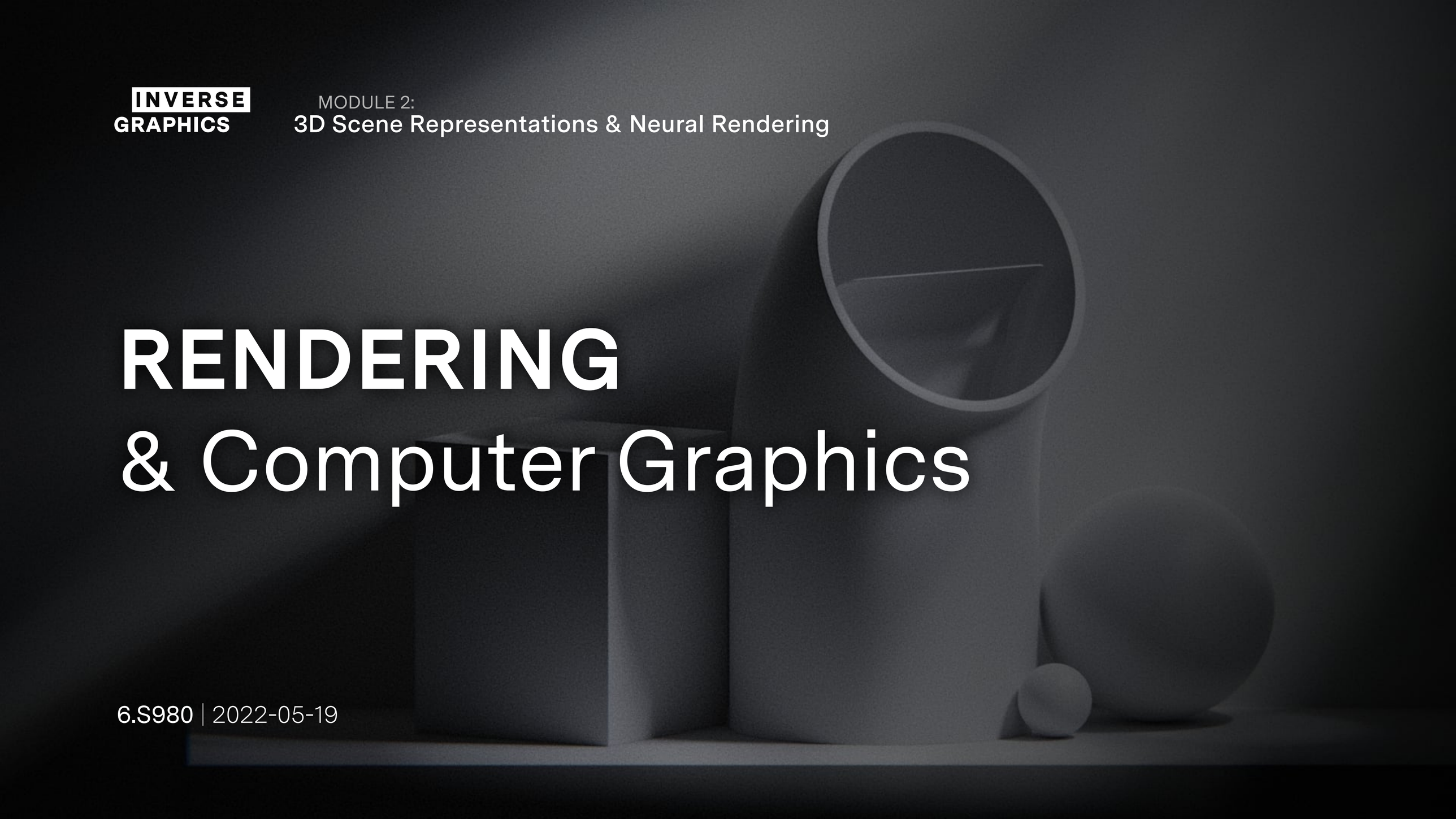


INVERSE GRAPHICS

MODULE 2:
3D Scene Representations & Neural Rendering

RENDERING & Computer Graphics

6.S980 | 2022-05-19

3D Scene

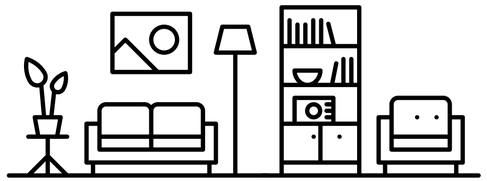


Image Formation

Inference

Neural Scene Representation

Downstream Tasks

Graphics

“inverse graphics”…?

Today: Light Transport & High-level of Physics-based Rendering

3D Scene

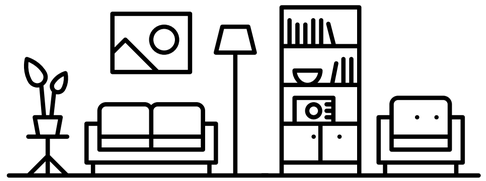


Image Formation

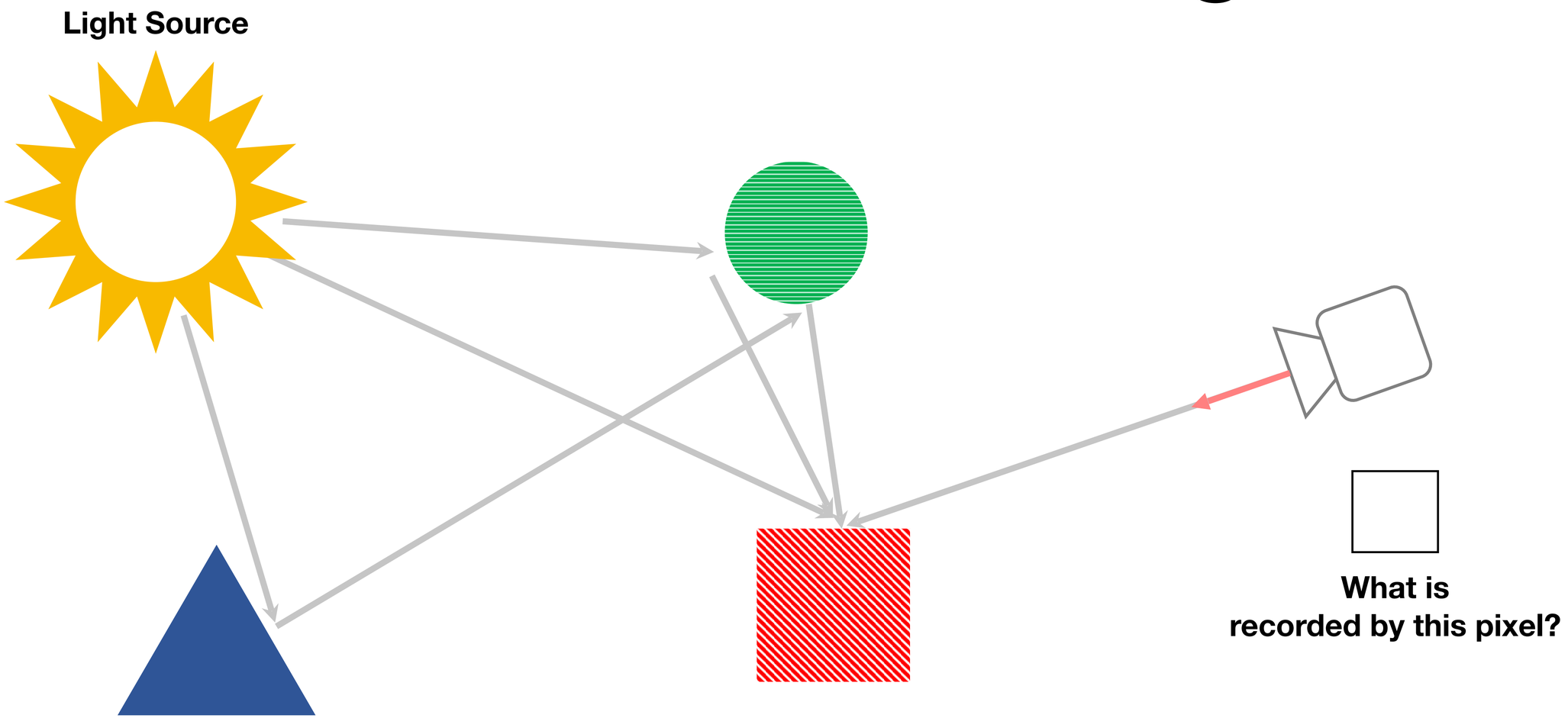
Inference

Neural Scene Representation

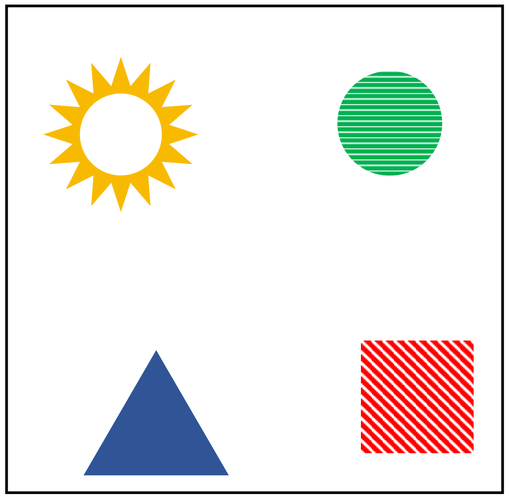
Downstream Tasks

Why?	In order to reason about the 3D world from images, we need to understand how 3D properties such as materials, lighting, etc. relate to the measurements observed by a camera.
What you'll learn.	The rendering equation, simulating light transport via multi-bounce ray-tracing, bidirectional radiance distribution functions, rasterization, rendering.

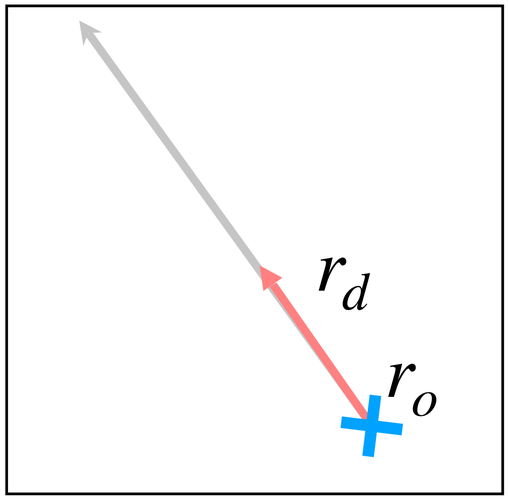
What is Rendering?



Function Signature of Rendering



Scene : Light sources, materials, shapes, …



Camera Ray

Radiance of the ray

Some Slides Adopted From

Stanford CS348I: Computer Graphics in the Era of AI

Profs. C. Karen Liu and Jiajun Wu

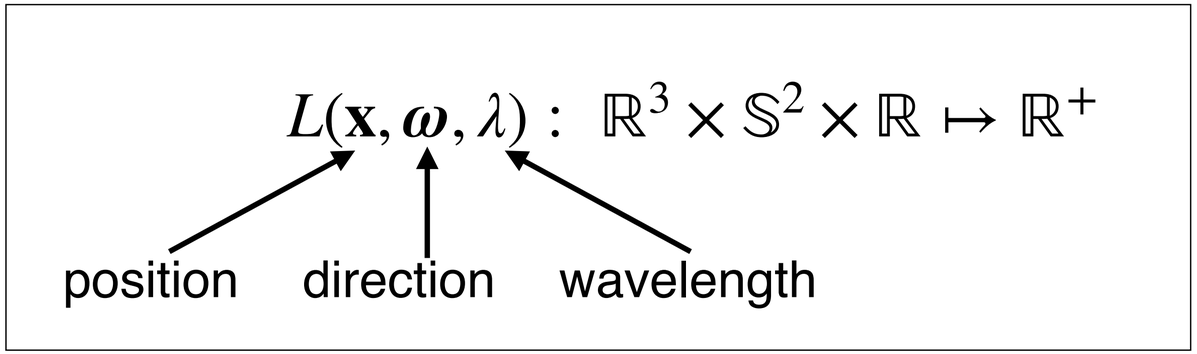
The Rendering equation

The moral of the story:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

Radiance

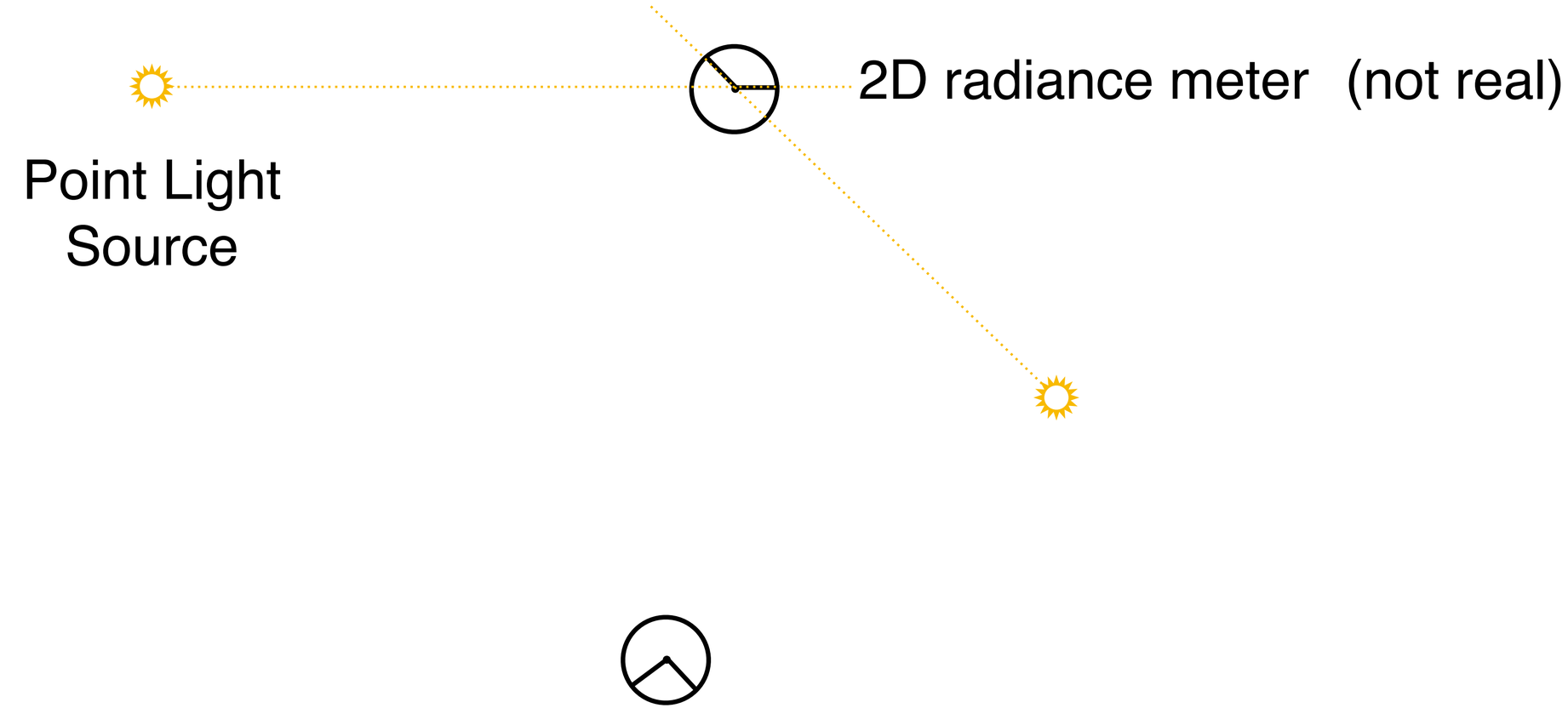
- Radiance is a fundamental quantity of light.
- Radiance is density of photons at a point, traveling in the same direction, at given wavelength.
- Radiance is energy per time, per unit area, per unit solid angle, per wavelength.



The “Light Field”: Radiance for every possible ray

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Radiance intuition

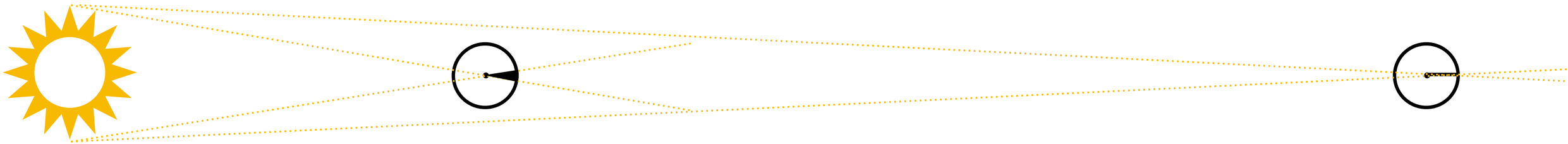


Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Radiance intuition

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Radiance intuition



- Radiance properties
  - Radiance along an unblocked ray is constant.
  - Response of sensors is proportional to radiance of visible surface.
  - Radiance of reflected light is proportional to that of incoming light:

$$L(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) \propto L(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)$$

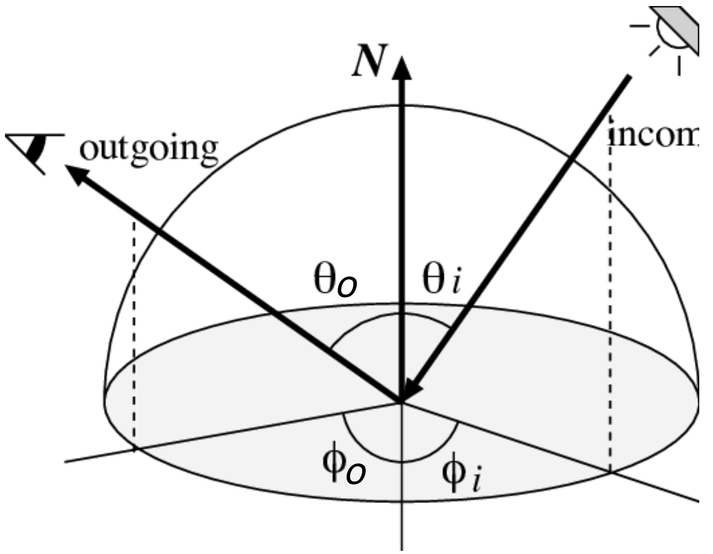
reflected incoming

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

BRDF

- Constant of proportionality defines Bidirectional Reflectance Distribution Function (BRDF).

$$f(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = \frac{\text{outgoing light in direction } \boldsymbol{\omega}_o}{\text{incoming light in direction } \boldsymbol{\omega}_i}$$



- BRDF describes how a material reflects light.

