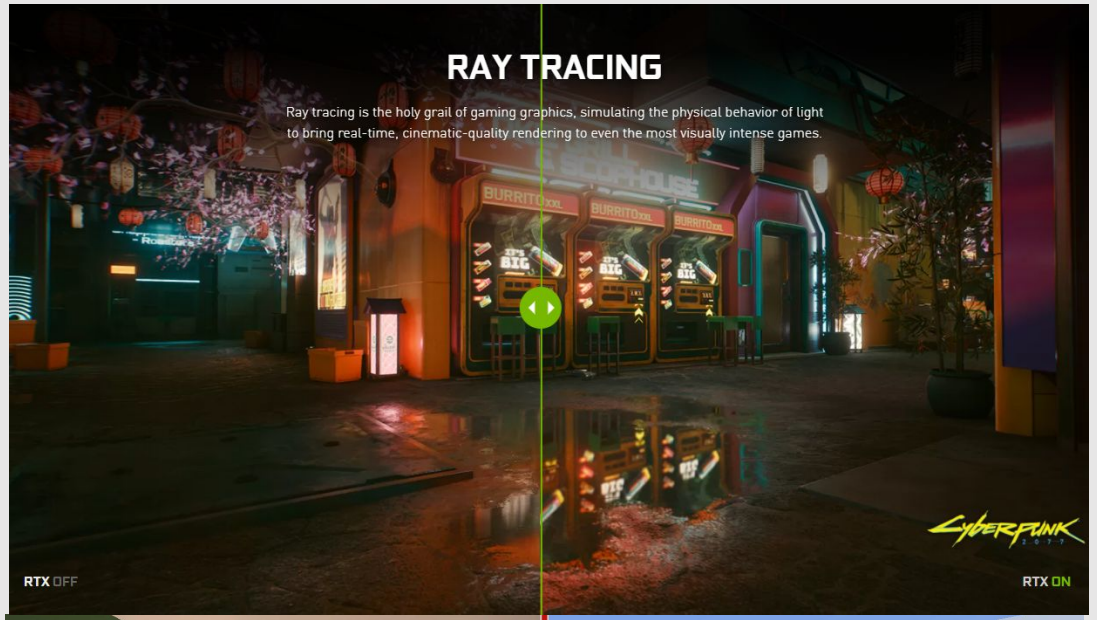
A3.5: PathTracer

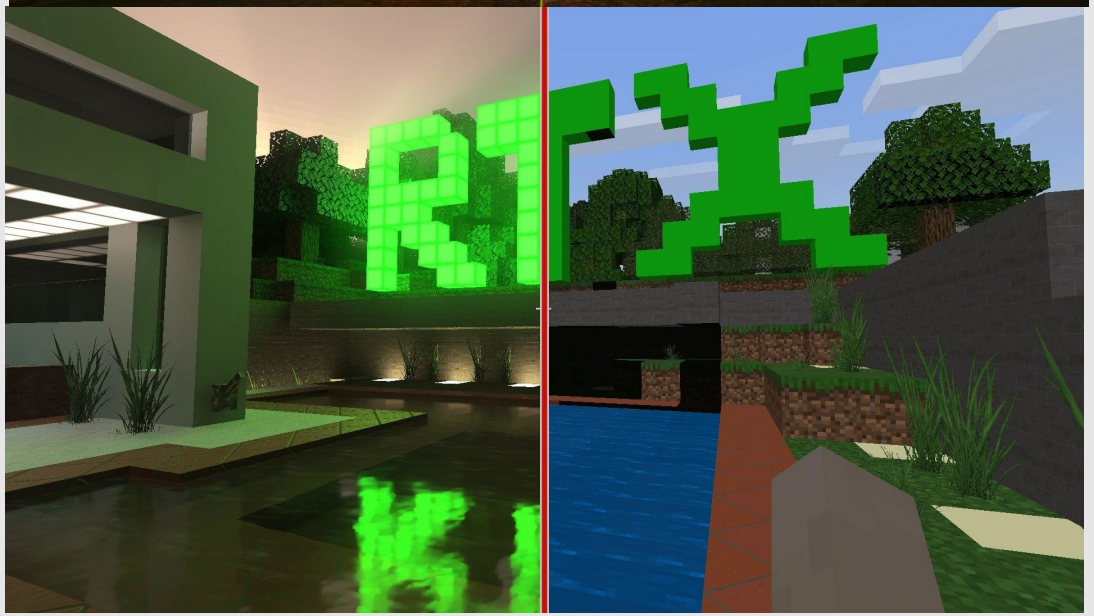
- Ray Tracing Overview
- BRDFs
- Direct vs Indirect Lighting
- Environment Lighting

- Ray Tracing Overview
- BRDFs
- Direct vs Indirect Lighting
- Environment Lighting

Ray Tracing Overview

- High Level Idea:
  - Let's simulate how light works in the real world by shooting out a ton of light rays from our camera and seeing how it interacts with our simulated world
- Leads to very photorealistic results
- But is quite expensive :(
- In the two examples on the right, can you pick out the advantages of ray tracing?





Rendering Equation

- Want to add up all the incoming light reaching our hemisphere

$$\text{(recursive definition)} = \text{(base case)} + \int_{\mathcal{H}^2} \text{(scattering function)} * L_i(\mathbf{p}, \omega_i) \cos\theta \, d\omega_i$$

- Why multiply by cos(theta)?
  - Want to give less weight to light rays hitting at more grazing angles
  - And more weight to rays hitting at direct angles

Rendering Equation

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{\mathcal{H}^2} f_r(\mathbf{p}, \omega_i \rightarrow \omega_o) L_i(\mathbf{p}, \omega_i) \cos\theta \, d\omega_i$$

Amount of outgoing light

Amount of light our current point emits

BRDF! (how to scatter a given incoming ray)

Incoming light ray

not so* SCARY EQUATION :)

Monte Carlo Sampling

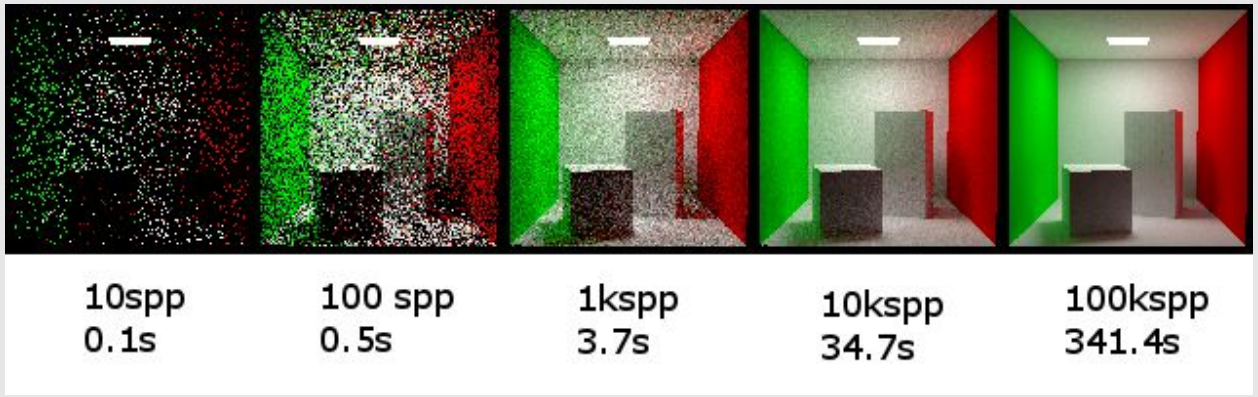
- $\int_{\mathcal{H}^2}$ = perfectly integrating across all possible points, which is impossible :(
- So, instead let's randomly sample possible incoming rays, recursively trace them backwards to figure out how much light they bring, add all these up together, and divide by the number of rays we sampled

