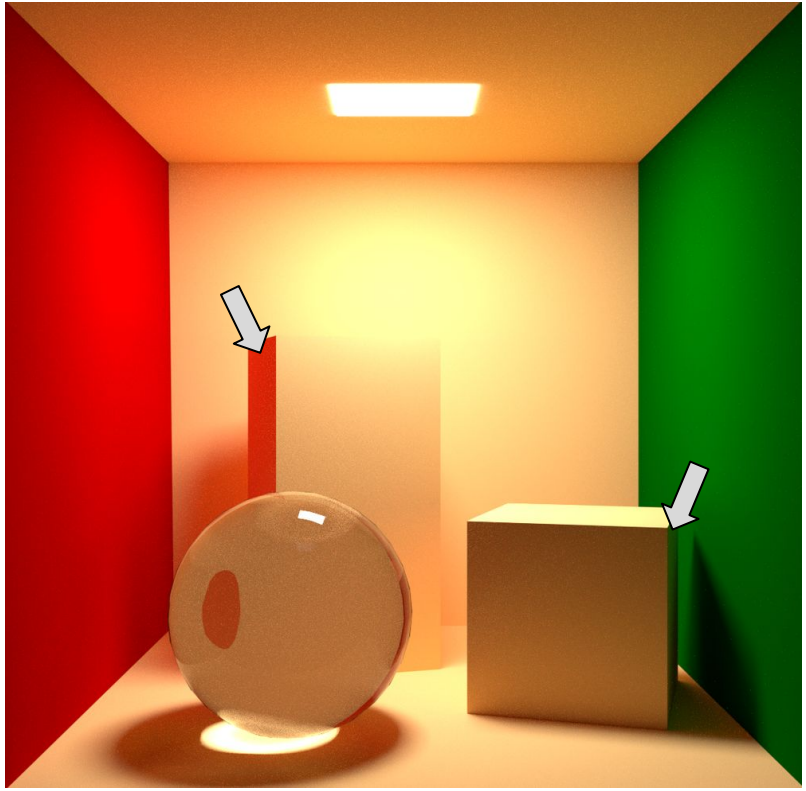
Monte-Carlo Path Tracing Transparency/Translucency

Max Ismailov, Ryan Lingg

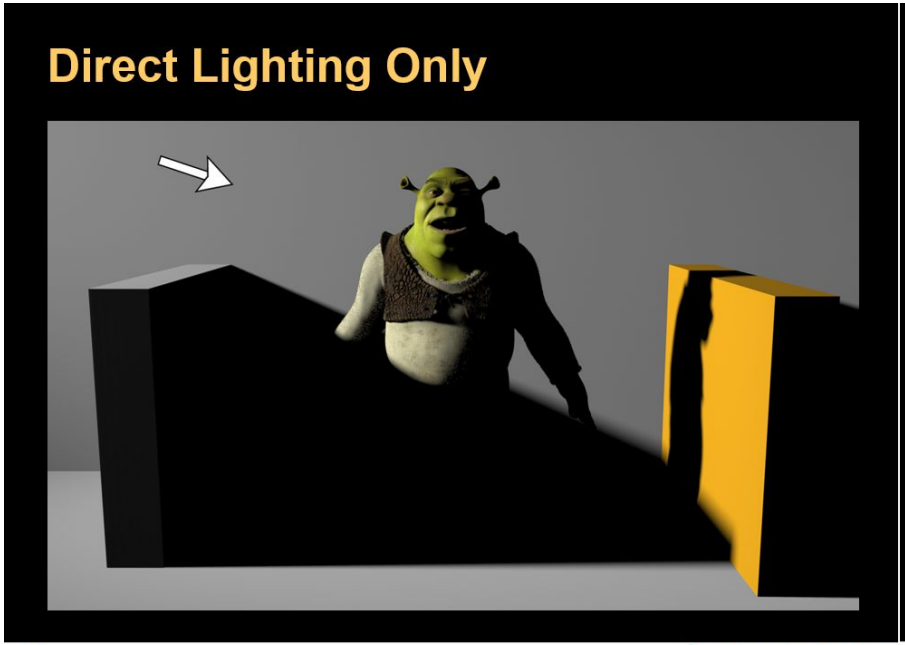
Light Does a Lot of Reflecting

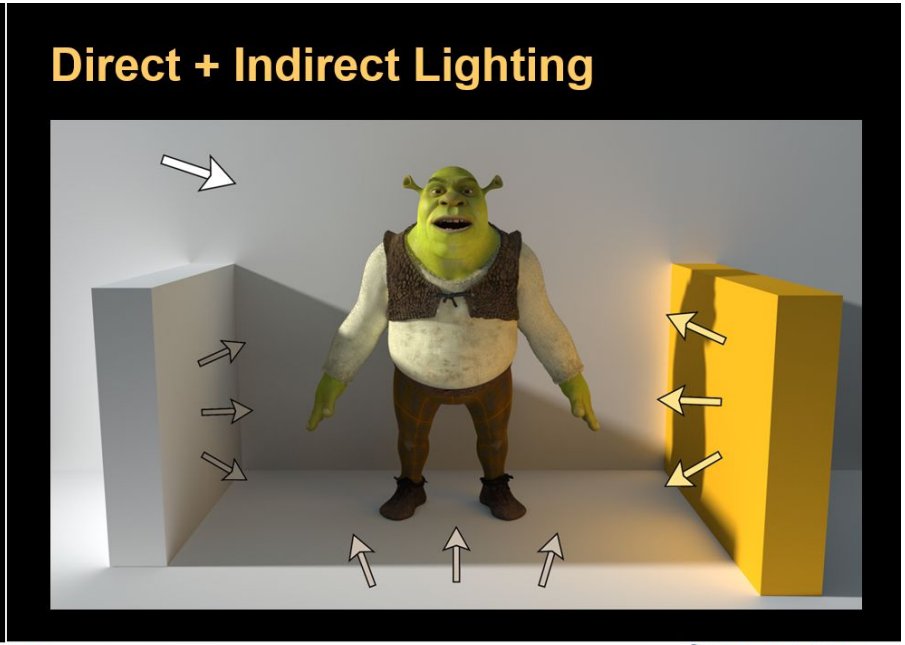
- Our raytracer only supported direct lighting
- Objects that are not light sources emit light!

