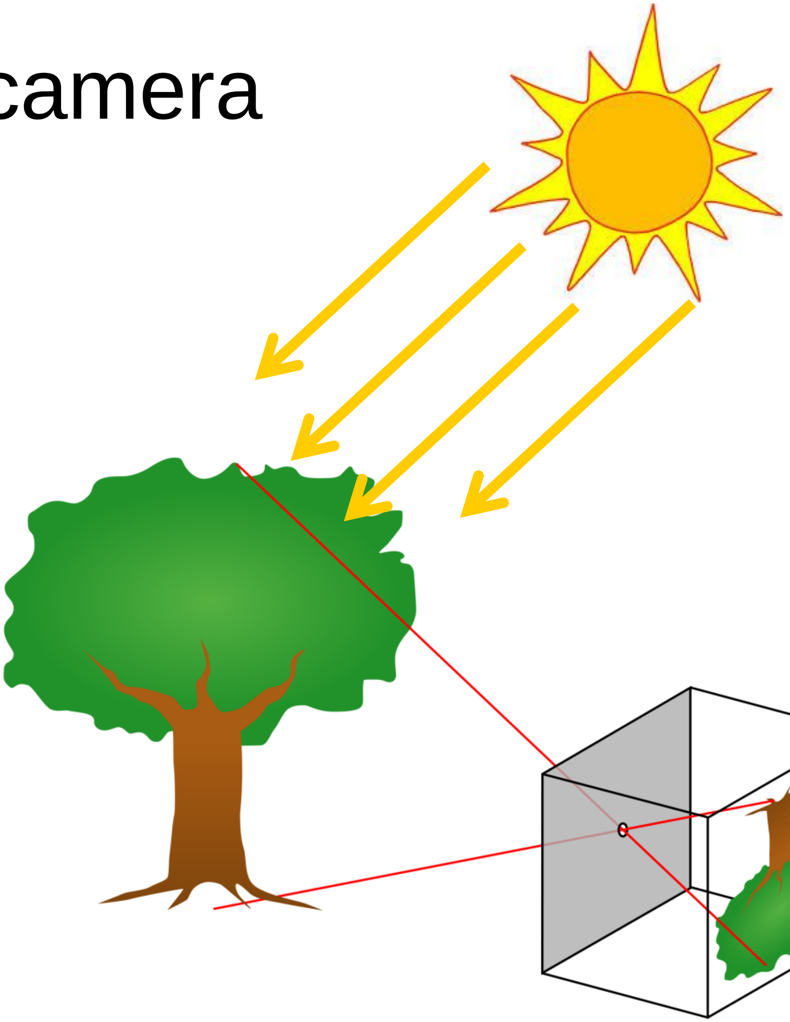
Basic Ray Tracing

Rendering: Reality

Eye acts as pinhole camera

Photons from light
hit objects

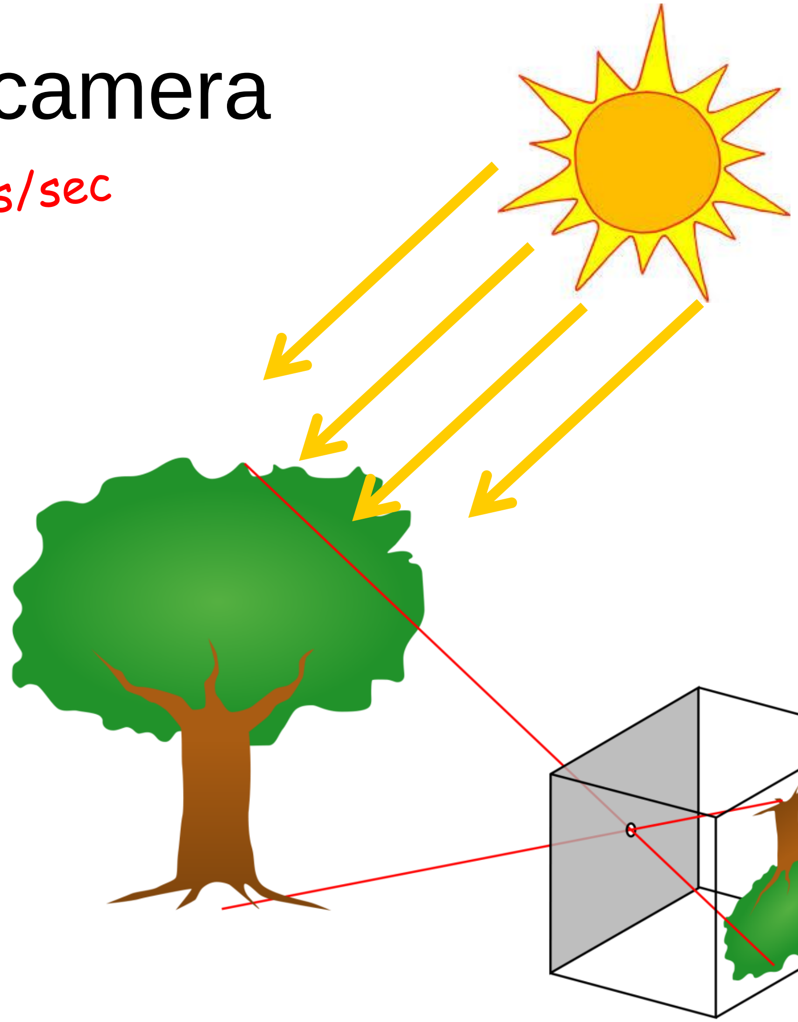


Rendering: Reality

Eye acts as pinhole camera

one lightbulb = 10^{19} photons/sec

Photons from light
hit objects



Rendering: Reality

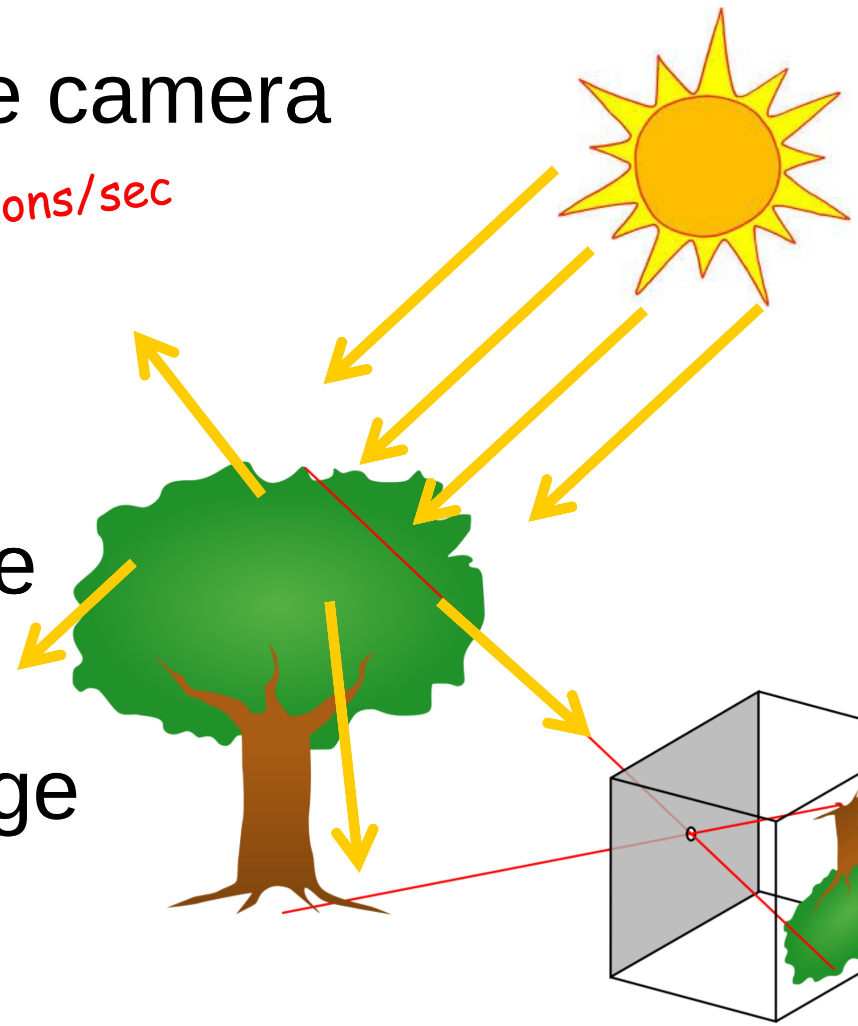
Eye acts as pinhole camera

one lightbulb = 10^{19} photons/sec

Photons from light
hit objects

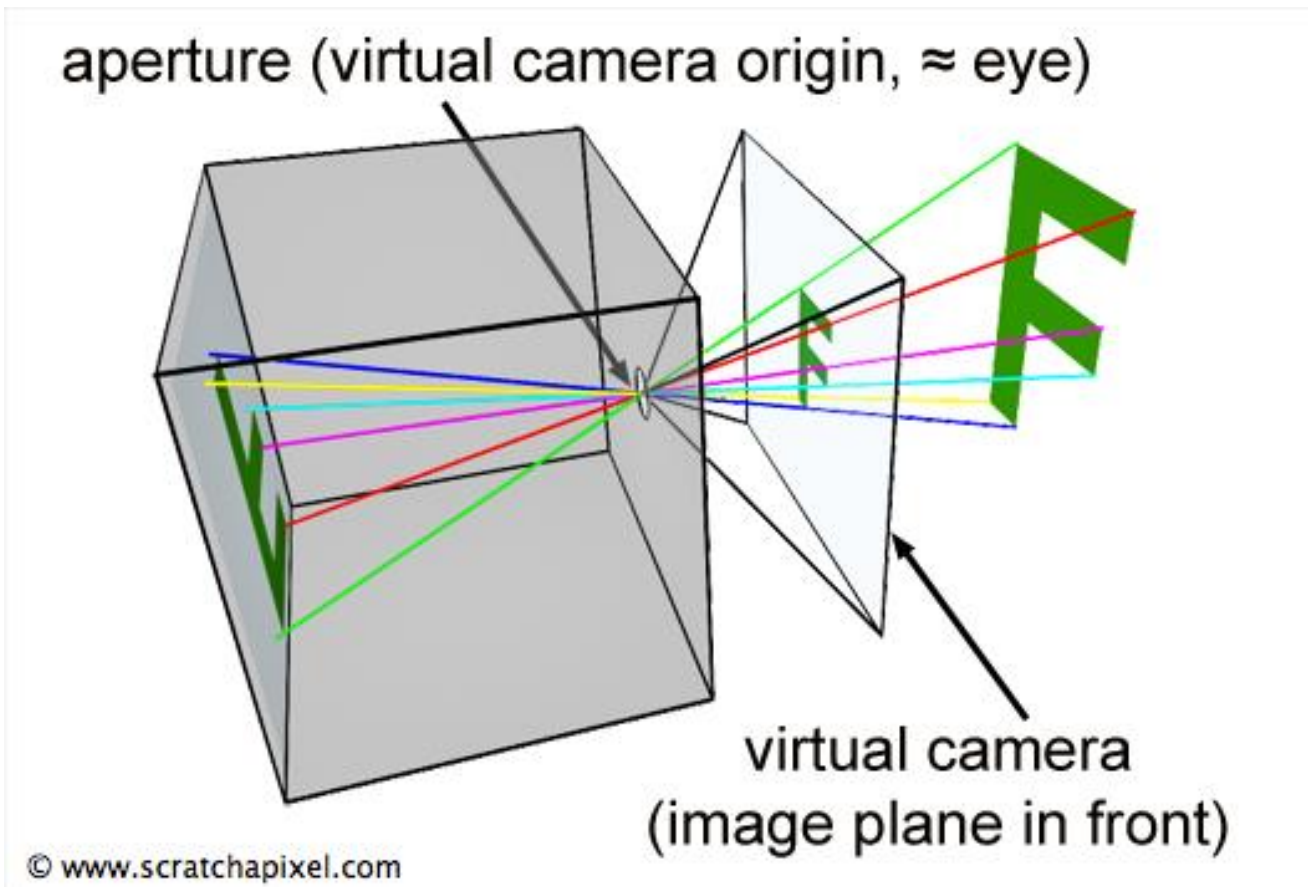
Bounce everywhere

Extremely few
hit eye, form image



Rendering: Reality

Useful abstraction: virtual image plane



Rendering: Reality

Pros

- photorealistic
- embarrassingly parallel?