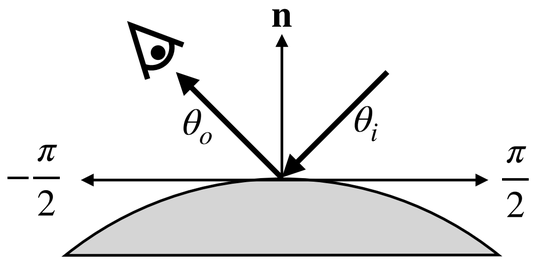
- Helmholtz reciprocity: $f(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2) = f(\boldsymbol{\omega}_2, \boldsymbol{\omega}_1)$

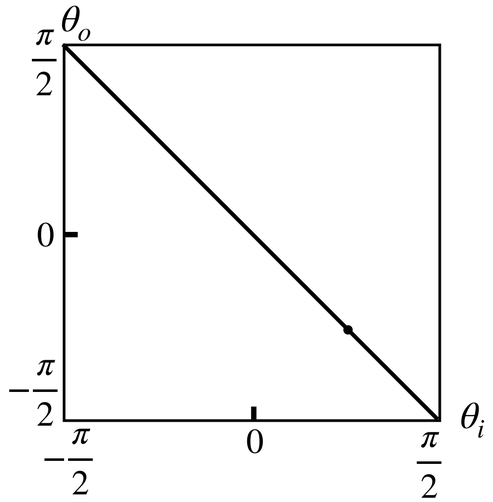
$$\boldsymbol{\omega} = (\theta, \phi)$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

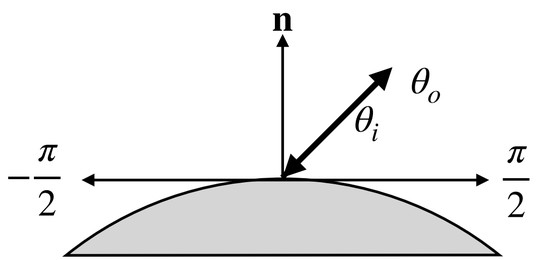
BRDF intuition

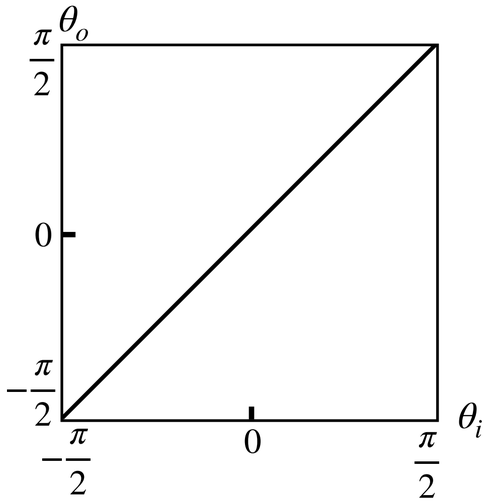
perfect mirror



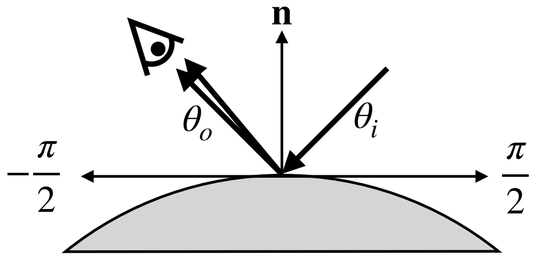


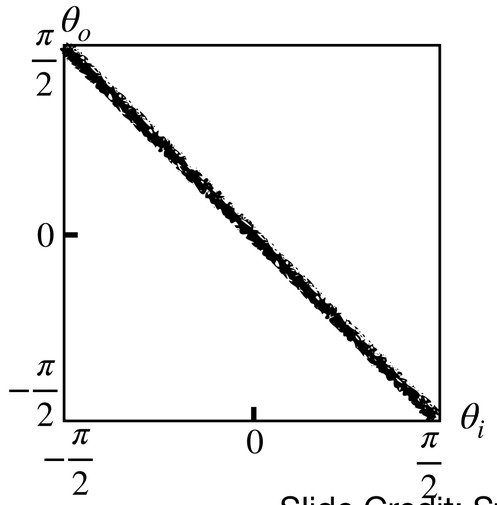
retro-reflector



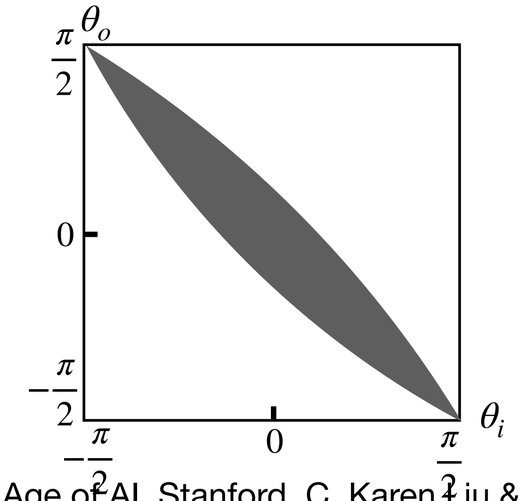


fairly shiny mirror



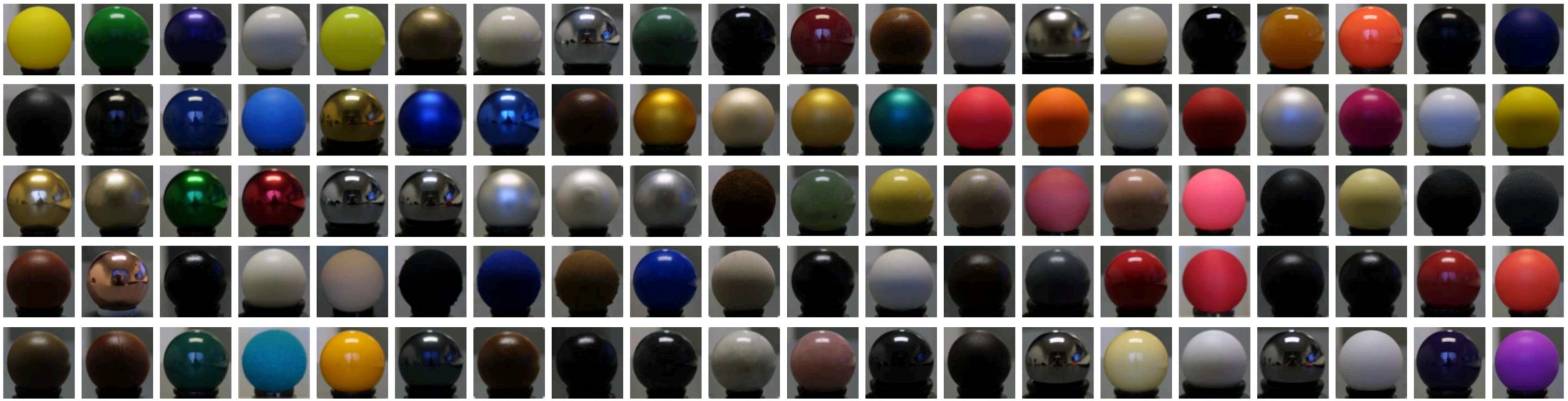


How does BRDF for paper look like?



Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

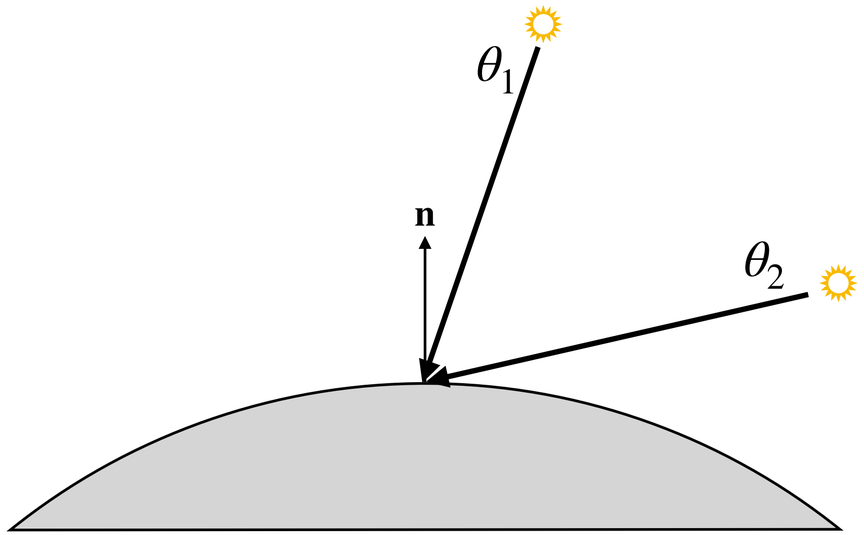
Different materials

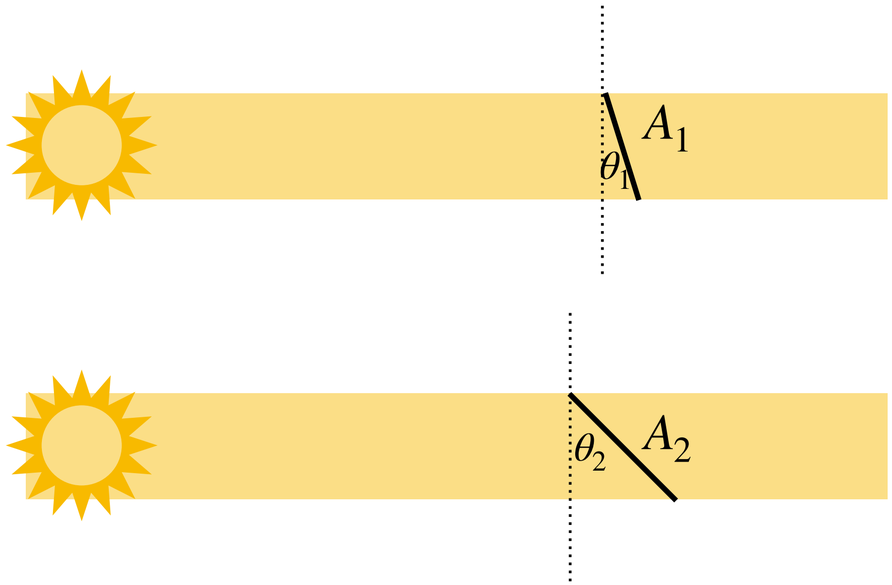


Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Angle Weight of Photon density

Which incoming light results in more photon density at the surface?





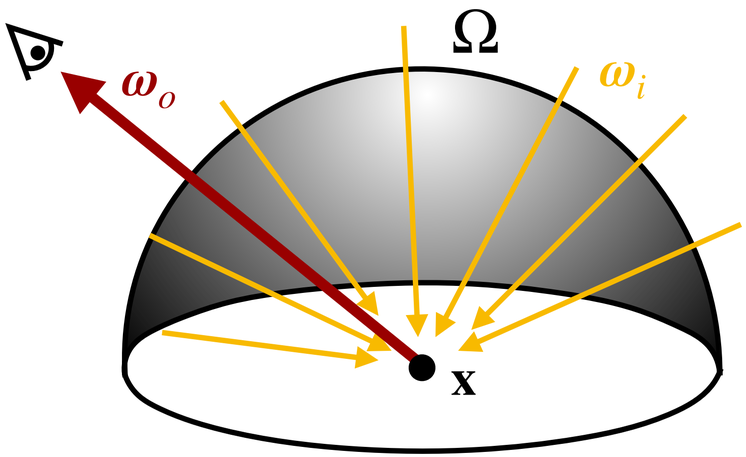
Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Rendering equation

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

- emitted radiance
- outgoing radiance in $\boldsymbol{\omega}_o$
- fraction of incoming light that is reflected into $\boldsymbol{\omega}_o$
- incoming radiance
- angle weight of photon density
- solid angle differential

Conservation of energy: $\int_{\boldsymbol{\omega}_i \in \Omega} f(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)\, (\boldsymbol{\omega}_i \cdot \mathbf{n})\, d\boldsymbol{\omega}_i \leq 1$

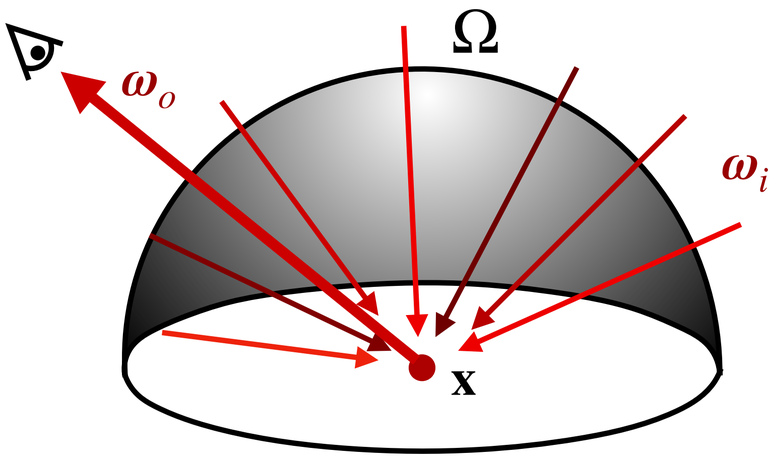


Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Colors

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^R) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^R) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda^R)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda^R)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$



Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Colors

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^R) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^R) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda^R)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda^R)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^G) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^G) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda^G)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda^G)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^B) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^B) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda^B)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda^B)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

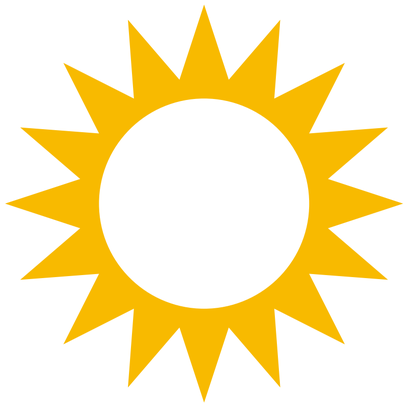
Colors

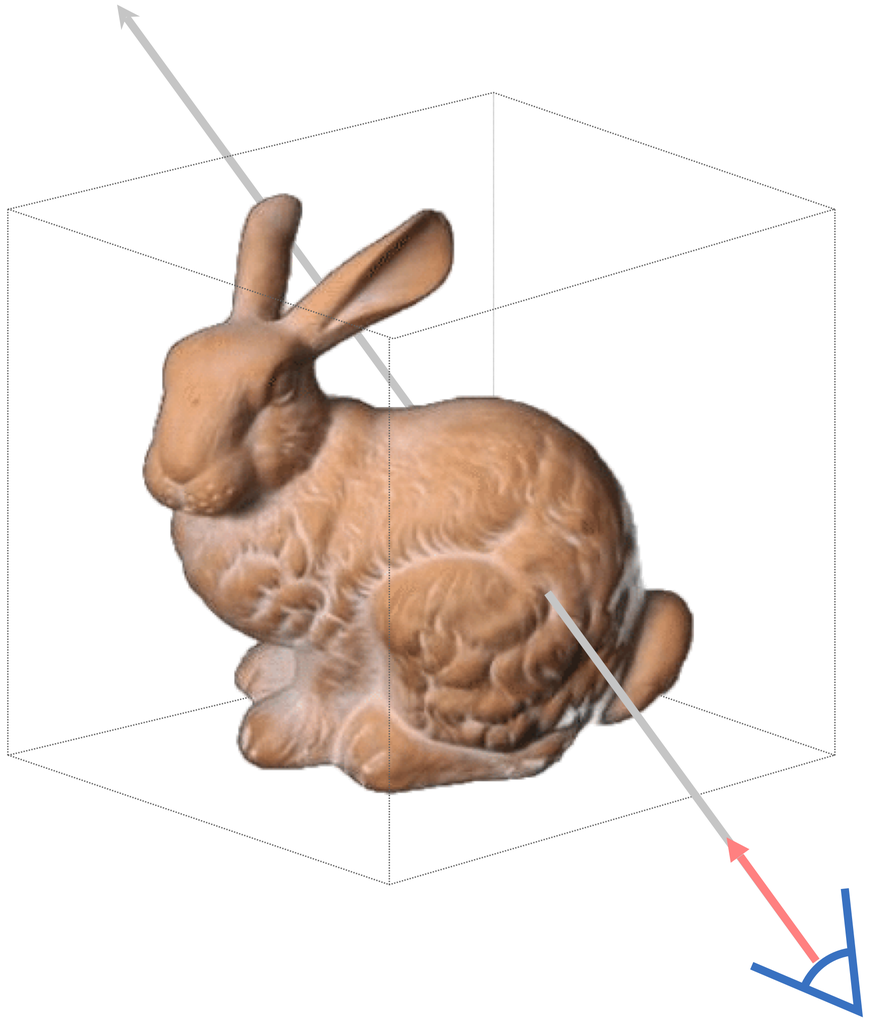
$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L(\mathbf{x}, \boldsymbol{\omega}) : \ \mathbb{R}^3 \times \mathbb{S}^2 \mapsto \mathbb{R}^3$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Ok, sure, but how do we actually render our bunny?





Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Rendering equation

Rendering equation: $L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\ L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Shading

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

reflectance property

light source

angle weight

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

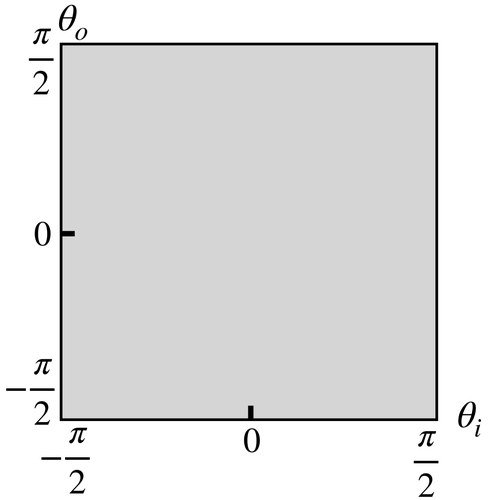
Lambertian surfaces

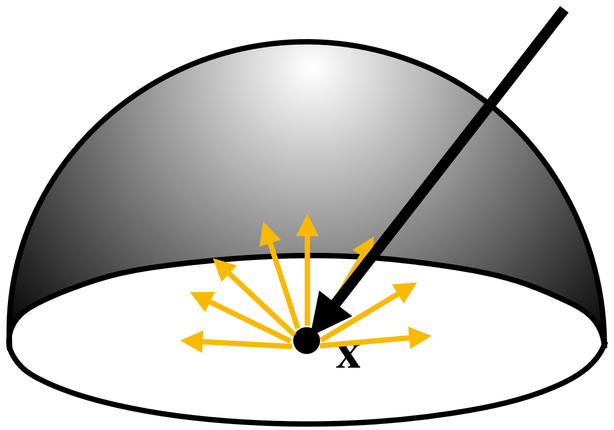
$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

reflectance property — light source — angle weight

What does the BRDF look like for Lambertian surfaces?

$$f(\mathbf{x}, \cdot, \cdot, \lambda) = \text{constant} = C_{surf}$$





Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Lambertian surfaces

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\ L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

reflectance property — light source — angle weight

What does the BRDF look like for

Assumption: Lambertian surfaces

$$f(\mathbf{x}, \cdot, \cdot, \lambda) = \text{constant} = C_{surf}$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + C_{surf} \int_\Omega L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

Assumption: Direct lights only

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + C_{surf} \sum_i L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n})$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Diffuse light

C_{surf} and C_{l_i} are 3x1 vectors representing RGB colors

$\circ$ is elementwise multiplication

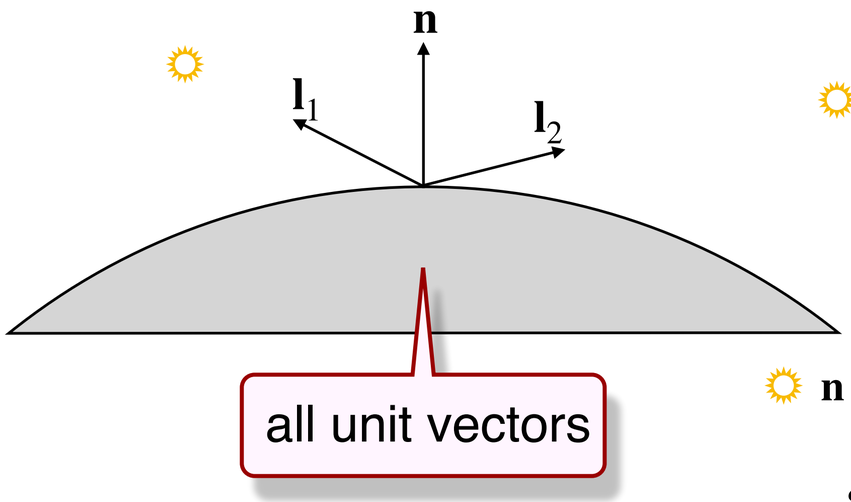
$$C = C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i)$$

surface color

light color

Light locations matter,

but eye location doesn't!



$\mathbf{n} \cdot \mathbf{l} < 0$

What's the perceived color if we shine a blue light on a red diffuse surface?

1. purple
2. red
3. black

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + C_{surf} \sum_i L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n})$$

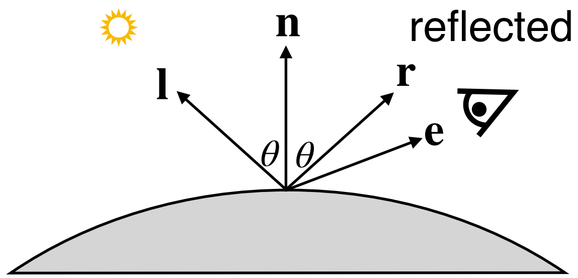
Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Ambient light and specular light

$$C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$$

ambient light

specular light



$(\mathbf{r} \cdot \mathbf{e})^p$

Assumption: Lambertian surfaces

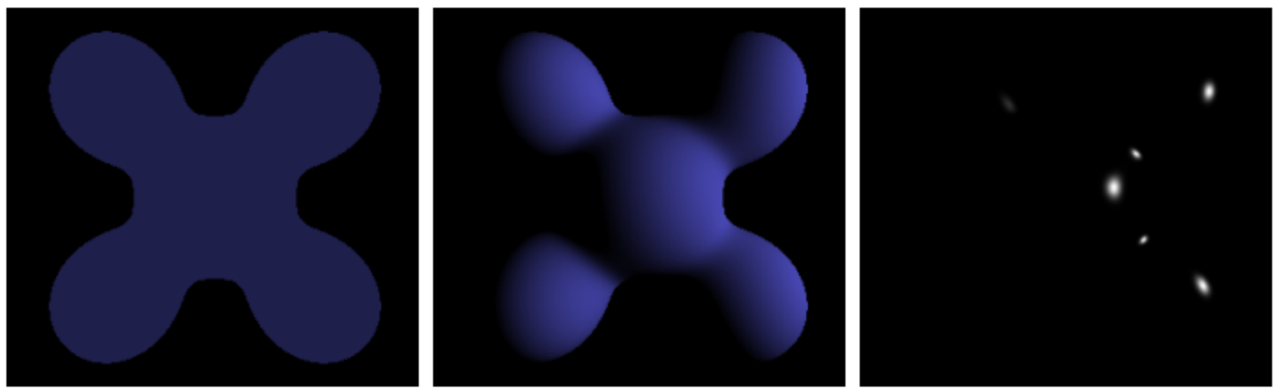
Assumption: Direct lights only

Ambient term: approximates the average color of all surfaces in the scene to make up for indirect lights from every direction.

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Phong illumination model

$$C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$$

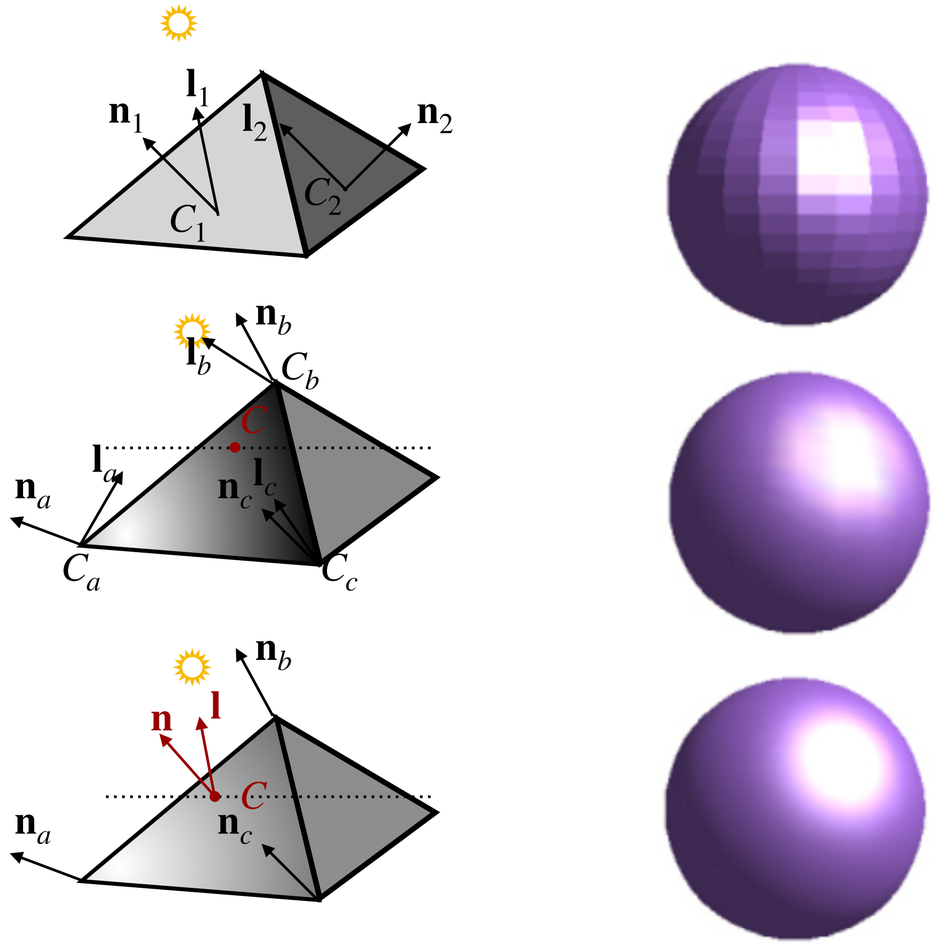


Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Apply shading

- Where to apply shading equation?
  - Per polygon (Flat Shading): one normal and one light vector per triangle.
  - Per vertex (Gouraud Interpolation): compute color at vertices of meshes and interpolate color during **rasterization**.
  - Per pixel (Phong interpolation): interpolate normals of vertices during rasterization and compute color for each interpolated normal.

The process of computing the mapping from scene geometry to pixels



Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Rendering equation

photorealism

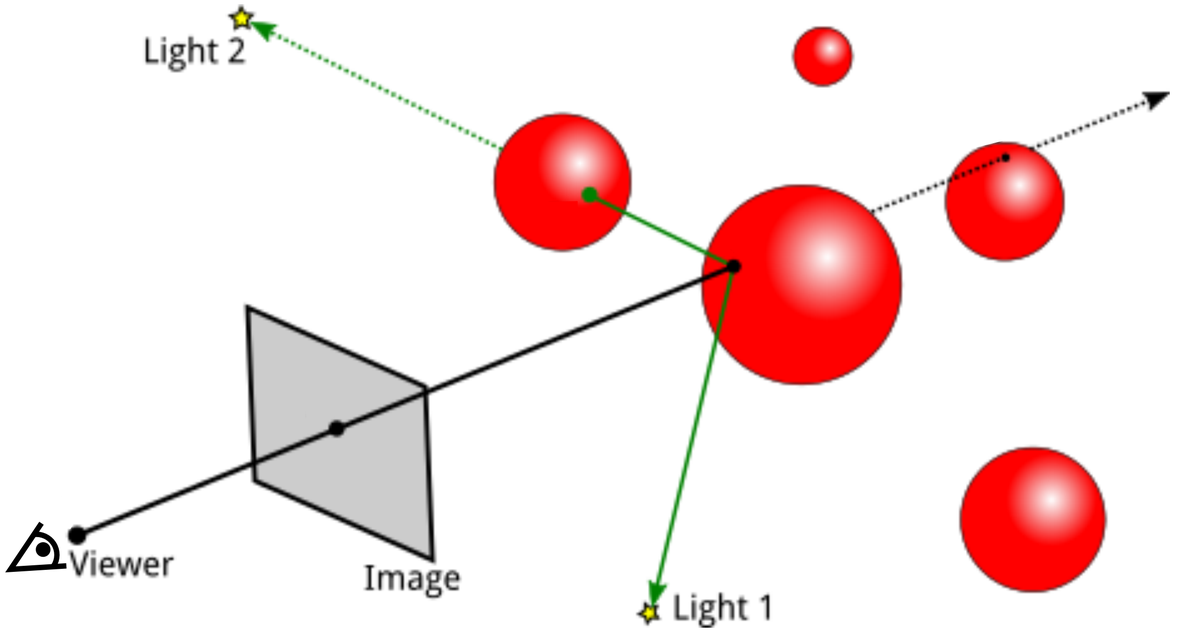
Rendering equation: $L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\ L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$

Can’t do shadows, refraction, indirect light and inter-reflections…

Phong model with **rasterization**: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Ray tracing



for each pixel (x, y)

create ray R from eye through (x, y)

for each object O in scene

if R intersects O and is closest so far

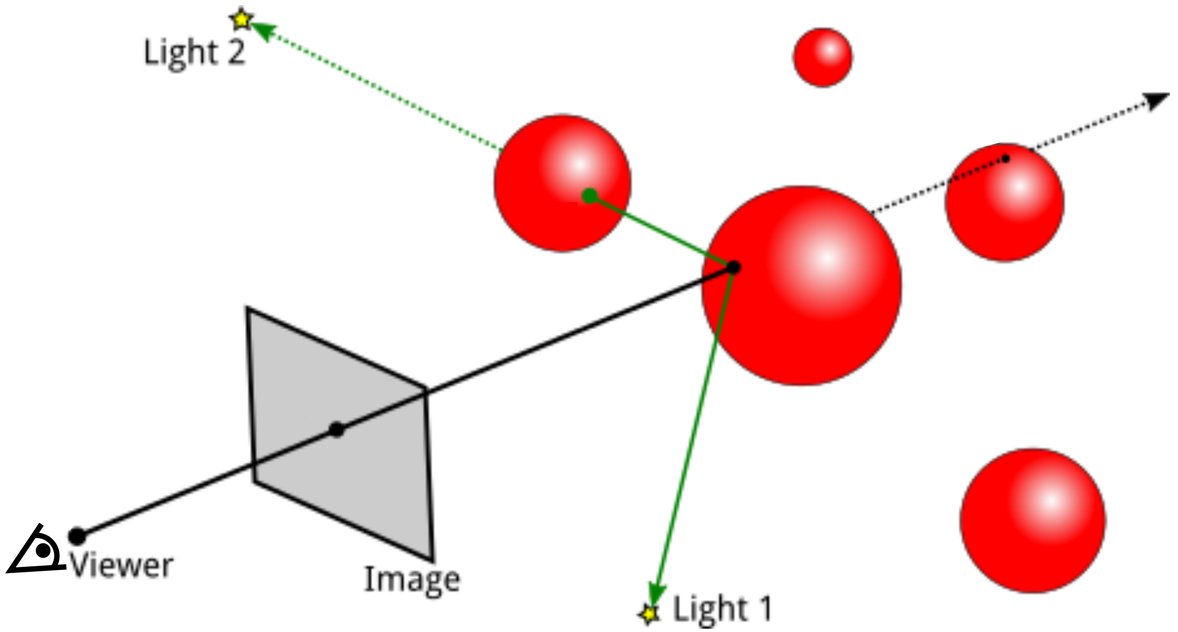
record this intersection

shade pixel (x, y) based on nearest intersection

image: David Eck

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Ray tracing



for each pixel (x, y)

create ray R from eye through (x, y)

for each object O in scene

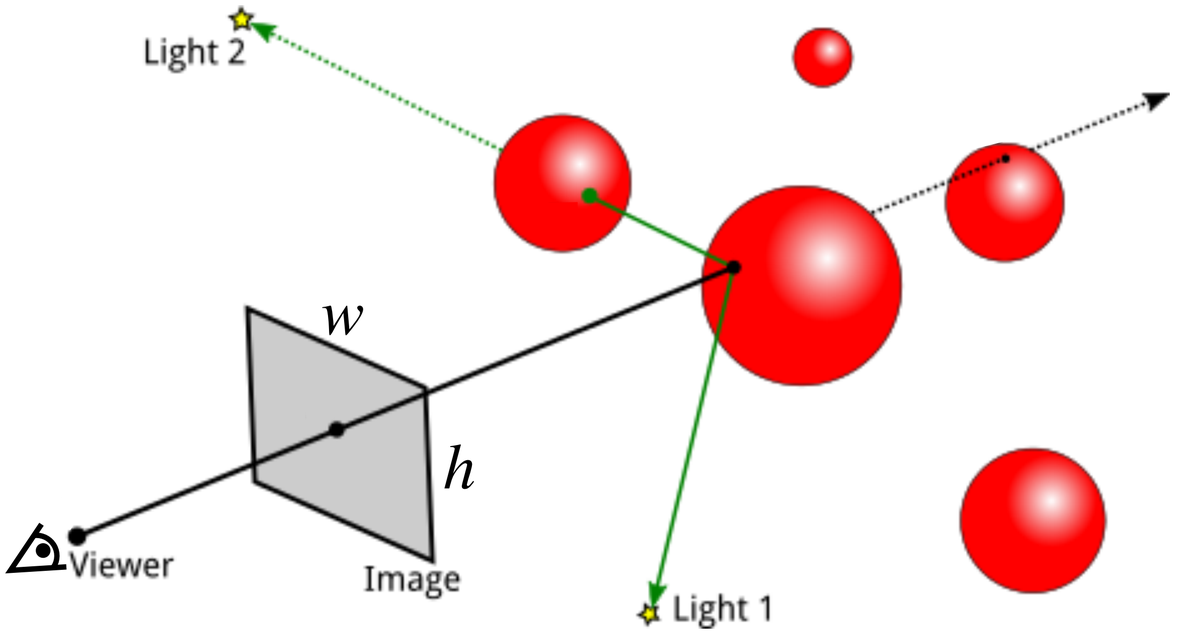
if R intersects O and is closest so far

record this intersection

shade pixel (x, y) based on nearest intersection

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

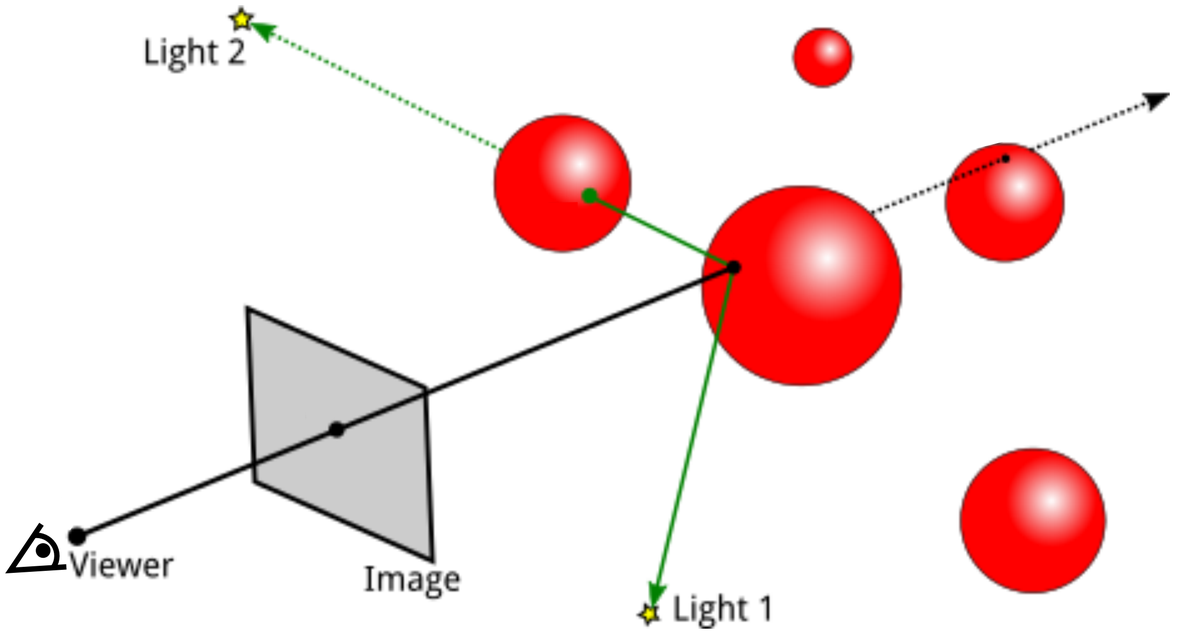
Create an eye ray



$$\mathbf{X}(d) = d * \mathbf{R}_1^{C2W}\mathbf{K}_1^{-1}\tilde{\mathbf{x}}_1 + \mathbf{O}_1$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Ray tracing



