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?

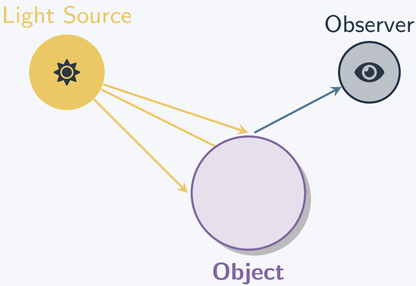
Elsa's Castle in Frozen

Cyberpunk 2077 with RTX

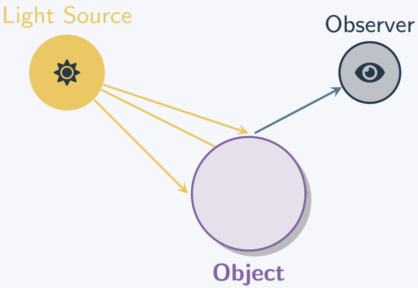
- **Realistic graphics** of your favourite animated movies are the result of ground-breaking work in Ray Tracing by studios like Disney, Pixar, and DreamWorks. Do you know these films take years to render? 30 hours per frame!
- Lately, **RTX** is all the rage in gaming. New titles boast ray-tracing effects 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?



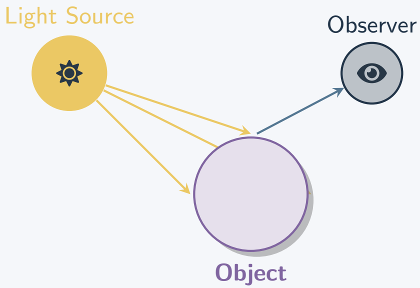
How Do We See?



Natural Process

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

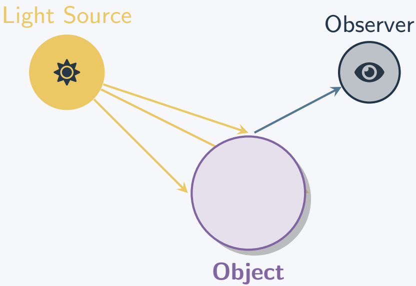
How Do We See?



Physical Process

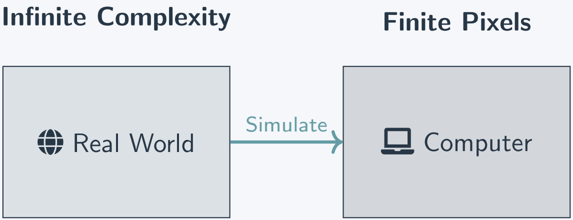
1. Photon is emitted from source
2. Photon hits objects
3. Part of the photon is reflected or absorbed
4. The reflected photons reach our eyes
5. The rods and cones in our retina detect the photons
6. Our brain interprets the signal
7. **Colour**: The wavelength of the photons
8. **Brightness**: The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

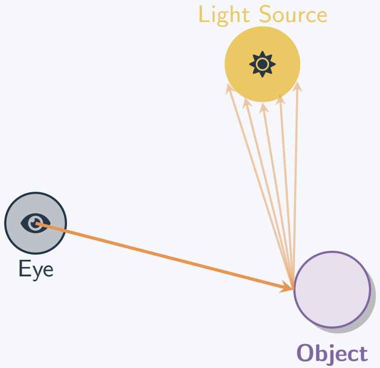
The Key Insight

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?**

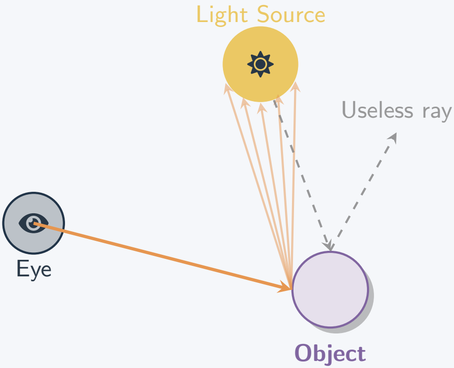


The Key Insight

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

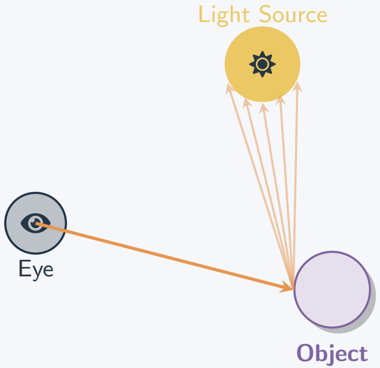
1. Reverse Engineering

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

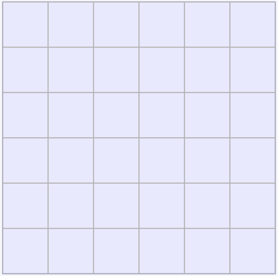
2. Cutting Costs

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

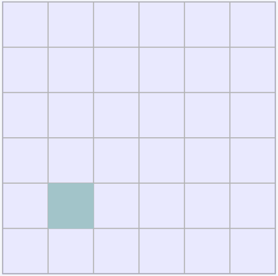
2. Cutting Costs

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

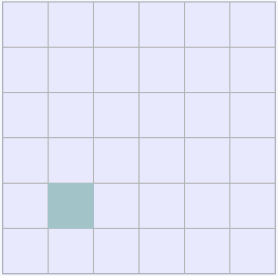
2. Cutting Costs

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*

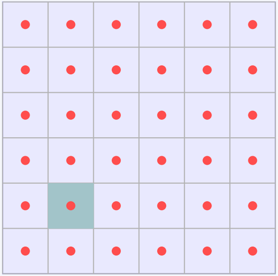


From Infinite Rays to Finite Pixels

2. Cutting Costs

Instead of tracing infinite rays —

Trace one ray per pixel.

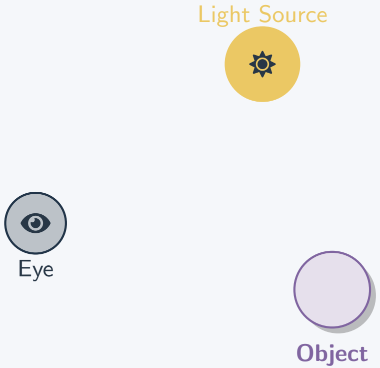


This comes with little tradeoff, because:

- 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*

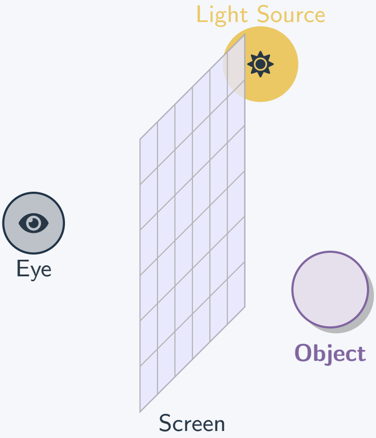
* We will discuss more advanced techniques later that improve quality

The Full Picture



The Full Picture

Place screen in front of eye

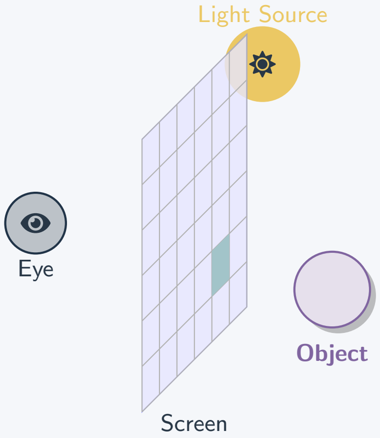


The Full Picture

Place screen in front of eye

↓

For each pixel



The Full Picture

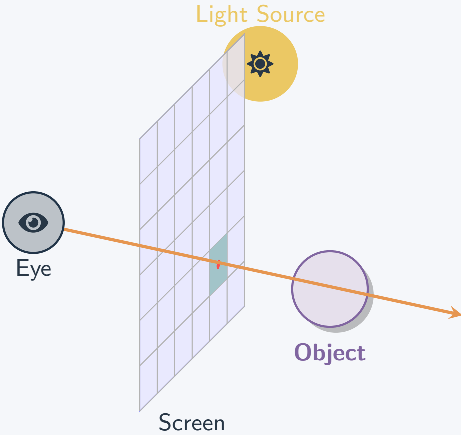
Place screen in front of eye

↓

For each pixel

↓

Shoot ray from eye



The Full Picture

Place screen in front of eye

↓

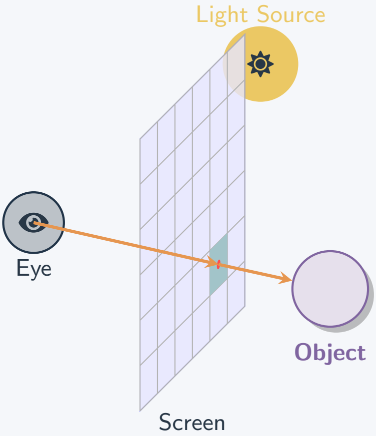
For each pixel

↓

Shoot ray from eye

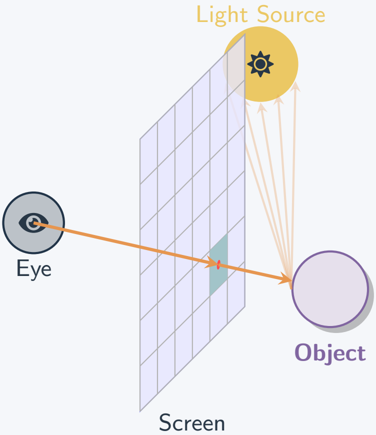
↓

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

↓

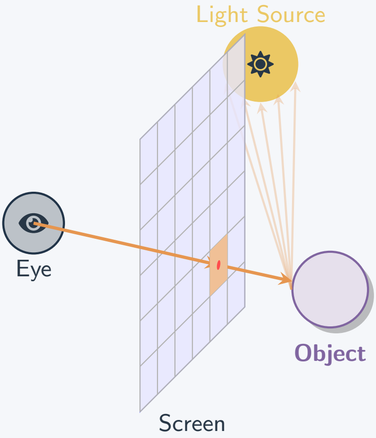
Find closest intersection

↓

Calculate color*

↓

Set pixel color



The Full Picture

Place screen in front of eye

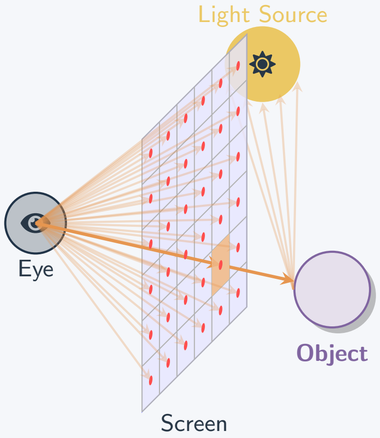
For each pixel

Shoot ray from eye

Find closest intersection

Calculate color*

Set pixel color



Mathematics of Rays

What is a Ray?

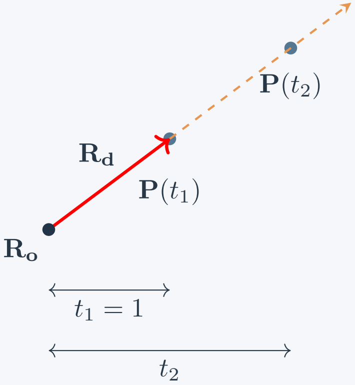
Ray Representation

A ray is defined by:

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

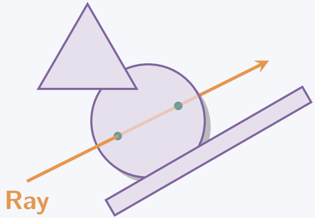
where:

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



Check out here on desmos.