➔ We need more attention to realism than direct light sources can offer



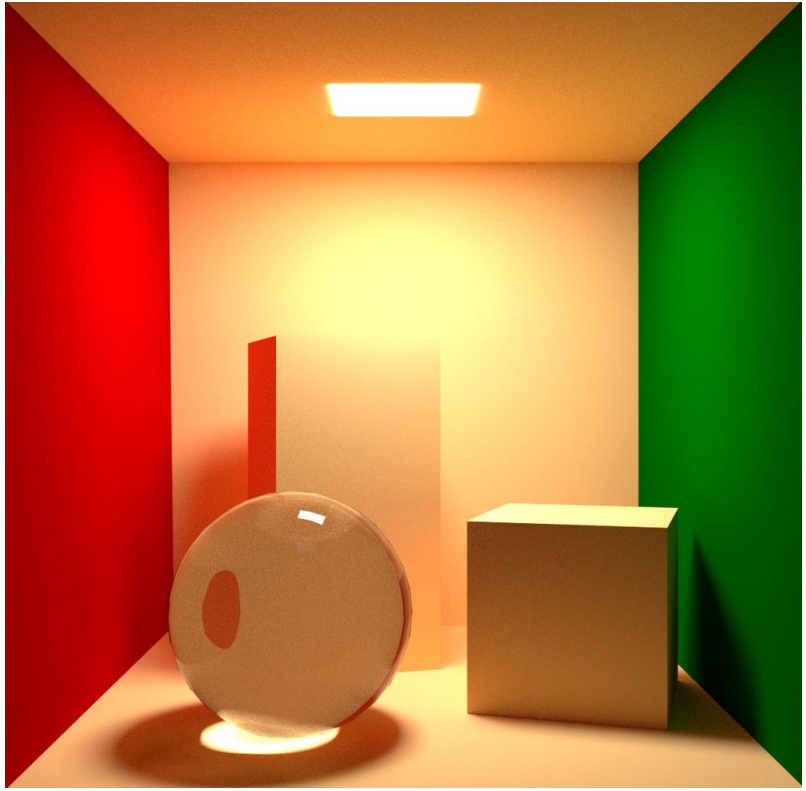
Global Illumination: Shrek





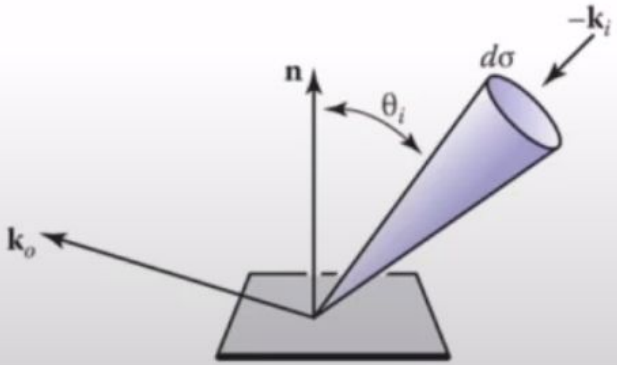
Tools to Solve this Problem

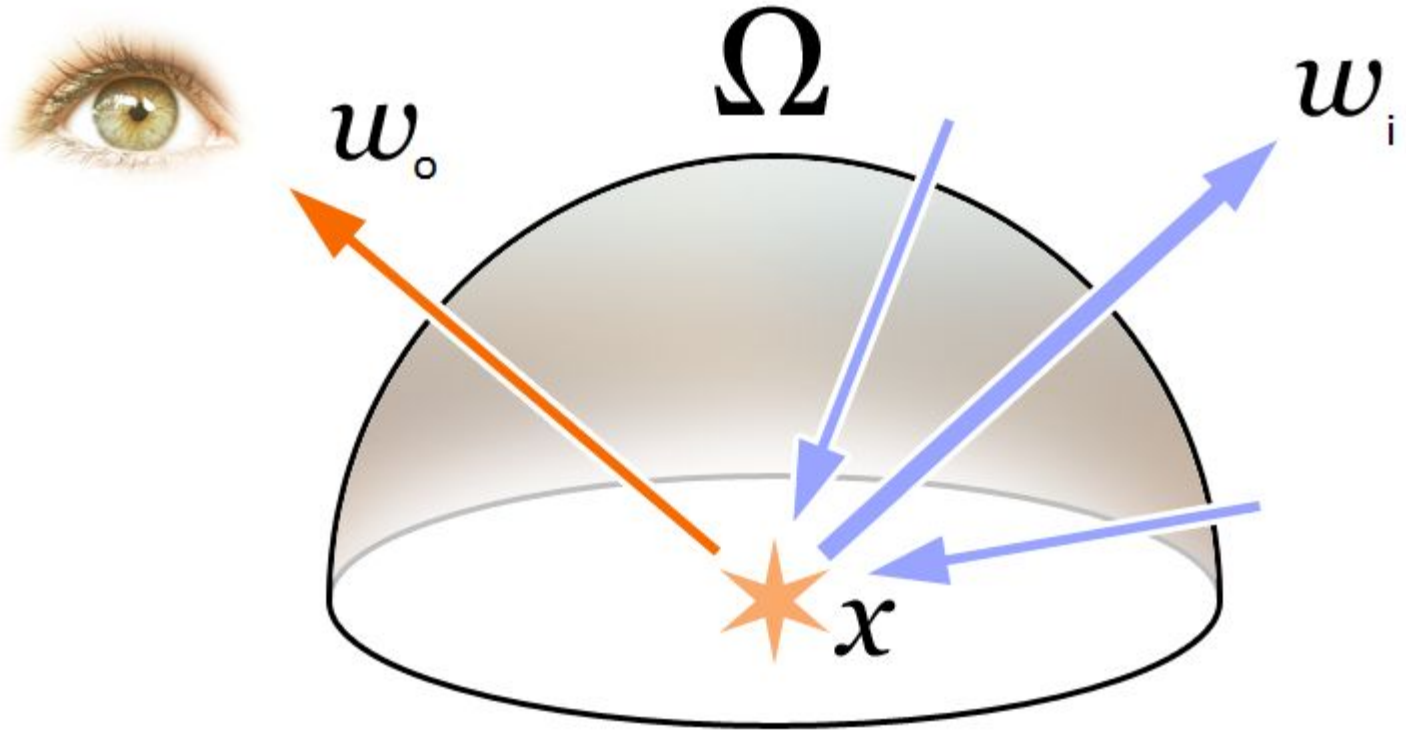
1. Appropriate model of reflectance for objects
2. A way to aggregate incoming light sources



Formally: We're Solving the Rendering Equation!

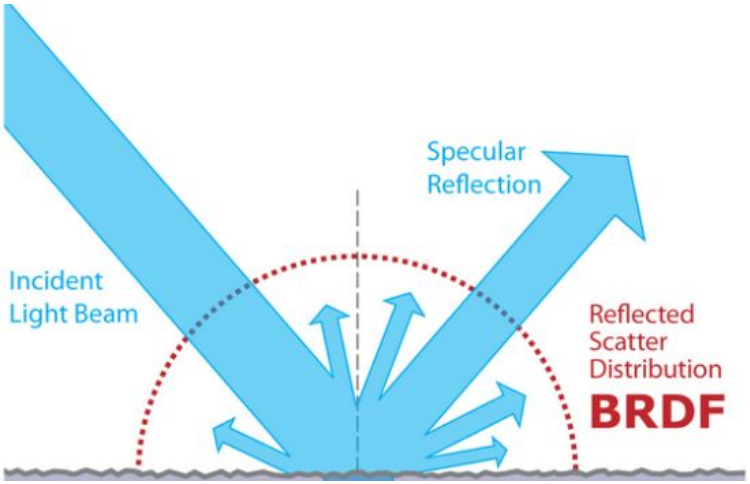
$$\underbrace{L_s(\mathbf{k}_o)}_{\text{Light in Outgoing Direction}} = \underbrace{\int_{\mathbf{k_i}}}_{\text{Aggregating Light Sources}} \underbrace{\rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k_i})}_{\text{Reflectance Model for Incoming Light}} \cos\theta_i \, d\sigma_i$$



