Lecture 24: Shading (Surface Scattering, Volume Scattering, and Texture Mapping)

Yuhao Zhu

http://yuhaozhu.com
yzhu@rochester.edu

CSC 259/459, Fall 2024
Computer Imaging & Graphics

The Roadmap

Theoretical Preliminaries

Human Visual Systems

Digital Camera Imaging

Modeling and Rendering

3D Modeling

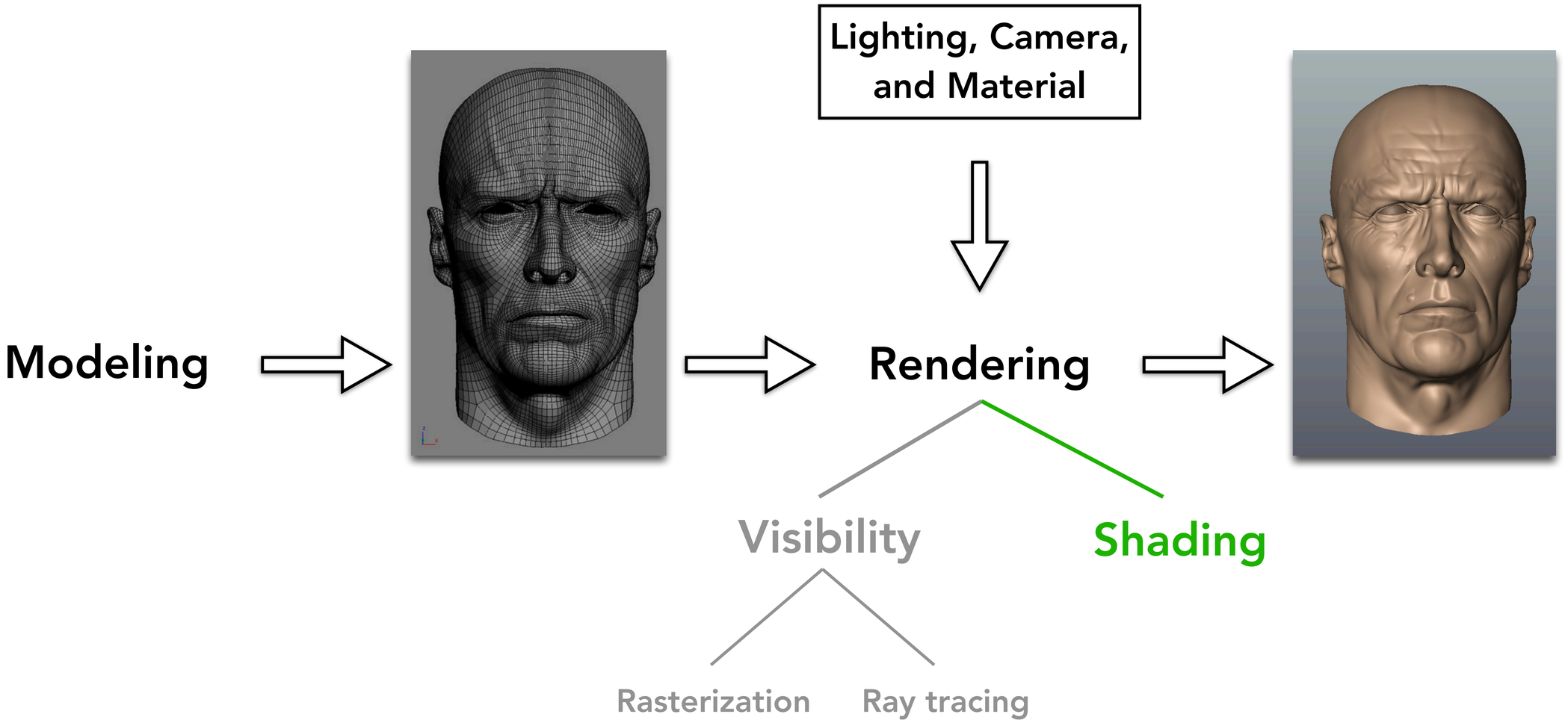
Rasterization and GPU

Ray Tracing

Shading

Rendering in AR/VR

Graphics



https://docs.blender.org/manual/en/dev/render/introduction.html http://www.cgarena.com/freestuff/tutorials/max/thomas_highway/sergeant.html

https://helpx.adobe.com/cy_en/dimension/using/apply-materials-images-to-models.html

Shading == Calculate Pixel Color

Approaches range from:

- pure hack (e.g., interpolation from vertices)
- empirical models (e.g., Blinn-Phong model)
- physically-based simulation of light-matter interaction (e.g., rendering equation)
- texture mapping (a form of pre-computation)

Photorealistic (visually pleasing) vs. physically real

Local vs. global illumination

- Do we consider lights only from light sources or also mutual, indirect illumination between objects?

Interpolation

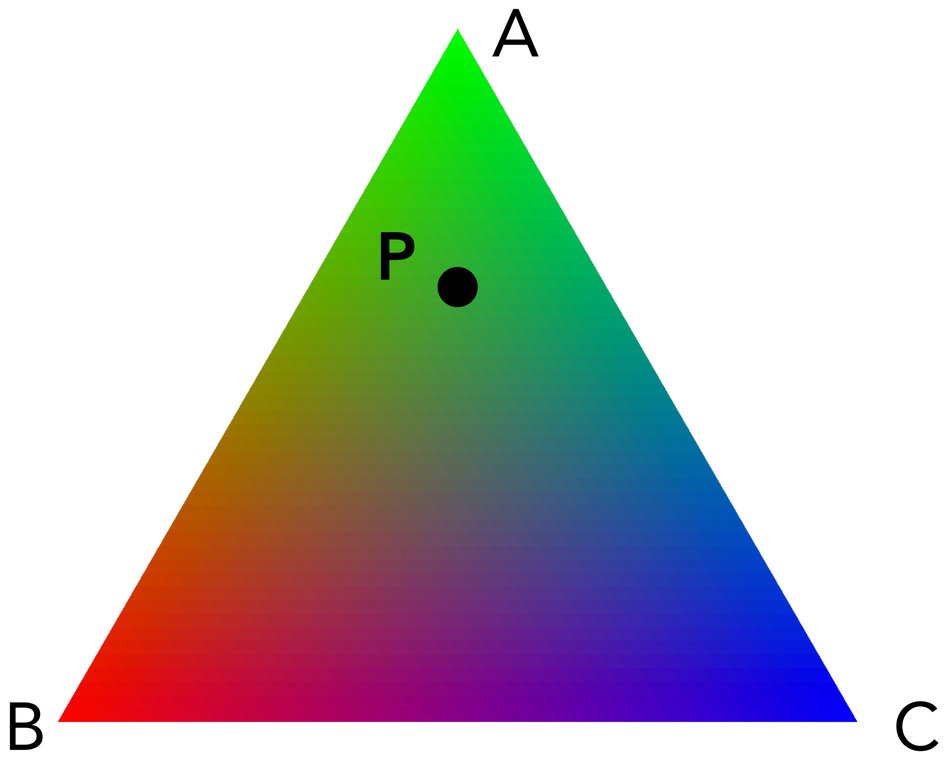
If you know the colors of the three vertices of a triangle and nothing else, how would you calculate the color of a point P inside the triangle?

Interpolate the color from the vertex colors (using barycentric coordinates).

- There is absolutely no reason why this has to be true.
- But perhaps the best bet if you have no other information.

$$C_P = \alpha C_A + \beta C_B + \gamma C_C$$

$$\alpha + \beta + \gamma = 1$$



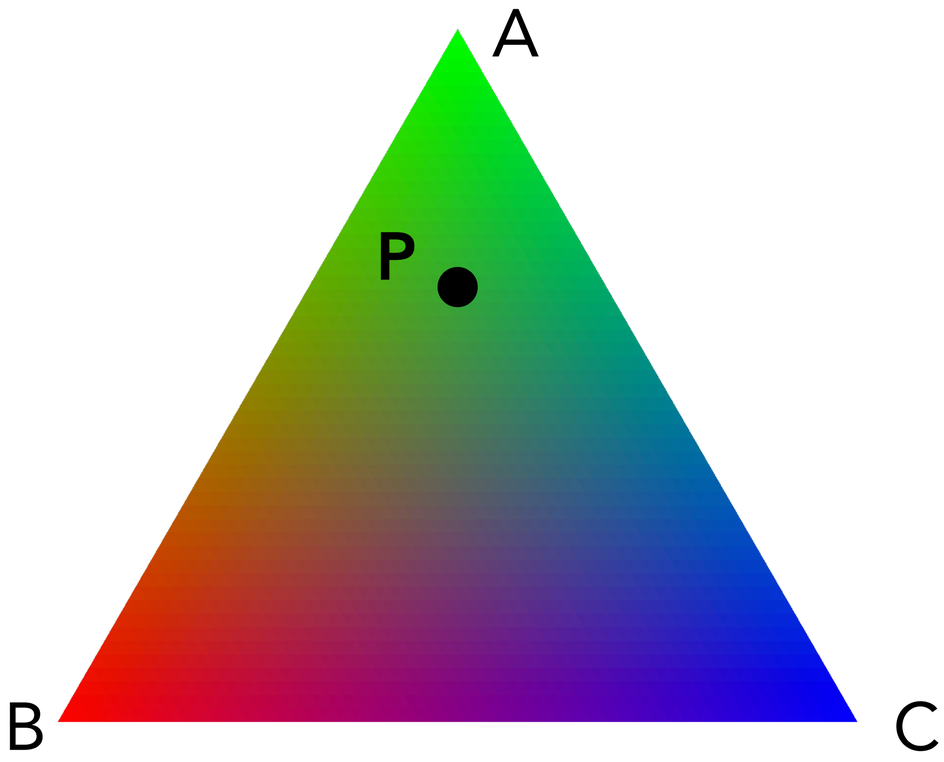
Interpolation

Interpolation is very useful in rendering. We interpolate many things: color, normal, texel coordinates, etc.

Interpolation assumes smooth changes, which will miss high-frequency signals, but it's the best bet when no other information is available.

$$C_P = \alpha C_A + \beta C_B + \gamma C_C$$

$$\alpha + \beta + \gamma = 1$$



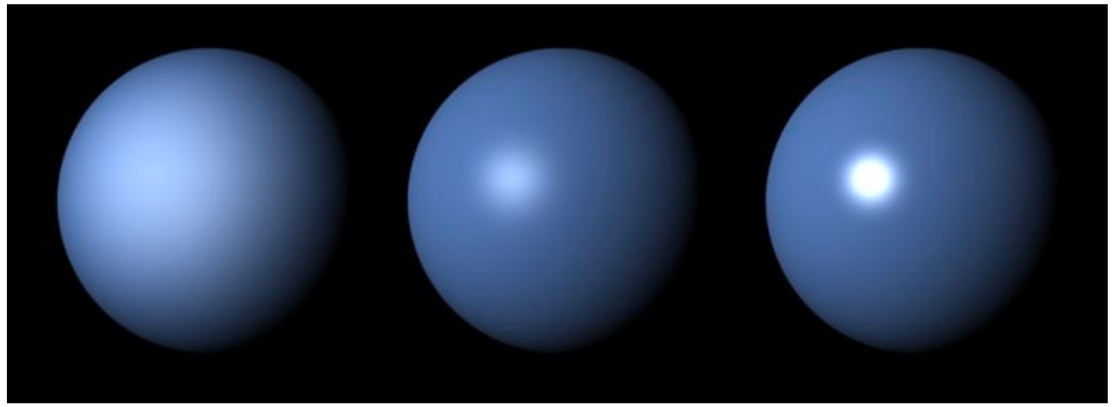
1942-1975

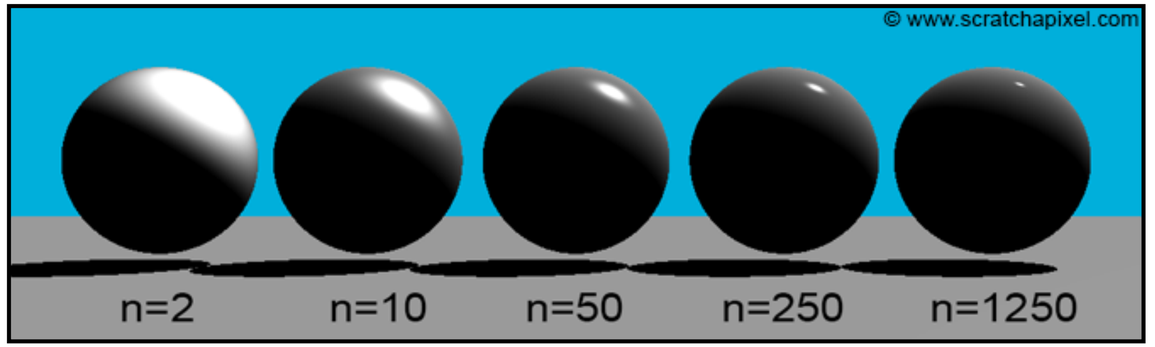
Phong Model

Phong Model

Physics-inspired but doesn't simulate actual light matter interaction; still, very widely used in practice.

- A **local** method that accounts for some degree of global illumination: a pixel's color is calculated based on the light source and an empirical term for indirect illumination.
- Has "fudge factors" (e.g., n below) that can be tuned for pleasing visual appearance.





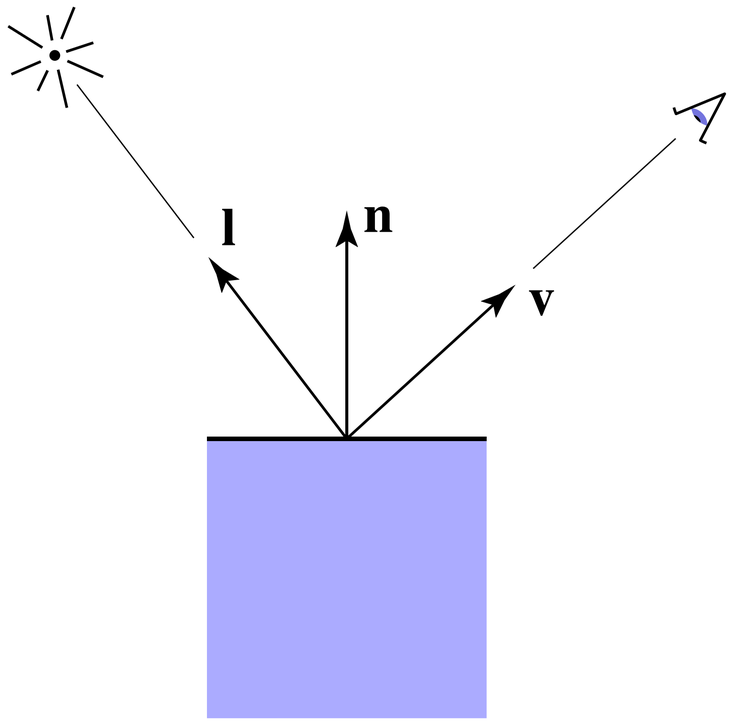
https://medium.com/shader-coding-in-unity-from-a-to-z/light-in-computer-graphics-2-3b2a5b04ac6d

A Local Model

Being a local method, when shading a point it doesn't need information from other points. Inputs are:

- Camera ray direction (**v**)
- Surface normal of the point (**n**)
- Incident light direction (**l**)
- Surface parameter (empirically modeled)

But we will see how it "hacks it way" to account for indirect illumination.



A Local Model

Can be easily plugged into either a rasterization or a ray-tracing pipeline.

```
foreach triangle in mesh
  perspective project triangle to canvas;
  foreach pixel in image
    if (pixel is in the projected triangle)
      pixel.color = PhongShade(…);
```

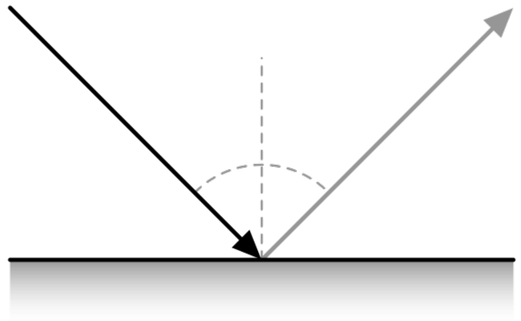
As part of a rasterization pipeline

As part of ray tracing. No recursion; a hint that Phong model isn't physically real!

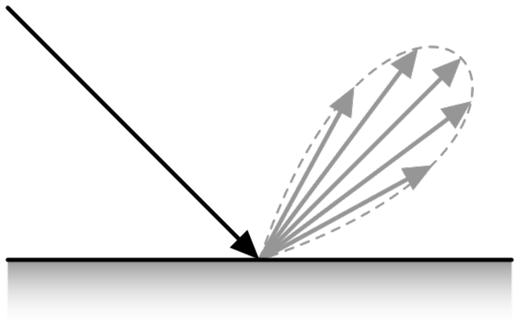
```
foreach pixel in image
  ray = buildRay(camera, pixel)
  if (P = intersect(ray, mesh))
    pixel.color = PhongShade(…)
  else
    pixel.color = backgroundColor
```

Basic Observation: Color Depends on Surface

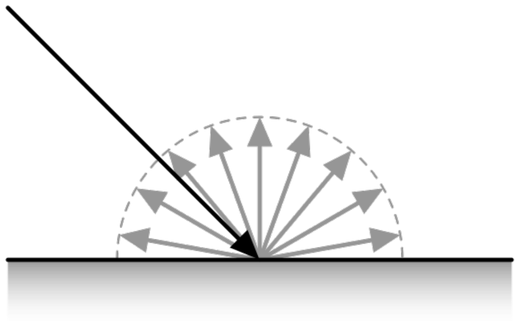
Mirror/Glass (Specular)



Glossy (Specular)



Diffuse (Not Specular)



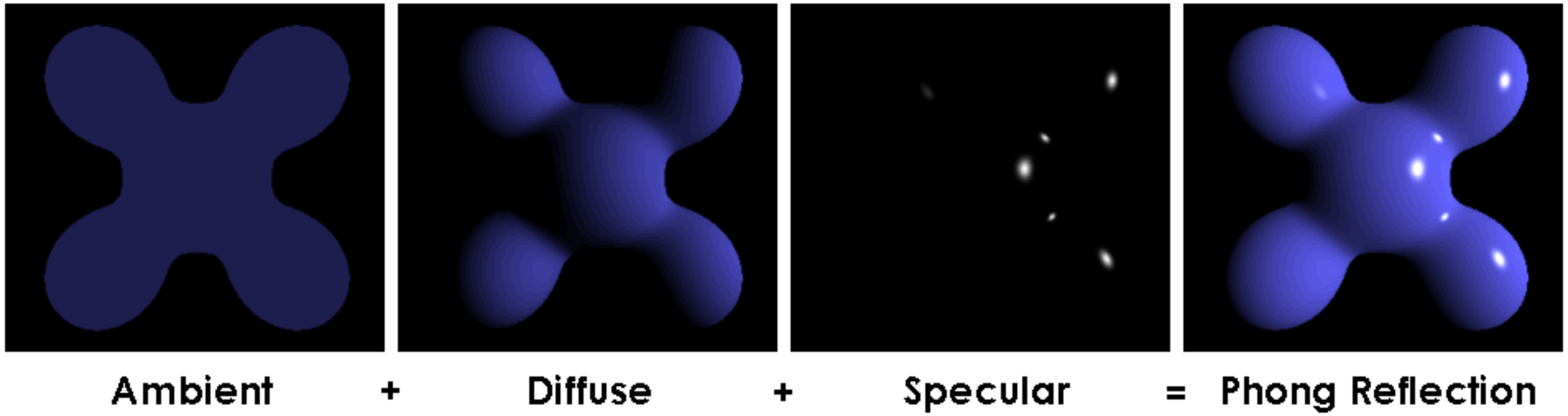
Phong Model

Model surface as a combination of specular and diffuse components.

- Assign/tune weights to adjust the proportion.

Add an ambient shading term to emulate indirect illumination.

- Again, that term is tunable.

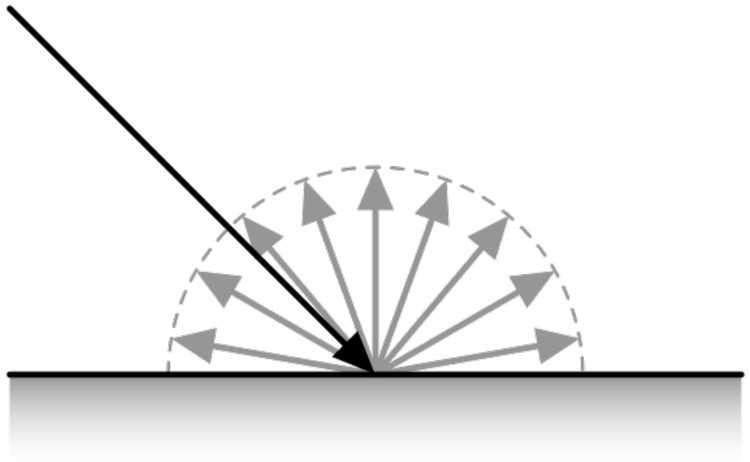


Diffuse Shading

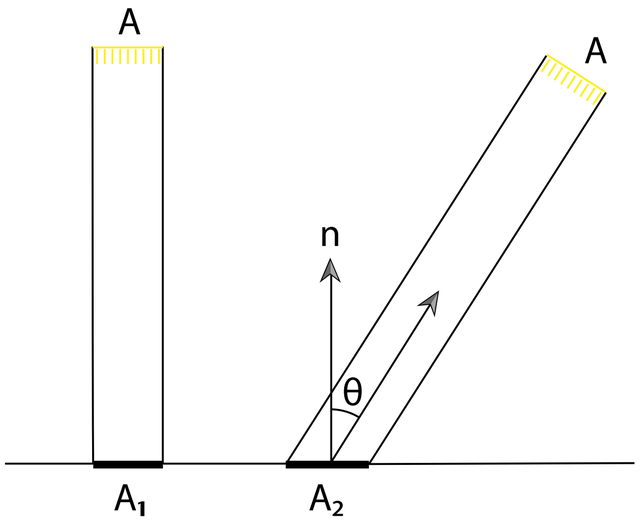
A perfectly diffuse surface distributes energy uniformly across all directions.

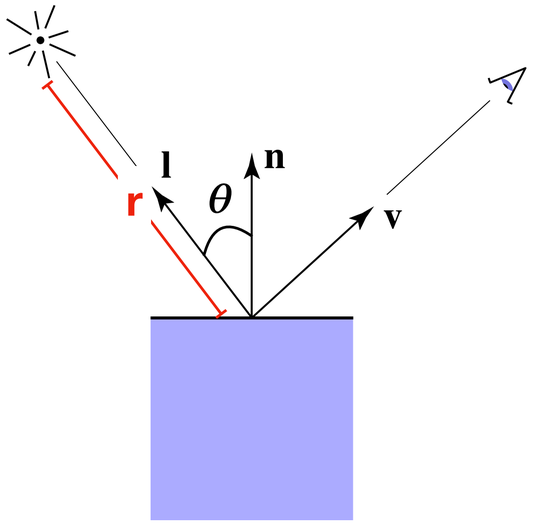
- The energy the surface receives from the incident light is weakened by the incident light angle (w.r.t. surface normal).

The color of a point thus doesn’t depend on viewing angle.



Diffuse Shading





Recall the Lambertian cosine law (from the emission/display lecture)

Diffuse component

Light intensity (falls off quadratically w.r.t. distance)

$$L_d = k_d \frac{I}{r^2} \cos\theta$$

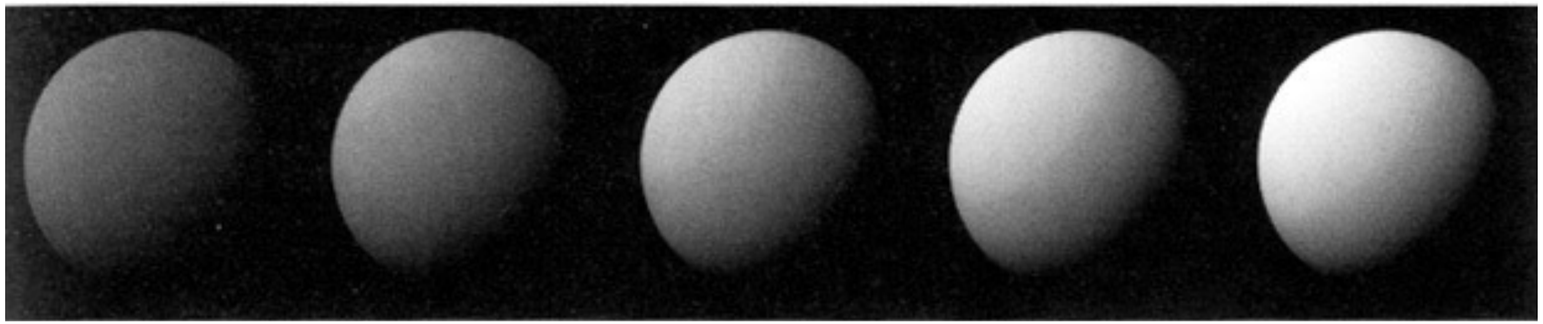
Tunable diffuse coefficient

(e.g., simulate how much energy is absorbed by the surface)

= dot product of the light angle and normal vector

$$l \cdot n$$

Diffuse Shading



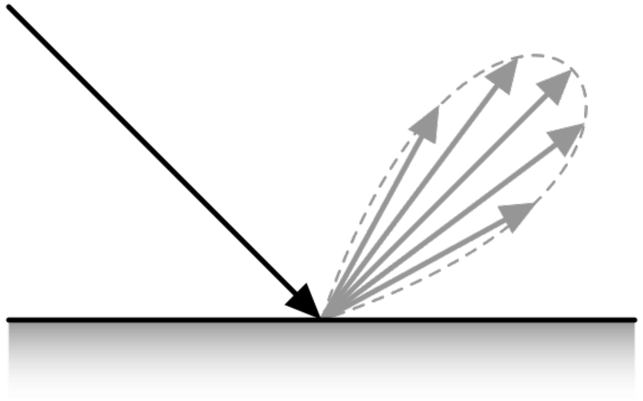
k_d ⟶

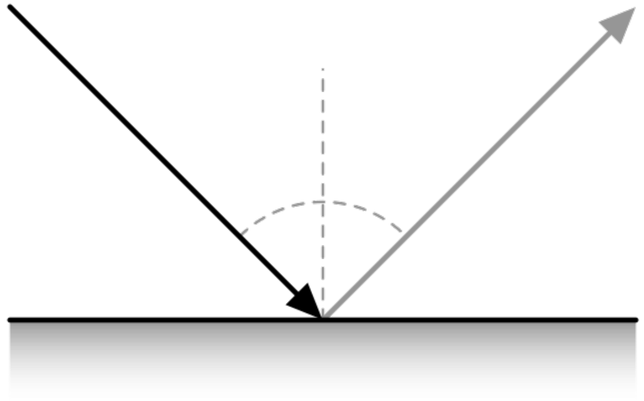
$$L_d = k_d \frac{I}{r^2} \cos\theta$$

Specular Shading

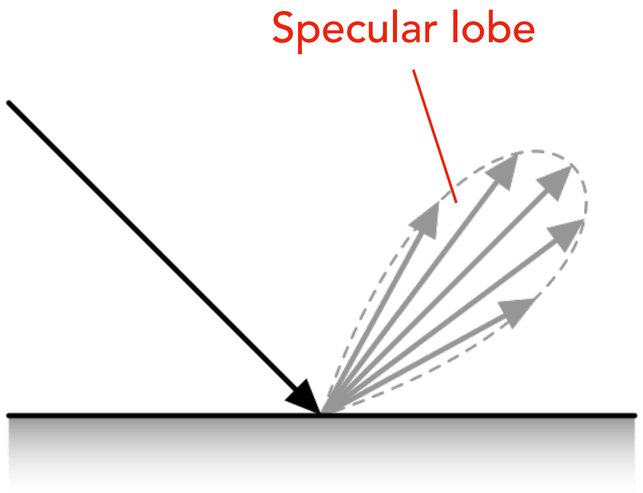
Color of a point *is* dependent on viewing angle.

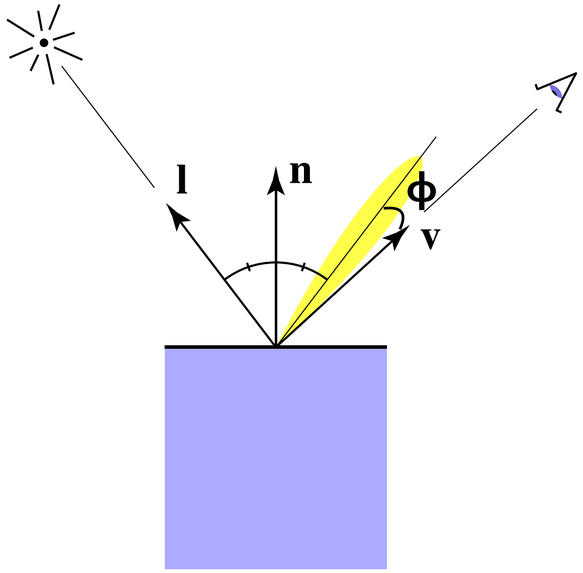
- For a perfect mirror, only when the viewing angle = incident angle will we see color.
- For a general specular surface, reduce the light intensity when the viewing angle moves away from the reflection angle.





