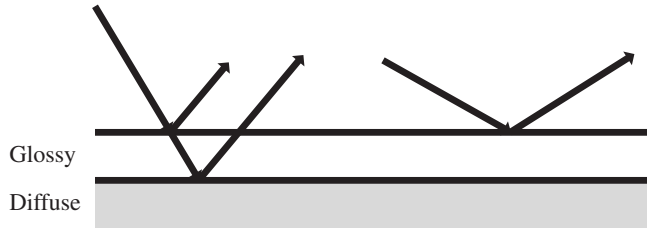


Figure 8.21: The `FresnelBlend` BRDF models the effect of a surface with a glossy layer on top of a diffuse substrate. As the angle of incidence of the vectors ω_i and ω_o heads toward glancing (right), the amount of light that reaches the diffuse substrate is reduced by Fresnel effects and the diffuse layer becomes less visibly apparent.

⟨*BxDF Method Definitions*⟩ +≡
```
FresnelBlend::FresnelBlend(const Spectrum &d, const Spectrum &s,
                           MicrofacetDistribution *dist)
    : BxDF(BxDFType(BSDF_REFLECTION | BSDF_GLOSSY)), Rd(d), Rs(s) {
    distribution = dist;
}
```

⟨*FresnelBlend Private Data*⟩ ≡ 460
```
Spectrum Rd, Rs;
MicrofacetDistribution *distribution;
```

This model is based on the weighted sum of a glossy specular term and a diffuse term. Accounting for reciprocity and energy conservation, the glossy specular term is derived as

$$f_r(p, \omega_o, \omega_i) = \frac{D(\omega_h)F(\omega_o)}{4\pi(\omega_h \cdot \omega_i)(\max((\mathbf{n} \cdot \omega_o), (\mathbf{n} \cdot \omega_i)))},$$

where $D(\omega_h)$ is a microfacet distribution term and $F(\omega_o)$ represents Fresnel reflectance. Note that this is quite similar to the Torrance–Sparrow model.

The key to Ashikhmin and Shirley's model is the derivation of a diffuse term such that the model still obeys reciprocity and conserves energy. The derivation is dependent on an approximation to the Fresnel reflection equations due to Schlick (1993), who approximated Fresnel reflection as

$$F_r(\cos\theta) = R + (1 - R)(1 - \cos\theta)^5,$$

where R is the reflectance of the surface at normal incidence.

Given this Fresnel term, the diffuse term in the following equation successfully models Fresnel-based reduced diffuse reflection in a physically plausible manner:

$$f_r(p, \omega_i, \omega_o) = \frac{28R_d}{23\pi}(1 - R_s)\left(1 - \left(1 - \frac{(\mathbf{n} \cdot \omega_i)}{2}\right)^5\right)\left(1 - \left(1 - \frac{(\mathbf{n} \cdot \omega_o)}{2}\right)^5\right).$$

We will not include the derivation of this result here.

BSDF_GLOSSY 428
BSDF_REFLECTION 428
BxDF 428
BxDFType 428
FresnelBlend 460
FresnelBlend::distribution 461
FresnelBlend::Rd 461
FresnelBlend::Rs 461
MicrofacetDistribution 454
Spectrum 263

⟨*FresnelBlend Public Methods*⟩ ≡ 460
```
Spectrum SchlickFresnel(float costheta) const {
    return Rs + powf(1 - costheta, 5.f) * (Spectrum(1.) - Rs);
}
```