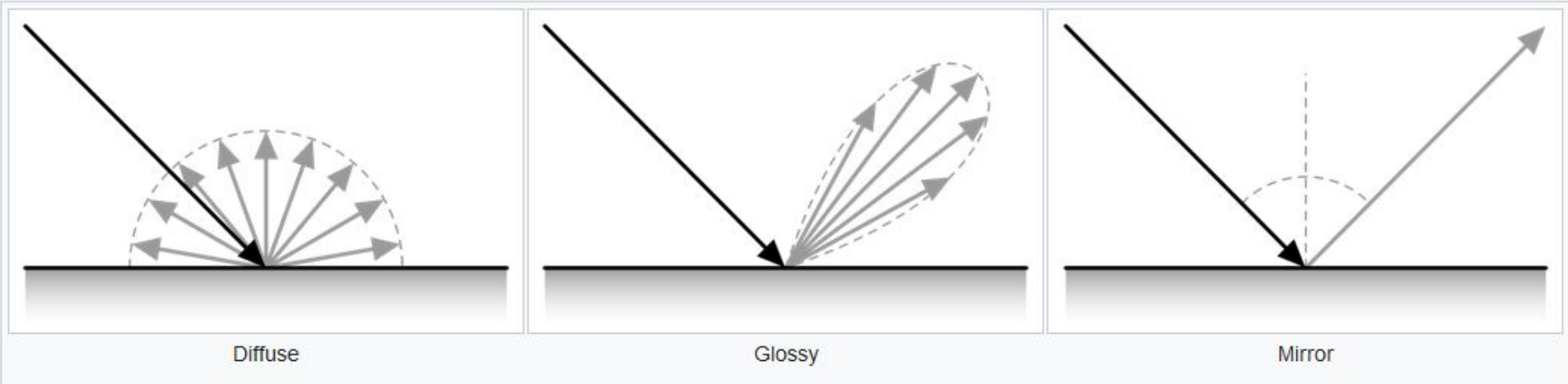

Ω
w_o
w_i
x

Reflectance Models - BRDF’s

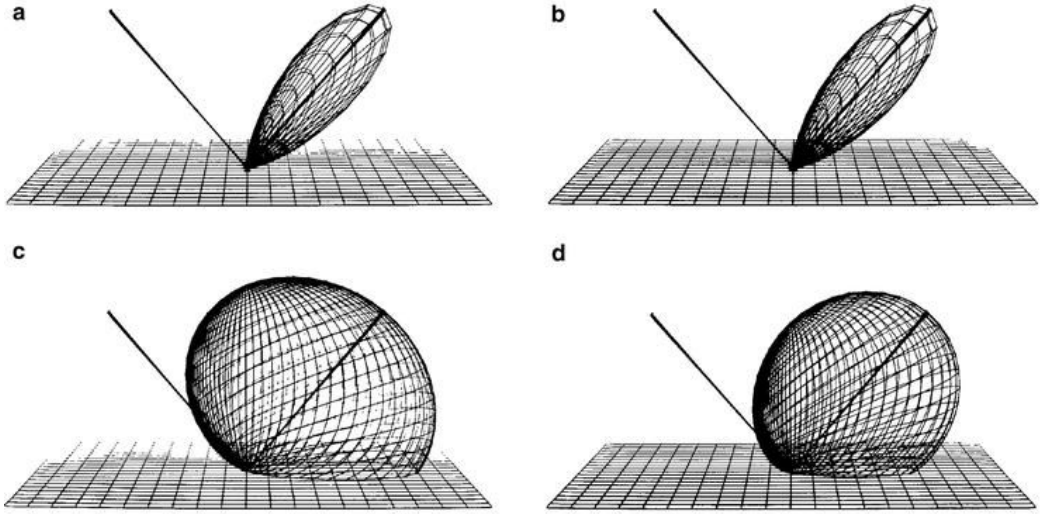
- To properly globally illuminate, we need accurate models of light reflectance
  - “Bidirectional Reflectance Distribution Function”
- We already have a few of these!
  - Lambertian
  - Blinn-Phong
  - Many more exist

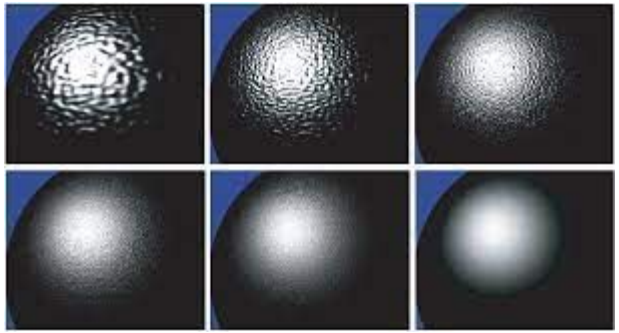


Basic Ingredients of a BRDF

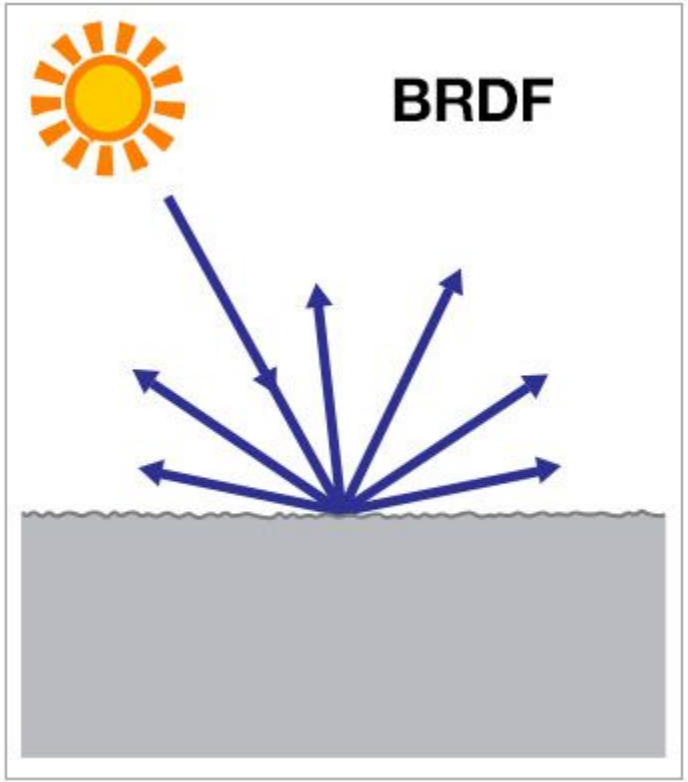


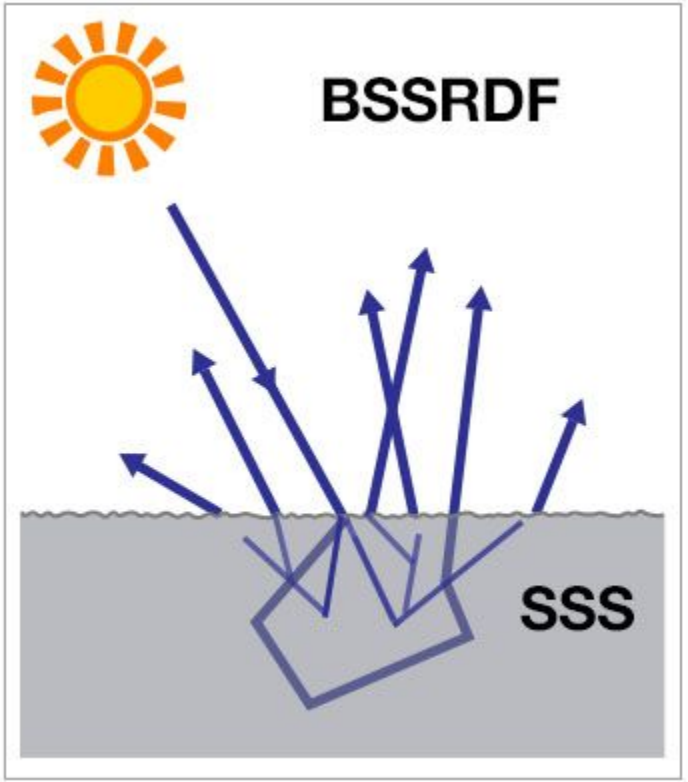
Microfaceted Surfaces



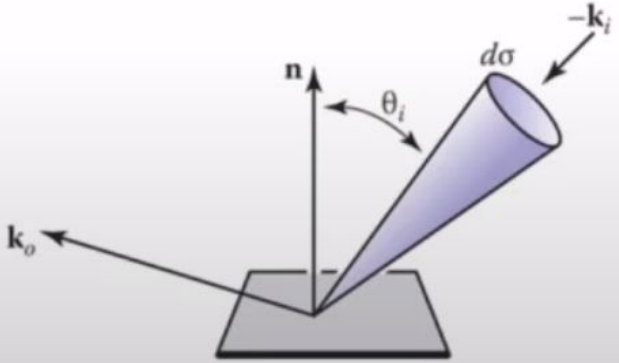


Subsurface Scattering



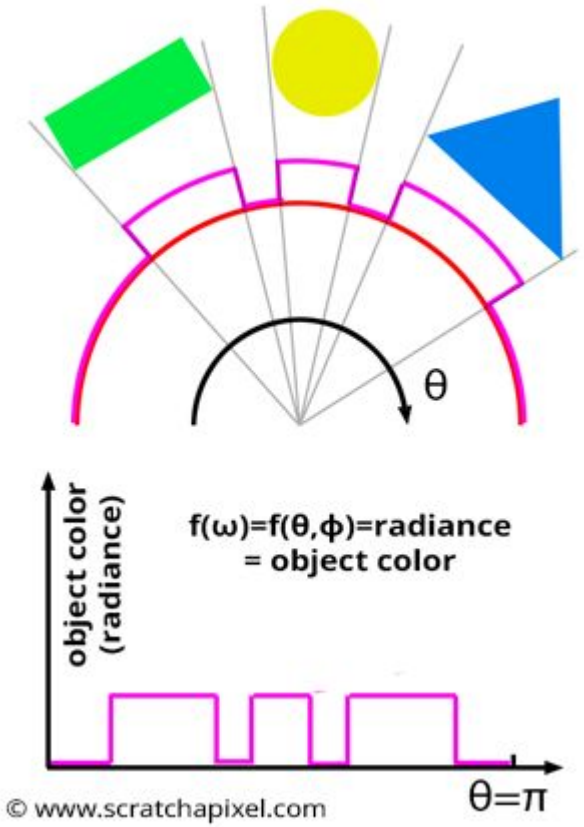


$$\underbrace{L_s(\mathbf{k}_o)}_{\text{Light in Outgoing Direction}} = \underbrace{\int_{\mathbf{k_i}}}_{\text{Aggregating Light Sources}} \underbrace{\rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k_i}) \cos\theta_i \, d\sigma_i}_{\text{Reflectance Model for Incoming Light}}$$