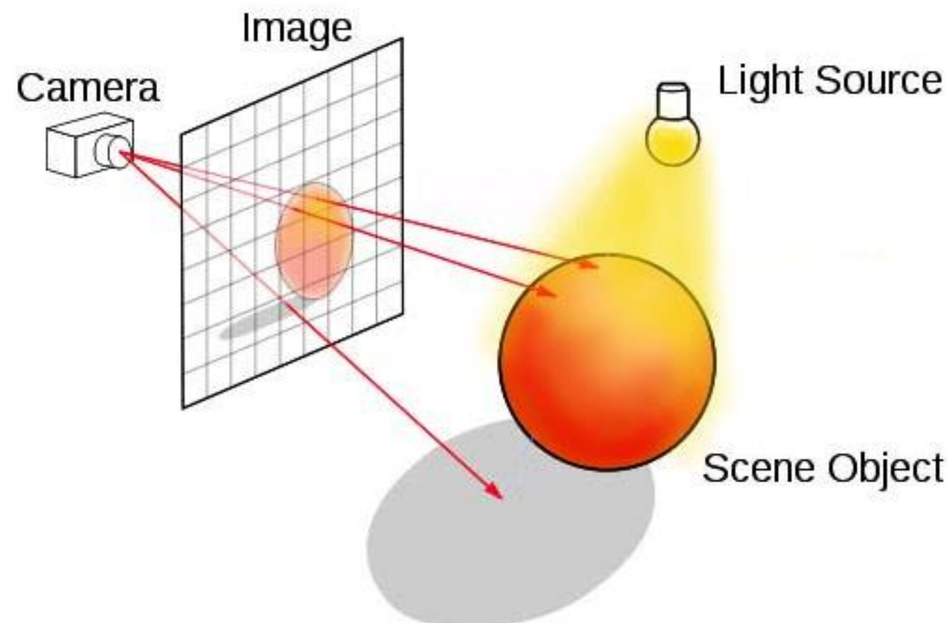
Cons

- **<u>SLOW</u>** for all but extremely trivial scenes

Rendering: Ray Tracing

Reverse of reality

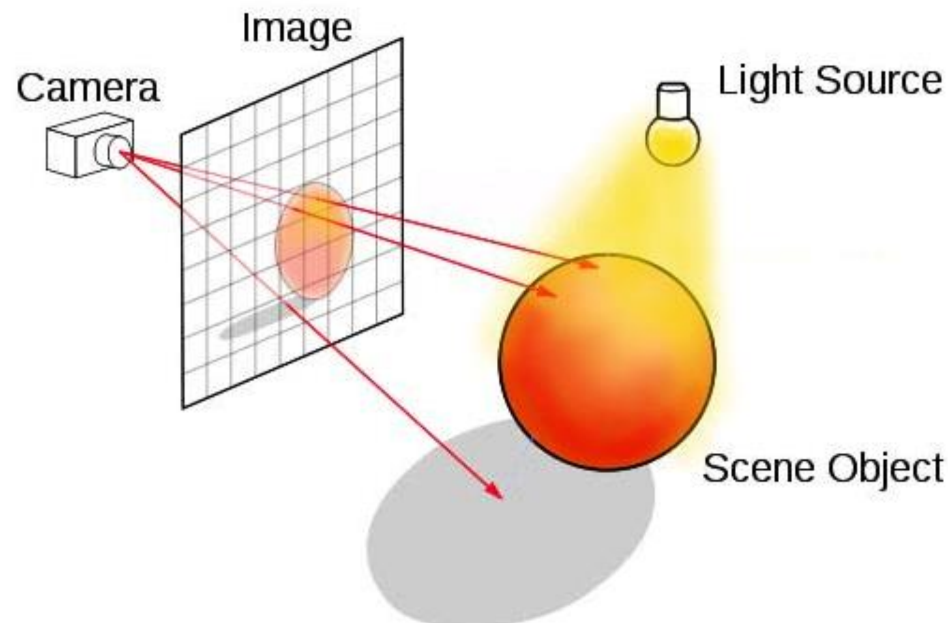
- shoot rays through image plane
- see what they hit



Rendering: Ray Tracing

Reverse of reality

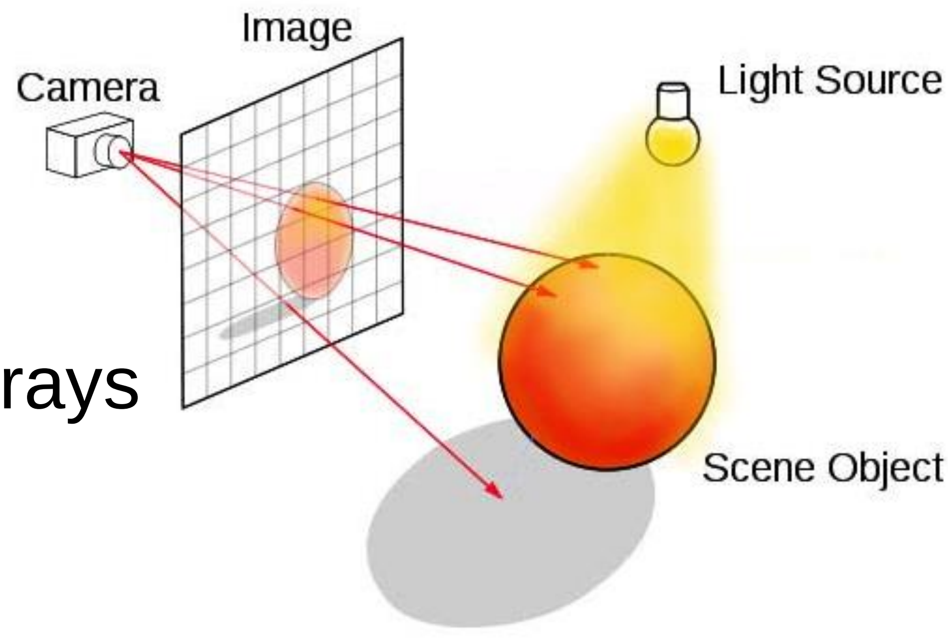
- shoot rays through image plane
- see what they hit
- reflections?
  shadows?



Rendering: Ray Tracing

Reverse of reality

- shoot rays through image plane
- see what they hit
- reflections?
  shadows?
  - shoot **secondary** rays

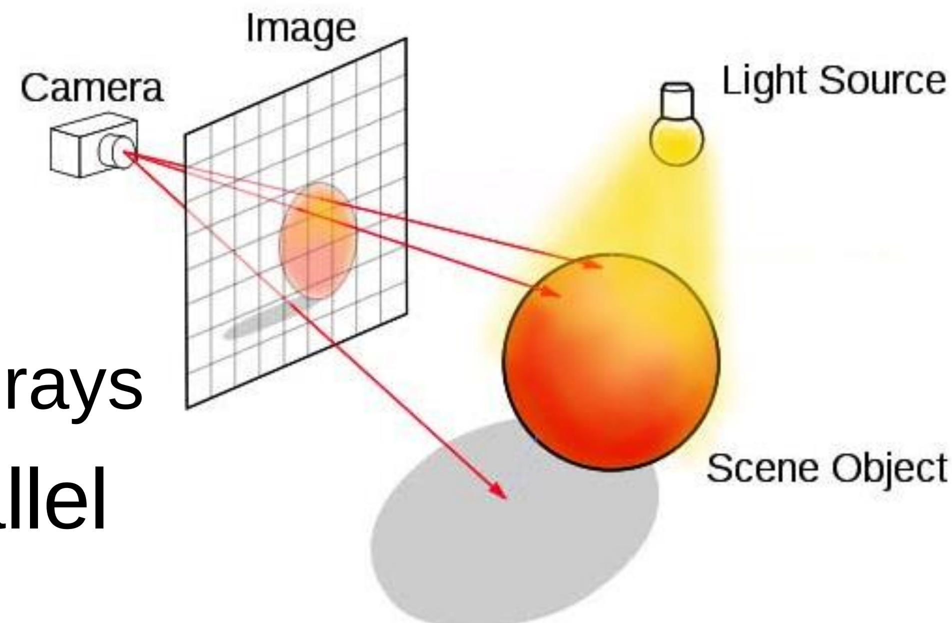


Rendering: Ray Tracing

Reverse of reality

- shoot rays through image plane
- see what they hit
- reflections? shadows?
    - shoot **secondary** rays

Embarrassingly parallel



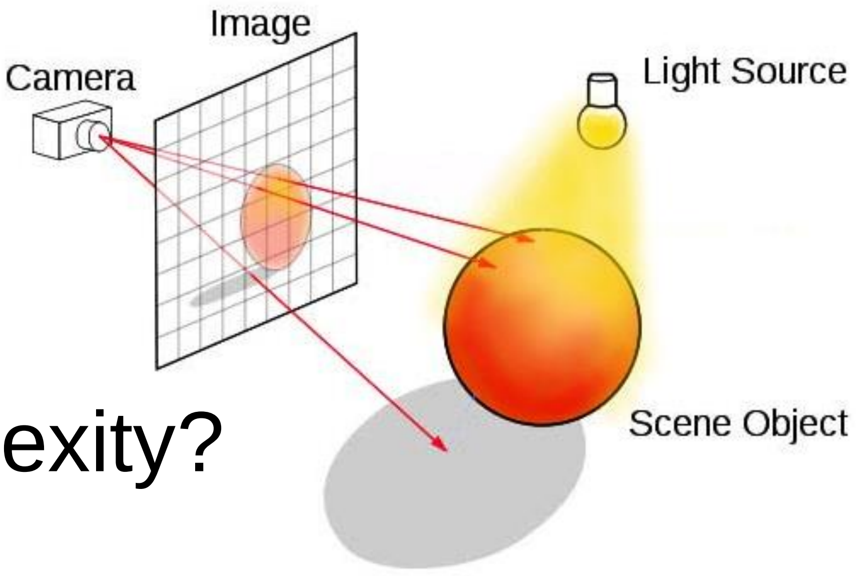
“Ray Tracing is Slow”

Very true in the past; still true today

But real-time ray tracing is coming

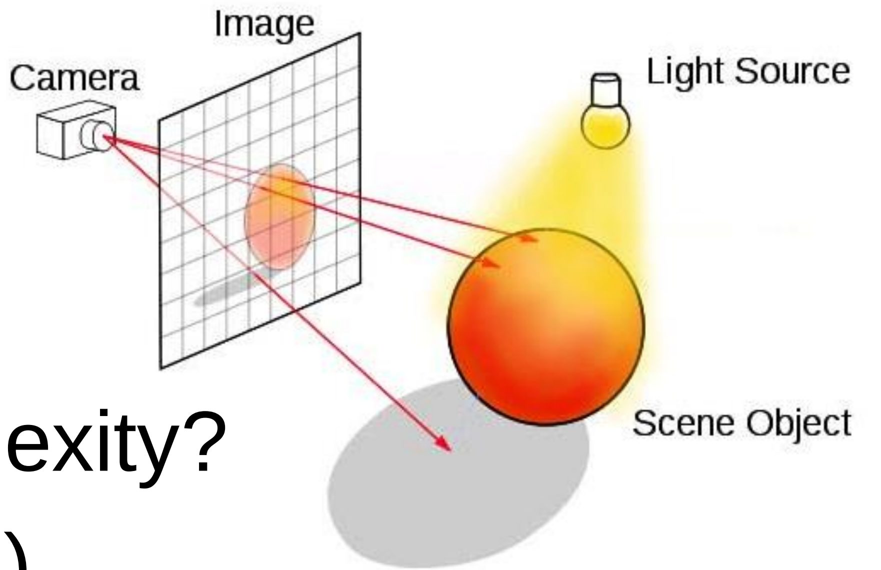
[Nvidia OptiX]

Why Slow?



What is the time complexity?

Why Slow?

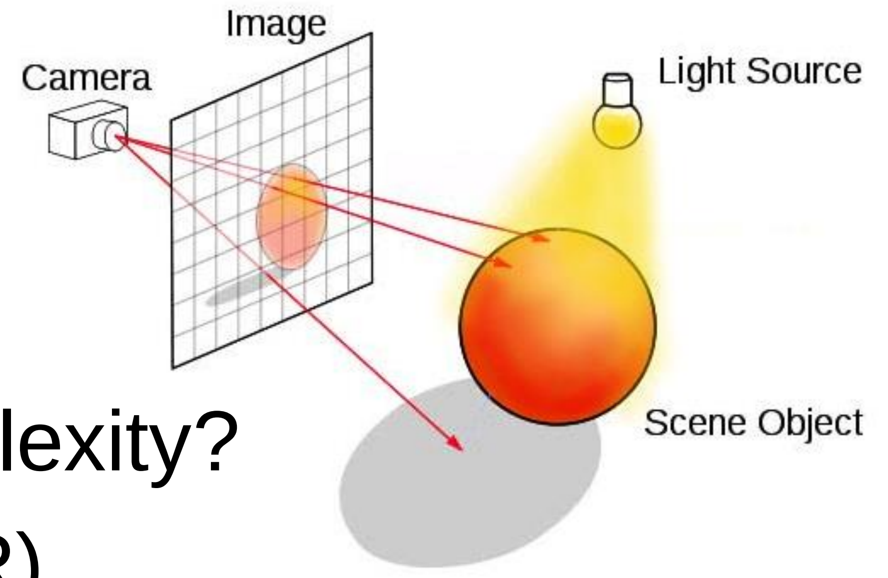


What is the time complexity?