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum FresnelBlend::f(const Vector &wo, const Vector &wi) const {
    Spectrum diffuse = (28.f/(23.f*M_PI)) * Rd *
        (Spectrum(1.f) - Rs) *
        (1.f - powf(1.f - .5f * AbsCosTheta(wi), 5)) *
        (1.f - powf(1.f - .5f * AbsCosTheta(wo), 5));
    Vector wh = Normalize(wi + wo);
    Spectrum specular = distribution->D(wh) /
        (4.f * AbsDot(wi, wh) * max(AbsCosTheta(wi), AbsCosTheta(wo))) *
        SchlickFresnel(Dot(wi, wh));
    return diffuse + specular;
}
```

8.6 MEASURED BRDFs

Using measured data about the reflection properties of real surfaces is one of the most effective approaches for rendering realistic materials. One way to do so is to use the measured data to set the parameter values for a parameterized BRDF like the Torrance–Sparrow model. If this can be done without introducing too much error, then doing so is worthwhile for the memory savings of not needing to store all of the measurement data in memory for rendering. (This parameter fitting would normally be done as a preprocess outside of the renderer.)

However, parameterized BRDF models are often not flexible enough to model the full complexity of scattering characteristics of many interesting surfaces. If accurate measured reflection data is available for a surface of interest, then directly using it for rendering can faithfully re-create the surface's appearance. In this section, we'll implement `BxDF`s that interpolate sampled BRDF values directly. While this is a memory-intensive approach for scenes that use many different measured BRDFs, it works well for reflection distributions that don't fit analytic models well and can provide a reference for comparison to other approaches. Figure 8.22 shows an image of the Killeroo model rendered with measured data from a red fabric.

Rendering with measured BRDF data does have challenges. It can be difficult to make adjustments for artistic purposes ("like that, just a little more shiny").[5] If there is excessive error in the data, results may be poor as well: especially for highly specular and anisotropic surfaces, the 4D BRDF must be densely sampled, leading to a large amount of data to store and many measurements to be taken. Finally, BRDFs may have considerable variation in their values, which adds to the challenges of acquiring accurate BRDF data sets.

AbsCosTheta() 426
BxDF 428
FresnelBlend 460
FresnelBlend::Rd 461
FresnelBlend::Rs 461
FresnelBlend::SchlickFresnel() 462
MicrofacetDistribution::D() 454
M_PI 1002
Spectrum 263
Vector 57
Vector::Normalize() 63

5 Though see the work of Matusik et al. (2003b) for an innovative approach to this problem.

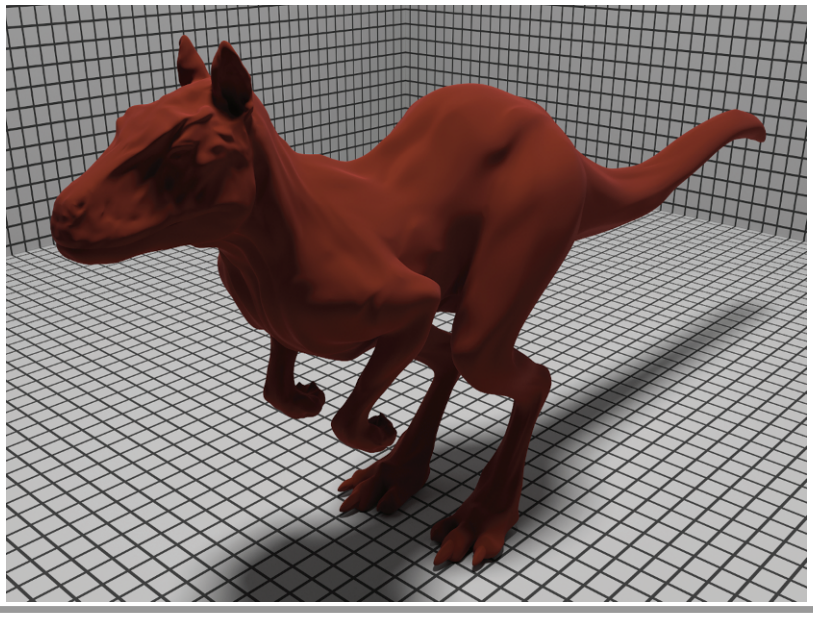


Figure 8.22: The Killeroo model rendered using measured red fabric BRDF data.

Measured BRDF data may come in one of two forms: as regularly spaced tabularized data, or as a large number of irregularly spaced individual samples. Regularly spaced data makes efficient lookups possible. (For example, if we have a 4D table of values for an anisotropic BRDF, indexed by $(\theta_i, \phi_i, \theta_o, \phi_o)$, then, given a pair of directions, we just need to index into the table and interpolate between the values.) However, it can be difficult to acquire BRDF values with such a regular spacing. Acquiring good measurements for directions close to the horizon can be particularly difficult, for example. Irregular sample spacings avoid these issues, though are not as easy to use for rendering. (Irregular sets of BRDF samples are often resampled in a preprocess to generate a set of regular samples.)

Therefore, `pbrt` has implementations of two `BxDF`s for using measured reflection data for rendering. The first, `IrregIsotropicBRDF` interpolates from irregularly spaced samples of isotropic BRDFs. The second, `RegularHalfangleBRDF`, supports the regular sampling used by Matusik et al.'s data set (2003a, 2003b). Both of these are used by the `MeasuredMaterial` defined in Section 9.2.4.

`BxDF` 428
`IrregIsotropicBRDF` 464
`MeasuredMaterial` 489
`RegularHalfangleBRDF` 467

8.6.1 IRREGULAR ISOTROPIC MEASURED BRDF

`IrregIsotropicBRDF` supports irregularly sampled isotropic BRDF data. Given a pair of directions, it finds a few BRDF samples that are nearby in the space of incident and outgoing directions and interpolates between their values, based on how close they are to the actual directions.

⟨*BxDF Declarations*⟩ +≡
```
class IrregIsotropicBRDF : public BxDF {
public:
    ⟨IrregIsotropicBRDF Public Methods 465⟩
private:
    ⟨IrregIsotropicBRDF Private Data 465⟩
};
```

Finding a good encoding for the BRDF directions is important. Consider a table of isotropic BRDF samples, where each sample records the BRDF value for a given pair of directions defined by $(\theta_i, \phi_i, \theta_o, \phi_o)$. This is a natural form for measured data to be stored in, but it is not a good one for resampling and interpolation. One shortcoming of this representation is that the isotropy of the BRDF isn't reflected in the representation. Recall that for isotropic BRDFs rotation about the normal direction leaves the value unchanged, such that

$$f_r((\theta_i, \phi_i), (\theta_o, \phi_o)) = f_r((\theta_i, \phi_i + \delta), (\theta_o, \phi_o + \delta))$$