for each pixel (x, y)

create ray R from eye through (x, y)

for each object O in scene

if R intersects O and is closest so far

record this intersection

shade pixel (x, y) based on nearest intersection

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

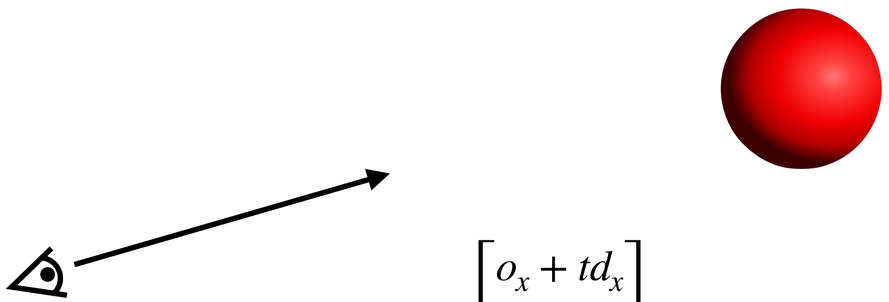
Find intersection

- How to find an intersection of a ray with a sphere?

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - r^2 = 0$$

work out algebra to solve t

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} = \begin{bmatrix} o_x + td_x \\ o_y + td_y \\ o_z + td_z \end{bmatrix}$$

- What about a ray with a polygon?

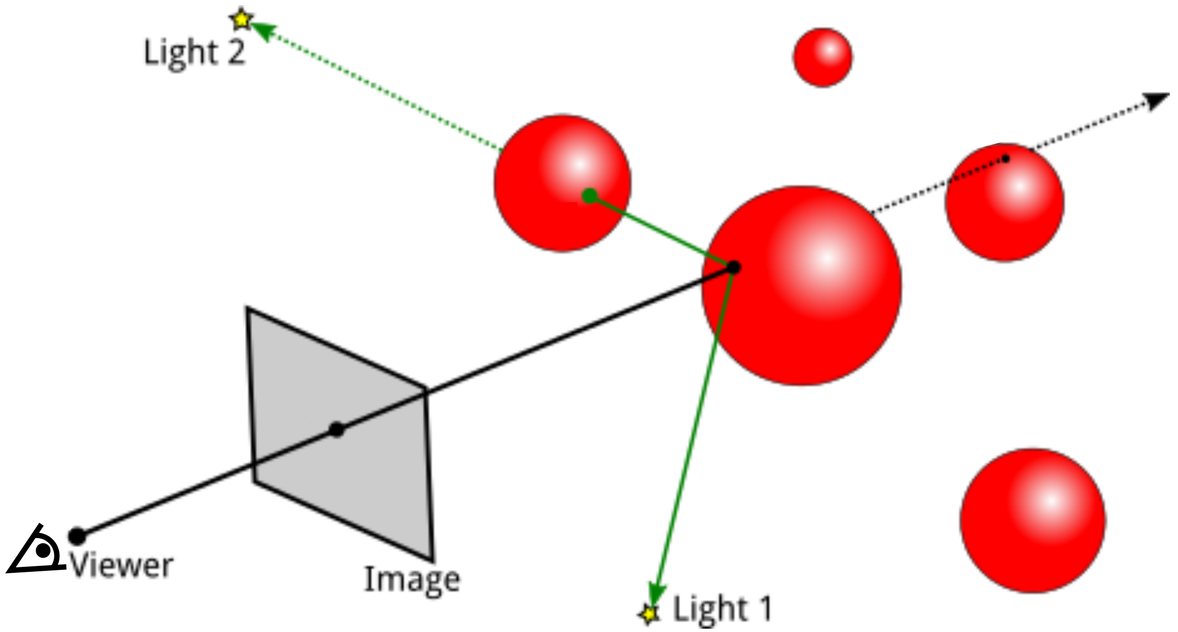
Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Find intersection

- Intersect the ray with the plane containing the polygon.
- Find out if the intersection is inside of the polygon.

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Ray tracing



for each pixel (x, y)

create ray R from eye through (x, y)

for each object O in scene

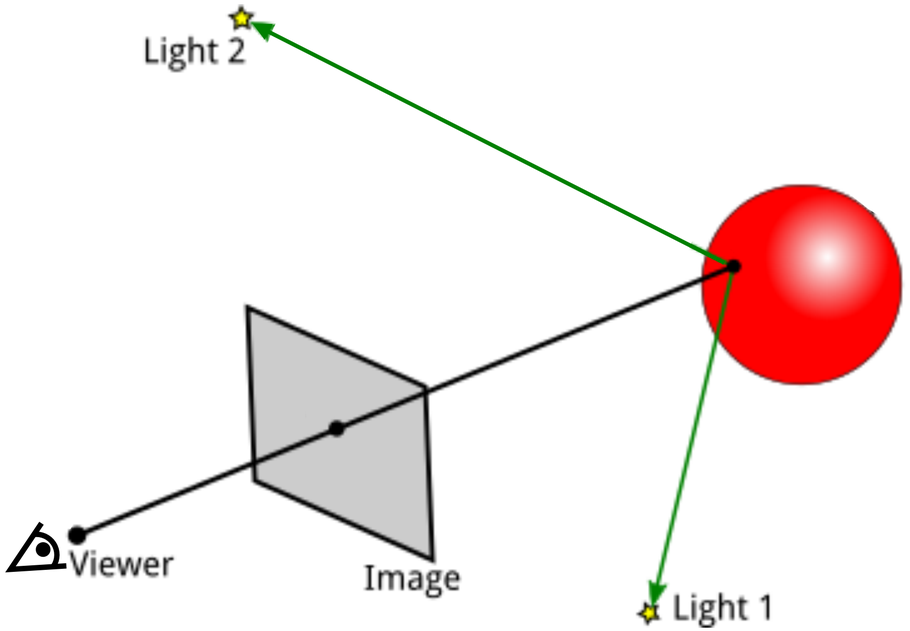
if R intersects O and is closest so far

record this intersection

shade pixel (x, y) based on nearest intersection

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Compute the color of a pixel

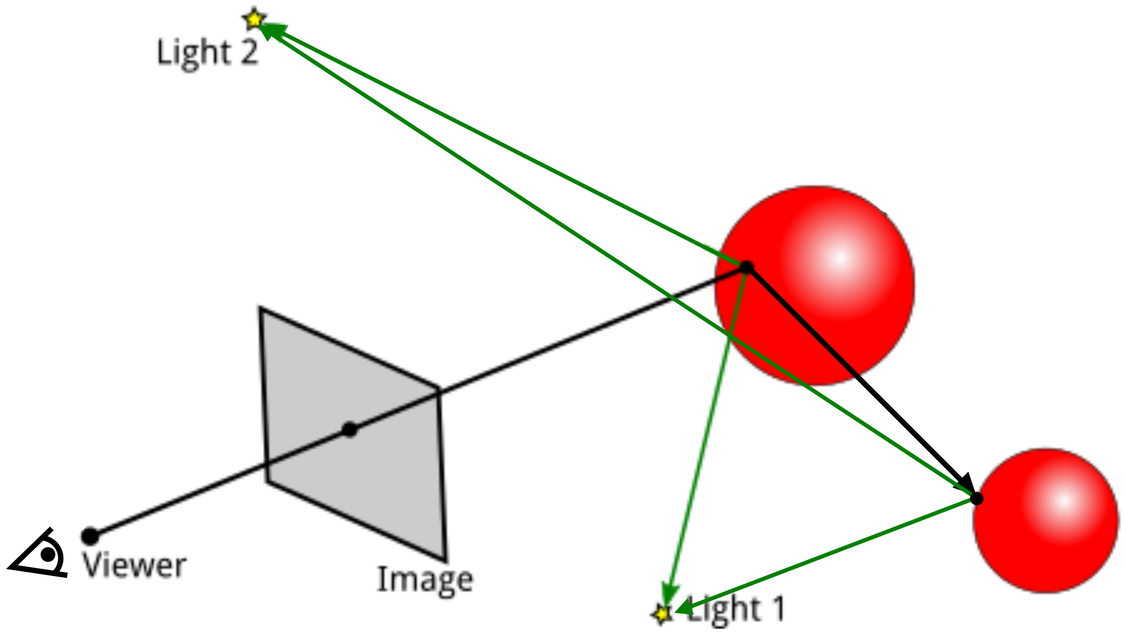


$$C = C_{ambi} + \sum_{i=1}^{2} C_{l_i} \circ C_{surf} \, (\mathbf{n} \cdot \mathbf{l}_i) + C_{spec}$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Reflection

Generate single, “most likely” direction

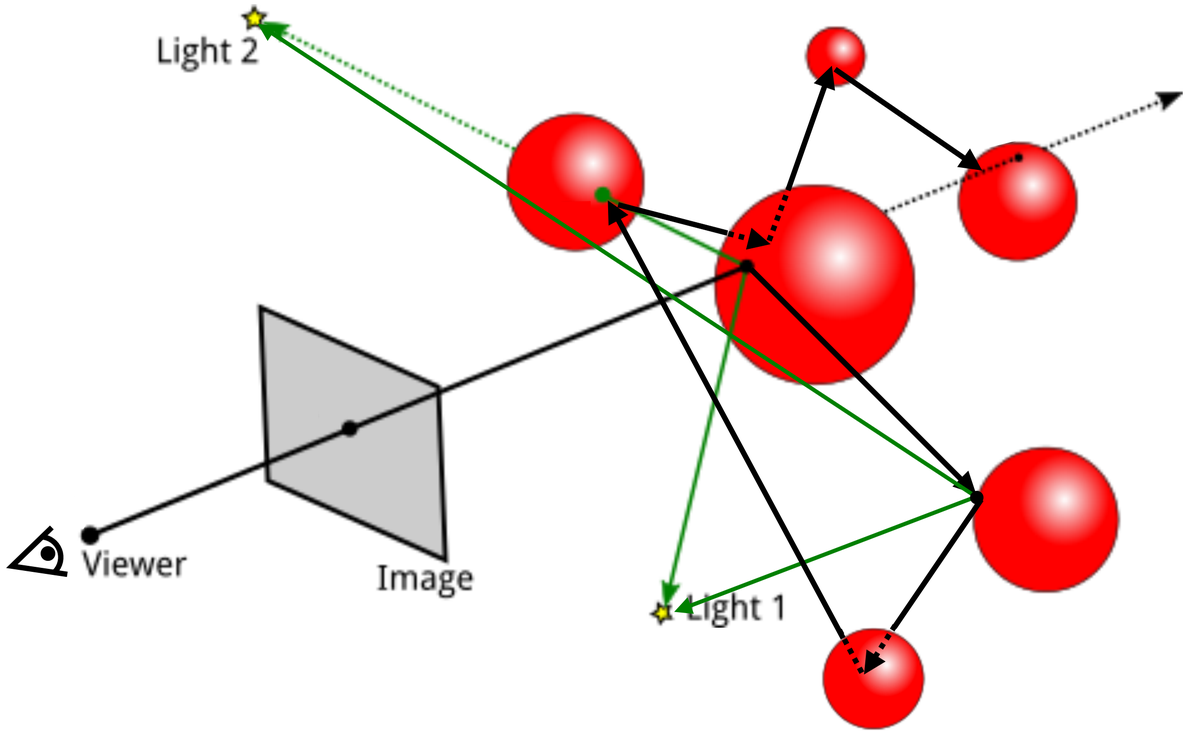


$$C = C_{ambi} + \sum_{i=1}^{2} C_{l_i} \circ C_{surf} \, (\mathbf{n} \cdot \mathbf{l}_i) + C_{spec} + k_{refl}^{(1)} \cdot C_{refl}^{(1)}$$

$$C_{refl}^{(1)} = C_{ambi} + \sum_{i=1}^{2} C_{l_i} \circ C_{surf}^{(1)} \, (\mathbf{n}^{(1)} \cdot \mathbf{l}_i^{(1)}) + C_{spec}^{(1)}$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Recursive reflection



$$C = C_{ambi} + \sum_{i=1}^{2} C_{l_i} \circ C_{surf}\,(\mathbf{n} \cdot \mathbf{l}_i) + C_{spec} + k_{refl}^{(1)} \cdot C_{refl}^{(1)}$$

$$C_{refl}^{(1)} = C_{ambi} + \sum_{i=1}^{2} C_{l_i} \circ C_{surf}^{(1)}\,(\mathbf{n}^{(1)} \cdot \mathbf{l}_i^{(1)}) + C_{spec}^{(1)} + k_{refl}^{(2)} \cdot C_{refl}^{(2)}$$

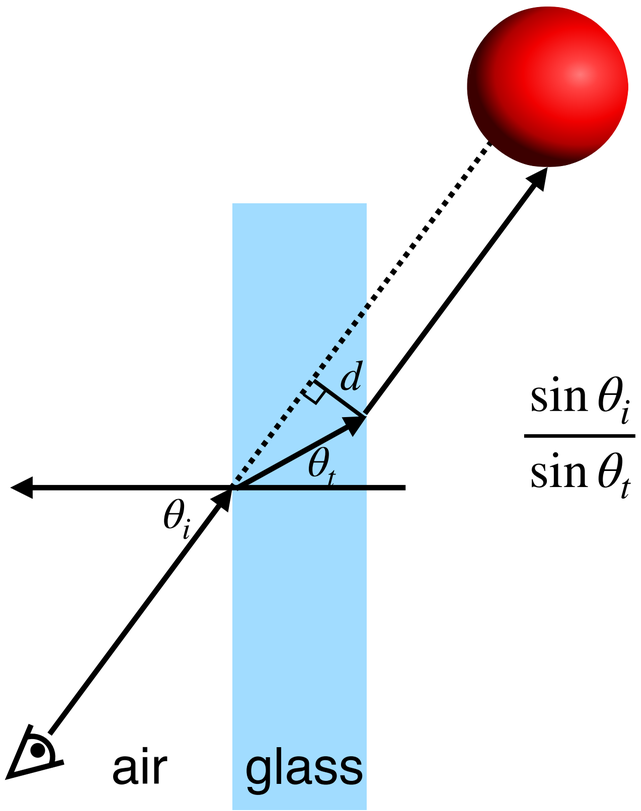
$$C_{refl}^{(2)} = C_{ambi} + \sum_{i=1}^{2} C_{l_i} \circ C_{surf}^{(2)}\,(\mathbf{n}^{(2)} \cdot \mathbf{l}_i^{(2)}) + C_{spec}^{(2)} + k_{refl}^{(3)} \cdot C_{refl}^{(3)}$$

… When to stop recursion?