Specular Shading





Another tunable coefficient controlling the side of the specular lobe

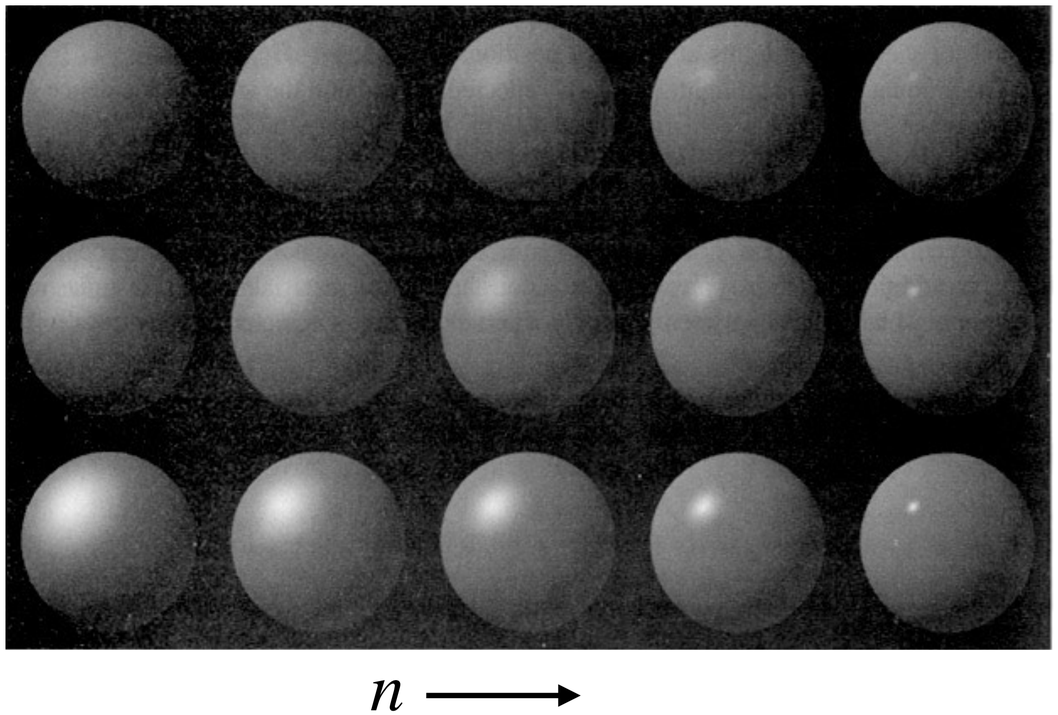
Specular component

$$L_s = k_s \frac{I}{r^2} (\cos \phi)^n$$

Tunable specular coefficient

(e.g., simulate how much energy is absorbed by the surface and distributed to viewing direction)

Specular Shading



Another tunable coefficient controlling the side of the specular lobe

Specular component

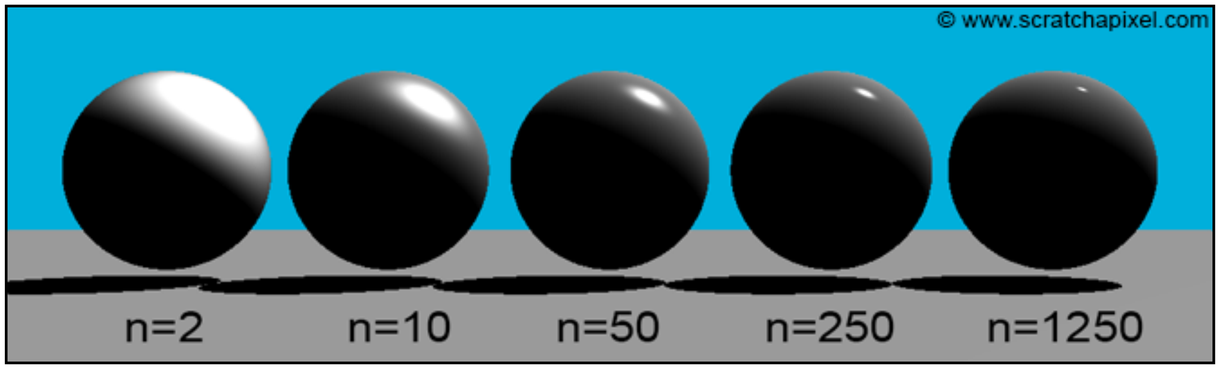
$$L_s = k_s \frac{I}{r^2} (\cos \phi)^n$$

Tunable specular coefficient

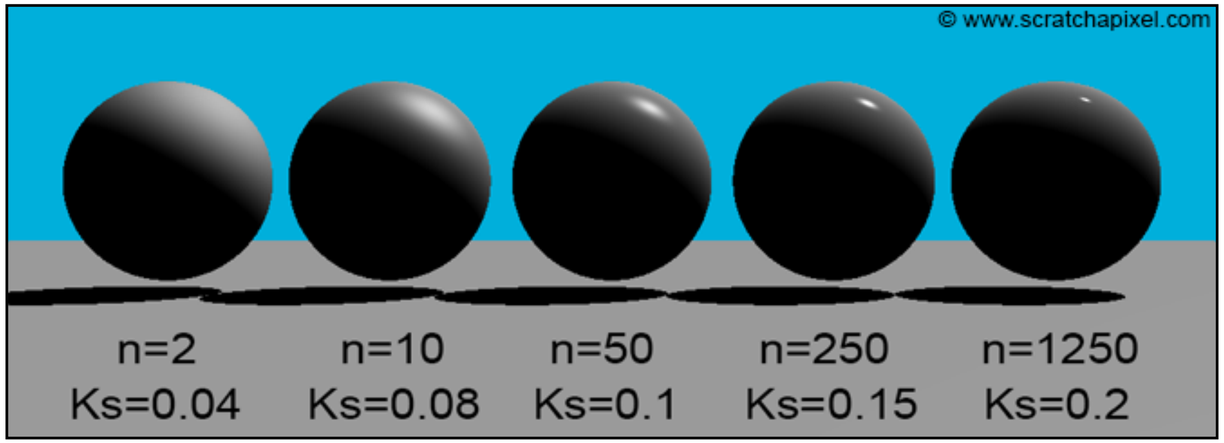
(e.g., simulate how much energy is absorbed by the surface)

Specular Shading

Same k_s. Are these rendering results physically plausible?



Tune n and k_s *together.*



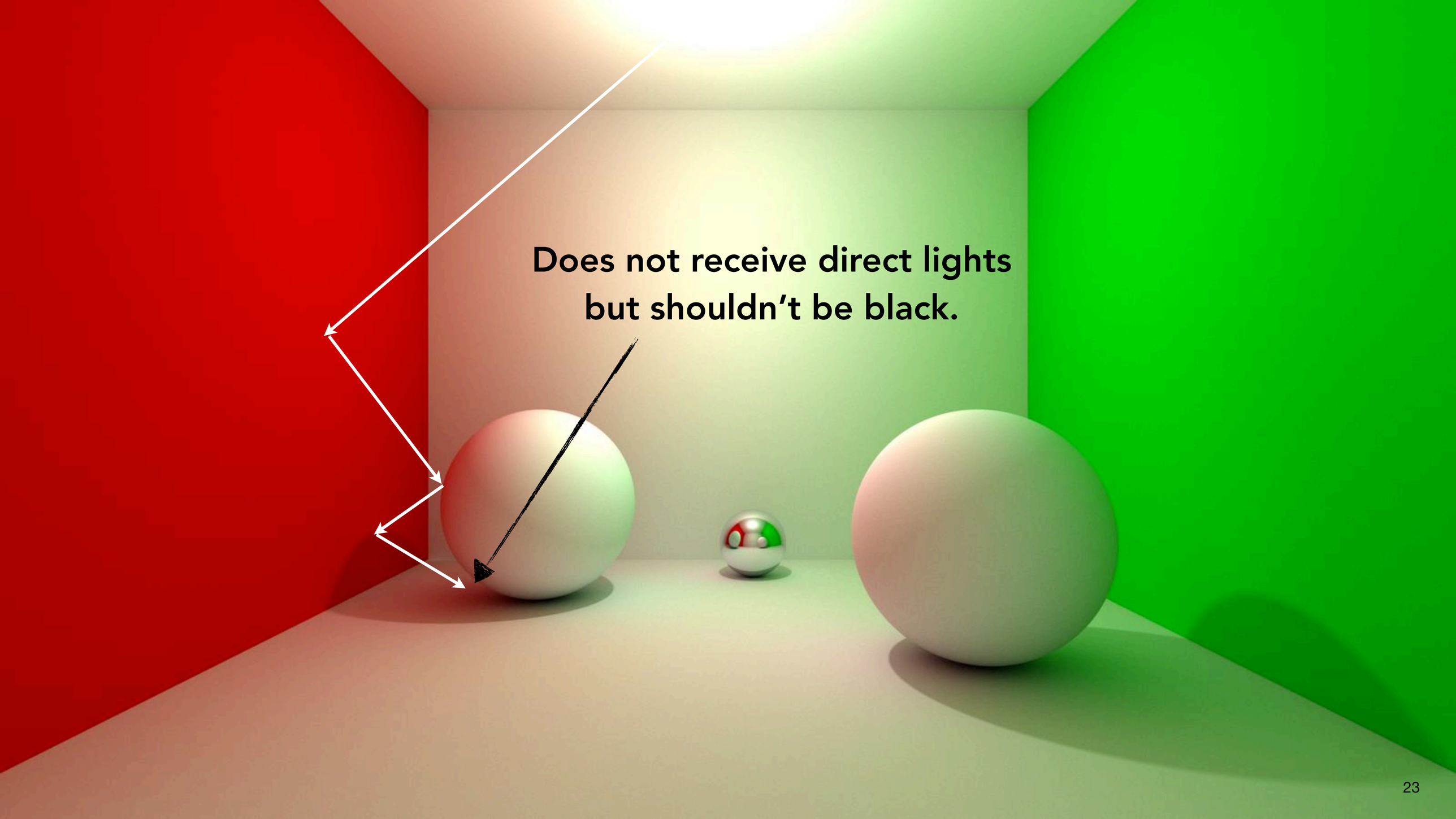
Another tunable coefficient controlling the side of the specular lobe

Specular component

$$L_s = k_s \frac{I}{r^2} (\cos \phi)^n$$

Tunable specular coefficient

(e.g., simulate how much energy is absorbed by the surface)

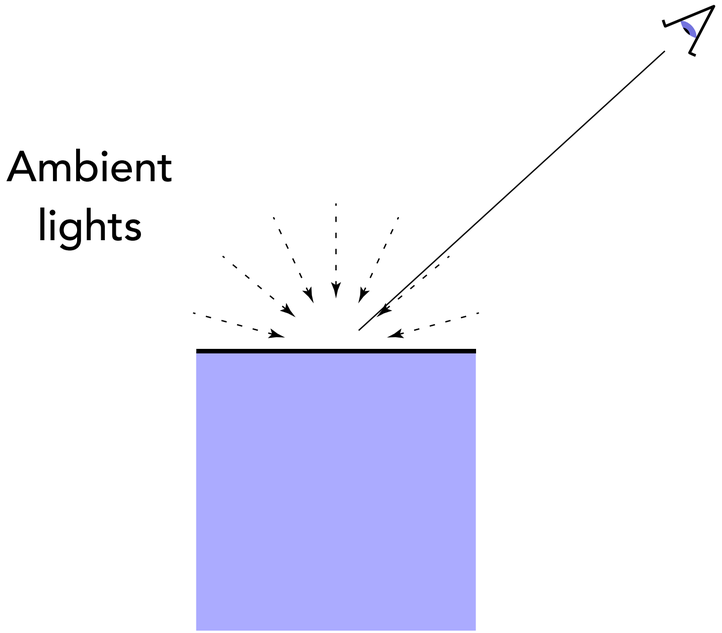

Does not receive direct lights
but shouldn’t be black.

http://filmicworlds.com/blog/occlusion-that-needs-forward-shading/

Ambient Shading

Add a constant term to model the effect of ambient lights (read: indirect illumination) just so that points in shadow are not completely black.

Accurately modeling ambient lights requires recursive ray tracing.



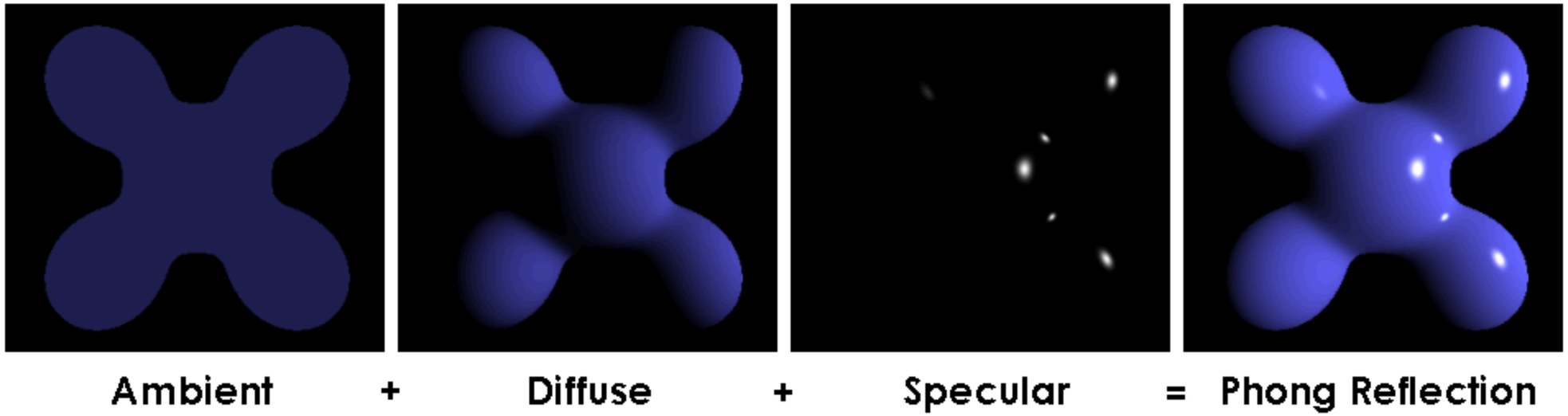
Ambient component

Tunable ambient light intensity

$$L_a = k_a I_a$$

Tunable ambient coefficient

Phong Model Summary



A mix of hack + empirical models + physics:

- Diffuse is physics-based; specular component is a physics-inspired empirical model; ambient component is purely a hack.
- A **local** method that accounts for **global** illumination through the (rather hacky) ambient component.

Where/When to Apply Phong Model?

For every triangle (flat shading):

- Apply Phong model to each triangle on the mesh; each triangle has a normal vector; all points on the triangle share the same color.

For every vertex (Gouraud shading):

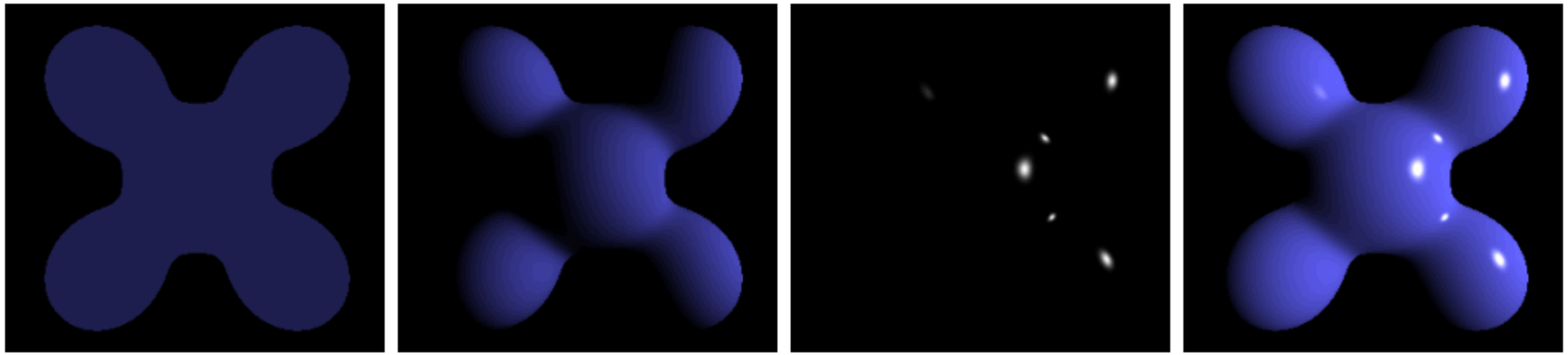
- Apply Phong model to each vertex on the mesh; each vertex has a normal vector; other points on the triangle are interpolated from vertices.

For every point (“Phong” shading):

- Each point on the triangle has a normal; point normal is interpolated from vertex normal; apply Phone model to each point.

Rendering Equation for Surface Scattering

Revisiting Phong Model

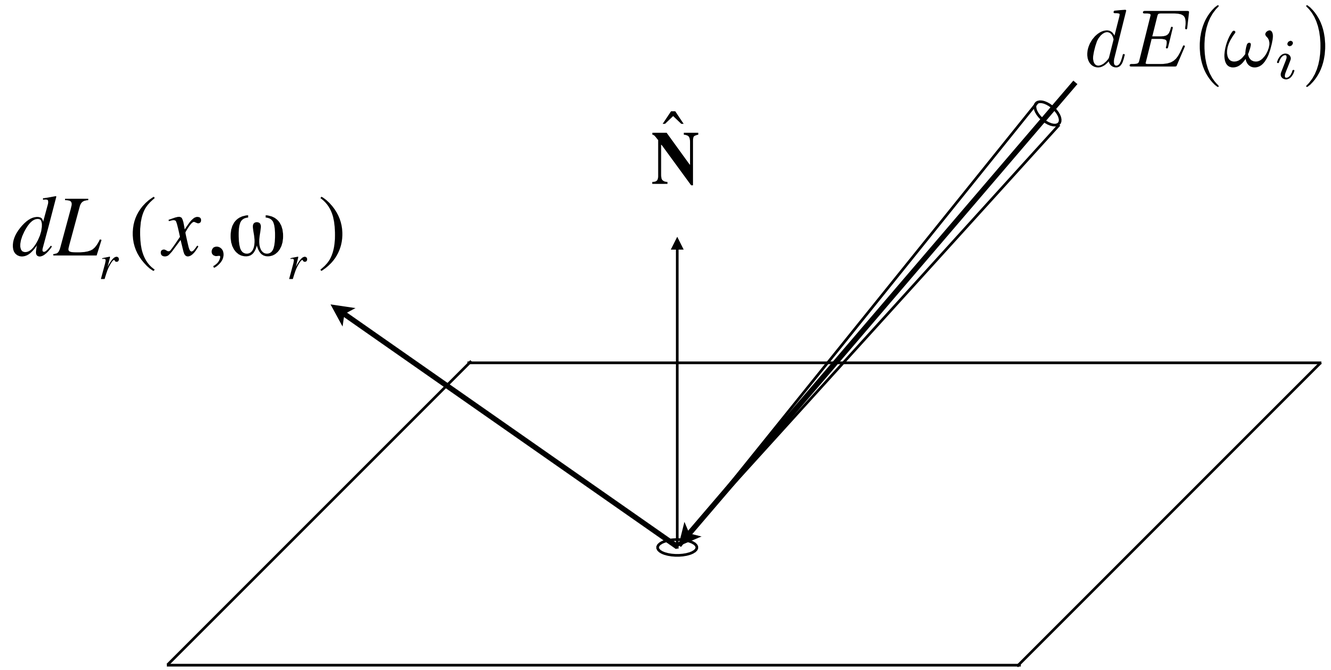


These terms are pure hacks

$$L_a = k_a I_a \qquad L_d = k_d \frac{I}{r^2} \cos\theta \qquad L_s = k_s \frac{I}{r^2} (\cos\phi)^n$$

Diffuse and specular coefficients are physics-inspired

Reflection at a Point

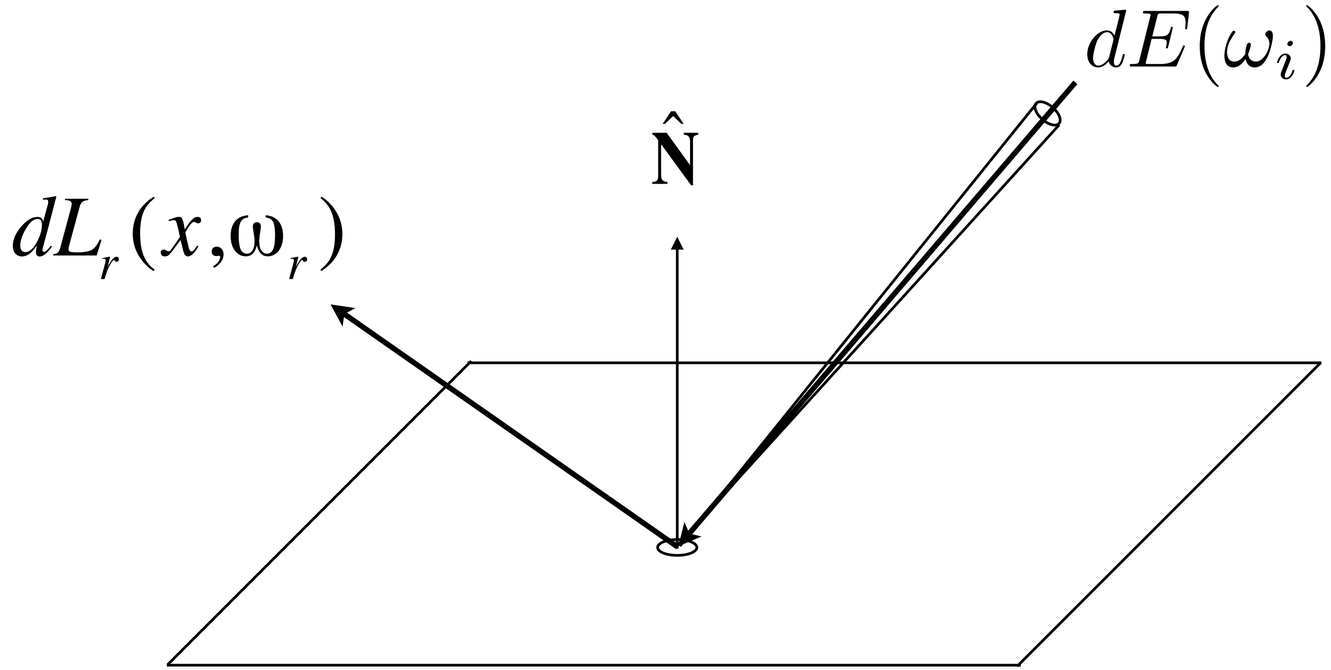


Differential irradiance incoming: $dE(\omega_i) = L(\omega_i)\, cos\, \theta_i\, d\omega_i$

Differential radiance exiting (due to $dE(\omega_i)$) $dL_r(\omega_r)$

Reflectance of a Point

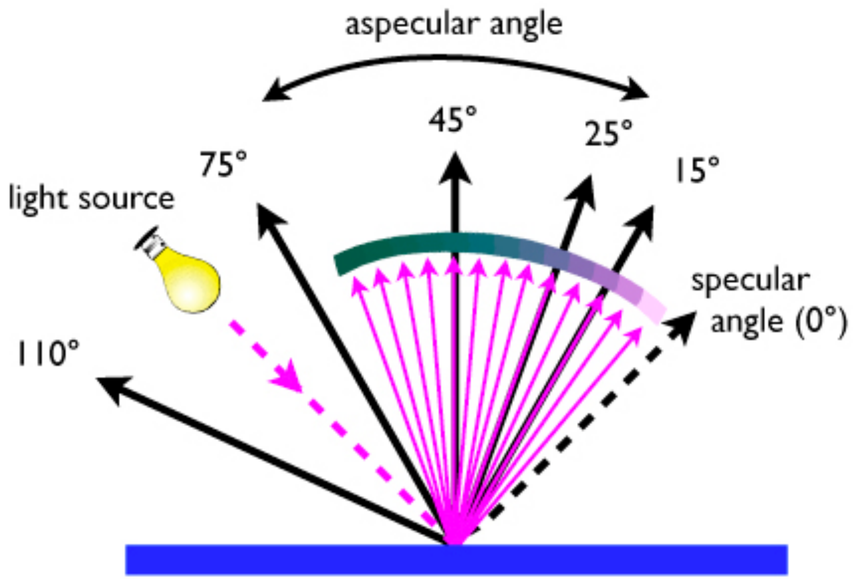
Bidirectional Reflectance Distribution Function (BRDF)

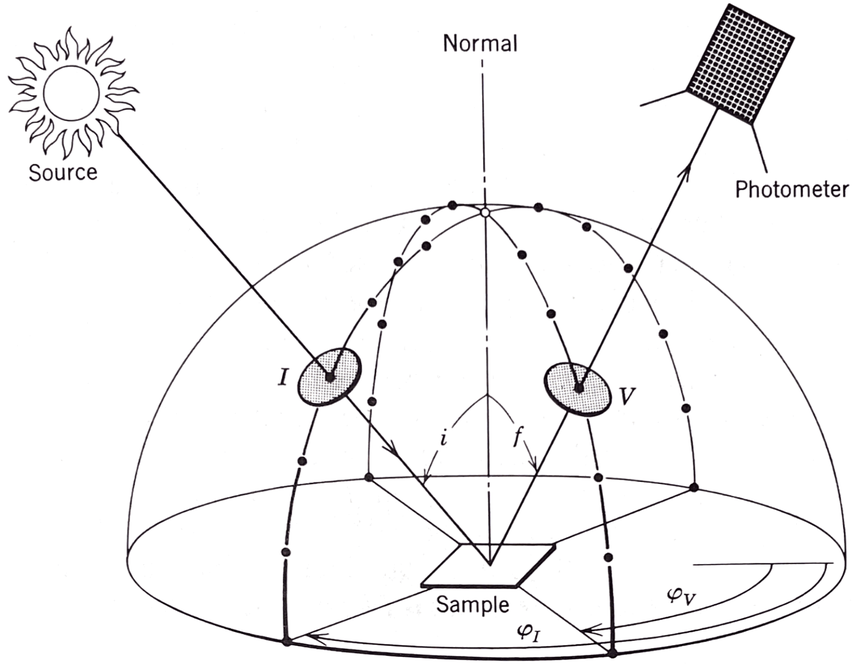


Informally, BRDF quantifies the energy transfer from an incident ray to an existing ray

$$f_r(\omega_i \rightarrow \omega_r) = \frac{dL_r(\omega_r)}{dE_i(\omega_i)}$$

Measuring BRDF Using Gonio Spectrophotometer

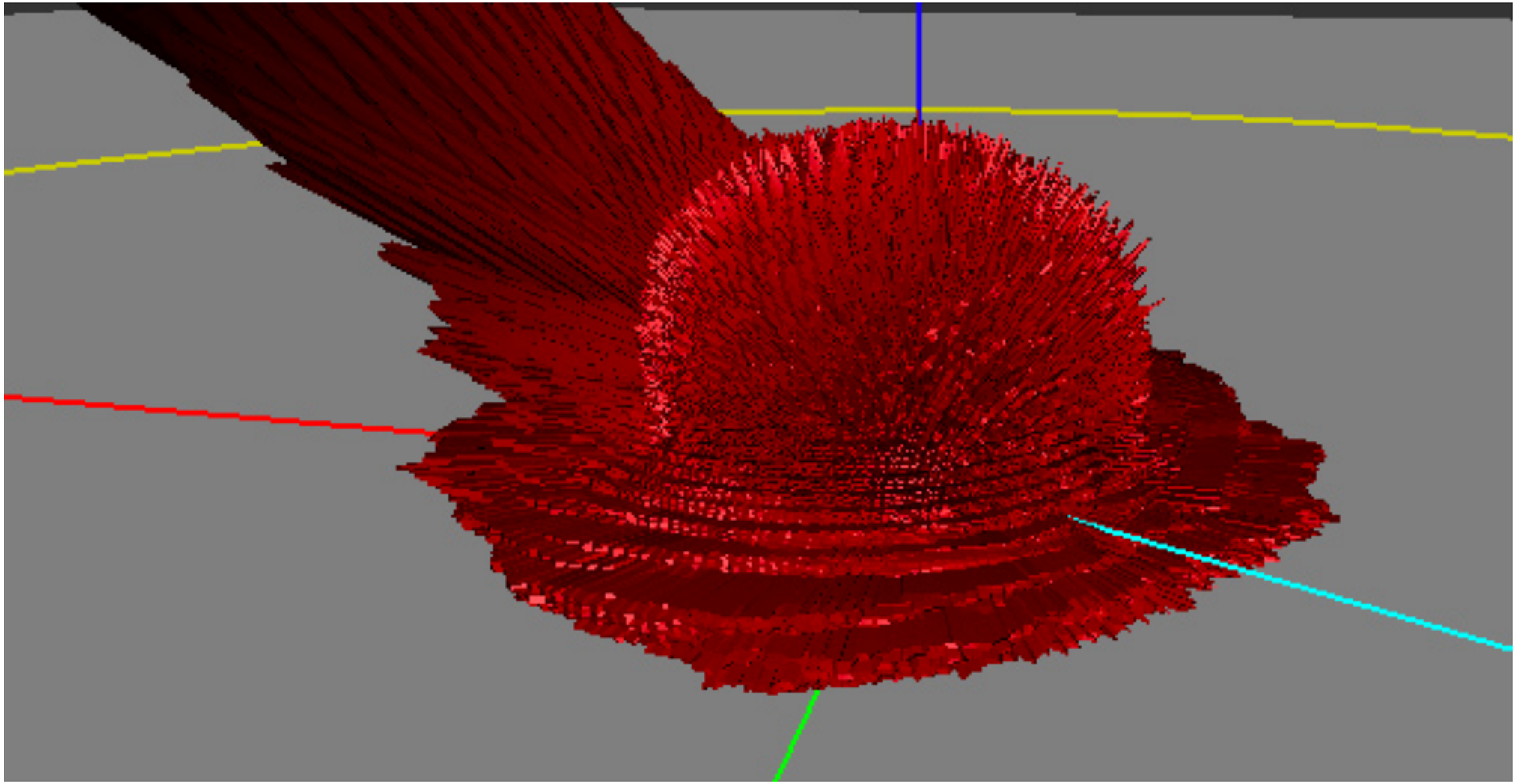




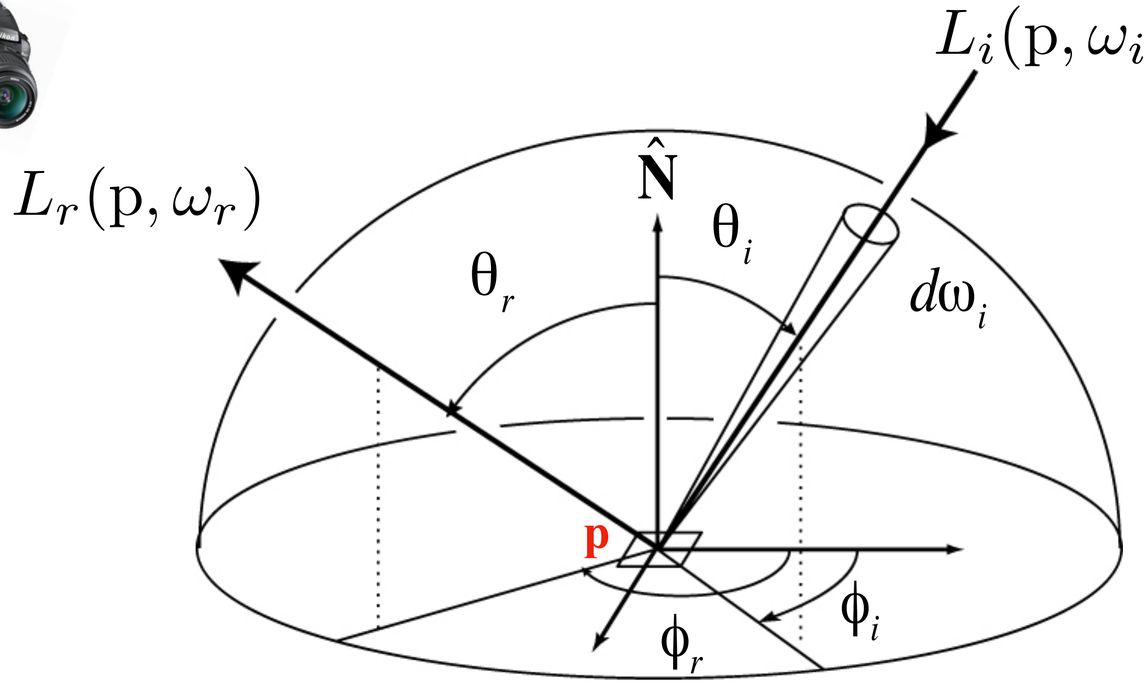
Color in Business, Science, and Industry 3ed, Judd and Wyszecki https://www.avianrochester.com/gonioapparent.php

https://rgl.epfl.ch/pages/lab/pgII

Material == BRDF (So Far)



The Rendering Equation



The incident light can be from a light source, or could be indirect illumination from other objects.