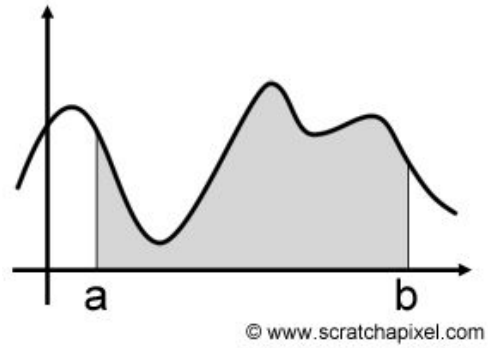
Aggregating Light Sources

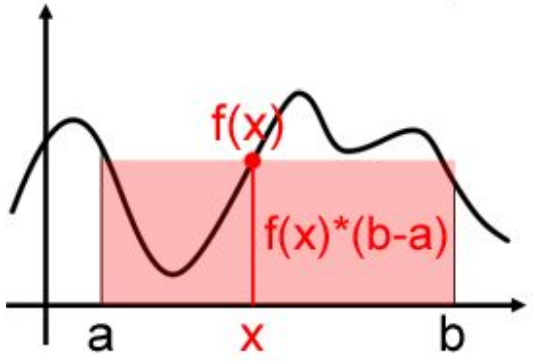
- Ideally, we want to integrate over the light functions contributing to a point.
  - Integrals require an integrand, a function that we can integrate!
    - ...we don't have this
  - We need something we can do in practice numerically



Approximating an Integral

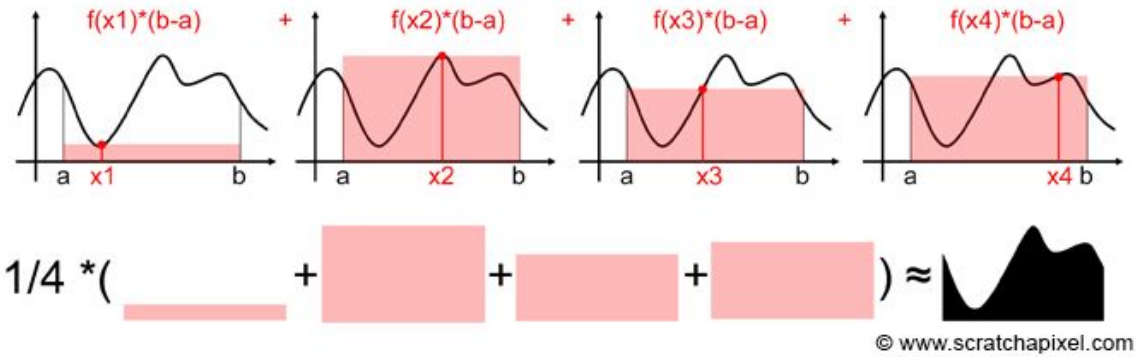
- Recall: an integral is the area under the curve of a function along interval [a,b]
- Rough Area Under a Curve: evaluate the function at a point that looks good, multiply it by the difference in the interval
    - You get a very crude approximation of the integral!





Approximating an Integral

- Like most things in calculus: do it a lot, and you get closer to the truth!
- Sample random variable, X ∈ [a,b], from a uniform distribution
- Keep evaluating f(X), average the results, you get a better approximation of the integral
  - This is a “Basic Monte-Carlo Estimator”



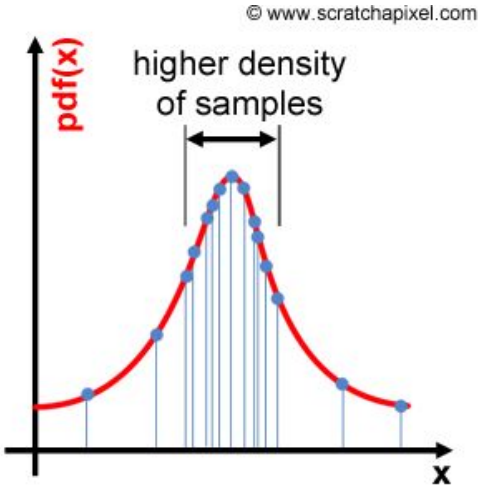
$$\langle F^N \rangle = (b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(X_i).$$

Approximating an Integral: Non-Uniform Distributions

- In practice, we often sample our random variable from a non-uniform distribution.
- Simple fix: divide f(X) by the PDF X is drawn from
  - This cancels out the (b-a) term
  - Allows drawing samples from arbitrary PDF's

$$Pr(\lim_{N\to\infty}\langle F^N\rangle = F) = 1.$$

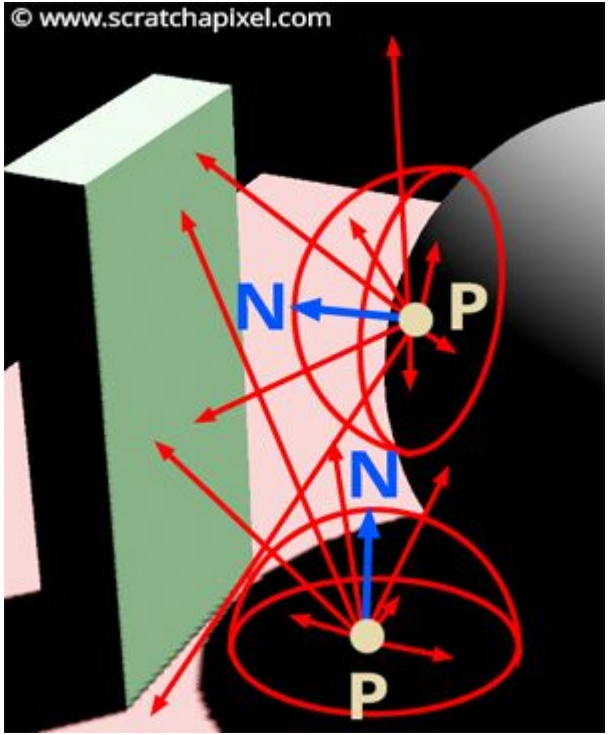
$$\langle F^N\rangle = \frac{1}{N}\sum_{i=0}^{N-1}\frac{f(X_i)}{pdf(X_i)}.$$

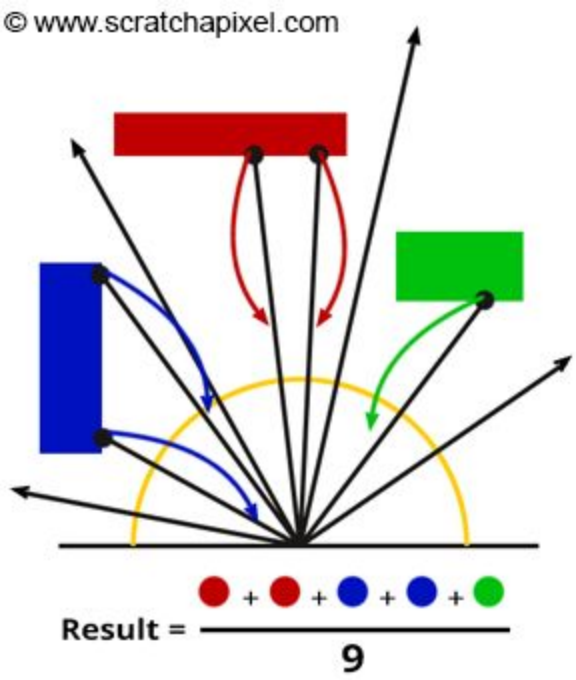


Approximating an Integral: Light Sources

- To approximate the integral of all incoming light, just take random samples!
- This is practical to implement, and respects light's "bouncy" nature

