Fresnel Surfaces

Steve Rotenberg

CSE168: Rendering Algorithms

UCSD, Spring 2014

Augustin-Jean Fresnel

- 1788-1827
- Fresnel was a French engineer and physicist
- He made many contributions to optics and the wave theory of light
- Perhaps best known for inventing the Fresnel lens, which was originally used for lighthouses
- The Fresnel Equations are used in ray tracing to determine the amount of light reflected and refracted when it hits a surface

Dielectric Materials

- Pure dielectric materials (insulators) are usually optically clear
- Examples include air, water, glass, and diamond
- Their surface optics are typically described by an *index of refraction n* which describes how much light gets bent when entering the surface
- The phase velocity of light v_{phase} in the medium is related to the refractive index n by $n=c/v_{phase}$ where c is the speed of light in a vacuum

Index of Refraction

- The index of refraction *n* for some common materials:

– Vacuum:	1.0
– Air:	1.0003
– Water:	1.333
– Ice:	1.309
– Glass:	1.4 – 1.8
– Diamond:	2.42

- These are based on a light wavelength of 589 nm
- The index of refraction actually varies with wavelength, giving rise to *optical dispersion*, which is responsible for the colors of rainbows and from prisms
- For today, we will ignore the dispersion effects and assume that the index of refraction is constant… (we will look at dispersion in a later lecture)

Dielectrics and Refraction

- When an incident light ray hits the interface between two dielectric media (such as from air to glass), the beam is split into a *reflected* ray and a *refracted* (or *transmitted*) ray

n normal

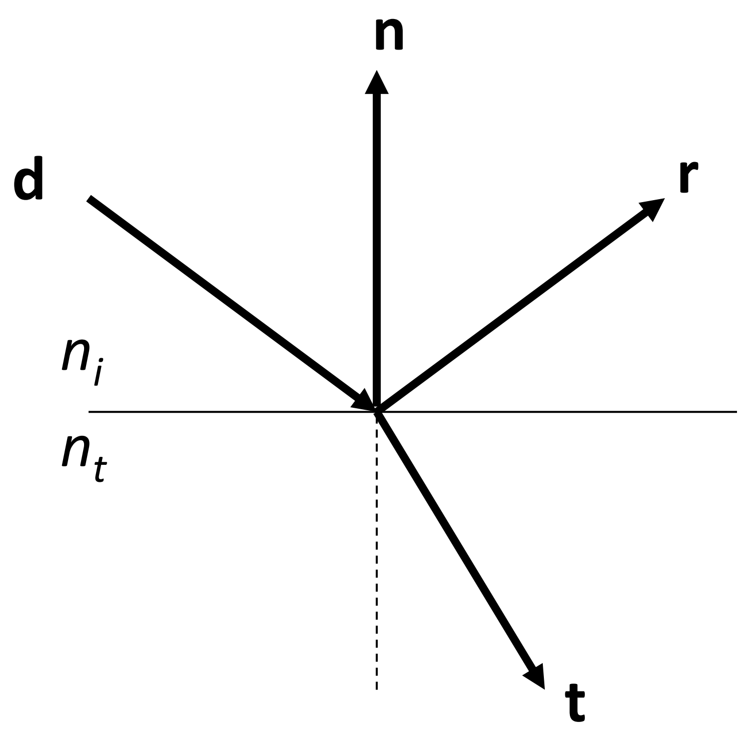
d incoming direction

r reflected ray

t transmitted ray

n_i initial index of refraction

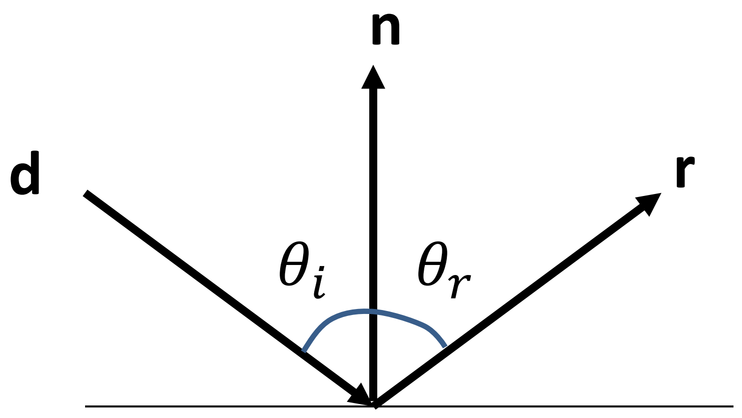
n_t final index of refraction



Reflections

- For reflections, the angle of incidence θ_i equals the angle of reflection θ_r
- To compute the reflected vector, we can use:

$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}$$

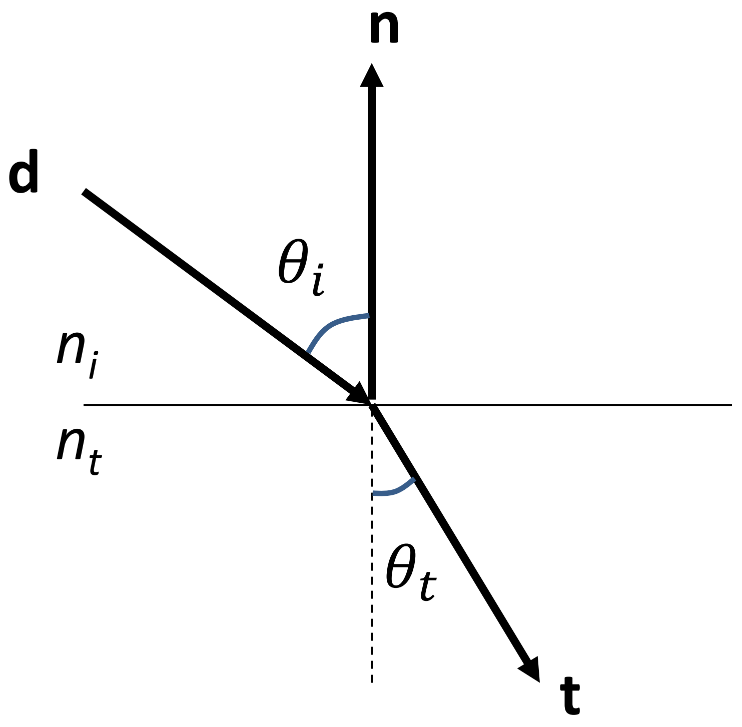


NOTE: some sources draw this with the **d** vector reversed, leading to a slightly different result