for all δ values. Therefore, a better approach would be to store the samples indexed by the two θ angles and the difference between ϕ directions, $(\theta_i, \theta_o, \Delta\phi)$, where $\Delta\phi = \phi_i - \phi_o$, thus reflecting this structure in the data. This gives a mapping from directions to a three-tuple:

$$m(\theta_i, \phi_i, \theta_o, \phi_o) \rightarrow (\theta_i, \theta_o, \phi_i - \phi_o)$$

If we did not account for the isotropy with a mapping like this one, and just did a 4D table lookup, then the closest relevant BRDF samples often would not be found for a particular pair of directions.

Tabularizing the data using the two θ angles and the ϕ difference has problems of its own, however. One shortcoming of this representation is that reciprocity isn't reflected in the mapping: interchanging the two directions maps to a different three-tuple unless both directions happen to have equal θ values.

Another shortcoming is that distance between samples isn't well correlated to actual distance between directions. For example, consider the pair of (θ, ϕ) directions $(.01, 0)$ and $(.01, \pi)$ and the pair of directions $(\pi/2 - .01, 0)$ and $(\pi/2 - .01, \pi)$. Both have $\Delta\phi = \pi$, but the first pair of directions is almost pointing in the same direction (toward the top of the hemisphere), and the second pair is pointing in nearly opposite directions at the horizon. This mapping also doesn't lead to a meaningful distance between samples due to a discontinuity in $\Delta\phi$: given two pairs of directions with equal θ values but where $\Delta\phi = 0$ for the first and $\Delta\phi = 2\pi - .01$ for the second, the mapping doesn't reflect that the two directions are actually close together.

We'd therefore like to use a mapping function for isotropic BRDFs that takes a 4D pair of directions ω_i and ω_o and converts them into a point in a three-dimensional space such that the distance between two points is meaningful, the isotropy of the BRDF is reflected, and reciprocity is represented. For the `IrregIsotropicBRDF`, we'll implement a mapping proposed by Marschner, denoted by ϕ_3 in his thesis (Marschner 1998; Section 5.6.3):

$$m(\theta_i, \phi_i, \theta_o, \phi_o) \rightarrow \left(\sin\theta_i \sin\theta_o, \Delta\phi/\pi, \cos\theta_i \cos\theta_o\right),$$

BxDF 428
IrregIsotropicBRDF 464

where $\Delta\phi = (\phi_i - \phi_o)$, and $\Delta\phi$ is remapped to be in the range $[0, \pi]$. (Thanks to the isotropy assumption, $\Delta\phi$ values in $[\pi, 2\pi]$ are equivalent to $2\pi - \Delta\phi$, which is in $[0, \pi]$.)

This mapping accounts for isotropy and reciprocity, and the distance between two three tuples in the mapping's range has a meaningful relationship to the distance between their respective pairs of directions. The `BRDFRemap()` function implements this mapping.

⟨*BxDF Method Definitions*⟩ +≡
```
Point BRDFRemap(const Vector &wo, const Vector &wi) {
    float cosi = CosTheta(wi), coso = CosTheta(wo);
    float sini = SinTheta(wi), sino = SinTheta(wo);
    float phii = SphericalPhi(wi), phio = SphericalPhi(wo);
    float dphi = phii - phio;
    if (dphi < 0.) dphi += 2.f * M_PI;
    if (dphi > 2.f * M_PI) dphi -= 2.f * M_PI;
    if (dphi > M_PI) dphi = 2.f * M_PI - dphi;
    return Point(sini * sino, dphi / M_PI, cosi * coso);
}
```

Given this remapping function, we can define the structure that stores each BRDF sample. The three-tuple for the remapped directions is stored in `p` and the sample value is stored in `v`.

⟨*Reflection Declarations*⟩ ≡
```
struct IrregIsotropicBRDFSample {
    IrregIsotropicBRDFSample(const Point &pp, const Spectrum &vv)
        : p(pp), v(vv) { }
    Point p;
    Spectrum v;
};
```

When a `IrregIsotropicBRDF` is created by the `MeasuredMaterial::GetBSDF()` method, it is given a pointer to a three-dimensional kd-tree that stores the BRDF samples, where the sample positions are represented with the values returned by `BRDFRemap()`. Note that this kd-tree is managed by the `MeasuredMaterial` and is only used for lookups (i.e., in a read-only manner) by the code here.

⟨*IrregIsotropicBRDF Public Methods*⟩ ≡ 464
```
IrregIsotropicBRDF(const KdTree<IrregIsotropicBRDFSample> *d)
    : BxDF(BxDFType(BSDF_REFLECTION | BSDF_GLOSSY)), isoBRDFData(d) { }
```

⟨*IrregIsotropicBRDF Private Data*⟩ ≡ 464
```
const KdTree<IrregIsotropicBRDFSample> *isoBRDFData;
```

The `IrregIsotropicBRDF::f()` method first remaps the given pair of directions and then performs a series of kd-tree lookups, searching with a progressively wider search radius for nearby points until a few samples have been found. In general, it's preferable to smoothly interpolate between a few BRDF samples rather than just find the single closest sample for the lookup here.

BRDFRemap() 465
BSDF_GLOSSY 428
BSDF_REFLECTION 428
BxDF 428
BxDFType 428
CosTheta() 426
IrregIsotropicBRDF 464
IrregIsotropicBRDFSample 465
KdTree 1029
MeasuredMaterial 489
MeasuredMaterial::
GetBSDF() 489
M_PI 1002
Point 63
SinTheta() 426
Spectrum 263
SphericalPhi() 292
Vector 57

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum IrregIsotropicBRDF::f(const Vector &wo,
                               const Vector &wi) const {
    Point m = BRDFRemap(wo, wi);
    float lastMaxDist2 = .001f;
    while (true) {
        ⟨Try to find enough BRDF samples around m within search radius 466⟩
    }
}
```