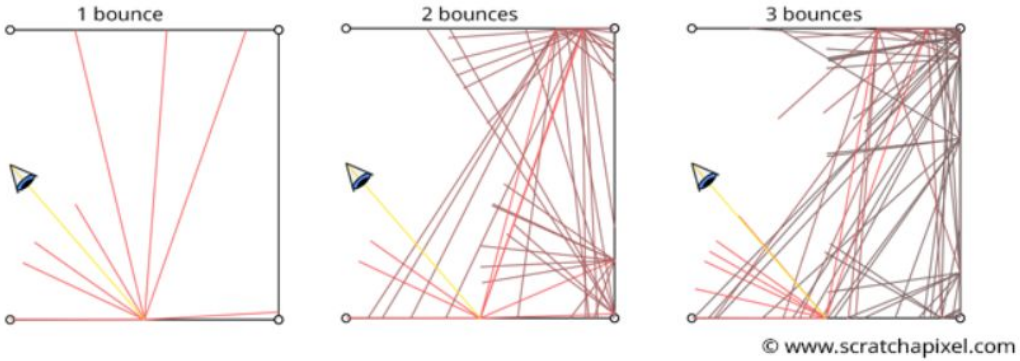
$$\text{Gather Light} \approx \frac{1}{N} \sum_{n=0}^{N} \text{castRay}(\text{P}, \text{randomDirectonAboveP}) \, .$$




© www.scratchapixel.com
Result =
9

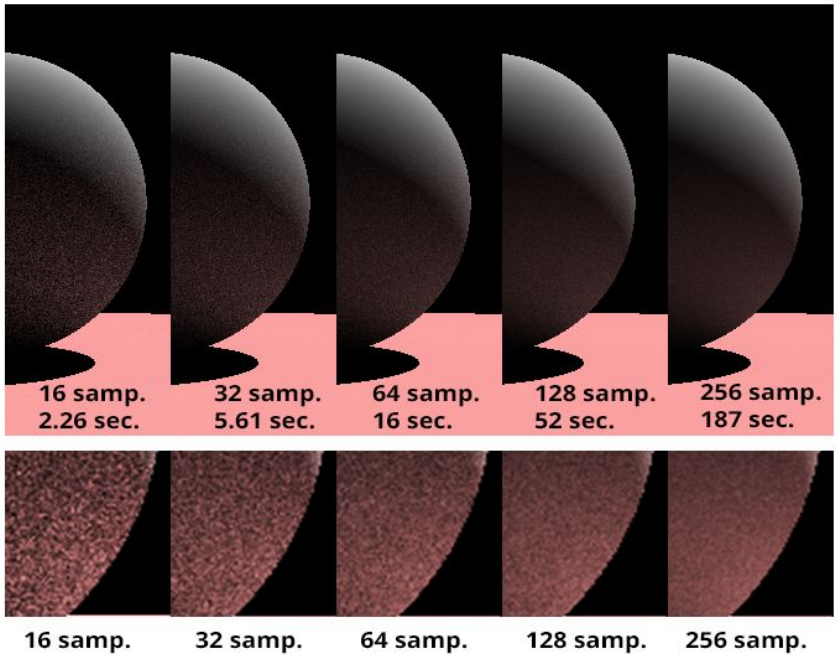
Glaring Problems

- Recursively tracing sample rays gets out of hand



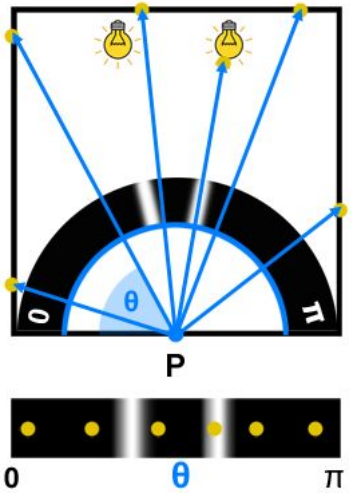
Glaring Problems

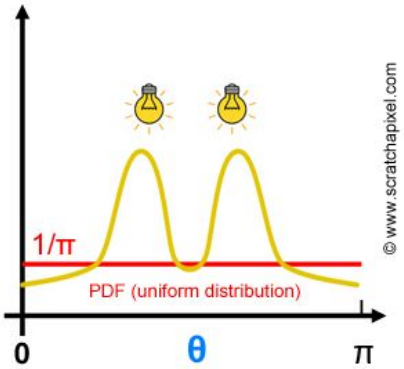
- Monte carlo path tracing creates noisy images
  - High variance in samples



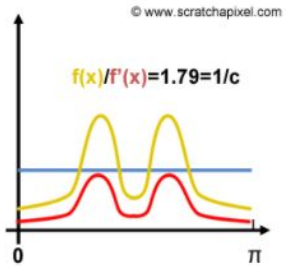
Importance Sampling

- We want a way to reduce the variance of our samples without just throwing more samples at it
  - Idea: direct our samples towards areas that contribute more light to a point





$$\langle F^N \rangle = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(x)}{pdf(x)}.$$



Transparency & Translucency

Topics:

- Indices of refraction
- Transparency calculations
- Fresnel equations
  - Reflection vs. Reflection
- Attenuation
- Pseudocode

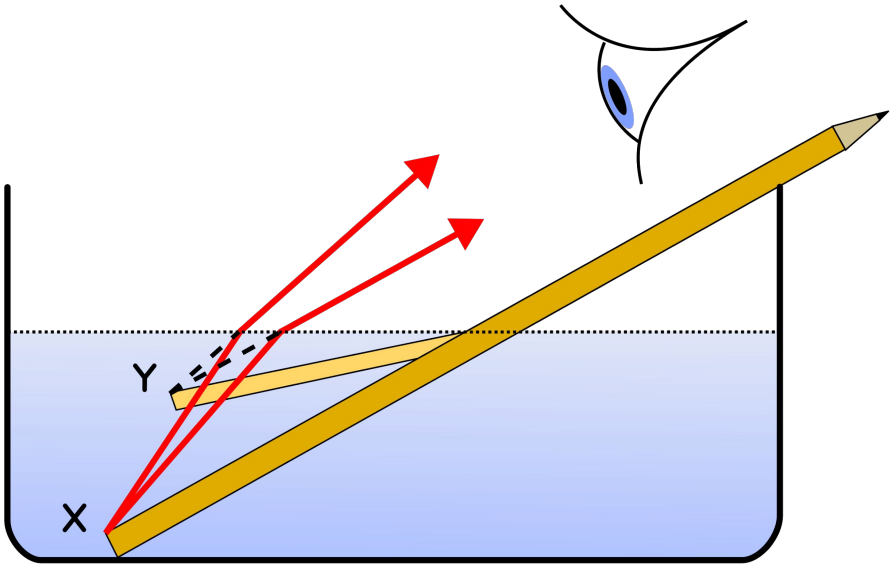
https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

What is transparency and translucency?

- Transparent
  - Light completely passing through an object
- Translucent
  - Light being filtered by an object
- Opaque
  - Light being completely blocked by an object

https://grammar.yourdictionary.com/vs/transparent-vs-translucent-vs-opaque-compared.html