Naïve algorithm: O(NR)

- R: number of rays
- N: number of objects

Why Slow?



What is the time complexity?

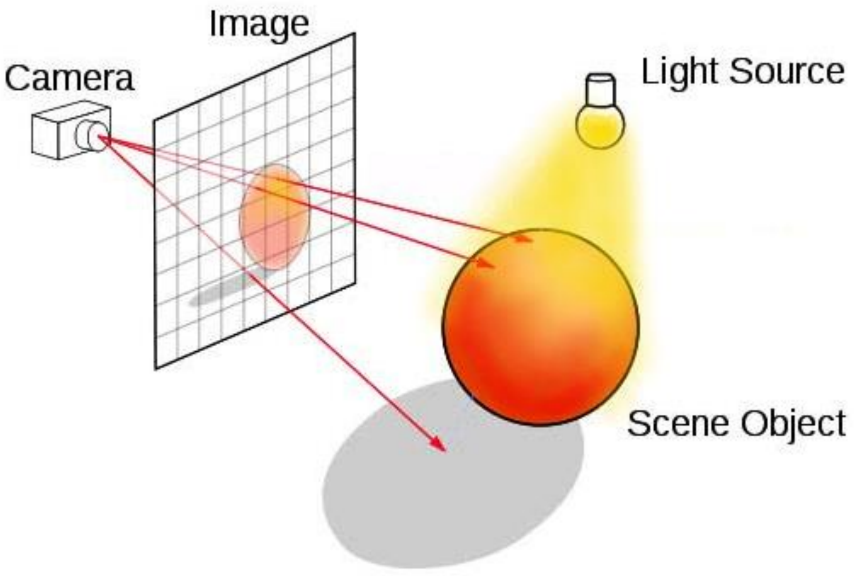
Naïve algorithm: O(NR)

- R: number of rays
- N: number of objects

But rays can be cast in parallel

- each ray O(N)
- even faster with good culling

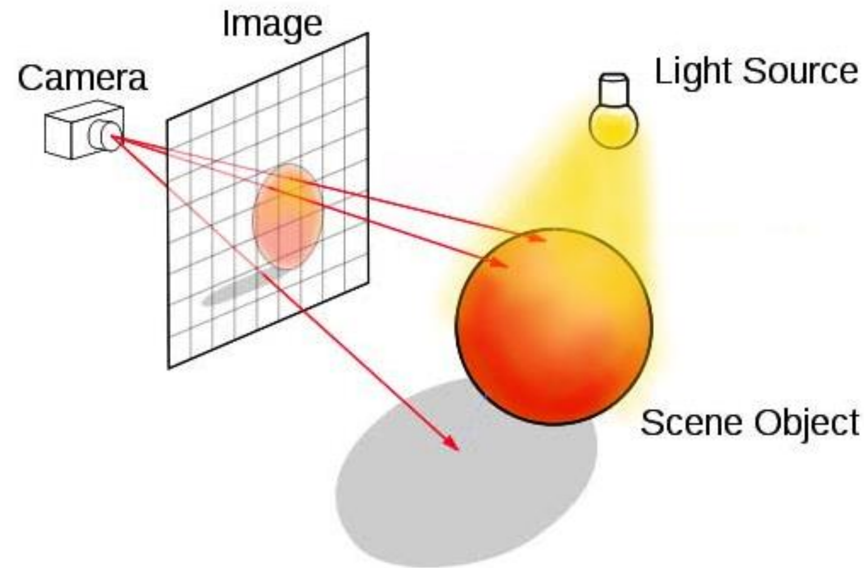
Why Slow?



Despite being parallel:

1. poor cache coherence
   - nearby rays can hit different geometry

Why Slow?



Despite being parallel:

1. poor cache coherence
   - nearby rays can hit different geometry
2. unpredictable
   - must **shade** pixels whose rays hit object
   - may require tracing rays **recursively**

Basic Algorithm

For each pixel:

- shoot ray from camera through pixel
- find first object it hits
- if it hit something
  - shade that pixel

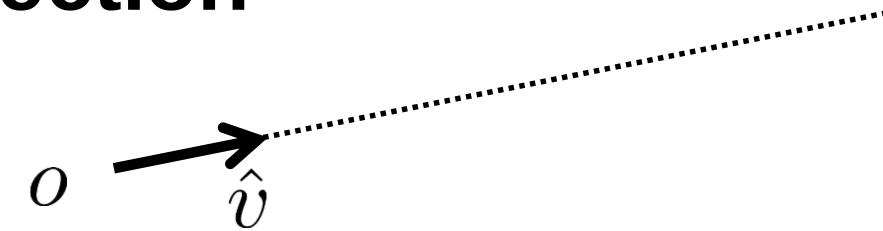
Basic Algorithm

For each pixel:

- shoot ray from camera through pixel
- find first object it hits
- if it hit something
  - shade that pixel
  - maybe shoot secondary rays

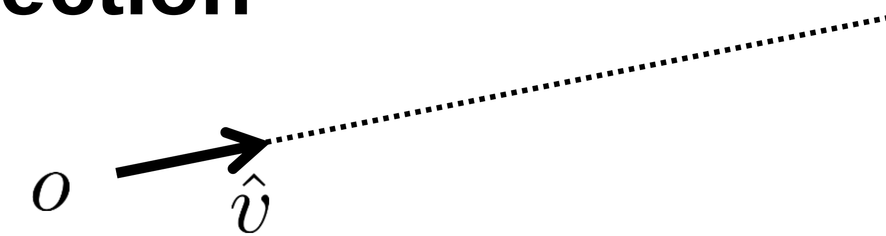
Shoot Rays From Camera

Ray has **origin** and **direction**



Shoot Rays From Camera

Ray has **origin** and **direction**

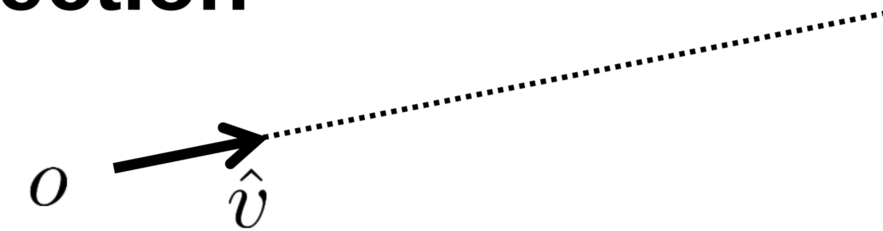


Points on ray are the **positive span**

$$o + \hat{v}t, \quad t \geq 0$$

Shoot Rays From Camera

Ray has **origin** and **direction**



Points on ray are the **positive span**

$$o + \hat{v}t, \quad t \geq 0$$

(why positive?)

Shoot Rays From Camera

How to pick ray?

- obviously origin is eye

Shoot Rays From Camera

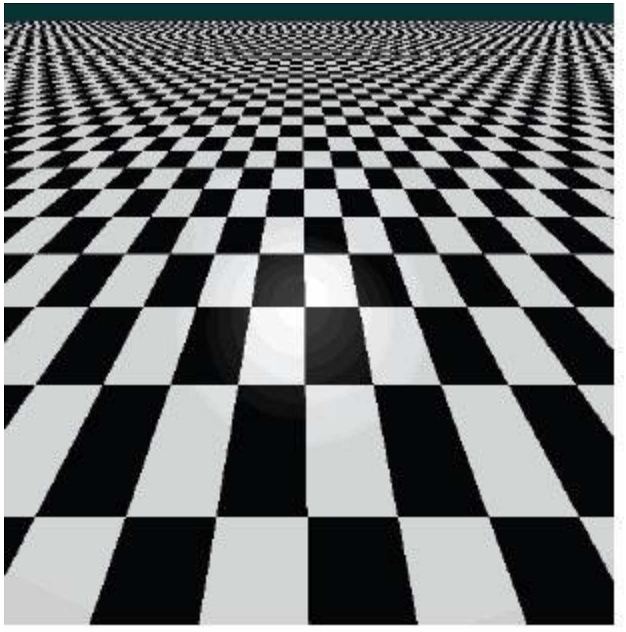
How to pick ray?

- obviously origin is eye
- pick direction to pierce **center** of pixel

Shoot Rays From Camera

How to pick ray?

- obviously origin is eye
- pick direction to pierce **center** of pixel



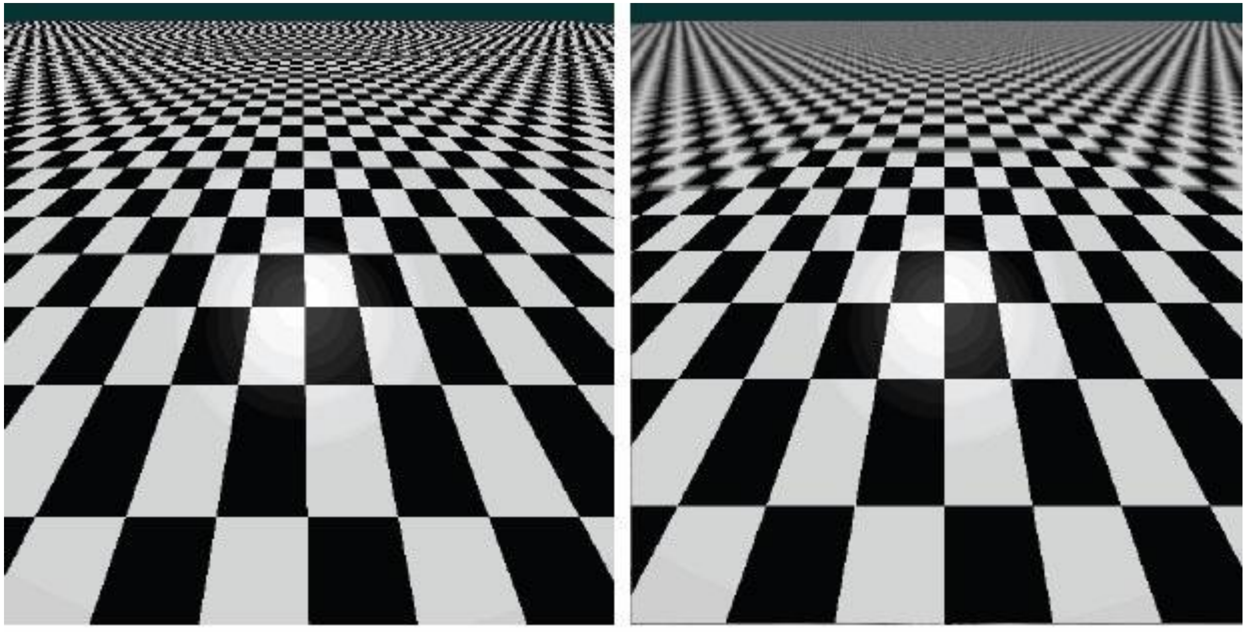
Shoot Rays From Camera

How to pick ray?

