Ray Tracing & Ray Casting

Realistic Graphics Inpsired by Nature

Ashrafur Rahman

Adjunct Lecturer

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

Index

Motivation

Why Learn This?

Why Learn This?

Elsa's Castle in Frozen

Why Learn This?

Elsa's Castle in Frozen

Cyberpunk 2077 with RTX

- **Realistic graphics** of your favourite animated movies are the result of ground-breaking work in Ray Tracing by studios like Disney, Pixar, and DreamWorks. Do you know these films take years to render? 30 hours per frame!

Why Learn This?

Elsa's Castle in Frozen

Cyberpunk 2077 with RTX

- **Realistic graphics** of your favourite animated movies are the result of ground-breaking work in Ray Tracing by studios like Disney, Pixar, and DreamWorks. Do you know these films take years to render? 30 hours per frame!
- Lately, **RTX** is all the rage in gaming. New titles boast ray-tracing effects in real-time, not 30 hours per frame!

Why Learn This?

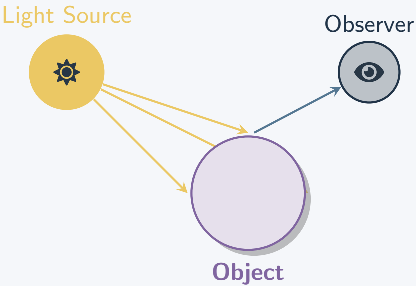
Elsa's Castle in Frozen

Cyberpunk 2077 with RTX

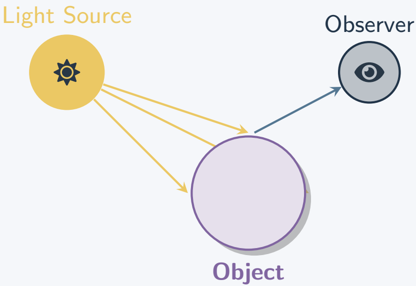
- **Realistic graphics** of your favourite animated movies are the result of ground-breaking work in Ray Tracing by studios like Disney, Pixar, and DreamWorks. Do you know these films take years to render? 30 hours per frame!
- Lately, **RTX** is all the rage in gaming. New titles boast ray-tracing effects in real-time, not 30 hours per frame!
- It's fun! You will know when you create your first ray-traced image!

The Story of Light

How Do We See?



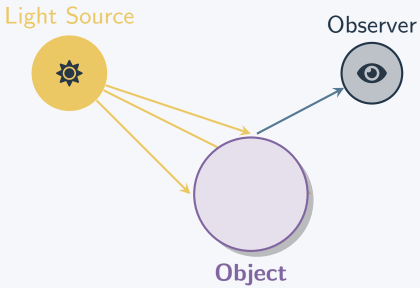
How Do We See?



Natural Process

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

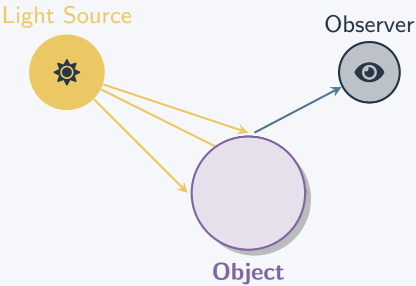
How Do We See?



Physical Process

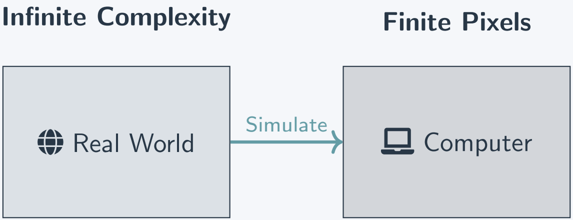
1. Photon is emitted from source
2. Photon hits objects
3. Part of the photon is reflected or absorbed
4. The reflected photons reach our eyes
5. The rods and cones in our retina detect the photons
6. Our brain interprets the signal
7. **Colour**: The wavelength of the photons
8. **Brightness**: The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

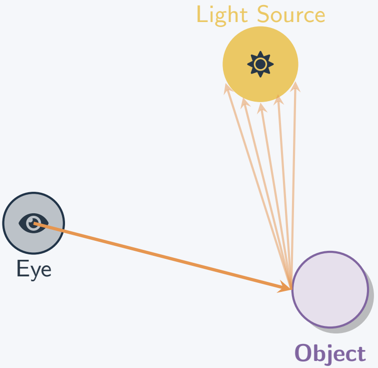
The Key Insight

1. Reverse Engineering

Instead of following light rays from light sources —

Let's trace backwards!

