Ray Tracing & Ray Casting

Realistic Graphics Inpsired by Nature

Ashrafur Rahman

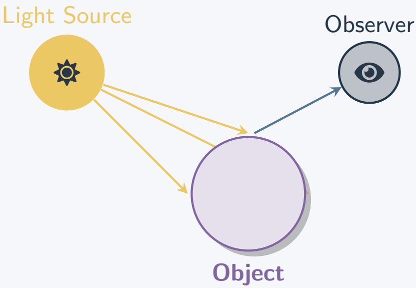
Adjunct Lecturer

Department of Computer Science
Bangladesh University of Engineering and Technology (BUET)

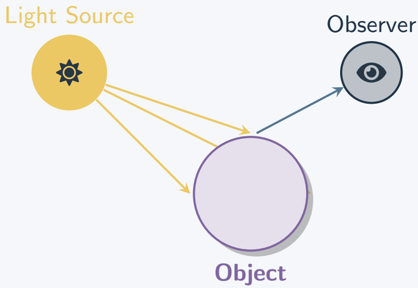
Index

The Story of Light

How Do We See?



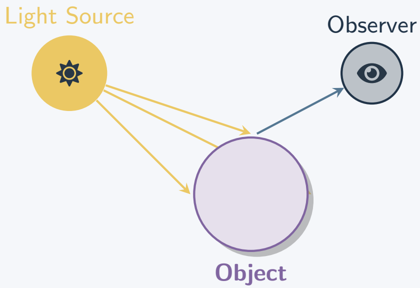
How Do We See?



Natural Process

1. Light travels from source
2. Light hits objects
3. Light bounces to our eyes
4. Our brain interprets the signal

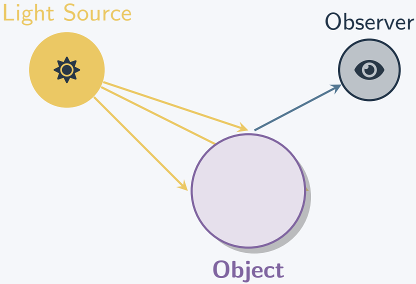
How Do We See?



Physical Process

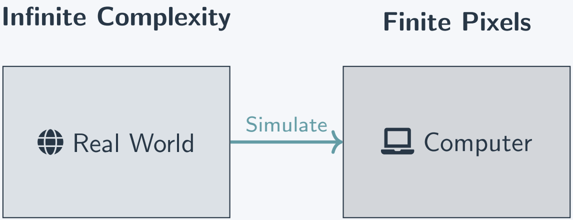
1. Photon is emitted from source
2. Photon hits objects
3. Part of the photon is reflected or absorbed
4. The reflected photons reach our eyes
5. The rods and cones in our retina detect the photons
6. Our brain interprets the signal
7. **Colour**: The wavelength of the photons
8. **Brightness**: The number of photons

How Do We See?



Question: How do we simulate this?

The Computer Graphics Challenge



Challenges:

- Infinite light rays/photons
- Complex physics
- High computational cost

Ray Casting: The Foundation

The Brilliant Insight

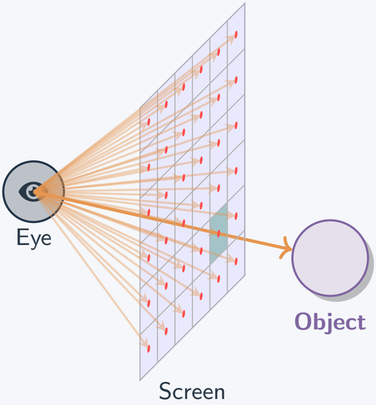
Reverse Engineering Vision

Instead of following light from sources...

Let's trace backwards from our eyes!

Why does this work?

- Most light never reaches our eyes
- Only trace rays that matter
- Much more efficient!



Ray Casting Algorithm

For each pixel

↓

Shoot ray from eye

↓

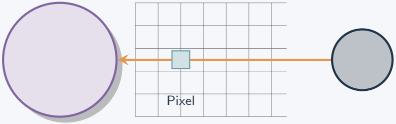
Find closest intersection

↓

Calculate color

↓

Set pixel color



The Mathematics of Rays

What is a Ray?