- curr_light = vec3(0.f)
- for number of samples:
  - randomly sample incoming ray
  - curr_light += ray_trace(ray)
- curr_light /= number of samples

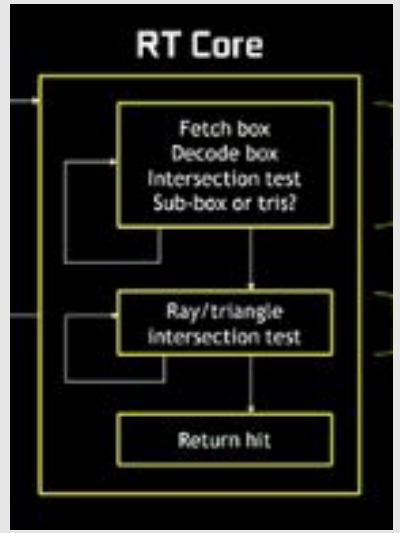
Side Note: How does RTX/DLSS work?

- Obviously what we've just talked about takes a long time to run, which is something that real time applications (video games, renderers, etc.) can't have

- Deep Learning Super Sampling
- Key Idea:
  - Let's run "ray tracing" for a short amount of time, then run an AI denoising algorithm to predict what our final image would look like



- RTX
- Key Idea:
  - Have dedicated hardware cores *just* for ray tracing
  - "NVIDIA RT Cores"

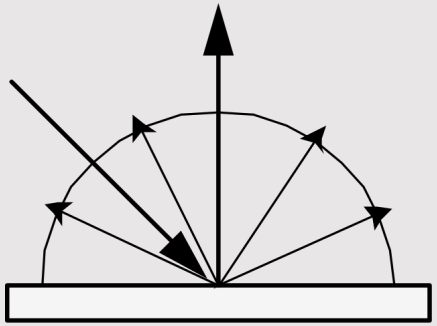


- Ray Tracing Overview
- BRDFs
- Direct vs Indirect Lighting
- Environment Lighting

Code Overview

- **`Scatter scatter(Vec3 out_dir)`**
  - given `out_dir`, generates a random sample for `in_dir`. It returns a `Scatter`, which contains both the sampled `direction` and the `attenuation` for the in/out pair.
- **`Spectrum evaluate(Vec3 out_dir, Vec3 in_dir)`**
  - evaluates the BSDF for a given pair of directions. This is only defined for continuous BSDFs
  - used to get the `attenuation`
- **`float pdf(Vec3 out, Vec3 in)`**
  - only for Lambertian, you will also need to calculate the pdf

Lambertian BSDF



- Also known as diffuse
- Light is equally likely to be reflected in each output direction
  - BRDF is a constant, relying on **albedo** (ρ)

$$f_r = \frac{\rho}{\pi}$$

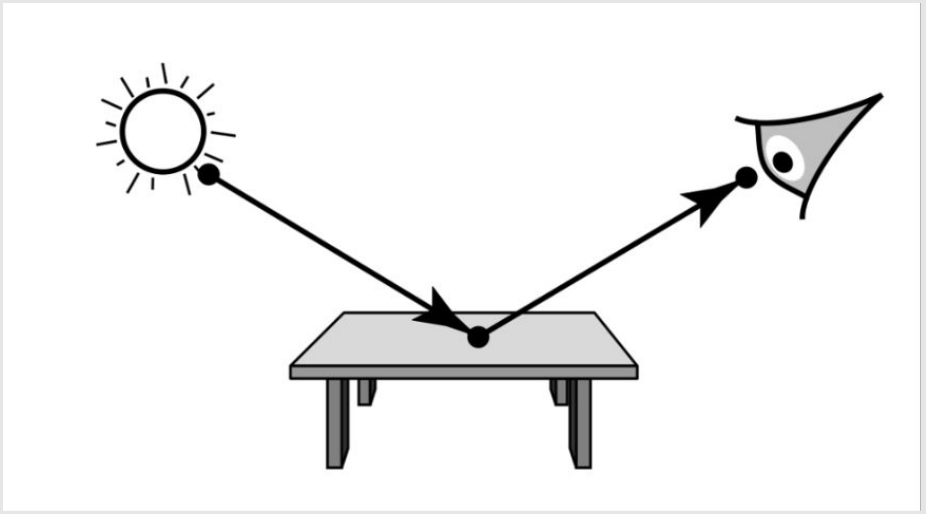
  - BRDF can be pulled out of the integral

$$\begin{aligned} L_o(\omega_o) &= \int_{H^2} f_r \, L_i(\omega_i) \, \cos\theta_i \, \mathrm{d}\omega_i \\ &= f_r \int_{H^2} L_i(\omega_i) \, \cos\theta_i \, \mathrm{d}\omega_i \\ &= f_r E \end{aligned}$$

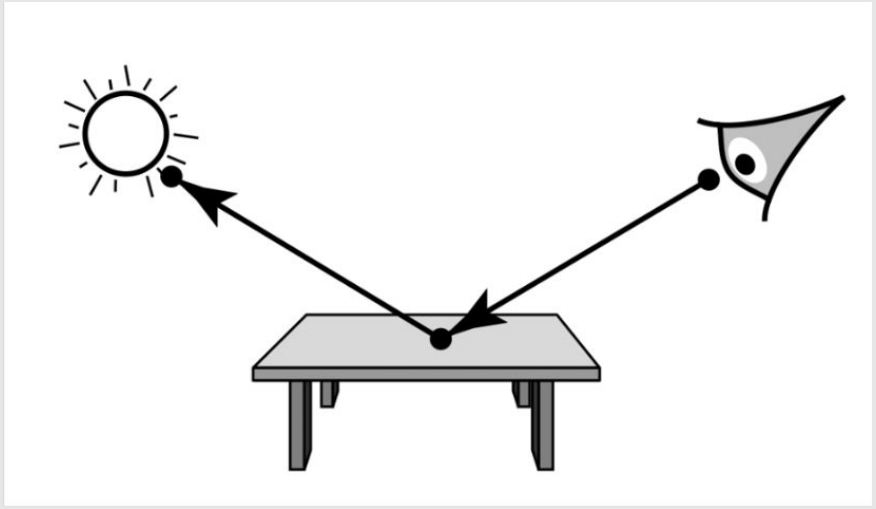
Minions (2015) Illumination Entertainment