Shoot rays from the eye, find where it hits and find out how much light reaches there.



This is the opposite of what happens in reality. **Why does this work?**

The Key Insight

1. Reverse Engineering

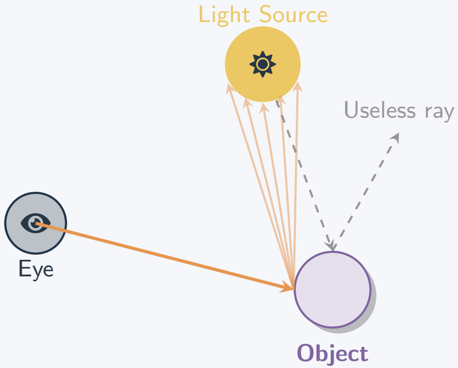
Instead of following light rays from light sources —

Let's trace backwards!

Shoot rays from the eye, find where it hits and find out how much light reaches there.

This is the opposite of what happens in reality. **Why does this work?**

- Most light never reaches our eyes

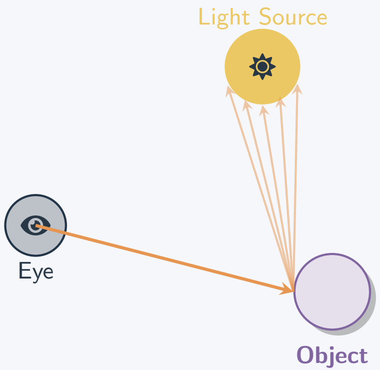


The Key Insight

1. Reverse Engineering

Instead of following light rays from light sources —

Let's trace backwards!
Shoot rays from the eye, find where it hits and find out how much light reaches there.



This is the opposite of what happens in reality. **Why does this work?**

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

From Infinite Rays to Finite Pixels

2. Cutting Costs

Instead of tracing infinite rays —

Trace one ray per pixel.

This comes with little tradeoff, because:

From Infinite Rays to Finite Pixels

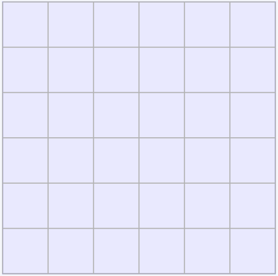
2. Cutting Costs

Instead of tracing infinite rays —

Trace one ray per pixel.

This comes with little tradeoff, because:

- An image is just a grid of pixels



From Infinite Rays to Finite Pixels

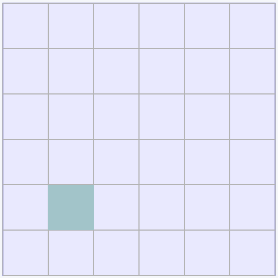
2. Cutting Costs

Instead of tracing infinite rays —

Trace one ray per pixel.

This comes with little tradeoff, because:

- An image is just a grid of pixels
- Each pixel can only be of one color



From Infinite Rays to Finite Pixels

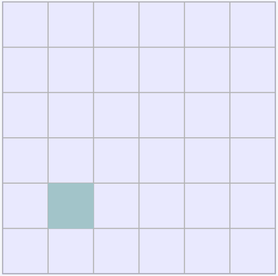
2. Cutting Costs

Instead of tracing infinite rays —

Trace one ray per pixel.