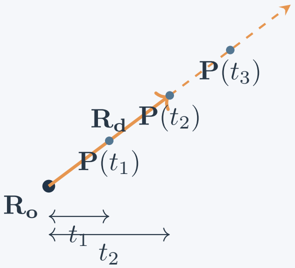
Ray Representation

A ray is defined by:

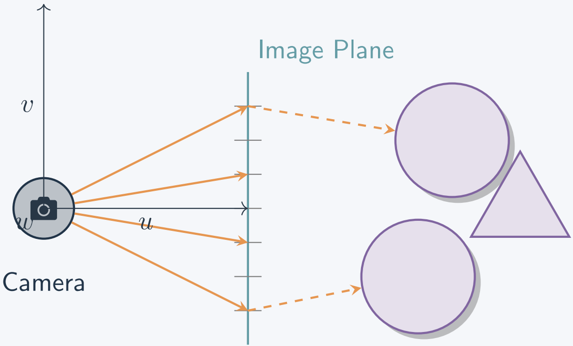
$$\mathbf{P}(t) = \mathbf{R_o} + t \cdot \mathbf{R_d} \quad (1)$$

where:

- $\mathbf{R_o}$ = Origin point
- $\mathbf{R_d}$ = Direction vector
- t = Parameter ($t \geq 0$)



Camera and Ray Generation



Camera Parameters

Camera Definition: Eye point $\mathbf{e}$, orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$, field of view, image dimensions

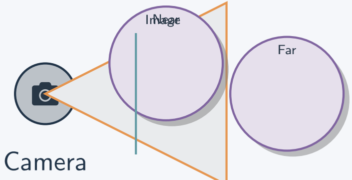
Camera Models in Ray Tracing

Why Camera Models Matter

The Camera's Role

The camera determines:

- **Field of view** - What we see
- **Perspective** - How objects appear
- **Ray generation** - Where rays start
- **Image formation** - Final rendering



Different camera models = Different visual effects!

The Pinhole Camera Model

Pinhole Camera

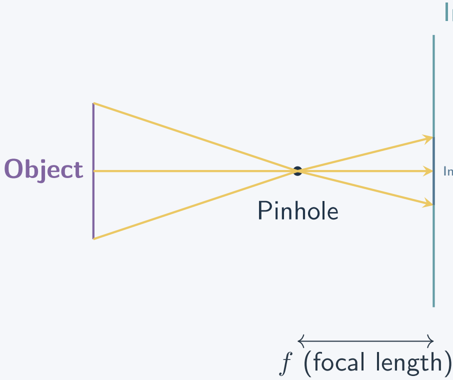
Key Properties:

- Point aperture (no lens)
- Perfect focus everywhere
- Linear perspective
- No depth of field

Ray Generation:

$$\mathbf{R_o} = \mathbf{eye} \tag{2}$$

$$\mathbf{R_d} = \mathbf{pixel} - \mathbf{eye} \tag{3}$$



Physical Reality

Real pinhole cameras exist! They create sharp images but require

Simplified Pinhole Camera

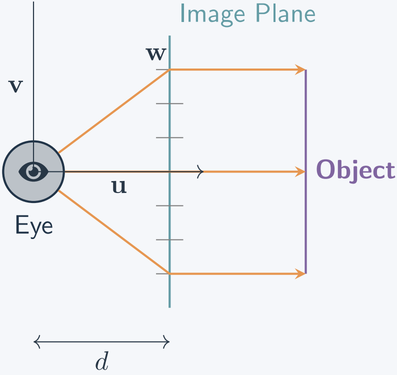
Simplification

Problem: Real pinhole creates inverted image
Solution: Place image plane in front!

$$\mathbf{pixel} = \mathbf{eye} + d \cdot \mathbf{w} + u \cdot \mathbf{u} + v \cdot \mathbf{v} \tag{4}$$

where:

- d = distance to image plane
- u, v = pixel coordinates
- $\mathbf{u}, \mathbf{v}, \mathbf{w}$ = camera basis



Advantage

View Frustum

Viewing Frustum

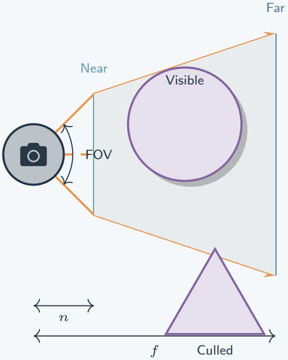
Definition: The 3D region visible to the camera

Boundaries:

- **Near plane** - Closest visible distance
- **Far plane** - Farthest visible distance
- **Left/Right** - Horizontal field of view
- **Top/Bottom** - Vertical field of view

Field of View:

$$\text{FOV} = 2\arctan\left(\frac{h}{2d}\right) \quad (5)$$



Orthographic Camera

Orthographic Projection

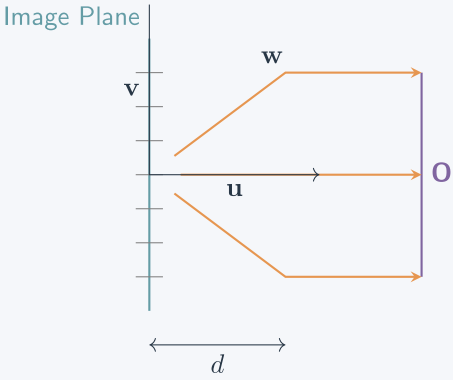
Key Properties:

- No perspective distortion
- Parallel projection rays
- Objects same size regardless of distance
- Infinite focal length

Ray Generation:

$$\mathbf{R_o} = \mathbf{pixel} \tag{6}$$

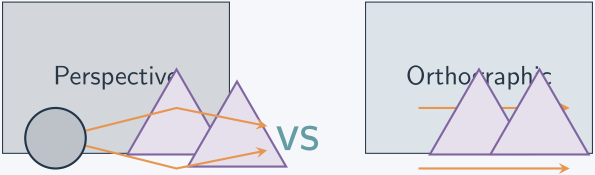
$$\mathbf{R_d} = \mathbf{w}\ \text{(constant)} \tag{7}$$



Applications

Technical drawings, CAD software, 2D games, architectural

Perspective vs Orthographic



Converging rays
Size $\propto$ 1/distance

Parallel rays
Constant size

When to use Perspective

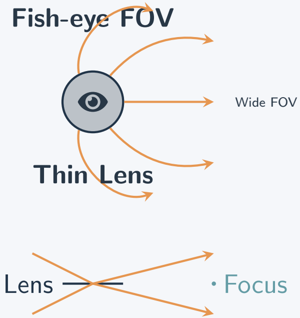
- Natural/realistic scenes
- Human vision simulation
- Games and films
- Depth perception important

When to use Orthographic

- Technical illustrations
- CAD/Engineering
- UI elements overlay
- Precise measurements

Fish-eye Camera

- Very wide field of view (¿180°)
- Non-linear distortion
- Curved ray paths
- Surveillance, VR applications

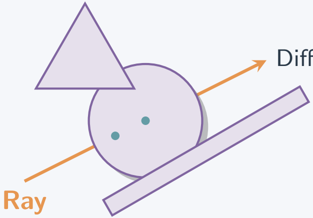


Thin Lens Camera

- Simulates real camera lens
- Depth of field effects
- Focal blur

Ray-Object Intersections

Finding Intersections



Key Objects:

- **Planes** - Linear equations
- **Spheres** - Quadratic equations
- **Triangles** - Barycentric coordinates
- **General Quadrics** - Polynomial solving

Challenge: Find the **closest** intersection efficiently!

Ray-Plane Intersection

Plane Equation

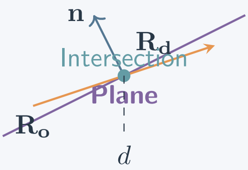
Implicit form:

$$\mathbf{n} \cdot \mathbf{P} + D = 0 \qquad (8)$$

Substituting ray equation:

$$\mathbf{n} \cdot (\mathbf{R_o} + t\mathbf{R_d}) + D = 0 \qquad (9)$$

$$t = -\frac{D + \mathbf{n} \cdot \mathbf{R_o}}{\mathbf{n} \cdot \mathbf{R_d}} \qquad (10)$$



Key Insight

Explicit ray equation meets **implicit** plane equation = Clean intersection formula!