- Easy! Pick any outgoing ray w_o

Remember…



forwards path tracing



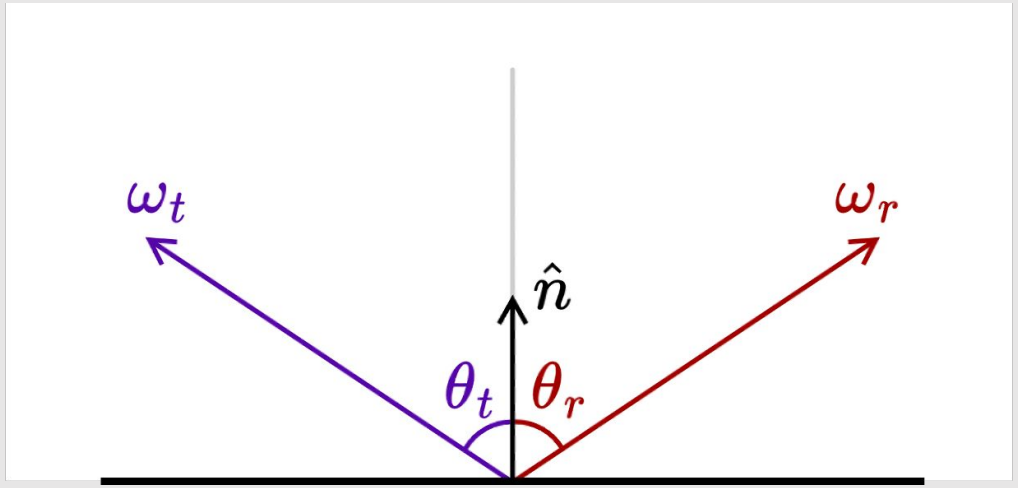
backwards path tracing

We are tracing rays backwards, from the camera into the scene!

Mirror BSDF

Given ω_t we want to calculate ω_r

$$\omega_r = -\omega_t + 2(\omega_t \cdot \vec{n})\vec{n}$$



Remember, that we are doing everything in surface-local space

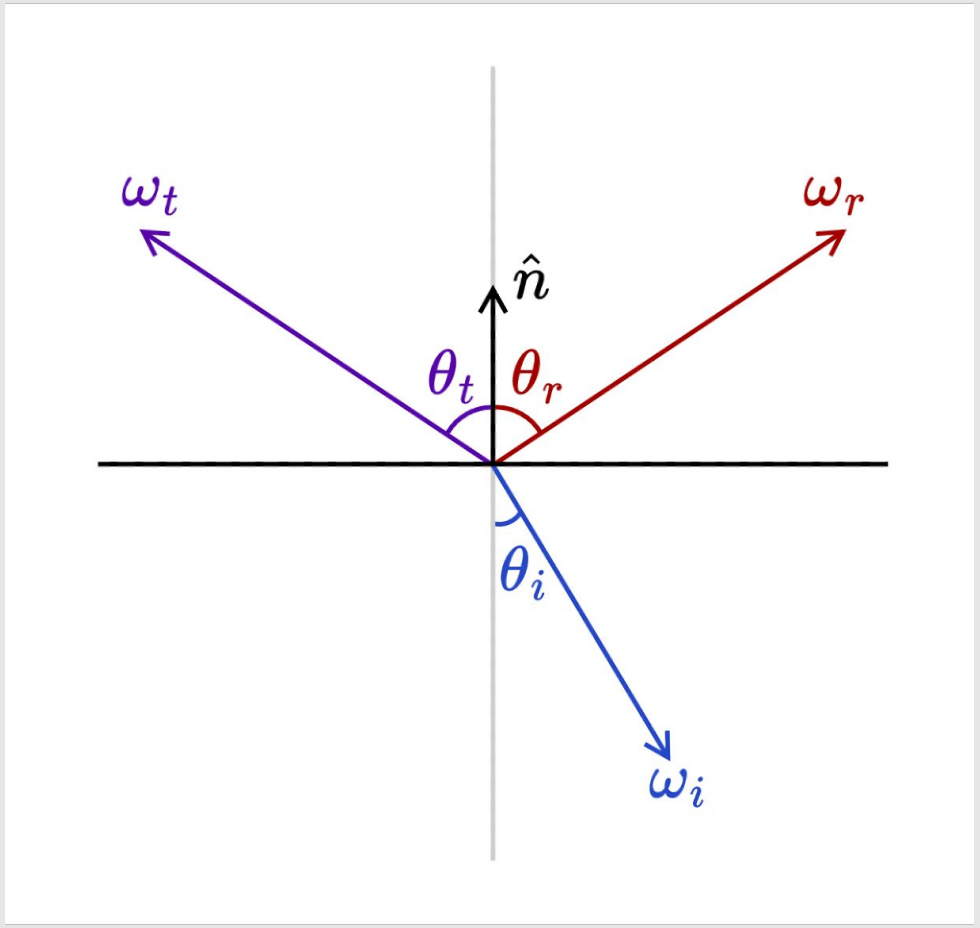
⇒ **Surface normals are always (0, 1, 0)**

Glass BSDF

1. Try to refract the ray
2. If total internal reflection is happening, **reflect** the ray
3. Else, calculate the **Fresnel coefficient, Fr**
4. Reflect or refract probabilistically based on **Fr**
   - Reflect with probability **Fr**
   - Refract with probability (**1-Fr**)

TIR:

$$1 - \left(\frac{\eta_i}{\eta_t}\right)^2 (1 - \cos^2\theta_i) < 0$$



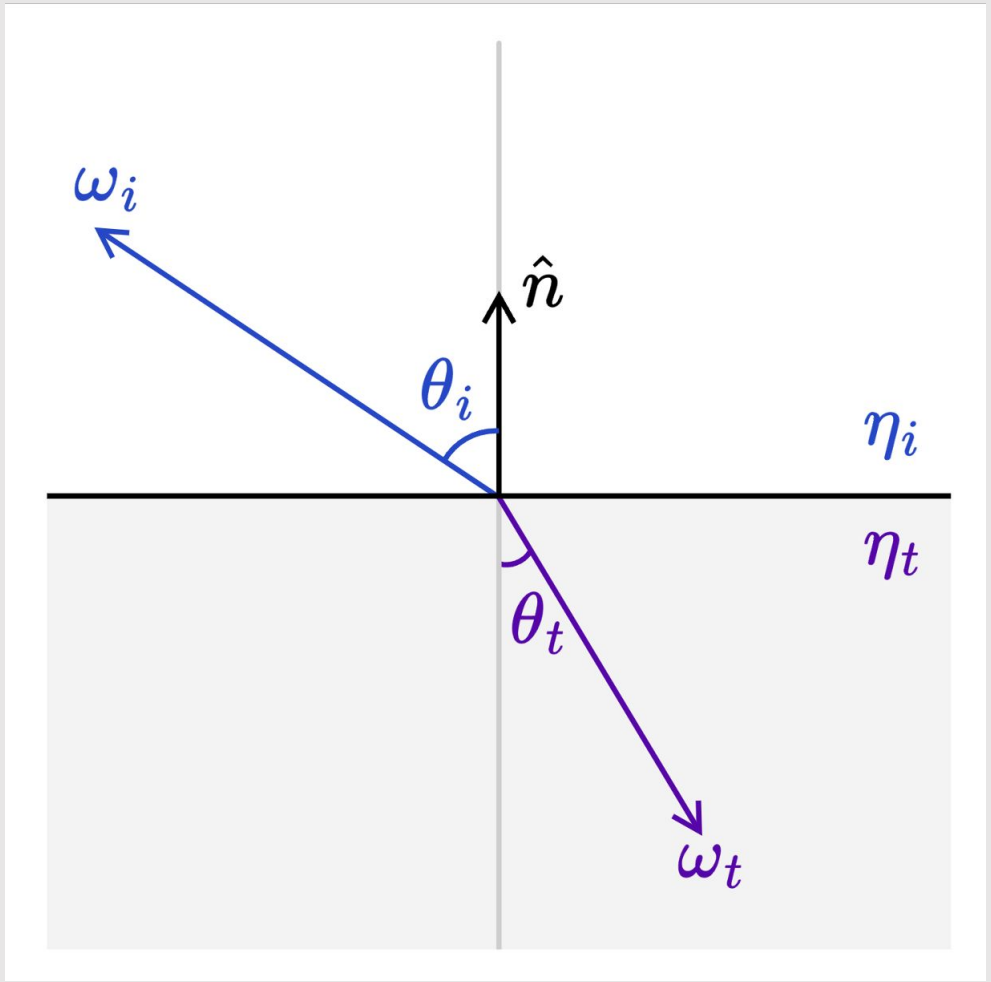
Refraction

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

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

$$= \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 \sin^2\theta_i}$$

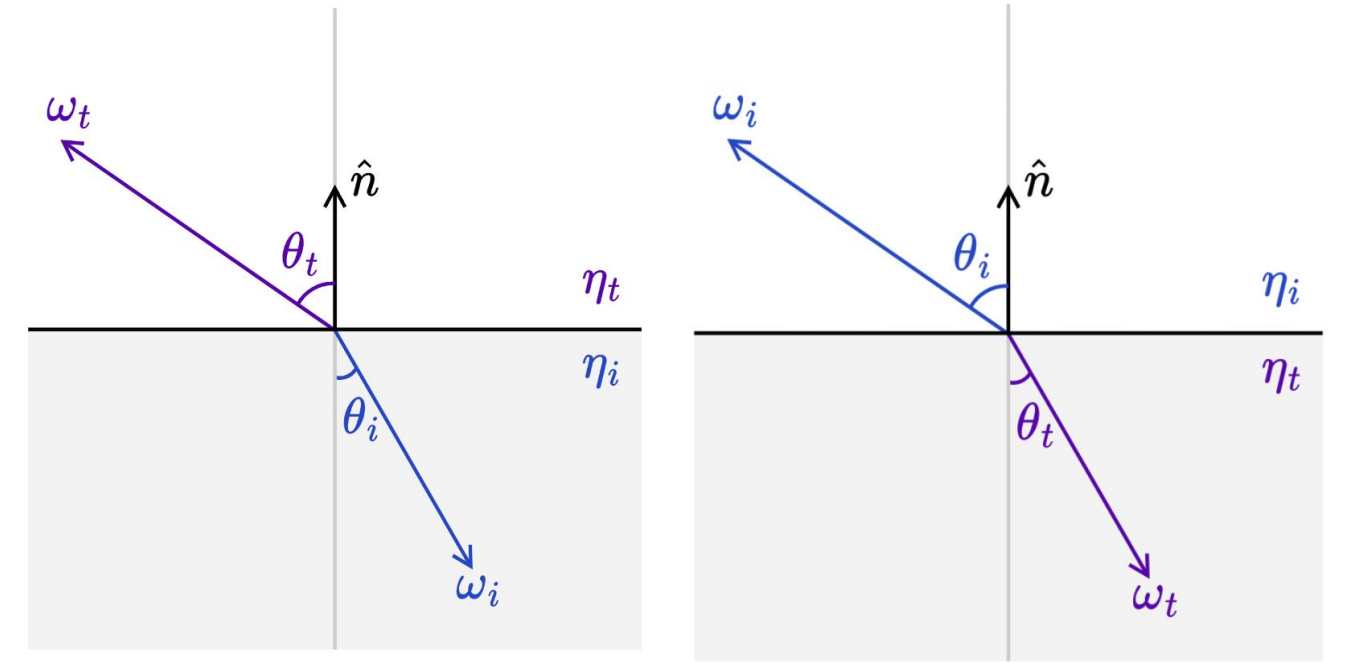
$$= \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 (1 - \cos^2\ \theta_i)}$$



Look at the textbook for further derivation

(the useful stuff is at the bottom of section 8.2.3)

Ray Directions



Look at the textbook for more information (the useful stuff is right above figure 8.11)

Your implementation should assume that when ω_i enters the surface then the ray was previously travelling in a vacuum (i.e. ior = 1.0)

If $\hat{n} \cdot \omega_t > 0$ then we are **entering** the material, so $\eta_i = 1,\ \eta_t = ior$

If $\hat{n} \cdot \omega_t < 0$ then we are **exiting** the material, so $\eta_i = ior,\ \eta_t = 1$

Fresnel

Schlick's Approximation

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

where

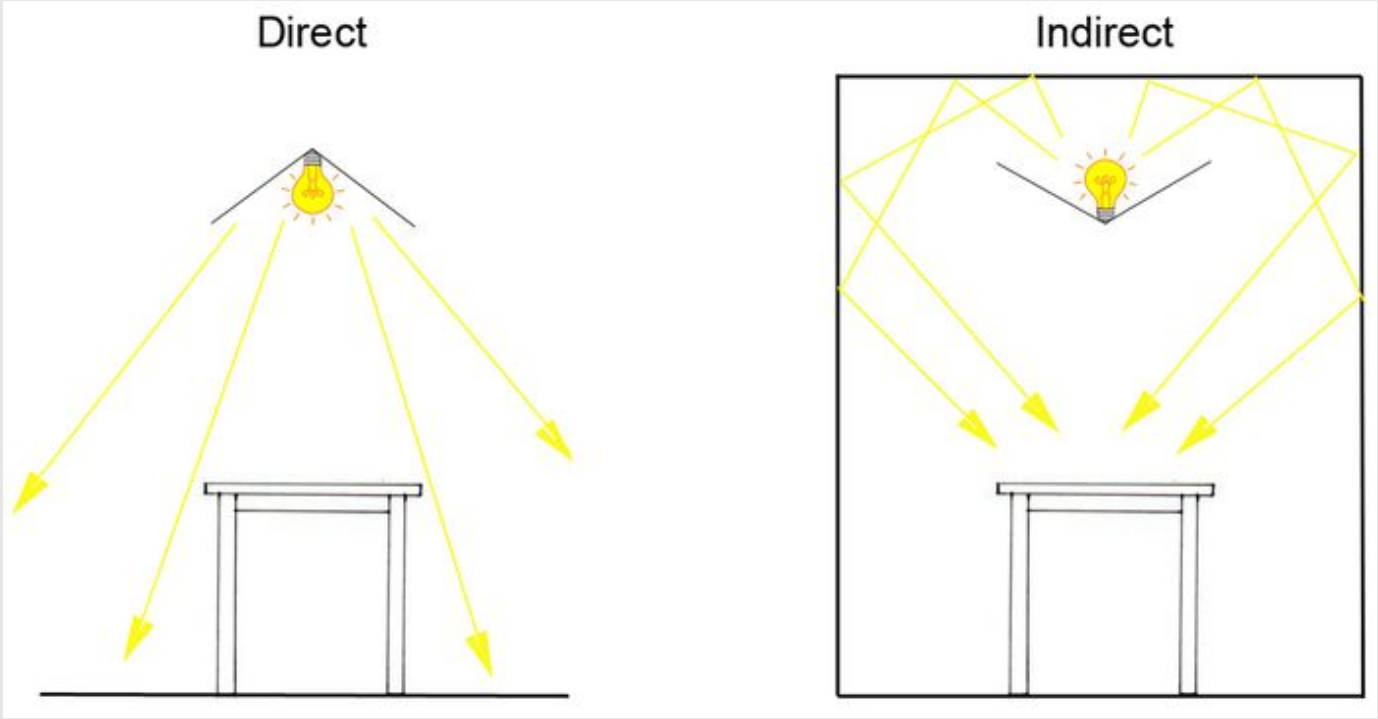
$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