Snell's Law

- For refraction, the angle of transmission θ_t is related to the angle of incidence θ_i by Snell's Law:

$$n_i \sin\theta_i = n_t \sin\theta_t$$



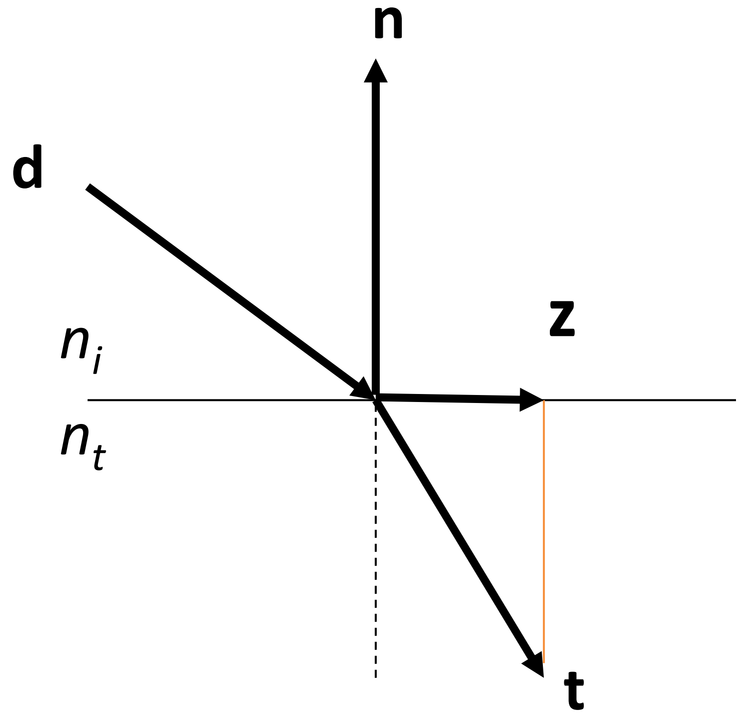
NOTE: named after Dutch astronomer Willebrord Snellius (1580-1626), although it was first described by Ibn Sahl of Baghdad in 984 A.D.

Transmission Vector

- To compute the transmission vector **t**:

$$\mathbf{z} = \frac{n_i}{n_t}(\mathbf{d} - (\mathbf{d} \cdot \mathbf{n})\mathbf{n})$$

$$\mathbf{t} = \mathbf{z} - \left(\sqrt{1 - |\mathbf{z}|^2}\right)\mathbf{n}$$

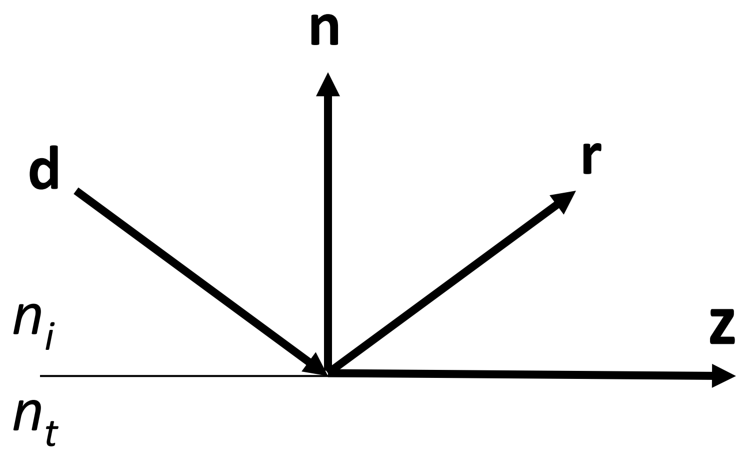


Total Internal Reflection

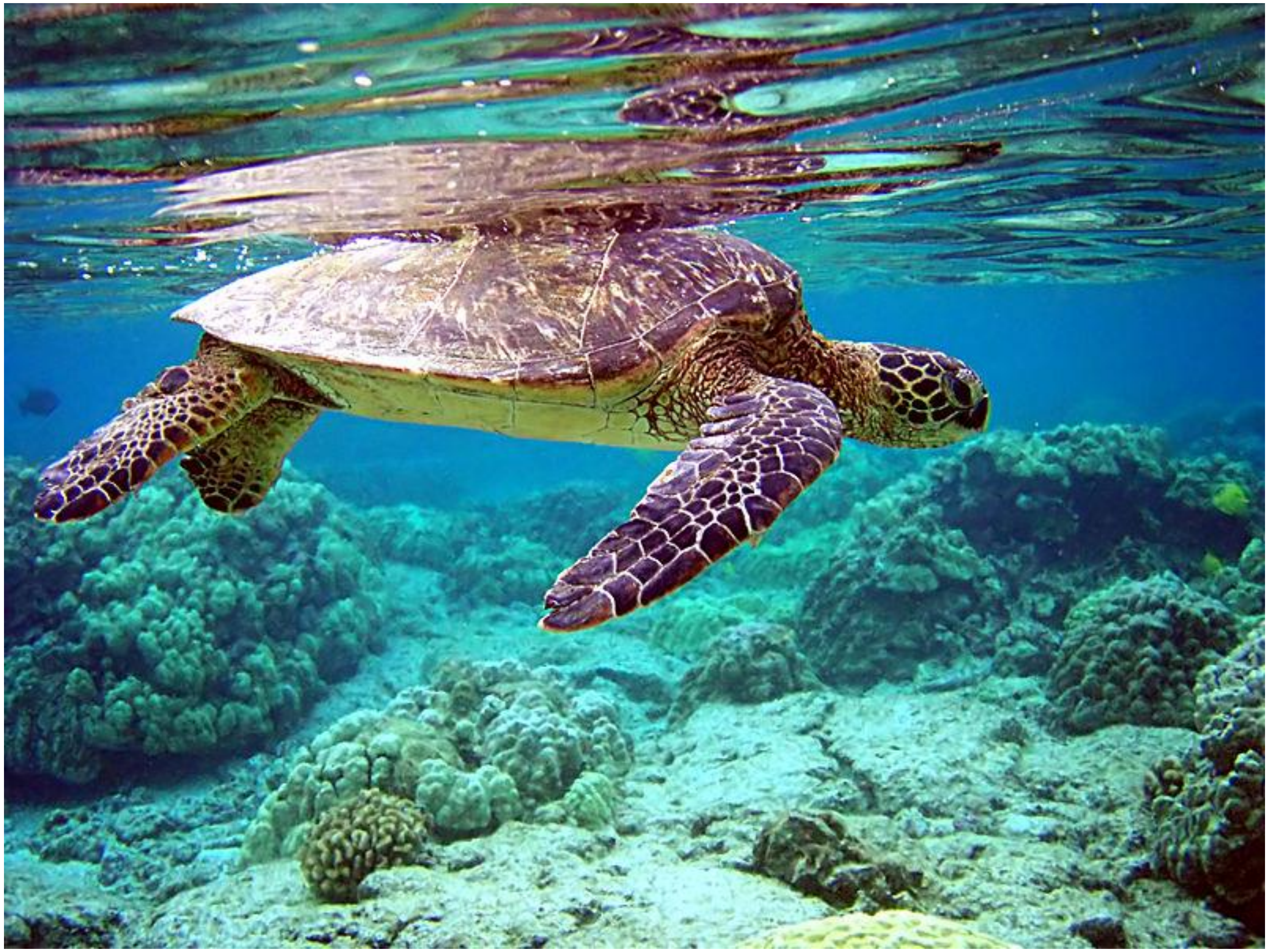
- When light traveling in a material with a high index of refraction hits a material with a low index of refraction at a shallow angle, we can get a total internal reflection
- In this case, 100% of the light is reflected and there is no refraction
- This effect can be visible when one is scuba diving and looks up at the water surface. One can only see rays refracting to the outside world in a circular area on the water surface above
- Total internal reflection can be detected when the magnitude squared of the **z** vector is greater than 1, causing the square root operation to become undefined