Ray-Sphere Intersection

Sphere Equation

Implicit form (centered at origin):

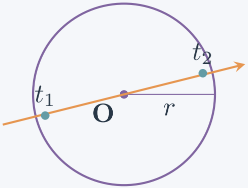
$$\mathbf{P} \cdot \mathbf{P} - r^2 = 0 \tag{11}$$

Substituting ray equation:

$$(\mathbf{R_o} + t\mathbf{R_d}) \cdot (\mathbf{R_o} + t\mathbf{R_d}) - r^2 = 0 \tag{12}$$

$$t^2 + 2(\mathbf{R_d} \cdot \mathbf{R_o})t + (\mathbf{R_o} \cdot \mathbf{R_o} - r^2) = 0 \tag{13}$$

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



Sphere

$\Delta > 0$: 2 roots
$\Delta = 0$: 1 root
$\Delta < 0$: no roots

Ray-Triangle Intersection

Barycentric Approach

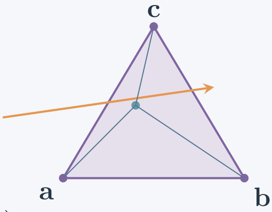
Triangle defined by vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$:

$$\mathbf{P}(\beta, \gamma) = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{14}$$

Set equal to ray equation:

$$\mathbf{R_o} + t\mathbf{R_d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{15}$$

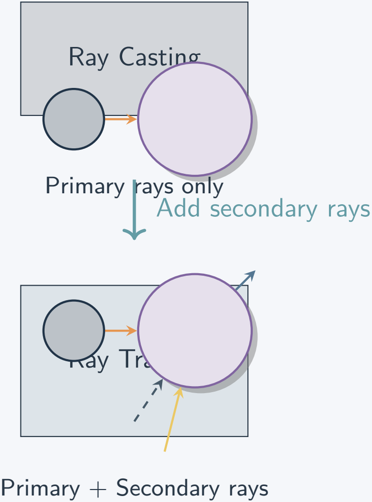
Solve 3×3 system for t, β, γ



Inside if:
$\beta \geq 0$
$\gamma \geq 0$
$\beta + \gamma \leq 1$

From Ray Casting to Ray Tracing

Ray Casting vs Ray Tracing

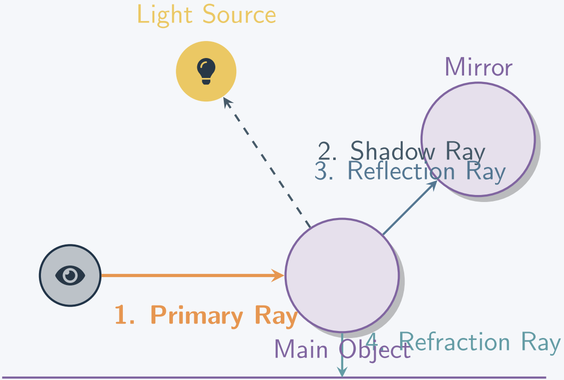


Ray Casting

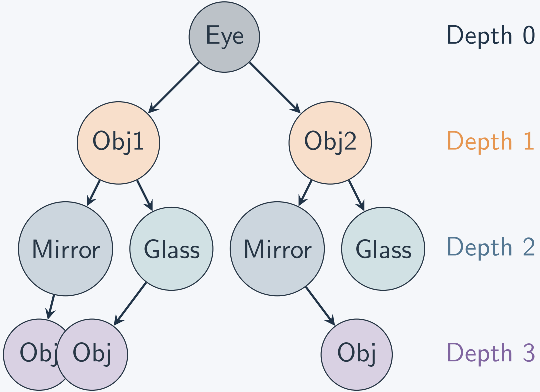
Ray Tracing

- Multiple ray types

Secondary Rays: The Magic Ingredients



Recursive Ray Tracing



Recursion Control

Base Cases: Maximum depth reached OR ray contribution becomes negligible

Advanced Ray Tracing Effects

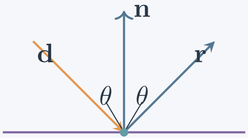
Mirror Reflections

Reflection Law

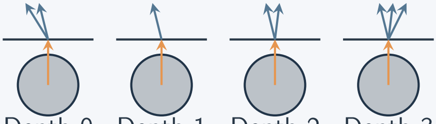
Given incident ray $\mathbf{d}$ and surface normal $\mathbf{n}$:

$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n} \qquad (16)$$

Physical principle:
Angle of incidence = Angle of reflection



Reflection Depth



Refraction and Snell's Law

Snell's Law

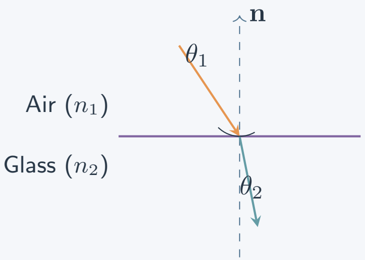
$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \quad (17)$$

where:

- $n_1, n_2 =$ refractive indices
- $\theta_1 =$ incident angle
- $\theta_2 =$ refracted angle

Examples:

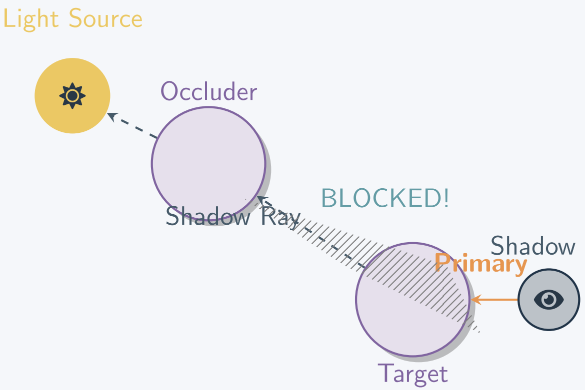
- Air: $n = 1.0$
- Water: $n = 1.33$
- Glass: $n = 1.5$



Total inte
reflection
$\theta_1 > \theta$

Physical Intuition

Shadows: The Absence of Light



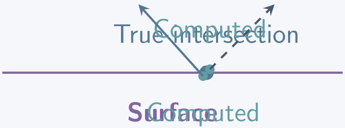
Shadow Ray Algorithm

For each intersection point:

1. Cast ray toward each light source
2. Check if ray intersects any object before reaching light

Implementation Challenges

The Floating Point Precision Problem



Problems:

- Self-intersection
- Incorrect shadows
- Ray escaping

The Evil Epsilon

Solution: Add small offset ε when starting secondary rays from surfaces

Challenge: Too small $\rightarrow$ still problems; Too large $\rightarrow$ visible artifacts

Performance Considerations

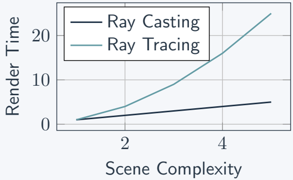
Computational Complexity

Basic ray tracing:

- $O(n \times m)$ where:
- $n =$ number of pixels
- $m =$ number of objects

With secondary rays:

- Exponential growth with depth
- Multiple rays per intersection

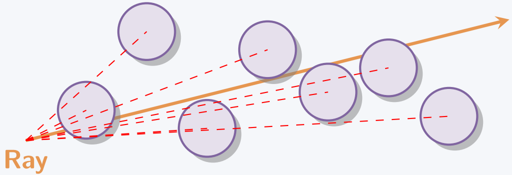


Optimization strategies:

- Spatial data structures
- Early ray termination
- Parallel processing

Acceleration Structures: Making Ray Tracing Fast

The Naive Approach Problem



Problem: Test every object for every ray!
Scene with 1M objects = 1M tests per ray

Computational Explosion

Complexity: For N objects and R rays $\rightarrow O(N \times R)$ intersection tests

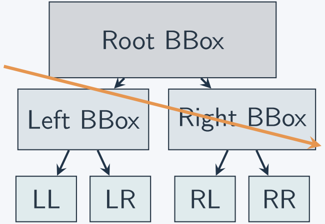
Real scenes: Millions of triangles, millions of rays $\rightarrow$ Billions of tests!

Bounding Volume Hierarchy (BVH)

The Big Idea

Divide and Conquer:

1. Group objects into bounding boxes
2. Build hierarchical tree structure
3. Test ray against boxes first
4. Only test objects in hit boxes



Ray tests hierarchy top-down

Key Benefits:

- $O(\log N)$ instead of $O(N)$
- Massive speedup for complex scenes

BVH Construction and Traversal

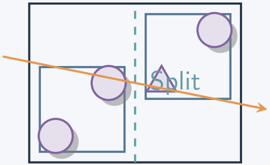
Construction Algorithm

Recursive subdivision:

1. Compute bounding box for all objects
2. Choose split axis (longest dimension)
3. Sort objects by centroid
4. Split into two groups
5. Recursively build subtrees

Split strategies:

- Median split
- Surface Area Heuristic (SAH)