Idea: the energy of an exiting ray is contributed by that of all the incident rays, each modulated by the corresponding BRDF.

$$L_r(p, \omega_r) = \int_{\Omega} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i$$

The total exiting radiance at **p** toward $\boldsymbol{\omega_r}$

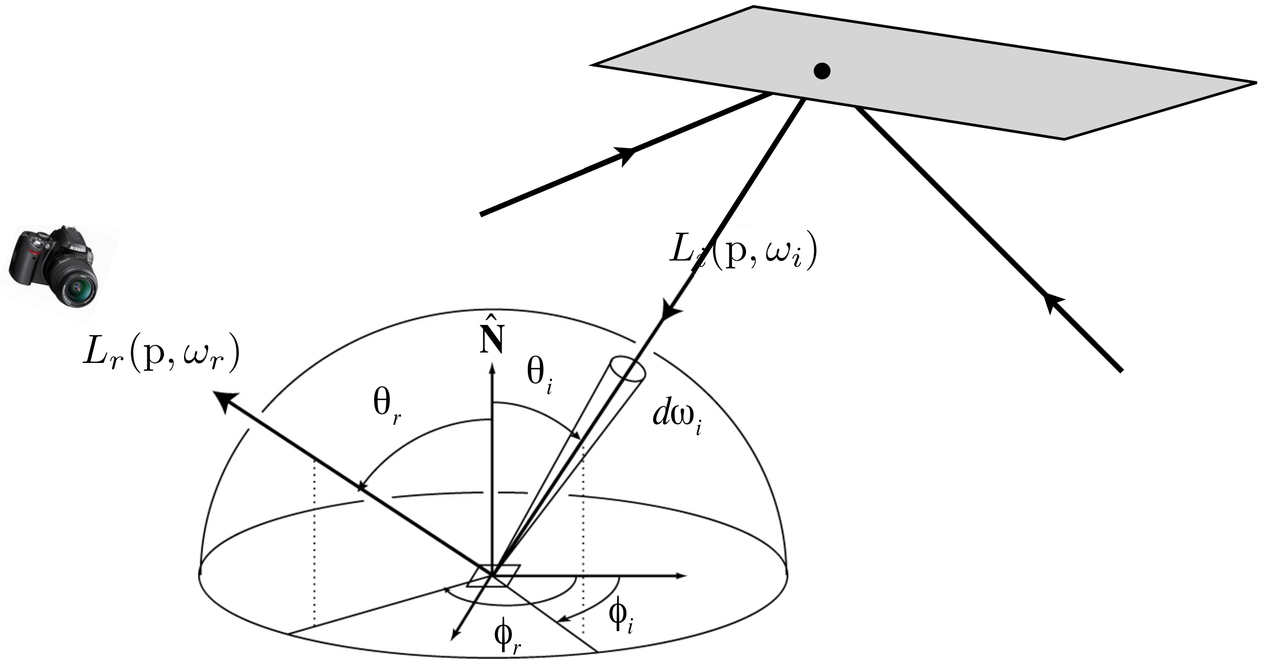
Integrate over the hemisphere

Solving the Rendering Equation Needs Recursion

$$L_r(p, \omega_r) = \int_\Omega f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i$$

Radiance appears on both sides of the equation: recursive ray tracing!

Exiting radiance depends on incident radiances. To calculate an incident radiance, we need to recursively solve the rendering equation!



How to Integrate?

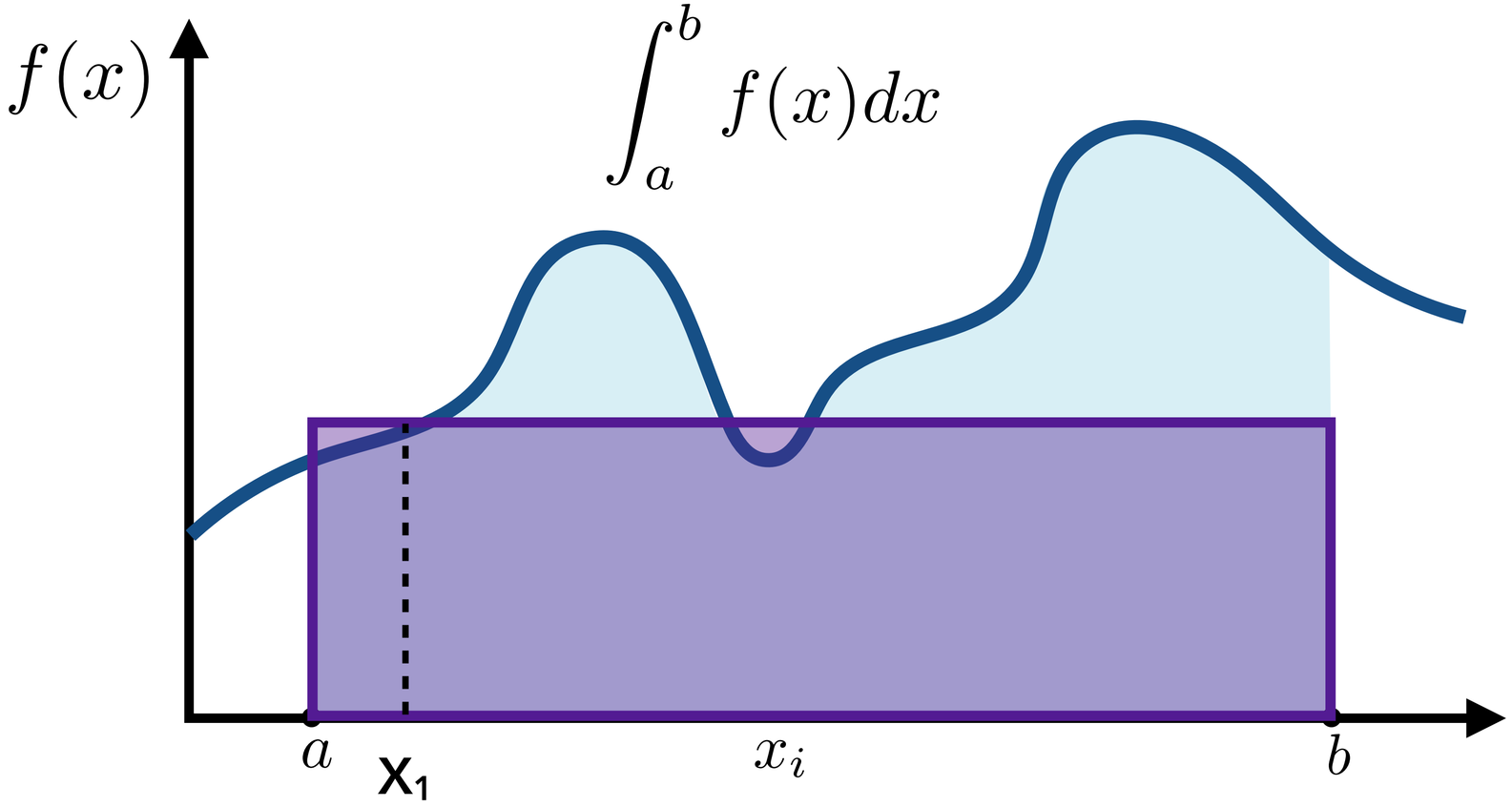
$$L_r(p, \omega_r) = \int_{\Omega} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i$$

The integrand has no analytical form

We don't know the analytical form of the integrand. How do we integrate?

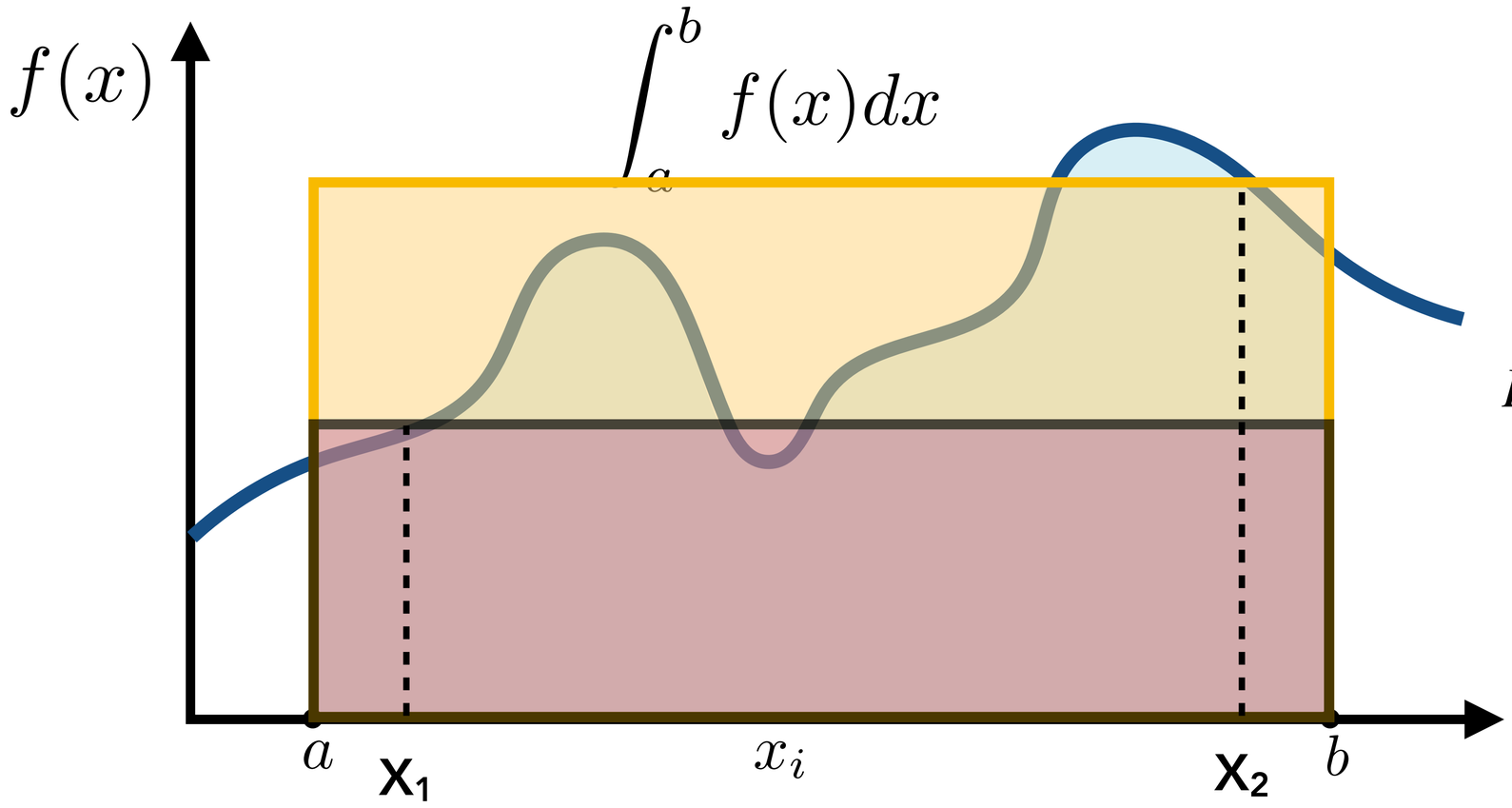
- We know the analytical form of BRDF, but most likely not the incident radiance term (the whole point of solving the rendering equation is to calculate each incident radiance!)

Integrate Over an Unknown Function



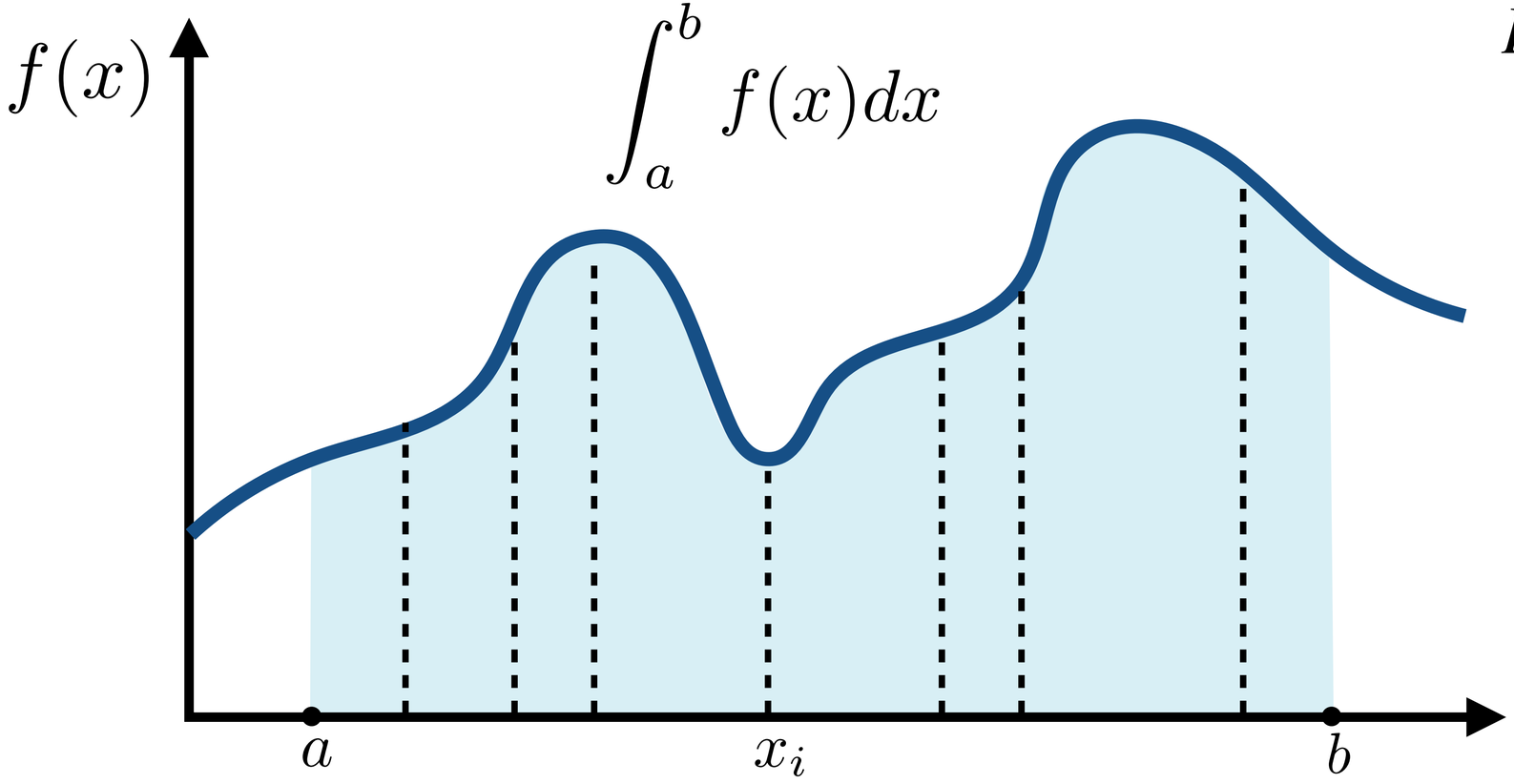
$$F = f(X_1)(b - a)$$

Integrate Over an Unknown Function



$$F = \frac{1}{2}(f(X_1)(b-a) + f(X_2)(b-a))$$

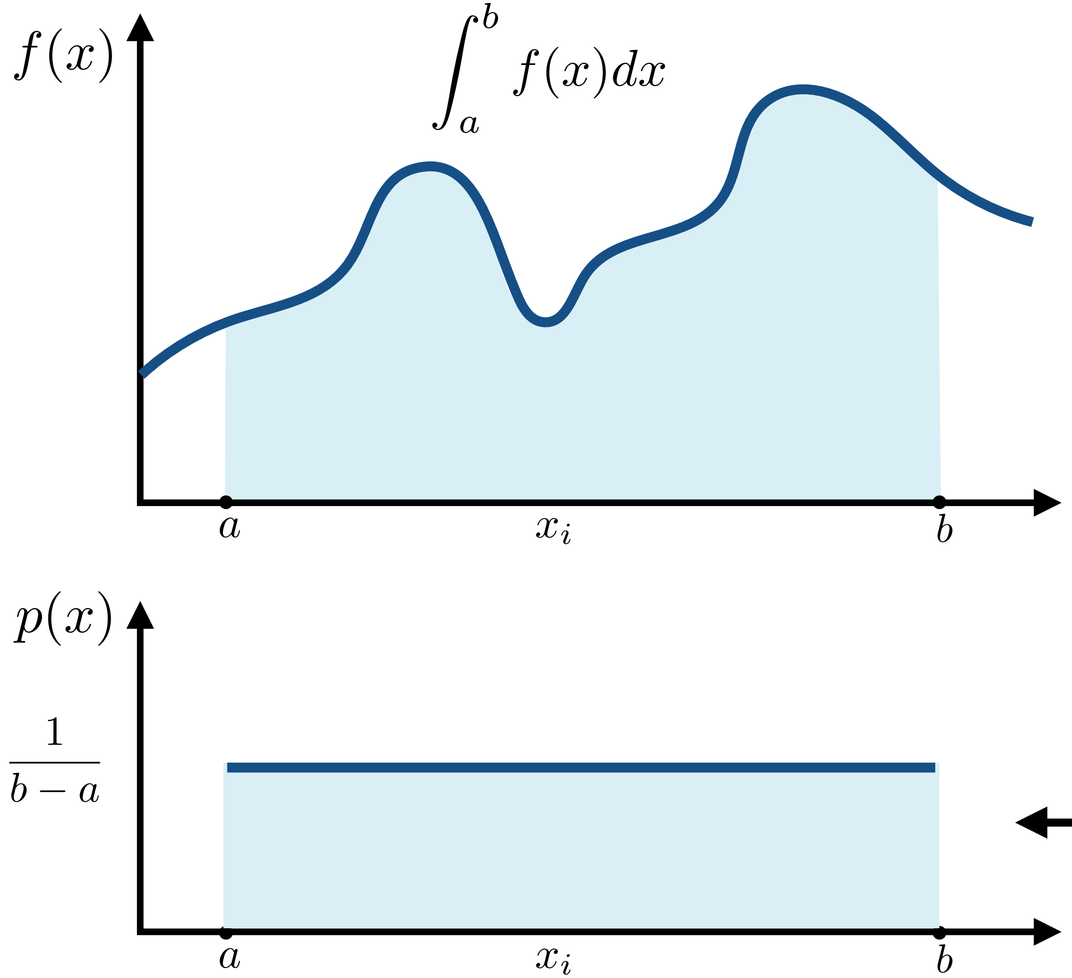
Integrate Over an Unknown Function



$$F_N = \frac{1}{N}\sum_{i=1}^{N} f(X_i)(b-a)$$

Uniformly at random take N samples, evaluate at each sample point, and take the average.

Integration Through Sampling

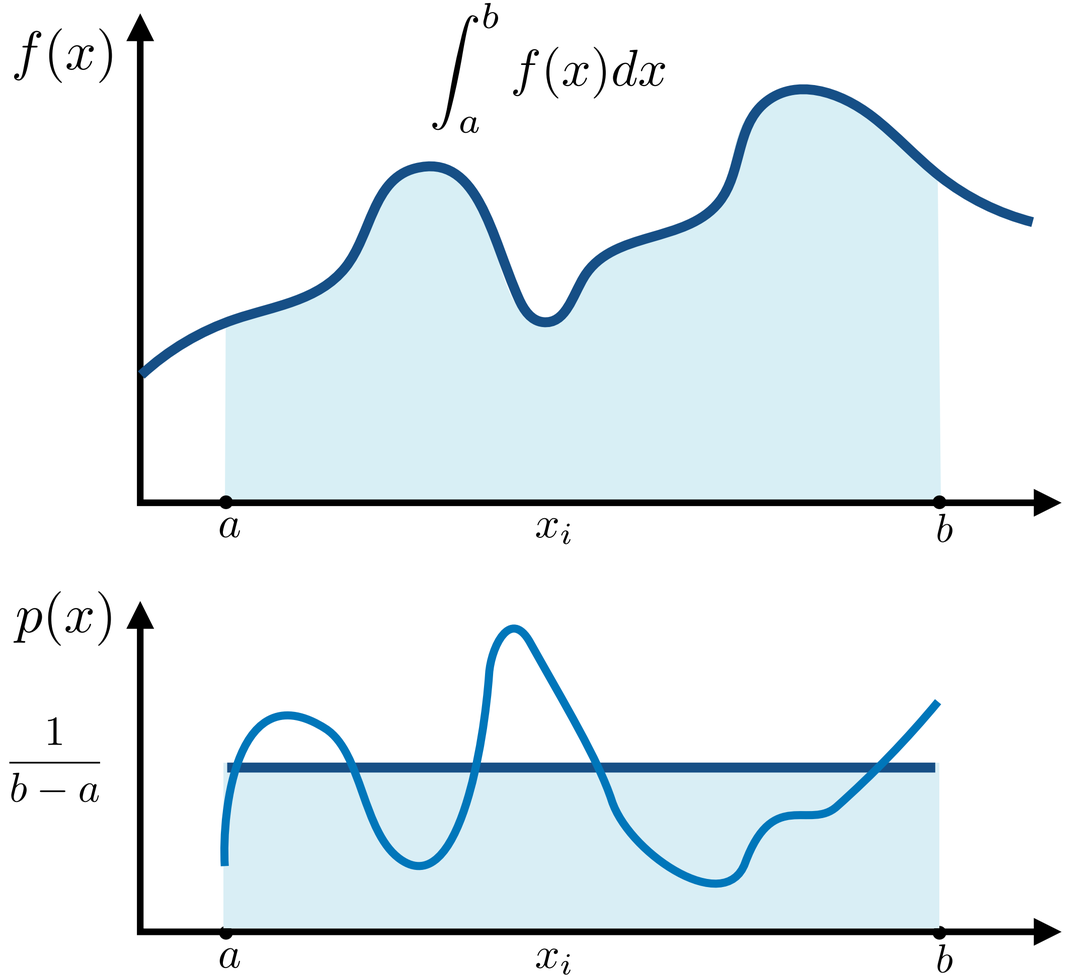


$$F_N = \frac{1}{N}\sum_{i=1}^{N} f(X_i)(b-a)$$

$$F_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{\frac{1}{(b-a)}}$$

The probability density function (PDF) of sampling

Generalize to Arbitrary Sampling Function

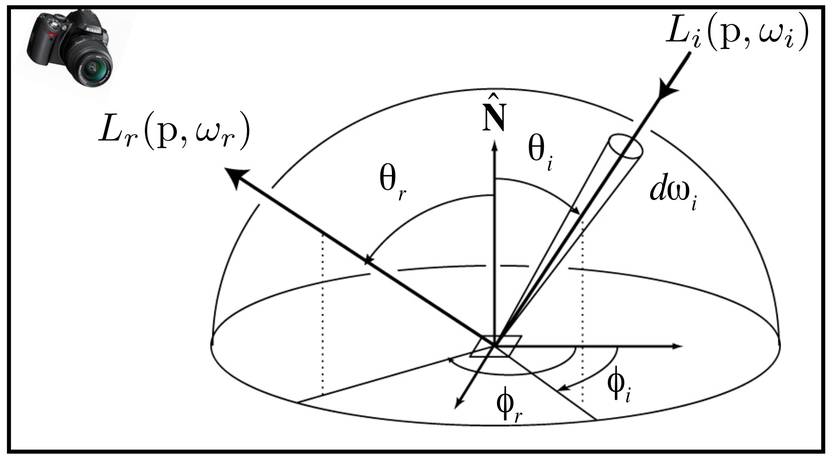


Monte Carlo Estimator

$$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$$

where $\int_a^b p(x)dx = 1$

Solving the Rendering Equation



$$L_r(p, \omega_r) = \int_\Omega f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i$$

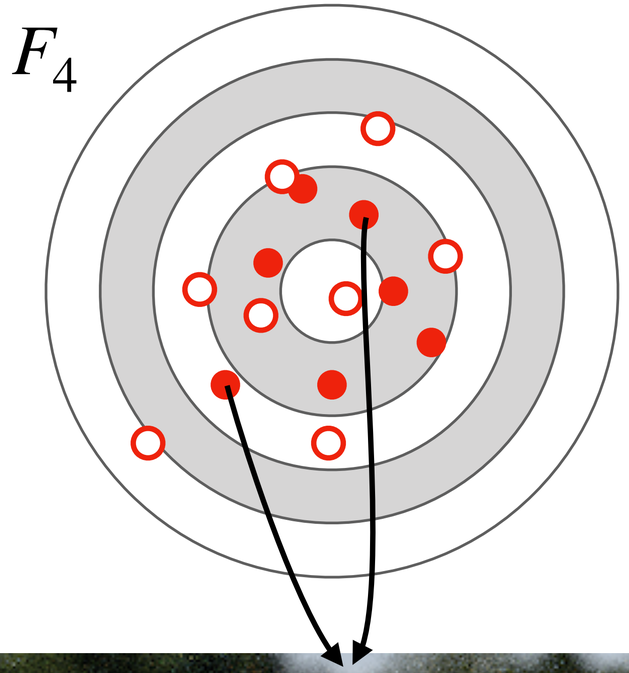
Monte Carlo Estimator

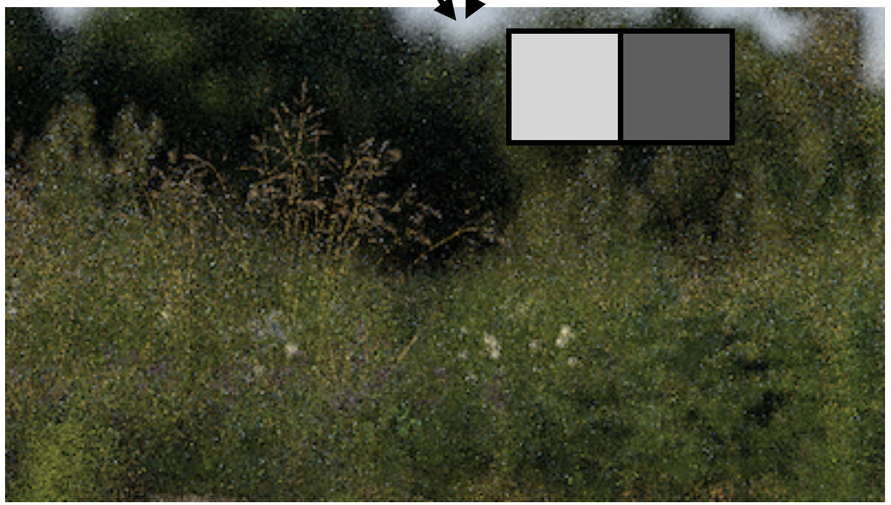
$$\frac{1}{N} \sum_1^N \frac{f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i}{p(\omega_i)}$$

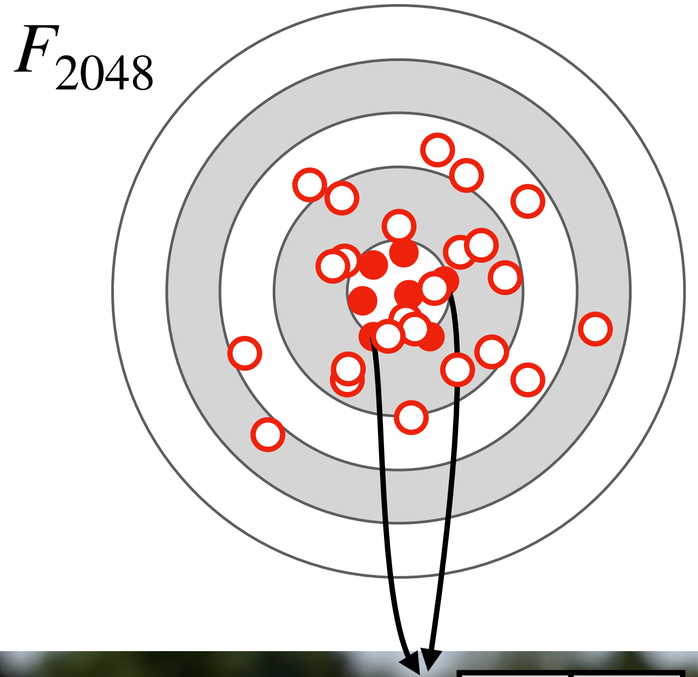
Probability density used to sample incident rays