Finding Intersections



Key Objects:

- **Planes**
- **Spheres**
- **Triangles**
- **AABB (Bounding Boxes)**
- **General Quadrics**

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

3D Plane Representation

Plane Definition

A plane is defined by:

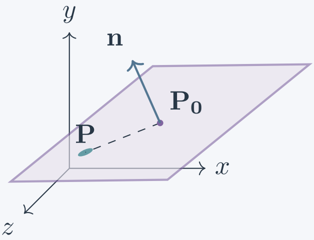
- Point $\mathbf{P_0} = (x_0, y_0, z_0)$ on plane
- Normal vector $\mathbf{n} = (A, B, C)$

Implicit equation:

$$\mathbf{n} \cdot (\mathbf{P} - \mathbf{P_0}) = 0$$

$\boxed{\mathbf{n} \cdot \mathbf{P} + D = 0}$ where $D = -\mathbf{n} \cdot \mathbf{P_0}$

$$\boxed{Ax + By + Cz + D = 0}$$



3D Plane Representation

Plane Definition

A plane is defined by:

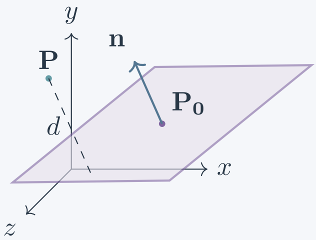
- Point $\mathbf{P_0} = (x_0, y_0, z_0)$ on plane
- Normal vector $\mathbf{n} = (A, B, C)$

Implicit equation:

$$\mathbf{n} \cdot (\mathbf{P} - \mathbf{P_0}) = 0$$

$\boxed{\mathbf{n} \cdot \mathbf{P} + D = 0}$ where $D = -\mathbf{n} \cdot \mathbf{P_0}$

$$\boxed{Ax + By + Cz + D = 0}$$



Point-Plane Distance

If $\mathbf{n}$ is normalized: $d = \mathbf{n} \cdot \mathbf{P} + D = \mathbf{n} \cdot (\mathbf{P} - \mathbf{P_0})$

Signed distance: $d > 0$ (front), $d < 0$ (back), $d = 0$ (on plane)

Ray-Plane Intersection

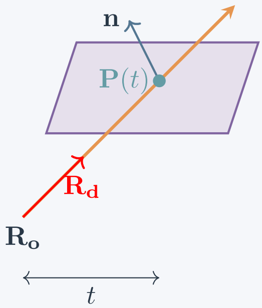
Intersection Method

Step 1: Substitute ray into equation

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

Step 2: Solve for parameter t

$$\boxed{t = -\frac{D + \mathbf{n} \cdot \mathbf{R_o}}{\mathbf{n} \cdot \mathbf{R_d}}}$$



Ray-Plane Intersection

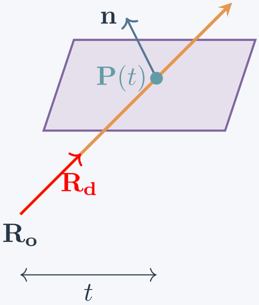
Intersection Method

Step 1: Substitute ray into equation

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

Step 2: Solve for parameter t

$$\boxed{t = -\frac{D + \mathbf{n} \cdot \mathbf{R_o}}{\mathbf{n} \cdot \mathbf{R_d}}}$$



Cases

- If $\mathbf{n} \cdot \mathbf{R_d} = 0$: Ray parallel to plane (0 or infinite)
- If $\mathbf{n} \cdot \mathbf{R_d} < 0$: Ray hits front face
- If $\mathbf{n} \cdot \mathbf{R_d} > 0$: Ray hits back face

Additional Checks

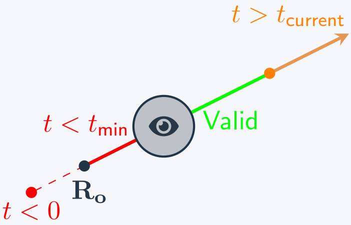
Validation Rules

After computing t, verify:

1. **Behind check:** $t > t_{\text{min}}$
2. **Closest check:** $t < t_{\text{current}}$
3. **Valid range:** $t \geq 0$

Where:

- t_{min}: Minimum ray distance (not behind eye/screen)
- t_{current}: Distance to closest intersection so far



Ray-Triangle Intersection Overview

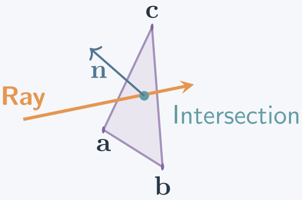
Two Main Approaches

Method 1: Two-Step Process

1. Ray-plane intersection
2. Inside/outside triangle test

Method 2: Direct Barycentric

1. Set up 3×3 linear system
2. Solve for t, β, γ simultaneously



What Are Barycentric Coordinates?

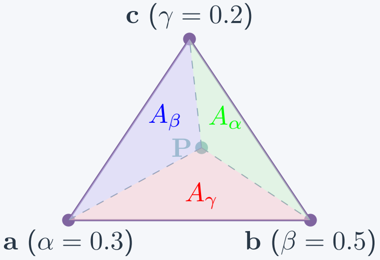
Barycentric Definition

Any point P in the triangle's plane:

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

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

Check out the Desmos demo.



What Are Barycentric Coordinates?

Barycentric Definition

Any point P in the triangle's plane:

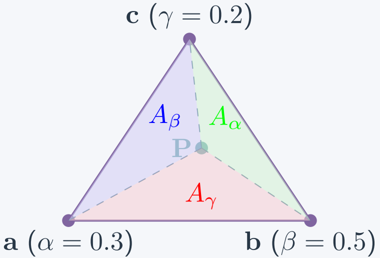
$$\mathbf{P}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

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

Physical Interpretation:

- α, β, γ are *weights*
- P is the *center of mass*
- Also called *barycenter*

Check out the Desmos demo.



What Are Barycentric Coordinates?

Barycentric Definition

Any point P in the triangle's plane:

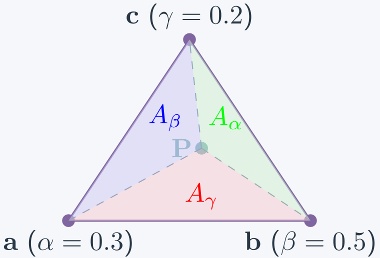
$$\mathbf{P}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

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

Physical Interpretation:

- α, β, γ are *weights*
- P is the *center of mass*
- Also called *barycenter*

Check out the Desmos demo.



Area Relationship

$\alpha = \frac{A_\alpha}{A_{\text{total}}}$, $\beta = \frac{A_\beta}{A_{\text{total}}}$,
$\gamma = \frac{A_\gamma}{A_{\text{total}}}$

Barycentric Coordinates: Inside vs Outside

Triangle Interior Test

Point P is **inside** triangle if:

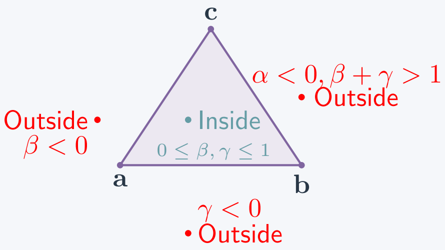
$$\alpha, \beta, \gamma \geq 0$$

Since $\alpha + \beta + \gamma = 1$, we can rewrite as:

$$\beta \geq 0$$

$$\gamma \geq 0$$

$$\alpha \geq 0 \text{ or } \beta + \gamma \leq 1$$



Barycentric Coordinates: Inside vs Outside

Triangle Interior Test

Point P is **inside** triangle if:

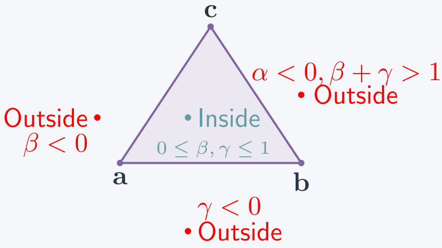
$$\alpha, \beta, \gamma \geq 0$$

Since $\alpha + \beta + \gamma = 1$, we can rewrite as:

$$\beta \geq 0$$

$$\gamma \geq 0$$

$$\alpha \geq 0 \text{ or } \beta + \gamma \leq 1$$



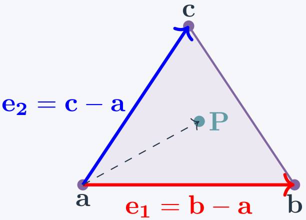
Insight

Barycentric coordinates doesn't just tell us if a point is inside a triangle, but also it's position with respect to other vertices.

Barycentric Coordinates: Derivation

Key Idea

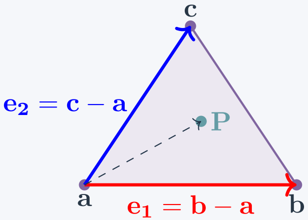
- The sides $\mathbf{e_1} = \mathbf{b} - \mathbf{a}$ and $\mathbf{e_2} = \mathbf{c} - \mathbf{a}$ are linearly independent vectors on the triangle's plane.



Barycentric Coordinates: Derivation

Key Idea

- The sides $\mathbf{e_1} = \mathbf{b} - \mathbf{a}$ and $\mathbf{e_2} = \mathbf{c} - \mathbf{a}$ are linearly independent vectors on the triangle's plane.
- Therefore, any vector in the triangle's plane (e.g. $\mathbf{P} - \mathbf{a}$) can be expressed as a linear combination of these vectors.



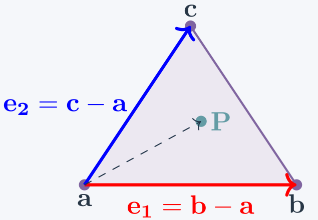
Barycentric Coordinates: Derivation

Key Idea

- The sides $\mathbf{e_1} = \mathbf{b} - \mathbf{a}$ and $\mathbf{e_2} = \mathbf{c} - \mathbf{a}$ are linearly independent vectors on the triangle's plane.
- Therefore, any vector in the triangle's plane (e.g. $\mathbf{P} - \mathbf{a}$) can be expressed as a linear combination of these vectors.
- We can express $\mathbf{P}$ as:

$$\begin{aligned}\mathbf{P} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

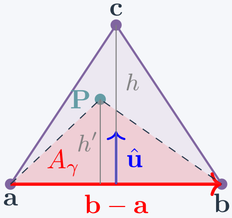
Where $\alpha = 1 - \beta - \gamma$.



Barycentric Coordinates: Derivation

Area Interpretation

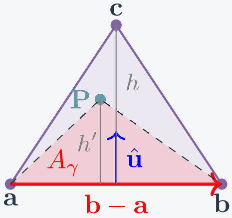
- Let $\hat{\mathbf{u}}$ be an unit vector in the direction of the altitude towards C.



Barycentric Coordinates: Derivation

Area Interpretation

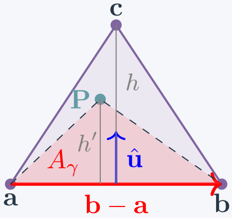
- Let $\hat{\mathbf{u}}$ be an unit vector in the direction of the altitude towards C.
- The height of the triangle is $h = \hat{\mathbf{u}} \cdot (\mathbf{c} - \mathbf{a})$ (projection).



Barycentric Coordinates: Derivation

Area Interpretation

- Let $\hat{\mathbf{u}}$ be an unit vector in the direction of the altitude towards C.
- The height of the triangle is $h = \hat{\mathbf{u}} \cdot (\mathbf{c} - \mathbf{a})$ (projection).
- The height of the shaded triangle is $h' = \hat{\mathbf{u}} \cdot (\mathbf{P} - \mathbf{a})$.

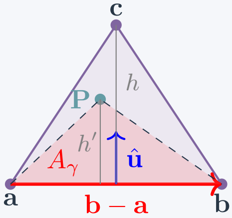


Barycentric Coordinates: Derivation

Area Interpretation

- Let $\hat{\mathbf{u}}$ be an unit vector in the direction of the altitude towards C.
- The height of the triangle is $h = \hat{\mathbf{u}} \cdot (\mathbf{c} - \mathbf{a})$ (projection).
- The height of the shaded triangle is $h' = \hat{\mathbf{u}} \cdot (\mathbf{P} - \mathbf{a})$.
- Hence,

$$\begin{aligned} A_\gamma &= \frac{1}{2} \cdot h' \cdot |\mathbf{b} - \mathbf{a}| \\ &= \frac{1}{2} \cdot (\hat{\mathbf{u}} \cdot (\mathbf{P} - \mathbf{a})) \cdot |\mathbf{b} - \mathbf{a}| \\ &= \frac{1}{2} \cdot \gamma\, (\hat{\mathbf{u}} \cdot (\mathbf{c} - \mathbf{a})) \cdot |\mathbf{b} - \mathbf{a}| \\ &= \gamma \frac{1}{2} \cdot h \cdot |\mathbf{b} - \mathbf{a}| = \gamma A_{\text{total}} \end{aligned}$$

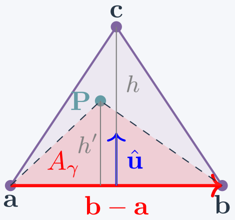


Barycentric Coordinates: Derivation

Area Interpretation

- Let $\hat{\mathbf{u}}$ be an unit vector in the direction of the altitude towards C.
- The height of the triangle is $h = \hat{\mathbf{u}} \cdot (\mathbf{c} - \mathbf{a})$ (projection).
- The height of the shaded triangle is $h' = \hat{\mathbf{u}} \cdot (\mathbf{P} - \mathbf{a})$.
- Hence,

$$\begin{aligned} A_\gamma &= \frac{1}{2} \cdot h' \cdot |\mathbf{b} - \mathbf{a}| \\ &= \frac{1}{2} \cdot (\hat{\mathbf{u}} \cdot (\mathbf{P} - \mathbf{a})) \cdot |\mathbf{b} - \mathbf{a}| \\ &= \frac{1}{2} \cdot \gamma \left(\hat{\mathbf{u}} \cdot (\mathbf{c} - \mathbf{a})\right) \cdot |\mathbf{b} - \mathbf{a}| \\ &= \gamma \frac{1}{2} \cdot h \cdot |\mathbf{b} - \mathbf{a}| = \gamma A_{\text{total}} \end{aligned}$$



Since,

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

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

$$\hat{\mathbf{u}} \cdot (\mathbf{P} - \mathbf{a}) = \gamma(\hat{\mathbf{u}} \cdot (\mathbf{c} - \mathbf{a}))$$

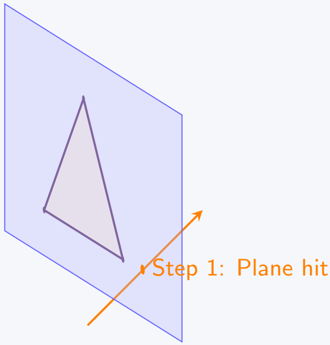
Since $\hat{\mathbf{u}}$ is perpendicular to $\mathbf{b} - \mathbf{a}$.