Spatial subdivision

Traversal

Stack-based traversal: Test bounding boxes, push hit children onto stack

Hardware Ray Tracing Revolution

The Hardware Revolution

2018 2020 2021 2022+

NVIDIA RTX RTX 30 Series Vulkan RT RDNA3, RTX 40

From Software to Silicon

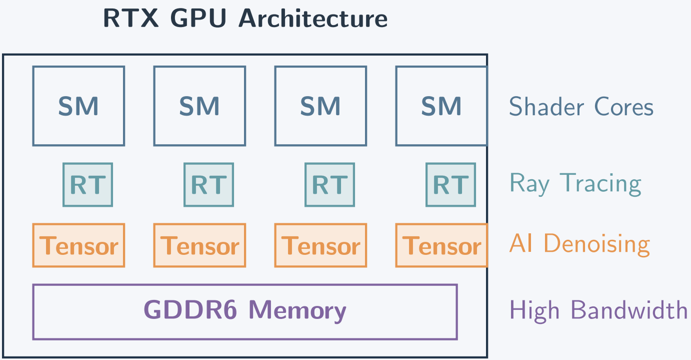
Hardware Features:

- Dedicated RT cores
- Hardware BVH traversal
- Triangle intersection units
- Tensor cores for denoising

Performance Impact:

- 10-100x speedup over compute shaders
- Real-time ray tracing in games
- Interactive path tracing
- AI-accelerated denoising

RTX Architecture Deep Dive



RT Core Functions

Hardware accelerated: BVH traversal, ray-triangle intersection, ray-box intersection

Result: Massive parallel ray processing with dedicated silicon

Modern Ray Tracing APIs

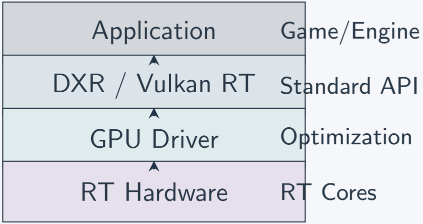
DirectX Raytracing (DXR)

Microsoft's API:

- Raytracing Pipeline State Objects
- Acceleration Structure builds
- Ray generation/intersection/hit shaders
- Widely adopted in games

Vulkan Ray Tracing

Cross-platform standard:

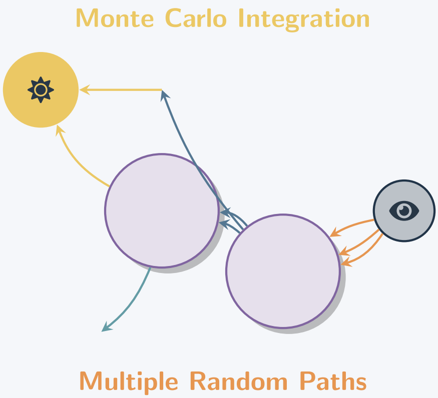


Key Innovation:

- Shader-driven ray tracing
- Flexible hit/miss handling
- Integration with rasterization

Path Tracing: The Ultimate Goal

Beyond Ray Tracing: Path Tracing



Ray Tracing Limitations

- Perfect mirrors only
- Direct illumination focus
- Limited global effects

Path Tracing Advantages

- Physically accurate lighting
- Global illumination

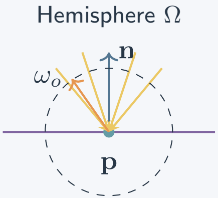
The Rendering Equation

The Holy Grail of Computer Graphics

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_\Omega f(\mathbf{p}, \omega_i, \omega_o) L_i(\mathbf{p}, \omega_i)(\mathbf{n} \cdot \omega_i) d\omega_i \tag{18}$$

Components:

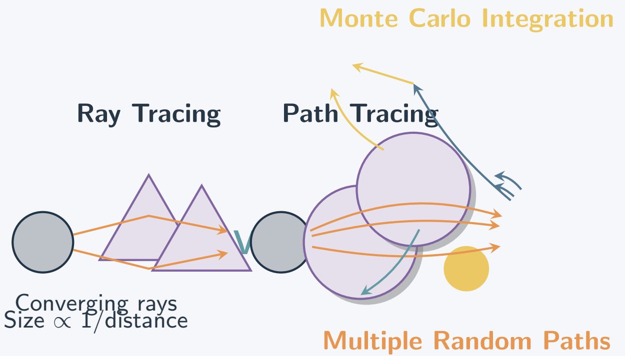
- L_o = Outgoing radiance
- L_e = Emitted light
- f = BRDF (material properties)
- L_i = Incoming radiance
- Ω = Hemisphere of directions