1. Hit a non-shiny surface.
2. When contribution to final color is small.
3. Max depth is reached.

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Refraction



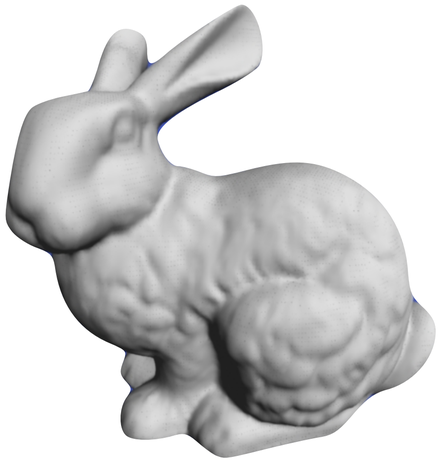
$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i}$$

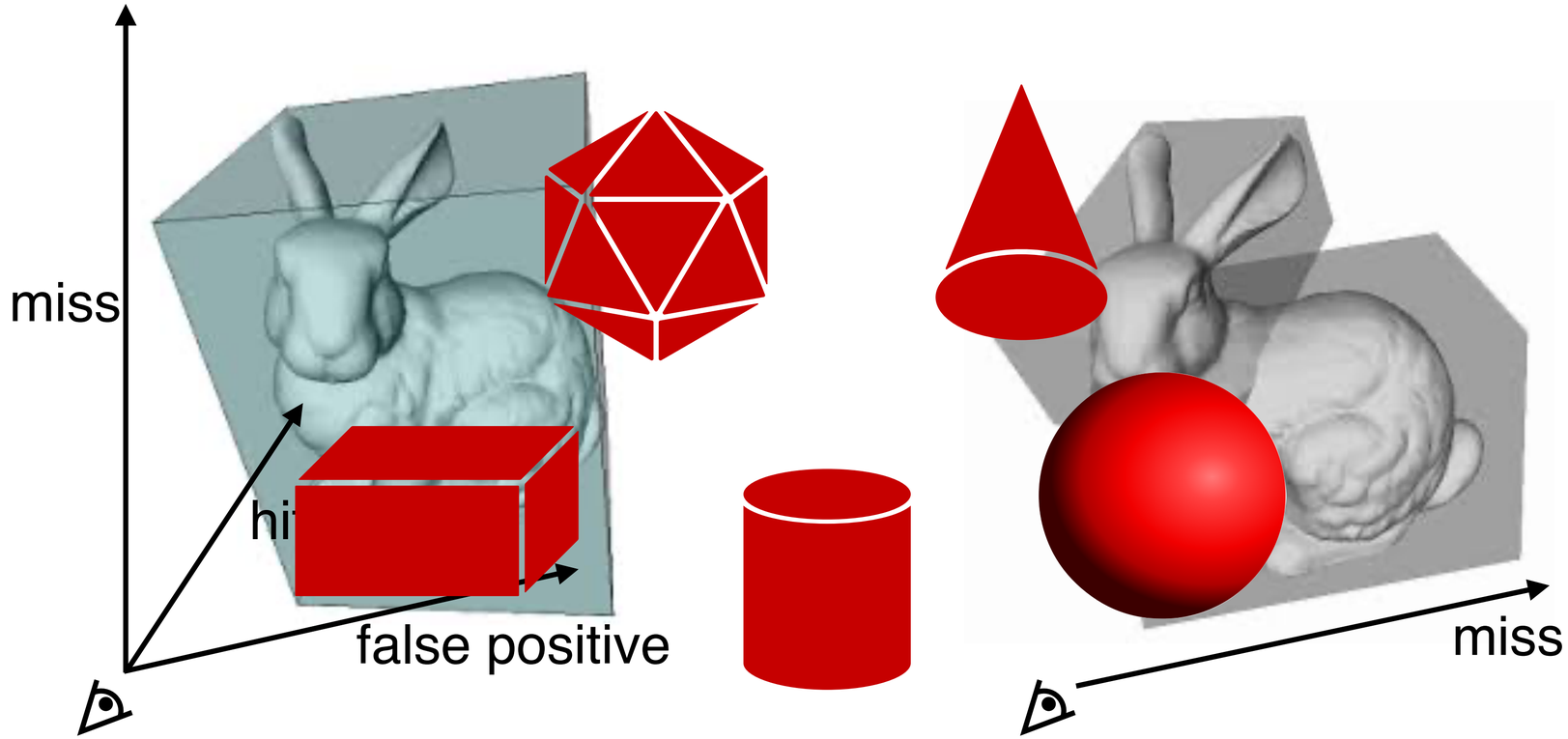
$$\text{index of refraction } (\eta) = \frac{\text{speed of light thru vacuum}}{\text{speed of light thru medium}}$$

materials	η
vacuum	1
air	1.0003
water	1.33
ice	1.309
diamond	2.47

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

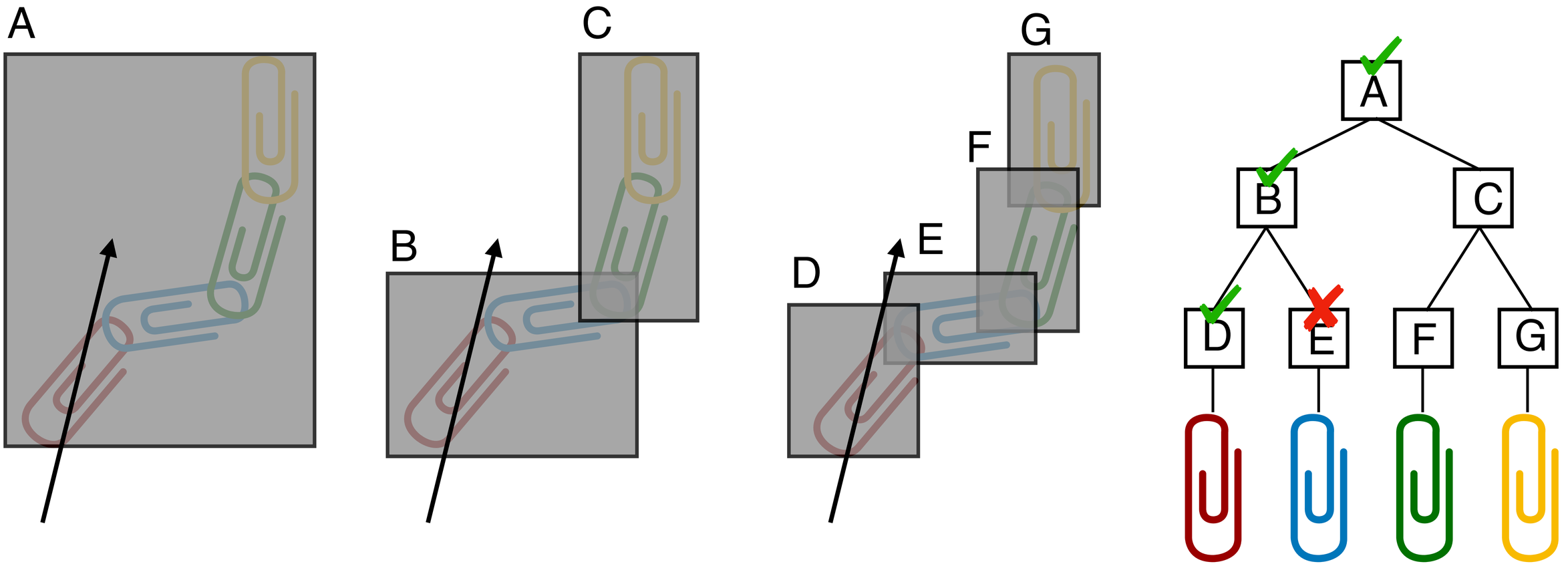
Ray tracing acceleration





Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Bounding hierarchy

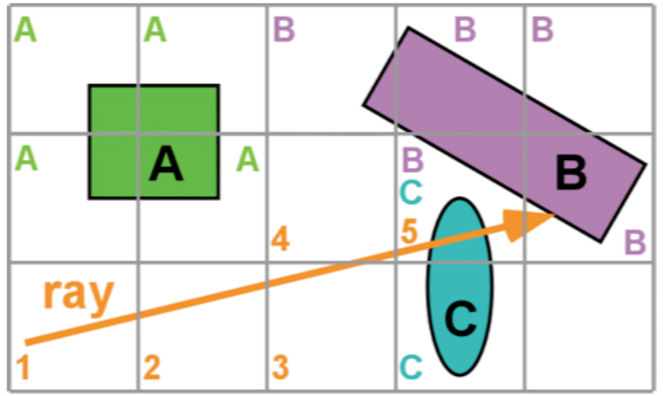


Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

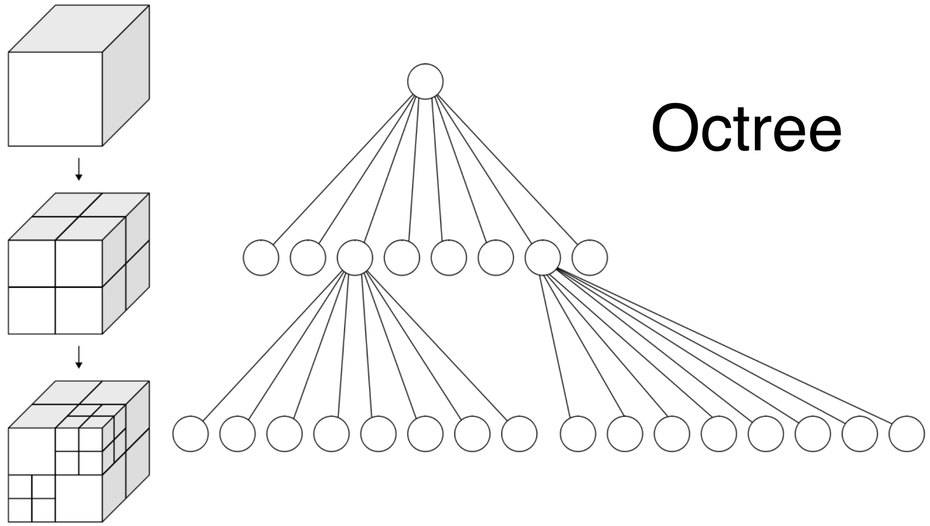
Create bounding hierarchy

- Use a tree-like structure to organize polygons into bounding hierarchy.
- For objects with similar size, **Spatial Partition** (or **Grids**) method works well.
  - Build an object list for each cell.
  - Shoot a ray thru the grid and check the object list of each cell the ray touches.
- For varying sized objects, build an **Octree** to store the objects in a binary hierarchy.

Grids

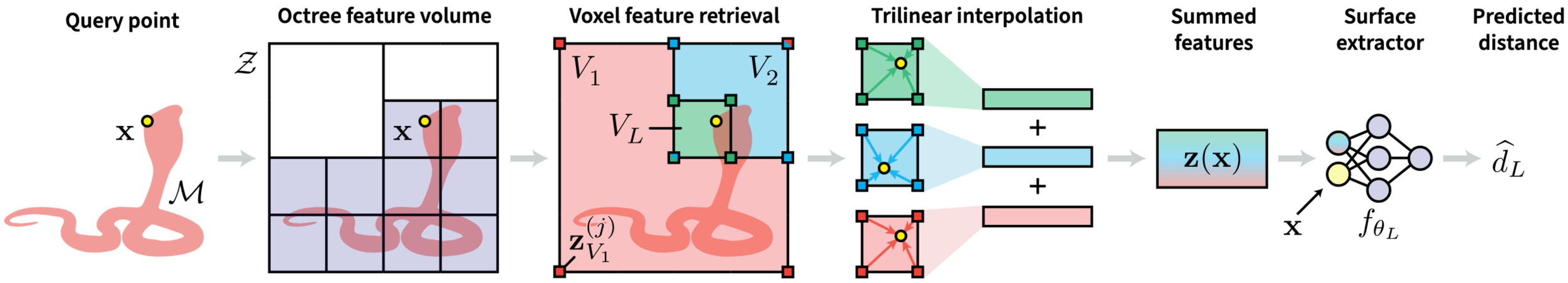


Octree



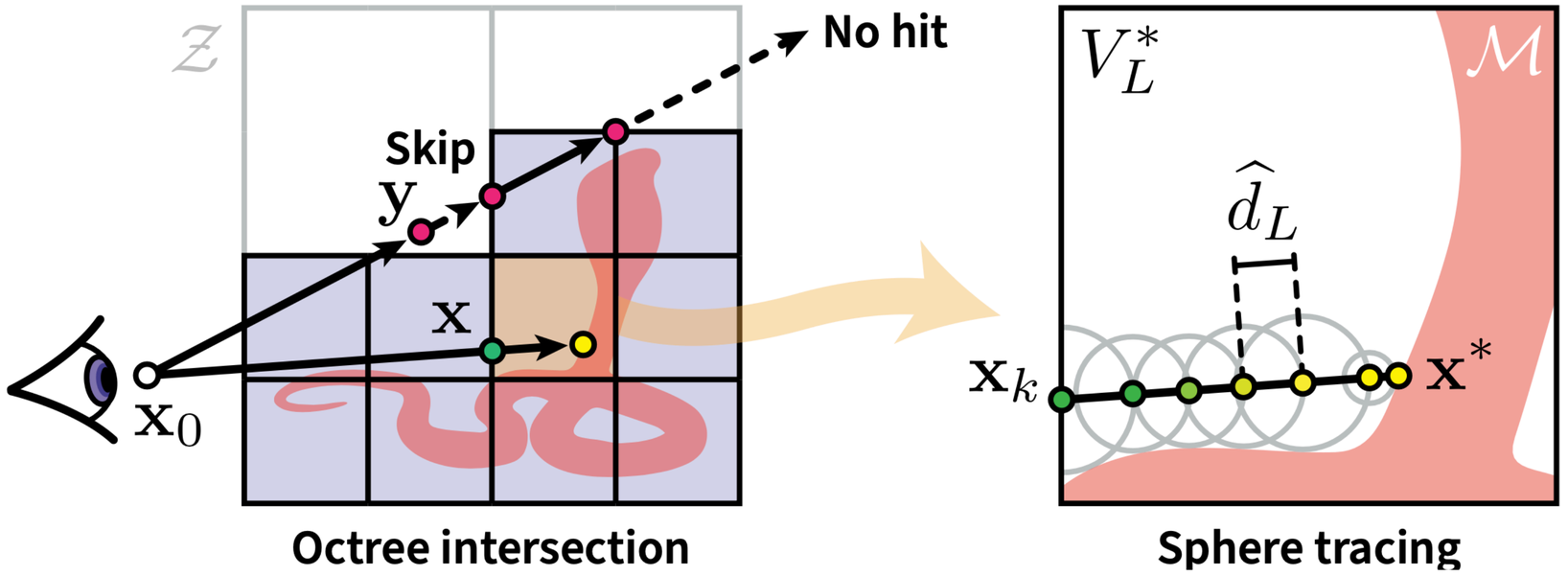
Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Neural Geometric Level of Detail

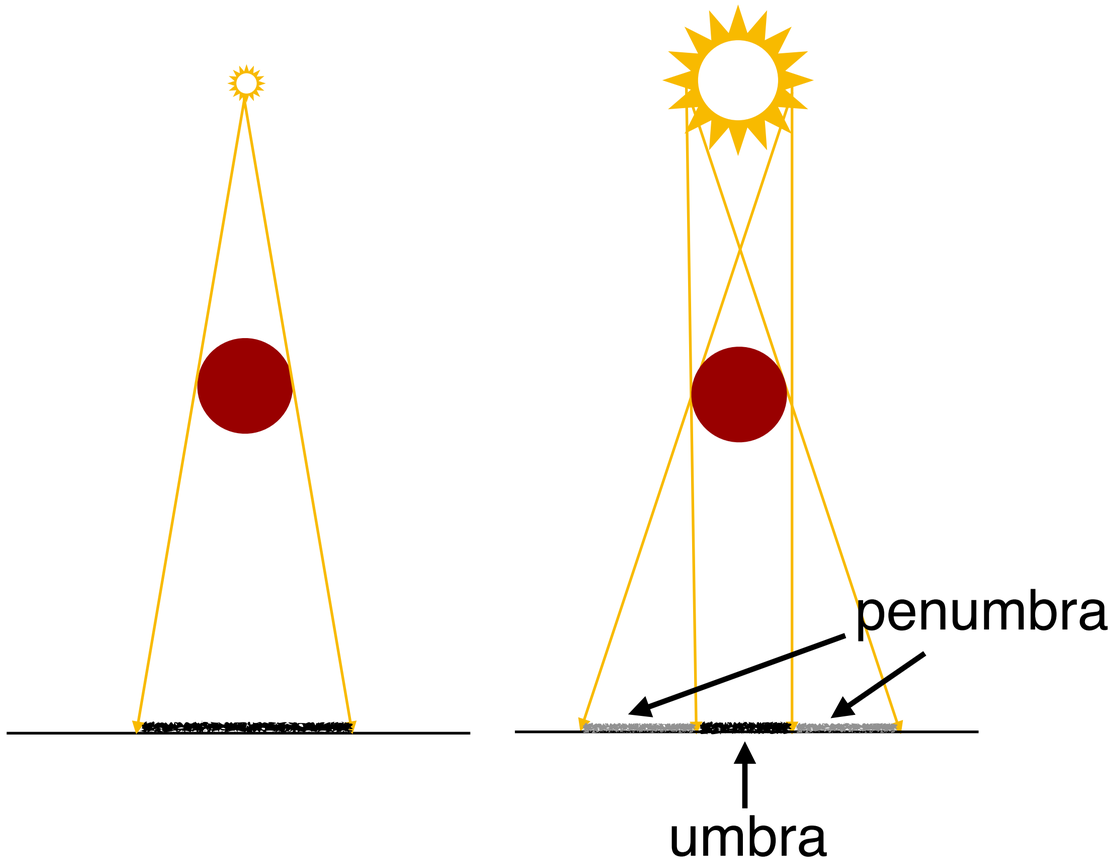


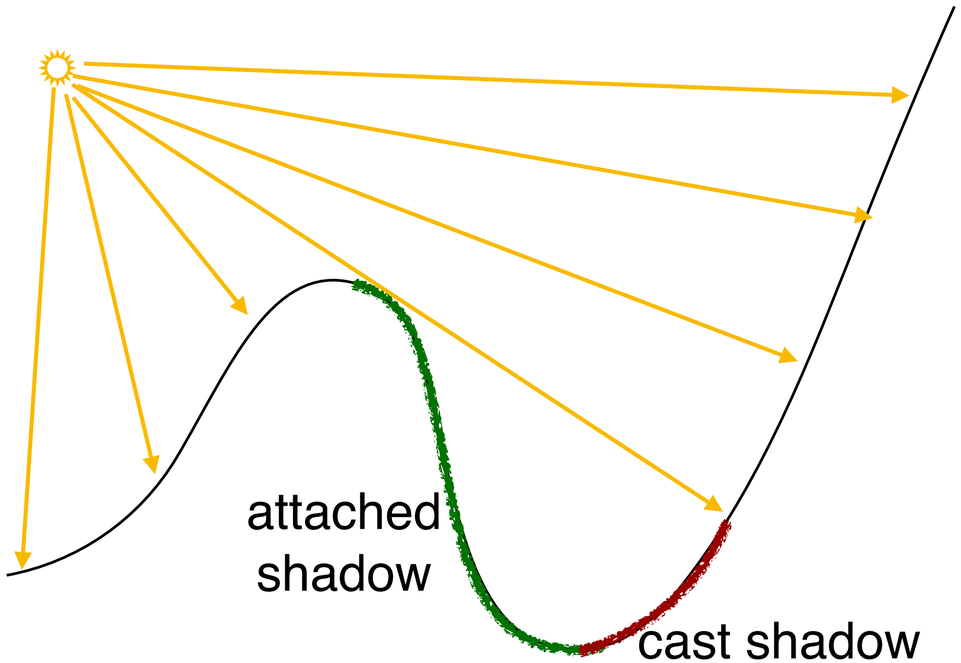
Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes, Takikawa, Litalien et al. 2021