Method 1: Two-Step Ray-Triangle Intersection

Algorithm Steps

Step 1: Ray-Plane Intersection

$$t = -\frac{D + \mathbf{n} \cdot \mathbf{R_o}}{\mathbf{n} \cdot \mathbf{R_d}}$$



Method 1: Two-Step Ray-Triangle Intersection

Algorithm Steps

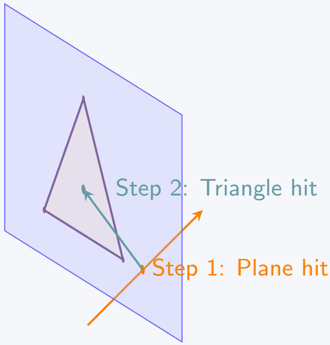
Step 1: Ray-Plane Intersection

$$t = -\frac{D + \mathbf{n} \cdot \mathbf{R_o}}{\mathbf{n} \cdot \mathbf{R_d}}$$

Step 2: Inside/Outside Test

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

Solve for β, γ and check bounds.



Method 2: Direct Barycentric Intersection

Direct Approach

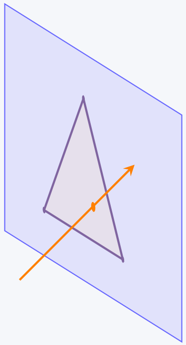
Set ray equation equal to barycentric form:

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

Rearrange to linear system:

$$\begin{bmatrix} -\mathbf{R_d} & (\mathbf{b} - \mathbf{a}) & (\mathbf{c} - \mathbf{a}) \end{bmatrix} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \mathbf{R_o} - \mathbf{a}$$

Solve using Cramer's rule or LU decomposition.



Cramer's Rule Solution

Matrix Form

$$\underbrace{\begin{bmatrix} -R_{dx} & b_x - a_x & c_x - a_x \\ -R_{dy} & b_y - a_y & c_y - a_y \\ -R_{dz} & b_z - a_z & c_z - a_z \end{bmatrix}}_{A} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} R_{ox} - a_x \\ R_{oy} - a_y \\ R_{oz} - a_z \end{bmatrix}$$

Cramer's Rule Solution

Matrix Form

$$\underbrace{\begin{bmatrix} -R_{dx} & b_x - a_x & c_x - a_x \\ -R_{dy} & b_y - a_y & c_y - a_y \\ -R_{dz} & b_z - a_z & c_z - a_z \end{bmatrix}}_{A} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} R_{ox} - a_x \\ R_{oy} - a_y \\ R_{oz} - a_z \end{bmatrix}$$

Cramer's Rule

$$t = \frac{1}{|A|} \begin{vmatrix} R_{ox} - a_x & b_x - a_x & c_x - a_x \\ R_{oy} - a_y & b_y - a_y & c_y - a_y \\ R_{oz} - a_z & b_z - a_z & c_z - a_z \end{vmatrix}$$

$$\beta = \frac{1}{|A|} \begin{vmatrix} -R_{dx} & R_{ox} - a_x & c_x - a_x \\ -R_{dy} & R_{oy} - a_y & c_y - a_y \\ -R_{dz} & R_{oz} - a_z & c_z - a_z \end{vmatrix}$$

$$\gamma = \frac{1}{|A|} \begin{vmatrix} -R_{dx} & b_x - a_x & R_{ox} - a_x \\ -R_{dy} & b_y - a_y & R_{oy} - a_y \\ -R_{dz} & b_z - a_z & R_{oz} - a_z \end{vmatrix}$$

Cramer's Rule Solution

Matrix Form

$$\underbrace{\begin{bmatrix} -R_{dx} & b_x - a_x & c_x - a_x \\ -R_{dy} & b_y - a_y & c_y - a_y \\ -R_{dz} & b_z - a_z & c_z - a_z \end{bmatrix}}_{A} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} R_{ox} - a_x \\ R_{oy} - a_y \\ R_{oz} - a_z \end{bmatrix}$$

Cramer's Rule

$$t = \frac{1}{|A|} \begin{vmatrix} R_{ox} - a_x & b_x - a_x & c_x - a_x \\ R_{oy} - a_y & b_y - a_y & c_y - a_y \\ R_{oz} - a_z & b_z - a_z & c_z - a_z \end{vmatrix}$$

$$\beta = \frac{1}{|A|} \begin{vmatrix} -R_{dx} & R_{ox} - a_x & c_x - a_x \\ -R_{dy} & R_{oy} - a_y & c_y - a_y \\ -R_{dz} & R_{oz} - a_z & c_z - a_z \end{vmatrix}$$

$$\gamma = \frac{1}{|A|} \begin{vmatrix} -R_{dx} & b_x - a_x & R_{ox} - a_x \\ -R_{dy} & b_y - a_y & R_{oy} - a_y \\ -R_{dz} & b_z - a_z & R_{oz} - a_z \end{vmatrix}$$

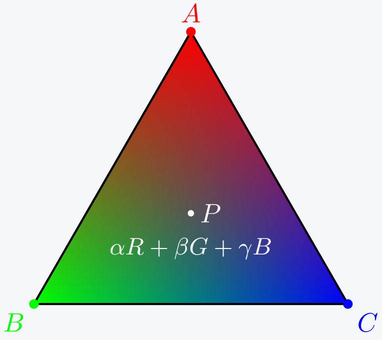
Checks

- $t_{\text{min}} < t < t_{\text{current}}$ (valid intersection)
- $\beta, \gamma \geq 0$ and $\beta + \gamma \leq 1$ (inside triangle)

Bonus of Using Barycentric Coordinates

Advantages

- Efficient to compute
- Get Barycentric coordinates for free
- Enables interpolation of vertex attributes
  Used in —
  - Textures
  - Normals
  - Colors



Ray-Sphere Intersection Overview

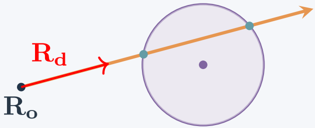
Two Main Approaches

Method 1: Algebra

1. Setup quadratic equation
2. Solve for t

Method 2: Geometry

1. Use geomety to find intersection step by step
2. Reject early if hit is not possible



Sphere Representation

Implicit Sphere Equation

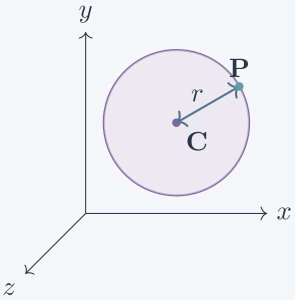
Sphere centered at origin:

$$\mathbf{P} \cdot \mathbf{P} - r^2 = 0$$
$$x^2 + y^2 + z^2 - r^2 = 0$$

General sphere at center $\mathbf{C}$:

$$(\mathbf{P} - \mathbf{C}) \cdot (\mathbf{P} - \mathbf{C}) - r^2 = 0$$

Note: Translation to origin simplifies calculation!

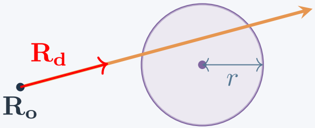


Ray-Sphere Intersection: Algebraic Method

Algebraic Solution

Step 1: Substitute ray equation $\mathbf{P}(t) = \mathbf{R_o} + t\mathbf{R_d}$ into sphere

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

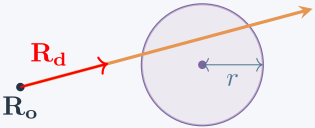


Ray-Sphere Intersection: Algebraic Method

Algebraic Solution

Step 2: Expand and rearrange

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



Ray-Sphere Intersection: Algebraic Method

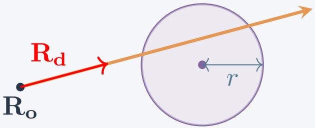
Algebraic Solution

Step 3: Quadratic formula
$(ax^2 + bx + c = 0)$

$$a = \mathbf{R_d} \cdot \mathbf{R_d} = 1 \text{ (normalized)}$$

$$b = 2\mathbf{R_d} \cdot \mathbf{R_o}$$

$$c = \mathbf{R_o} \cdot \mathbf{R_o} - r^2$$



Ray-Sphere Intersection: Algebraic Method

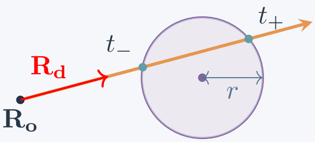
Algebraic Solution

Step 3: Quadratic formula ($ax^2 + bx + c = 0$)

$$a = \mathbf{R_d} \cdot \mathbf{R_d} = 1 \text{ (normalized)}$$

$$b = 2\mathbf{R_d} \cdot \mathbf{R_o}$$

$$c = \mathbf{R_o} \cdot \mathbf{R_o} - r^2$$



Discriminant Analysis

$$\Delta = b^2 - 4ac = (2\mathbf{R_d} \cdot \mathbf{R_o})^2 - 4(\mathbf{R_o} \cdot \mathbf{R_o} - r^2)$$

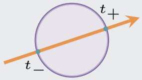
$$t_\pm = \frac{-b \pm \sqrt{\Delta}}{2a} = -\mathbf{R_d} \cdot \mathbf{R_o} \pm \frac{\sqrt{\Delta}}{2}$$

Algebraic Method: Three Cases

The discriminant Δ determines the number of intersections:

$\Delta > 0$

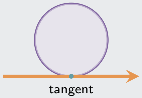
2 intersections



Choose closest positive (usually t_-)

$\Delta = 0$

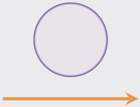
1 intersection



Ray tangent to sphere

$\Delta < 0$

No intersection



Ray doesn't hit sphere

Additional Check

Remember to check t_{min} to find closest valid intersection.