$$\mathbf{z} = \frac{n_i}{n_t}(\mathbf{d} - (\mathbf{d} \cdot \mathbf{n})\mathbf{n})$$

$$\mathbf{t} = \mathbf{z} - \left(\sqrt{1 - |\mathbf{z}|^2}\right)\mathbf{n}$$



Total Internal Reflection



Total Internal Reflection Critical Angle

Dielectric Interfaces

- Once again, we say that when light hits the interface between two dielectric media, it will either reflect 100% or it will be split into a reflected ray and a transmitted ray
- If it is split, then how much light goes in each direction? Or what percentage is reflected and what percentage is transmitted?

Fresnel Equations

- The Fresnel equations can be used to determine the proportion of the light reflected f_r and transmitted f_t when a ray hits an interface between two dielectrics
- They describe separate formulas for the parallel r_{par} and perpendicular r_{perp} polarized light, but these are usually averaged into a single set of values

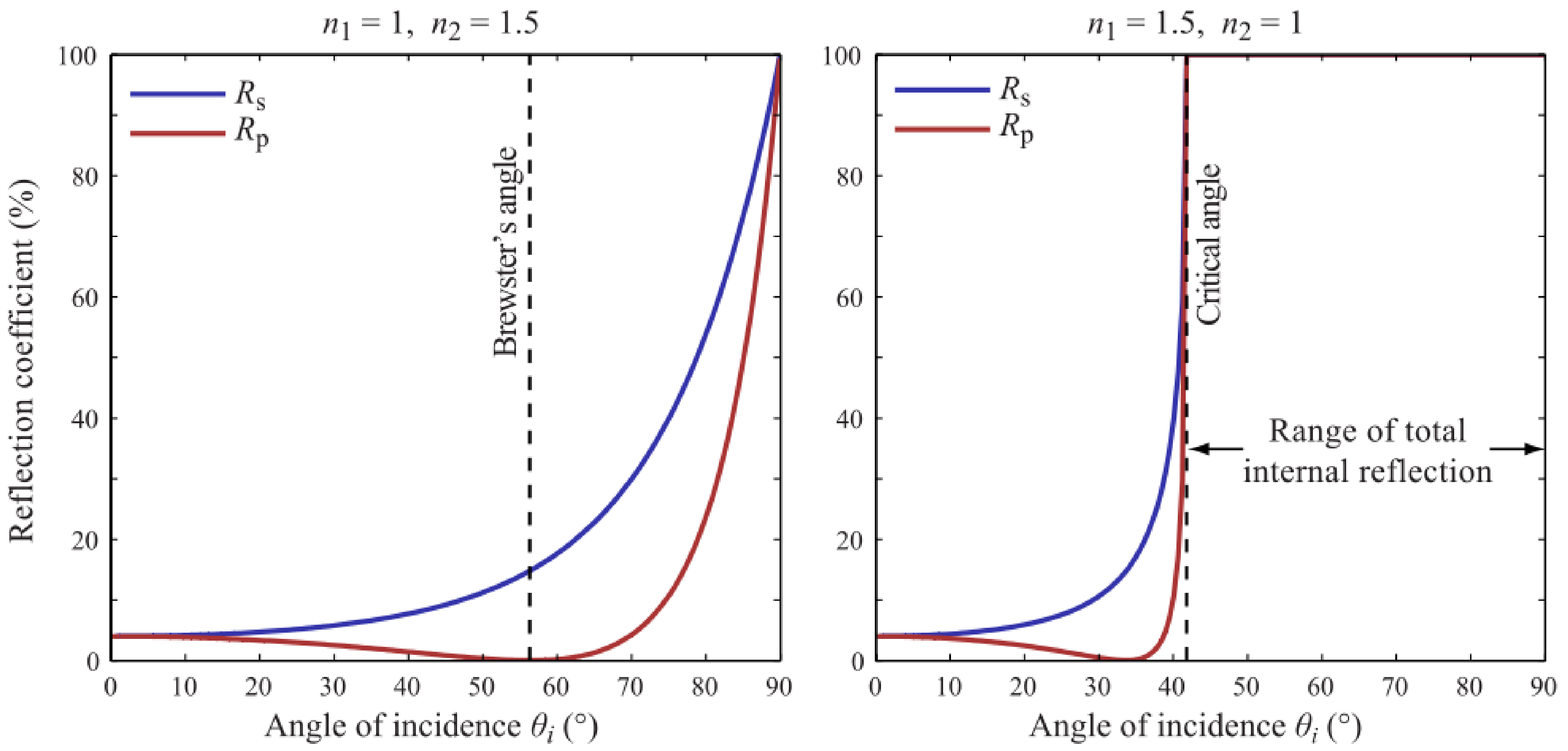
$$r_{par} = \frac{n_t(\mathbf{n} \cdot \mathbf{d}) - n_i(\mathbf{n} \cdot \mathbf{t})}{n_t(\mathbf{n} \cdot \mathbf{d}) + n_i(\mathbf{n} \cdot \mathbf{t})}$$

$$r_{perp} = \frac{n_i(\mathbf{n} \cdot \mathbf{d}) - n_t(\mathbf{n} \cdot \mathbf{t})}{n_i(\mathbf{n} \cdot \mathbf{d}) + n_t(\mathbf{n} \cdot \mathbf{t})}$$