Neural Geometric Level of Detail



Shadows

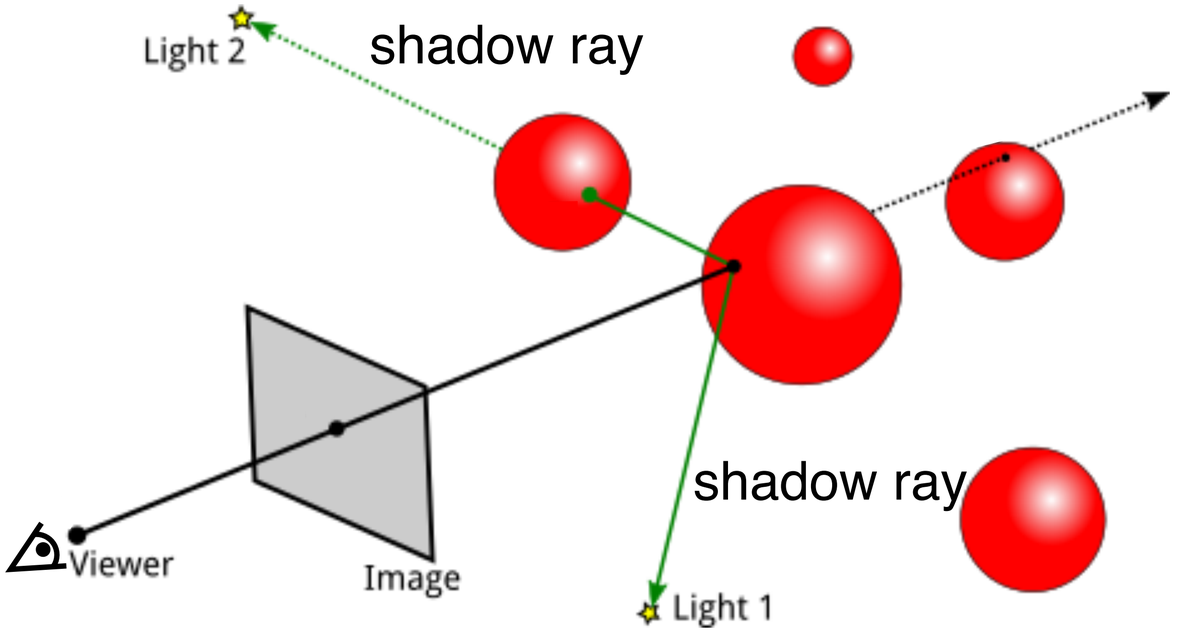




Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Shadow rays

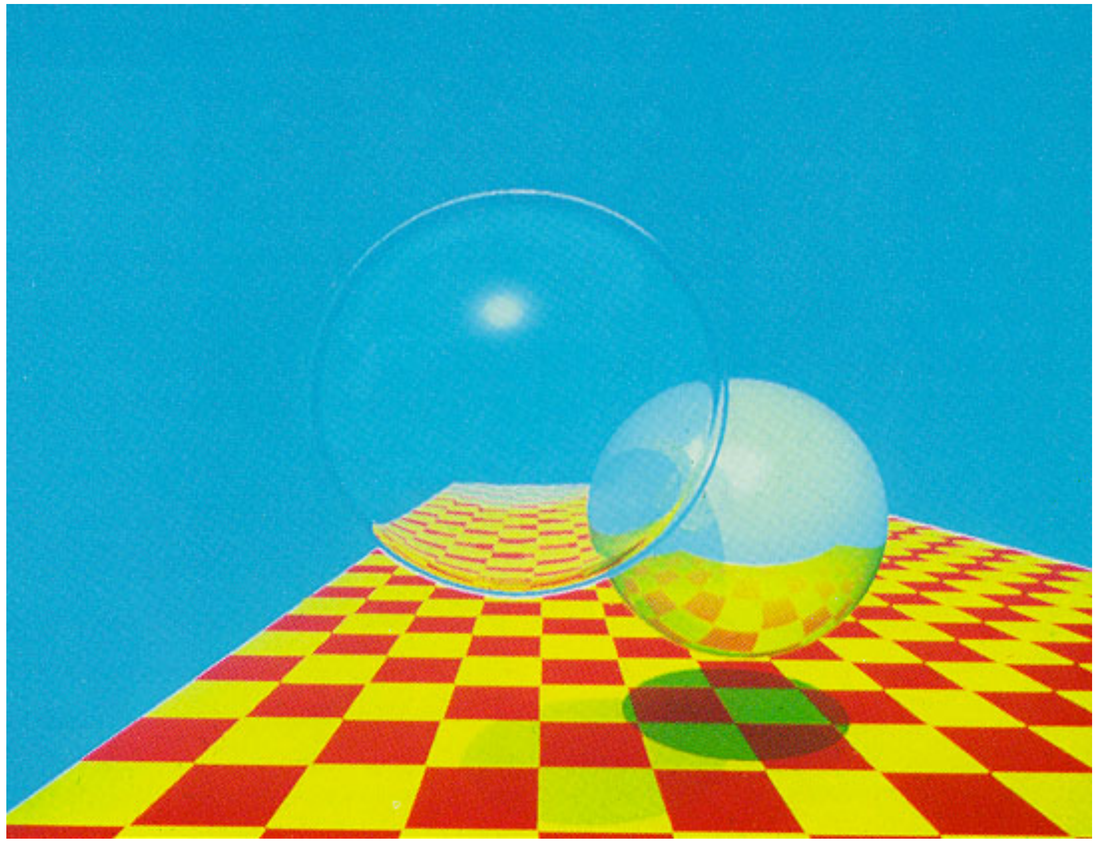
Add shadows in ray tracing

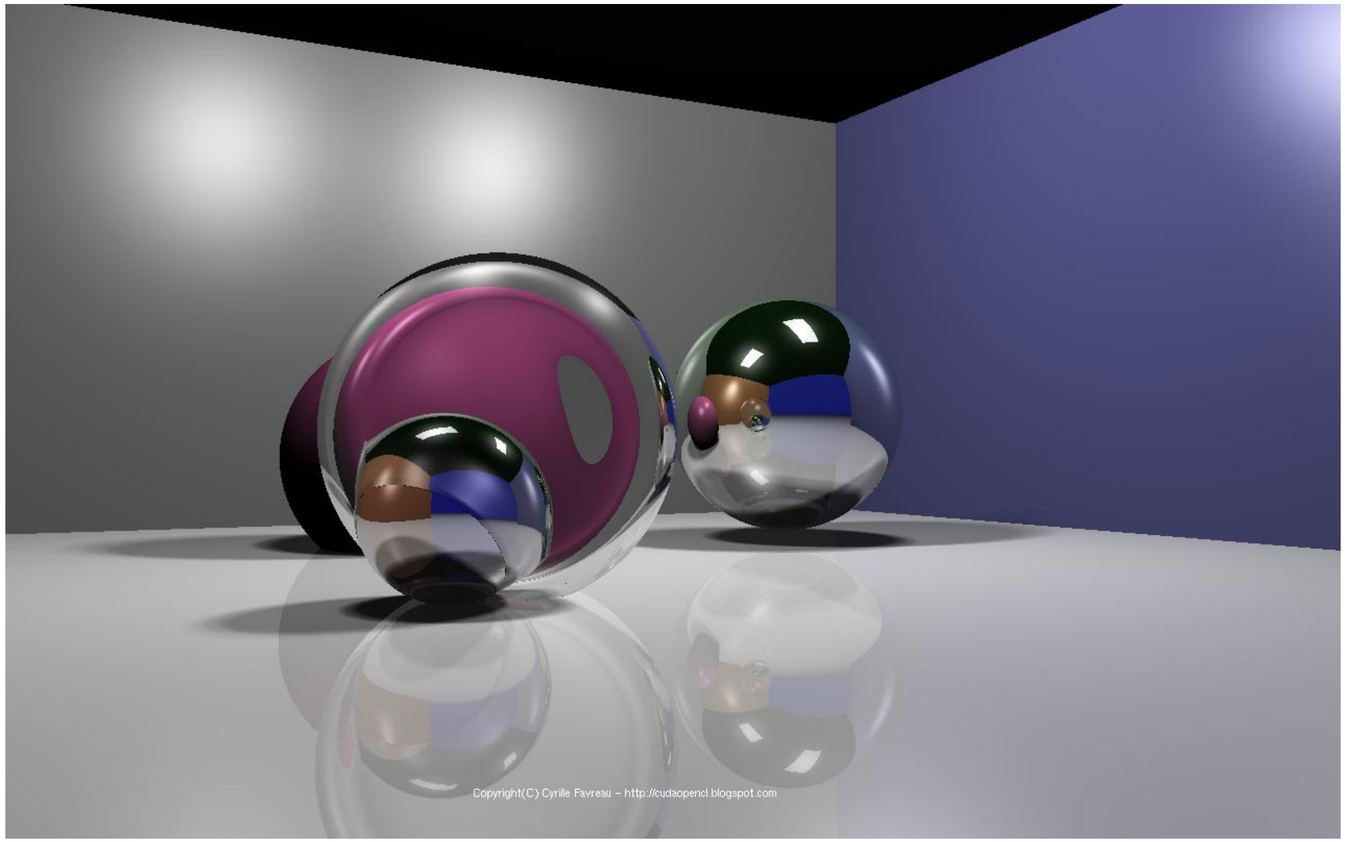


$$C = C_{ambi} + \sum_{i=1}^{2} \text{visible}(\mathbf{l}_i, \mathbf{x}) \cdot C_{l_i} \circ C_{surf} \ \max(\mathbf{n} \cdot \mathbf{l}_i, 0) + C_{spec}$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Ray tracing examples





Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Rendering equation

Rendering equation: $L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\ L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$

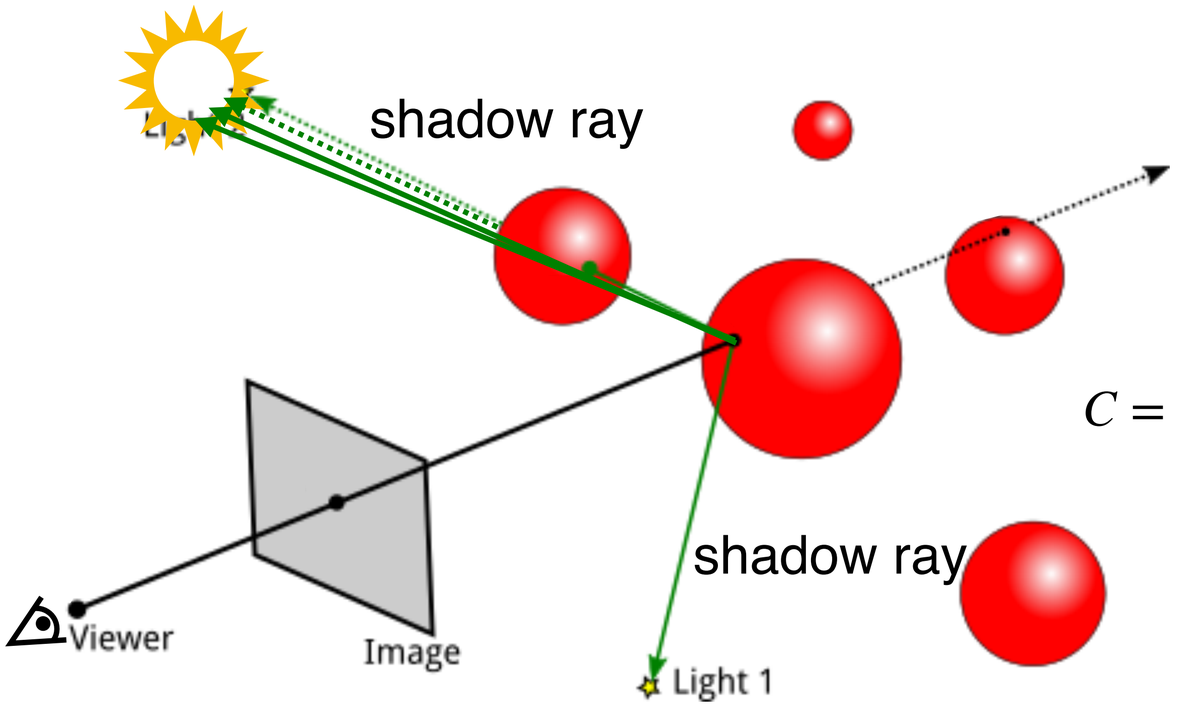
Phong model with **ray tracing**: Add reflection, refraction and hard shadows to basic Phong model.

Phong model with **rasterization**: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Distribution ray tracing

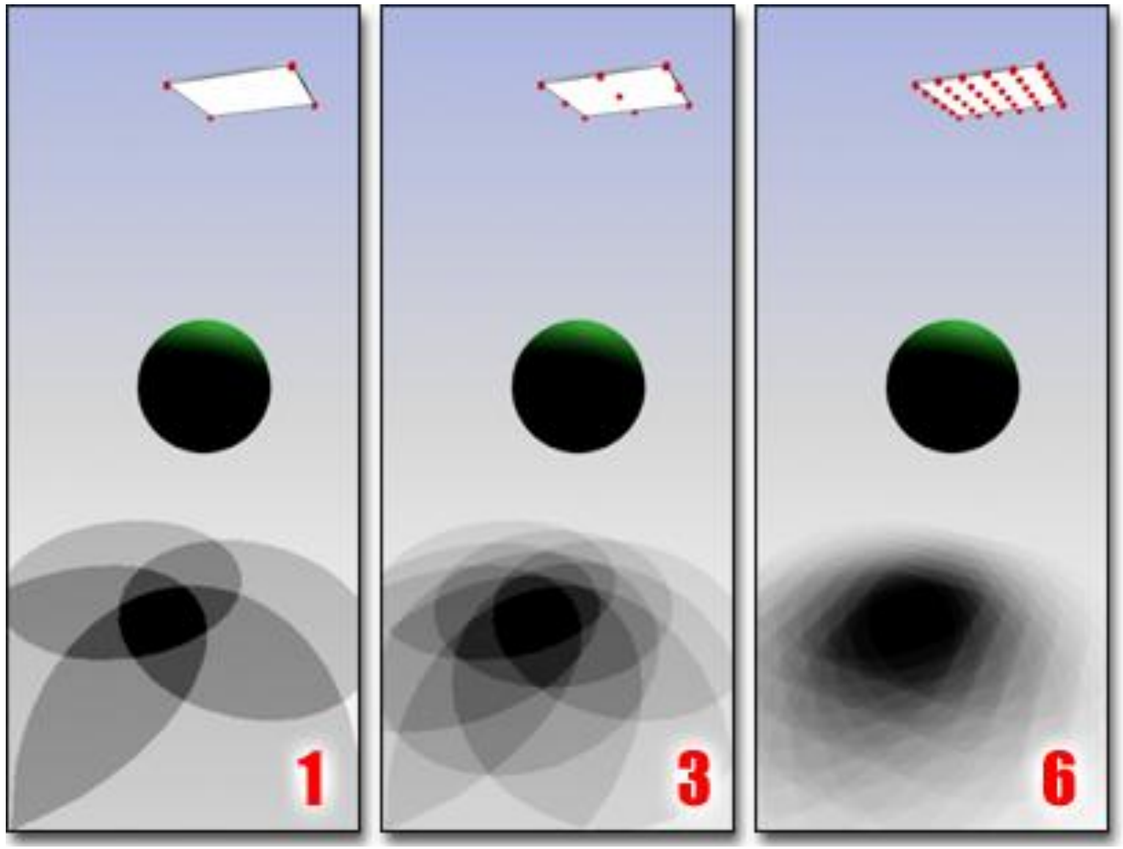
Add soft shadow effects in ray tracing

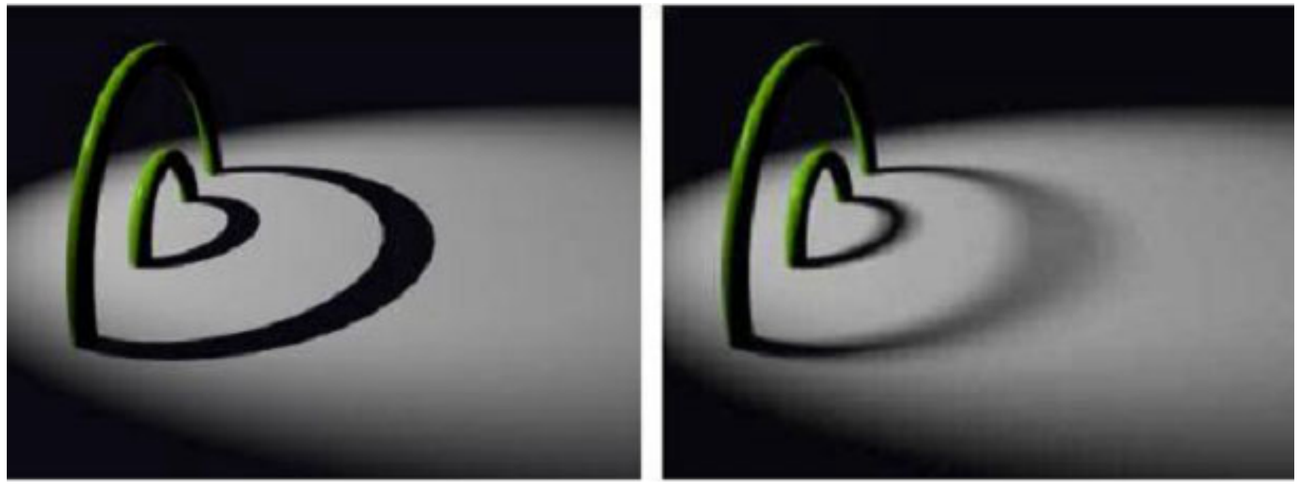


$$C = C_{ambi} + \sum_{i=1}^{2} \left(\frac{1}{M} \sum_{j=1}^{M} \text{visible}(\mathbf{l}_i^{(j)}, \mathbf{x}) \cdot C_{l_i} \circ C_{surf} \ \max(\mathbf{n} \cdot \mathbf{l}_i^{(j)}, 0) \right) + C_{spec}$$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Distribution ray tracing examples