Ray-Sphere Intersection: Geometric Method

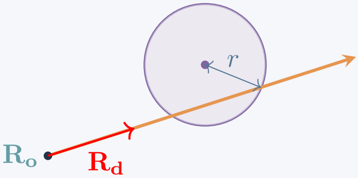
Geometric Approach

Step 1: Check ray origin (eye) position

$$\text{Inside: } \mathbf{R_o} \cdot \mathbf{R_o} < r^2$$

$$\text{Outside: } \mathbf{R_o} \cdot \mathbf{R_o} > r^2$$

$$\text{On surface: } \mathbf{R_o} \cdot \mathbf{R_o} = r^2$$

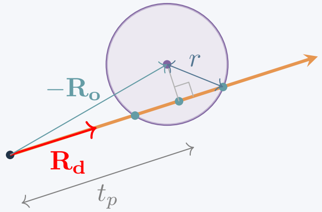


Ray-Sphere Intersection: Geometric Method

Geometric Approach

Step 2: Find parameter t_p for the point on the ray closest to the sphere center

$$t_P = -\mathbf{R_o} \cdot \mathbf{R_d}$$

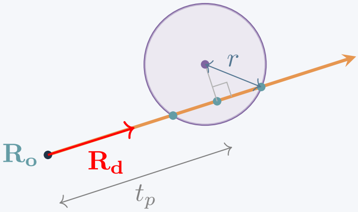


Ray-Sphere Intersection: Geometric Method

Geometric Approach

Step 3: Early rejection test

If ray origin outside & $t_P < 0 \Rightarrow$ no hit

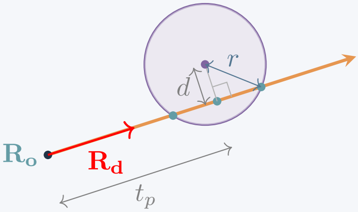


Ray-Sphere Intersection: Geometric Method

Geometric Approach

Step 4: Find squared distance to sphere center

$$d^2 = \mathbf{R_o} \cdot \mathbf{R_o} - t_P^2$$

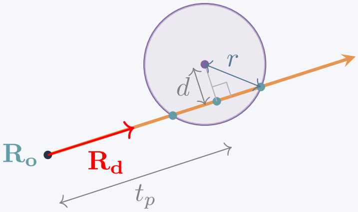


Ray-Sphere Intersection: Geometric Method

Geometric Approach

Step 5: Second rejection test

If $d^2 > r^2 \Rightarrow$ no hit



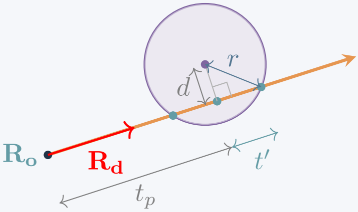
Ray-Sphere Intersection: Geometric Method

Geometric Approach

Step 6: Find intersection distance

$$t'^2 = r^2 - d^2$$

$$t' = \sqrt{r^2 - d^2}$$



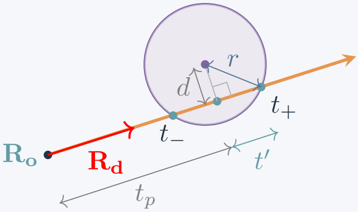
Ray-Sphere Intersection: Geometric Method

Geometric Approach

Step 7: Choose correct intersection parameter

$$\textbf{Outside: } t_- = t_P - t'$$
$$\textbf{Inside: } t_+ = t_P + t'$$

$t_{\text{min}} < t_\pm < t_{\text{current}} \Rightarrow$ hit



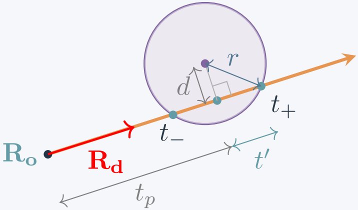
Ray-Sphere Intersection: Geometric Method

Geometric Approach

Step 7: Choose correct intersection parameter

$$\textbf{Outside: } t_- = t_P - t'$$
$$\textbf{Inside: } t_+ = t_P + t'$$

$t_{\text{min}} < t_\pm < t_{\text{current}} \Rightarrow$ hit



Benefits of Method

- **Early rejection:** Avoid extra work for rays missing sphere
- **Optimized:** Efficient for rays outside pointing away

General Quadric Surfaces

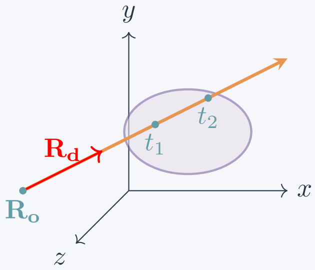
Quadric Surface Definition

General equation:

$$Ax^2 + By^2 + Cz^2 + Dxy + Eyz$$
$$+ Fxz + Gx + Hy + Iz + J = 0$$

Common Quadric Surfaces:

- **Ellipsoid:** $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$
- **Cone:** $\frac{x^2}{a^2} - \frac{y^2}{b^2} + \frac{z^2}{c^2} = 0$
- **Cylinder:** $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$
- **Hyperboloid & Paraboloid**



Reference: Quadric Surfaces in Paul's Online Notes

Ray-Quadric Surface Intersection

Intersection Method

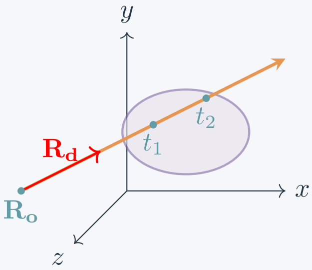
Step 1: Substitute ray equation into quadric

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

$$P_x = R_{0x} + t \cdot R_{dx}$$

$$P_y = R_{0y} + t \cdot R_{dy}$$

$$P_z = R_{0z} + t \cdot R_{dz}$$

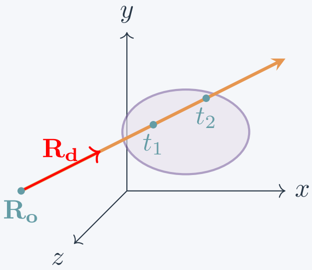


Ray-Quadric Surface Intersection

Intersection Method

Step 2: Results in quadratic equation

$$ax^2 + bx + c = 0$$

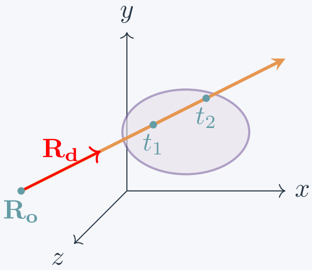


Ray-Quadric Surface Intersection

Intersection Method

Step 3: Solve using quadratic formula

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

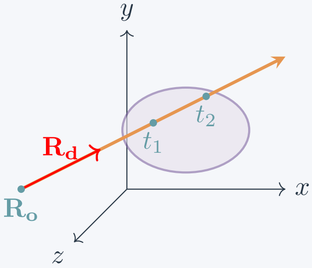


Ray-Quadric Surface Intersection

Intersection Method

Step 3: Solve using quadratic formula

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



Solution Cases

Check the discriminant $\Delta = b^2 - 4ac$:

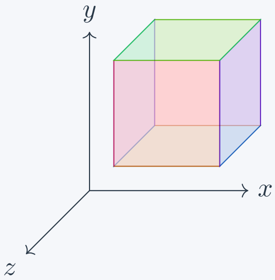
- $\Delta > 0$**:** Two real solutions (ray intersects surface twice)
- $\Delta = 0$**:** One solution (ray tangent to surface)
- $\Delta < 0$**:** No real solutions (ray misses surface)
- **Accept:** Accept smaller t such that $t_{\text{min}} < t < t_{\text{current}}$

Ray-AABB Intersection: Overview

Axis-Aligned Bounding Box

A simple 3D box or rectangle aligned with the coordinate axes.

- The sides are parallel to the axes, that's why it's called **axis-aligned**.
- Usually used to enclose complex objects, which is why it's called a **bounding box**.
- Very efficient for intersection tests. (Just test 6 planes)



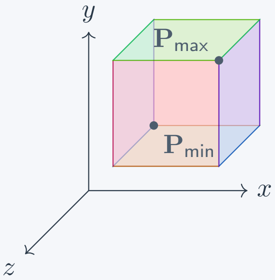
AABB Mathematical Representation

AABB Representation

$$\text{AABB} = \left\{ (x, y, z) \;\middle|\; \begin{array}{c} x_{\min} \leq x \leq x_{\max} \\ y_{\min} \leq y \leq y_{\max} \\ z_{\min} \leq z \leq z_{\max} \end{array} \right\}$$

We can store,

$$\mathbf{P}_{\text{min}} = (x_{\text{min}}, y_{\text{min}}, z_{\text{min}})$$
$$\mathbf{P}_{\text{max}} = (x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$$



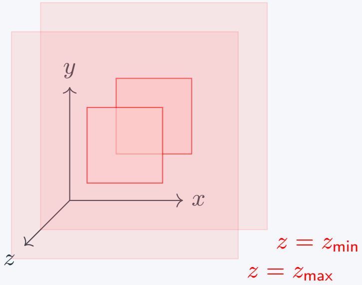
Ray-AABB Intersection: Slab Method

Approach

Consider z axis first. There are two planes:

$$z = z_{\text{min}}$$

$$z = z_{\text{max}}$$



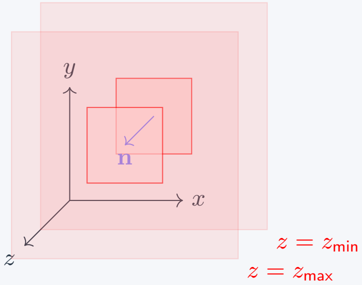
Ray-AABB Intersection: Slab Method

Approach

Consider z axis first. There are two planes:

$$\underbrace{z}_{\mathbf{n}=(0,0,1)} \quad \underbrace{-z_{\text{min}}}_{D=-z_{\text{min}}} = 0$$

$$z - z_{\text{max}} = 0$$



Ray-AABB Intersection: Slab Method

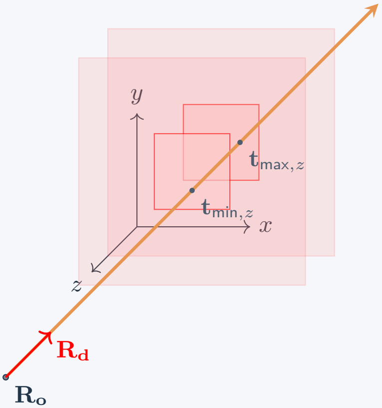
Approach

Compute intersection with ray:

$$t_{\text{min},z} = -\frac{D + \mathbf{n} \cdot \mathbf{R_o}}{\mathbf{n} \cdot \mathbf{R_d}}$$
$$= -\frac{-z_{\text{min}} + R_{oz}}{R_{dz}}$$
$$t_{\text{min},z} = \frac{z_{\text{min}} - R_{oz}}{R_{dz}}$$

Similarly,

$$t_{\text{max},z} = \frac{z_{\text{max}} - R_{oz}}{R_{dz}}$$



Well, actually $t_{\min,z}$ should be $t_{\max,z}$ and $t_{\max,z}$ should be $t_{\min,z}$ in the diagram. This is why we need the **swap** in step 2 of the algorithm.

Ray-AABB Intersection: Slab Method

Approach

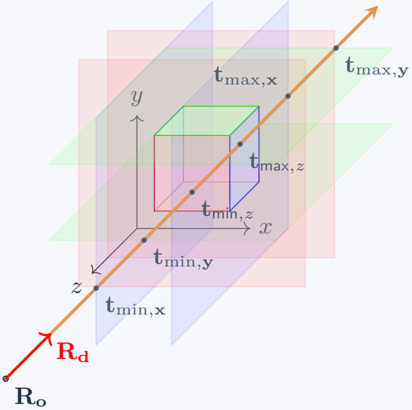
Similarly for y and z axes:

$$t_{\text{min},x} = \frac{x_{\text{min}} - R_{ox}}{R_{dx}}$$

$$t_{\text{max},x} = \frac{x_{\text{max}} - R_{ox}}{R_{dx}}$$

$$t_{\text{min},y} = \frac{y_{\text{min}} - R_{oy}}{R_{dy}}$$

$$t_{\text{max},y} = \frac{y_{\text{max}} - R_{oy}}{R_{dy}}$$



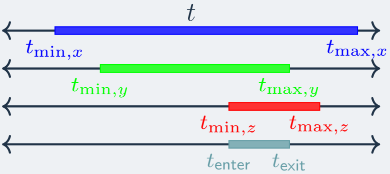
Ray-AABB Intersection: Slab Method

Approach

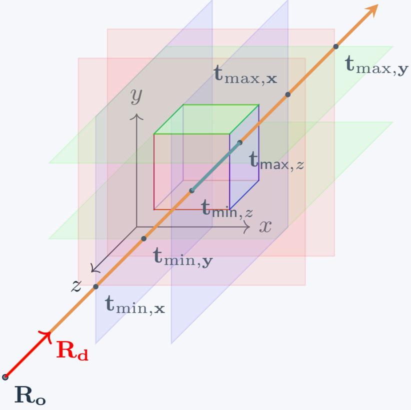
Find the overlap of the intervals:

$$t_{\text{enter}} = \max(t_{\text{min},x}, t_{\text{min},y}, t_{\text{min},z})$$

$$t_{\text{exit}} = \min(t_{\text{max},x}, t_{\text{max},y}, t_{\text{max},z})$$



If there is overlap, i.e. $t_{\text{enter}} \leq t_{\text{exit}}$, then the ray intersects the AABB.