$$f_r = \frac{1}{2}\left({r_{par}}^2 + {r_{perp}}^2\right)$$

$$f_t = 1 - f_r$$

Fresnel Equations



Source: Wikipedia

Fresnel Equations

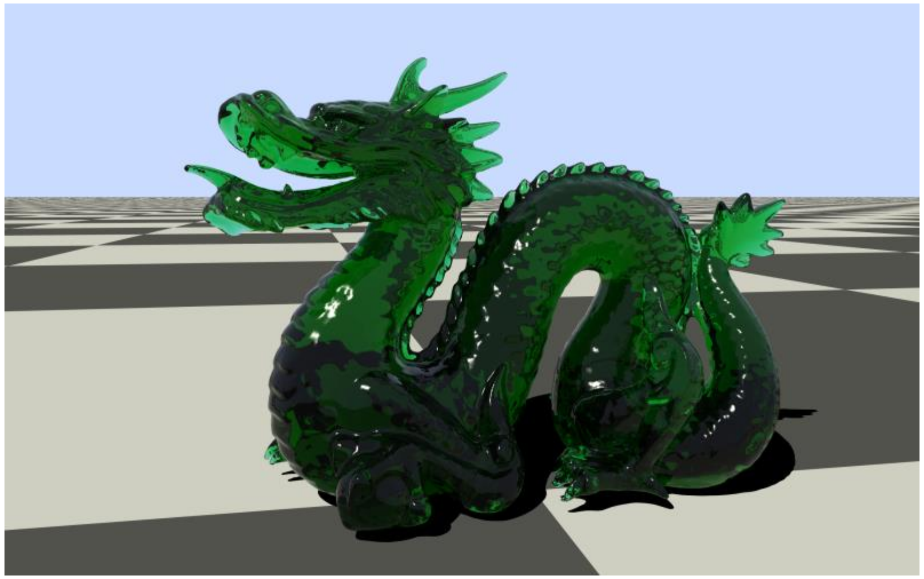
- From examining the equations and the graphs, we can learn a couple important things:
  - The reflectivity of transparent materials like glass doesn’t drop to 0 when we are looking directly at it (for glass, the reflectivity at the front surface is about 4%)
  - As we look at the surface from more edge on, the reflectivity increases, and approaches 100% as we approach 90 degrees

Dielectric Interfaces

- These formulas tell us what we need to know in order to ray trace dielectric surfaces like glass, diamond, and water
- They tell us what directions the reflection **r** and transmission **t** vectors go, plus the percentage of the light that goes in each direction (f_r & f_t)

Absorption

- Transparent materials can absorb light
- The rate of absorption can vary significantly with wavelength, leading to coloration of the light that makes it through



Beer-Lambert Law

- The Beer-Lambert law (sometimes just called Beer’s law) relates the absorption of light to the properties of the material through which the light is traveling
- The law states that there is a logarithmic dependence between the transmission T of light through a substance and the product of the absorption coefficient α, and the distance ℓ light travels through the material

$$T = I/I_0 = e^{-\alpha \ell}$$

NOTE: named after German physicist August Beer (1825-1863)

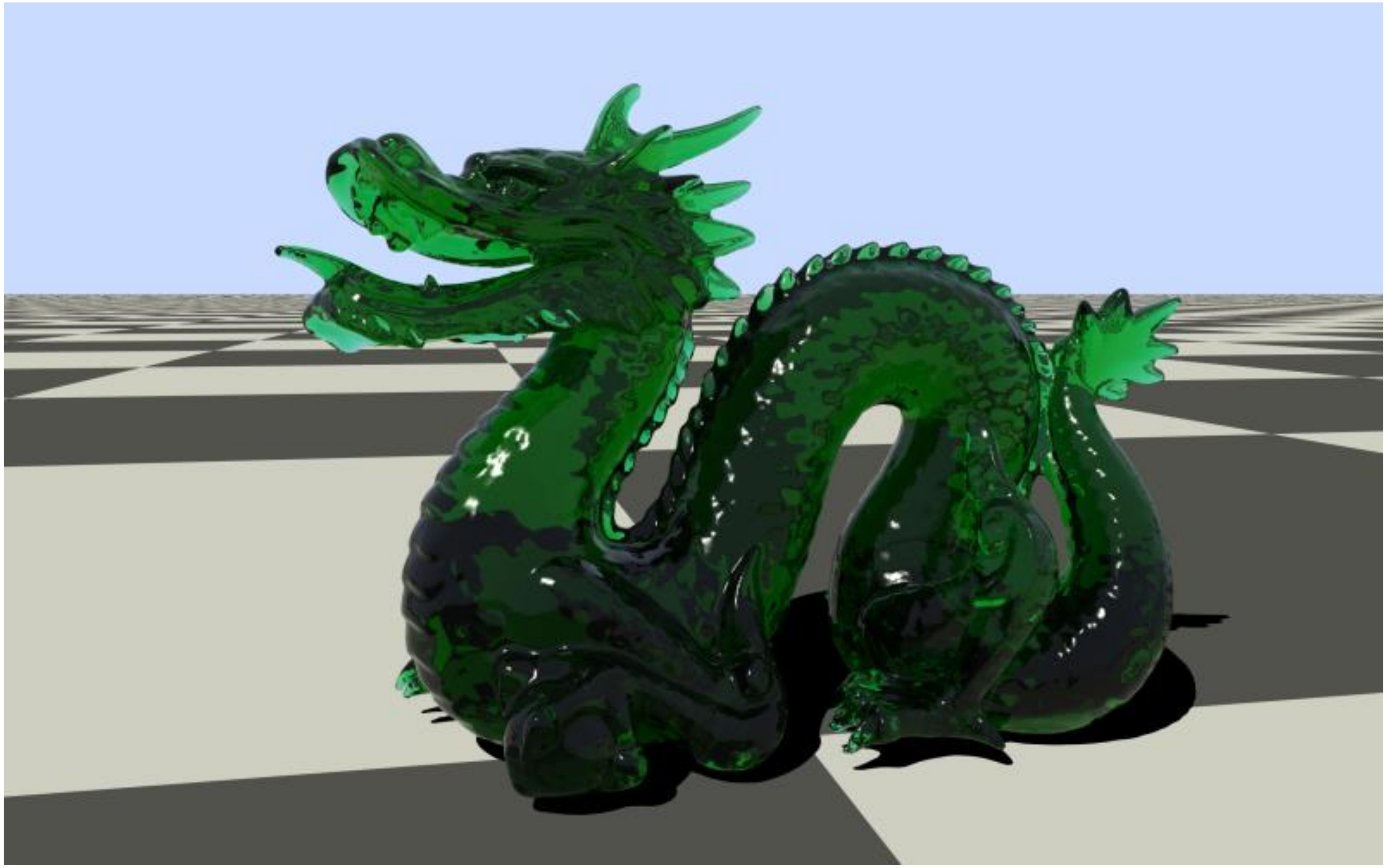
Beer-Lambert Law

- The absorption coefficient α varies with wavelength, leading to the colorization of the transmitted light
- Due to the actual particles absorbing light and their concentration levels, we can get a wide range of colors and levels of absorption
- The Beer-Lambert law only handles transparent materials with absorbing particles and isn't accurate for materials that contain scattering particles which lead to translucency