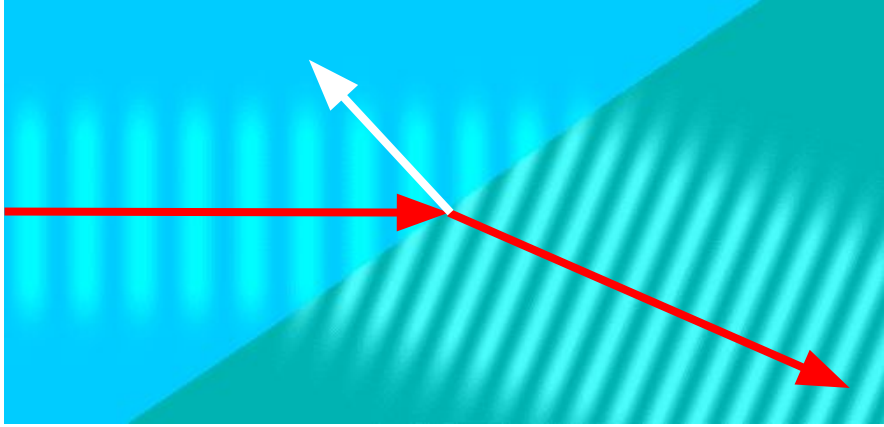
Light Direction Change



https://en.wikipedia.org/wiki/Refraction

https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Light Direction Change



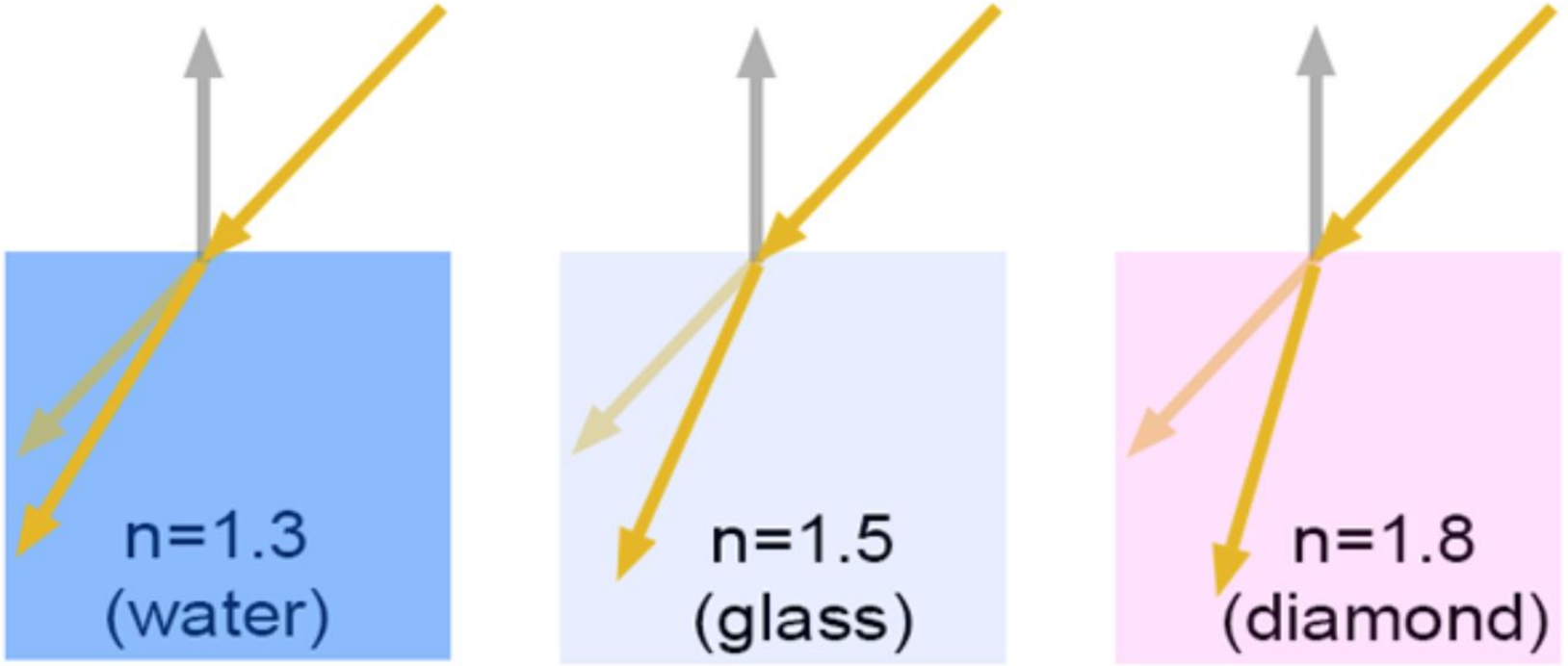
https://en.wikipedia.org/wiki/Refraction#/media/File:Refraction_animation.gif

- Refractance

- When light hits a transparent object the light changes direction based on the materials **refractive index** and the **incident angle**

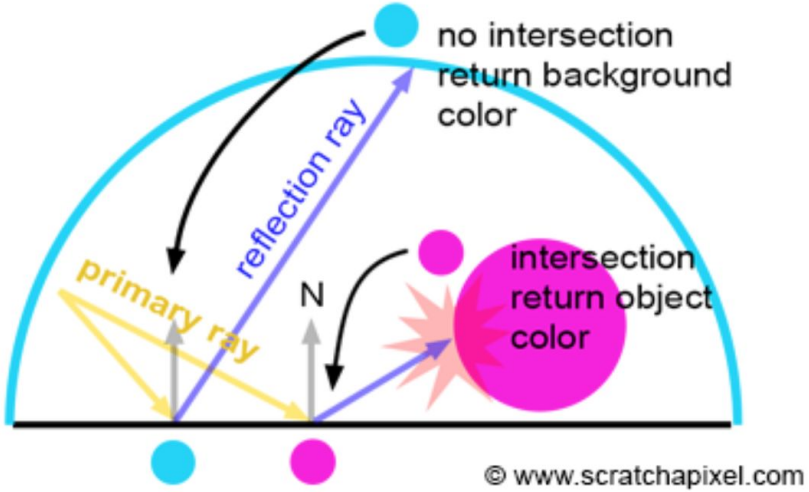
- The **refractive index** is the ratio of how much slower light travels through that material
  - E.g. a refractive index of 1.5 ➜ light travels 1.5 times slower

Refractive Indices and Light Direction Change

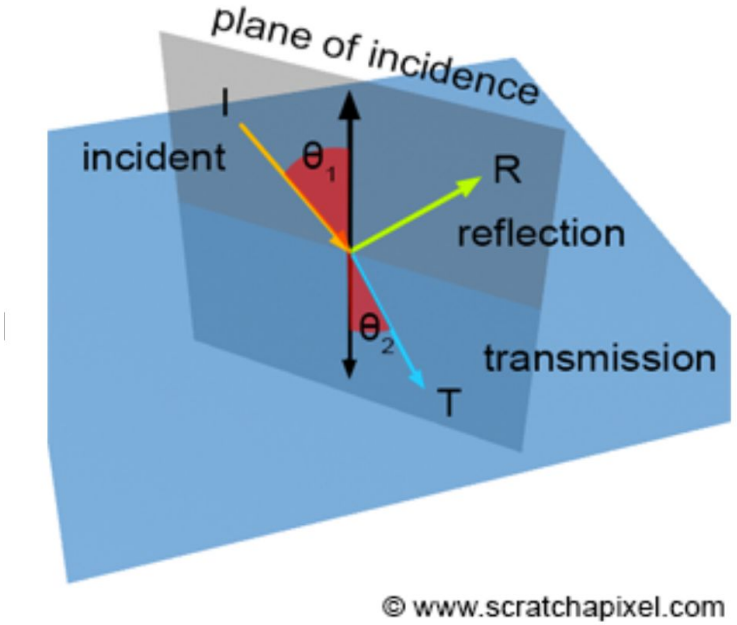


https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Refractive Indices and Light Direction Change