Ray-AABB Intersection: Slab Method

Algorithm

Step 1: Compute t_{min} and t_{max} for each axis

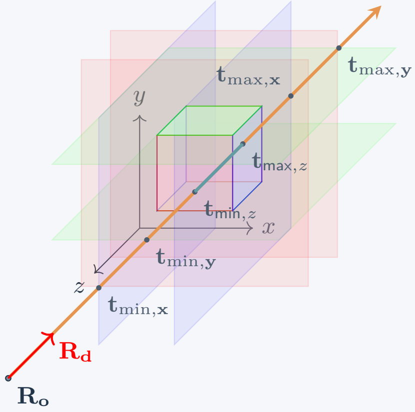
Step 2: If $t_{\text{min}} > t_{\text{max}}$ for any axis, swap $t_{\min}$ and $t_{\max}$

Step 3: Find

$$t_{\text{enter}} = \max_{i \in x,y,z} t_{\min,i}$$

$$t_{\text{exit}} = \min_{i \in x,y,z} t_{\max,i}$$

Step 4: There is an intersection if $t_{\text{enter}} \leq t_{\text{exit}}$ and $t_{\text{exit}} \geq 0$



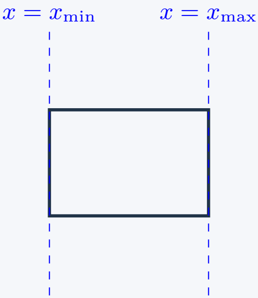
Ray-AABB Intersection: Slab Method

Edge Case

Ray parallel to an axis
$(R_{dx}/R_{dy}/R_{dz} = 0)$

Automatically handled by division by zero.
For example, if $R_{dx} = 0$, then:

$$t_{\min\,/\,\max,x} = \frac{x_{\min\,/\,\max} - R_{ox}}{0} = \pm\infty$$



Ray-AABB Intersection: Slab Method

Edge Case

Ray parallel to an axis
$(R_{dx}/R_{dy}/R_{dz} = 0)$

Automatically handled by division by zero.
For example, if $R_{dx} = 0$, then:

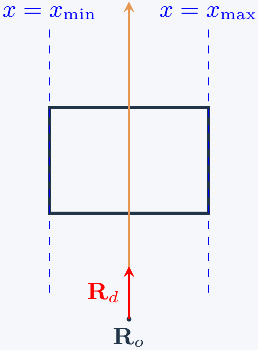
$$t_{\min\,/\,\max,x} = \frac{x_{\min\,/\,\max} - R_{ox}}{0} = \pm\infty$$

- If the ray is within the slab:

  $$x_{\text{min}} \leq R_{ox} \leq x_{\text{max}}$$

  In this case,

  $$t_{\min,x} = -\infty \text{ and } t_{\max,x} = \infty$$



Ray-AABB Intersection: Slab Method

Edge Case

Ray parallel to an axis
$(R_{dx}/R_{dy}/R_{dz} = 0)$

Automatically handled by division by zero.
For example, if $R_{dx} = 0$, then:

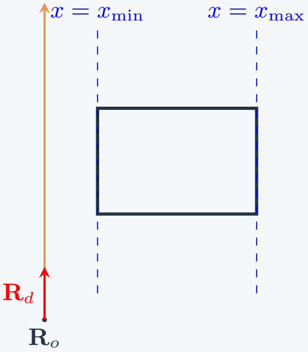
$$t_{\min\,/\,\max,x} = \frac{x_{\min\,/\,\max} - R_{ox}}{0} = \pm\infty$$

- If ray is outside the slab:

$$R_{ox} < x_{\text{min}} \text{ or } x_{\text{max}} < R_{ox}$$

Then:

$$t_{\min,x} = t_{\max,x} = \infty \text{ or}$$
$$t_{\min,x} = t_{\max,x} = -\infty$$



Ray-AABB Intersection: Slab Method

Edge Case

Ray parallel to an axis

$(R_{dx}/R_{dy}/R_{dz} = 0)$

Automatically handled by division by zero. For example, if $R_{dx} = 0$, then:

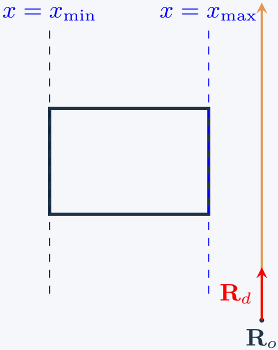
$$t_{\min\,/\,\max,x} = \frac{x_{\min\,/\,\max} - R_{ox}}{0} = \pm\infty$$

- If ray is outside the slab:

  $$R_{ox} < x_{\text{min}} \text{ or } x_{\text{max}} < R_{ox}$$

  Then:

  $$t_{\min,x} = t_{\max,x} = \infty \text{ or}$$
  $$t_{\min,x} = t_{\max,x} = -\infty$$

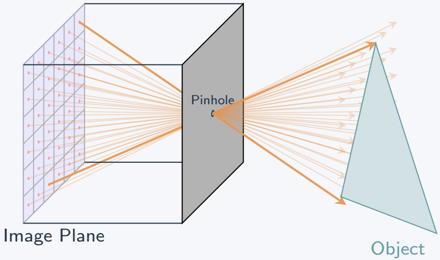


Cameras

The Pinhole Camera Model

Pinhole Camera

The simplest camera. A box with a pinhole. Light entering the pinhole creates an image on the other side of the box.

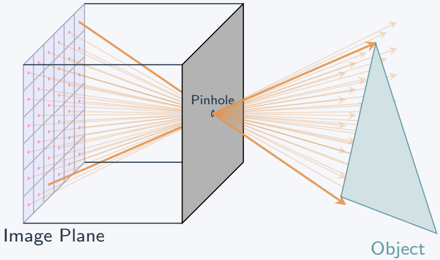


The Pinhole Camera Model

Pinhole Camera

The simplest camera. A box with a pinhole. Light entering the pinhole creates an image on the other side of the box.

- **Point aperture**
  Just a single point for rays to pass through.
- **Perfect focus everywhere**
  All rays go through the pinhole. Therefore, all points in the image plane correspond to exactly one point in the scene.

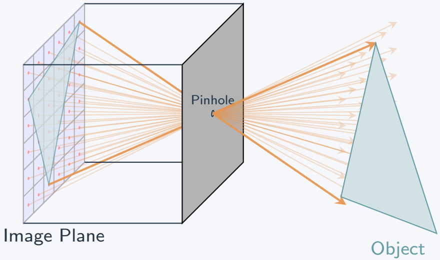


The Pinhole Camera Model

Pinhole Camera

The simplest camera. A box with a pinhole. Light entering the pinhole creates an image on the other side of the box.

- **Point aperture**
- **Perfect focus everywhere**
- **Inverted Image**
  Rays from the top of the scene hit the bottom of the image plane, and vice versa.



The Pinhole Camera Model

Pinhole Camera

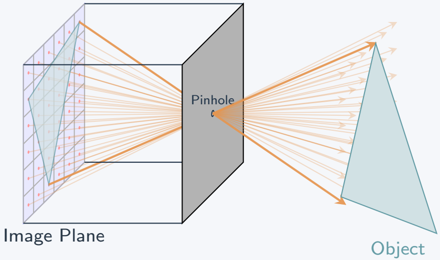
The simplest camera. A box with a pinhole. Light entering the pinhole creates an image on the other side of the box.

- **Point aperture**
- **Perfect focus everywhere**
- **Inverted Image**

Ray Generation:

$$\mathbf{R_o} = \mathbf{hole}$$

$$\mathbf{R_d} = \frac{\mathbf{hole} - \mathbf{pixel}}{|\mathbf{hole} - \mathbf{pixel}|}$$



The Pinhole Camera Model

Pinhole Camera

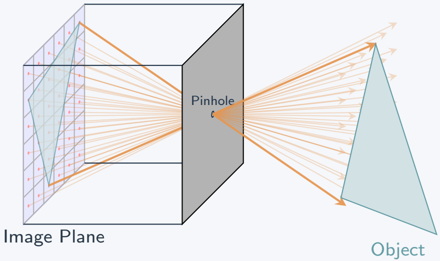
The simplest camera. A box with a pinhole. Light entering the pinhole creates an image on the other side of the box.

- **Point aperture**
- **Perfect focus everywhere**
- **Inverted Image**

Ray Generation:

$$\mathbf{R_o} = \mathbf{hole}$$

$$\mathbf{R_d} = \frac{\mathbf{hole} - \mathbf{pixel}}{|\mathbf{hole} - \mathbf{pixel}|}$$

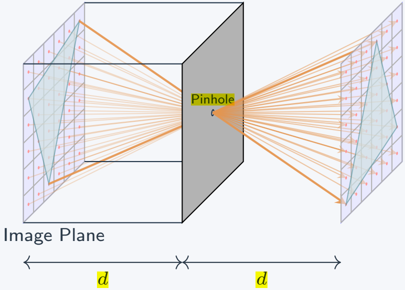


Real pinhole cameras exist! They create sharp images but require very long exposure times due to tiny aperture.

The Pinhole Camera Model

Imaginary Plane

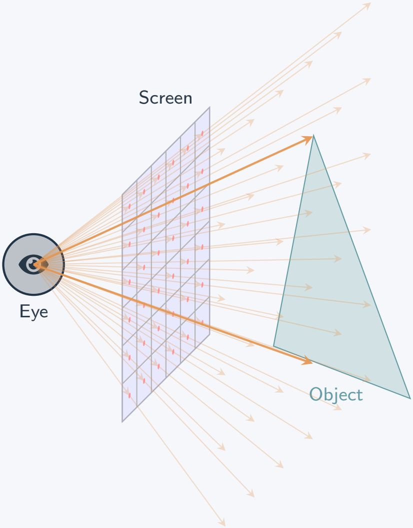
Let's place an imaginary plane in front of the hole.



Simplified Pinhole Camera

Simplification

Place image plane in front!
Equivalent to pinhole camera. The image plane will mirror the image of the pinhole camera if the plane is placed in the same distance in front of the eye.

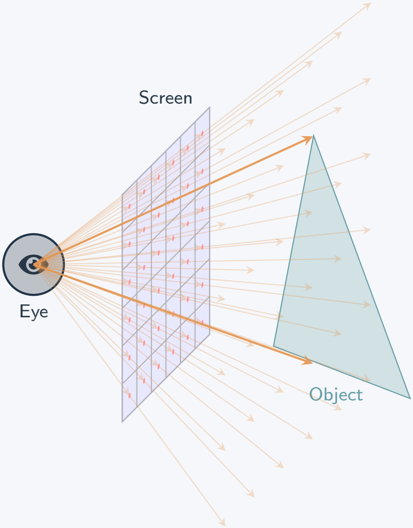


Simplified Pinhole Camera

Simplification

Place image plane in front!
Equivalent to pinhole camera.

- **Physically unrealizable**
  A real cannot have a sensor/film in front of the pinhole.

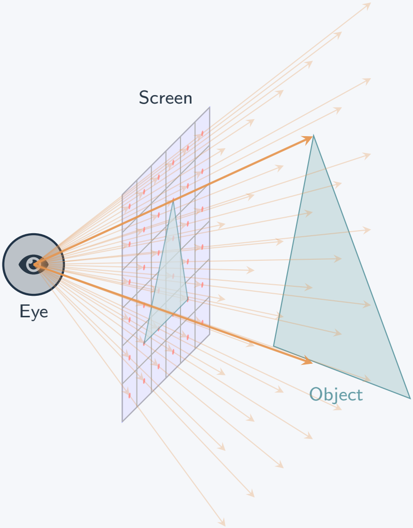


Simplified Pinhole Camera

Simplification

Place image plane in front!
Equivalent to pinhole camera.

- **Physically unrealizable**
- **Non-inverted image**
  The image plane is in front of the pinhole, so the rays hit the image plane directly.



Simplified Pinhole Camera

Simplification

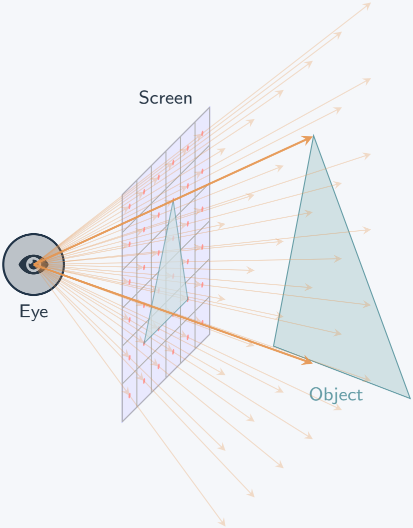
Place image plane in front!
Equivalent to pinhole camera.