`IrregIsoProc` is the callback structure used by the `KdTree::Lookup()` method for each sample within the search radius. (The squared search radius is provided to the `Lookup()` method in `maxDist2`.) If not enough points are found, the squared search distance is doubled, and another lookup is performed.

⟨*Try to find enough BRDF samples around* `m` *within search radius*⟩ ≡ 466
```
IrregIsoProc proc;
float maxDist2 = lastMaxDist2;
isoBRDFData->Lookup(m, proc, maxDist2);
if (proc.nFound > 2 || lastMaxDist2 > 1.5f)
    return proc.v.Clamp() / proc.sumWeights;
lastMaxDist2 *= 2.f;
```

`IrregIsoProc` processes each BRDF sample within the search radius, accumulating a weighted sum of samples close to the lookup point.

⟨*BxDF Local Definitions*⟩ ≡
```
struct IrregIsoProc {
    ⟨IrregIsoProc Public Methods 466⟩
    Spectrum v;
    float sumWeights;
    int nFound;
};
```

⟨*IrregIsoProc Public Methods*⟩ ≡ 466
```
IrregIsoProc() { sumWeights = 0.f; nFound = 0; }
```

BRDF sample values are weighted with an *ad hoc* exponential falloff based on the squared distance between the lookup point and the sample point. The sum of these weights is accumulated so that the final value can be normalized.

⟨*IrregIsoProc Public Methods*⟩ +≡ 466
```
void operator()(const Point &p, const IrregIsotropicBRDFSample &sample,
                float d2, float &maxDist2) {
    float weight = expf(-100.f * d2);
    v += weight * sample.v;
    sumWeights += weight;
    ++nFound;
}
```

BRDFRemap() 465
IrregIsoProc 466
IrregIsoProc::nFound 466
IrregIsoProc::sumWeights 466
IrregIsoProc::v 466
IrregIsotropicBRDFSample 465
IrregIsotropicBRDFSample::v 465
KdTree::Lookup() 1032
Point 63
Spectrum 263
Spectrum::Clamp() 265
Vector 57

8.6.2 REGULAR HALFANGLE FORMAT

There are advantages to using BRDF samples with a regular spacing: looking up the value for a given direction is straightforward, not requiring a spatial data structure or other interpolation scheme, and it can be easier to implement Monte Carlo sampling methods based on regularly tabularized data. In this section we implement a `BxDF` that supports isotropic measured BRDFs stored in a format used by Matusik et al. (2003a). This format is well designed, and a large body of measured data is available for it.[6]

⟨*BxDF Declarations*⟩ +≡
```
class RegularHalfangleBRDF : public BxDF {
public:
    ⟨RegularHalfangleBRDF Public Methods 468⟩
private:
    ⟨RegularHalfangleBRDF Private Data 468⟩
};
```

As with the irregular BRDF data in Section 8.6.1, choosing a remapping of the usual (ω_i, ω_o) representation of pairs of directions is important for accurate representation of BRDFs with tabularized data. As was first observed by Rusinkiewicz (1998), the usual $(\theta_i, \phi_i, \theta_o, \phi_o)$ representation of directions has the shortcoming that important features in the BRDF can move in an irregular manner through its 4D space. Consider for example a surface with perfect specular reflection: as (θ_o, ϕ_o) vary, the specular reflection direction moves accordingly in (θ_i, ϕ_i). A different mapping of the pair of directions that instead ensured that this feature was aligned to one of the 4D coordinate axes would lead to more accurate lookup and interpolation of tabularized reflection data and would also be better suited to efficiently fitting data with basis functions.

Rusinkiewicz proposed a mapping based on the half-angle vector $\omega_h = \omega_i \widehat{+} \omega_o$ and the difference vector ω_d,

$$m(\theta_i, \phi_i, \theta_o, \phi_o) \rightarrow (\theta_h, \phi_h, \theta_d, \phi_d) \qquad [8.10]$$

where the difference vector is found by finding the rotation that aligns ω_h with the (0, 0, 1) direction and then applying that rotation to ω_i. Figure 8.23 shows the effect of remapping the Torrance–Sparrow BRDF using Equation (8.10). Notice how the specular peak remains aligned along the $+z$ axis, even as the outgoing direction varies.