- Ray Tracing Overview
- BRDFs
- Direct vs Indirect Lighting
- Environment Lighting

Indirect vs Direct Lighting

Light shining onto an object is called *direct lighting*.

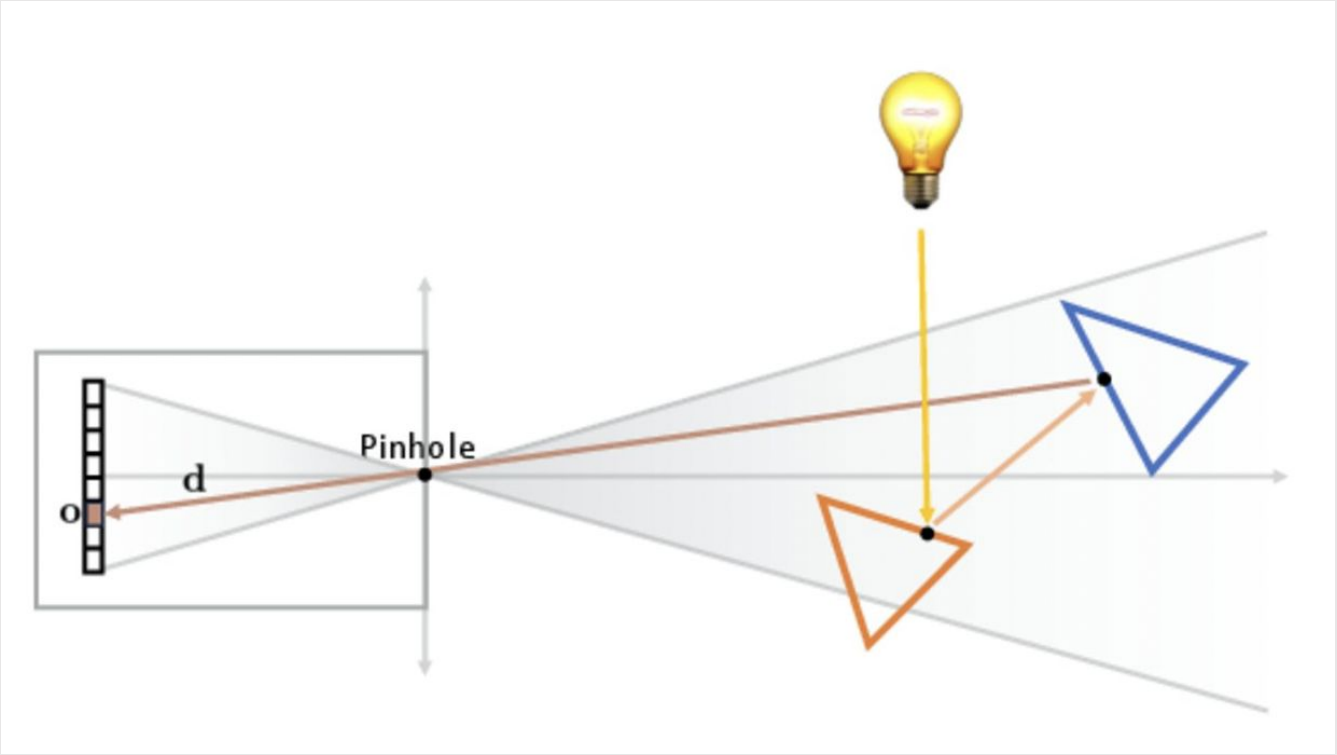
Light bouncing off a surface, illuminating other objects is called *indirect lighting*.



Indirect Lighting

Given a ray intersection point, want to estimate light that bounced off at least one other surface before reaching this point.

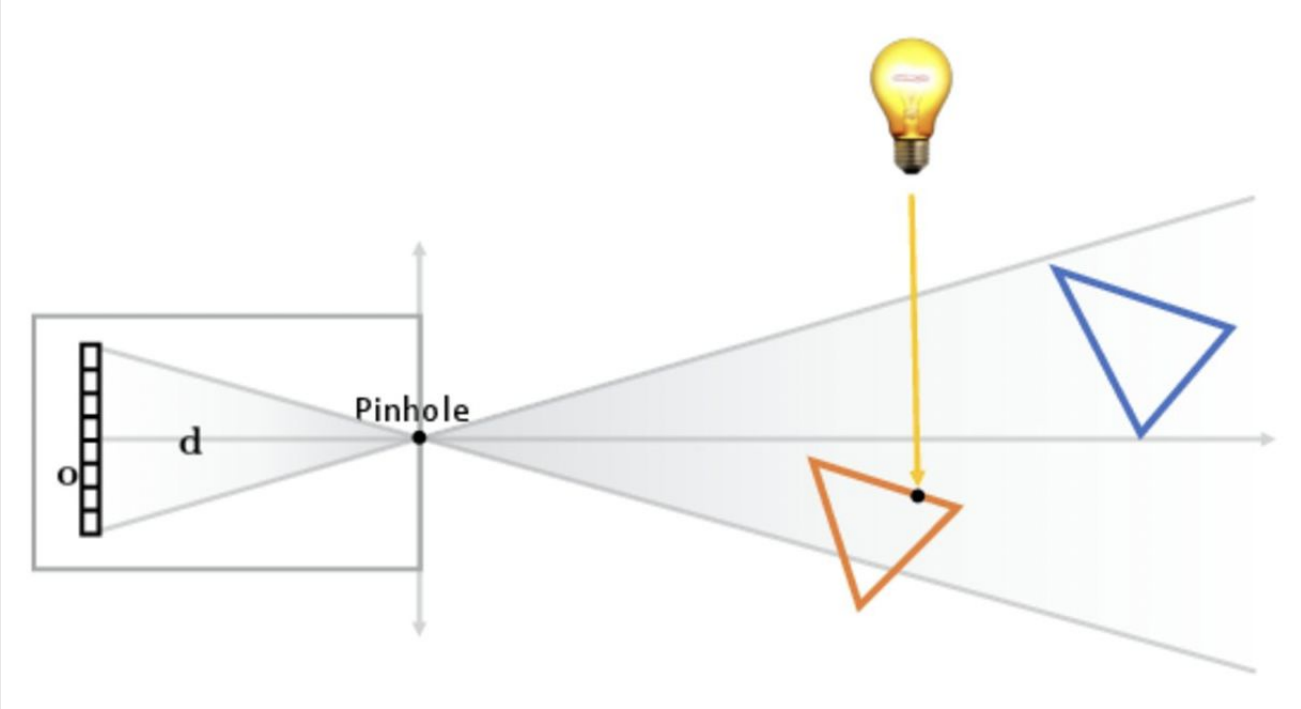
- Sample a ray from BSDF distribution for a sample direction
- Trace in this direction into the scene and gather incoming light
- Have a terminating depth to avoid infinite recursion