Impact of Amount of Samples

$$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$$



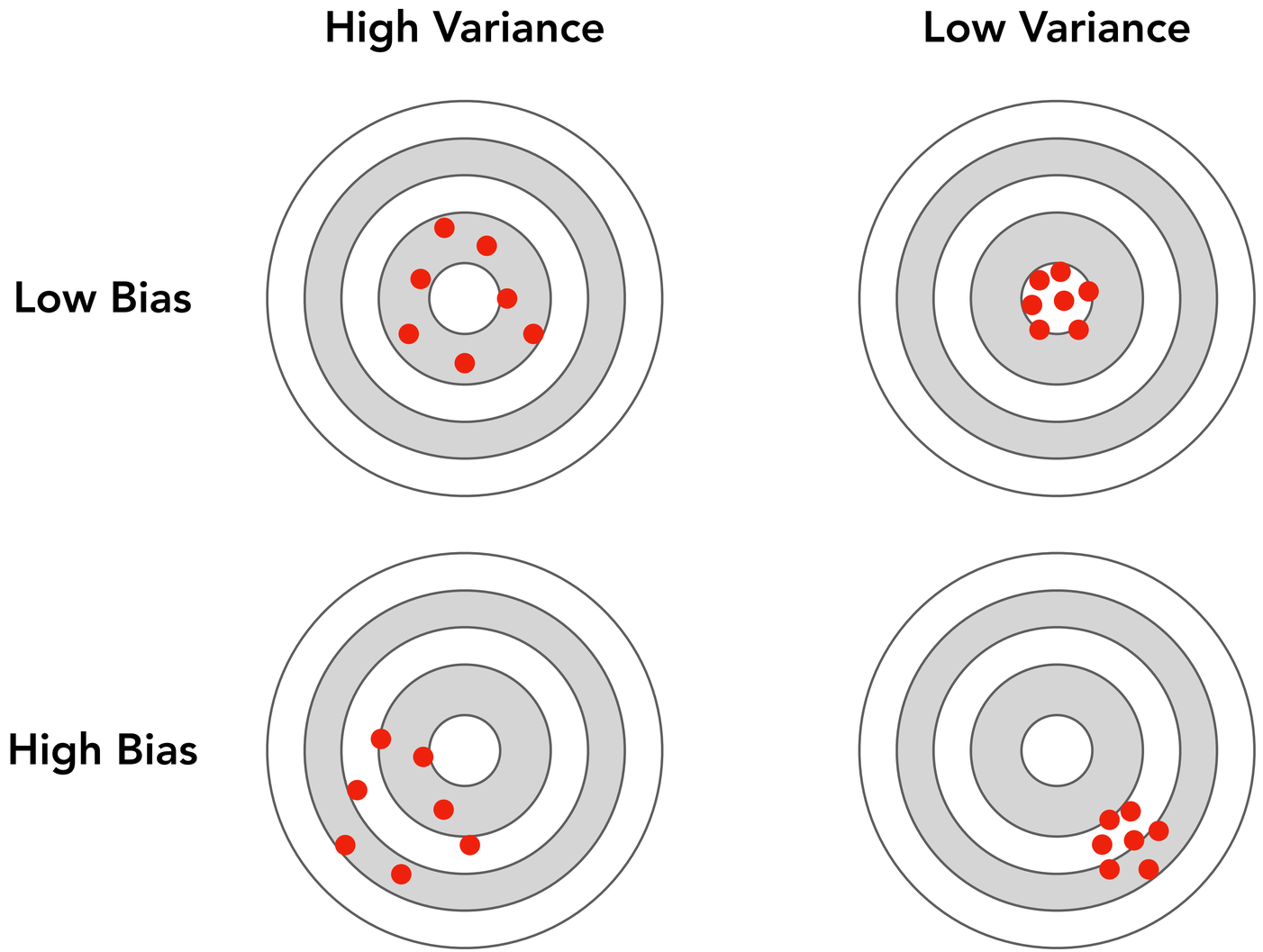




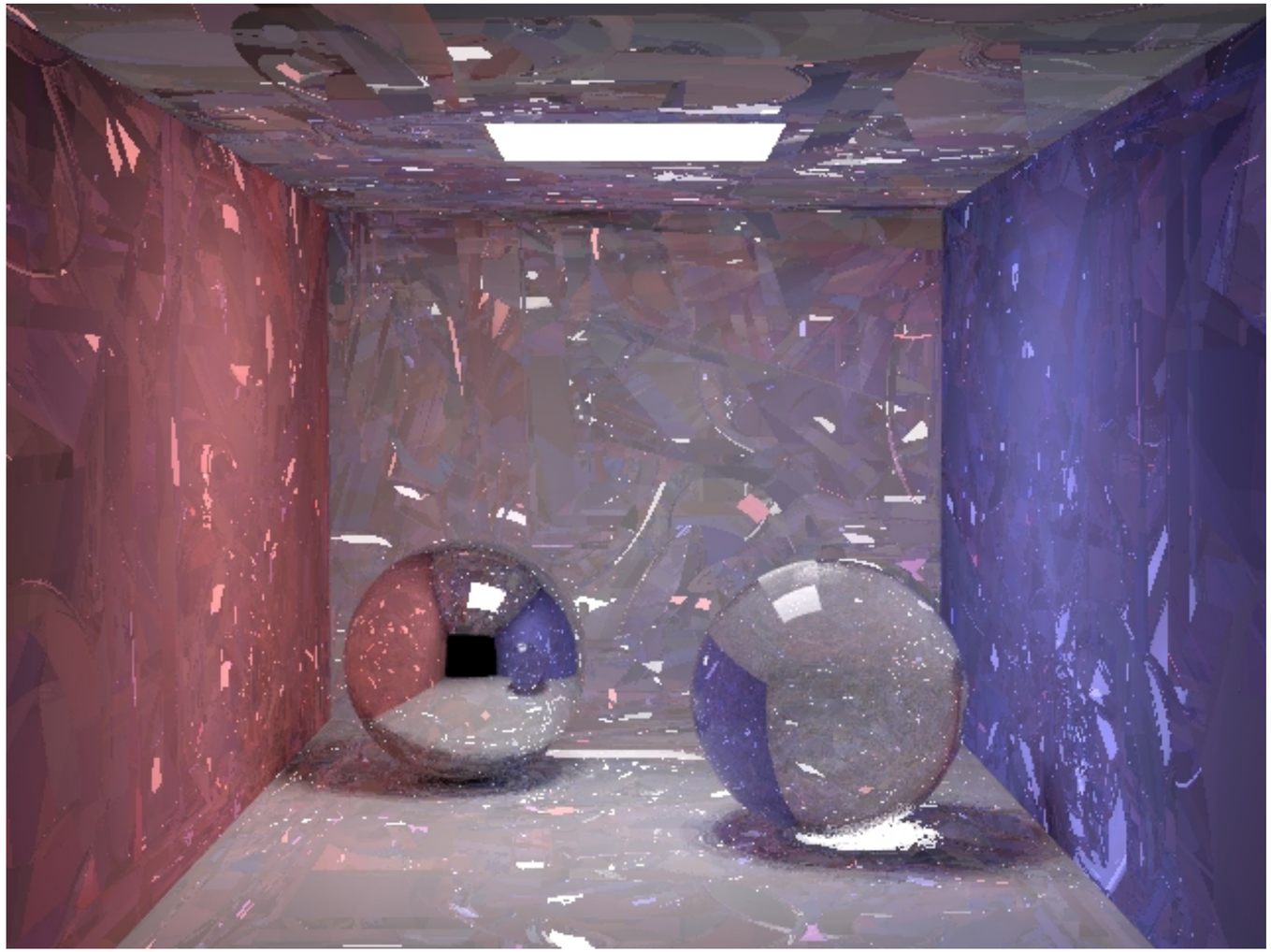
Neither has systematic bias, but F_4 has larger variance than F_{2048}; larger variance leads to more noise.

Calculating one pixel value requires one application of the estimator (one application == "taking N samples and averaging them"). Nearby pixels are supposed to have similar colors but are estimated to be different because of large variance in F_4.

Bias vs. Variance

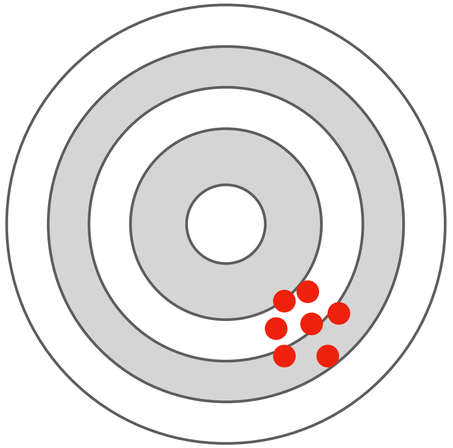


Fixed Sampling (Not Random Enough)

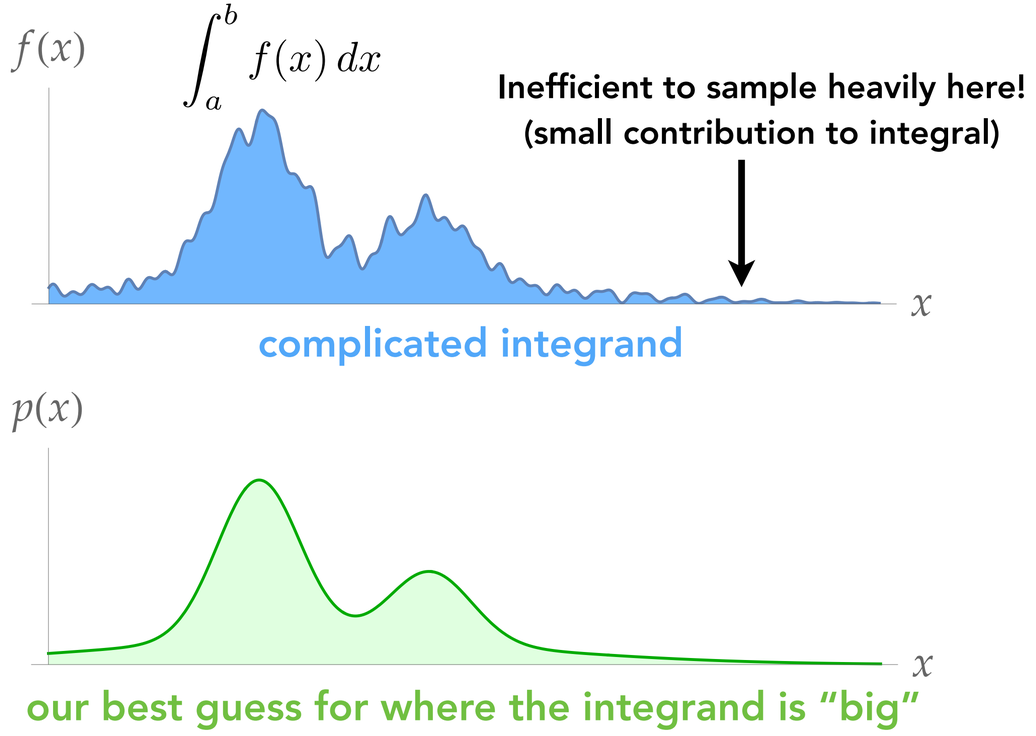


Fixed, rather than random, sampling leads to a biased estimator.

Why? A fixed pattern will systematically miss certain information.



Importance Sampling

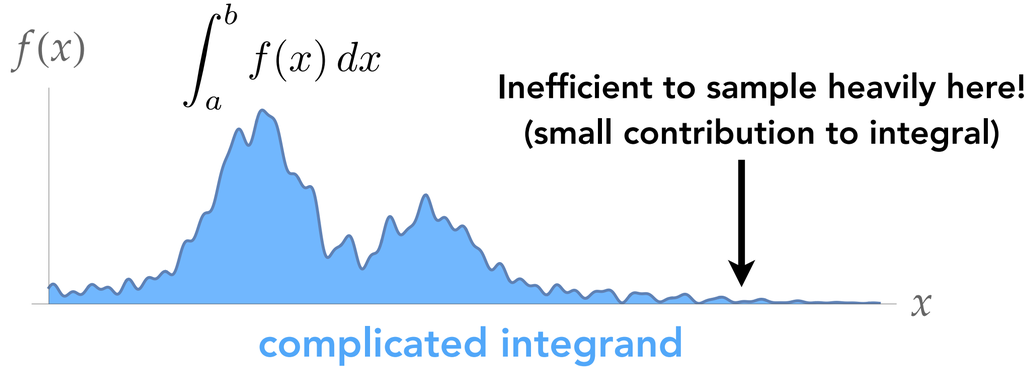


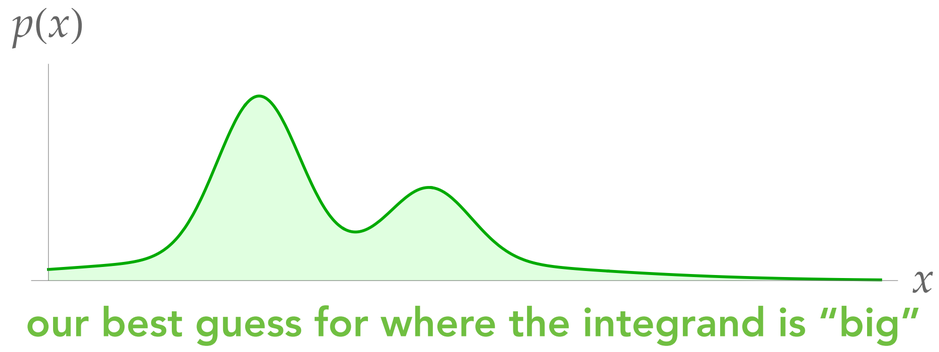
Note: p(x) must be non-zero where f(x) is non-zero

Issue: reducing variance requires taking more samples, but we can only afford to take certain amount of samples at rendering time.

Importance sampling: sample more often where the contribution to the integration is high.

Importance Sampling





Note: p(x) must be non-zero where f(x) is non-zero

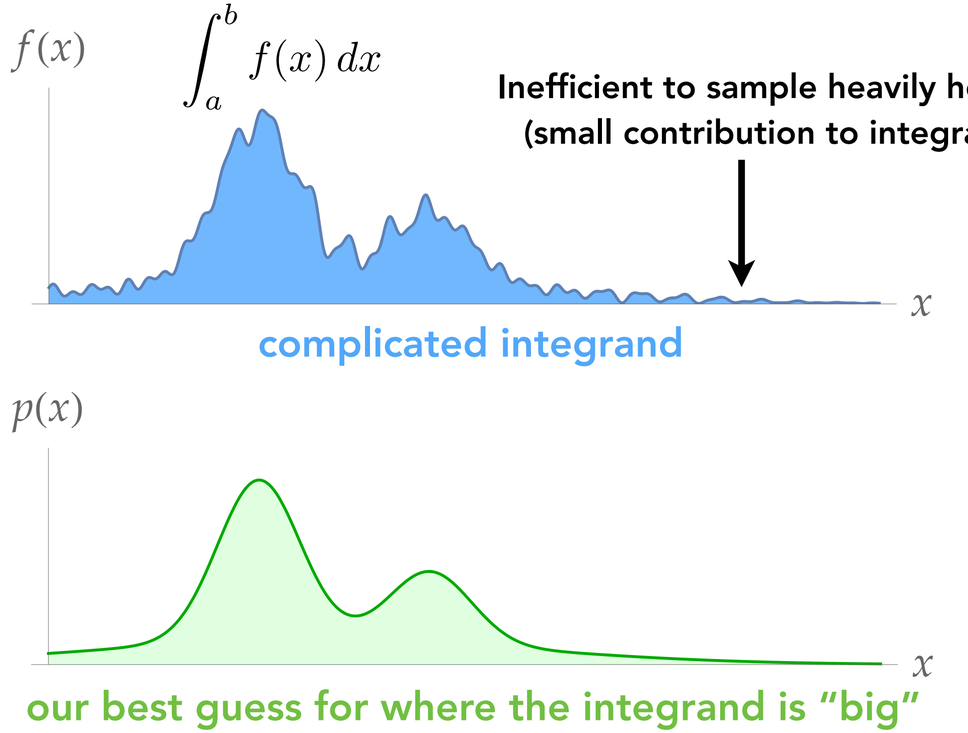
This is a biased estimator given a non-uniform sampling PDF; it would over-estimate.

$$F_N = \frac{1}{N}\sum_{i=1}^{N} f(X_i)(b-a)$$

The correct (unbiased) Monte Carlo estimator counts more frequently samples less.

$$F_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

What PDF to Use for Importance Sample?



Note: p(x) must be non-zero where f(x) is non-zero

$$\int_\Omega f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i$$

The perfect PDF has the same shape as the integrand, which we don’t know.

- Otherwise no need to sample!

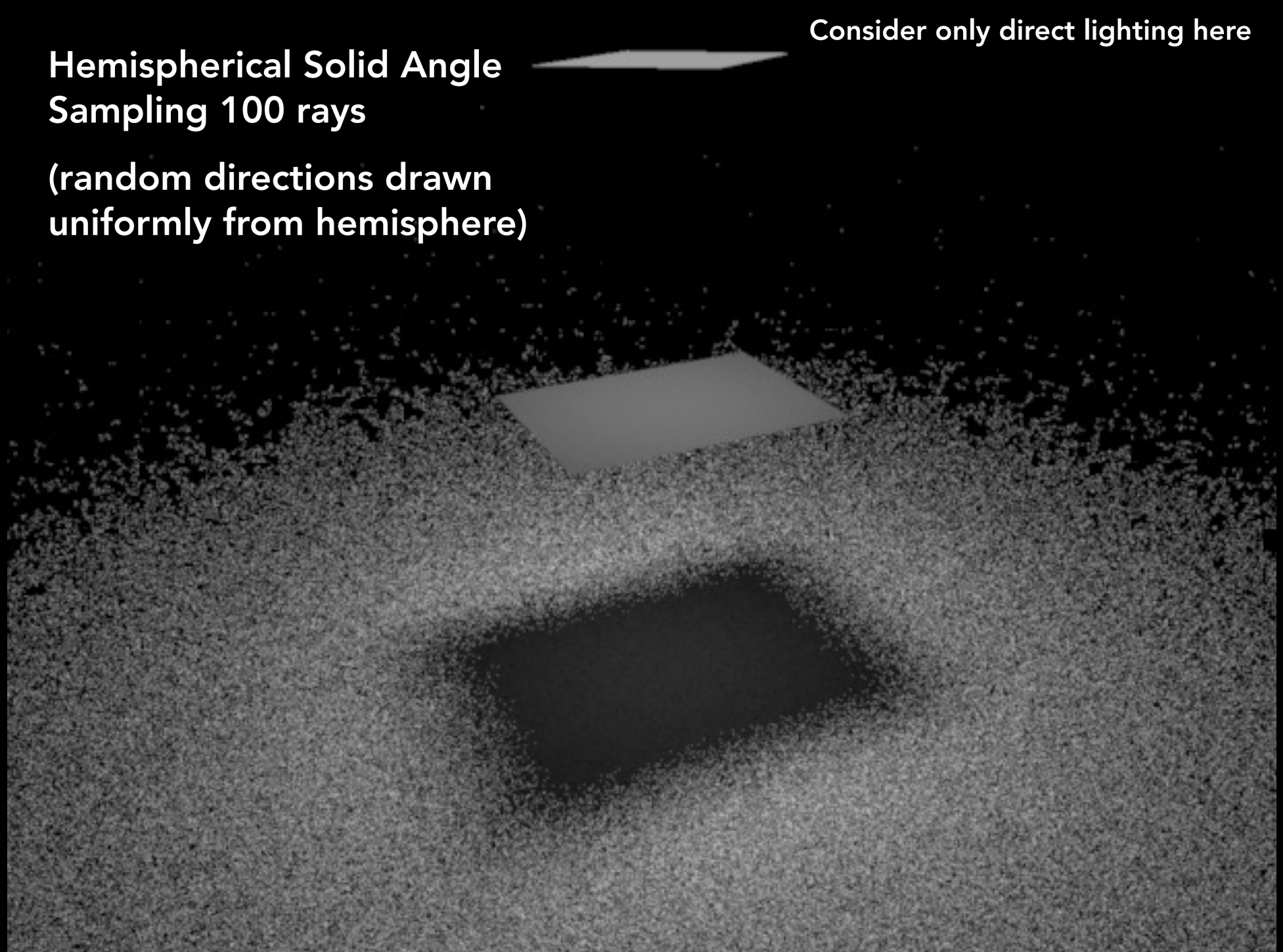
Heuristics that help us determine the general shape of the integrand:

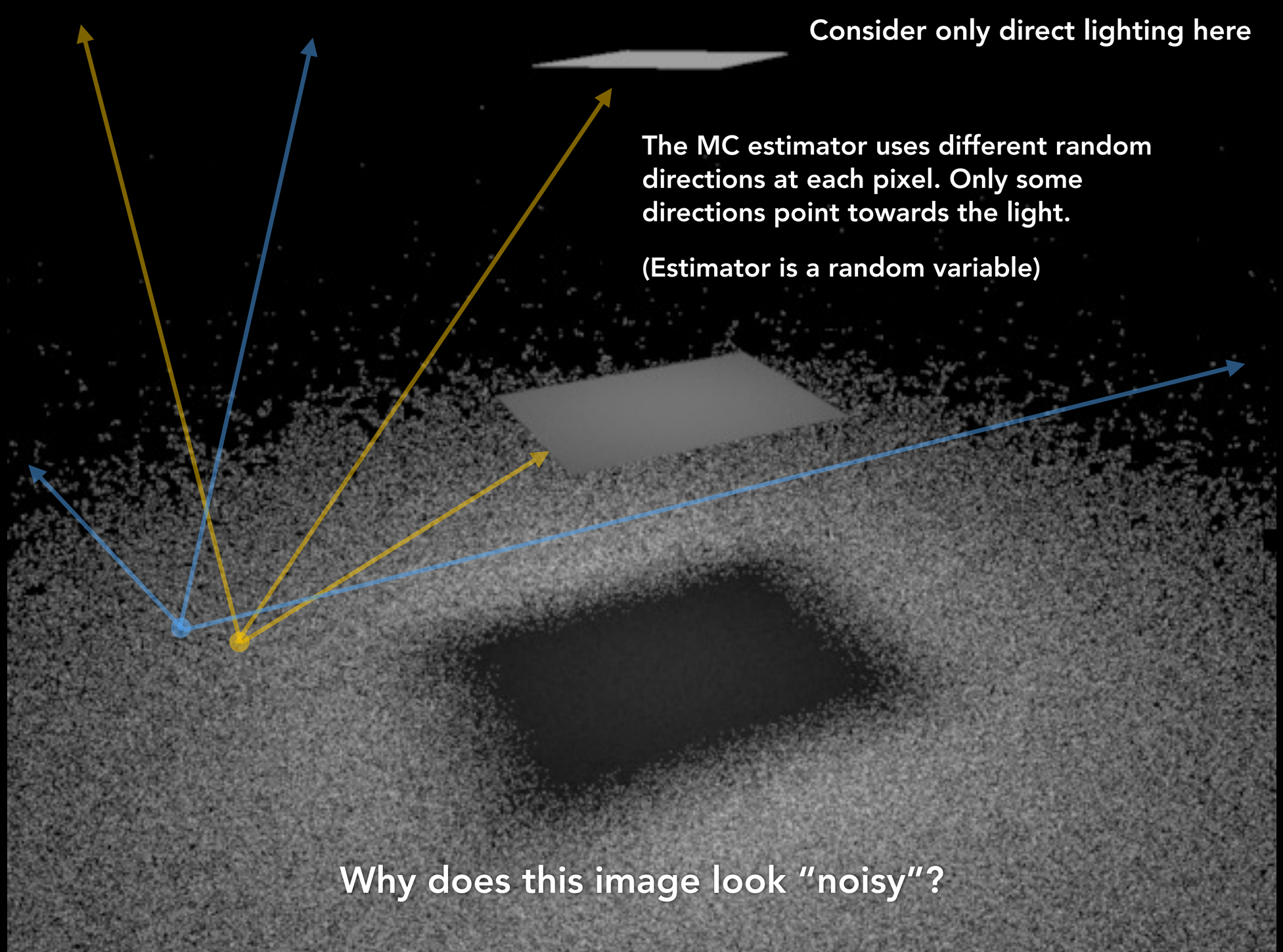
- Light sources
- BRDF
- Train a deep learning model to predict?

Consider only direct lighting here

Hemispherical Solid Angle
Sampling 100 rays

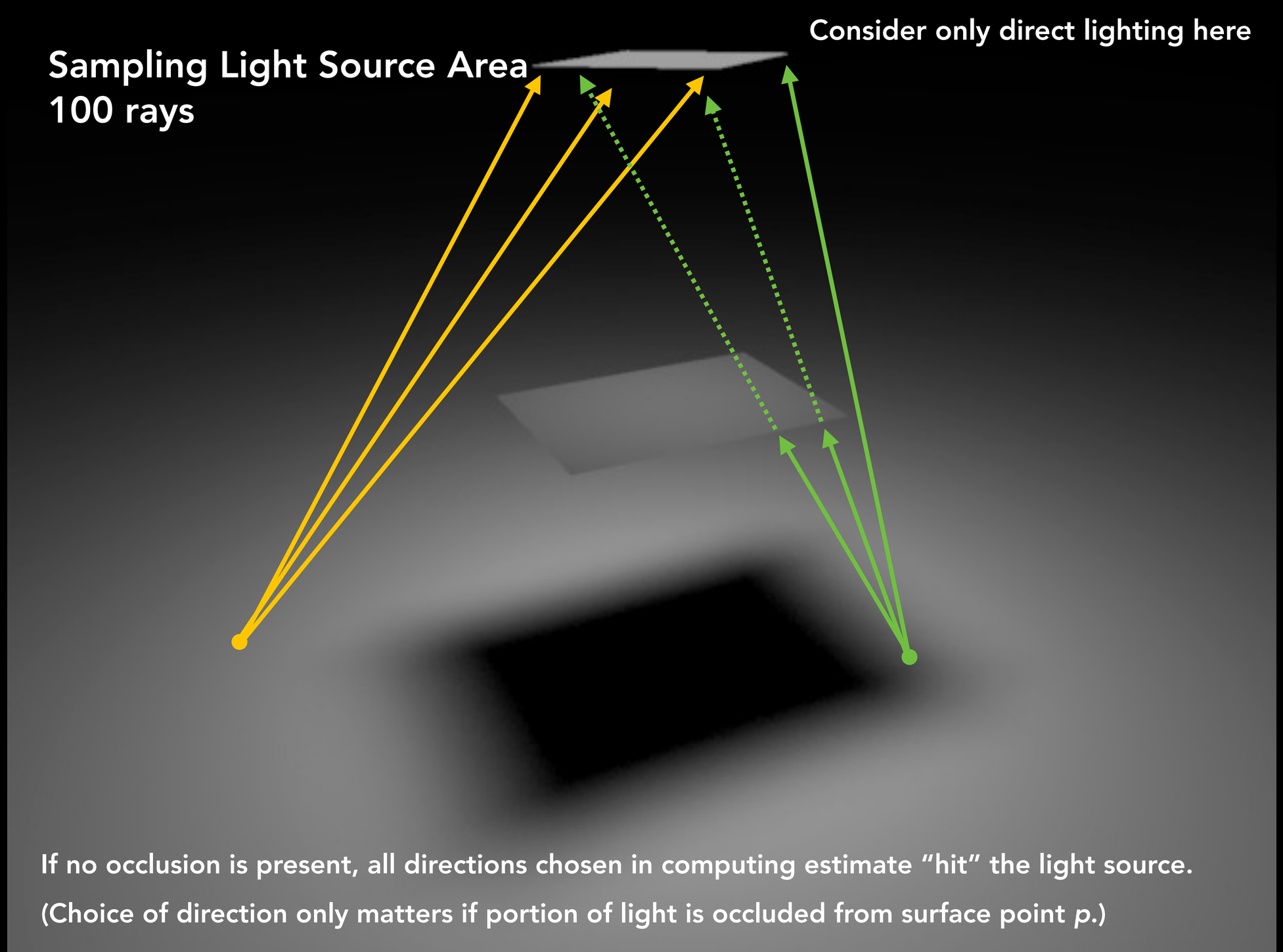
(random directions drawn
uniformly from hemisphere)




Consider only direct lighting here
The MC estimator uses different random directions at each pixel. Only some directions point towards the light.
(Estimator is a random variable)
Why does this image look “noisy”?

Consider only direct lighting here

Sampling Light Source Area
100 rays

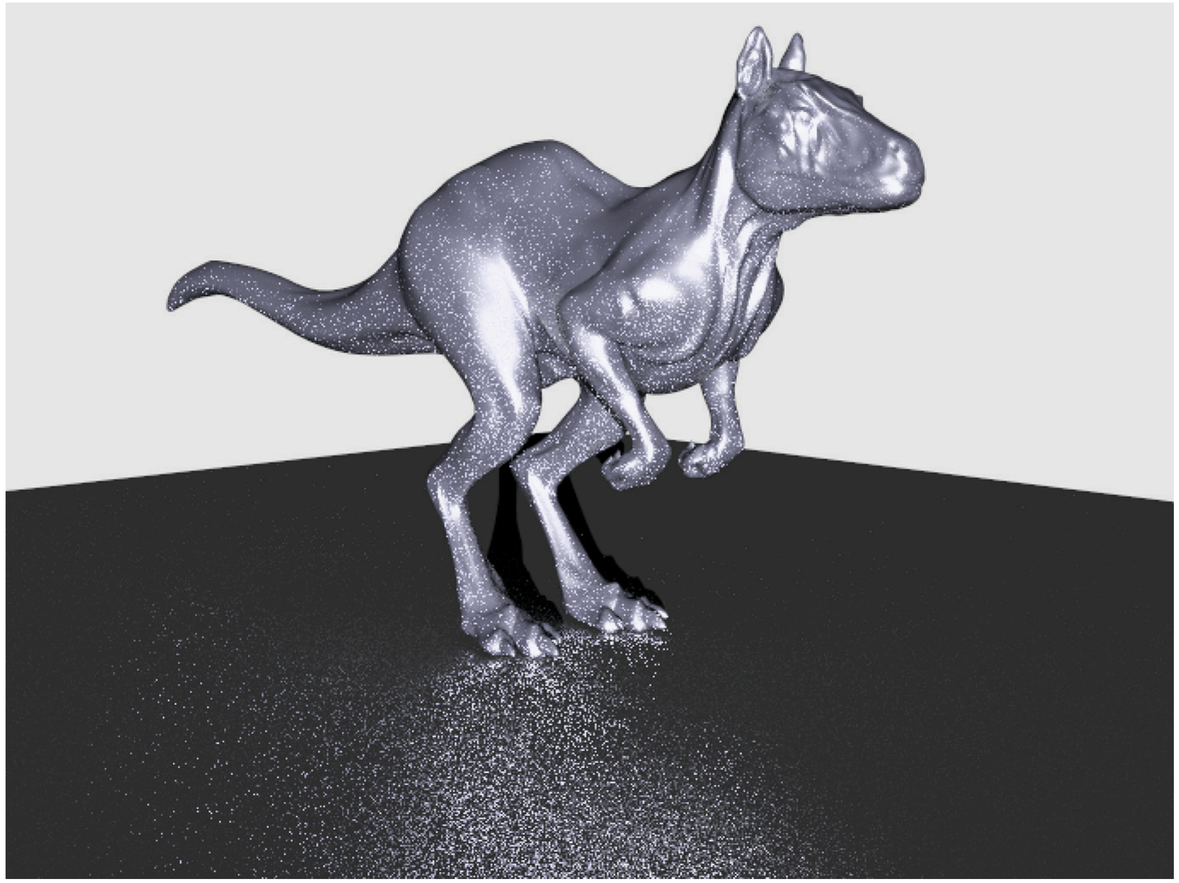


If no occlusion is present, all directions chosen in computing estimate “hit” the light source.