`RegularHalfangleBRDF` takes a pointer to a 3D table of sampled isotropic BRDF values, indexed by $(\sqrt{\theta_h}, \theta_d, \phi_d)$, where the number of samples in each direction is respectively `nThetaH`, `nThetaD`, and `nPhiD`. Due to the assumption of isotropy, the ϕ_h term can be dropped. The square root of θ_h is taken in order to slightly increase the sampling rate for θ_h values close to zero. The most rapid variation of sample values for many BRDFs is in this region of the domain, as specular highlights are generally well aligned to the ω_h axis after the remapping, so small changes in θ_h close to zero can represent significant changes in the function's value.

`BxDF` 428

`RegularHalfangleBRDF` 467

6 Matusik et al.'s measurement data for approximately 100 different materials is available from *www.merl.com/brdf*.

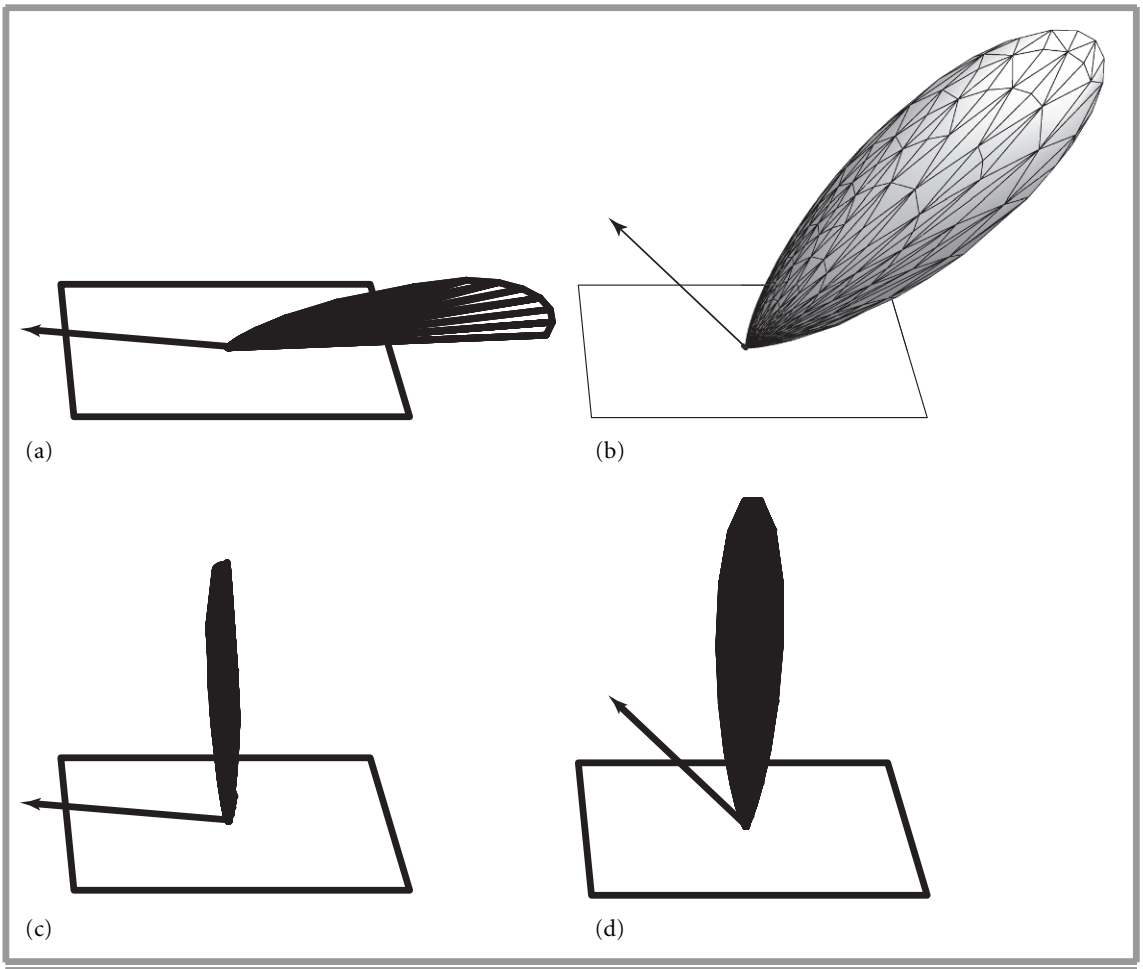


Figure 8.23: (a, b) The Torrance–Sparrow BRDF for two outgoing directions ω_o. (c, d) Torrance–Sparrow BRDF for the same two outgoing directions, after remapping the function with Equation (8.10). Note that the specular lobe remains aligned with the $+z$ axis even as the outgoing direction changes.

⟨*RegularHalfangleBRDF Public Methods*⟩ ≡ 467
```
RegularHalfangleBRDF(const float *d, uint32_t nth, uint32_t ntd,
                     uint32_t npd)
    : BxDF(BxDFType(BSDF_REFLECTION | BSDF_GLOSSY)), brdf(d),
      nThetaH(nth), nThetaD(ntd), nPhiD(npd) { }
```

⟨*RegularHalfangleBRDF Private Data*⟩ ≡ 467
```
const float *brdf;
const uint32_t nThetaH, nThetaD, nPhiD;
```

BSDF_GLOSSY 428
BSDF_REFLECTION 428
BxDF 428
BxDFType 428
RegularHalfangleBRDF 467

To compute the BRDF's value for a given pair of directions, it's necessary to apply the mapping given by Equation (8.10) to remap the directions to the half-angle coordi-

nate system, compute the corresponding indices, and to then perform the table lookup. `RegularHalfangleBRDF` returns the single closest sample value without further interpolation. Trilinear interpolation could be easily implemented, though the available data sets tend to have high sampling rates, making interpolation unnecessary.