- **Physically unrealizable**
- **Non-inverted image**

Ray Generation:

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

$$\mathbf{R_d} = \frac{\mathbf{pixel} - \mathbf{eye}}{|\mathbf{pixel} - \mathbf{eye}|}$$



Simplified Pinhole Camera

Simplification

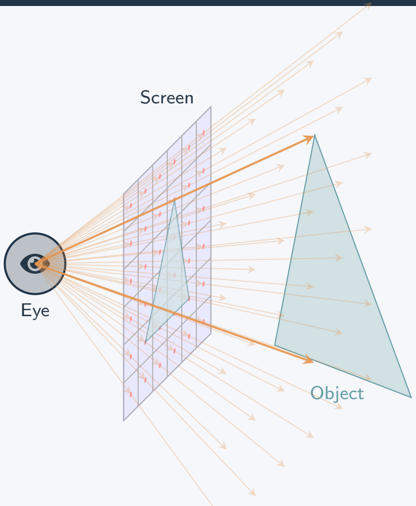
Place image plane in front!
Equivalent to pinhole camera.

- **Physically unrealizable**
- **Non-inverted image**

Ray Generation:

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

$$\mathbf{R_d} = \frac{\mathbf{pixel} - \mathbf{eye}}{|\mathbf{pixel} - \mathbf{eye}|}$$



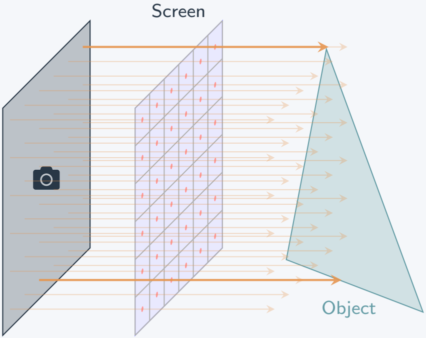
Advantage

Upright image, simpler ray generation, equivalent to real pinhole!

Orthographic Camera

Orthographic Projection

Rays are all parallel to a specific direction, the camera's view direction $\mathbf{w}$. It is called orthographic projection because rays are orthogonal to the image plane.

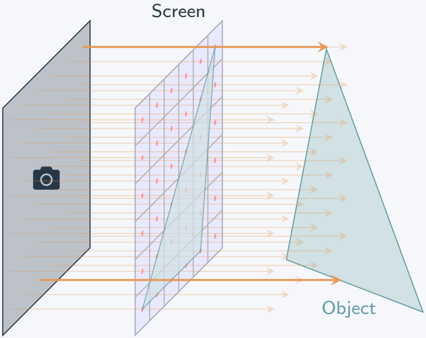


Orthographic Camera

Orthographic Projection

Rays are all parallel to a specific direction, the camera's view direction $\mathbf{w}$. It is called orthographic projection because rays are orthogonal to the image plane.

- **No perspective distortion**
  Object appear the same size regardless of distance.



Orthographic Camera

Orthographic Projection

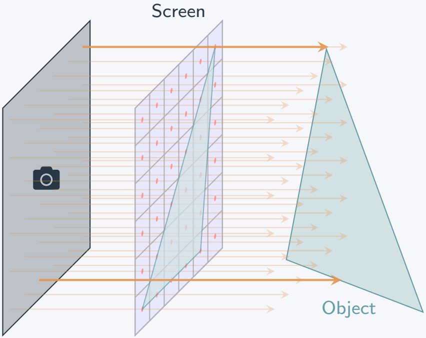
Rays are all parallel to a specific direction, the camera's view direction $\mathbf{w}$. It is called orthographic projection because rays are orthogonal to the image plane.

- **No perspective distortion**
  Object appear the same size regardless of distance.

Ray Generation:

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

$$\mathbf{R_d} = \mathbf{w}$$



Orthographic Camera

Orthographic Projection

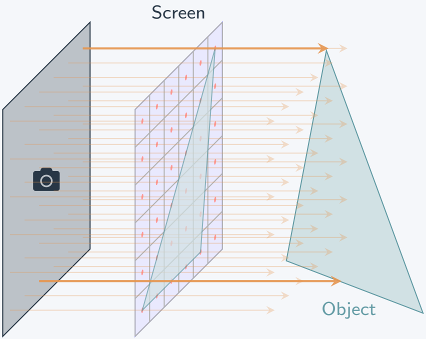
Rays are all parallel to a specific direction, the camera's view direction $\mathbf{w}$. It is called orthographic projection because rays are orthogonal to the image plane.

- **No perspective distortion**
  Object appear the same size regardless of distance.

Ray Generation:

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

$$\mathbf{R_d} = \mathbf{w}$$



Applications

Technical drawings, **CAD software**, **2D games**, architectural visualization

Perspective vs Orthographic

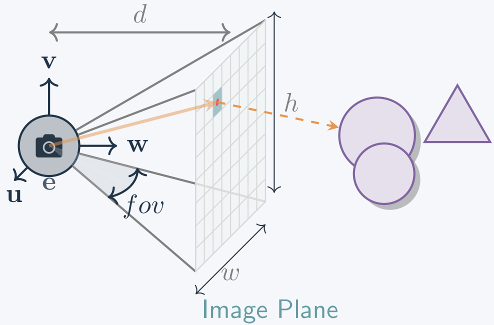
Perspective

- Natural/realistic scenes
- Depth perception
- Size $\propto \frac{1}{\text{distance}}$

Orthographic

- CAD/Engineering
- Precise measurements
- Size $=$ constant

Camera Representation

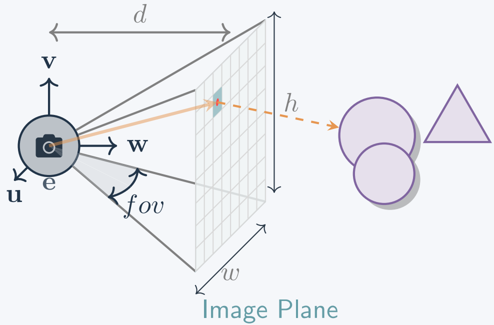


Camera Description

Camera position $\mathbf{e}$
Orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$

Field of View fov
Distance to image plane d
Image dimension $(w \times h)$

Camera Representation

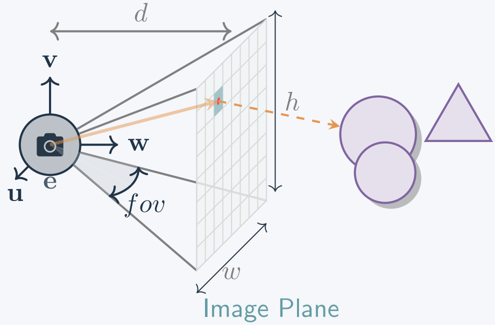


Camera Description

Camera position $\mathbf{e}$
Position of the camera in 3D space.
Orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$

Field of View fov
Distance to image plane d
Image dimension $(w \times h)$

Camera Representation



Camera Description

Camera position $\mathbf{e}$

Orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$

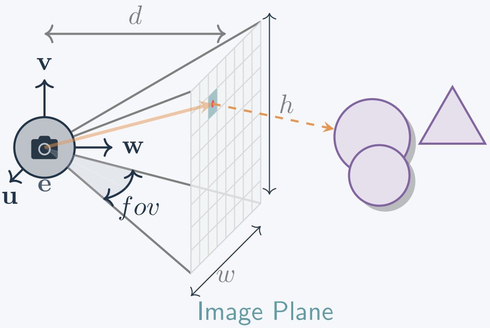
Basis vectors defining the camera's orientation. Up direction $\mathbf{v}$, right direction $\mathbf{u}$, and forward direction $\mathbf{w}$. The plane is perpendicular to $\mathbf{w}$.

Field of View fov

Distance to image plane d

Image dimension $(w \times h)$

Camera Representation

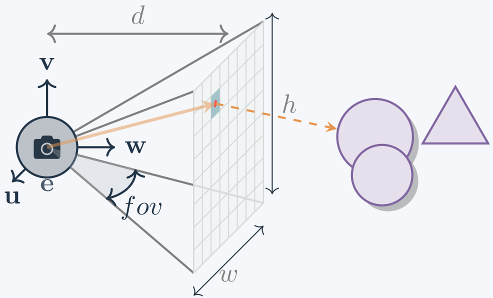


Camera Description

Camera position $\mathbf{e}$
Orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$

Field of View fov
fov is the angle created by the screen at the camera's position. It determines how big the image plane is.
Distance to image plane d
Image dimension $(w \times h)$

Camera Representation

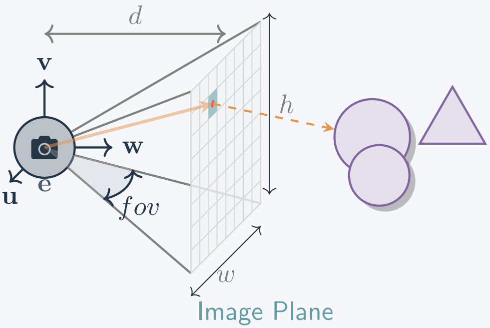


Camera Description

Camera position $\mathbf{e}$
Orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$

Field of View fov
Distance to image plane d
Distance to the plane where the image is formed, along the $\mathbf{w}$ direction.
Image dimension $(w \times h)$

Camera Representation

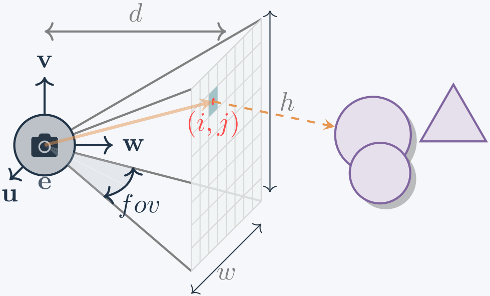


Camera Description

Camera position $\mathbf{e}$
Orthobasis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$

Field of View fov
Distance to image plane d
Image dimension $(w \times h)$
The resolution of the image.
Determines the aspect ratio.

Camera Representation

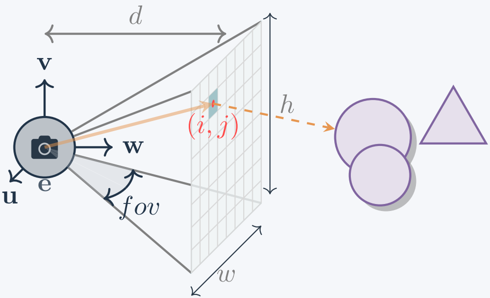


Mathematical Representation

Position of the camera is directly given by $\mathbf{e}$.

$$\mathbf{eye} = \mathbf{e}$$

Camera Representation



Mathematical Representation

Calculating the position of the pixel is a bit more involved. For pixel (i, j):

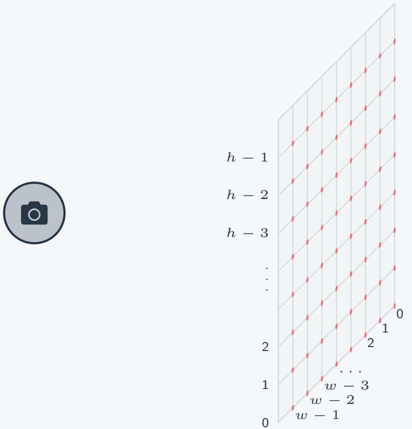
$$\mathbf{pixel} = \mathbf{e} + d\mathbf{w} + s\mathbf{u} + t\mathbf{v}$$

Where s and t are:

$$s = \left(2 \cdot \frac{i + 0.5}{w} - 1\right) \cdot d \tan\left(\frac{fov}{2}\right) \text{ and } t = \left(2 \cdot \frac{j + 0.5}{h} - 1\right) \cdot d \tan\left(\frac{fov}{2}\right) \cdot \frac{h}{w}$$

Assuming $i \in [0, w - 1]$ and $j \in [0, h - 1]$.

Pixel Coordinate Formula

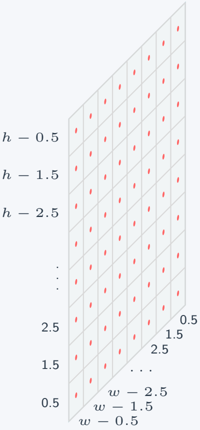


(i, j)

Intuition

Initially, we have a grid of pixels on the image plane. The pixels (i, j) are in the range $(0, 0)$ to $(w-1, h-1)$.

Pixel Coordinate Formula

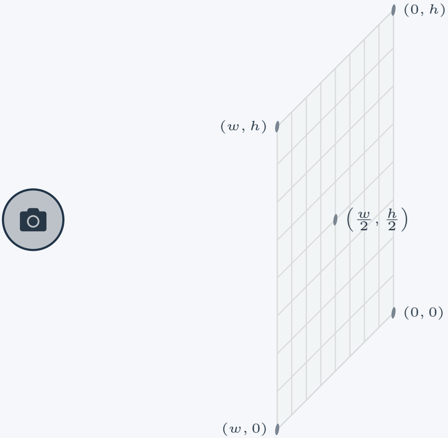


$$(i+0.5, j+0.5)$$

Intuition

The pixel center is at the midpoints. Shift by 0.5 to center the pixels.