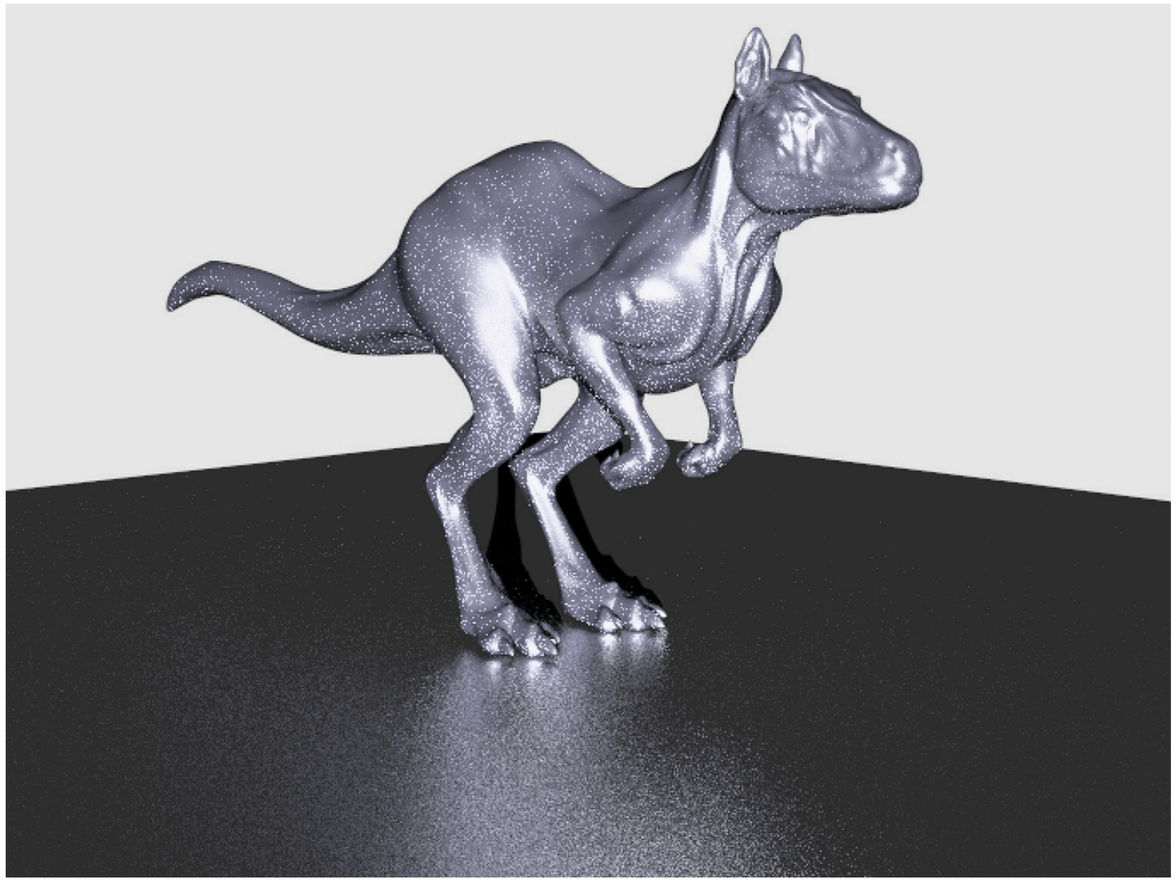
(Choice of direction only matters if portion of light is occluded from surface point *p*.)

Importance Sampling Examples

Uniform PDF



PDF proportional to BRDF

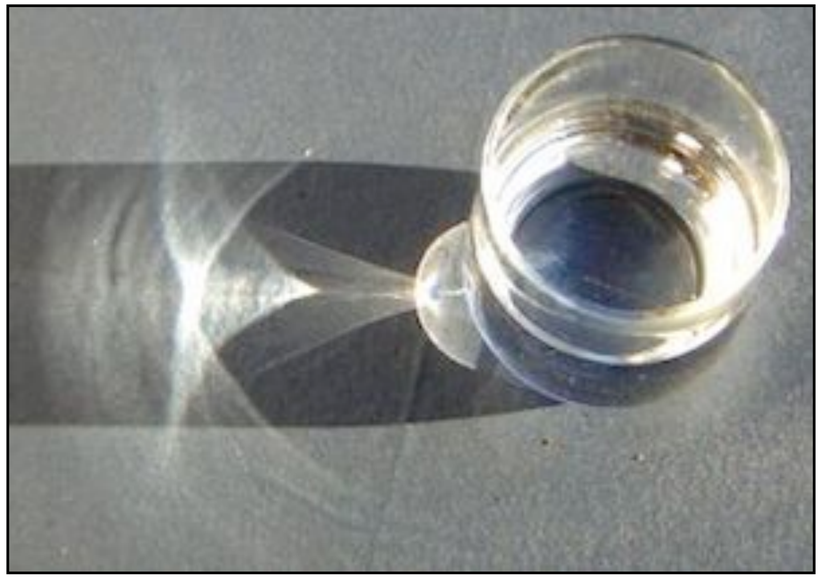


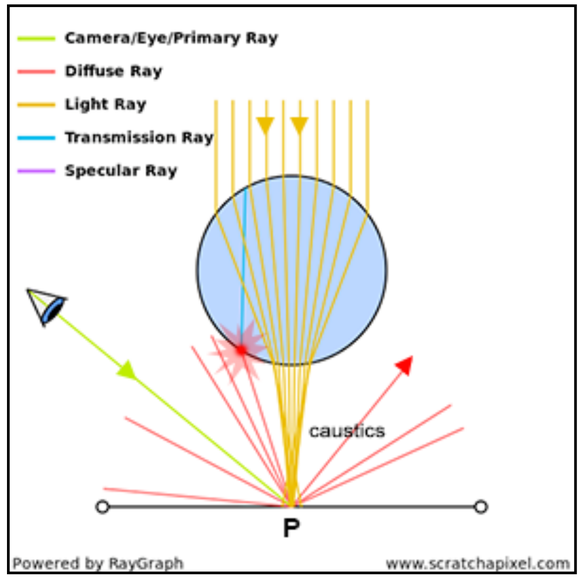
https://pbr-book.org/3ed-2018/Monte_Carlo_Integration/Importance_Sampling

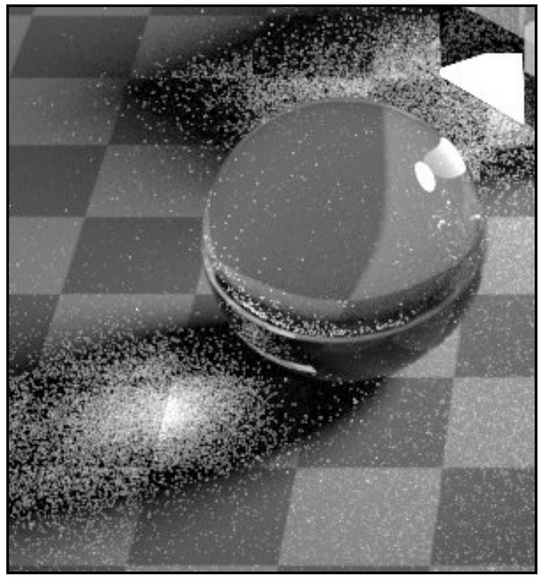
Nasty Case: Caustics

Caustics: light bounces off a surface in specular manner and then hits a diffuse surface as it illuminates it

- Importance sampling based on surface BRDF will lead to uniform sampling, but incident lights actually come from only a small region

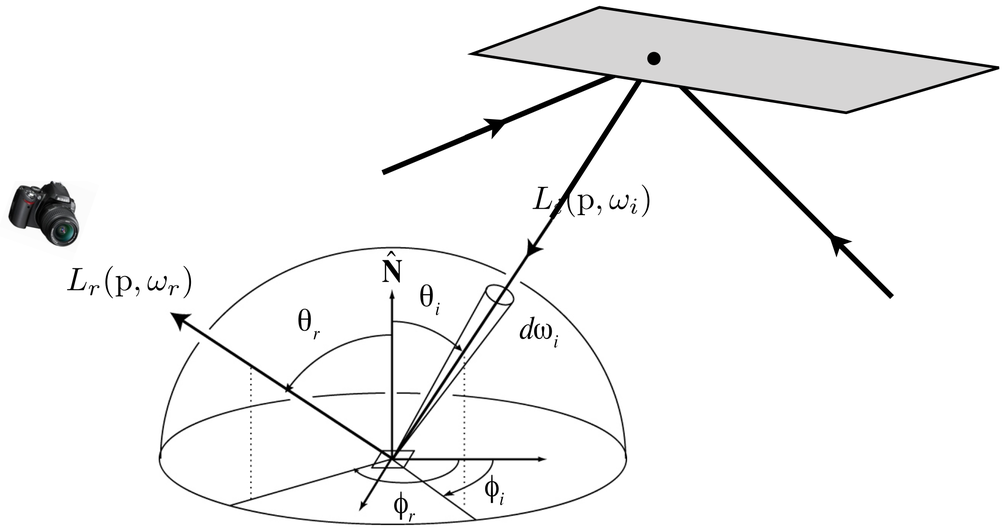






https://www.chaosgroup.com/blog/what-are-caustics-and-how-to-render-them-the-right-way

Still, It Requires Recursion

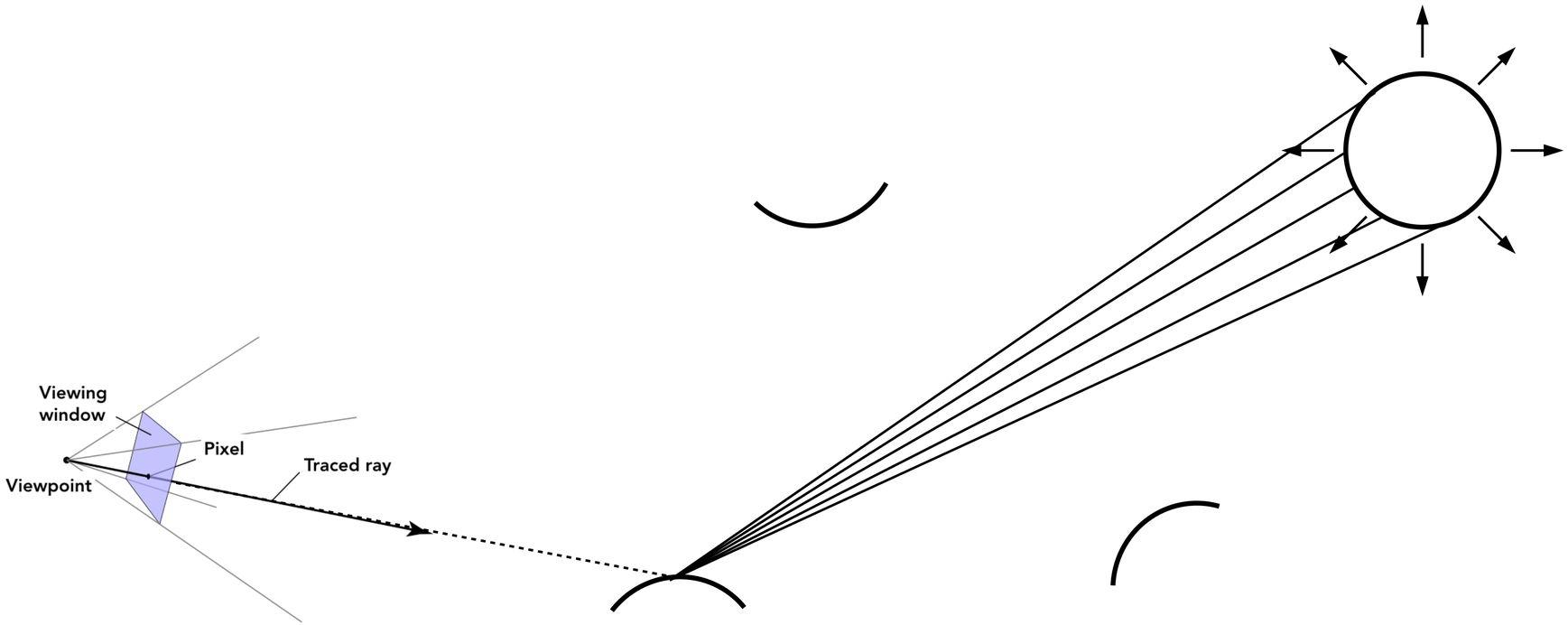


$$L_r(p, \omega_r) = \frac{1}{N} \sum_{1}^{N} \frac{f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i}{p(\omega_j)}$$

The moment we sample more than one ray at each point, the total number of rays grows exponentially, which is clearly intractable.

But it's necessary for global illumination (to account for indirect illumination).

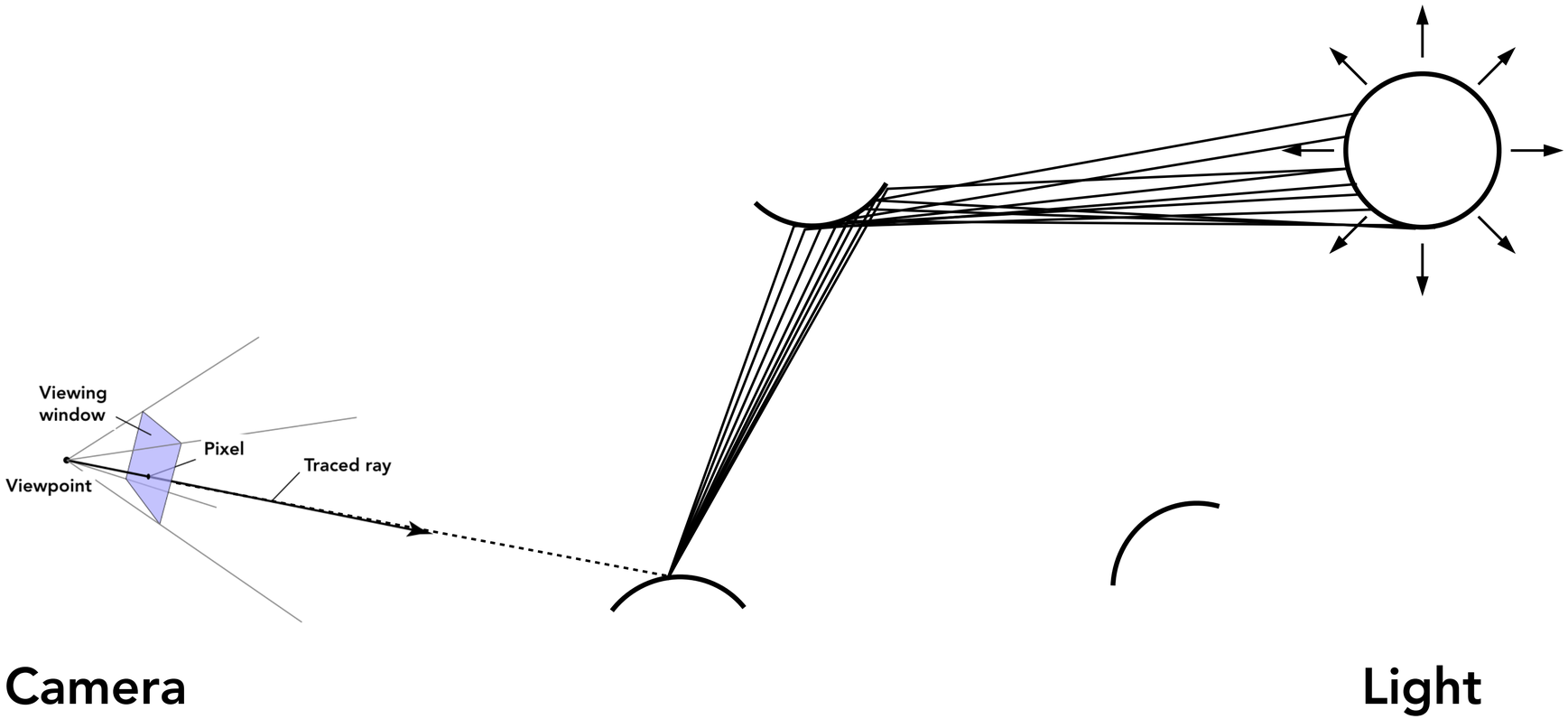
1-Bounce Paths Connecting Ray to Light



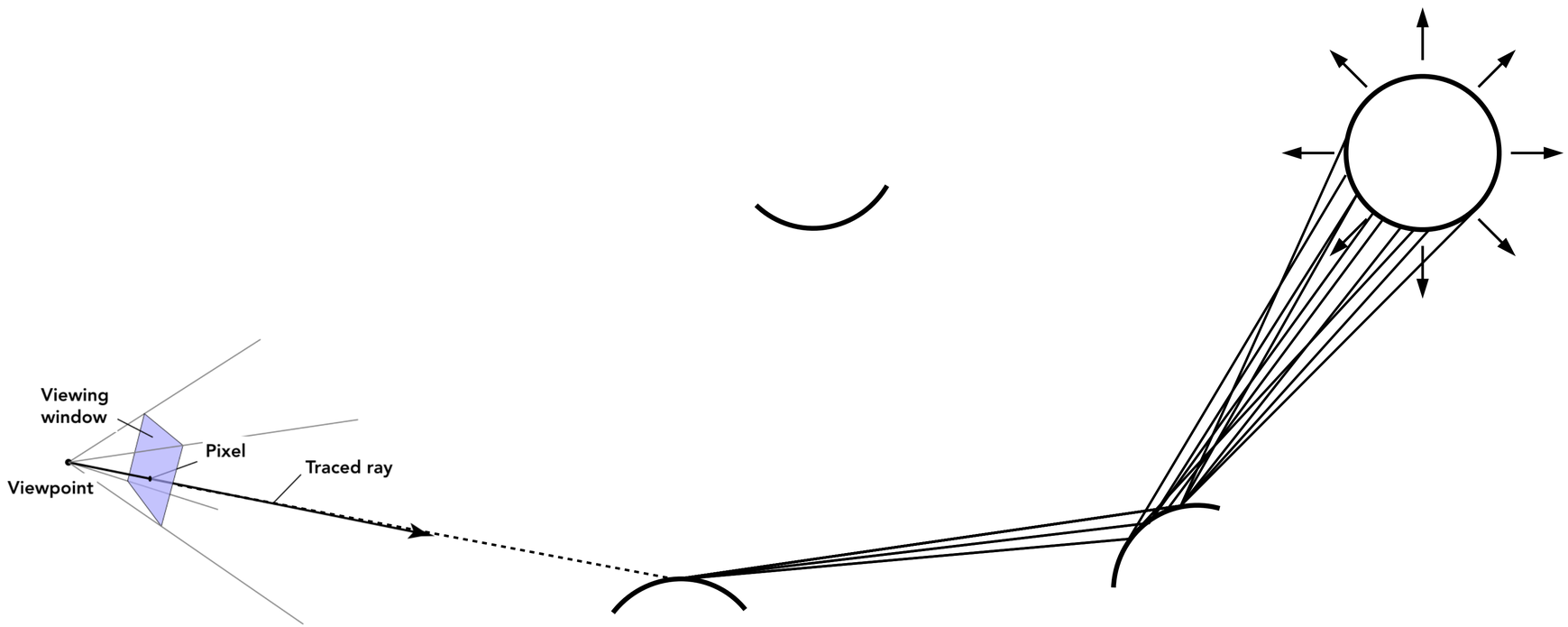
Camera

Light

2-Bounce Paths Connecting Ray to Light



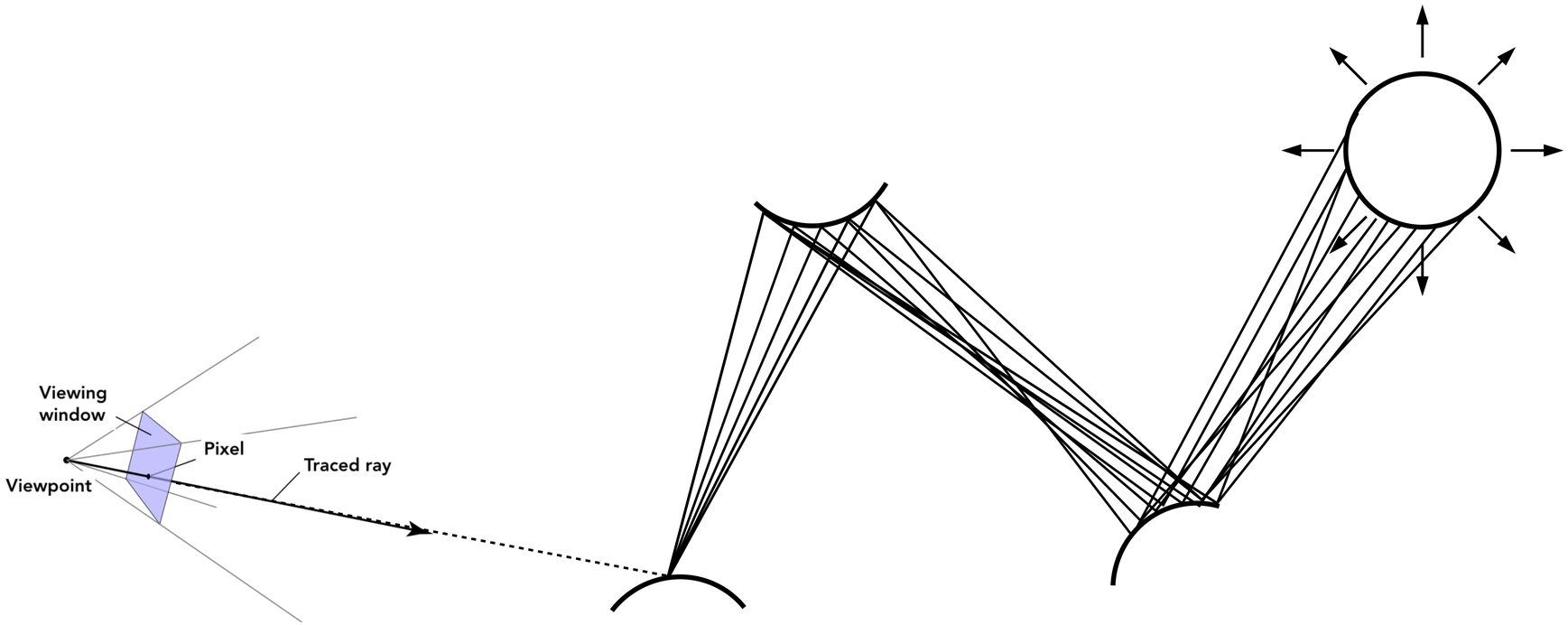
2-Bounce Paths Connecting Ray to Light



Camera

Light

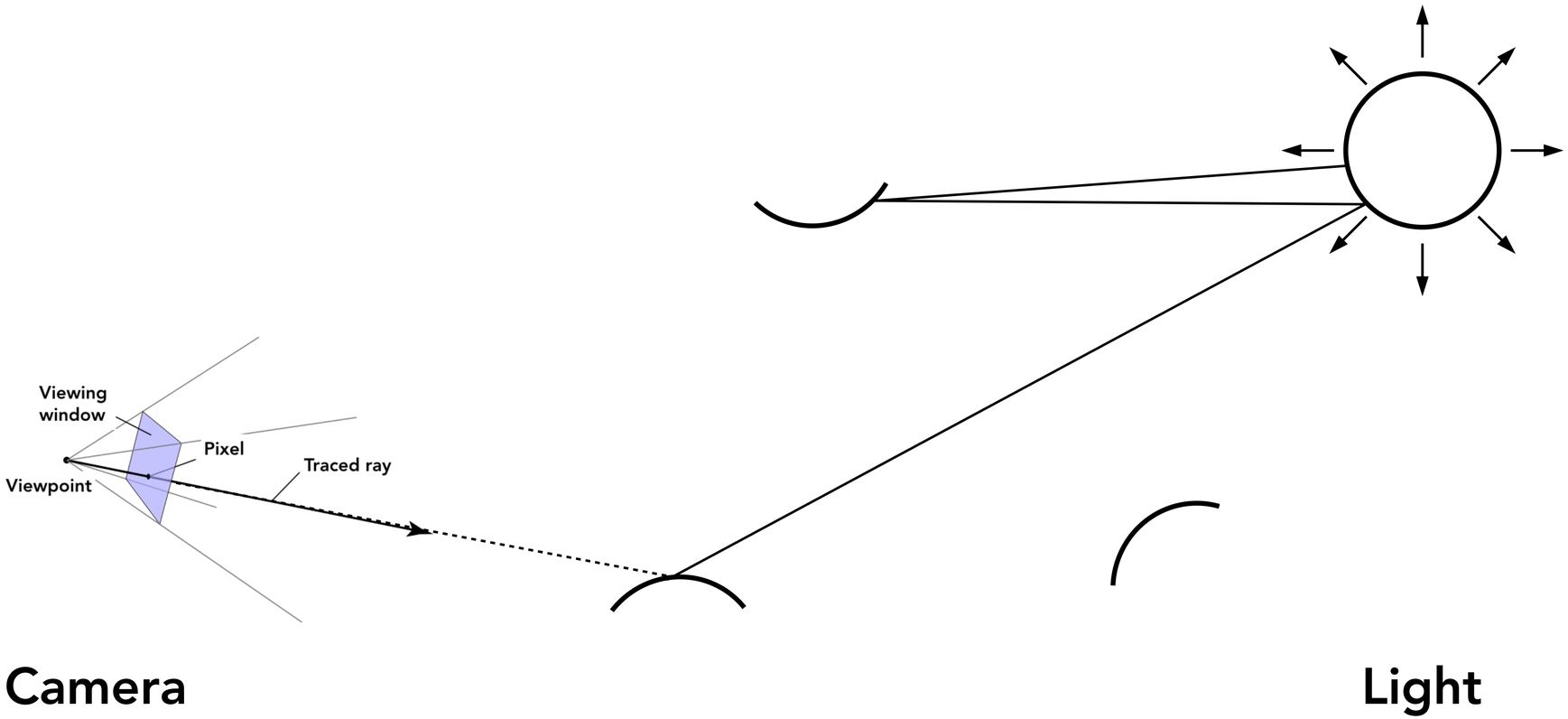
3-Bounce Paths Connecting Ray to Light



Camera

Light

3-Bounce Path Connecting Ray to Light



Integrating Over All Paths (Uncountably Infinite Set)

Energy of camera ray = energy of all one-bounce paths +

energy of all two-bounce paths +

energy of all three-bounce paths +

energy of all four-bounce paths +

……

Infinitely many groups

Infinitely many rays in each group

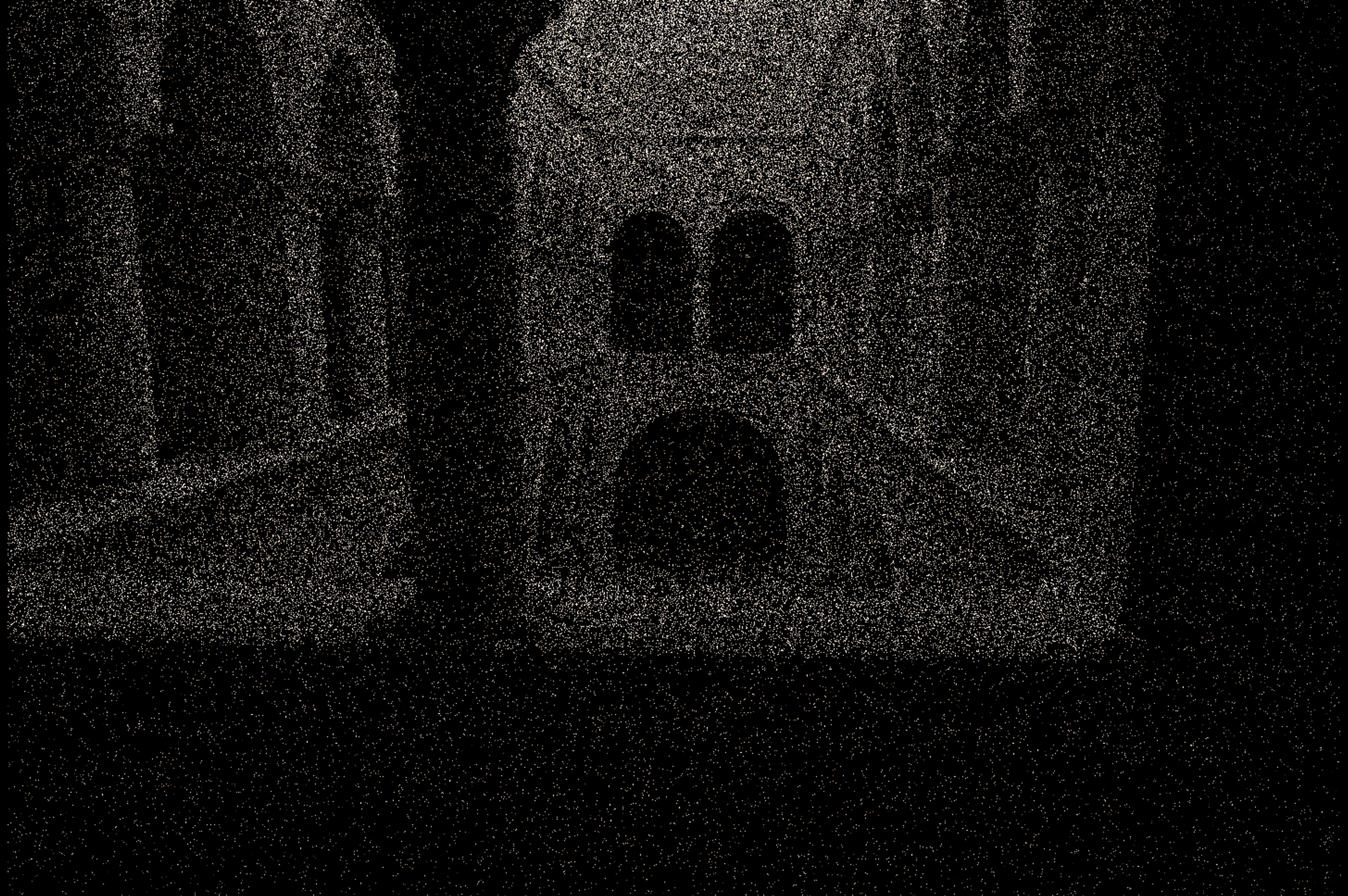
Path Tracing

Probabilistic termination of recursion.

- Need to change the estimator slightly to get it to be unbiased (not discussed here).