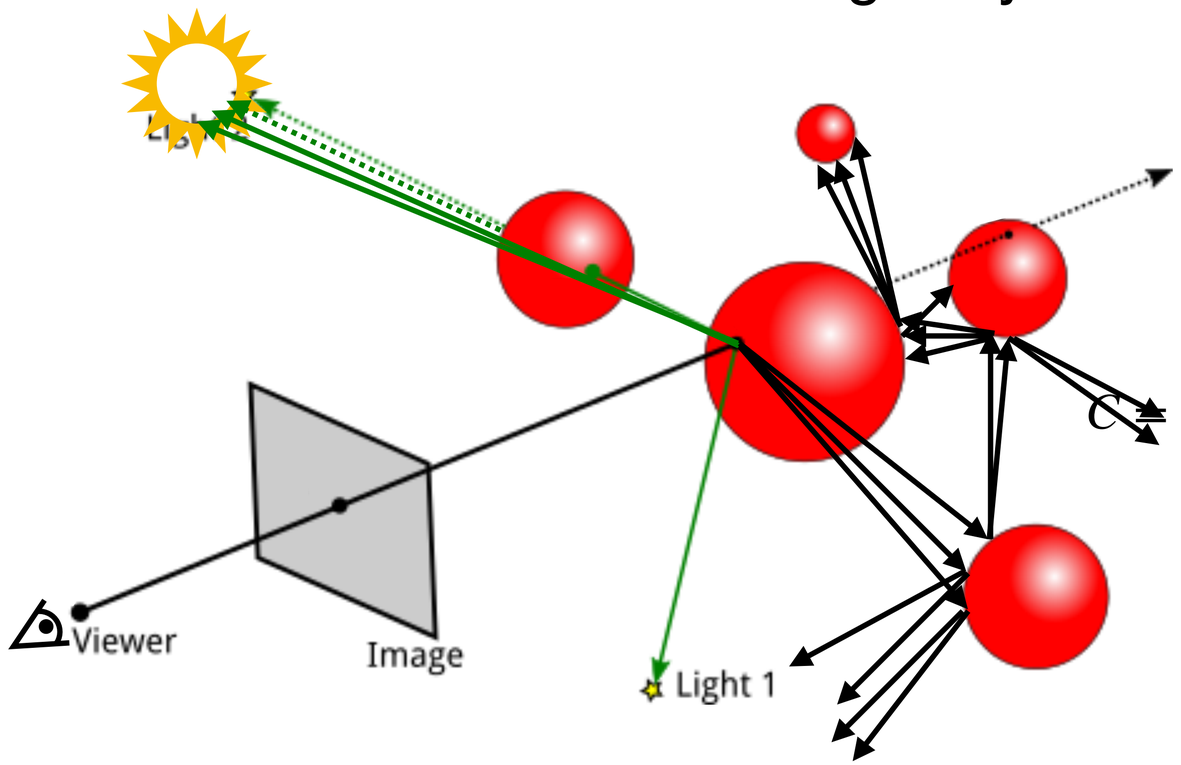




Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Distribution ray tracing

Add glossy reflection effects in ray tracing



$$C = C_{ambi} + \sum_{i=1}^{2}\left(\frac{1}{M}\sum_{j=1}^{M}\text{visible}(\mathbf{l}_i^{(j)},\mathbf{x})\cdot C_{l_i}\circ C_{surf}\ \max(\mathbf{n}\cdot\mathbf{l}_i^{(j)},0)\right) + C_{spec} + k_{refl}^{(1)}\cdot\sum_{k=1}^{P} C_{refl_p}^{(1)}$$

Exponentially expand the reflection terms.

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Rendering equation

Rendering equation: $L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\ L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$

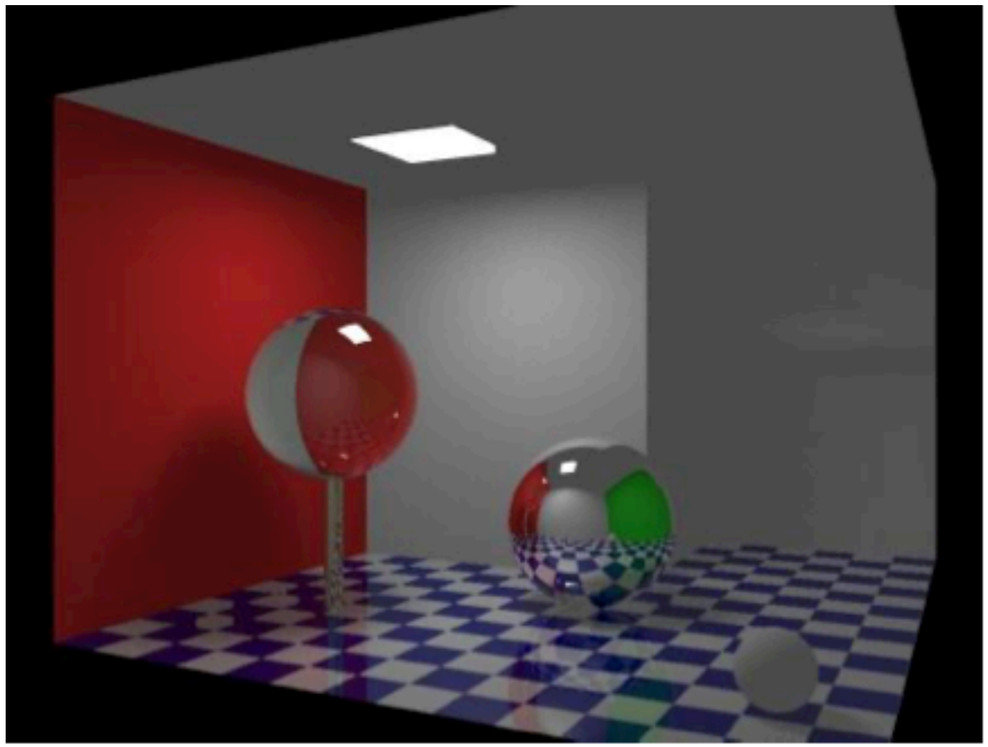
Distribution ray tracing: Can approximate soft shadows and glossy (but not perfectly reflective) surfa

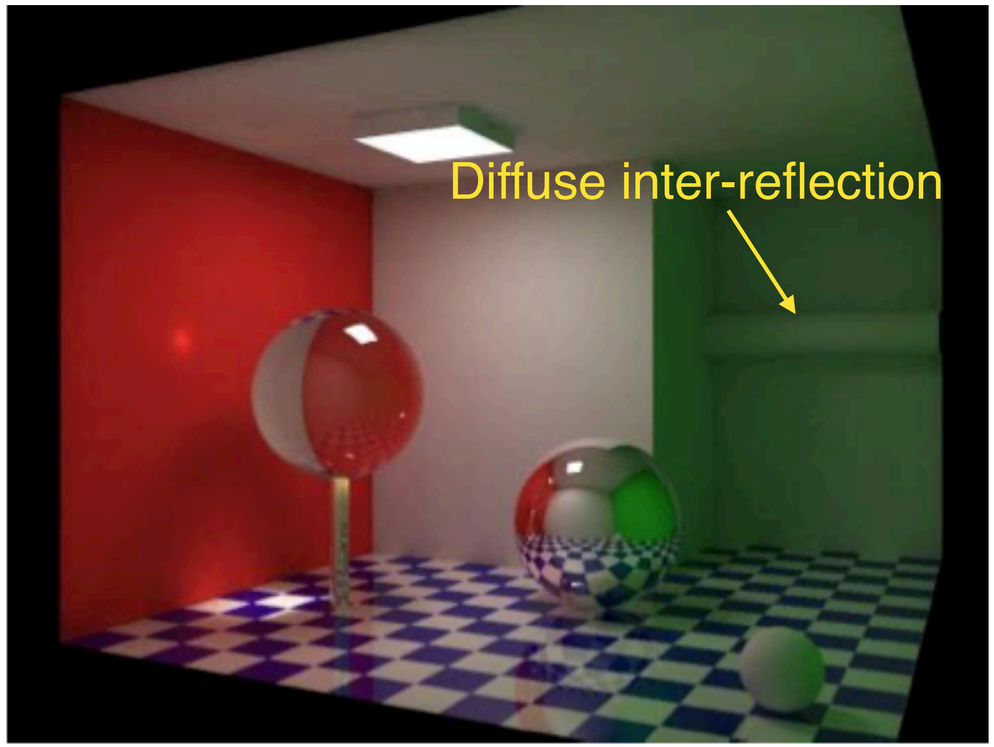
Phong model with **ray tracing**: Add reflection, refraction and hard shadows to basic Phong model.

Phong model with **rasterization**: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Beyond ray tracing





Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Rendering equation

photorealism

Rendering equation: $L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\ L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$

Path tracing: Close approximation of the solution to the rendering equation.

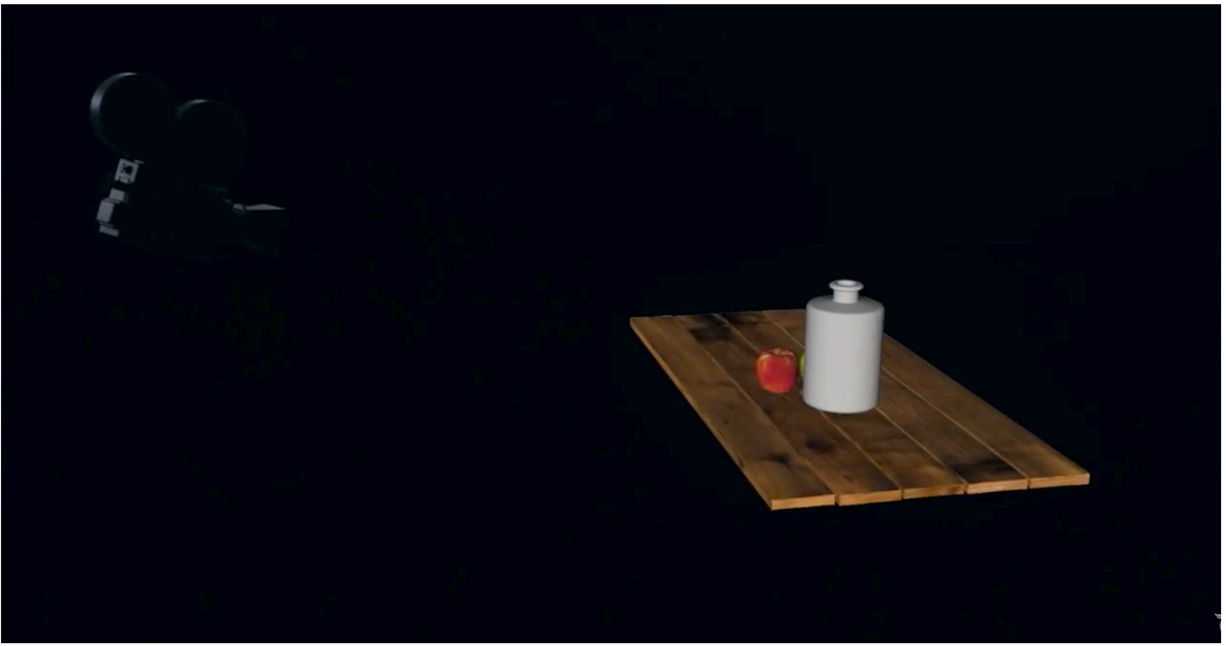
Distribution ray tracing: Can approximate soft shadows and glossy (but not perfectly reflective) surface

Phong model with ray tracing: Add reflection, refraction and hard shadows to basic Phong model.

Phong model with rasterization: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Path tracing



for each pixel *p*

create ray *r* from eye through *p*

for each sample

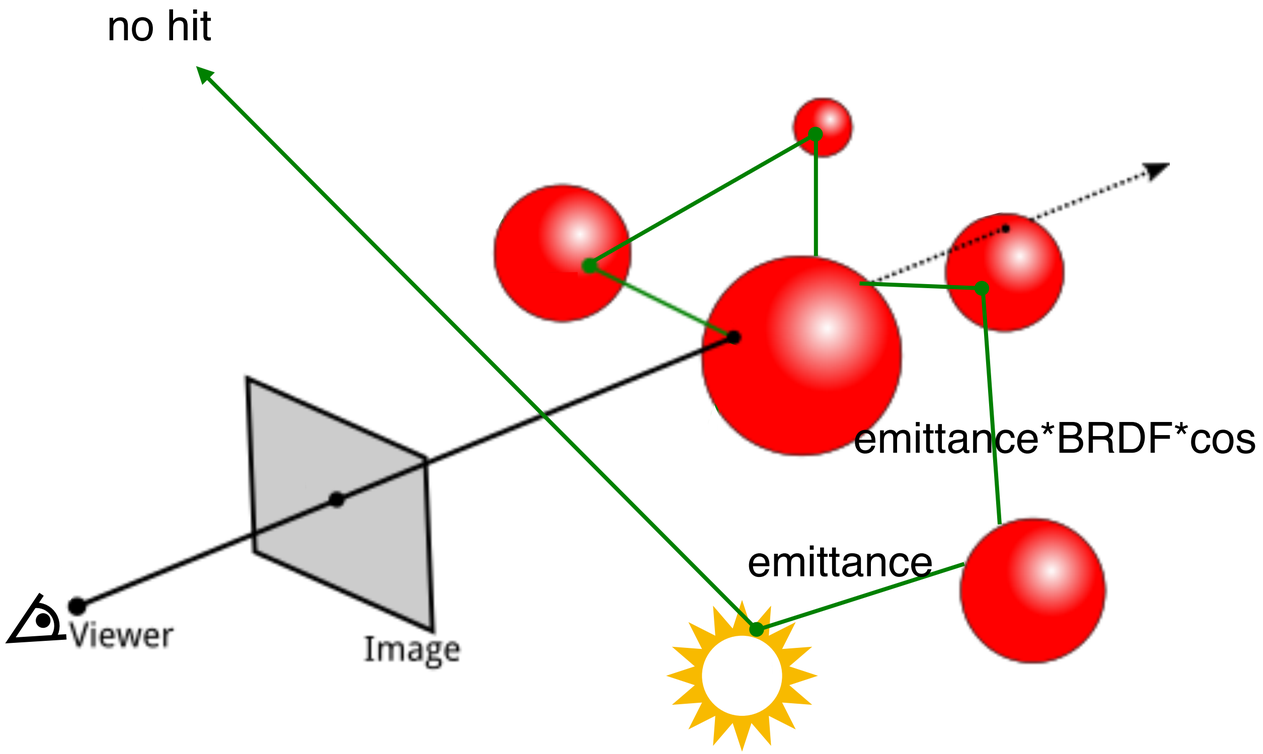
color += PathTracer(*r*)

render *p* in *color*/numSamples

- Send a ray through a pixel, find the first intersection on the surface of an object, and integrate over all the illuminance arriving to the intersection point.
- Compute illuminance by recursively projecting a ray from the surface to the scene in a bouncing path that terminates when a light source is intersected.
- The light is then sent backwards through the path and to the output pixel.