⟨*BxDF Method Definitions*⟩ +≡
```
Spectrum RegularHalfangleBRDF::f(const Vector &wo,
                                 const Vector &wi) const {
    ⟨Compute ωh and transform ωi to halfangle coordinate system 469⟩
    ⟨Compute index into measured BRDF tables 469⟩
    return Spectrum::FromRGB(&brdf[3*index]);
}
```

The half-angle direction ω_h is easily computed as usual. The rotation matrix that rotates ω_h to be aligned with (0, 0, 1) is then computed by finding the product of the matrices for rotating $-\phi_h$ about the z axis and then rotating $-\theta_h$ about the y axis.[7] The code in this fragment directly computes the entries of the product of these two matrices, the rows of which are represented here as `whx`, `why`, and `wh`, respectively. Multiplying ω_i by this matrix gives ω_d.

⟨*Compute ωh and transform ωi to halfangle coordinate system*⟩ ≡ 469
```
Vector wh = Normalize(wi + wo);
float whTheta = SphericalTheta(wh);
float whCosPhi = CosPhi(wh), whSinPhi = SinPhi(wh);
float whCosTheta = CosTheta(wh), whSinTheta = SinTheta(wh);
Vector whx(whCosPhi * whCosTheta, whSinPhi * whCosTheta, -whSinTheta);
Vector why(-whSinPhi, whCosPhi, 0);
Vector wd(Dot(wi, whx), Dot(wi, why), Dot(wi, wh));
```

Given the difference vector ω_d, we can directly compute the corresponding spherical coordinates. Due to the BRDF's reciprocity, its value for $\omega_d = (\theta_d, \phi_d)$ is equal to its value for $\omega_d = (\theta_d, \phi_d + \pi)$, so if `wDiffPhi` is greater than π, then π is subtracted from it. Integer indices for each value are computed by the fragment ⟨*Compute indices* `whThetaIndex, wdThetaIndex, wdPhiIndex`⟩ and then the index for a 3D array lookup is stored in `index`.

⟨*Compute* `index` *into measured BRDF tables*⟩ ≡ 469
```
float wdTheta = SphericalTheta(wd), wdPhi = SphericalPhi(wd);
if (wdPhi > M_PI) wdPhi -= M_PI;
⟨Compute indices whThetaIndex, wdThetaIndex, wdPhiIndex 470⟩
int index = wdPhiIndex + nPhiD * (wdThetaIndex + whThetaIndex * nThetaD);
```

Given the three angles, their respective indices into the BRDF data table are found using the `REMAP()` macro, which maps the given floating-point value `V`, expected to be between 0 and `MAX` to an integer from 0 to `COUNT-1`.

M_PI 1002
RegularHalfangleBRDF 467
RegularHalfangleBRDF::brdf 468
RegularHalfangleBRDF::nPhiD 468
RegularHalfangleBRDF::nThetaD 468
Spectrum 263
Spectrum::FromRGB() 277
Vector 57

7 This matrix could be computed more efficiently, without using the trigonometric functions used here, with the technique described by Akenine-Möller and Hughes (1999). However, the value of θ_h is also needed below for the table lookup, so here we just compute the matrix using the trigonometric approach.

⟨*Compute indices* whThetaIndex, wdThetaIndex, wdPhiIndex⟩ ≡ 469

```
#define REMAP(V, MAX, COUNT) \
    Clamp(int((V) / (MAX) * (COUNT)), 0, (COUNT)-1)
int whThetaIndex = REMAP(sqrtf(max(0.f, whTheta / (M_PI / 2.f))),
                         1.f, nThetaH);
int wdThetaIndex = REMAP(wdTheta, M_PI / 2.f, nThetaD);
int wdPhiIndex = REMAP(wdPhi, M_PI, nPhiD);
#undef REMAP
```

FURTHER READING

Phong (1975) developed an early empirical reflection model for glossy surfaces in computer graphics. Although not reciprocal or energy conserving, it was a cornerstone of the first synthetic images of non-Lambertian objects. The Torrance–Sparrow microfacet model is described in Torrance and Sparrow (1967); it was first introduced to graphics by Blinn (1977), and a variant of it was used by Cook and Torrance (1981, 1982). The Oren–Nayar Lambertian model is described in their 1994 SIGGRAPH paper (Oren and Nayar 1994).

Hall's book collected and described the state of the art in physically based surface reflection models for graphics in 1989; it remains a seminal reference (Hall 1989). It discusses the physics of surface reflection in detail, with many pointers to the original literature and with many tables of useful measured data about reflection from real surfaces.