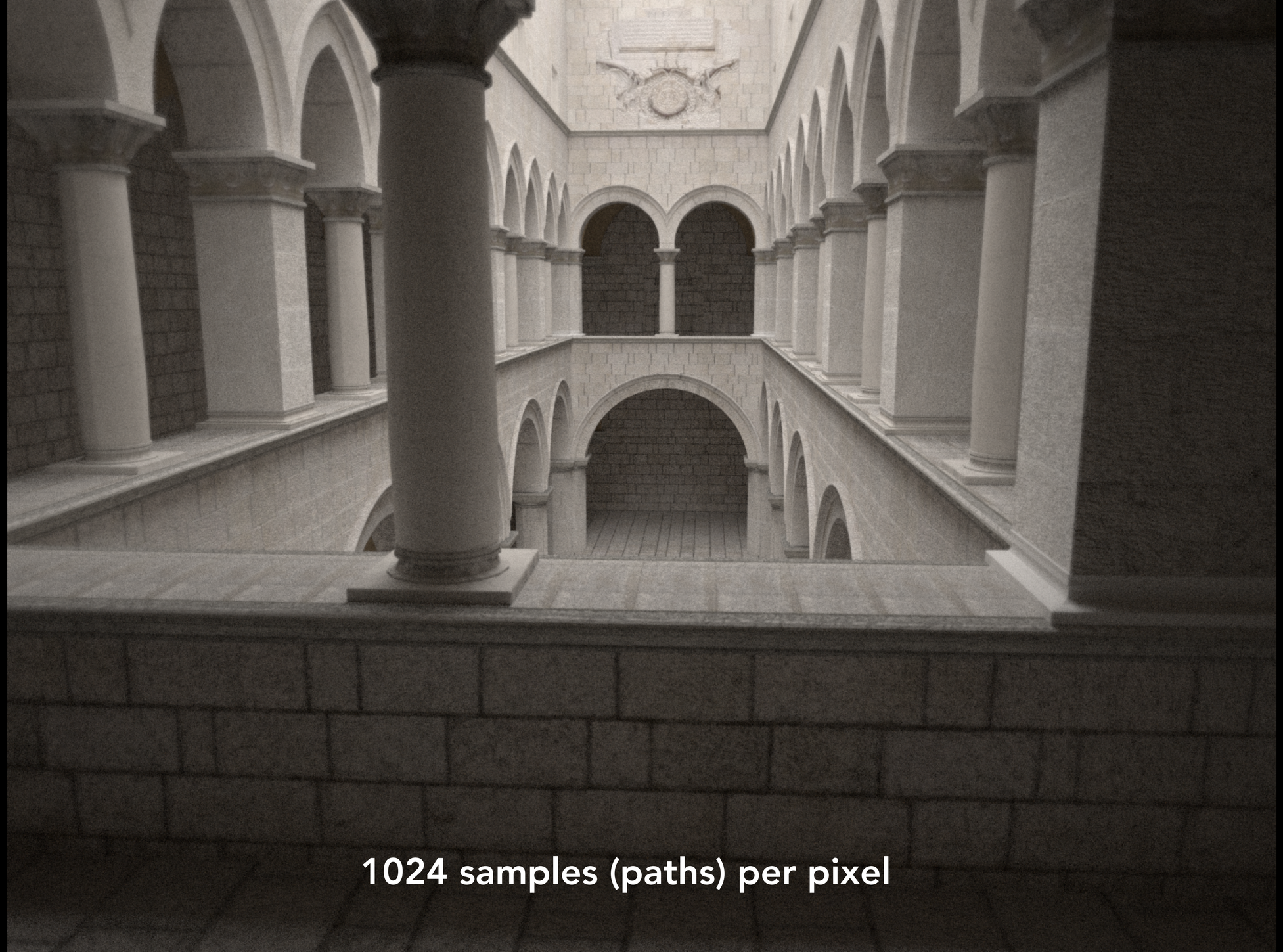
At each point, randomly cast two rays:

- One toward light source, not recursive, can importance sample the light source area
- One toward other area, recursive (with probabilistic termination), can importance sample based on BRDF
- Add the two up



One sample (path) per pixel

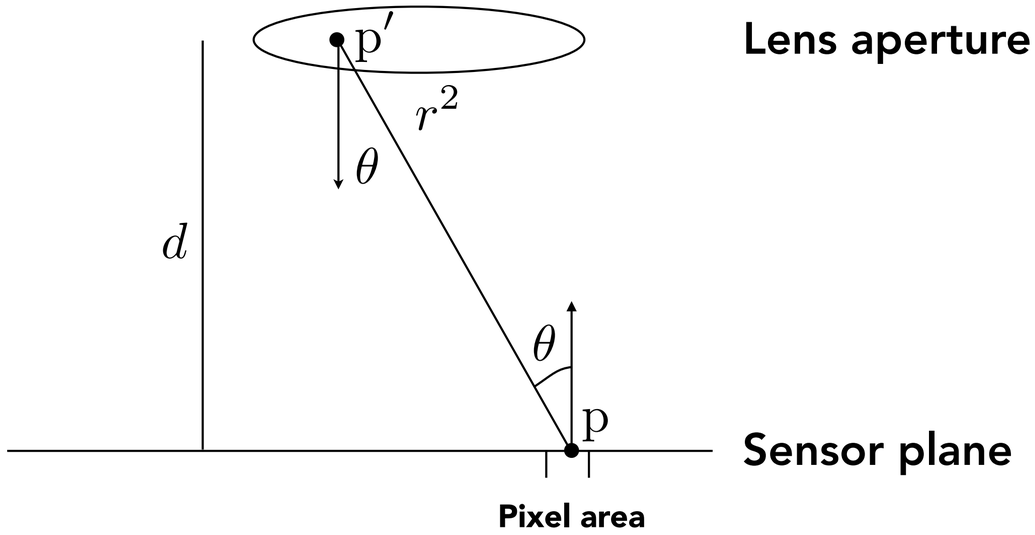
32 samples (paths) per pixel



1024 samples (paths) per pixel

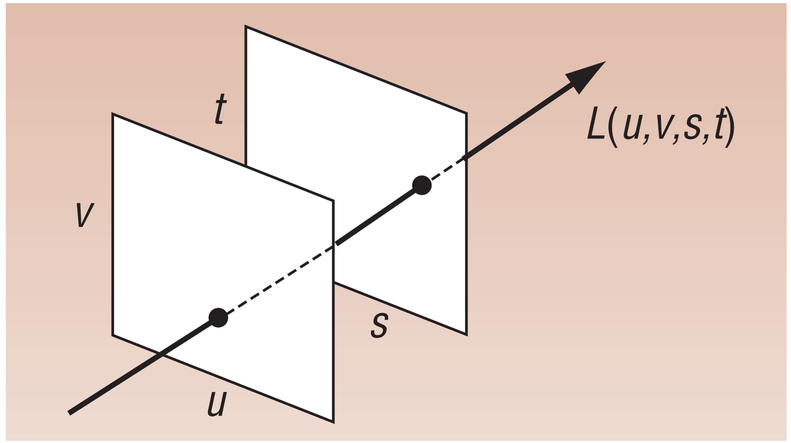
Integration inside Camera

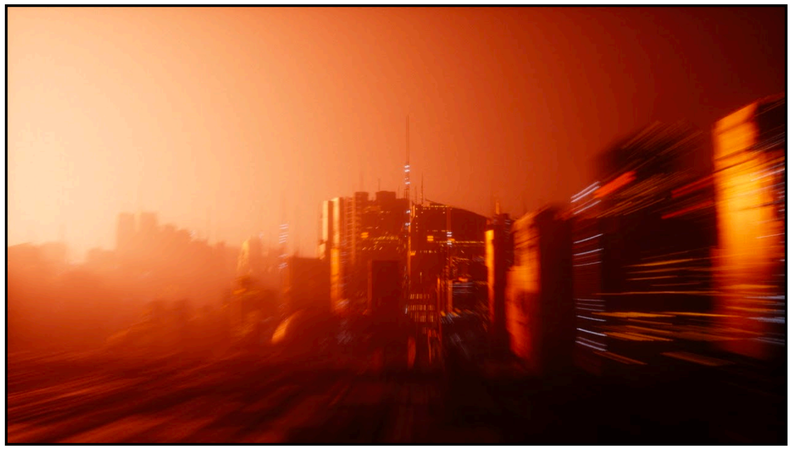
5D Integral: Real Camera Pixel Exposure



$$Q_{\text{pixel}} = \frac{1}{d^2} \int_{t_0}^{t^1} \int_{A_{\text{lens}}} \int_{A_{\text{pixel}}} L(\mathrm{p}' \to \mathrm{p}, t) \cos^4 \theta \, \mathrm{d}p \, \mathrm{d}p' \, \mathrm{d}t$$

4D for the light field (2D for pixel + 2D for aperture) **+ 1D for exposure time**

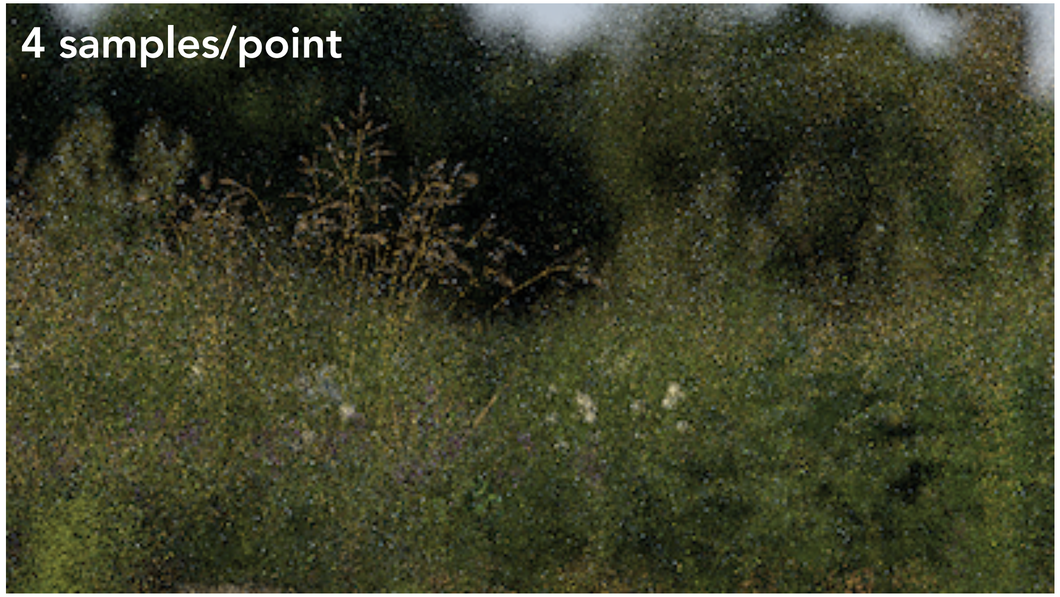




Sampling inside Camera

That’s why when generating rays we need to sample both pixel and lens.

- More samples reduce noise (lower variance).

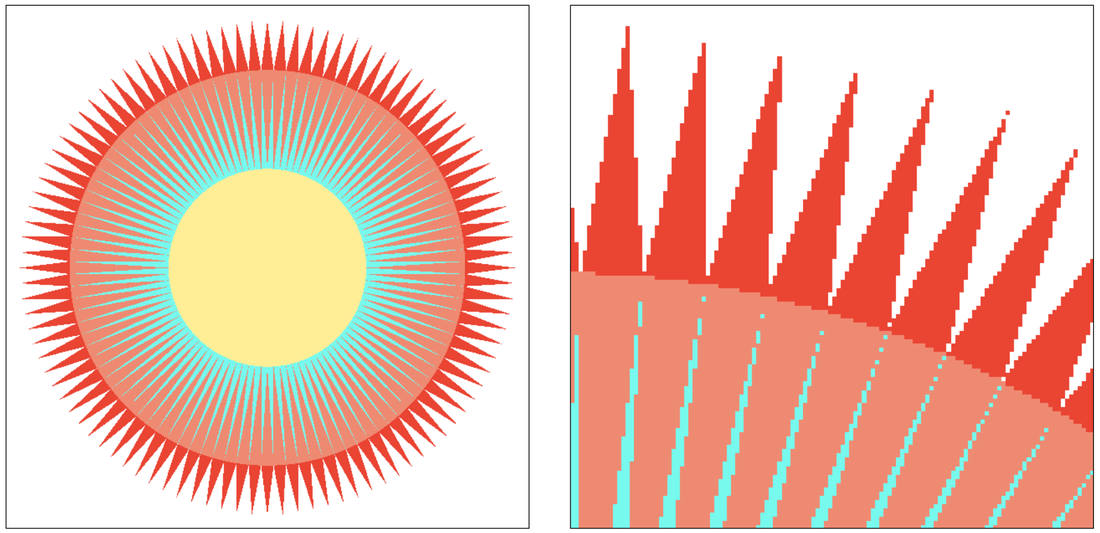


http://www.pbr-book.org/3ed-2018/Camera_Models/Projective_Camera_Models.html

Alias vs. Noise

Both are because of low sampling rate, but effects are different.

Aliasing: high-frequency signals (e.g., edge) **masquerading as low-frequency signals** (e.g., patterned jaggies)

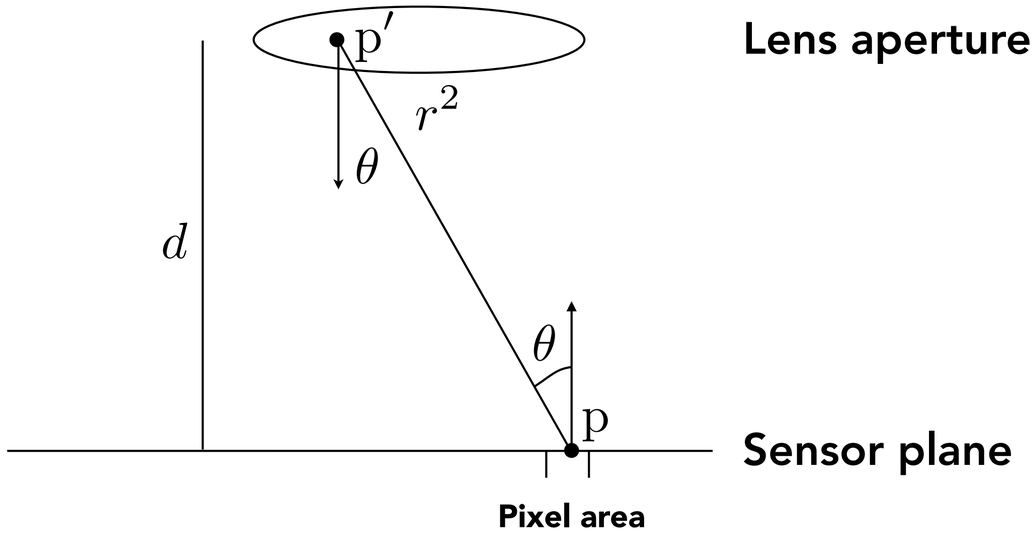


Noise: low-frequency signals (e.g., similar adjacent pixels) **showing up as high-frequency signals** (e.g., very different adjacent pixels)

http://www.pbr-book.org/3ed-2018/Camera_Models/Projective_Camera_Models.html

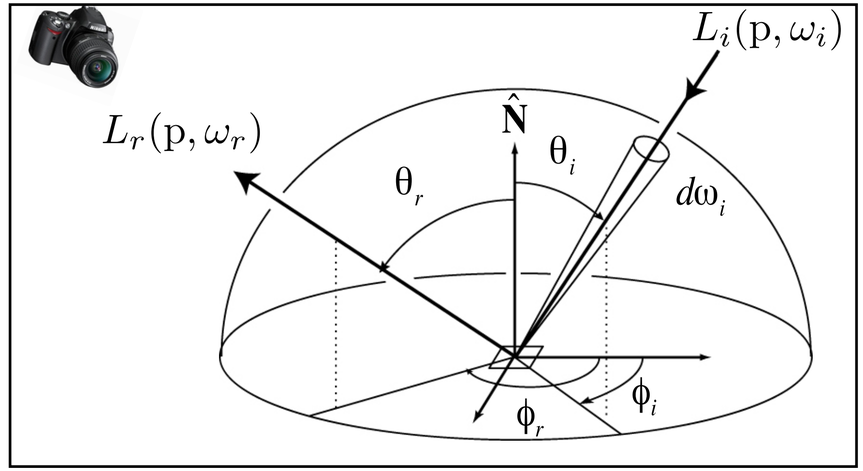
Integration in Rendering

5D Integral: Real Camera Pixel Exposure



$$Q_{\text{pixel}} = \frac{1}{d^2} \int_{t_0}^{t^1} \int_{A_{\text{lens}}} \int_{A_{\text{pixel}}} L(\text{p}' \to \text{p}, t) \cos^4 \theta \, \text{d}p \, \text{d}p' \, \text{d}t$$

Once a ray is in the scene and hits a point: 2D integration over hemisphere for each point

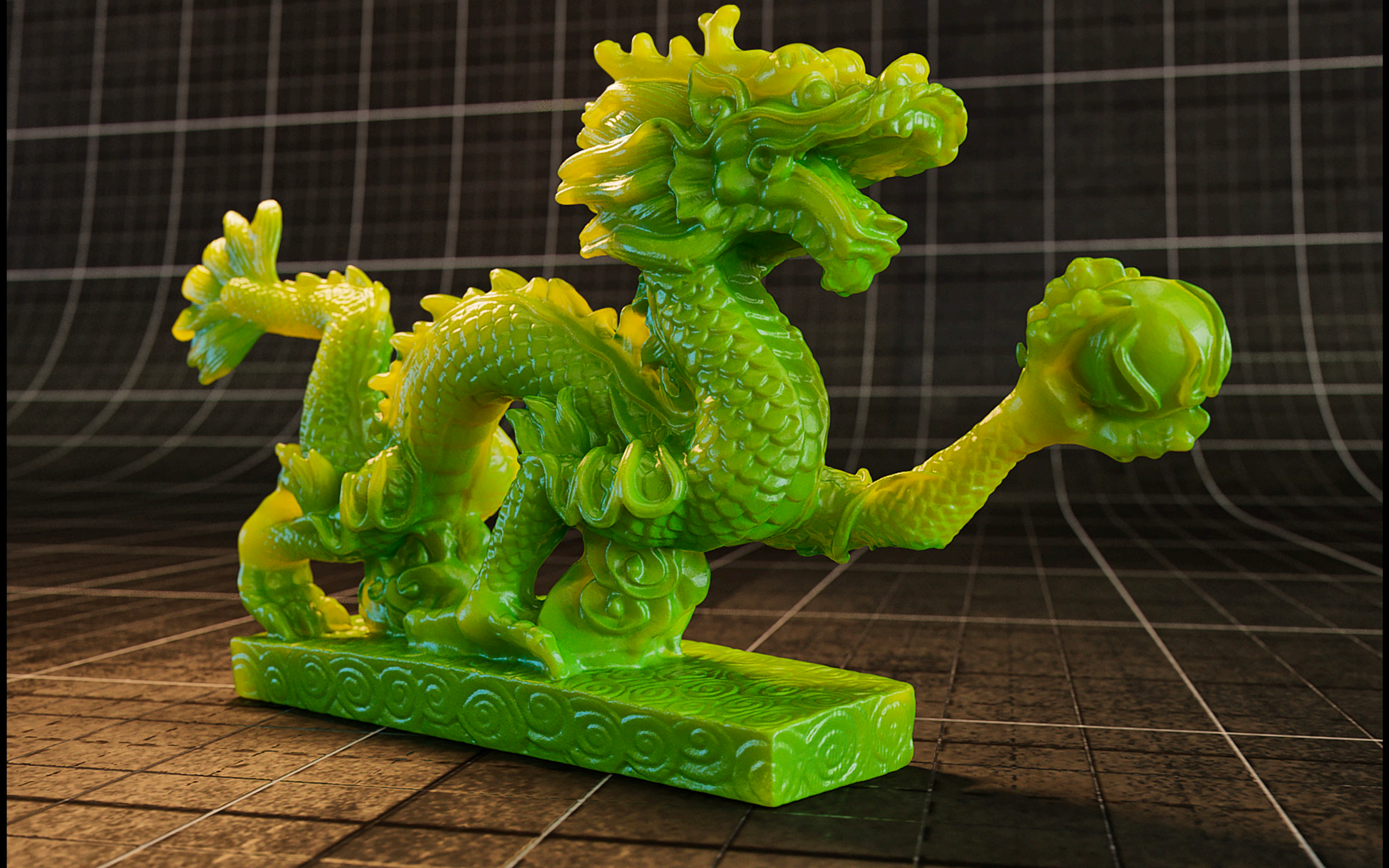


Volume Scattering

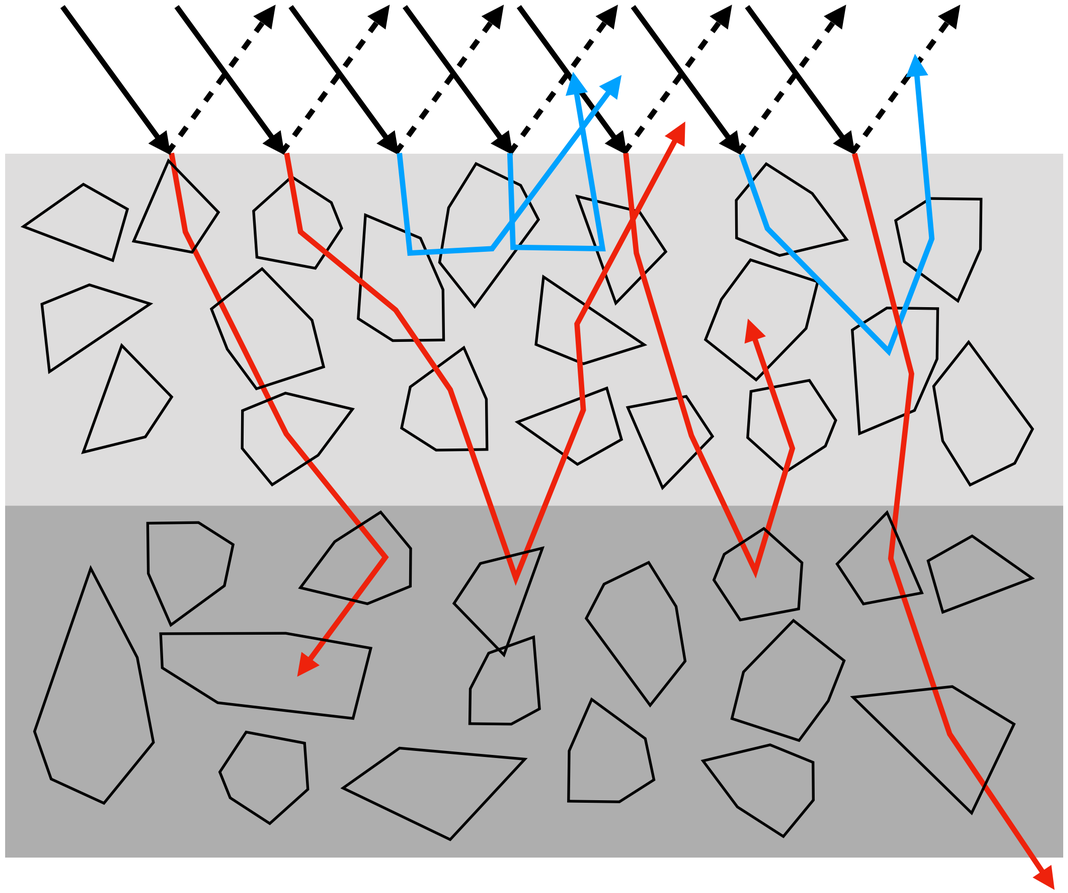
Volume Scattering

Particles scatter and absorb photons, and it's important to model in:

- Translucent materials (e.g., jade, skin), which have strong subsurface scattering (SSS) in addition to surface scattering. SSS can be ignored in cases where refraction is weak or particle absorption is strong so that the back-scattering doesn't contribute much to the overall material appearance.
- "Participating media" (e.g., cloud, smoke), which have no surface and everything in volume scattering.



Translucency == Subsurface Scattering



So far we consider only surface reflection, but not how photons interact with particles under surface.

Particles of translucent materials under the material surface scatter lights like crazy and some scattered photons make their way out of the surface.

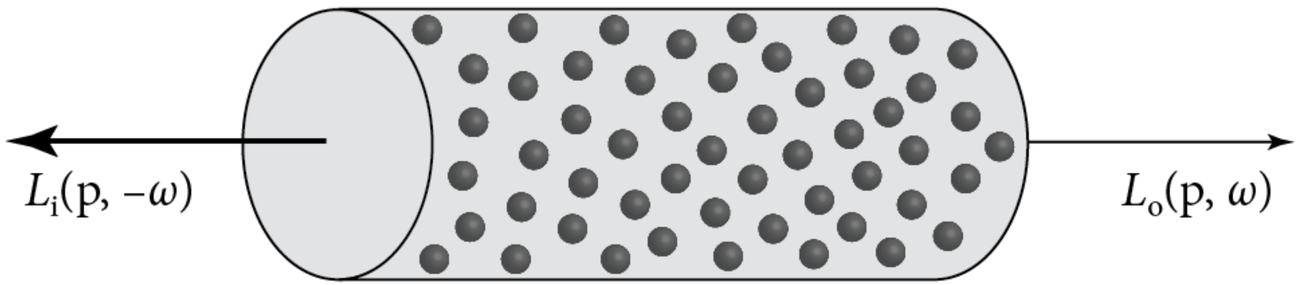
General Idea

The principle way to model volume scattering is the Radiative Transfer theory, which is an integro-differential equation that generally have no analytical solution.

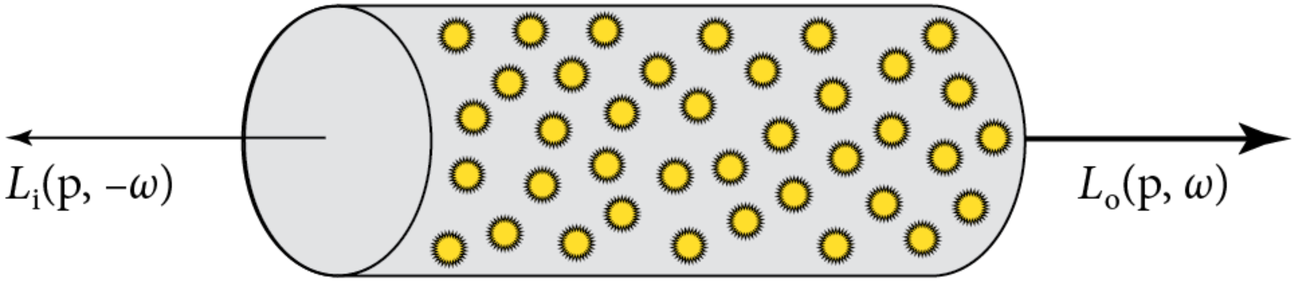
We solve it by:

- make simplified assumptions (e.g., Kubelka-Munk model for diffuse illumination and isotropic scattering, diffusion approximation applied to BSSDF [Jensen et al. 2000])
- plus: using numerical methods to approximate the solution (e.g, monte-carlo estimator)

Volume Scattering Processes

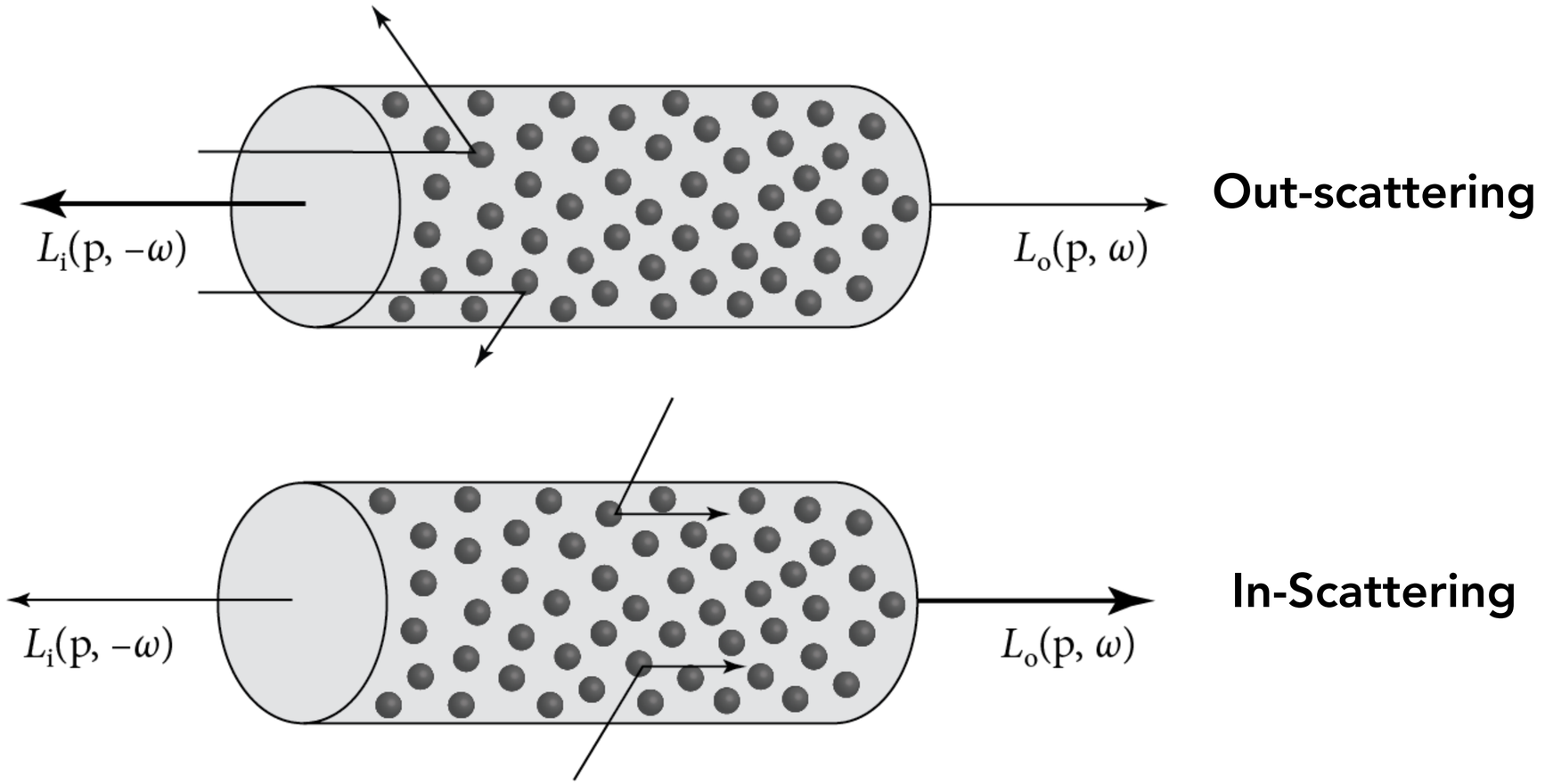


Absorption



Emission

Volume Scattering Processes



https://pbr-book.org/4ed/Volume_Scattering/Volume_Scattering_Processes

Radiative Transfer Equations

Change of radiance at **p** toward **ω** (∇ is directional derivative)

Extinction coefficient = absorption + scattering coefficients

Phase function: radiance distribution at **p** from ω_i to **ω** (the integration over ω_i is normalized to 1)

$$\omega \cdot \nabla L(\mathbf{p}, \omega) = -\sigma_t(\mathbf{p}, \omega)L(\mathbf{p}, \omega) + \sigma_a(\mathbf{p}, \omega)L_e(\mathbf{p}, \omega) + \sigma_s(\mathbf{p}, \omega)\int_{4\pi} p(\mathbf{p}, \omega, \omega_i)L(\mathbf{p}, \omega_i)d\omega_i$$