- obviously origin is eye
- pick direction to pierce **center** of pixel

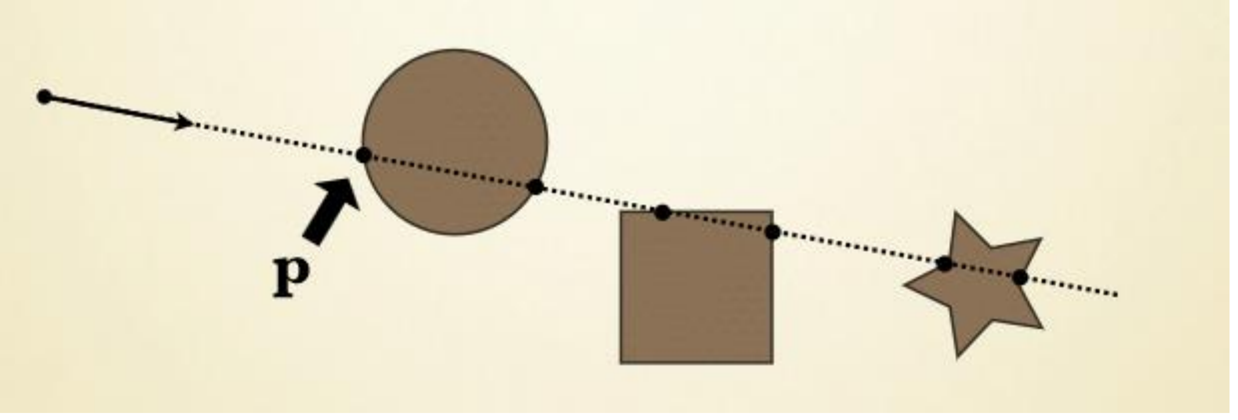
Antialiasing:

multiple rays/pixel



Find First Object Hit By Ray

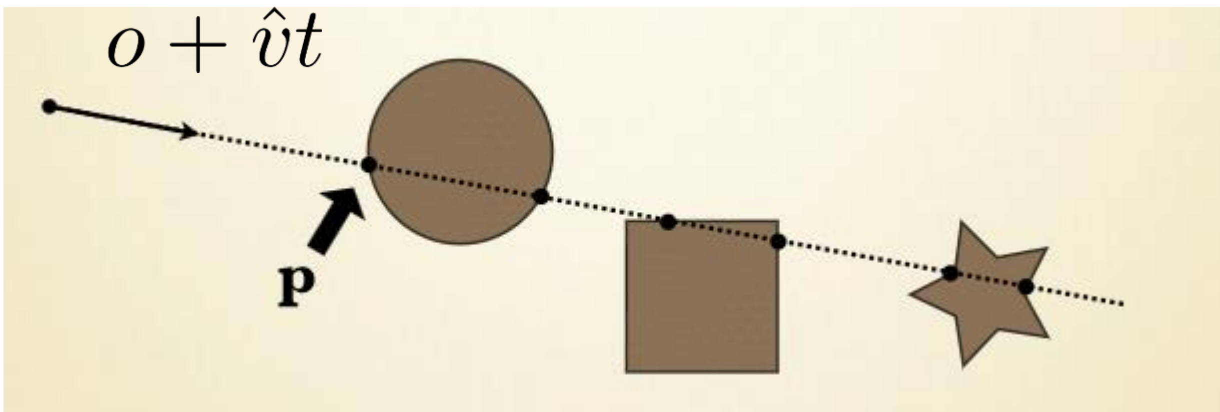
Collision detection: find all values of t where ray hits object boundary



Take smallest **positive** value of t

Find First Object Hit By Ray

Collision detection: find all values of t where ray hits object boundary

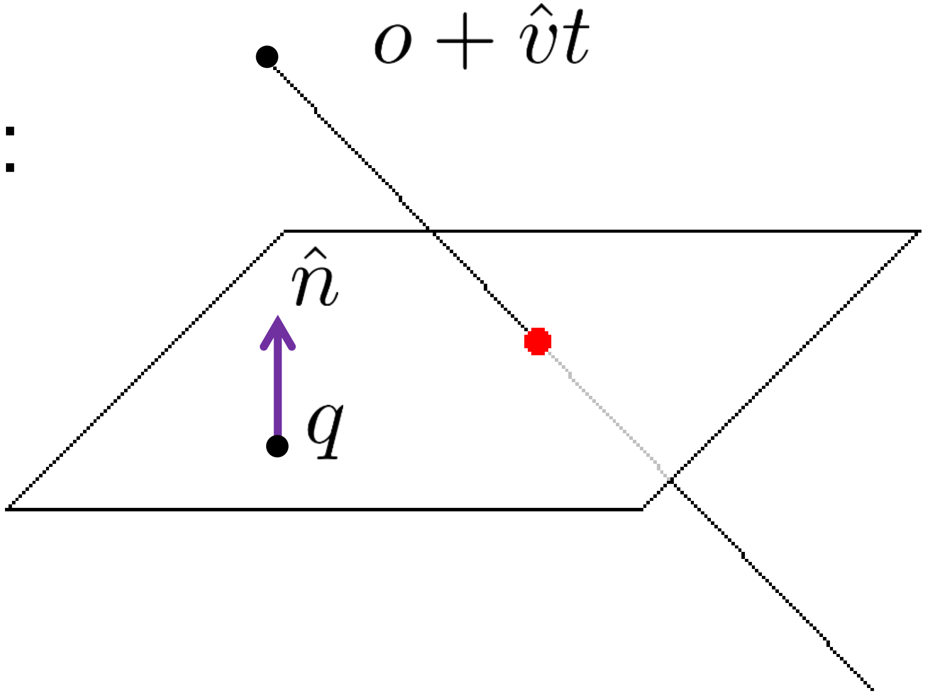


Take smallest **positive** value of t

Ray-Plane Collision Detection

Plane specified by:

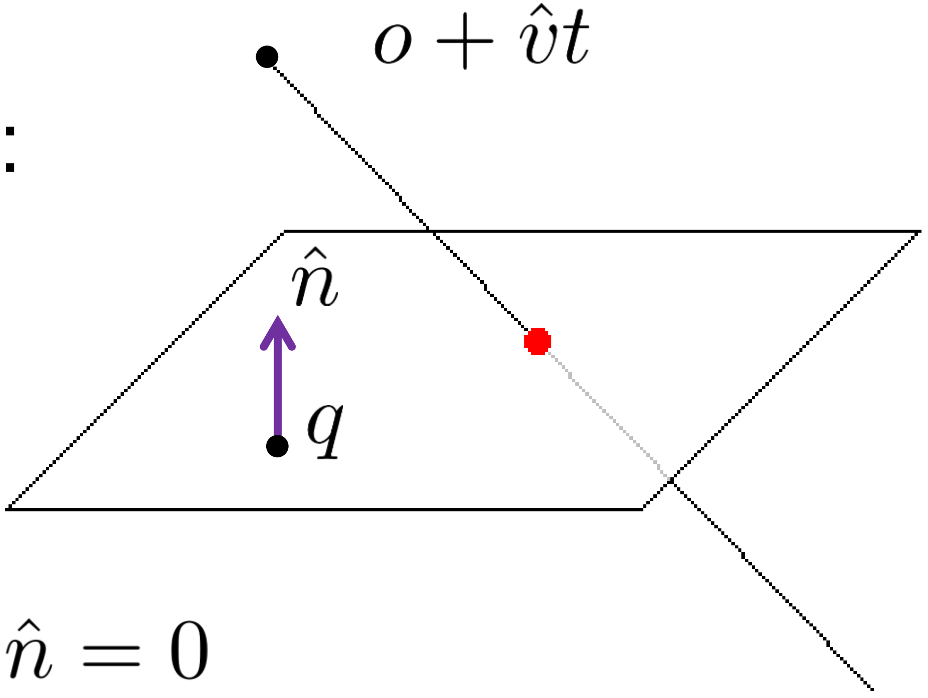
- point on plane
- plane normal



Ray-Plane Collision Detection

Plane specified by:

- point on plane
- plane normal



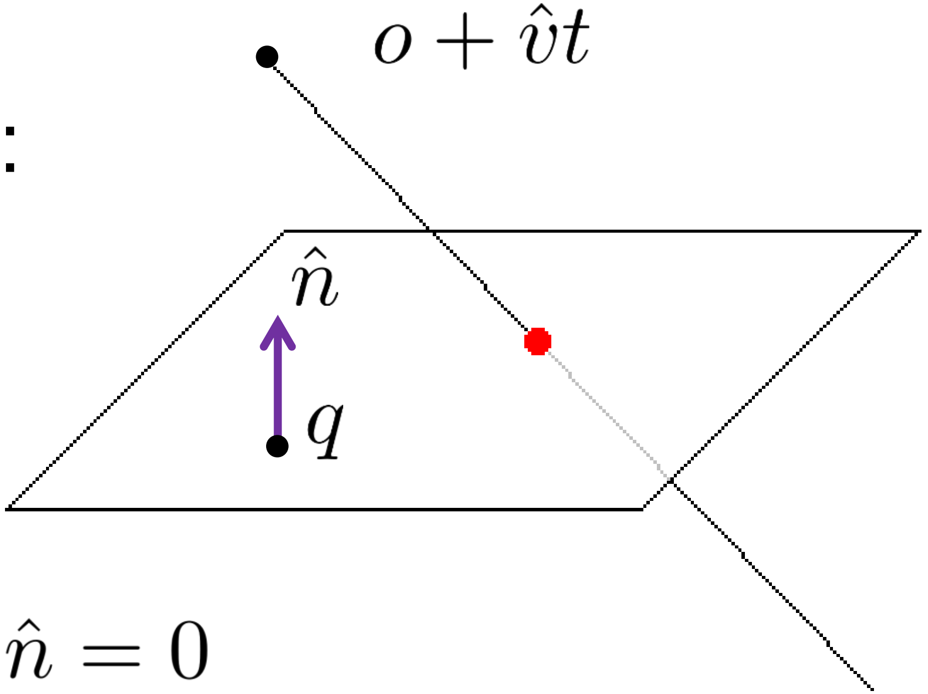
$$(o + \hat{v}t - q) \cdot \hat{n} = 0$$

$$t = \frac{(q-o)\cdot\hat{n}}{\hat{v}\cdot\hat{n}}$$

Ray-Plane Collision Detection

Plane specified by:

- point on plane
- plane normal



$$(o + \hat{v}t - q) \cdot \hat{n} = 0$$

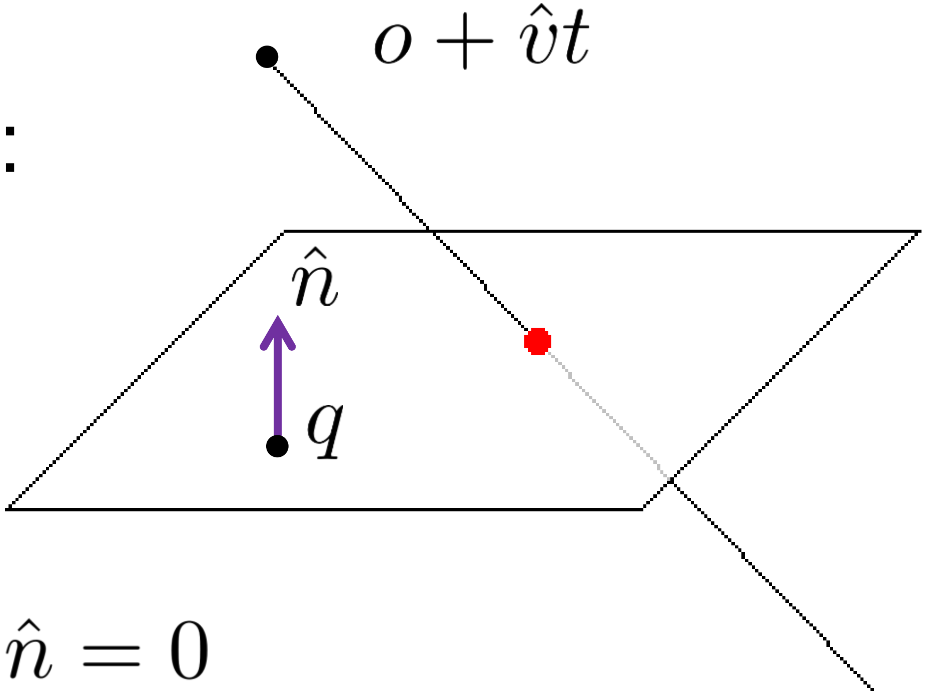
$$t = \frac{(q-o)\cdot\hat{n}}{\hat{v}\cdot\hat{n}}$$

(what if t < 0?)

Ray-Plane Collision Detection

Plane specified by:

- point on plane
- plane normal



$$(o + \hat{v}t - q) \cdot \hat{n} = 0$$

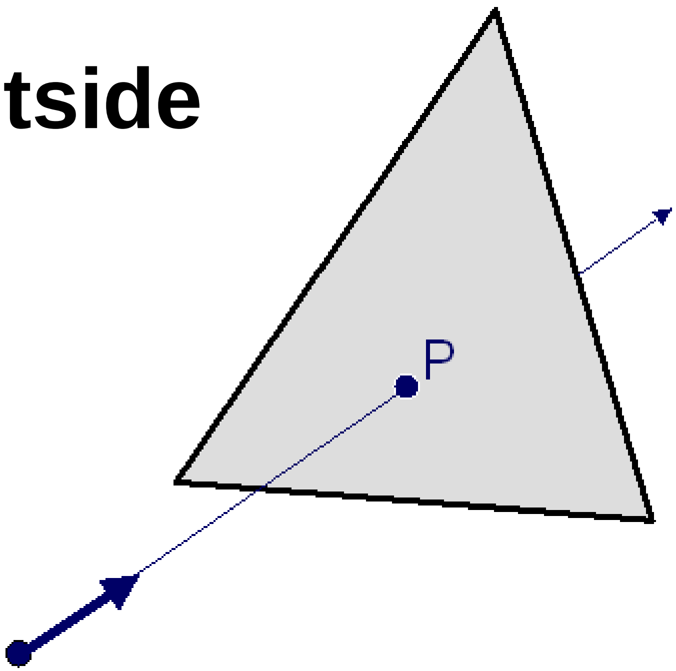
$$t = \frac{(q-o)\cdot\hat{n}}{\hat{v}\cdot\hat{n}}$$

(what if t < 0?)
(what if denominator = 0?)