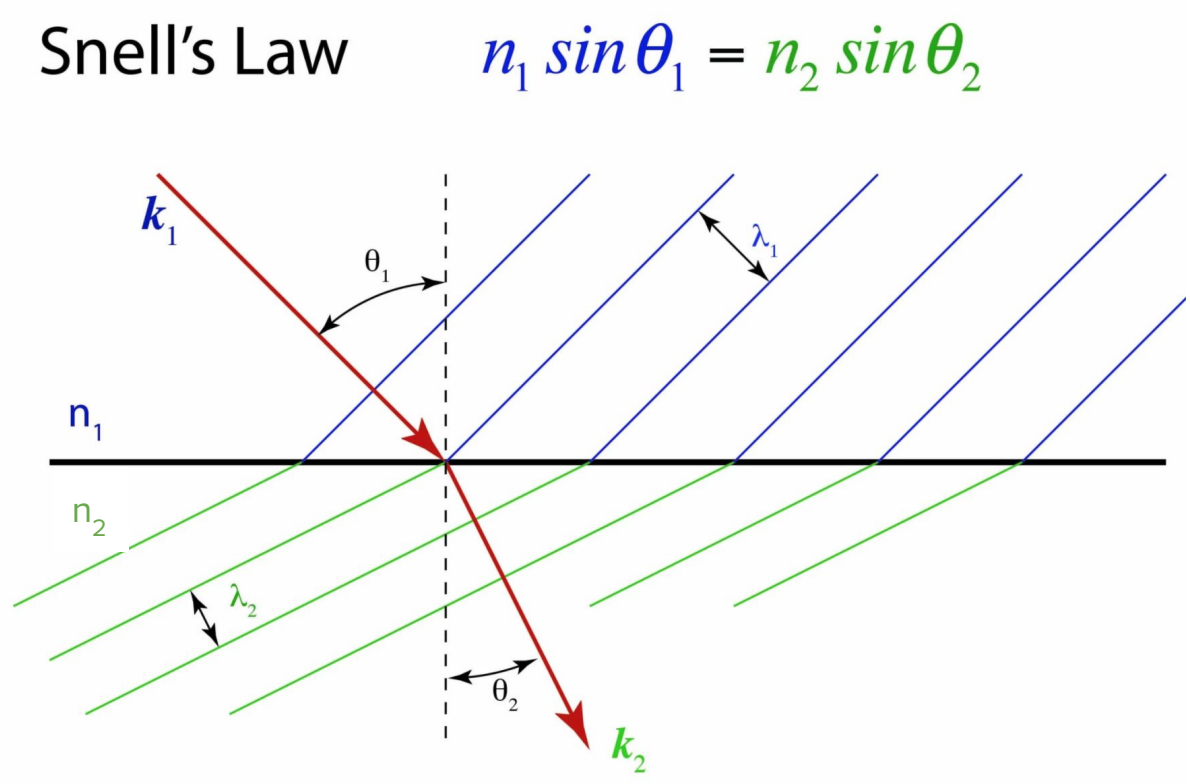


https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel



https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Calculating the Refracted Ray



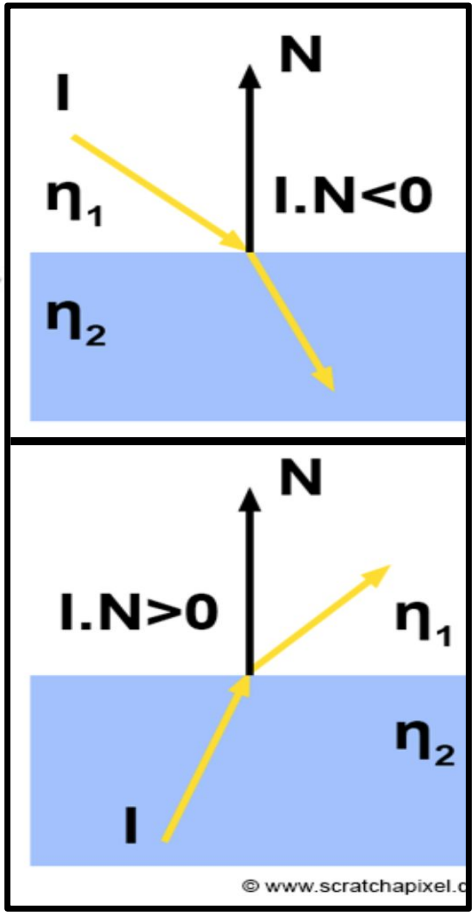
https://galileo-unbound.blog/2020/01/27/snells-law-the-five-fold-way/

Calculating the Refracted Ray

$$\eta = \frac{\eta_1}{\eta_2},$$
$$c_1 = \cos(\theta_1) = N \cdot I,$$
$$c_2 = \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 (1 - \cos^2(\theta_1))}$$

$$T = \eta(I + c_1 N) - N c_2,$$

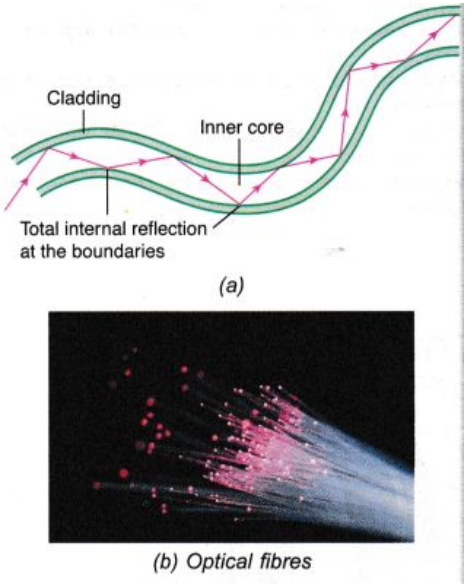
$$T = \eta I + (\eta c_1 - c_2) N.$$



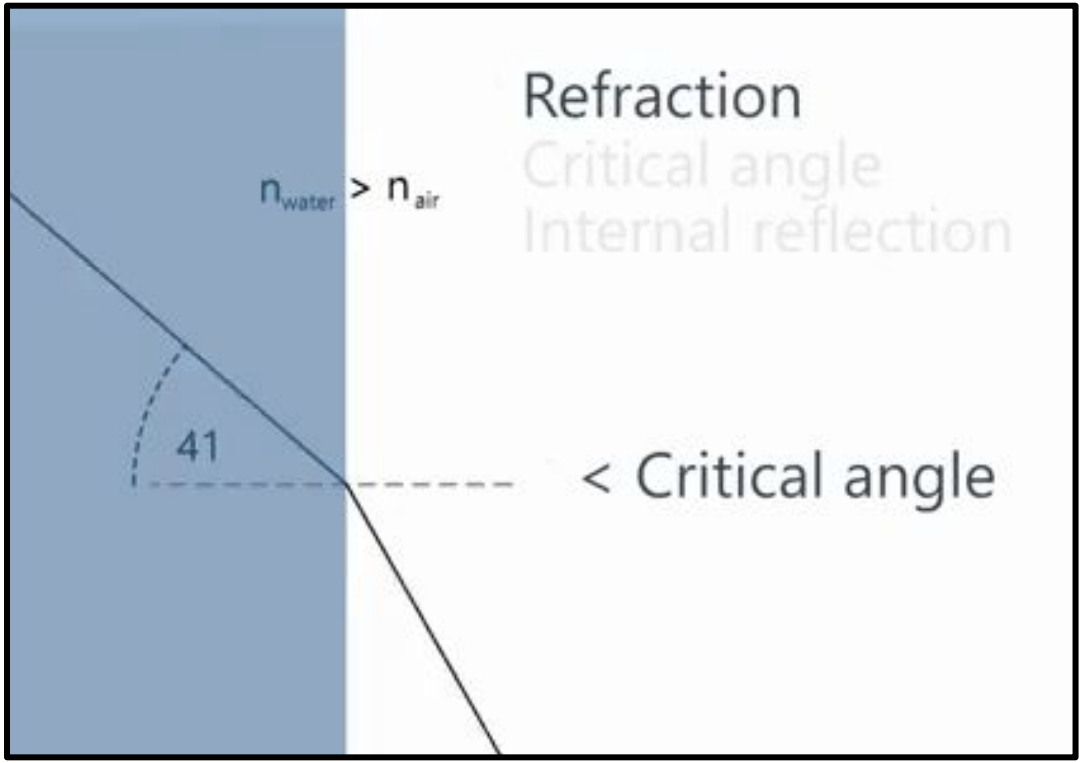
https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

What if C2 is negative?