Beer-Lambert Law

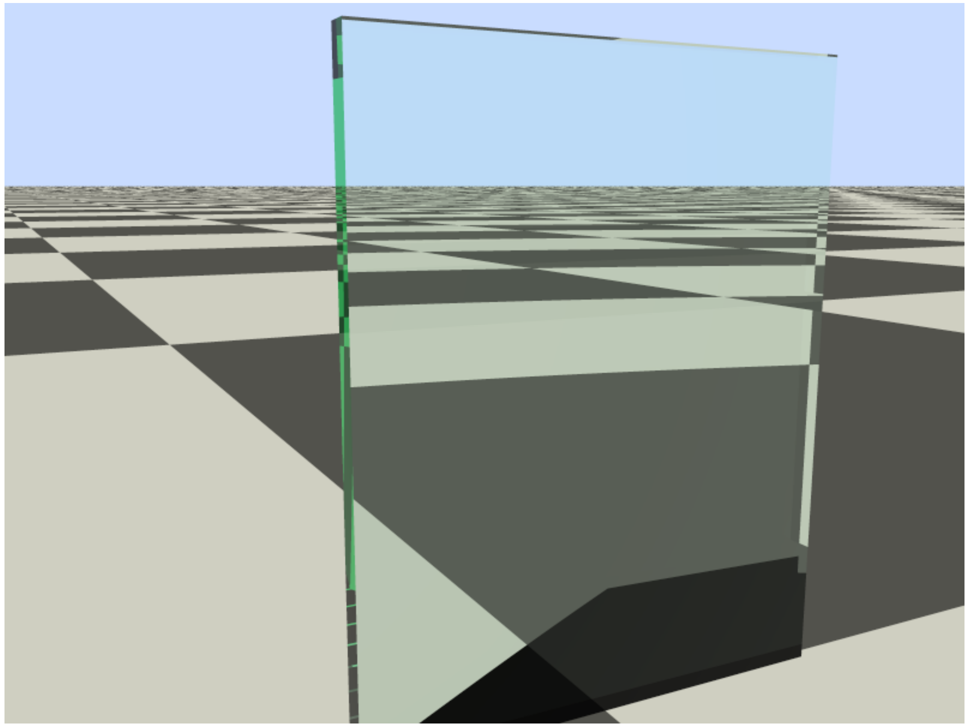
- In computer graphics, we generally don’t have precise absorption coefficient data, so we usually just model absorption by specifying coefficients for the red, green, and blue absorption for the material
- To compute the absorption for a ray passing through a transparent material, we compute the distance l that the ray travels in the material and then evaluate the equation for each coefficient

Absorption



Absorption

- Note that the green tint visible on the edge of panes of glass is due to a combination of absorption and total internal reflection



Metals

Metal Materials

- With conductors such as metals, incident light waves excite oscillations of free electrons in the surface, inducing a secondary radiated wave which is the reflection…
- Anyway, metals reflect light and don't transmit it
- However, metals can still be described with a index of refraction, but it is a complex number- traditionally broken into *n* for the real part and *k* for the imaginary part

Fresnel Equations for Metals

- The Fresnel equations governing the reflectance of metals are slightly more complicated and the following approximations are valid in most cases:

$$r_{par}{}^2 = \frac{\left(n_t{}^2 + k_t{}^2\right)(\mathbf{n} \cdot \mathbf{d})^2 + 2n_t(\mathbf{n} \cdot \mathbf{d}) + 1}{\left(n_t{}^2 + k_t{}^2\right)(\mathbf{n} \cdot \mathbf{d})^2 - 2n_t(\mathbf{n} \cdot \mathbf{d}) + 1}$$

$$r_{perp}{}^2 = \frac{\left(n_t{}^2 + k_t{}^2\right) + (\mathbf{n} \cdot \mathbf{d})^2 + 2n_t(\mathbf{n} \cdot \mathbf{d})}{\left(n_t{}^2 + k_t{}^2\right) + (\mathbf{n} \cdot \mathbf{d})^2 - 2n_t(\mathbf{n} \cdot \mathbf{d})}$$

$$f_r = \frac{1}{2}\left(r_{par}{}^2 + r_{perp}{}^2\right)$$

Fresnel Metals

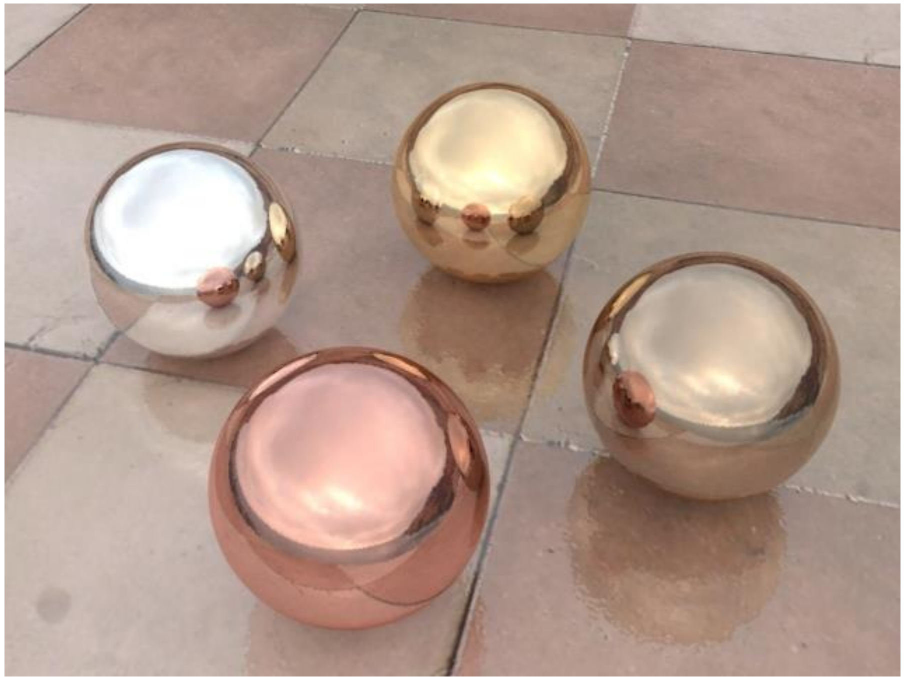
- The complex index of refraction for some common metals are:

– Steel:	n=2.485,	k=3.433
– Silver:	n=0.177,	k=3.638
– Gold:	n=0.37,	k=2.82
– Copper:	n=0.617,	k=2.63

Fresnel Metals

- We can calculate that the reflectance of copper when looking straight on is about 74%, but this increases up to 100% for edge-on views
- Another important property of metals is that this reflectance is wavelength dependent
- This causes the reflections to be tinted by the color of the metal (this is particularly noticeable with colored metals like copper and gold)
- This contrasts the behavior of dielectrics like colored plastic, which don’t actually tint the reflected color off the surface (the visible color comes from scattering and absorption below the actual surface)

Colored vs. Uncolored Reflections



Fresnel Surfaces

- We refer to metal and dielectrics as *Fresnel surfaces*, as light interaction with the surface can be described by the Fresnel equations
- Remember also, that we have only been discussing *optically smooth* (perfectly flat) surfaces
- Although the world contains lots of complex materials, the Fresnel surfaces actually make up just about the simplest and most fundamental types of optical surfaces
- Most other surfaces are built up from multiple components such as Fresnel surfaces, pigments, scattering particles, as well as having complex *microgeometry*. These things lead to far more complex light interaction with most real surfaces. We will examine some of these in later lectures
- Still, the idealized Fresnel surfaces make up an important class of useful surfaces for rendering