Ray-Triangle Collision Detection

First, intersect with triangle's plane

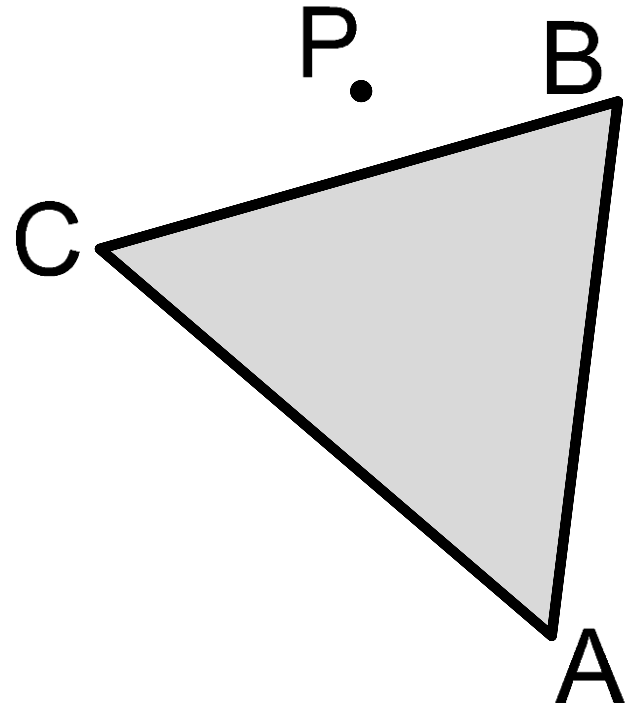
Next: is P **inside** or **outside** the triangle?



Ray-Triangle Collision Detection

Normal:

$$\hat{n} = \frac{(B-A)\times(C-A)}{\|(B-A)\times(C-A)\|}$$

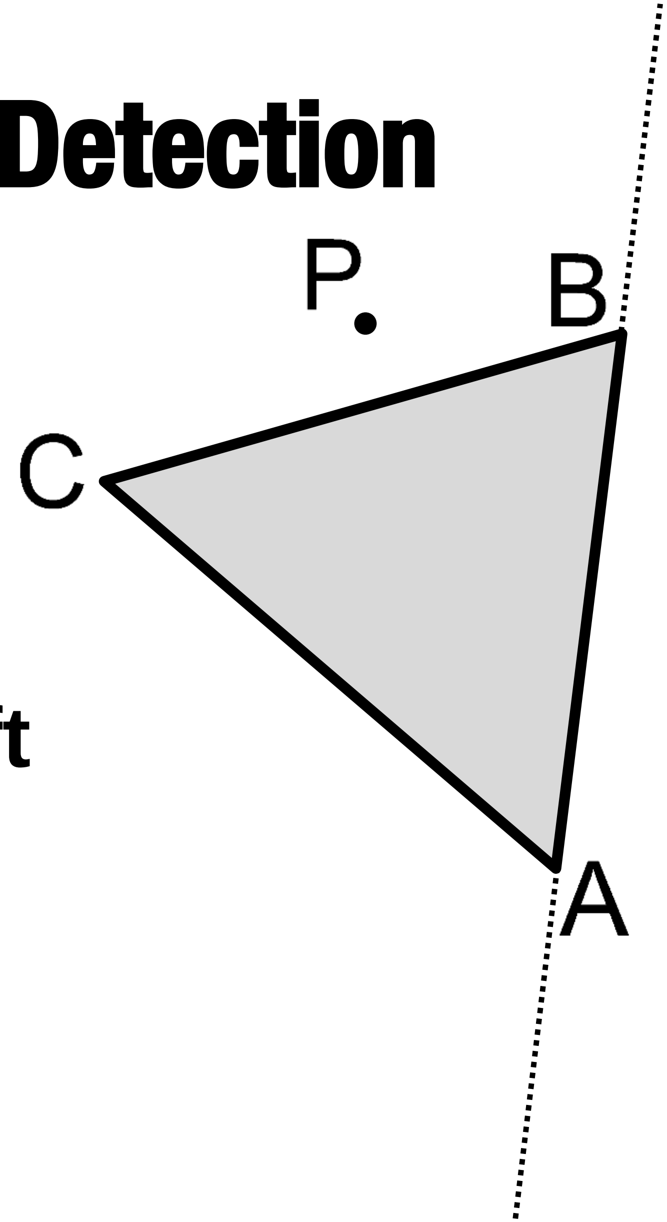


Ray-Triangle Collision Detection

Normal:

$$\hat{n} = \frac{(B-A)\times(C-A)}{\|(B-A)\times(C-A)\|}$$

Idea: if P inside, must be **left** of line AB



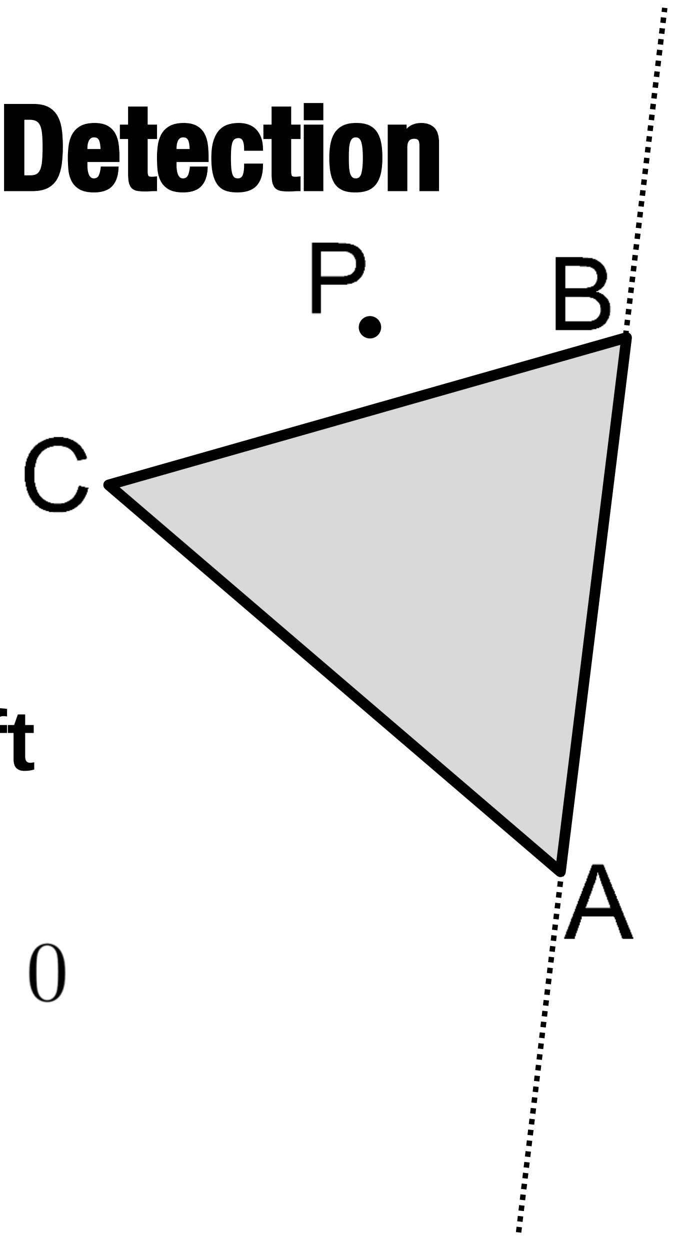
Ray-Triangle Collision Detection

Normal:

$$\hat{n} = \frac{(B-A)\times(C-A)}{\|(B-A)\times(C-A)\|}$$

Idea: if P inside, must be **left** of line AB

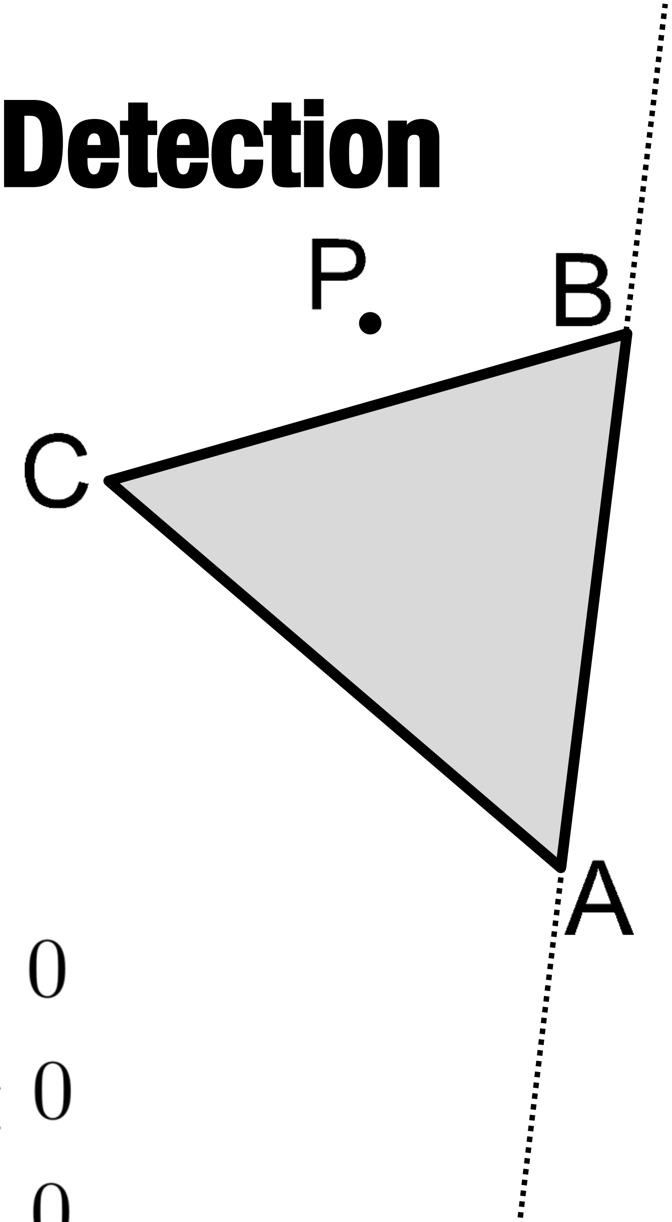
$$(B - A) \times (P - A) \cdot \hat{n} \geq 0$$



Ray-Triangle Collision Detection

Normal:

$$\hat{n} = \frac{(B-A)\times(C-A)}{\|(B-A)\times(C-A)\|}$$



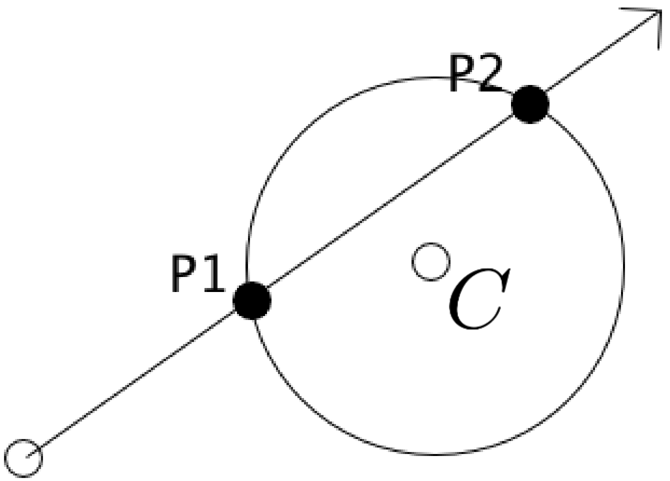
Idea: if P inside, must be on correct side of lines

$$(B - A) \times (P - A) \cdot \hat{n} \geq 0$$
$$(C - B) \times (P - B) \cdot \hat{n} \geq 0$$
$$(A - C) \times (P - C) \cdot \hat{n} \geq 0$$