Moravec (1981) was the first to apply a wave optics model to graphics. This area has also been investigated by Bahar and Chakrabarti (1987). Beckmann and Spizzichino (1963) developed an early physical optics model of surface reflection, which was used by Kajiya (1985) to derive an anisotropic reflection model for computer graphics. Beckmann and Spizzichino's work was built upon by He et al. (1991), who developed a sophisticated reflection model that modeled a variety of types of surface reflection, and Stam (1999), who applied wave optics to model diffraction effects.

Nayar, Ikeuchi, and Kanade (1991) have shown that some reflection models based on physical (wave) optics have substantially similar characteristics to those based on geometric optics. The geometric optics approximations don't seem to cause too much error in practice, except on very smooth surfaces. This is a helpful result, giving experimental basis to the general belief that wave optics models aren't usually worth their computational expense for computer graphics applications.

The effect of the polarization of light is ignored in pbrt, although for some scenes it can be an important effect; see, for example, the paper by Tannenbaum, Tannenbaum, and Wozny (1994) for information about how to extend a renderer to account for this effect. Similarly, the fact that indices of refraction of real-world objects usually vary as a function of wavelength is also not modeled here; see both Section 11.8 of Glassner's book (1995) and Devlin et al.'s survey article for information about these issues and references to previous work (Devlin et al. 2002). Fluorescence, where light is reflected at different wavelengths than the incident illumination, is also not modeled by pbrt; see Glassner (1994) and Wilkie et al. (2006) for more information on this topic.

M_PI 1002
RegularHalfangleBRDF::
nPhiD 468
RegularHalfangleBRDF::
nThetaD 468
RegularHalfangleBRDF::
nThetaH 468

Notable anisotropic BRDF models for computer graphics include those of Poulin and Fournier (1990), Ward (1992), and Schlick (1993). Schlick's model is both computationally efficient and easy to use with importance sampling for Monte Carlo integration. Other good references for anisotropic models are papers by Lu, Koenderink, and Kappers (1999) and Ashikhmin and Shirley (2000, 2002).

Ashikhmin, Premoze, and Shirley (2000) developed techniques for computing self-shadowing terms for arbitrary microfacet distributions, without requiring the assumptions that Torrance and Sparrow did. Their solutions cannot be evaluated in closed form, but must be approximated numerically. Weyrich et al. (2009) have developed methods to infer a microfacet distribution that matches a measured or desired reflection distribution. Remarkably, they show that physical surfaces that match the desired reflection distribution can be manufactured, with a good match to the desired distribution.

Stam (2001) also derived a generalization of the Cook–Torrance model for transmission, and more recently Walter et al. (2007) revisited this problem, addressing issues of finding appropriate microfacet and self-shadowing functions for transmission and finding effective importance sampling approaches for Monte Carlo integration. There are a number of important subtleties that are not accounted for in the simple approach taken in the `BRDFToBTDF` class in this chapter.

Improvements in data-acquisition technology have led to increasing amounts of measured real-world BRDF data, even including BRDFs that are spatially varying (sometimes called "bidirectional texture functions," BTFs). Matusik et al. (2003a, 2003b) assembled a large database of measured isotropic BRDF data; the `RegularHalfangleBRDF` uses data stored in their representation. See Müller et al. (2005) for a survey of recent work in BRDF measurement. Sun et al. (2007) measured BRDFs as they change over time—for example, due to paint drying, a wet surface becoming dry, or dust accumulating.

Fitting measured BRDF data to parametric reflection models is a difficult problem. Rusinkiewicz (1998) made the influential observation that reparameterizing the measured data can make it substantially easier to compress or fit to models; this topic has been further investigated by Stark et al. (2005). Ngan et al. (2005) analyzed the effectiveness of a variety of BRDF models for fitting measured data and showed that models based on the half-angle vector, rather than a reflection vector, tended to be more effective. See also the paper on this topic by Edwards et al. (2005).

Our approach for inteprolating irregular BRDF measurements in Section 8.6 is *ad hoc*. See Zickler et al. (2005) for a better-grounded method based on using radial basis functions (RBFs)—they use them to interpolate irregularly sampled 5D spatially varying BRDFs. More recently, Weistroffer et al. (2007) have shown how to efficiently represent scattered reflectance data with RBFs without needing to resample it to have regular spacing and Wang et al. (2008) demonstrated a successful approach for acquiring spatially varying anisotropic BRDFs.

Kajiya and Kay (1989) developed a reflection model for hair based on a model of individual hairs as cylinders with diffuse and glossy reflection properties. Their model determines the overall reflection from these cylinders, accounting for the effect of variation in surface normal over the hemisphere along the cylinder. See also the paper by Banks (1994), which discusses shading models for 1D primitives like hair. More recently,

BRDFToBTDF 431
RegularHalfangleBRDF 467

Goldman (1997) developed a probabilistic shading model that models reflection from collections of short hairs, and Marschner et al. (2003) developed a more accurate model of light scattering from long hair. A number of other specific types of surfaces have received specific attention from researchers, leading to specialized reflection models being developed for them, including Marschner et al.'s work on rendering wood (Marschner et al. 2005), Sattler et al.'s approach for rendering cloth (Sattler et al. 2003), Irawan's work on cloth rendering (Irawan 2008), and Günther et al.'s investigation of car paint (Günther et al. 2005).