Incident radiance at **p** toward **ω**

Absorption coefficient

Emission

Scattering coefficient

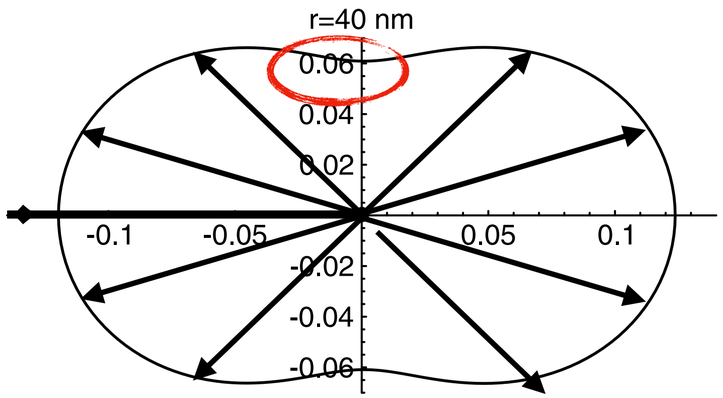
Intuition: we account for the change of radiance at p along direction **ω** by

- Removing the absorbed and out-scattered (extinction) portion of incident ray
- Adding emission
- Adding in-scattering (integrated over all directions)

Scattering vs. Size (Effects Phase Function)

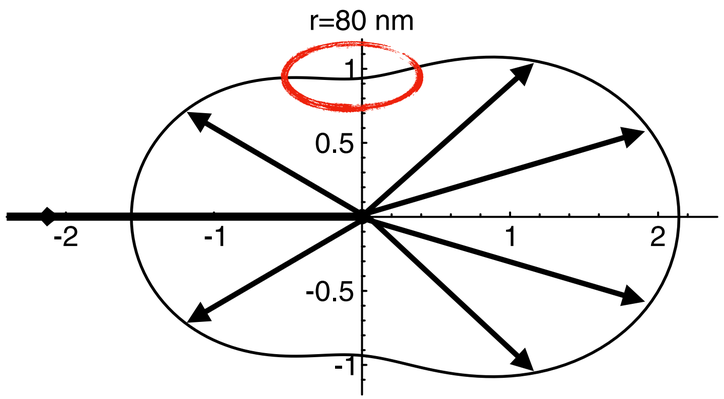
r = 40 nm

(<0.1 λ)



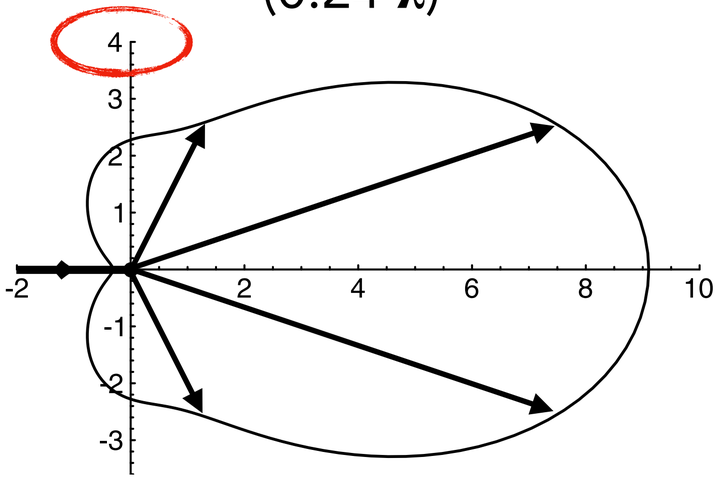
r = 80 nm

(0.16 λ)



r = 120 nm

(0.24 λ)

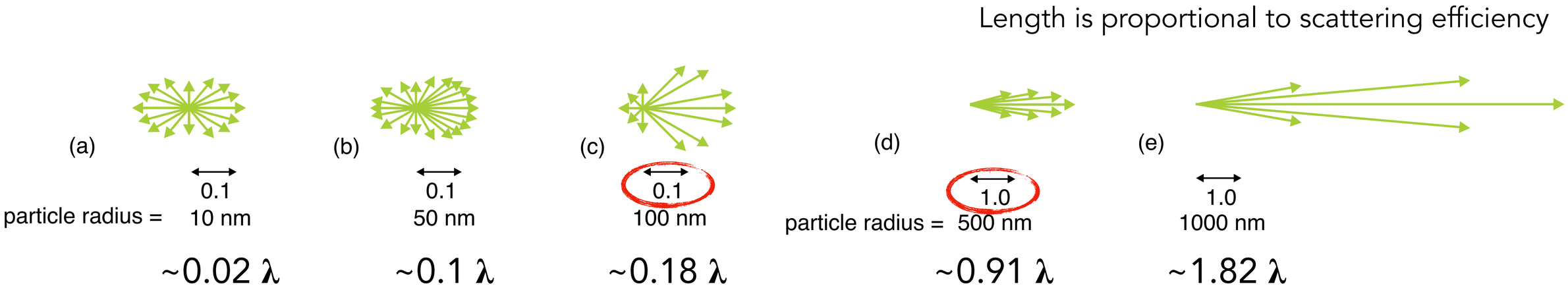


Spherical gold particle scattering **500 nm** photons.

As particle size increases

- significantly more forward scattering + higher scattering efficiency

Scattering vs. Size (Effects Phase Function)



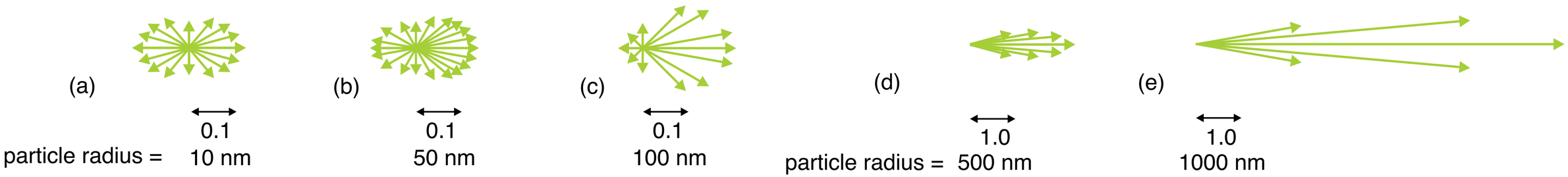
Scattering **550 nm** photons.

As particle size increases

- significantly more forward scattering + higher scattering efficiency

Scattering vs. Size

Length is proportional to scattering efficiency



Small → **Large**

Small cross section area

Low scattering efficiency

Even backward and forward scattering

Increase transparency

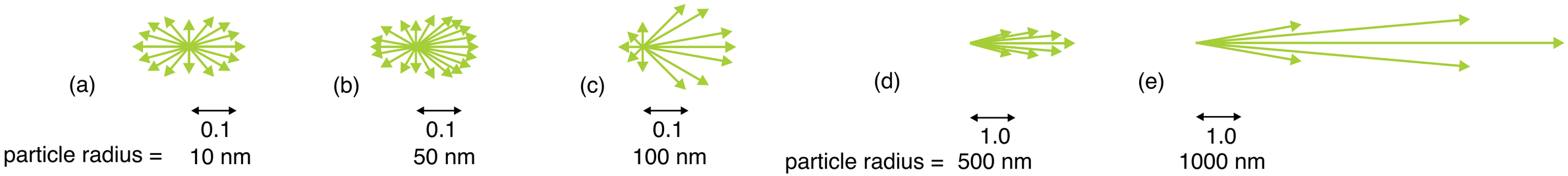
Large cross section area

High scattering efficiency

Mostly forward scattering

Scattering vs. Size

Length is proportional to scattering efficiency



Small → **Large**

Small cross section area

Low scattering efficiency

Even backward and forward scattering

Increase opacity

Large cross section area

High scattering efficiency

Mostly forward scattering

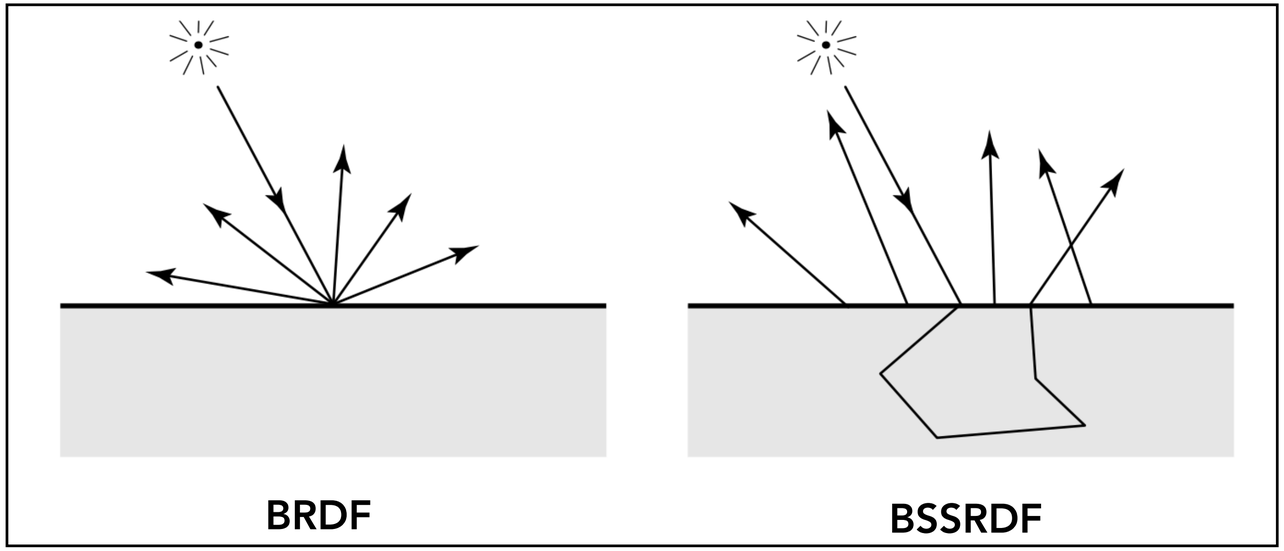
Modeling Translucency Using BSSDF

Bidirectional Subsurface Scattering Distribution Function (BSSDF).

Models how the irradiance of an incident ray affects radiance of other rays at **other spatial points**.

$$L_r(p_r, \omega_r) = \int_A \int_\Omega f_{bssdf}(p_i, \omega_i \rightarrow p_r, \omega_r) L_i(p_i, \omega_i) \cos\theta_i d\omega_i dA$$

Integrate over all points on the surface and from all directions



https://graphics.stanford.edu/papers/bssrdf/bssrdf.pdf

BRDF

[Jensen et al. 2001]

BSSRDF

Derive BSSDF from radiative transfer equation assuming diffuse/isotropic scattering and then use Monte Carlo integration to solve the integration equation.

[Jensen et al. 2001]

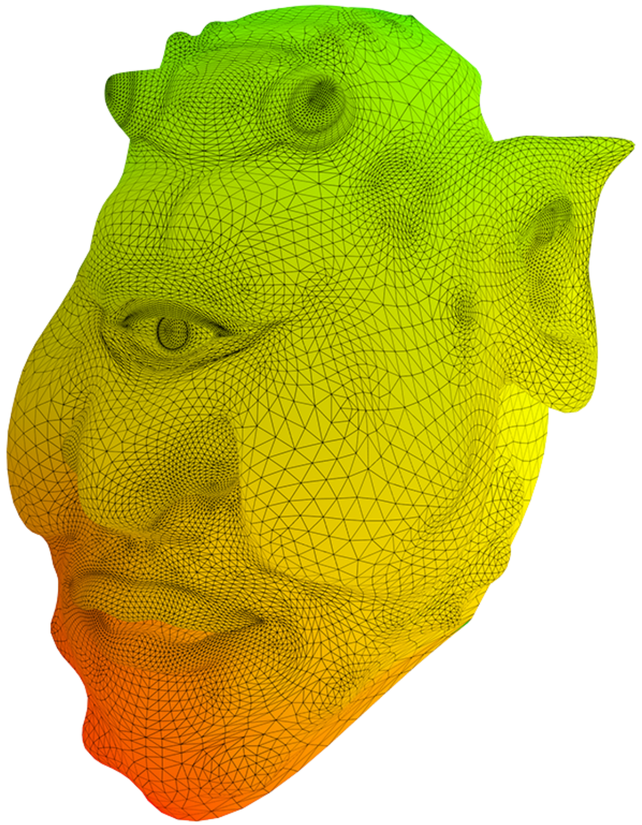
Texture Mapping

Chocolate Wrappers

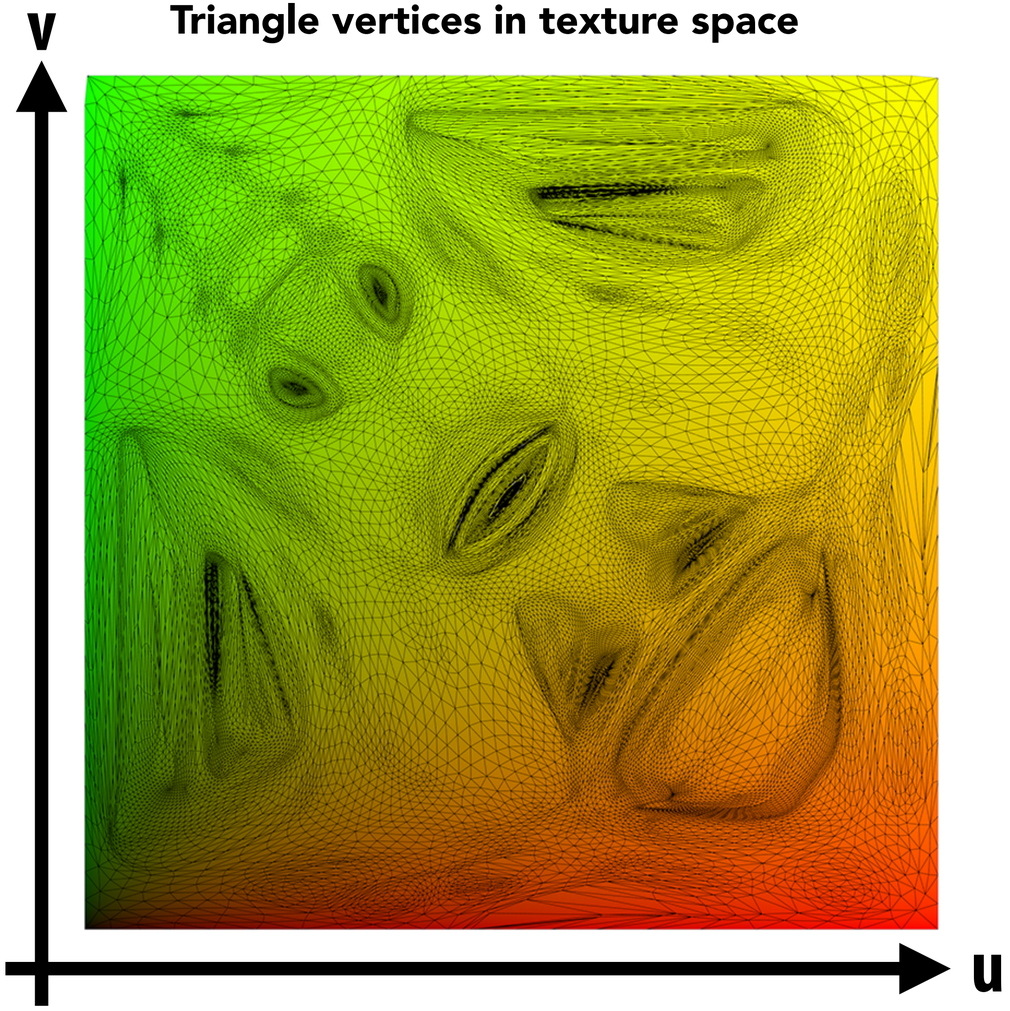
Texture image

Rendering Using Texture

Visualization of texture coordinates

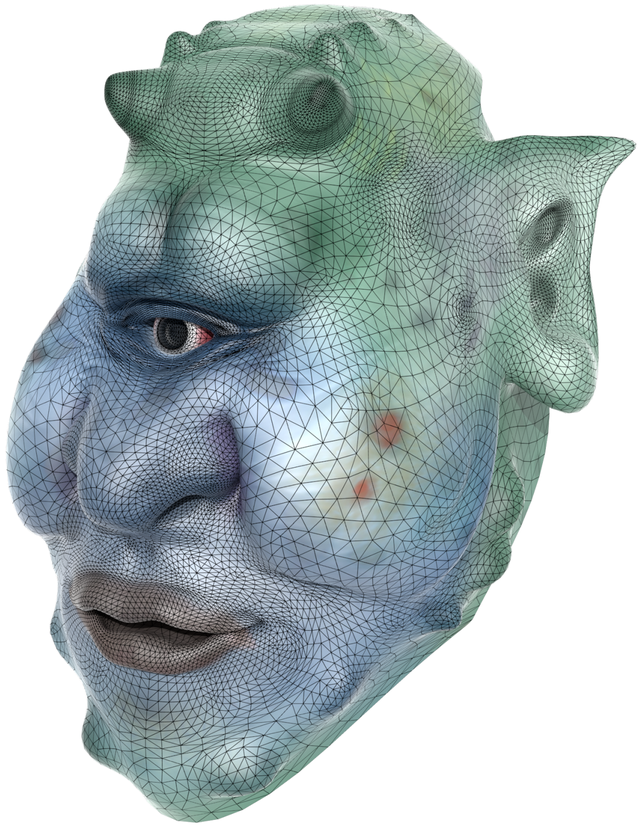


Triangle vertices in texture space

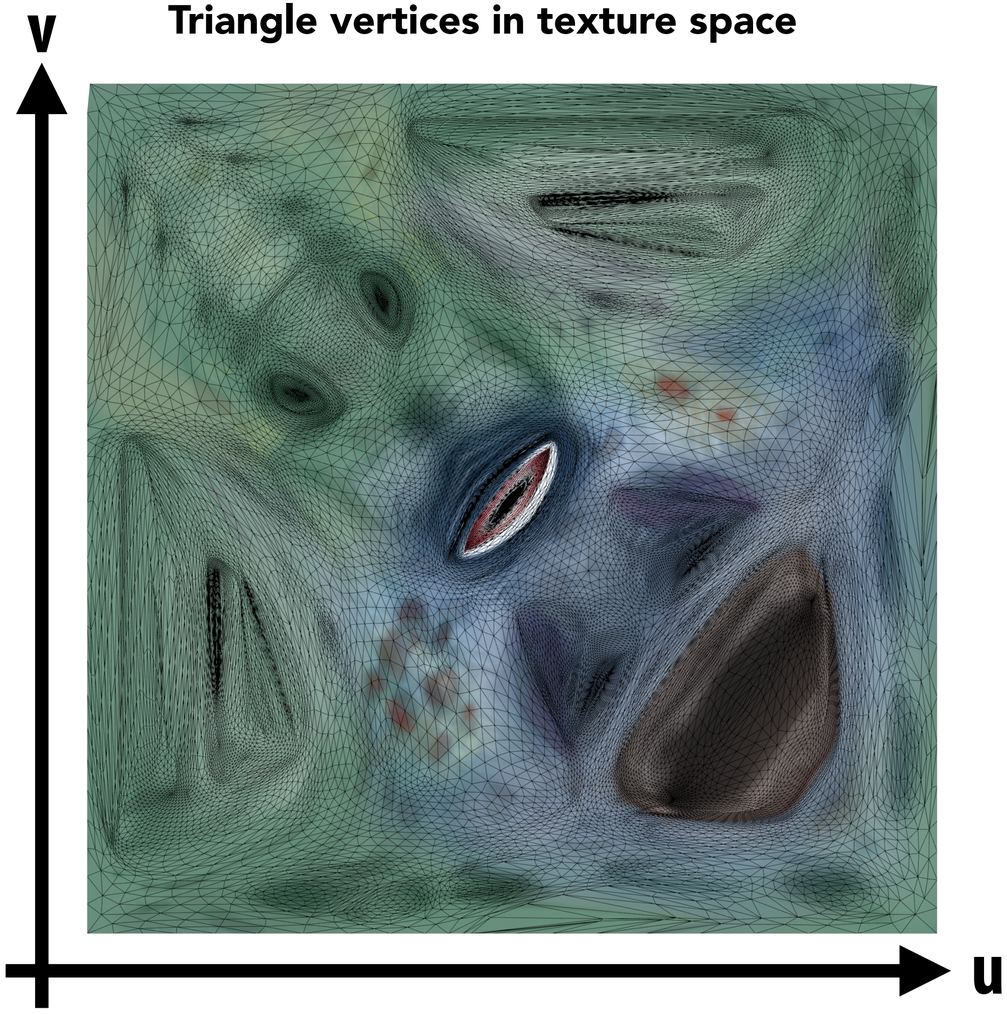


Rendering Using Texture

Rendered result



Triangle vertices in texture space

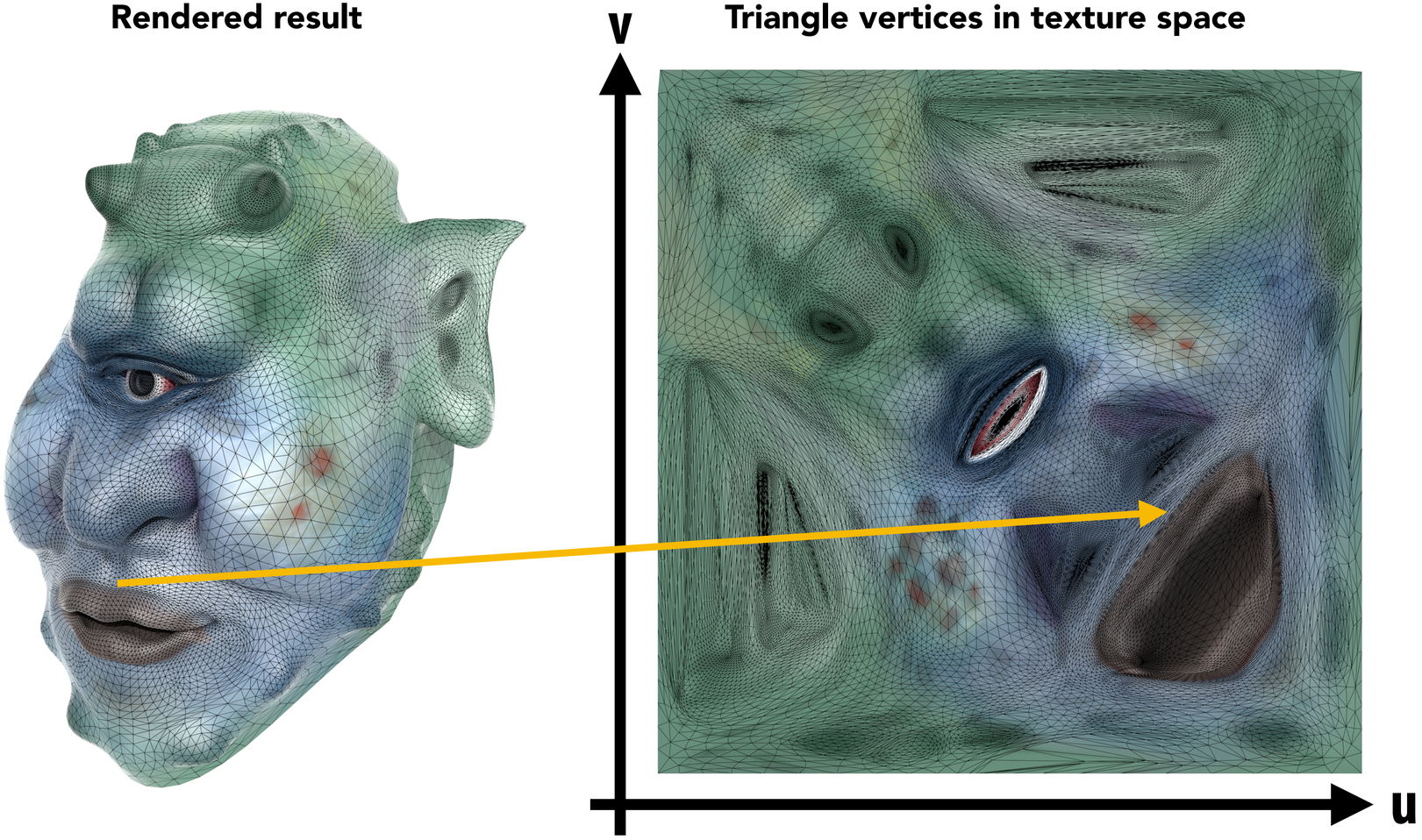


Main Idea (Ideal Scenario)

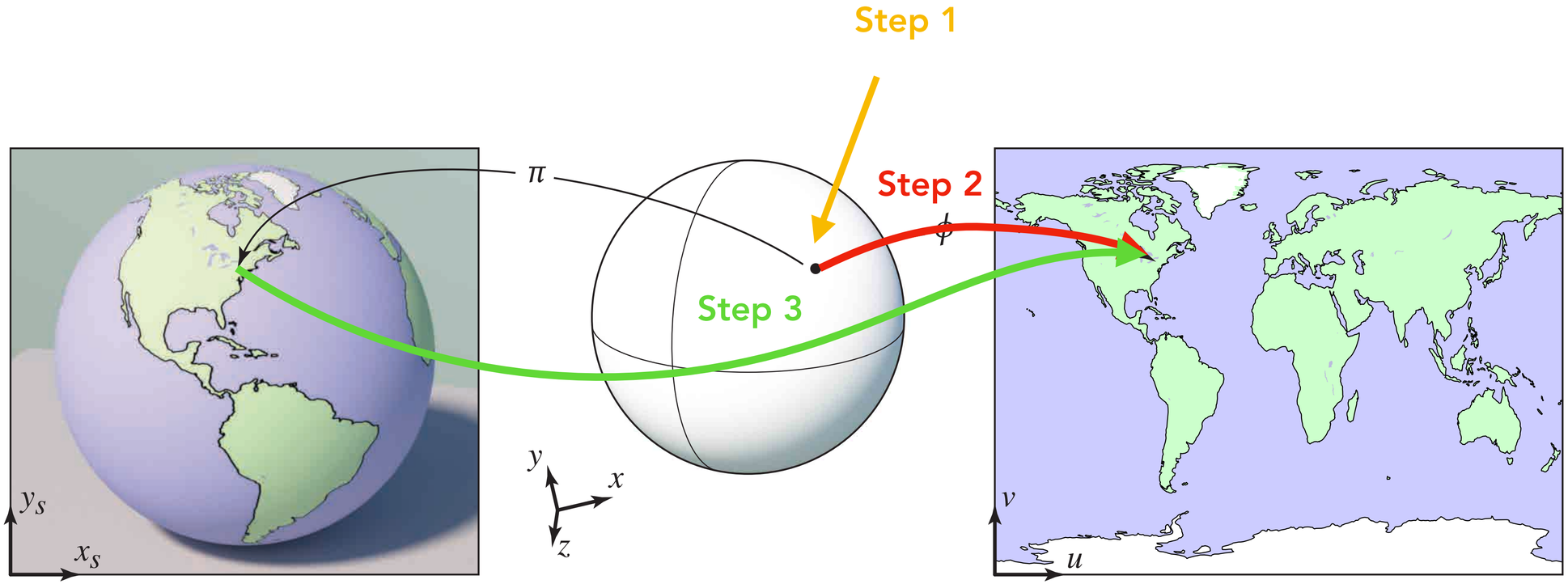
Step 1: pre-compute color information of each point of the scene/mesh.

Step 2: map the 3D scene to a 2D image (a.k.a., **texture map**); record the vertex to texel mapping (e.g., in mesh).

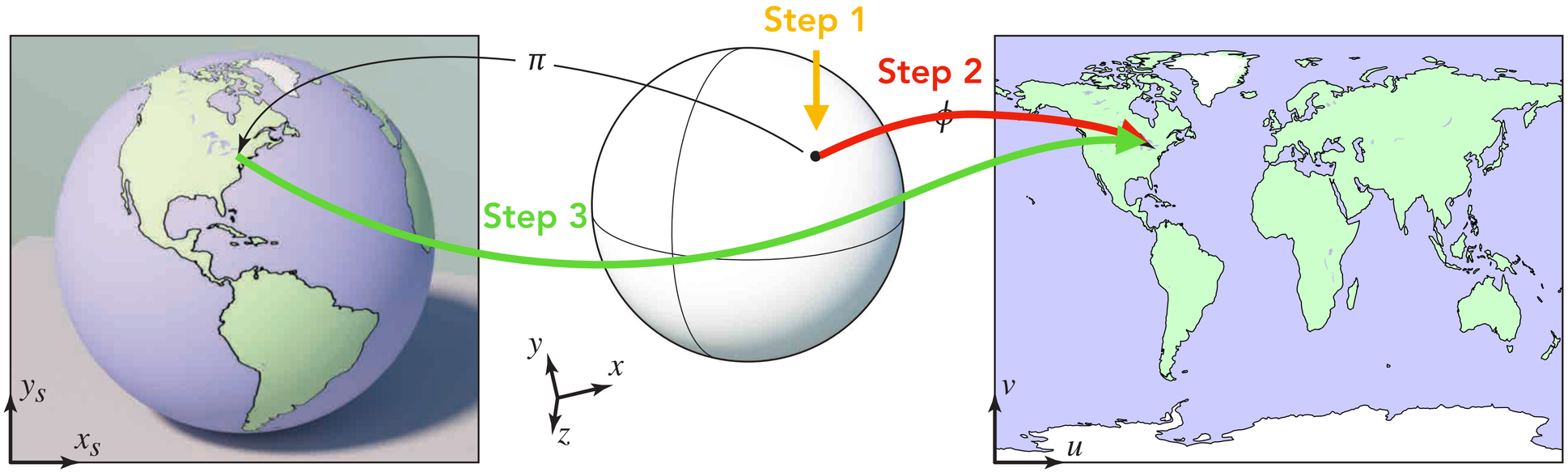
Step 3: during rendering, for each triangle, find the vertices, and then find the triangle in the texture map and copy it!