Ray-Sphere Collision Detection

Sphere specified by

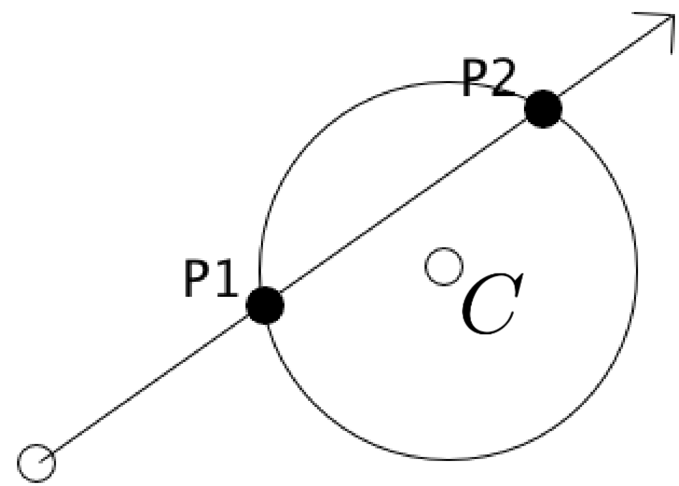
- center C
- radius r



Ray-Sphere Collision Detection

Sphere specified by

- center C
- radius r

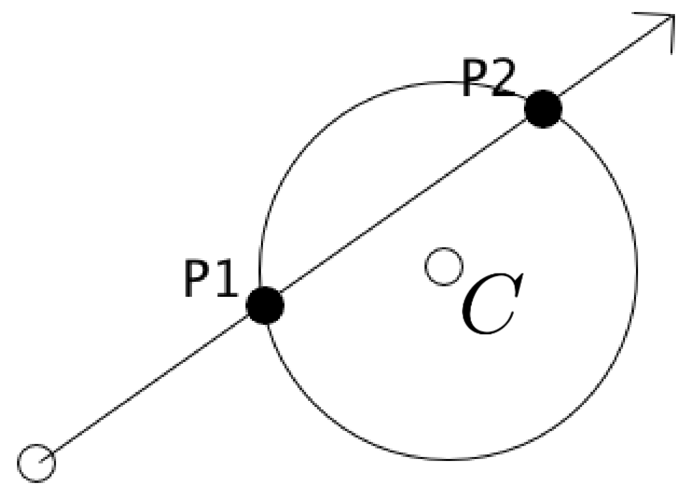


$$\|o + \hat{v}t - C\| = r$$

Ray-Sphere Collision Detection

Sphere specified by

- center C
- radius r



$$\|o + \hat{v}t - C\| = r$$

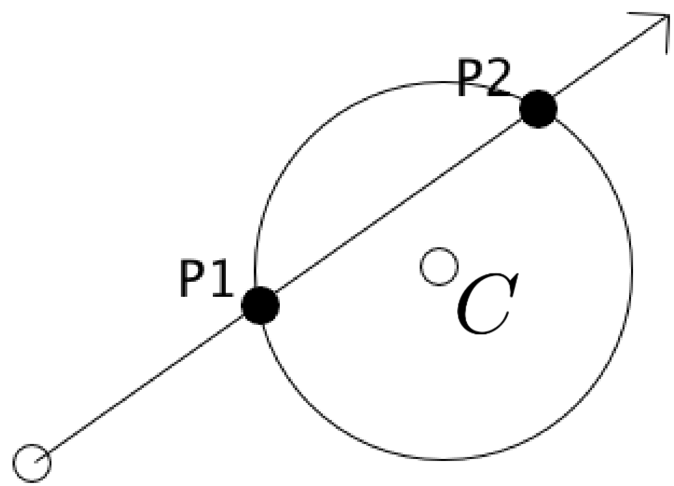
key idea: can **square both sides**

$$\|o + \hat{v}t - C\|^2 = r^2$$

Ray-Sphere Collision Detection

Sphere specified by

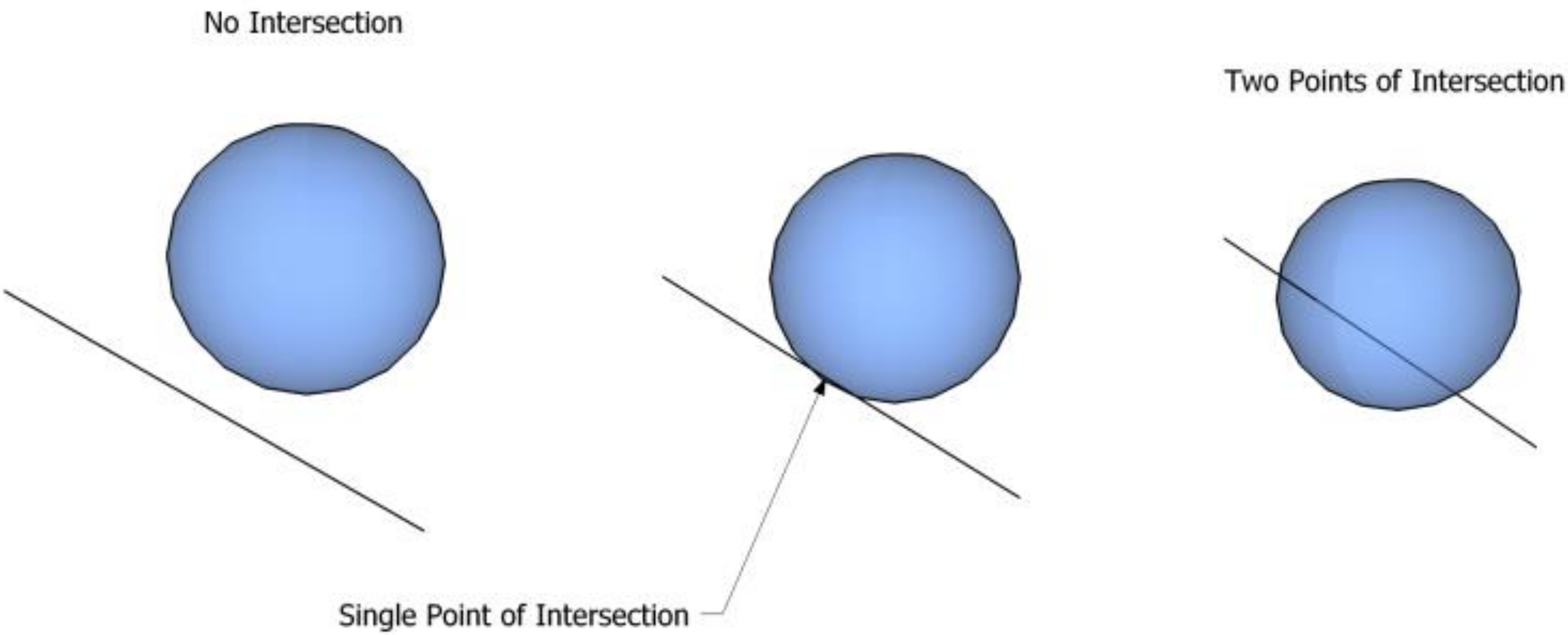
- center C
- radius r



$$t^2 + \left[2(o - C) \cdot \hat{v}\right] t + \left[(o - C) \cdot (o - C) - r^2\right] = 0$$

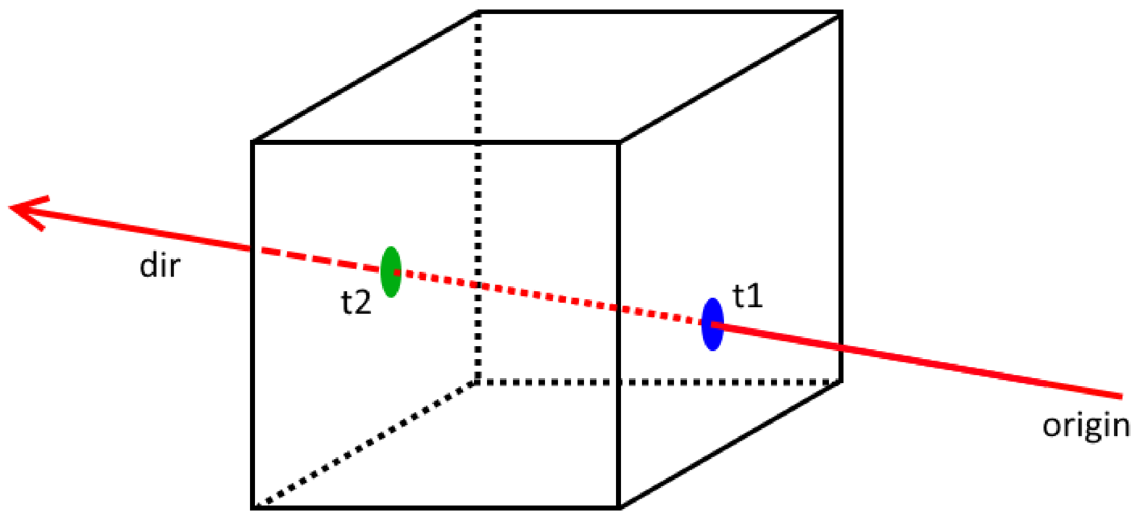
Quadratic equation!

Zero, One, or Two Roots



Ray-Box Collision Detection

Challenge: ray could hit any of six sides



Could do lots of ray-plane and point-in-rectangle checks…

What is Shading?

Shading: coloring the pixels

What does color depend on?

What is Shading?

Shading: coloring the pixels

What does color depend on?

- object material
- incoming light
- angle of viewer

Shading Materials

Different materials can behave **very** differently

- opaque vs translucent vs transparent
- shiny vs dull

Shading Materials

Different materials can behave **very** differently

- opaque vs translucent vs transparent
- shiny vs dull

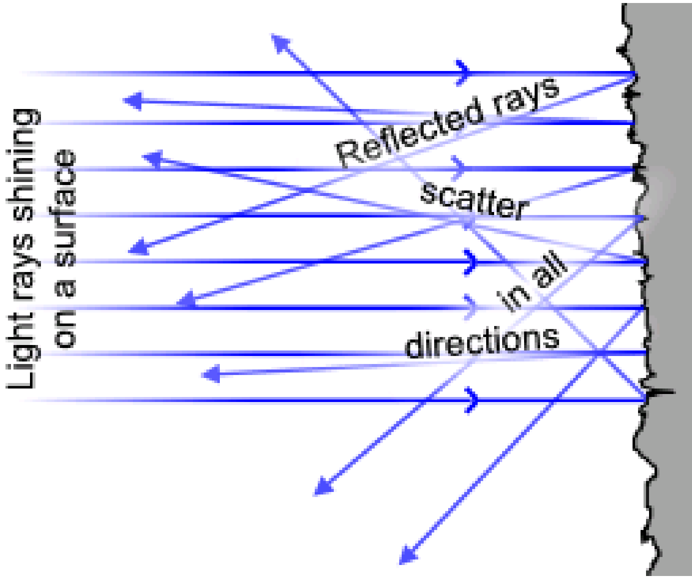
We classify different responses to light into “types”

Emissive Lighting

Light generated within material

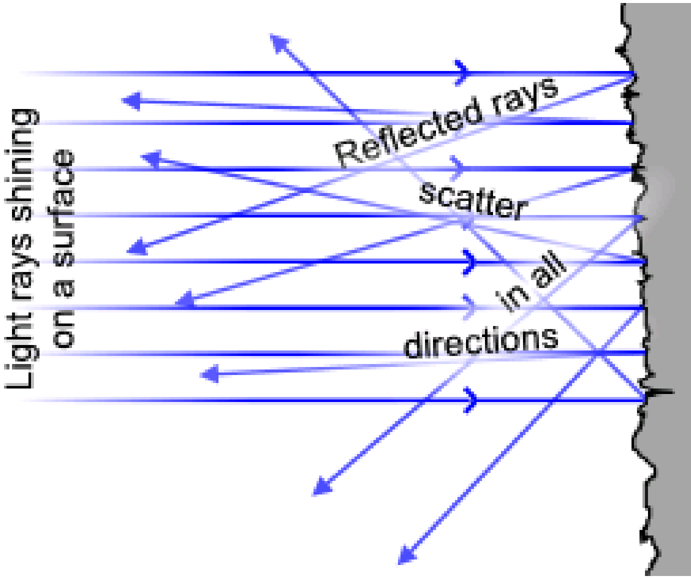
Diffuse Reflection

Light comes in, bounces out randomly



Diffuse Reflection

Light comes in, bounces out randomly

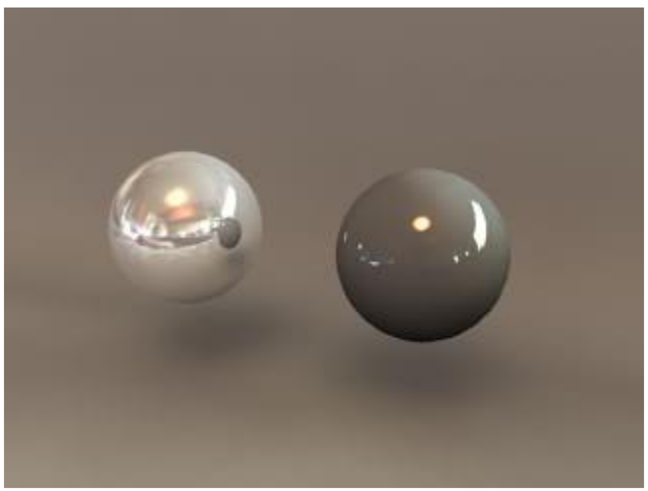


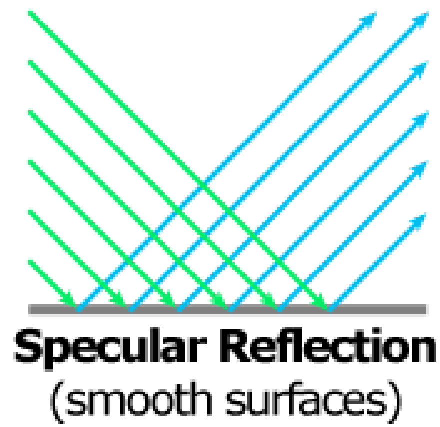
Typical for “rough” unpolished materials