The Challenge

Integral: Infinite directions to sample → Monte Carlo approxima-

Path Tracing vs Ray Tracing



Ray Tracing Limitations

- Perfect mirrors only
- Direct illumination focus
- Limited global effects

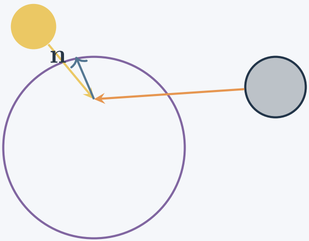
Path Tracing Advantages

- Physically accurate lighting
- Global illumination
- Soft shadows, caustics

The Art of Shading (Preview)

We've traced rays and found intersections...

Now what color should that pixel be?



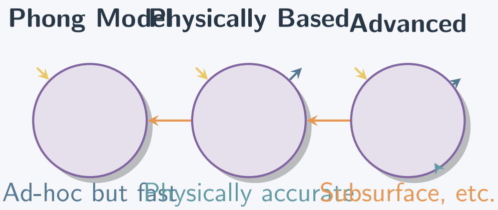
Shading determines:

- Surface appearance
- Material properties
- Light interaction
- Visual realism

The intersection is just the beginning

Next lesson: Deep dive into shading models!

Shading Models: A Sneak Peek



Phong/Blinn-Phong

Components:

- Ambient
- Diffuse
- Specular

Pro: Fast, sim-

Physically Based

Based on:

- Energy conservation
- Fresnel equations
- Microfacet theory

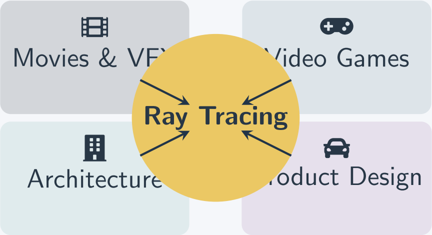
Advanced Models

Features:

- Subsurface scattering
- Volumetric effects
- Layered materials

Applications and Future

Ray Tracing in the Real World



Traditional (Offline):

- Movie rendering
- Architectural visualization
- Product design
- Scientific simulation

Modern (Real-time):

- RTX graphics cards
- Video games
- VR/AR applications
- Interactive design

The Future is Bright

Hardware Acceleration

Modern GPUs: Dedicated ray tracing cores, massive parallelization

Emerging Techniques:

- Machine learning denoising
- Hybrid rendering
- Path tracing
- Photon mapping

New Applications:

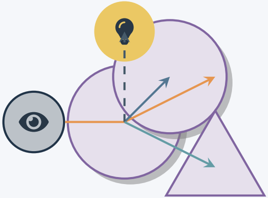
- Medical imaging
- Autonomous vehicles
- Metaverse platforms
- Scientific visualization

Ray tracing is becoming the future of computer graphics!

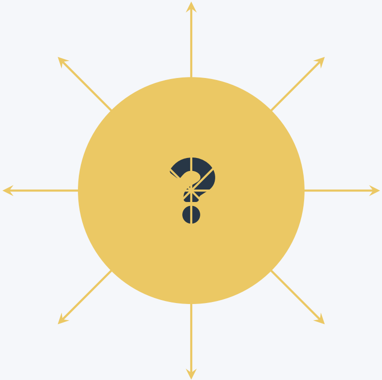
Wrapping Up

Key Takeaways

1. **Ray tracing simulates light transport** by reversing the natural process
2. **Mathematical foundation** involves solving intersection equations for different geometric primitives
3. **Secondary rays** enable realistic effects like reflections, refractions, and shadows
4. **Implementation challenges** include floating-point precision and performance optimization
5. **Real-world impact** spans from Hollywood movies to real-time gaming



Questions?



References & Further Reading

- Peter Shirley, Steve Marschner et al. *Fundamentals of Computer Graphics (4th Edition)*. CRC Press, 2016.
- Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd Edition)*. Morgan Kaufmann, 2016.
- Kevin Suffern. *Ray Tracing from the Ground Up*. A K Peters/CRC Press, 2007.
- MIT OpenCourseWare: 6.837 Computer Graphics. `https://ocw.mit.edu/courses/6-837-computer-graphics-fall-2012`
- Scratchapixel: Learn Computer Graphics Programming. `https://www.scratchapixel.com/`