Direct Lighting

Given a ray intersection point, want to estimate light that hit this point after being emitted from a light source without any bounces in between

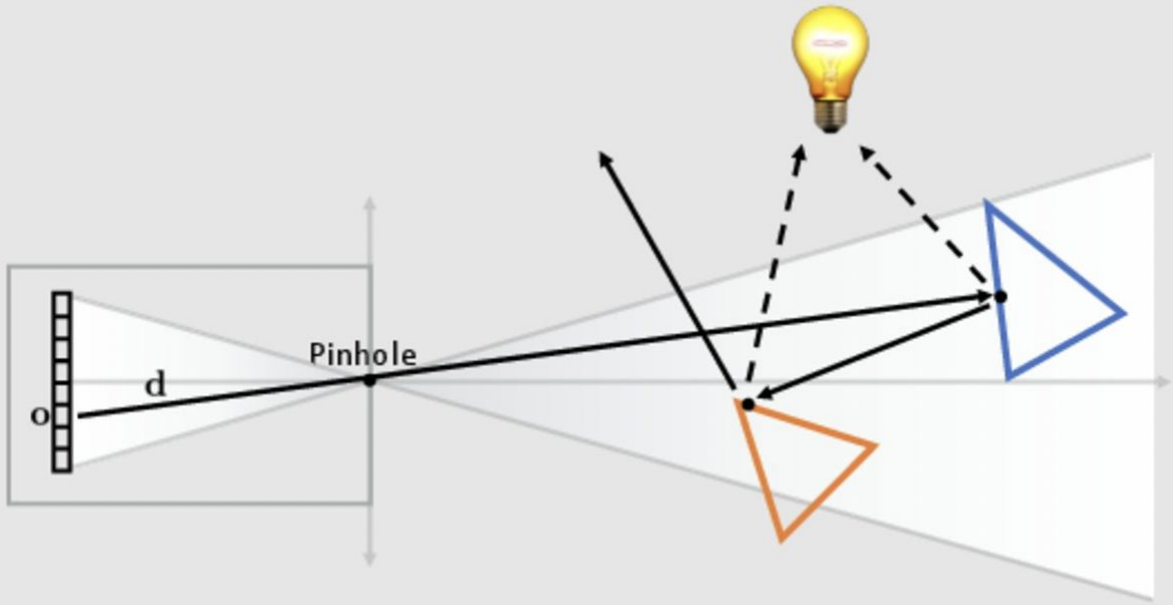
- Just like indirect lighting, but with max depth = 0



Importance Sampling with NEE

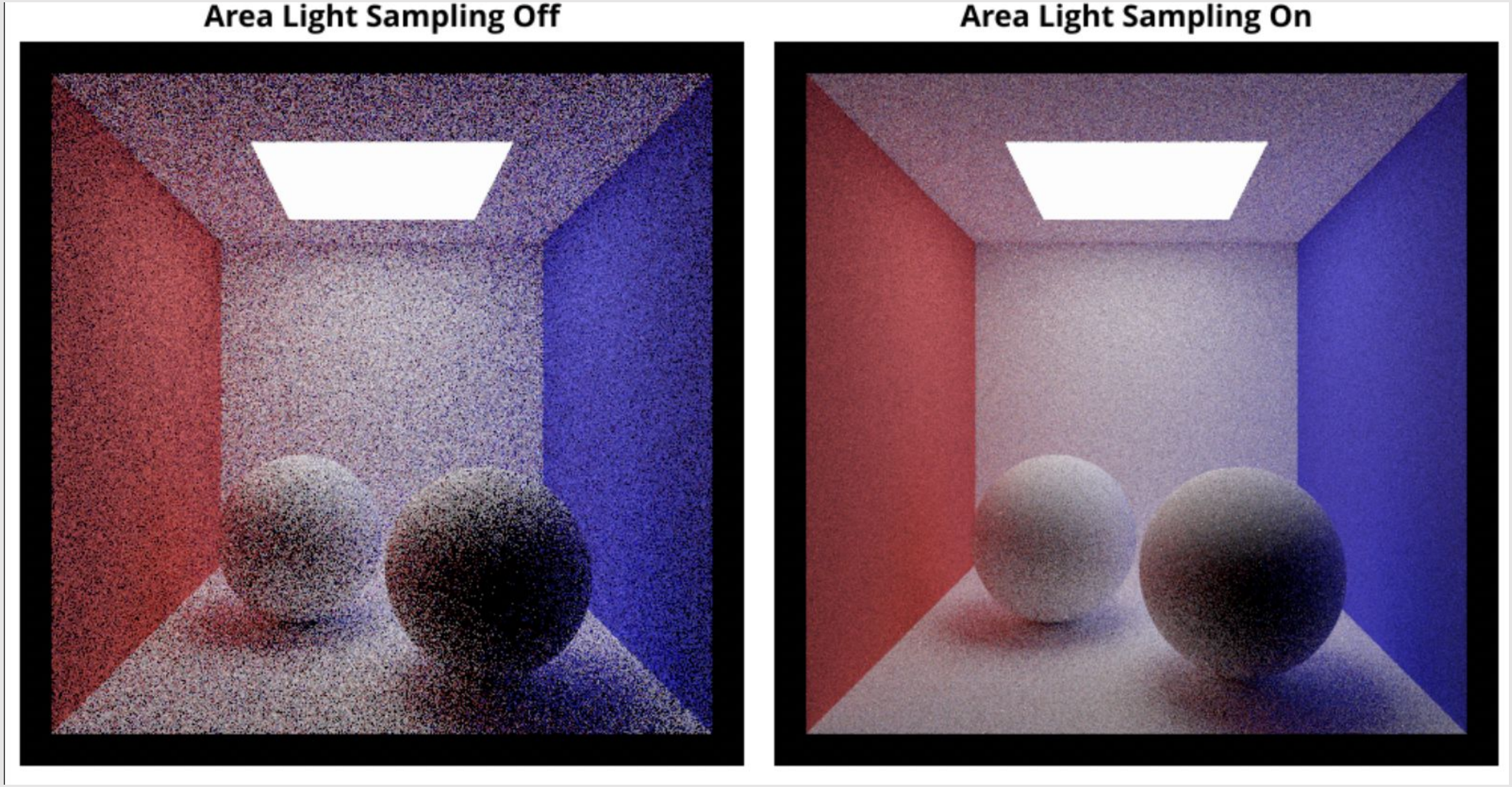
(Direct Lighting)

- Only importance sampling the BSDF term (via cos(theta)) => high variance
- Sampling from directions of area lights in the scene = better account for incoming radiance.
  - Trace two rays
    - Ray from BSDF
    - Ray towards light
  - Create a new distribution that has the average PDF of the two rays
    - Why not average?