A number of researchers have investigated BRDFs based on modeling the small-scale geometric features of a reflective surface. This work includes the computation of BRDFs from bump maps by Cabral, Max, and Springmeyer (1987), Fournier's normal distribution functions (Fournier 1992), and Westin, Arvo, and Torrance (1992), who applied Monte Carlo ray tracing to statistically model reflection from microgeometry and represented the resulting BRDFs with spherical harmonics.

EXERCISES

❶ 8.1 A consequence of Fermat's principle from optics is that light traveling from a point p_1 in a medium with index of refraction η_1 to a point p_2 in a medium with index of refraction η_2 will follow a path that minimizes the time to get from the first point to the second point. Snell's law can be shown to follow from this fact directly.

Consider light traveling between two points p_1 and p_2 separated by a planar boundary. The light could potentially pass through the boundary while traveling from p_1 to p_2 at any point on the boundary (see Figure 8.24, which shows two such possible points p' and p''). Recall that the time it takes light to travel between two points in a medium with a constant index of refraction is proportional to the distance between them times the index of refraction in the medium. Using this fact, show that the point p' on the boundary that minimizes the total time to travel from p_1 to p_2 is the point where $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$.

❸ 8.2 Read the papers of Wolff and Kurlander (1990) and Tannenbaum, Tannenbaum, and Wozny (1994) and apply some of the techniques described to modify `pbrt` to model the effect of light polarization. Set up scenes and render images of them that demonstrate a significant difference when polarization is accurately modeled.

❸ 8.3 Construct a scene with an actual geometric model of a rough plane with a large number of mirrored microfacets and illuminate with an area light source.[8] Place the camera in the scene such that a very large number of microfacets are in each pixel's area, and render images of this scene using hundreds or thousands of pixel samples. Compare the result to using a flat surface with a

8 An area light and not a point or directional light is necessary due to subtleties in how lights are seen in specular surfaces. With the light transport algorithms used in `pbrt`, infinitesimal point sources are never visible in mirrored surfaces. This is a typical limitation of ray-tracing renderers and usually not bothersome in practice.

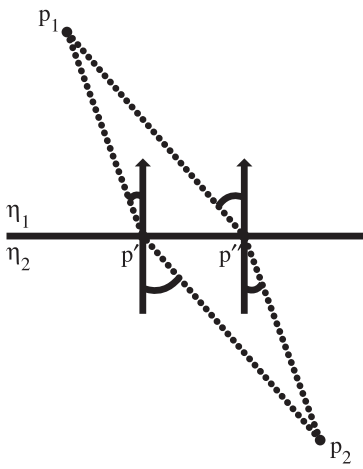


Figure 8.24: Derivation of Snell's Law. Snell's law can be derived using Fermat's principle, which says that light will follow the path that takes the least amount of time to pass between two points. The angle of refraction θ at the boundary between two media can thus be shown to be the one that minimizes the time spent going from p_1 to a point p on the boundary plus the time spent traveling the distance from that point to p_2.

microfacet-based BRDF model. How well can you get the two approaches to match if you try to tune the microfacet BRDF parameters? Can you construct examples where images rendered with the true microfacets are actually visibly more realistic due to better modeling the effects of masking, self-shadowing, and interreflection between microfacets?

❷ 8.4 Implement the approach for rendering transmission through microfacet-based surfaces described by Walter et al. (2007), including their self-shadowing term and importance sampling techniques. Compare images rendered with this approach to images rendered using the `BRDFToBTDF` adapter class defined in this chapter; does the more sound physical basis for their method lead to visually different images?

❸ 8.5 Replace the implementation of the `IrregIsotropicBRDF` with a more soundly grounded scattered BRDF interpolation technique such as the one described by Zickler et al. (2005) or Weistroffer et al. (2007). What sort of visual differences are evident with images rendered with your implementation compared to the current implementation in `pbrt`? How does the execution time change when your implementation is used for rendering images with irregularly sampled measured BRDF data?

❸ 8.6 Extend `pbrt` to be able to more accurately render interesting surfaces like wood (Marschner et al. 2005), cloth (Sattler et al. 2003), or car paint (Günther et al. 2005). Render images showing better visual results than when existing reflection functions in `pbrt` are used for these effects.

BRDFToBTDF 431
IrregIsotropicBRDF 464

3 8.7 Implement a simulation-based approach to modeling reflection from complex microsurfaces, such as the one described by Westin, Arvo, and Torrance (1992). Modify `pbrt` so that you can provide a description of the microgeometry of a complex surface (like cloth, velvet, etc.), fire rays at the geometry from a variety of incident directions, and record the distribution and throughput for the rays that leave the surface. (You will likely need to modify the `PathIntegrator` from Chapter 15 to determine the distribution of outgoing light.) Record the distribution in a three-dimensional table if the surface is isotropic or a four-dimensional table if it is anisotropic, and use the table to compute BRDF values for rendering images. Demonstrate interesting reflection effects from complex surfaces using this approach. Investigate how the size of the table and the number of samples taken to compute entries in the table affect the accuracy of the final result.

PathIntegrator 766