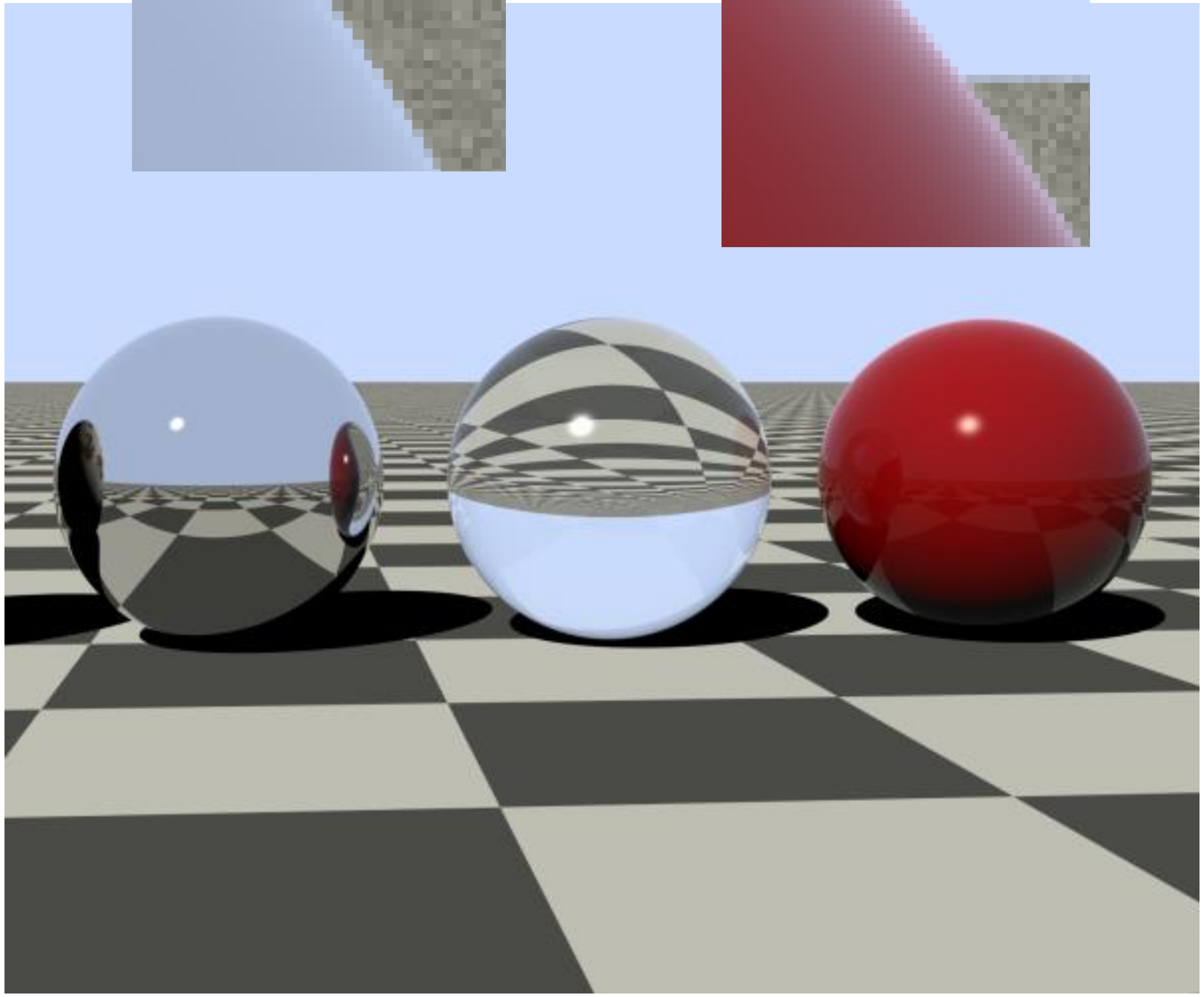
Fresnel Effect

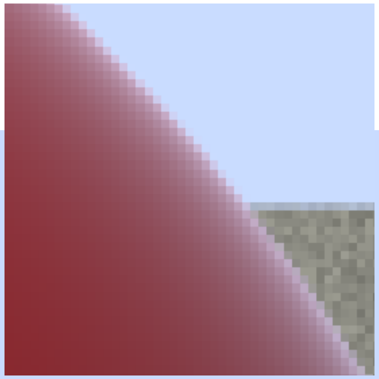
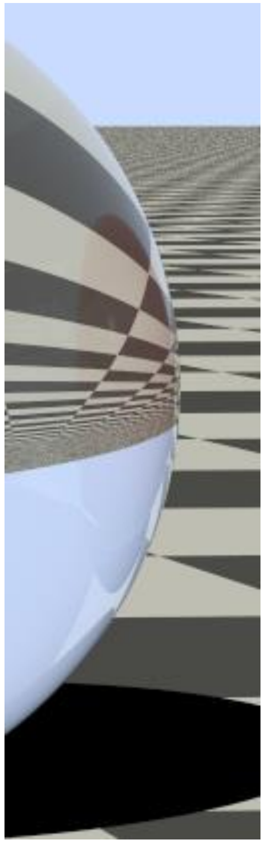
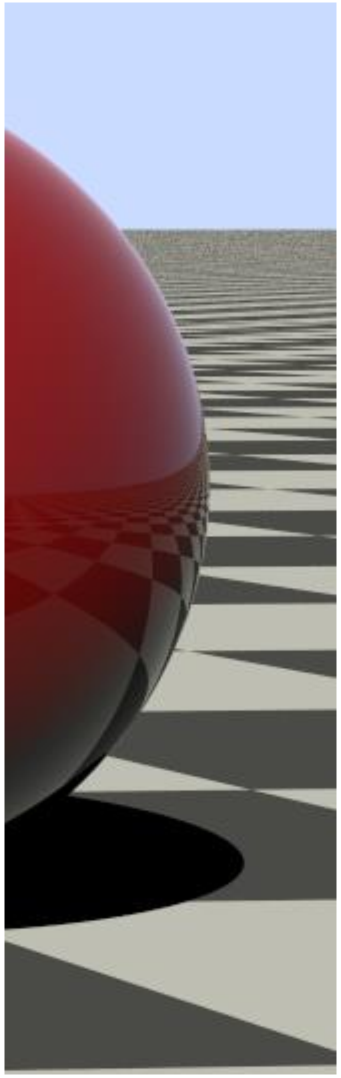
- In computer graphics, we often refer to the ‘Fresnel effect’, which is the phenomenon that materials become more reflective as you view them from more edge-on directions
- This effect can be seen in many everyday materials, not just the idealized smooth dielectric and metals
- For example, it can easily be seen in car paint, due to the clear coatings and wax layers on the surface