Pixel Coordinate Formula

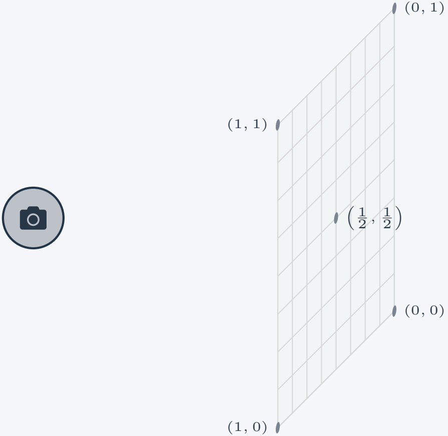


$$(i + 0.5, j + 0.5)$$

Intuition

We are still in the pixel space. We need to find actual distances.

Pixel Coordinate Formula

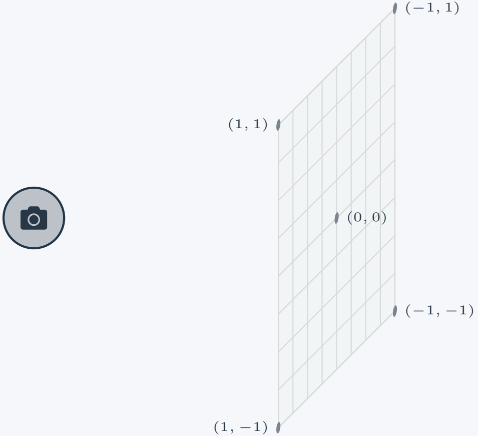


$$\left(\frac{i+0.5}{w}, \frac{j+0.5}{h}\right)$$

Intuition

We first normalize the pixel coordinates to the range $(-1, 1)$.

Pixel Coordinate Formula

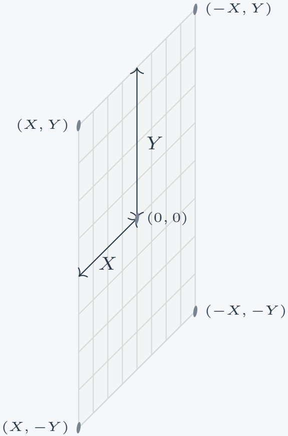


$$\left(2\cdot\frac{i+0.5}{w}-1, 2\cdot\frac{j+0.5}{h}-1\right)$$

Intuition

The pixel coordinates are now in the range $(-1,-1)$ to $(1,1)$. This makes it easier to calculate the pixel position.

Pixel Coordinate Formula

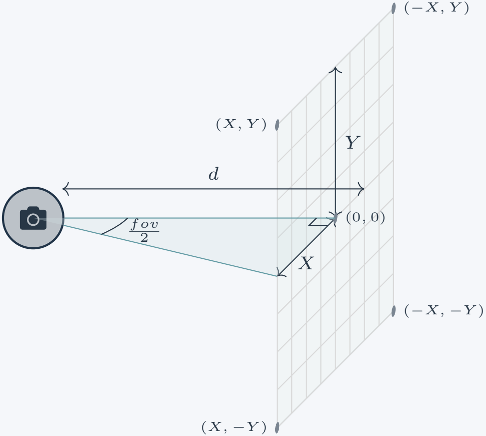


$$\left(\left(2 \cdot \frac{i+0.5}{w} - 1\right) \cdot X, \left(2 \cdot \frac{j+0.5}{h} - 1\right) \cdot Y\right)$$

Intuition

Now if we multiply the pixel coordinates with the half-length and half-width of the image plane, we get the pixel position in the image plane in actual distances.

Pixel Coordinate Formula



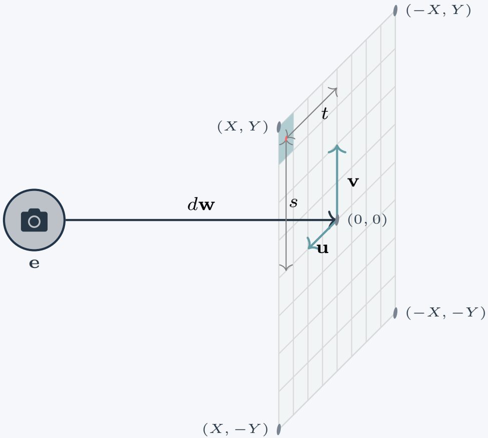
Intuition

X can be found by noticing that in the right angle triangle shown, $\tan\left(\frac{fov}{2}\right) = \frac{X}{d}$.
Y can calculated from X because $\frac{h}{w} = \frac{Y}{X}$, as the aspect ratio is preserved.

$$\left(\left(2 \cdot \frac{i+0.5}{w} - 1\right) \cdot X, \left(2 \cdot \frac{j+0.5}{h} - 1\right) \cdot Y\right)$$

$X = d\tan\left(\frac{fov}{2}\right)$ and $Y = d\tan\left(\frac{fov}{2}\right) \cdot \frac{h}{w}$

Pixel Coordinate Formula

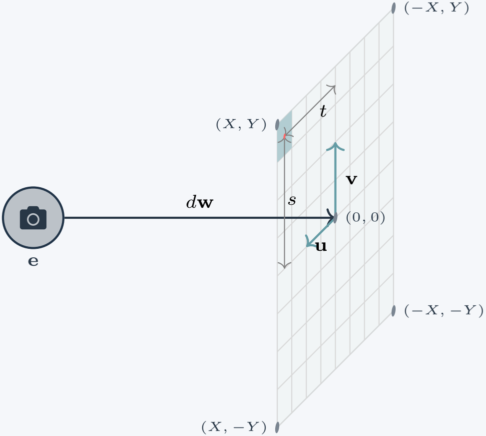


$$\left(\underbrace{\left(2 \cdot \frac{i + 0.5}{w} - 1\right) \cdot X}_{s}, \underbrace{\left(2 \cdot \frac{j + 0.5}{h} - 1\right) \cdot Y}_{t}\right)$$

Intuition

Now that we have the pixel position in the image plane (s, t), we can calculate the pixel position in 3D space.

Pixel Coordinate Formula



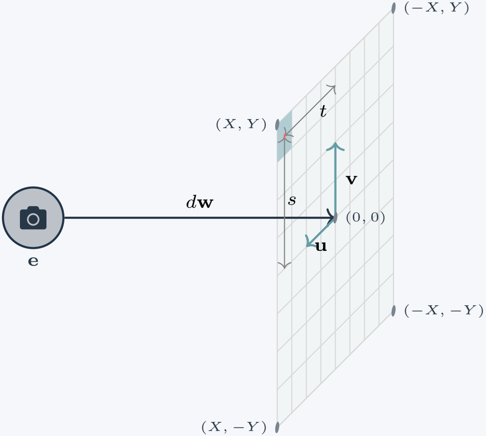
$$\left(\underbrace{\left(2 \cdot \frac{i + 0.5}{w} - 1\right) \cdot X}_{s}, \underbrace{\left(2 \cdot \frac{j + 0.5}{h} - 1\right) \cdot Y}_{t}\right)$$

Intuition

Now that we have the pixel position in the image plane (s, t), we can calculate the pixel position in 3D space.

Step 1: Go to the camera position $\mathbf{e}$.

Pixel Coordinate Formula



$$\left(\underbrace{\left(2 \cdot \frac{i + 0.5}{w} - 1\right) \cdot X}_{s}, \underbrace{\left(2 \cdot \frac{j + 0.5}{h} - 1\right) \cdot Y}_{t} \right)$$

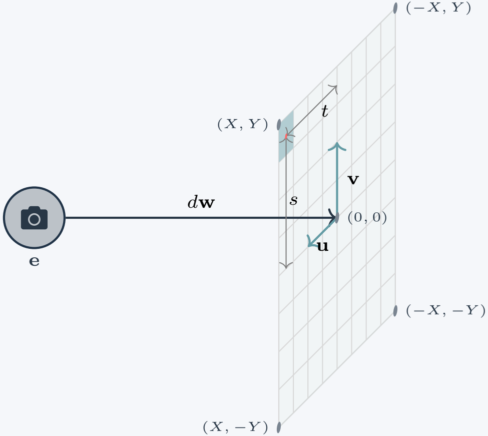
Intuition

Now that we have the pixel position in the image plane (s, t), we can calculate the pixel position in 3D space.

Step 1: Go to the camera position $\mathbf{e}$.

Step 2: Move along the forward direction $\mathbf{w}$ by d.

Pixel Coordinate Formula



$$\left(\underbrace{\left(2 \cdot \frac{i + 0.5}{w} - 1\right) \cdot X}_{s}, \underbrace{\left(2 \cdot \frac{j + 0.5}{h} - 1\right) \cdot Y}_{t}\right)$$

Intuition

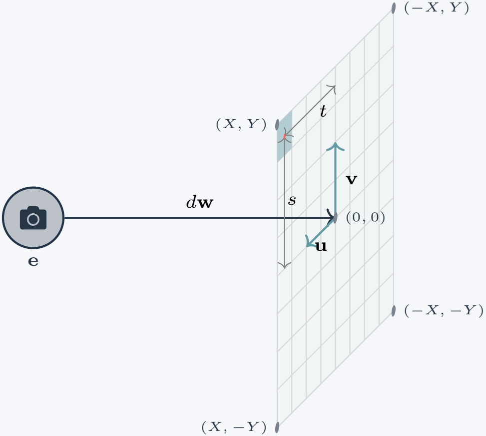
Now that we have the pixel position in the image plane (s, t), we can calculate the pixel position in 3D space.

Step 1: Go to the camera position $\mathbf{e}$.

Step 2: Move along the forward direction $\mathbf{w}$ by d.

Step 3: Move along the right direction $\mathbf{u}$ by s and along the up direction $\mathbf{v}$ by t.

Pixel Coordinate Formula



$$\left(\underbrace{\left(2 \cdot \frac{i + 0.5}{w} - 1\right) \cdot X}_{s}, \underbrace{\left(2 \cdot \frac{j + 0.5}{h} - 1\right) \cdot Y}_{t}\right)$$

Intuition

Now that we have the pixel position in the image plane (s, t), we can calculate the pixel position in 3D space.

Step 1: Go to the camera position $\mathbf{e}$.

Step 2: Move along the forward direction $\mathbf{w}$ by d.

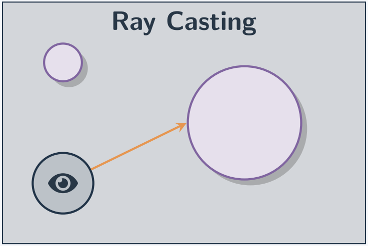
Step 3: Move along the right direction $\mathbf{u}$ by s and along the up direction $\mathbf{v}$ by t.

Putting it all together, we have:

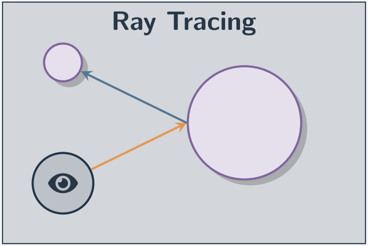
$$\mathbf{pixel} = \mathbf{e} + d\mathbf{w} + s\mathbf{u} + t\mathbf{v}$$

From Ray Casting to Ray Tracing

Ray Casting vs Ray Tracing

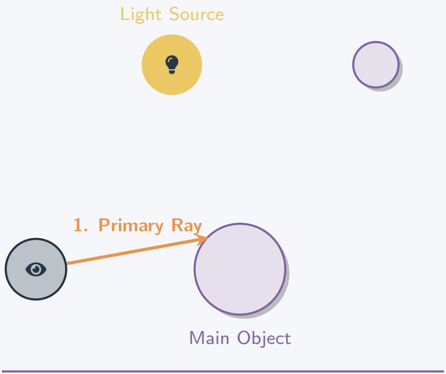


Primary rays only



Primary rays + Secondary Rays

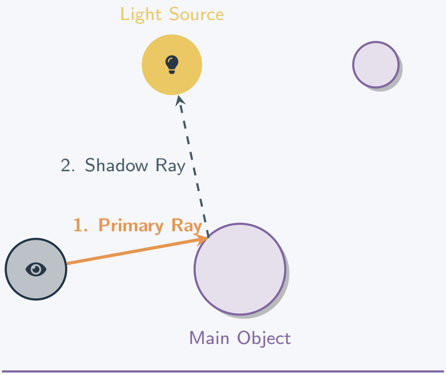
Secondary Rays



Types of Rays

1. **Primary Ray:** From camera to object. Determines closest intersection.