This comes with little tradeoff, because:

- An image is just a grid of pixels
- Each pixel can only be of one color
- In the end, we just need to know the *color of each pixel*



From Infinite Rays to Finite Pixels

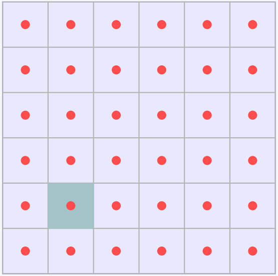
2. Cutting Costs

Instead of tracing infinite rays —

Trace one ray per pixel.

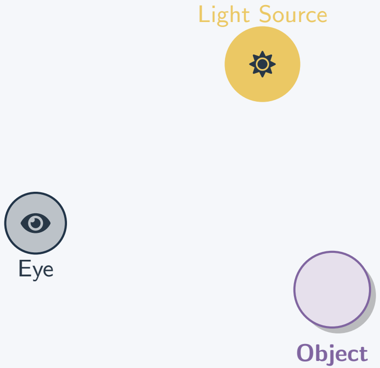
This comes with little tradeoff, because:

- An image is just a grid of pixels
- Each pixel can only be of one color
- In the end, we just need to know the *color of each pixel*
- Hence, one ray from the mid-point of each pixel should be a good approximation*



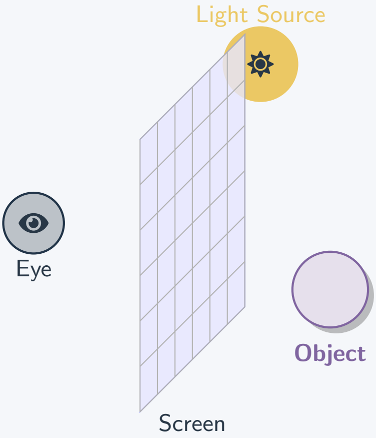
* We will discuss more advanced techniques later that improve quality