Fresnel Effect

- The Fresnel effect can also be easily seen when looking at a swimming pool or other calm body of water
- When you look straight down, you can see the bottom of the pool, but when you look across, you see a strong reflection
- The Fresnel effect can even be observed to a lesser degree on rough surfaces
- It is visible all around us every day, but most people are not consciously aware of it and are surprised when it is pointed out
- However, if we leave it out of computer renderings, it causes things to look noticeably wrong, although most casual observers can’t tell exactly why it looks wrong

Fresnel Effect



Ray Tracing Reflection and Refraction

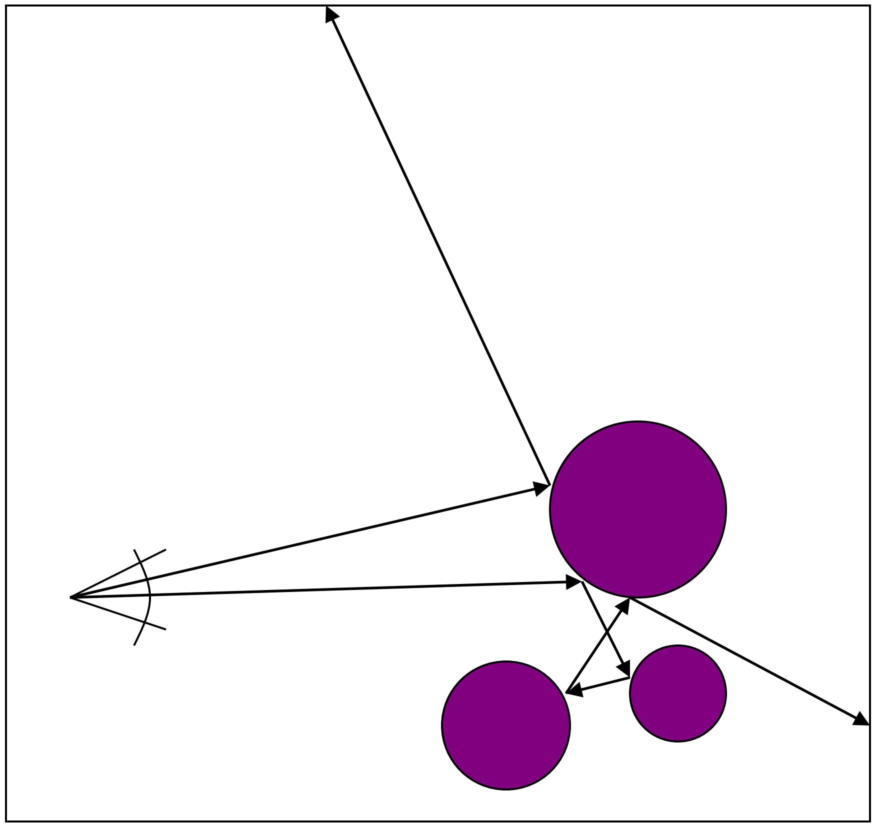
Ray Tracing

- When we shoot a primary ray out from the camera and it hits a surface, we then proceed to compute the color visible at the surface
- For diffuse surfaces, we could just compute a color based on the incoming light (as we did in project 1)
- For reflective and refractive surfaces, however, we can shoot secondary rays off to determine the reflected/refracted color

Reflections

- Lets say we just want to render a shiny metal like silver
- When the ray hits the surface, we compute the Fresnel reflectivity based on the geometry of the intersection. Let’s say that we come up with a reflectivity of 95%
- We then generate a secondary ray and shoot that off into the scene
- If the secondary ray hits a diffuse surface, we compute the color as in project 1
- This color then gets scaled by the reflectivity of the silver surface to determine the final color

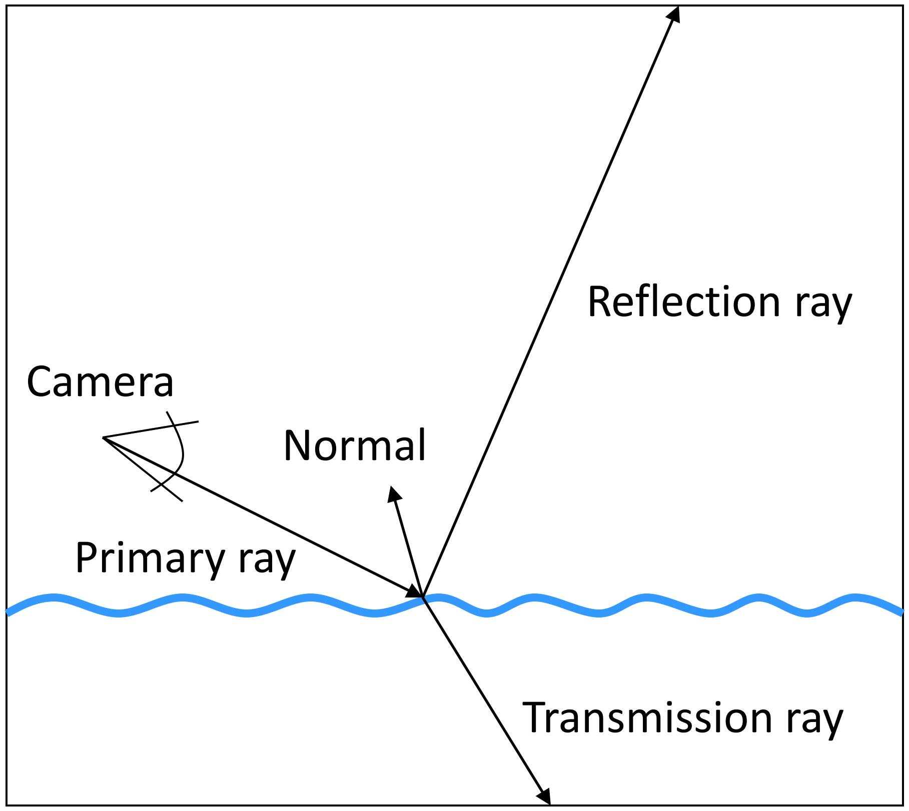
Reflections



Refraction

- If our primary ray hits a dielectric surface, we compute both the intensity of the reflection f_r and the transmission f_t according to the Fresnel equations
- We shoot off a reflection ray and refraction ray and determine the colors of the surfaces that they hit
- The final color is then the sum of the two, each scaled by the appropriate coefficient (f_r or f_t)