Importance Sampling with NEE

(Direct Lighting)



- Ray Tracing Overview
- BRDFs
- Direct vs Indirect Lighting
- Environment Lighting

Environment Lighting

- In addition to point, spot, and directional lights, we also have *environment lights*
  - A texture map with light intensity data in all directions
  - Acts as an infinitely big sphere that is infinitely big
  - Parameterized by θ and φ

Class Functions

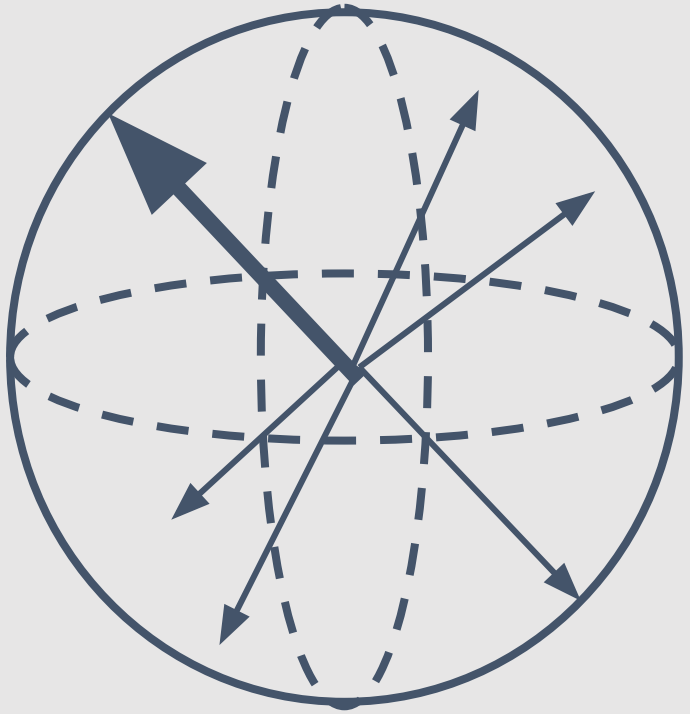
- **`Samplers::Sphere::Uniform`**
  - Creates struct that provides a method for uniformly sampling the sphere surface. Ensures that every point on the sphere's surface has an equal chance of being selected (uniform).
- **`Samplers::Sphere::Image`**
  - Creates struct that provides a method for importance sampling the sphere surface, the importance is determined by a latitude/longitude image with the north pole at (0,1,0).
- **::sample**
  - samples a point on the sphere's surface based on importance
- **::pdf**
  - given a `dir`, calculates the pdf for a direction (dir) under the importance sampling scheme

Uniform Sampling

- We use Monte Carlo sampling to estimate the incoming radiance at a particular location

- Uniform sampling samples the environment light sphere in any direction with equal probability
  - Means pdf is constant : $1/4\pi$
  - Recall pdf for uniform monte carlo hemisphere sampling is $1/2\pi$

- You are given sampling functions to sample a hemisphere, how can you use that to sample a sphere?

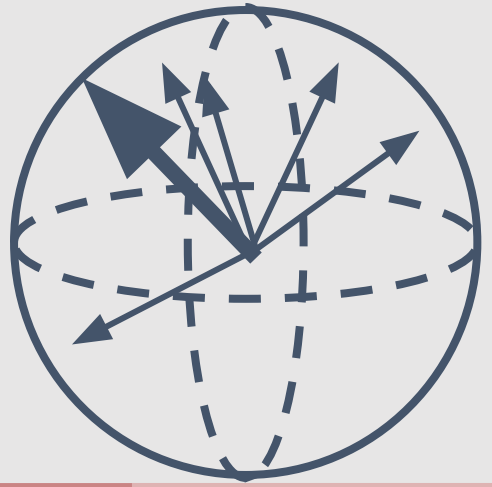


Importance Sampling

- Want to decrease the high variance of uniform sampling
  - Solution: Increase the chances of sampling a direction with higher radiance,
    - i.e. sample more in the direction of bright light sources
- Creating the pdf:
  - Assign probability to each pixel in the environment light texture according to the total radiance coming from the solid angle it subtends
    - Flux though pixel proportional to luma * sin $\theta \, d\theta \, d\varphi$
  - Divide by sum of flux to normalize the probabilities so they add up to 1

32 Uniform samples

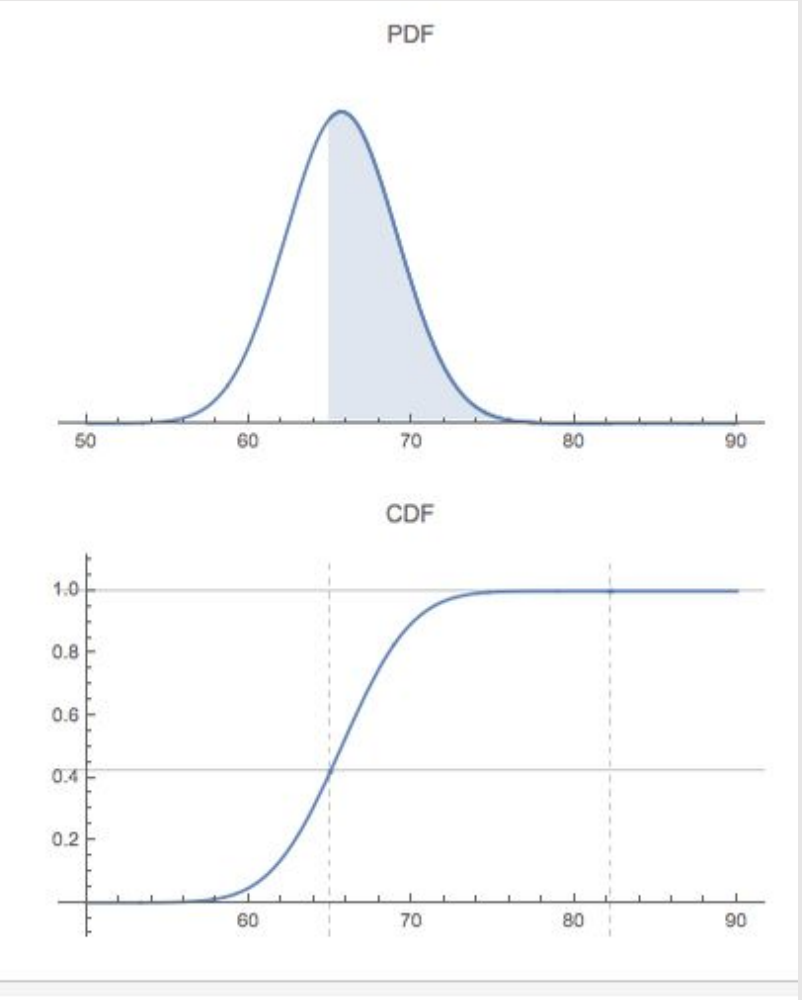
32 Importance samples



Importance Sampling - Computing the pdf & cdf

- We now have a discrete probability distribution
  - each pixel has a corresponding probability of it being sampled
  - How can we use this pdf to actually sample the pixels?