The Full Picture



The Full Picture

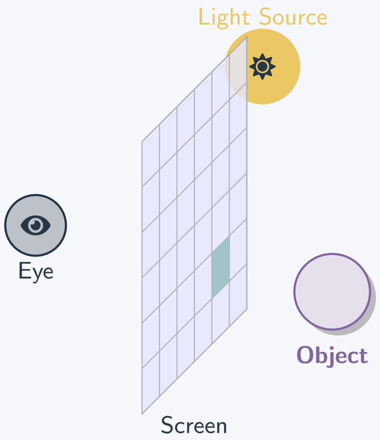
Place screen in front of eye



The Full Picture

Place screen in front of eye

For each pixel



The Full Picture

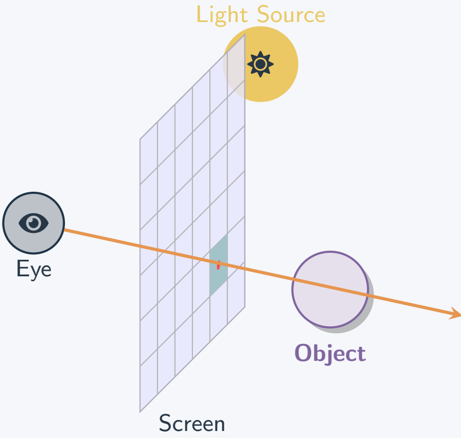
Place screen in front of eye

↓

For each pixel

↓

Shoot ray from eye



The Full Picture

Place screen in front of eye

↓

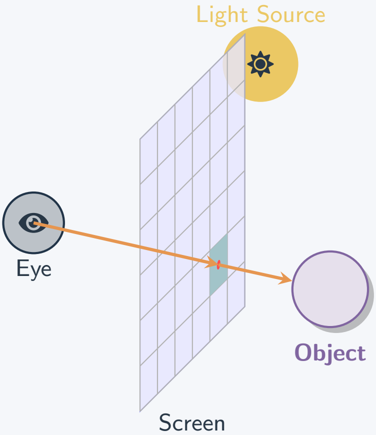
For each pixel

↓

Shoot ray from eye

↓

Find closest intersection



The Full Picture

Place screen in front of eye

↓

For each pixel

↓

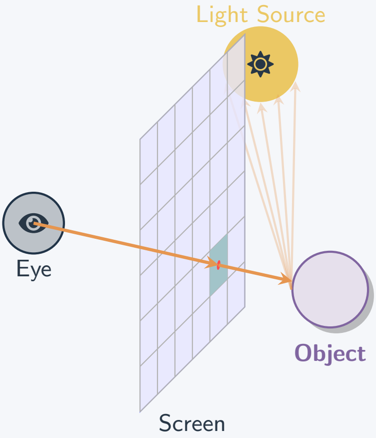
Shoot ray from eye

↓

Find closest intersection

↓

Calculate color*



The Full Picture

Place screen in front of eye

↓

For each pixel

↓

Shoot ray from eye

↓

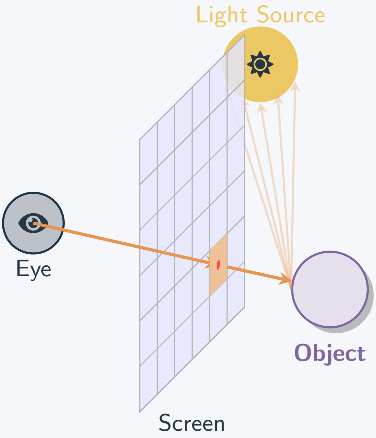
Find closest intersection

↓

Calculate color*

↓

Set pixel color



The Full Picture

Place screen in front of eye

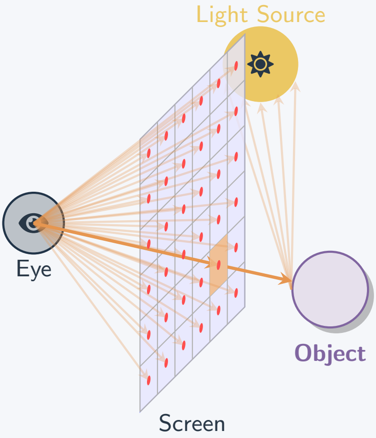
For each pixel

Shoot ray from eye

Find closest intersection

Calculate color*

Set pixel color



Rays and Cameras

What is a Ray?

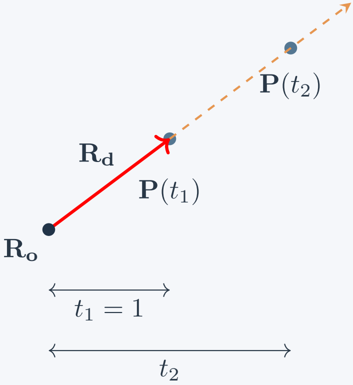
Ray Representation

A ray is defined by:

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

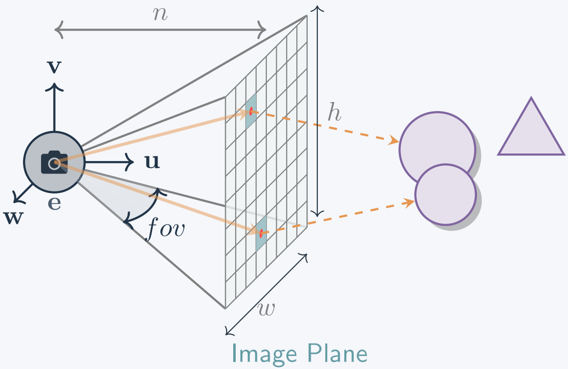
where:

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



Check out here on desmos.

Camera Representation



Camera Description

Camera position $\mathbf{e}$, orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$,
field of view fov, distance to near plane n,
image dimensions $(w \times h)$.