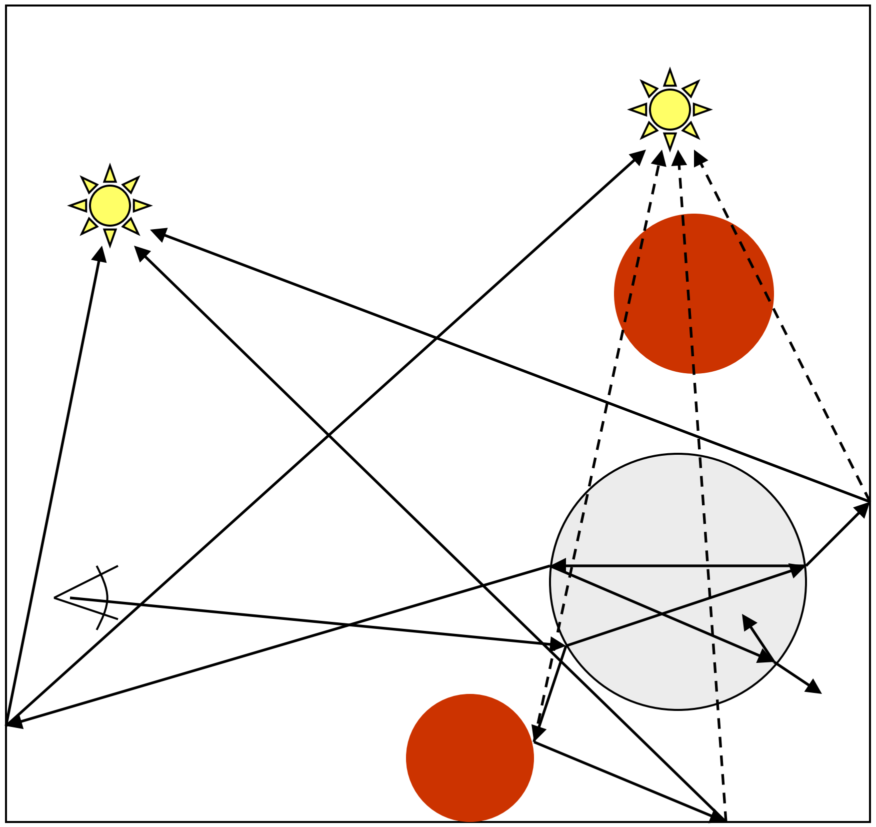
Reflection/Refraction

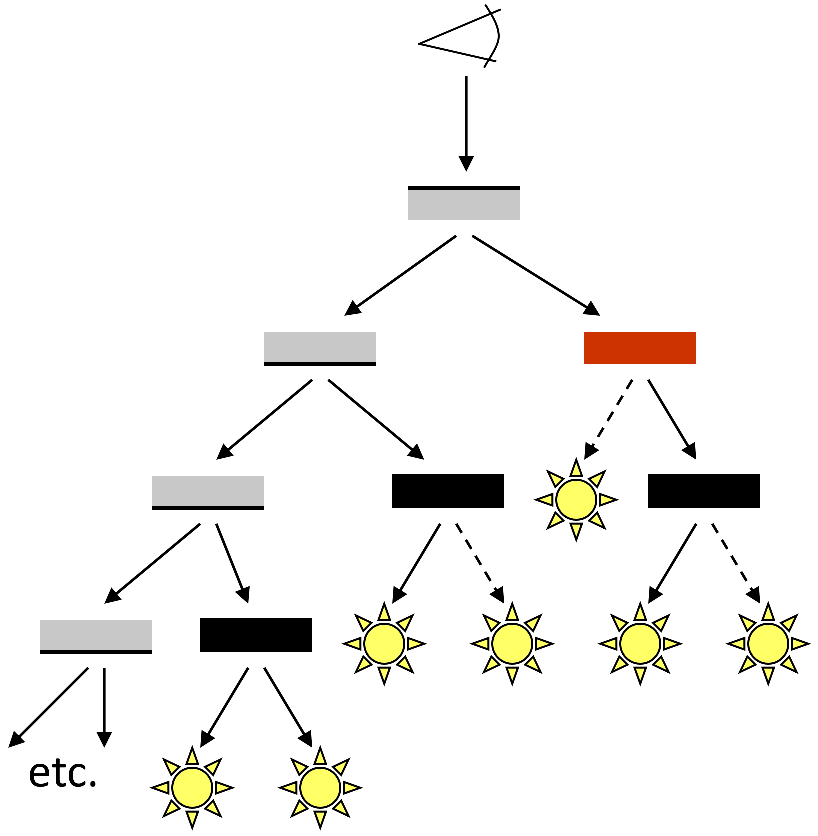


Recursive Ray Tracing

- When we shoot out a ray, we ultimately expect a color in return
- This can be done recursively as rays hit multiple metal and dielectric surfaces
- Therefore, a single primary ray may generate several new rays that can form a tree-like structure
- To prevent the system from getting stuck in an infinite loop (for example, if we’re in a room with mirrors on every wall), we typically put some sort of upper limit on the depth of the recursion

Recursive Ray Tracing





Surface Offsetting

- We want to avoid mathematical roundoff problems that might lead to the reflected rays hitting the original surface, so we add a slight offset to the secondary ray origin
- For reflected rays, we push the origin to slightly above the surface:

```
ray.Origin = hit.Position + 0.001*hit.Normal
```

- For a transmitted ray, we push it slightly below the surface

```
ray.Origin = hit.Position - 0.001*hit.Normal
```

RayTrace Class

- I suggest creating a RayTrace class that handles all of the actual ray tracing and shading
- That way, you don’t have to stuff all of that in the Camera class like you probably did in project 1
- This is also a convenient place to put various settings such as the maximum recursion depth and some other properties that we may add later
- It is also a convenient place to put some statistics functions, such as counting the total number of primary, secondary, or shadow rays used to render the image

RayTrace Class

```
class RayTrace {
public:
        RayTrace(Scene &s);

        bool TraceRay(const Ray &ray, Intersection &hit, int depth=1);

private:
        Scene *Scn;

        // Settings
        int MaxDepth;

        // Statistics
        int PrimaryRays;
        int SecondaryRays;
        int ShadowRays;
};
```

RayTrace Class

```
bool RayTrace::TraceRay(const Ray &ray, Intersection &hit, int depth) {
        // Find ray intersection
        if(Scn->Intersect(ray,hit)==false) {
                hit.Shade=Scn->GetSkyColor();
                return false;
        }

        // Compute shade due to lighting
        hit.Shade=Color::BLACK;
        for( each light ) {
                - Compute light.Illuminate()
                - if(illum==0 or hit.Normal doesn't face light) continue;
                - Create shadow ray towards light
                - if(shadow ray isn't blocked) add contribution from this light to hit.Shade
        }
        if(Depth==MaxDepth) return true;

        // Compute shade due to reflections/refractions
        for(each secondary ray) {
                - Generate newray & newhit
                - Compute ray intensity (based on $f_r$ or $f_t$, for example)
                - TraceRay(newray, newhit, depth+1);
                - hit.Shade += intensity*newhit.Shade;
        }
        return true;
}
```