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

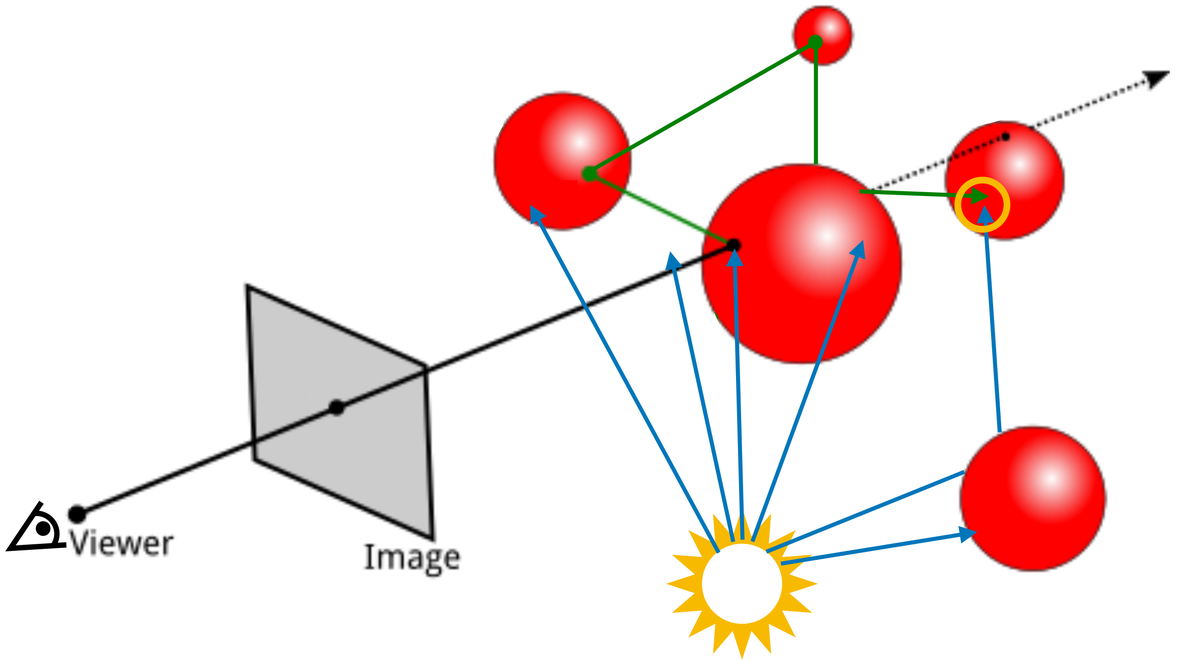
Path tracing




```
PathTracer(r)
    if r.hitNothing
        return Black
    if r.hit.material.emmitance
        return r.hit.material.emmitance
    create a new ray r' from r.hit in random direction
    light = PathTracer(r')
    cos = dot(r'.direction, r.hit.normal)
    BRDF = r.hit.material.reflectance
    return BRDF*light*cos
```

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Bidirectional path tracing (BDPT)



- Reflected light for a point can be approximated by two ways:
  - Shooting rays from the light sources into the scene. Billions of paths are created and bounced for a number of steps.
  - Gathering rays from a point on a surface. Bounce rays from the surface to the scene and terminate them at light sources.
- Bidirectional Path Tracing combines both **Shooting** and **Gathering** to obtain faster convergence:
  - Shooting and gathering paths are traced independently.
  - The head of the shooting path is connected to the tail of the gathering path.
  - Attenuate light at every bounce and back out into the pixel.

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu


2spp


2spp denoised


4spp


4spp denoised


8spp


8spp denoised


50spp


50spp denoised


1000spp


50spp denoised
1000spp denoised


12000spp


12000spp denoised


50spp denoised
12000spp

Summary

Rendering equation: $L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\ L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\ (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$

Path tracing: Close approximation of the solution to the rendering equation.

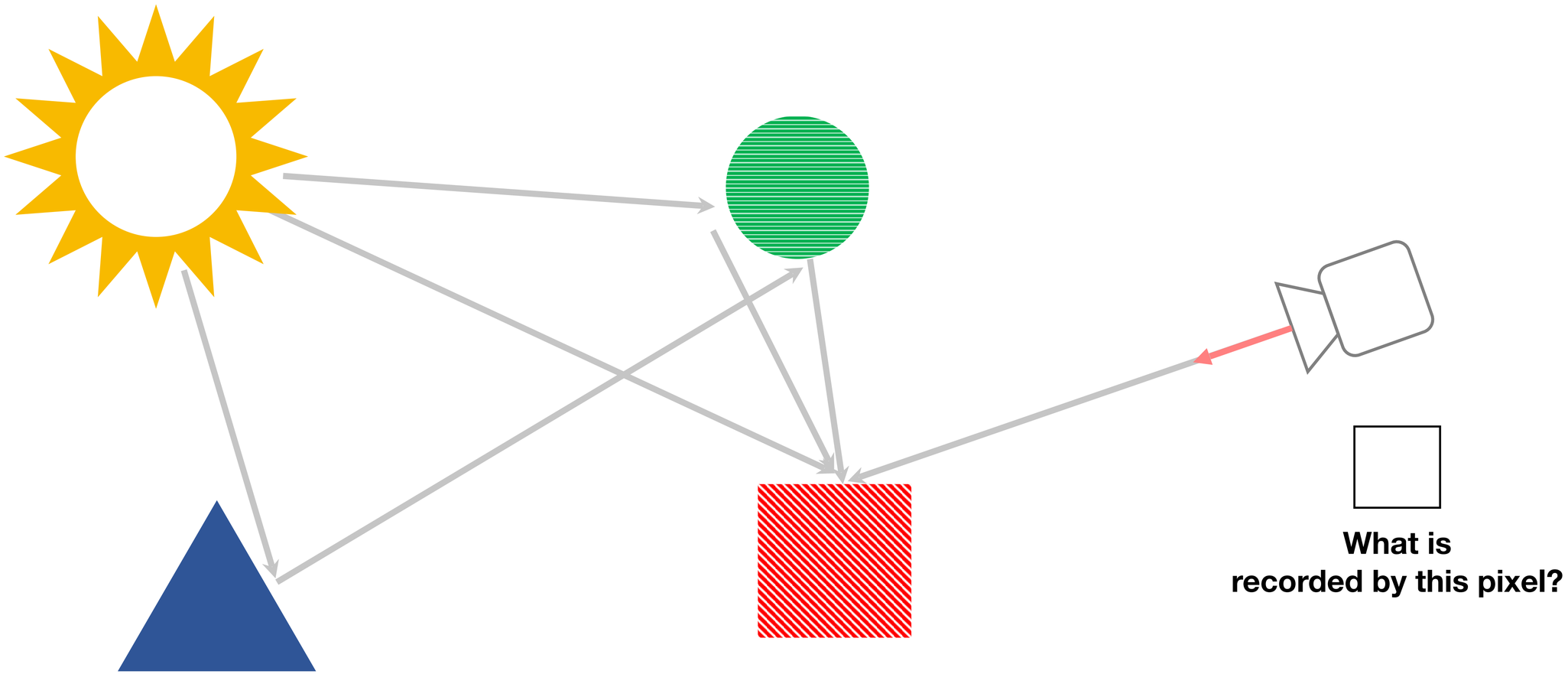
Distribution ray tracing: Can approximate soft shadows and glossy (but not perfectly reflective) surface

Phong model with ray tracing: Add reflection, refraction and hard shadows to basic Phong model.

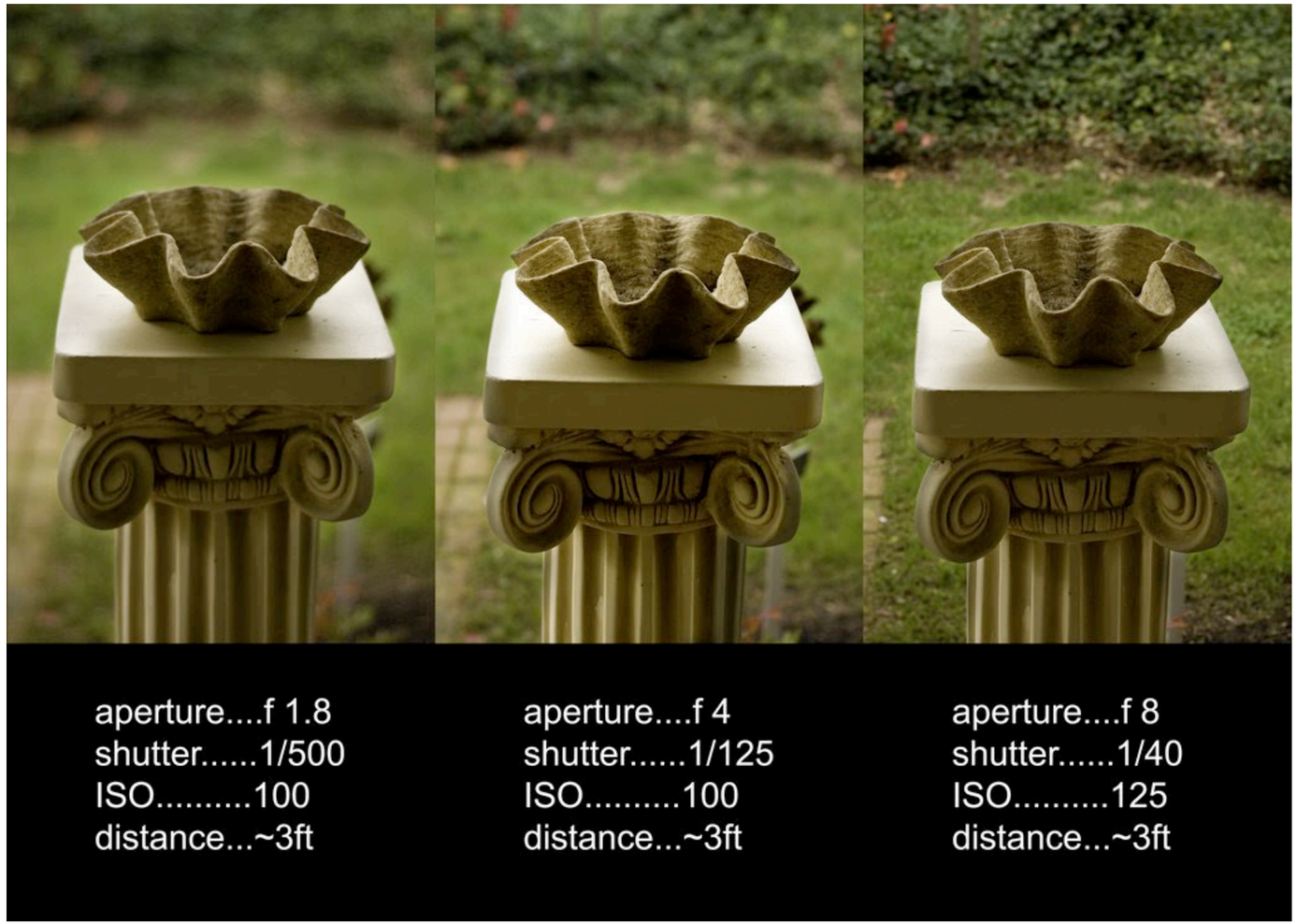
Phong model with rasterization: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^{N} C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Other effects to be aware of



Depth-of-Field



http://photographywisdom.com

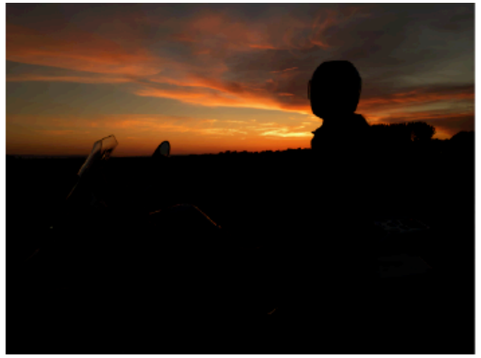
Slide credit: Gordon Wetzstein, Stanford EE367 / CS448I: Computational Imaging

Important Imaging Effects: Dynamic Range

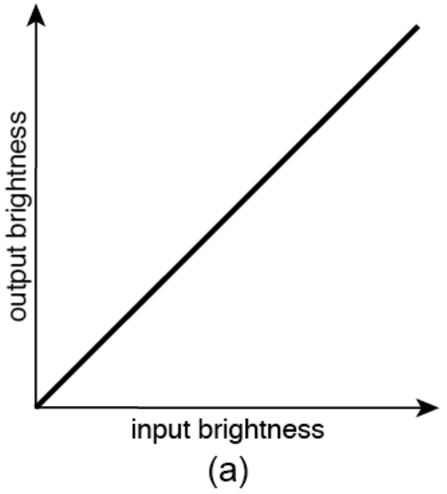
Wikipedia

Slide credit: Gordon Wetzstein, Stanford EE367 / CS448I: Computational Imaging

Important Imaging Effects: Tone Mapping

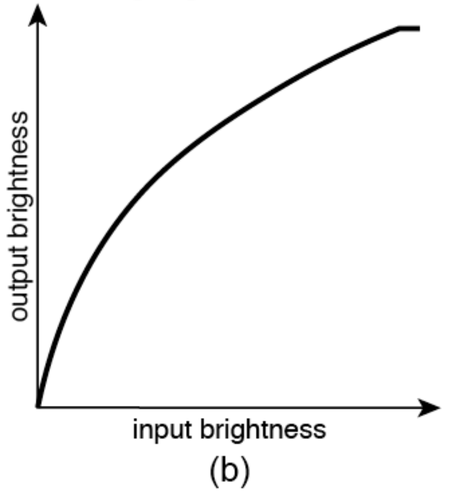


Linear RGB image



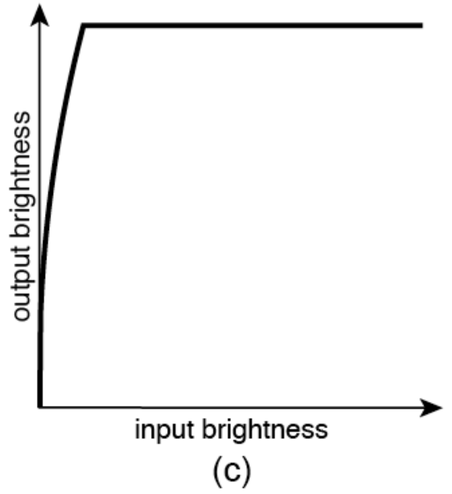
(a)

Highlights visible



(b)

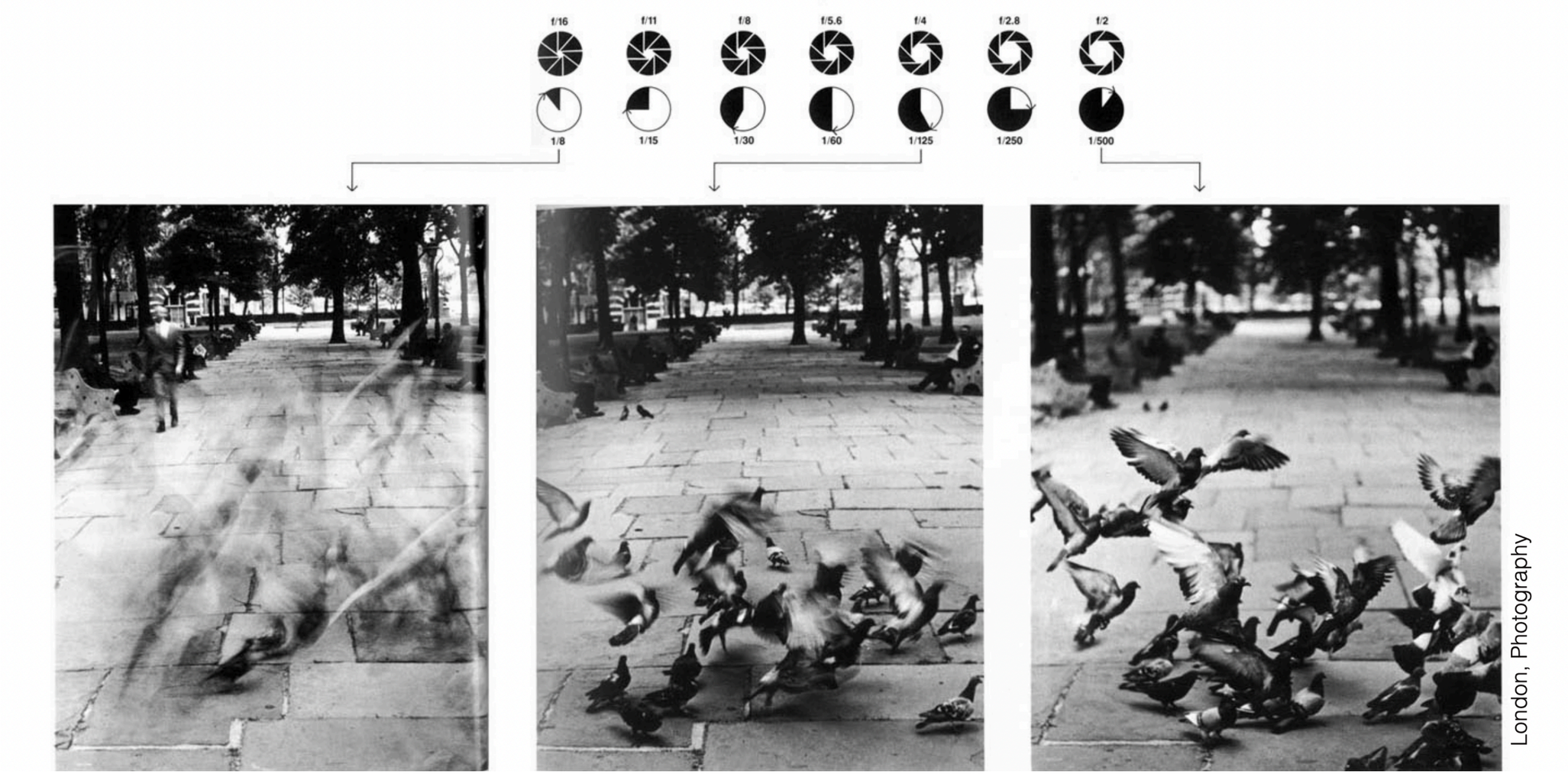
Shadows visible



(c)

https://ai.googleblog.com/2020/08/live-hdr-and-dual-exposure-controls-on.html

Important Imaging Effects: Motion Blur



Slide credit: Gordon Wetzstein, Stanford EE367 / CS448I: Computational Imaging

Today: Light Transport & High-level of Physics-based Rendering

3D Scene

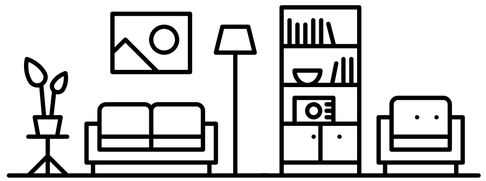


Image Formation

Inference

Neural Scene Representation

Downstream Tasks

Why?	In order to reason about the 3D world from images, we need to understand how 3D properties such as materials, lighting, etc. relate to the measurements observed by a camera.
What you'll learn.	The rendering equation, simulating light transport via multi-bounce ray-tracing, bidirectional radiance distribution functions, rasterization, rendering.