Main Idea


Step 1
Step 2
Step 3
π
ϕ
y_s
x_s
y
x
z
v
u
Screen space
World space
Texture space

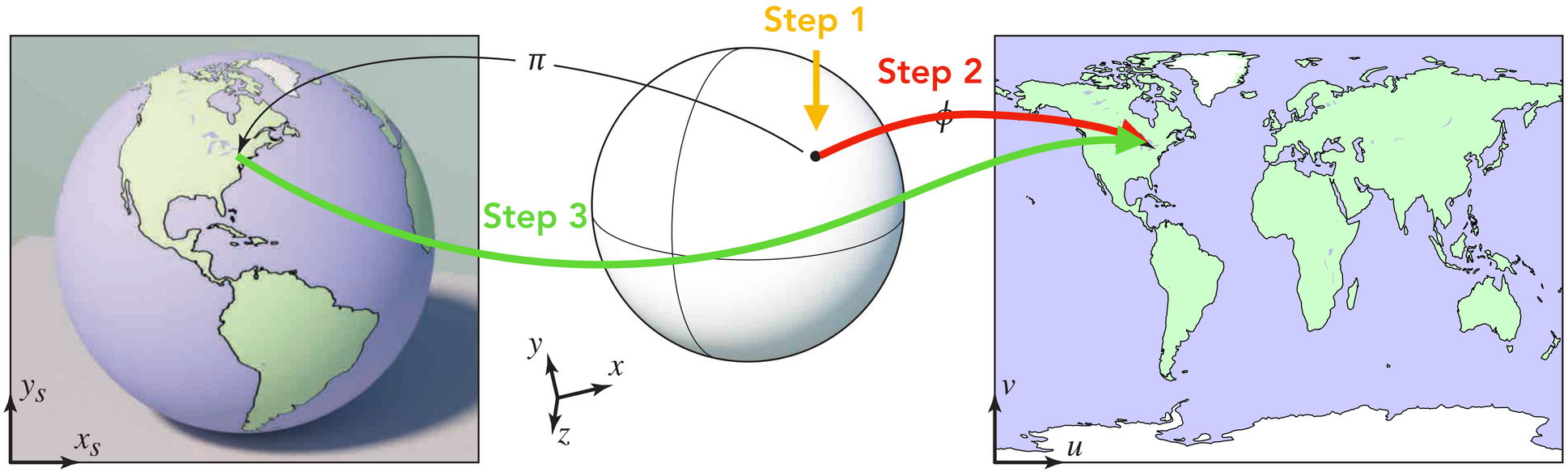
Texture Mapping is (Re-sampling)



Screen space **World space** **Texture space**

Step 1 necessarily has to sample the mesh, since we can't possible calculate color (or any other information) for infinitely many points.

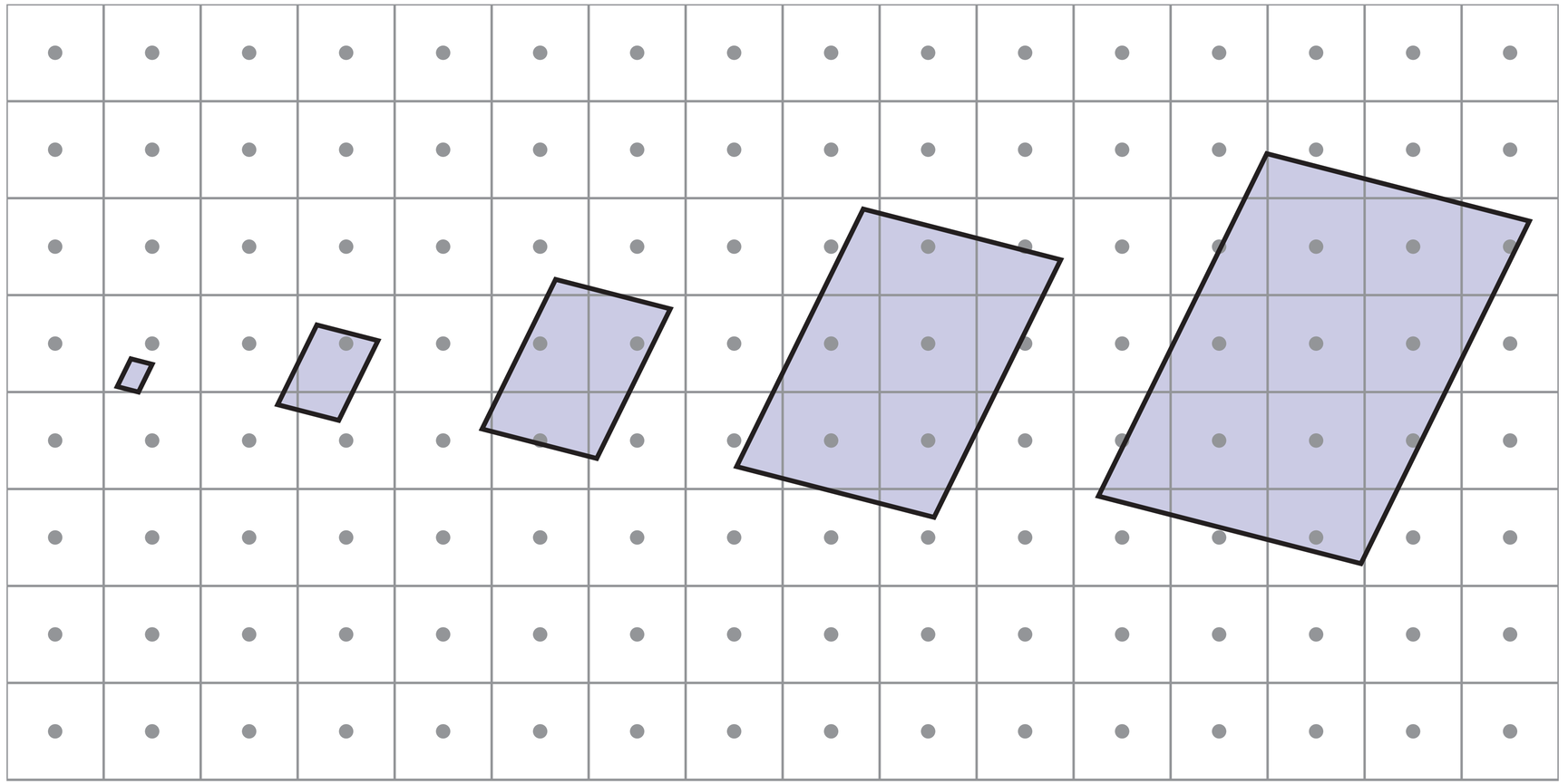
Texture Mapping is (Re-sampling)



Screen space World space Texture space

Step 3 again has to sample, because a screen space point most likely won't be a point whose information is calculated in Step 1.

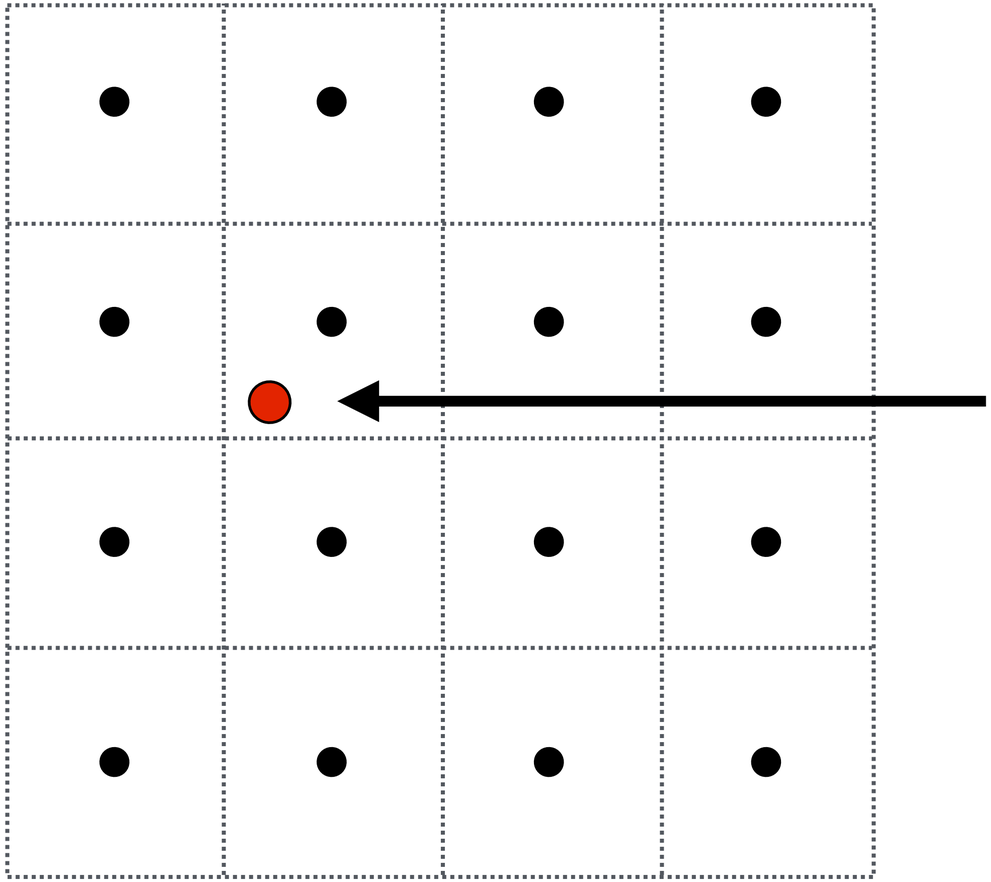
Screen Pixel Footprint in Texture



Upsampling
(Magnification)

Downsampling
(Minification)

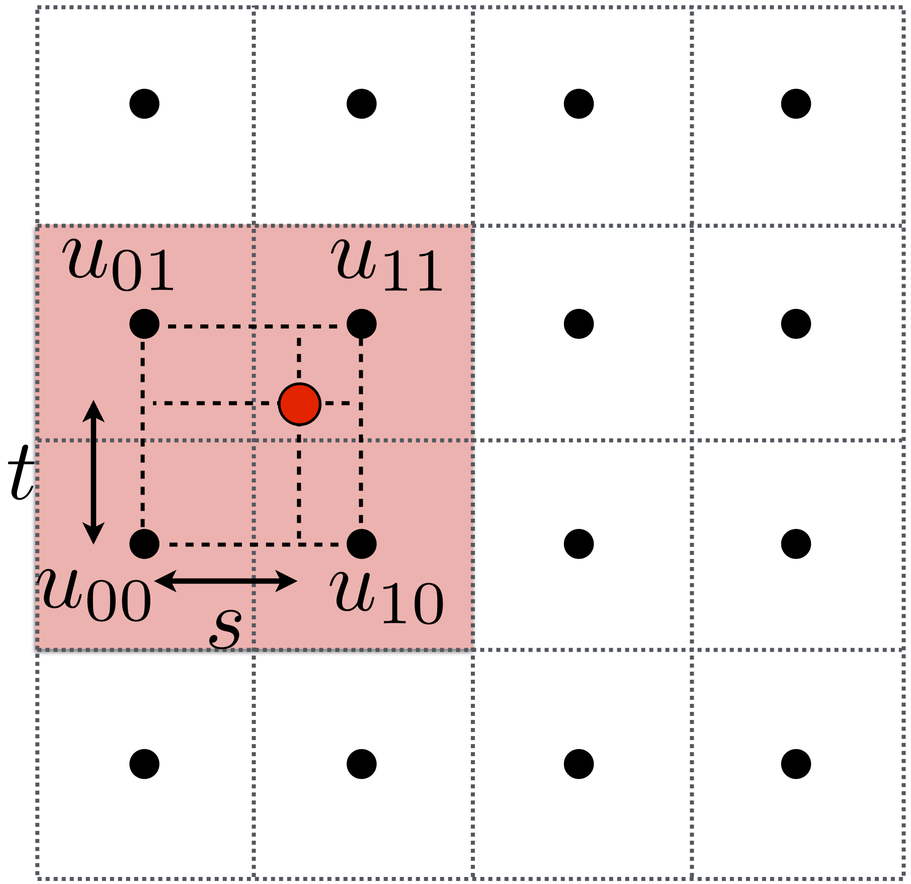
Bilinear Filtering



Want to sample texture value f(u,v) at red point

Black points indicate texture sample locations

Bilinear Filtering

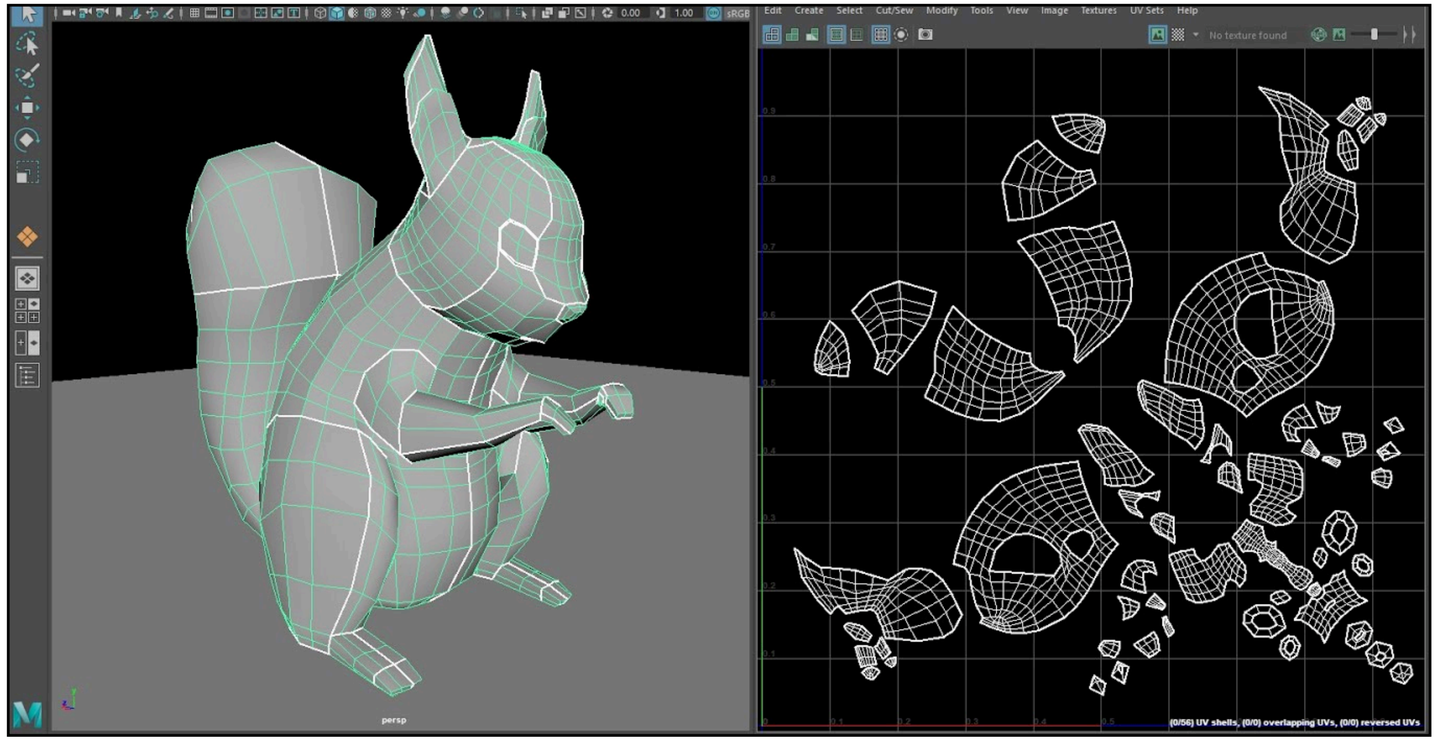


And fractional offsets, (s,t) as shown

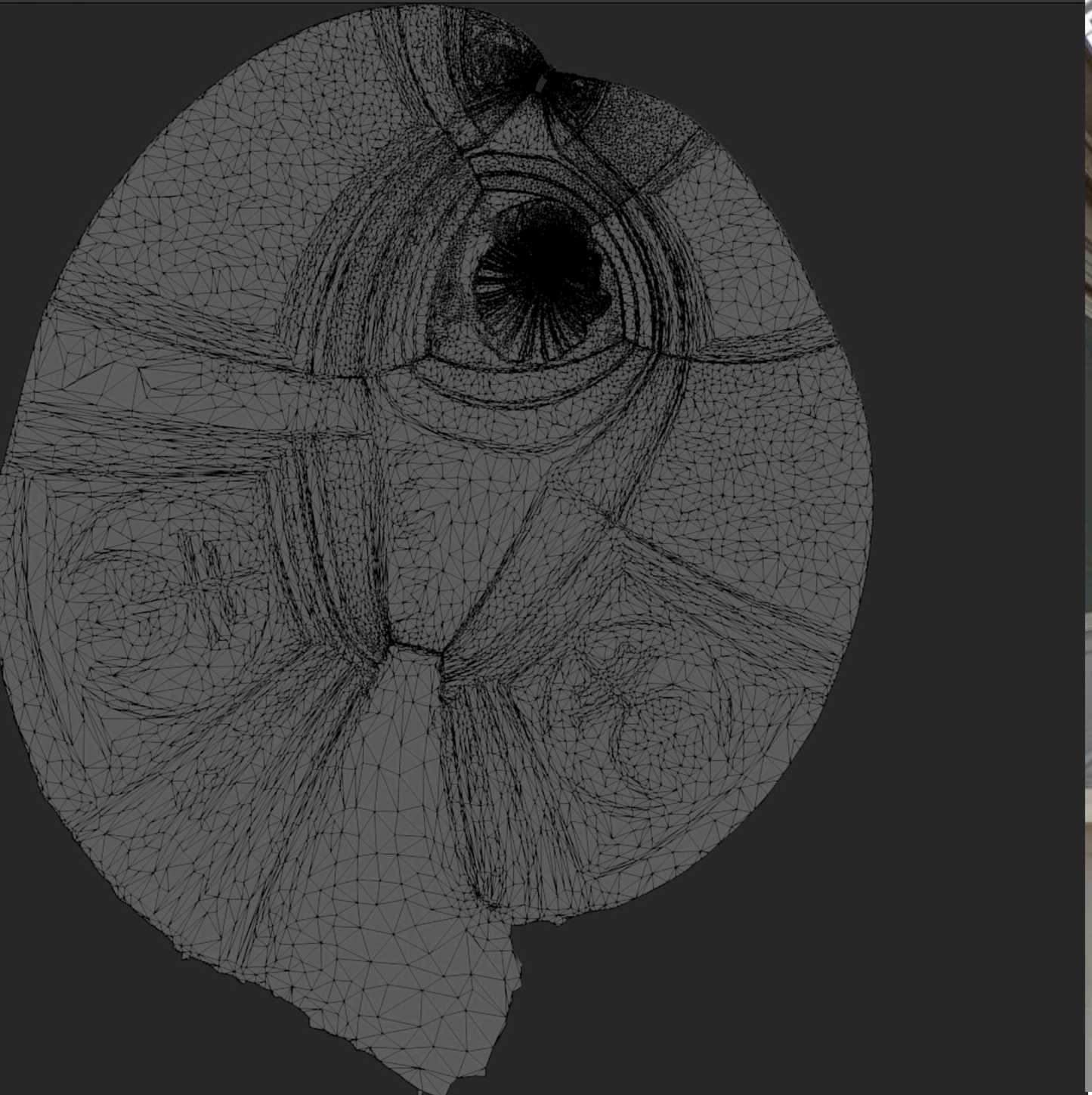
UV Unwrapping

Some complicated meshes can’t be unwrapped to one continuous area in the UV map; instead, many “UV islands” are created.

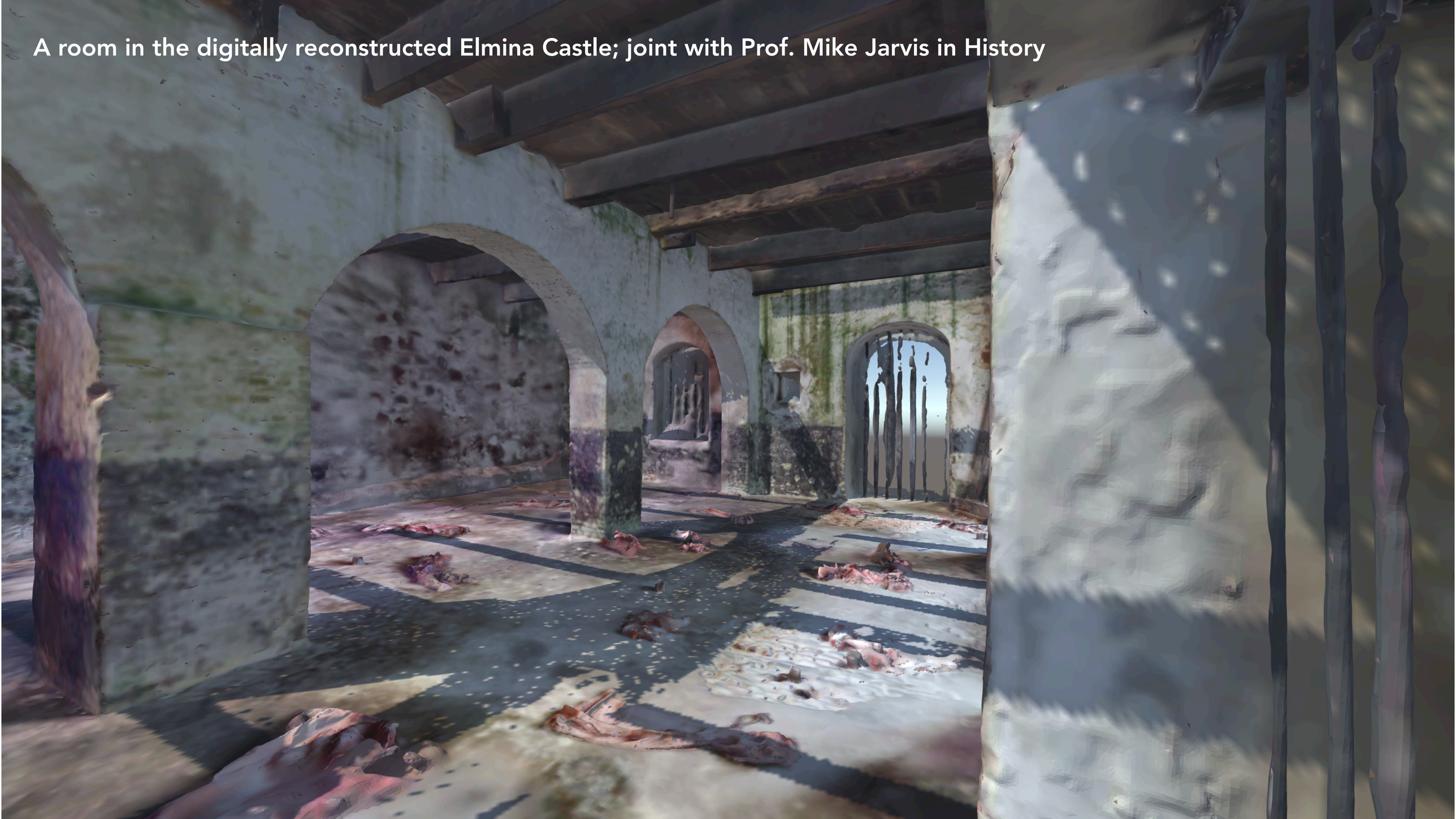
A trade-off between minimizing the total texture map area (compactness) vs. minimizing distortion within each face.



http://chandaniparekh.blogspot.com/2018/02/uv-unwrapping.html

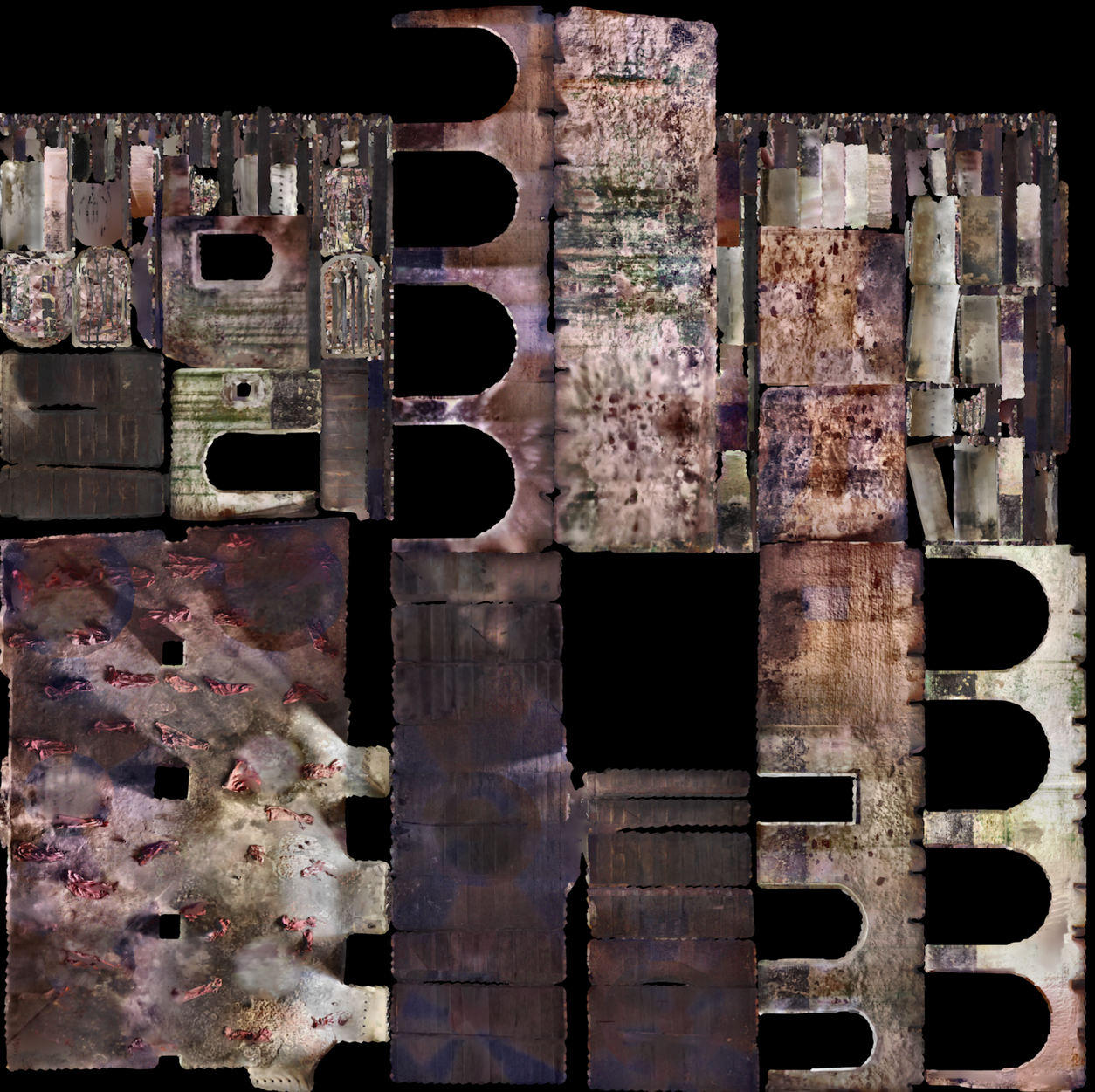


https://twitter.com/yzhu88/status/1266899006715760640

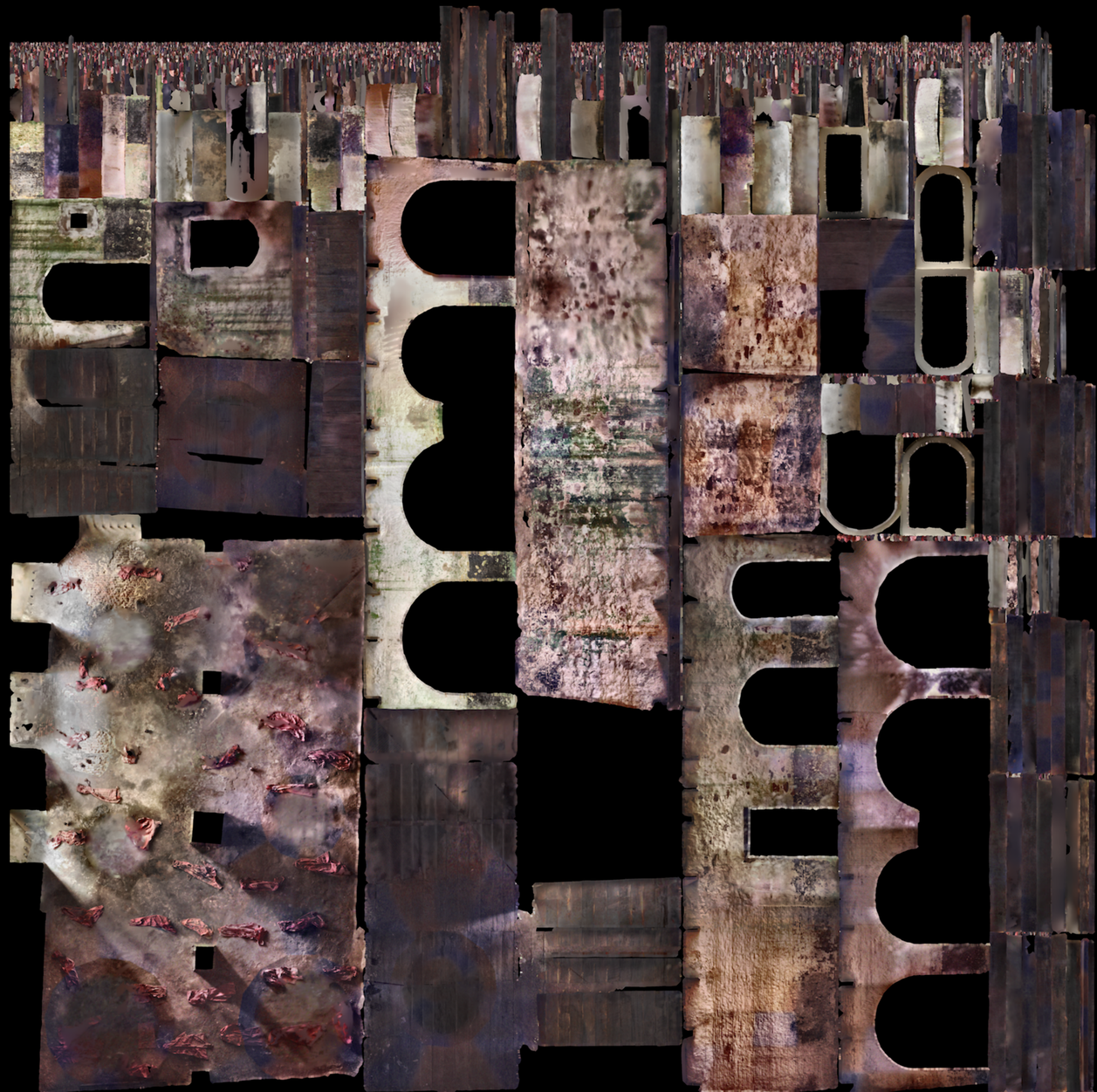


A room in the digitally reconstructed Elmina Castle; joint with Prof. Mike Jarvis in History

UV Wrapping by Meshlab



UV Wrapping by Blender



Another room in Elmina Castle