The Pinhole Camera Model

Pinhole Camera

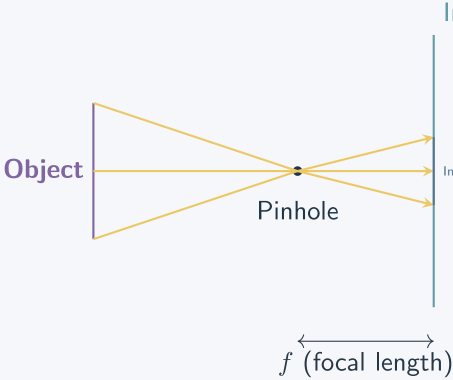
Key Properties:

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

Ray Generation:

$$\mathbf{R_o} = \mathbf{eye} \quad (2)$$

$$\mathbf{R_d} = \mathbf{pixel} - \mathbf{eye} \quad (3)$$



Physical Reality

Real pinhole cameras exist! They create sharp images but require

Simplified Pinhole Camera

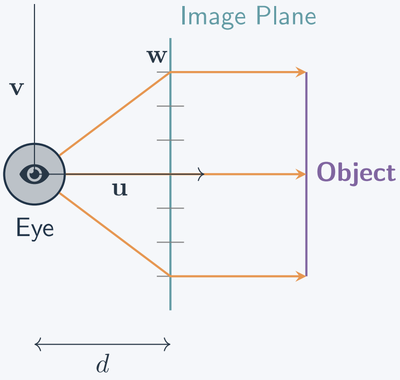
Simplification

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

Orthographic Camera

Orthographic Projection

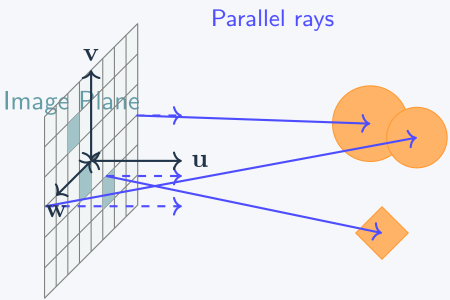
Key Properties:

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

Ray Generation:

$$\mathbf{R_o} = \mathbf{pixel} \tag{5}$$

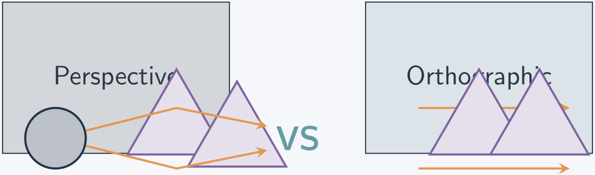
$$\mathbf{R_d} = \mathbf{w} \text{ (constant)} \tag{6}$$



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

Thin Lens Camera: Fundamentals

Gaussian Lens Equation

Fundamental relationship:

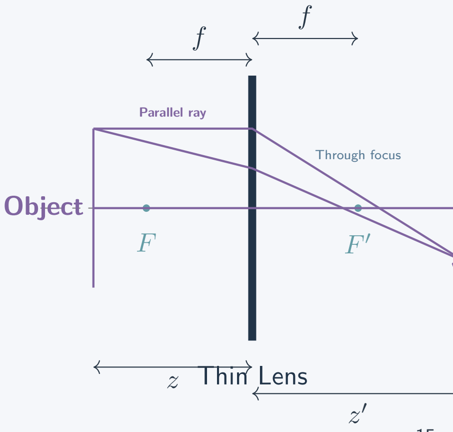
$$\frac{1}{f} = \frac{1}{z} + \frac{1}{z'} \qquad (7)$$

Where:

- f = focal length of lens
- z = object distance from lens
- z' = image distance from lens

Key Properties:

- Objects at focal plane are in perfect focus
- Other distances create



Depth of Field and Circle of Confusion

Circle of Confusion

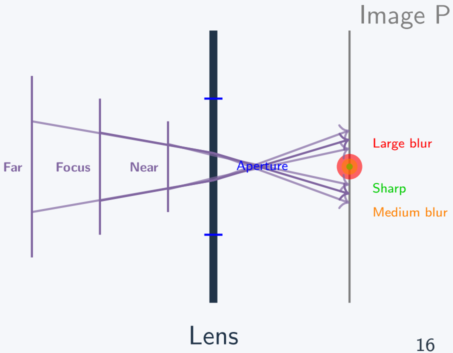
For objects not at focal distance:

$$c = \frac{A}{z'} \left| z'_{focus} - z' \right| \quad (8)$$

Where:

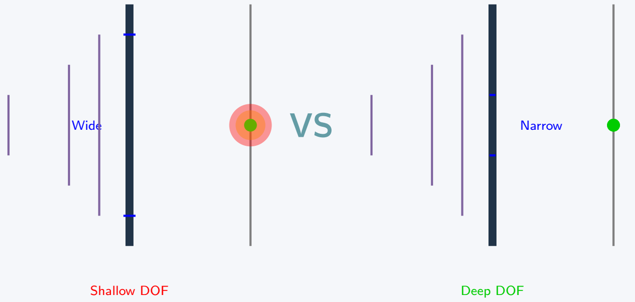
- $c =$ circle of confusion diameter
- $A =$ aperture diameter
- $z' =$ image distance for object
- $z'_{focus} =$ image distance for focus

Depth of Field:



Aperture Effects on Depth of Field

Large Aperture (f/1.4) **Small Aperture (f/11)**



Large Aperture

- More light gathering
- Faster shutter speeds

Small Aperture

- Less light gathering
- Slower shutter speeds

Thin Lens Ray Generation

Ray Sampling Process

1. Sample pixel position (x, y)

2. Sample lens position:

$$(u, v) \sim \text{Uniform disk} \quad (9)$$

$$\mathbf{p}_{lens} = (u \cdot r, v \cdot r, 0) \quad (10)$$

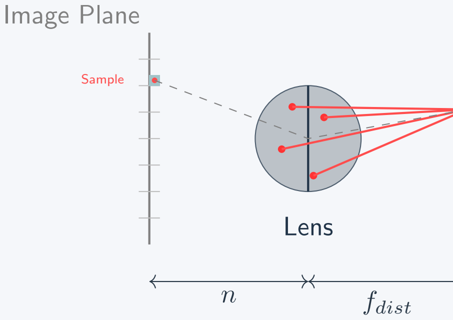
3. Compute focal point:

$$\mathbf{p}_{focus} = \mathbf{p}_{pixel} \cdot \frac{f_{dist}}{n} \quad (11)$$

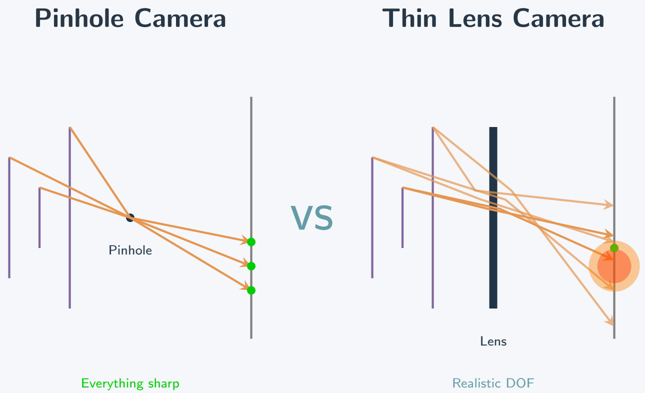
4. Ray from lens to focal point:

$$\mathbf{R_o} = \mathbf{p}_{lens} \quad (12)$$

$$\mathbf{R_d} = \mathbf{p}_{focus} - \mathbf{p}_{lens} \quad (13)$$



Pinhole vs Thin Lens Comparison



Pinhole Advantages

- Everything in focus
- Simple ray generation

Thin Lens Advantages

- Realistic camera behavior

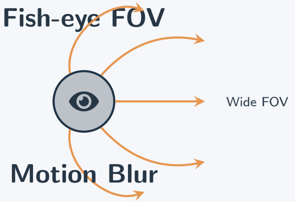
Other Camera Types

Fish-eye Camera

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

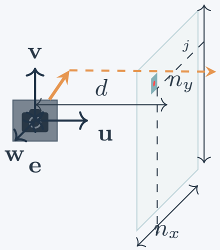
Environment Camera

- 360° panoramic view
- Spherical or cylindrical
- HDRI environment maps



Ray Generation

Ray Generation Mathematics



Ray Equation

For pixel (i, j):