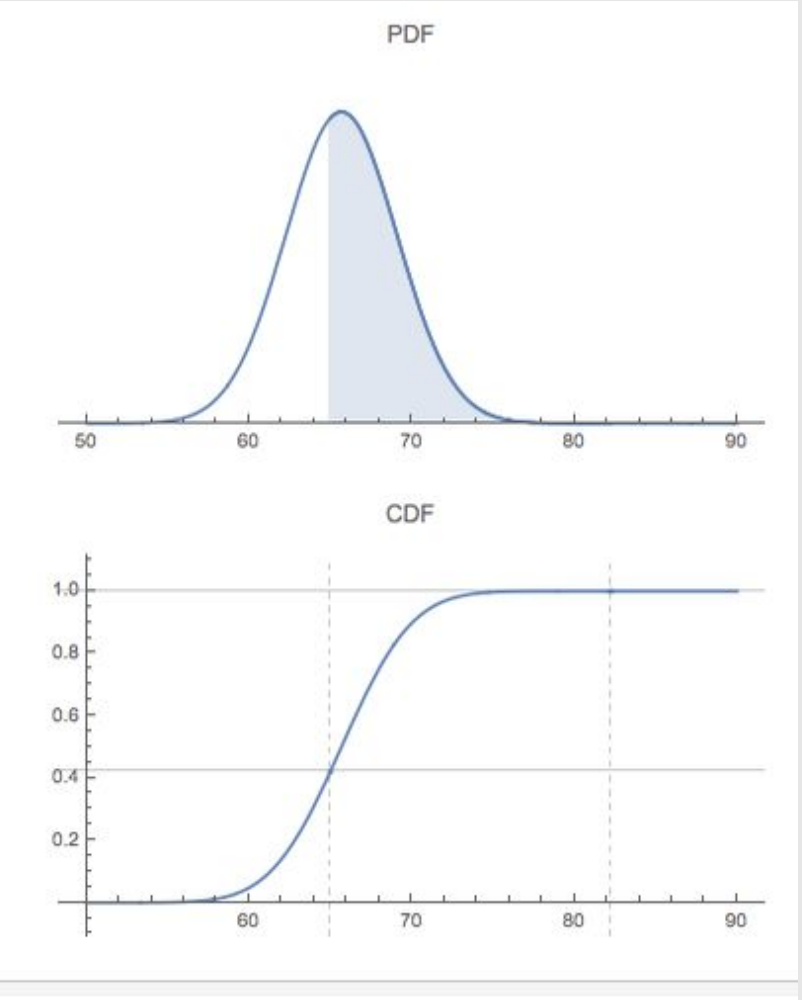
Use the CDF!



Importance Sampling - Computing the pdf & cdf

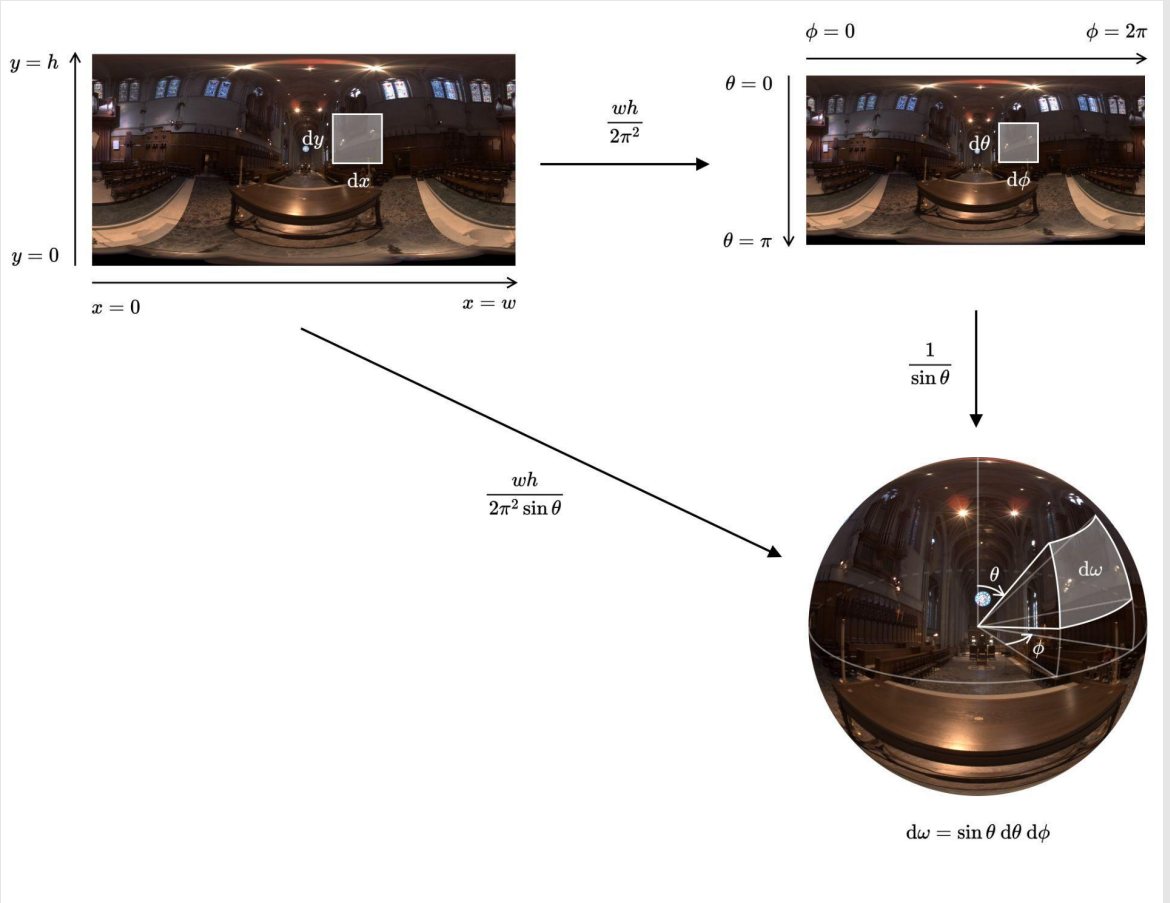
- What is a CDF?
  - gives probability that some random variable is less than or equal to x
    - i.e. probability some index from the environment map's pixel array is less than or equal to some x
    - Easy to do with a discrete distribution : cdf(x) is sum of pdf(y) for all y less than/equal to x

- With cdf, we can use inversion sampling to pick an index in the pixel array and convert it to a pixel & direction



Importance Sampling – Conversion

- Must take into account that solid angle each pixel in texture image subtends differs based on what location it represents on the unit sphere
  - i.e. pixels at top/bottom of texture take up less area than pixels near middle of image
- How do we do this?



Importance Sampling – Conversion

- Project rectangular coords to rectilinear coords, then rectilinear coords to unit sphere:
  1. Scale from w x h area to 2π x π area (rectilinear) by multiplying by ratio of areas
  2. Go from integrating over (dθ, dφ) to solid angle
     (Solid angle = sinθ dθ dφ, so divide by sinθ)