View angle doesn’t matter

Specular Reflection

Light reflects perfectly

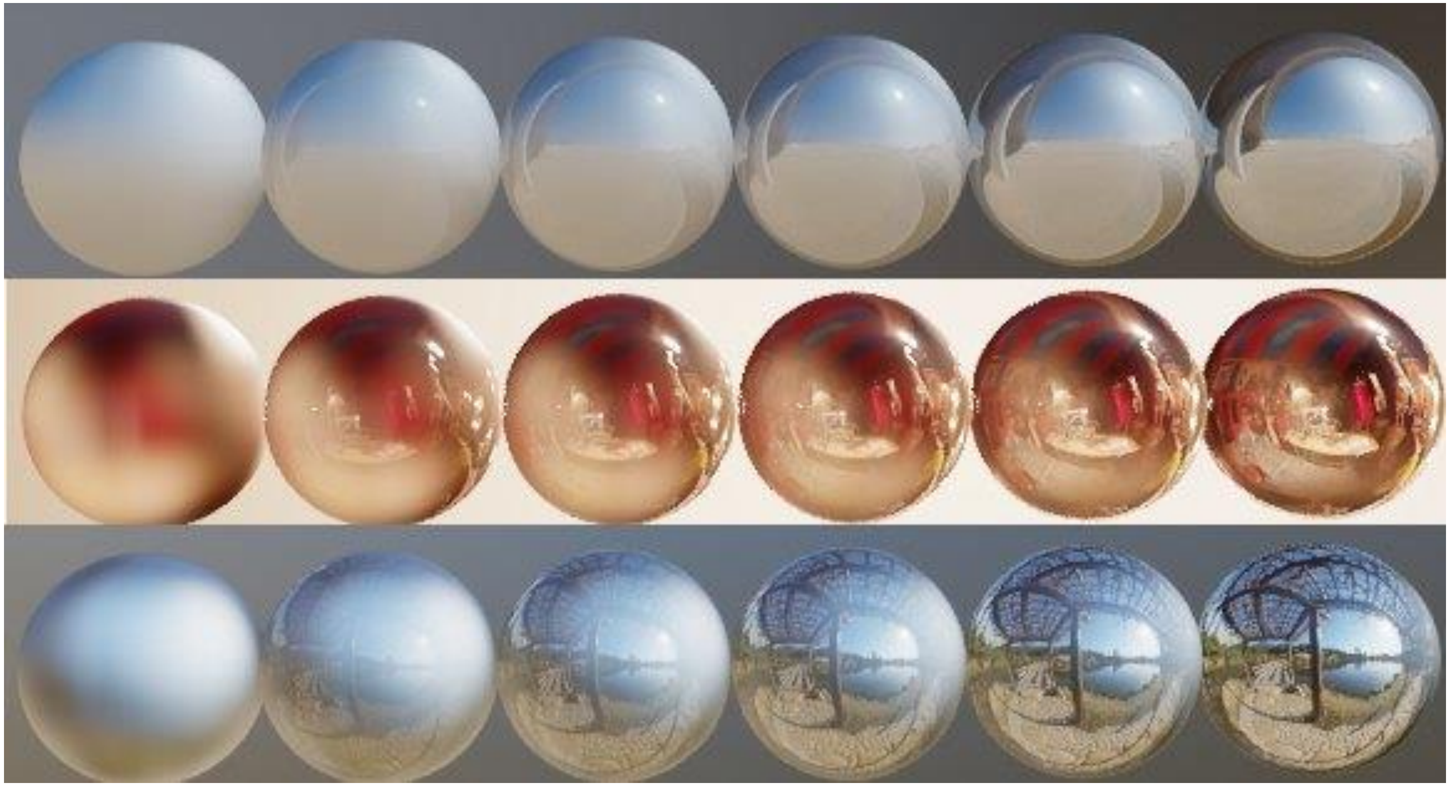




Typical for smooth, “polished” surfaces

General Opaque Materials

Lie on diffuse-specular spectrum



General Opaque Materials

Lie on diffuse-specular spectrum

Pure diffuse: **Lambertian**

- idealized material common in CV…

General Opaque Materials

Lie on diffuse-specular spectrum

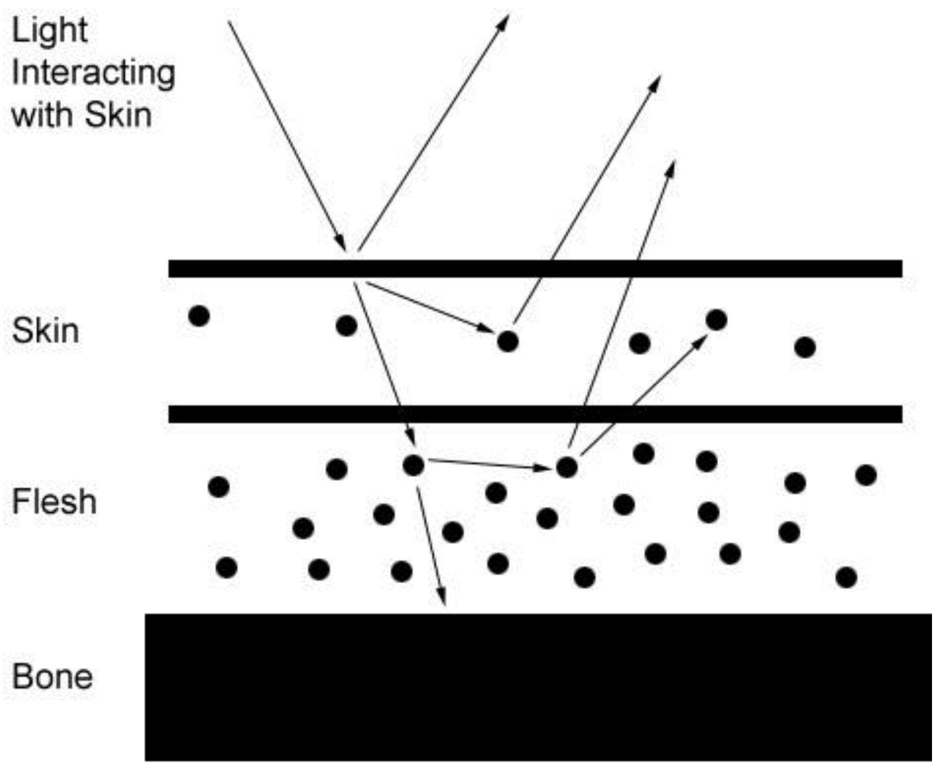
Pure diffuse: **Lambertian**

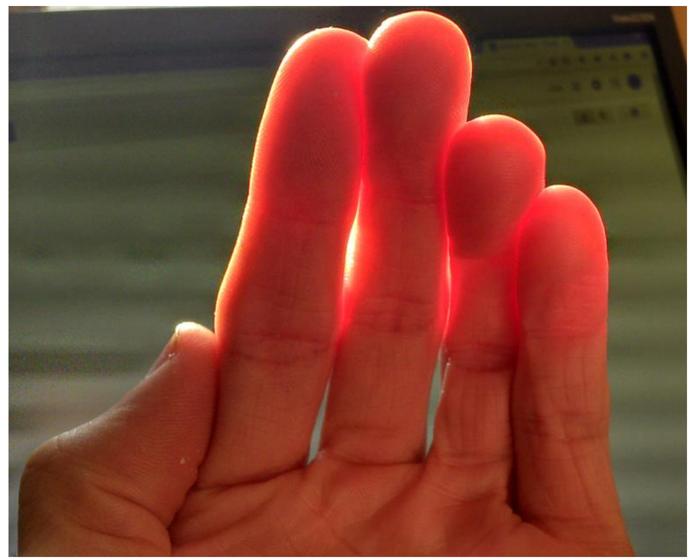
- idealized material common in CV…

Pure specular: mirror

What About Translucent?

Subsurface Scattering

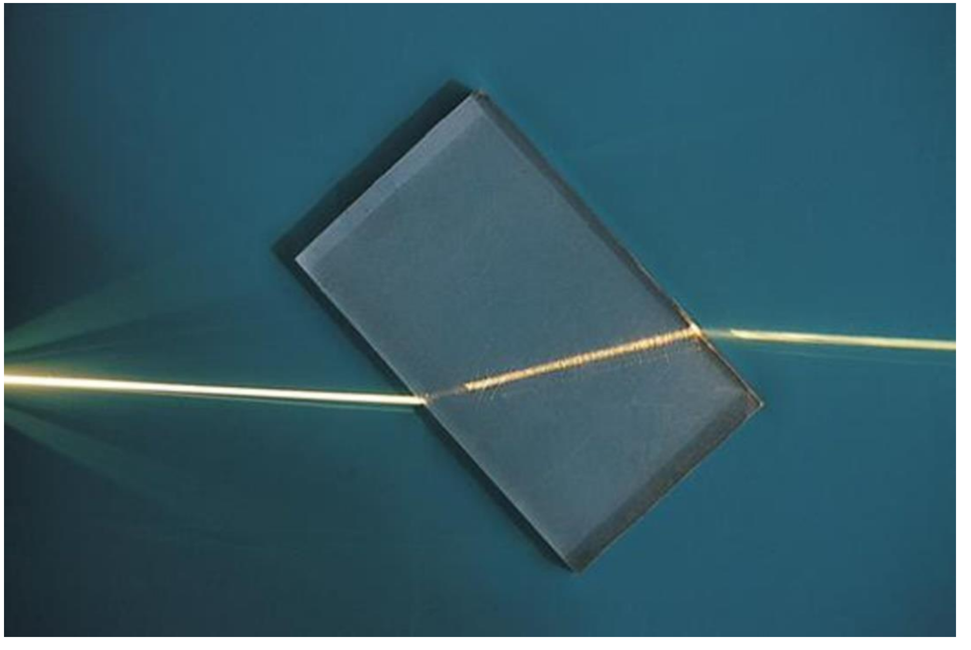




What About Translucent?

Subsurface Scattering

Refraction



What About Translucent?

Subsurface Scattering

Refraction

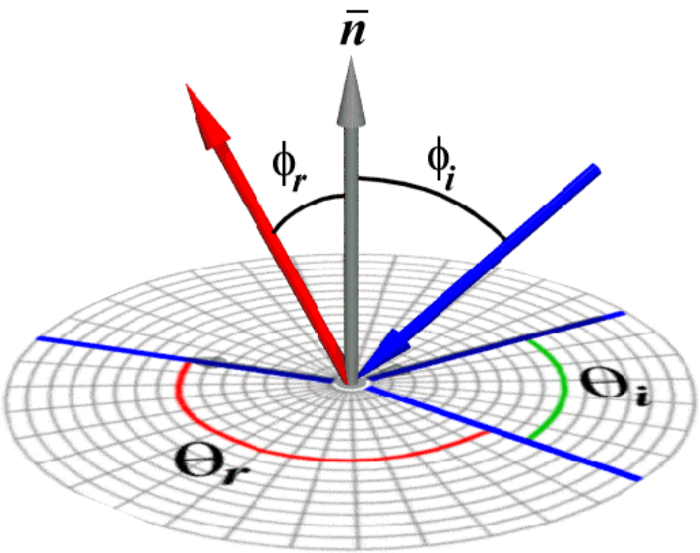
Structural Color

…

Not today.