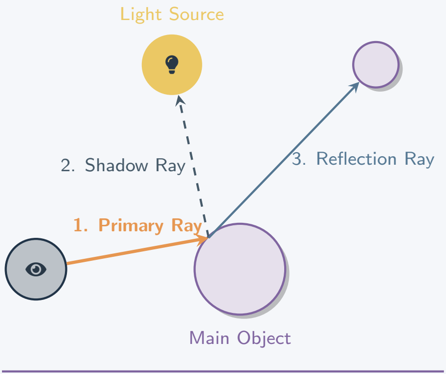
Secondary Rays



Types of Rays

1. **Primary Ray:** From camera to object. Determines closest intersection.
2. **Shadow Ray:** From intersection point to light source. Checks if point is illuminated by light.

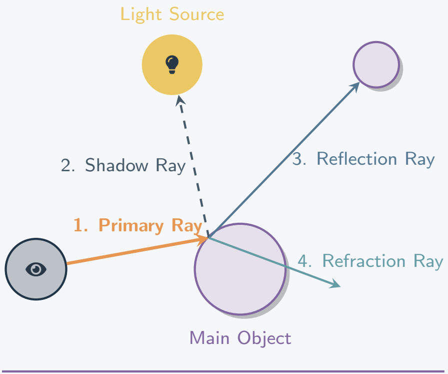
Secondary Rays



Types of Rays

1. **Primary Ray:** From camera to object. Determines closest intersection.
2. **Shadow Ray:** From intersection point to light source. Checks if point is illuminated by light.
3. **Reflection Ray:** From intersection point reflecting off surfaces. For reflections.

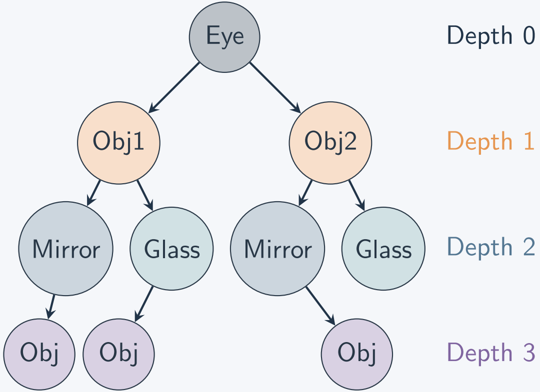
Secondary Rays



Types of Rays

1. **Primary Ray:** From camera to object. Determines closest intersection.
2. **Shadow Ray:** From intersection point to light source. Checks if point is illuminated by light.
3. **Reflection Ray:** From intersection point reflecting off surfaces. For reflections.
4. **Refraction Ray:** From intersection point through transparent surfaces. For refractions.

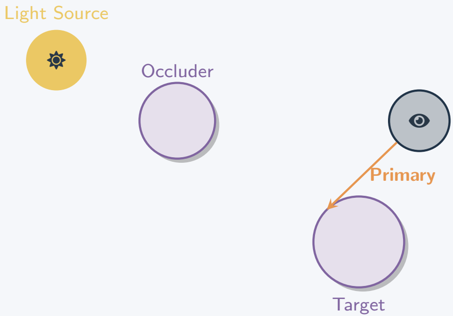
Recursive Ray Tracing



Recursion Control

Termination: Maximum depth reached OR ray contribution becomes negligible

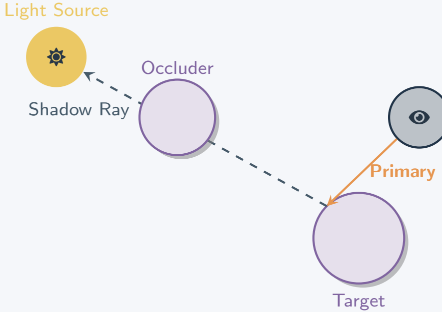
Shadows: The Absence of Light



Shadow Ray Algorithm

For each intersection point:

Shadows: The Absence of Light

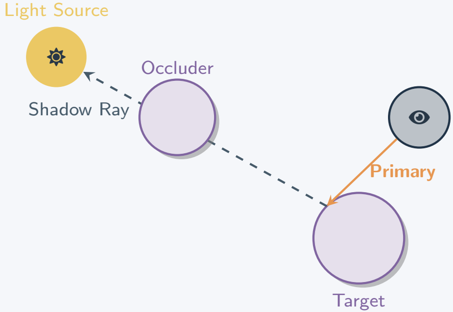


Shadow Ray Algorithm

For each intersection point:

1. Cast ray toward each light source

Shadows: The Absence of Light

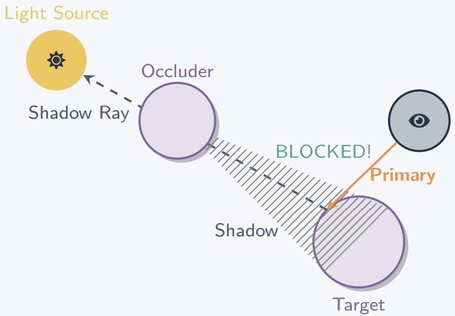


Shadow Ray Algorithm

For each intersection point:

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

Shadows: The Absence of Light

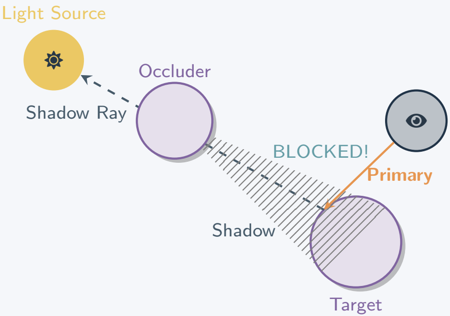


Shadow Ray Algorithm

For each intersection point:

1. Cast ray toward each light source
2. Check if ray intersects any object before reaching light
3. If **blocked** $\rightarrow$ point is in shadow

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
3. If **blocked** $\rightarrow$ point is in shadow
4. If clear $\rightarrow$ point is illuminated

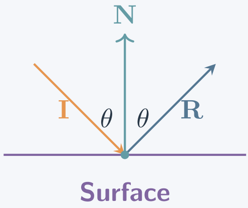
Reflections

Reflection Law

Given incident ray $\mathbf{I}$ and surface normal $\mathbf{N}$:

$$\mathbf{R} = \mathbf{I} - 2(\mathbf{I} \cdot \mathbf{N})\mathbf{N}$$

Physical principle:
Angle of incidence = Angle of reflection

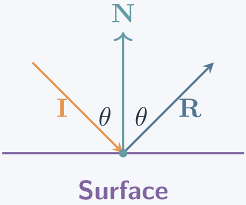


Reflections

Reflection Law

Given incident ray $\mathbf{I}$ and surface normal $\mathbf{N}$:

$$\mathbf{R} = \mathbf{I} - 2(\mathbf{I} \cdot \mathbf{N})\mathbf{N}$$



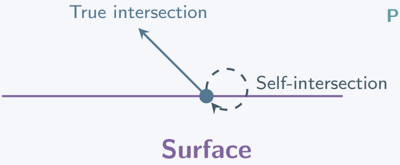
Raytraced Reflection

1. From the intersection point, cast a secondary ray in the direction of the reflection vector $\mathbf{R}$.
2. Calculate the colour at the intersection point of the secondary ray with the scene $\mathbf{I}_{\text{reflected}}$.
3. Combine the local colour $\mathbf{I}_{\text{local}}$ with the reflected colour:

$$\mathbf{I} = \mathbf{I}_{\text{local}} + \mathbf{k}_r \odot \mathbf{I}_{\text{reflected}}$$

Where $\mathbf{k}_r$ is the reflection coefficient (material property).

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 or set $t_{\min} = \varepsilon$

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

Acceleration Structures

Performance Considerations

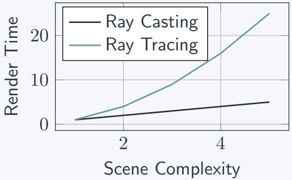
Computational Complexity

Basic ray tracing:

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

With secondary rays:

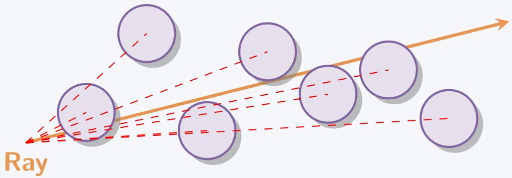
- Exponential growth with depth
- Multiple rays per intersection



Optimization strategies:

- Spatial data structures
- Parallel processing
- Early ray termination

The Naive Approach Problem



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

Computational Explosion

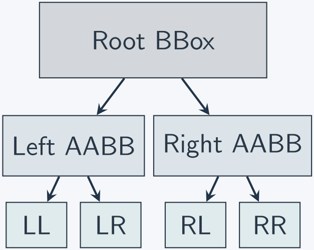
Complexity: For N objects and R rays $\rightarrow O(N \times R)$ intersection tests
Real scenes: Millions of triangles, millions of rays $\rightarrow$ Billions of tests!

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
- Works with any primitive type

BVH Construction and Traversal

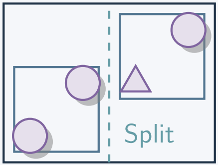
Construction Algorithm

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:

- **Surface Area Heuristic (SAH)**
- Median split
- Spatial splits



Spatial subdivision

The Hardware Revolution

From Software to Silicon

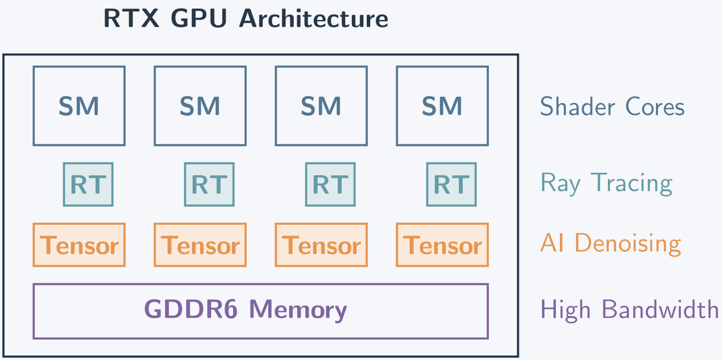
Hardware Features:

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

Performance Impact:

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

RTX Architecture



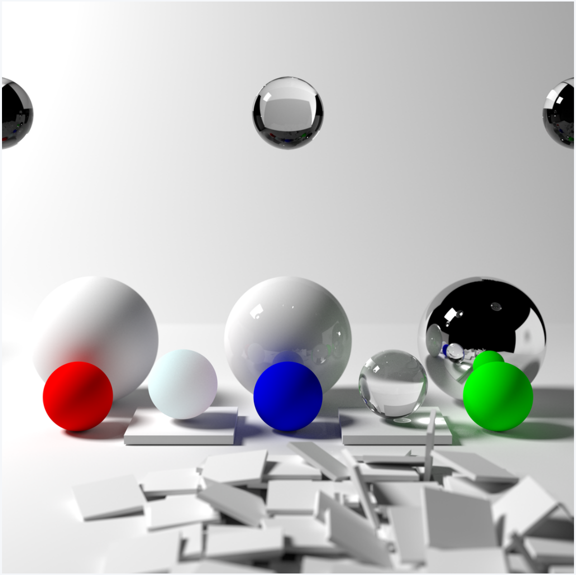
RT Core Functions

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

Result: Massive parallel ray processing with dedicated silicon

Path Tracing

Beyond Ray Tracing: Path Tracing



Path Tracing creates more realistic images

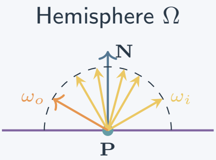
The Rendering Equation

The Holy Grail of Computer Graphics

$$L_o(\mathbf{P}, \omega_o, \lambda) = L_e(\mathbf{P}, \omega_o, \lambda) + \int_\Omega f(\mathbf{P}, \omega_i, \omega_o, \lambda) L_i(\mathbf{P}, \omega_i, \lambda)(\mathbf{N} \cdot \omega_i) d\omega_i$$

Components:

- L_i = Incoming radiance, L_o = Outgoing radiance
- L_e = Emitted light
- f = BRDF (material property)
- Ω = Hemisphere of directions
- ω_i = Incoming direction, ω_o = Outgoing direction
- $\mathbf{P}$ = Point on surface, $\mathbf{N}$ = Surface normal
- λ = Wavelength



The Challenge

Integral: Infinite directions to sample → Monte Carlo approximation needed

Path Tracing vs Ray Tracing

Ray Tracing Limitations

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

Path Tracing Advantages

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

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