$$s = \frac{i + 0.5}{n_x} \qquad (14)$$

$$t = \frac{j + 0.5}{n_y} \qquad (15)$$

Ray direction:

$$\mathbf{d} = (s - 0.5) \cdot \text{FOV} \cdot \mathbf{u} \qquad (16)$$

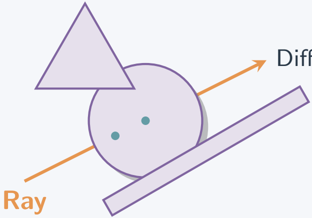
$$+ (t - 0.5) \cdot \text{FOV} \cdot \mathbf{v} \qquad (17)$$

$$+ d \cdot \mathbf{w} \qquad (18)$$

Parametric ray:

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

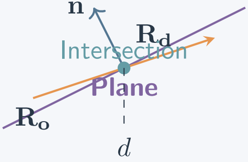
Implicit form:

$$\mathbf{n} \cdot \mathbf{P} + D = 0 \qquad (20)$$

Substituting ray equation:

$$\mathbf{n} \cdot (\mathbf{R_o} + t\mathbf{R_d}) + D = 0 \qquad (21)$$

$$t = -\frac{D + \mathbf{n} \cdot \mathbf{R_o}}{\mathbf{n} \cdot \mathbf{R_d}} \qquad (22)$$



Key Insight

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

Ray-Sphere Intersection

Sphere Equation

Implicit form (centered at origin):

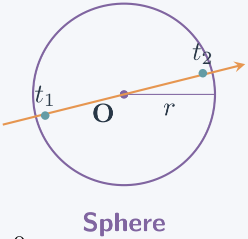
$$\mathbf{P} \cdot \mathbf{P} - r^2 = 0 \tag{23}$$

Substituting ray equation:

$$(\mathbf{R_o} + t\mathbf{R_d}) \cdot (\mathbf{R_o} + t\mathbf{R_d}) - r^2 = 0 \tag{24}$$

$$t^2 + 2(\mathbf{R_d} \cdot \mathbf{R_o})t + (\mathbf{R_o} \cdot \mathbf{R_o} - r^2) = 0 \tag{25}$$

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



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

Ray-Triangle Intersection

Barycentric Approach

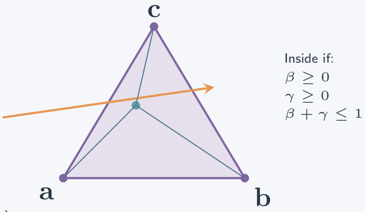
Triangle defined by vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$:

$$\mathbf{P}(\beta, \gamma) = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{26}$$

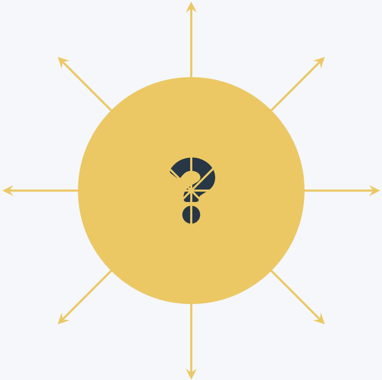
Set equal to ray equation:

$$\mathbf{R_o} + t\mathbf{R_d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{27}$$

Solve 3×3 system for t, β, γ



Questions?



References & Further Reading

- Peter Shirley and Steve Marschner et al. *Fundamentals of Computer Graphics (4th Edition)*. CRC Press, 2016.
  Available as PDF
- Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (4th Edition)*. Morgan Kaufmann, 2023.
  Availabe online
- Peter Shirley. *Ray Tracing in One Weekend*. Self-published, 2016–2020.
  Project Website
- MIT OpenCourseWare: 6.837 Computer Graphics.
  ocw.mit.edu/6-837
- Scratchapixel: Learn Computer Graphics Programming.
  scratchapixel.com