Total internal reflection!



https://www.aplustopper.com/applications-total-internal-reflection/



https://imgur.com/gallery/0XuPa

Transmission vs. Reflectance

https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Fresnel Equations

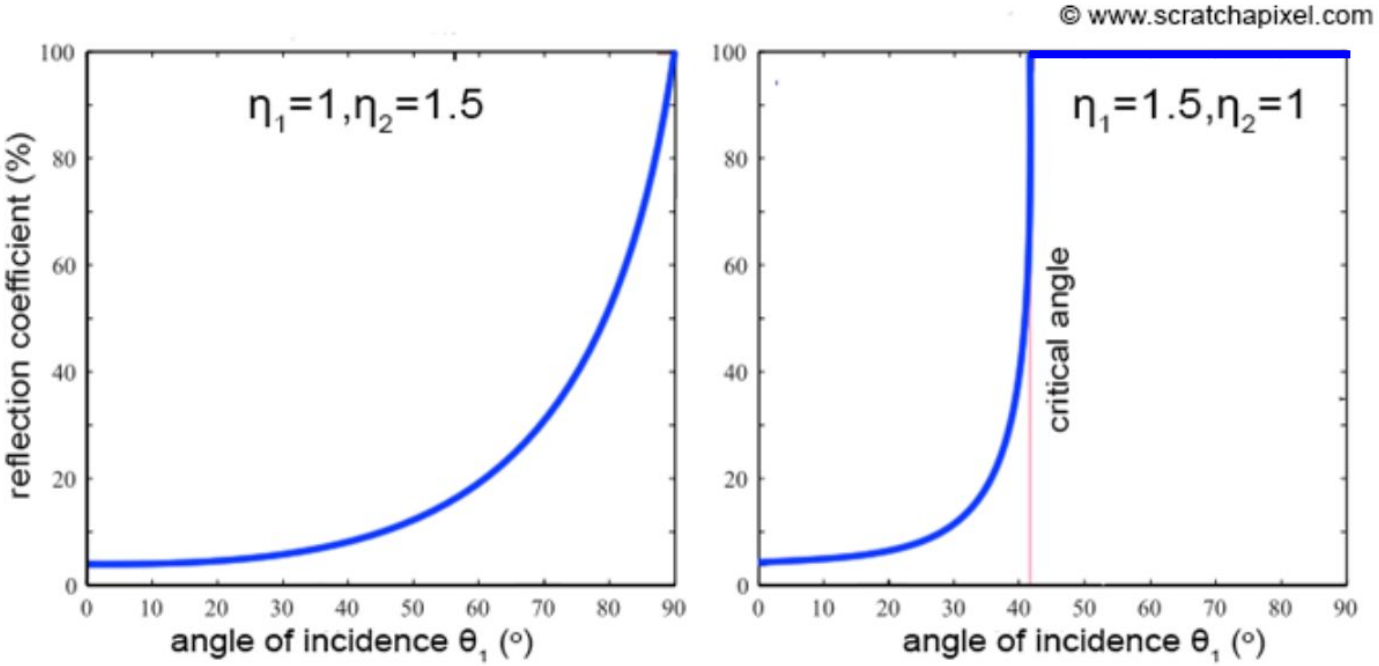
$$F_{R\parallel} = \left(\frac{\eta_2 \cos\theta_1 - \eta_1 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2} \right)^2,$$

$$F_{R\perp} = \left(\frac{\eta_1 \cos\theta_2 - \eta_2 \cos\theta_1}{\eta_1 \cos\theta_2 + \eta_2 \cos\theta_1} \right)^2.$$

$$F_R = \frac{1}{2}(F_{R\parallel} + F_{R\perp}).$$

https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Fresnel Equations

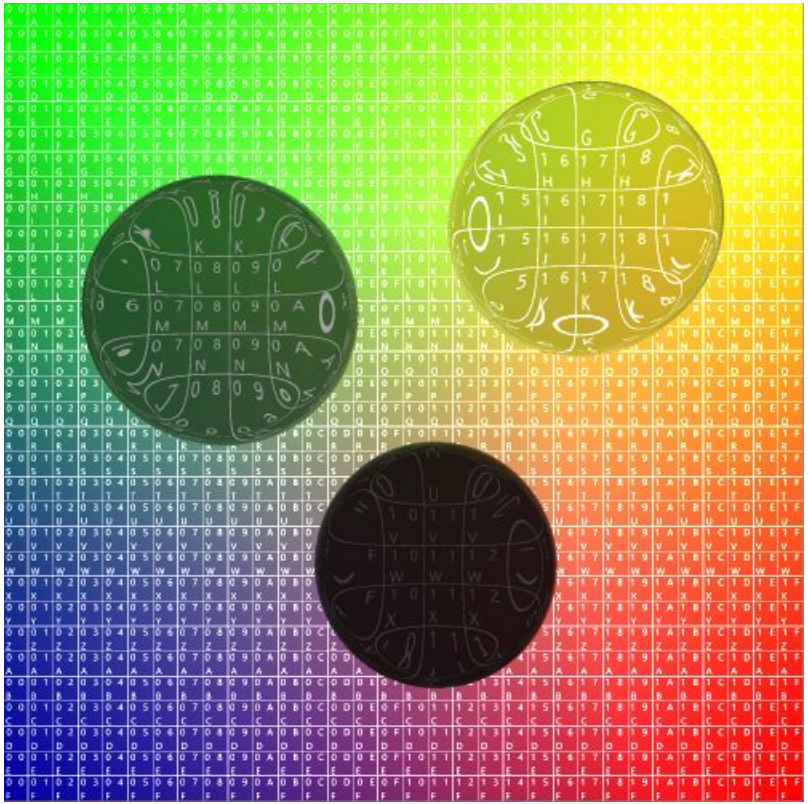


https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Covered so far:

- Refractive Indices
- Refracted Ray Directions
- Refracted Light vs. Reflected Light

Translucent object attenuation



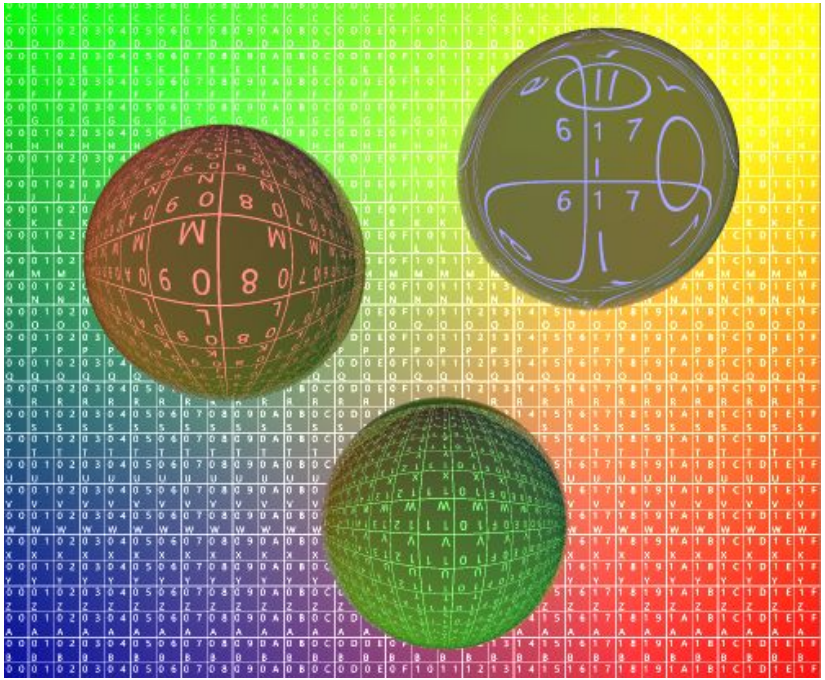
How much light is lost?

- Attenuation
  - Light gets absorbed the further it travels through a medium
  - **T** is the **distance** of ray in medium
  - **A** is **attenuation coefficient** for a color channel
  - Desmos graph

$$I(T) = I(0)*e^{-\ln(a)*T}$$

Attenuation for colored materials

- Only attenuate certain colors (RGB)



Transparency Implementation

```
def TraceRay(Scene, Ray, Attenuation, Depth):
    Closest = closest_intersect(Ray, Scene)
    Color = Scene.background
    if (Depth > 0):
        if Closest.is_transparent()
            Reflect = get_reflect_ray()
            Reflect_Color = TraceRay(Scene, Refract, Attenuation, Depth-1)
            Refract = get_refract_ray()
            Fresnel = get_fresnel()
            if (!Total_Internal_Reflection(Refract, Fresnel))
                Entering_Object = (dot(normal, ray.dir) < 0)
                if Entering_Object:
                    Refract_Color = TraceRay(Scene, Refract, Closest.Attenuation, Depth-1)
                else:
                    Refract_Color = TraceRay(Scene, Refract, Scene.Attenuation, Depth-1)
                end
                Color += Fresnel*Reflect_Color + (1-Fresnel)*Refract_Color
            else
                Color = Reflect_Color
            end
        else if Closest.is_mirror():
            # We did this in A2 already
        end
        Color += Local_Color(Closest)
        Color = Apply_Attenuation(Color, Attenuation)
    end
    Return Color
end
```

Questions or Comments?