The Rendering Equation

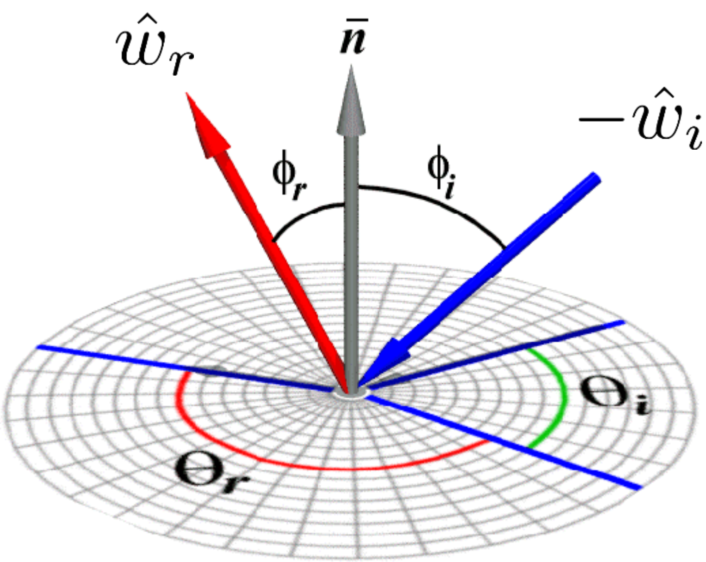
$$L_{\mathrm{out}}(\theta_r, \phi_r) = \int_{\theta_i} \int_{\phi_i} f_r(\theta_r, \phi_r, \theta_i, \phi_i) L_{\mathrm{in}}(\theta_i, \phi_i) \cos \theta_i$$



The Rendering Equation

$$L_{\text{out}}(\theta_r, \phi_r) = \int_{\theta_i} \int_{\phi_i} f_r(\theta_r, \phi_r, \theta_i, \phi_i) L_{\text{in}}(\theta_i, \phi_i) \cos \theta_i$$

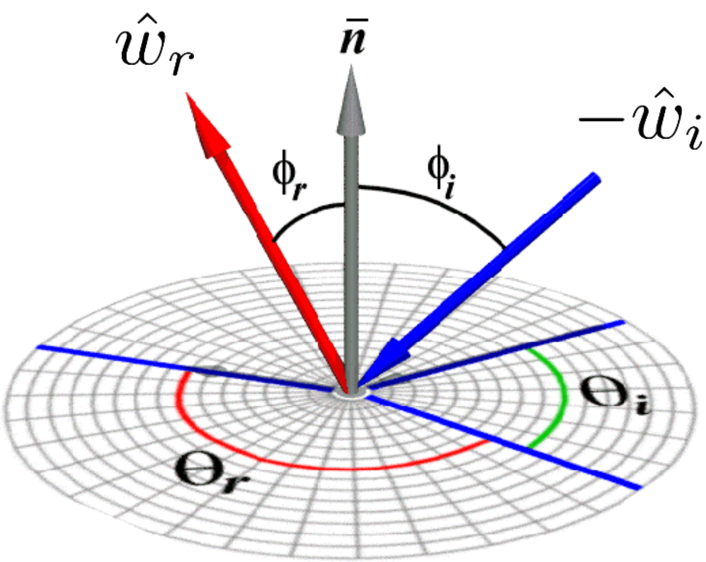
$$L_{\text{out}}(\hat{w}_r) = \int_{\hat{w}_i \in \text{hemisphere}} f_r(\hat{w}_r, \hat{w}_i)\ L_{\text{in}}(\hat{w}_i)\ \hat{w}_i \cdot \hat{n}$$



The Rendering Equation

$$L_{\text{out}}(\theta_r, \phi_r) = \int_{\theta_i} \int_{\phi_i} f_r(\theta_r, \phi_r, \theta_i, \phi_i) L_{\text{in}}(\theta_i, \phi_i) \cos\theta_i$$

$$L_{\text{out}}(\hat{w}_r) = \int_{\hat{w}_i \in \text{hemisphere}} f_r(\hat{w}_r, \hat{w}_i)\; L_{\text{in}}(\hat{w}_i)\; \hat{w}_i \cdot \hat{n}$$

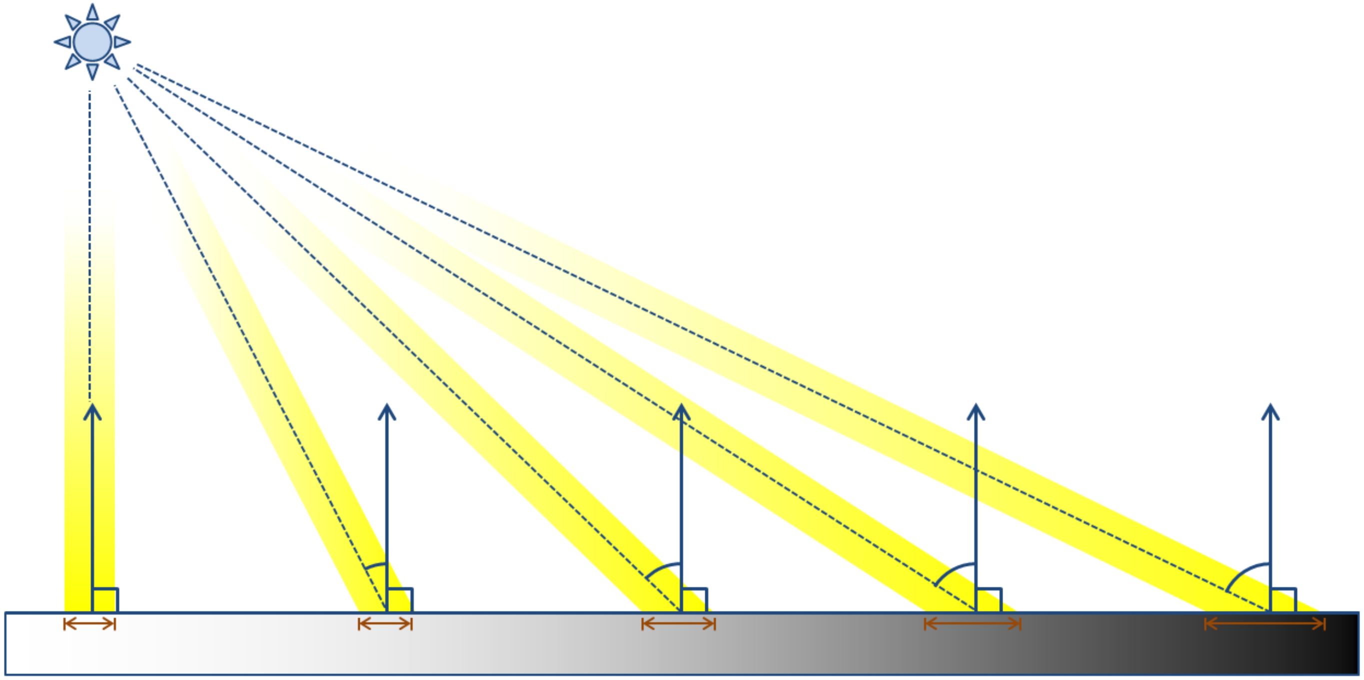


BRDF
"Bidirectional Reflectance Distribution Function"
(encodes material)

Why the Cosine Term?

Light at angle hits surface more sparsely

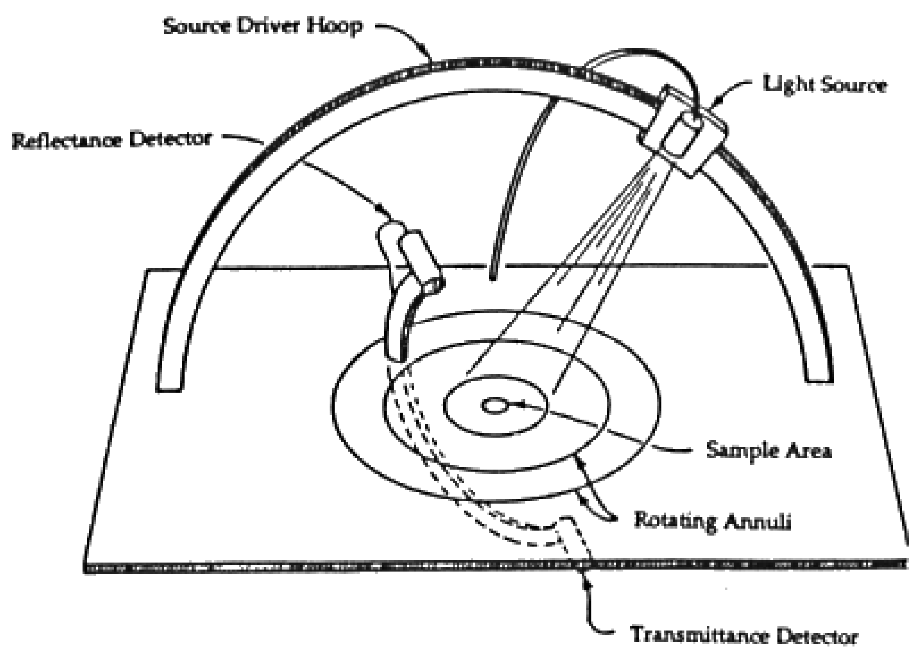
- "Lambert's Cosine Law"



BRDFs

Positive and bidirectional: $f_r(\hat{w}_r, \hat{w}_i) = f_r(\hat{w}_i, \hat{w}_r)$

Captured for different materials, stored in libraries



BRDFs

Positive and bidirectional: $f_r(\hat{w}_r, \hat{w}_i) = f_r(\hat{w}_i, \hat{w}_r)$

Captured for different materials, stored in libraries

More complicated versions exist that account for wavelength, subsurface scattering, transmission, etc etc

The Rendering Equation

$$L_{\text{out}}(\theta_r, \phi_r) = \int_{\theta_i} \int_{\phi_i} f_r(\theta_r, \phi_r, \theta_i, \phi_i) L_{\text{in}}(\theta_i, \phi_i) \cos\theta_i$$

$$L_{\text{out}}(\hat{w}_r) = \int_{\hat{w}_i \in \text{hemisphere}} f_r(\hat{w}_r, \hat{w}_i)\ L_{\text{in}}(\hat{w}_i)\ \hat{w}_i \cdot \hat{n}$$

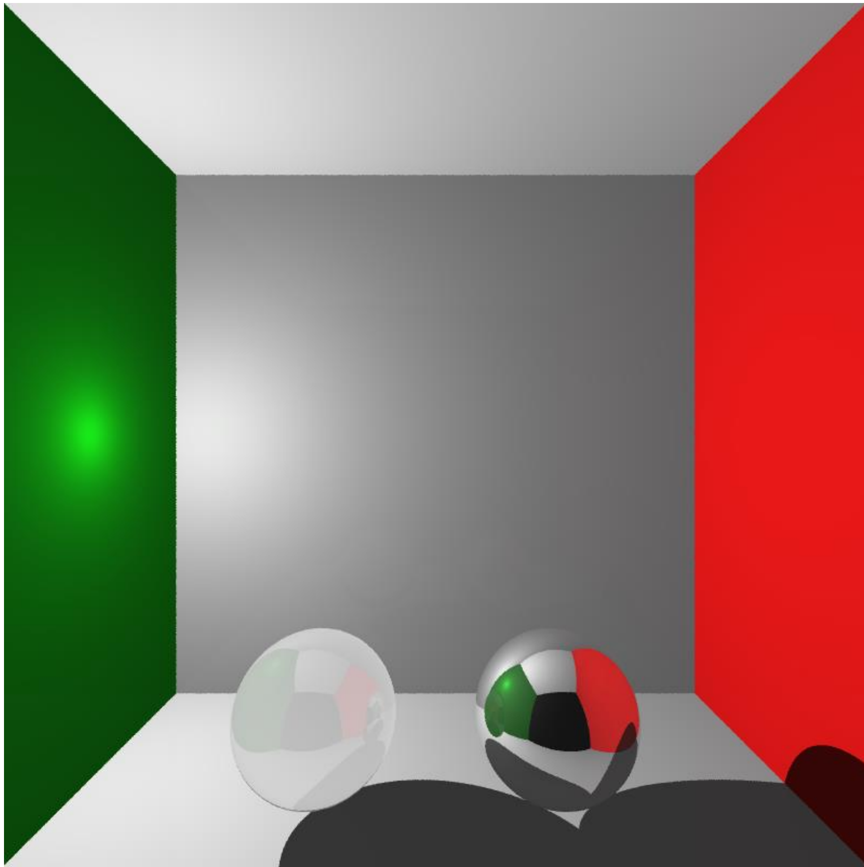
Often too slow for graphics

- **approximate**!

Local Illumination

Simplifying assumptions:

- ignore everything except: eye, light, and object



Local Illumination

Simplifying assumptions:

- ignore everything except eye, light, and object
  - basic version: no shadows, reflections, etc

Local Illumination

Simplifying assumptions:

- ignore everything except eye, light, and object
  - basic version: no shadows, reflections, etc
  - but can support basic shadows/reflection

Local Illumination

Simplifying assumptions:

- ignore everything except eye, light, and object
  - basic version: no shadows, reflections, etc
  - but can support basic shadows/reflection
- only point lights
- only simple (diffuse & specular) materials

Global Illumination