

# CMP205: Computer Graphics



## Lecture 10: Ray Tracing 1

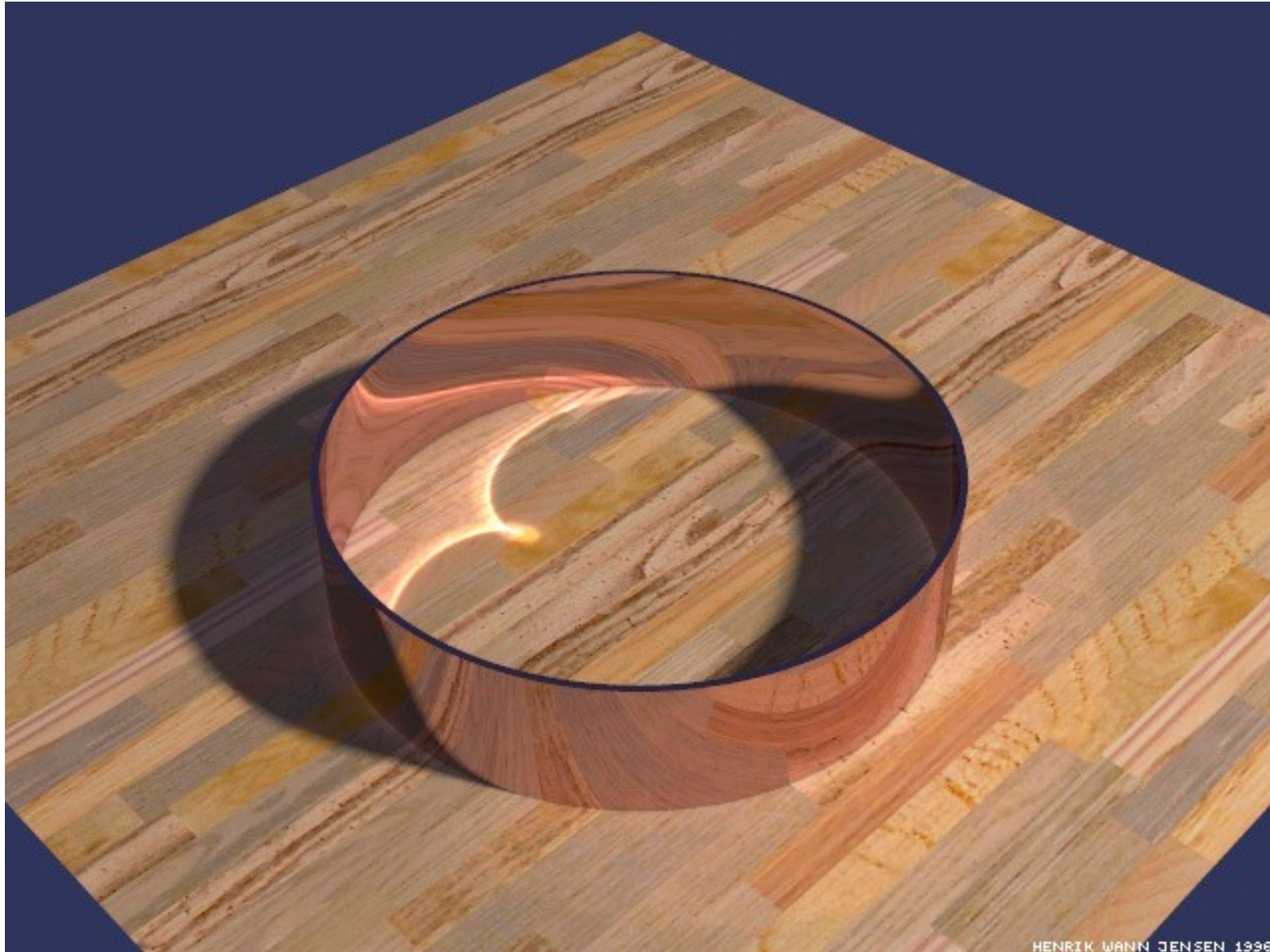
Mohamed Alaa El-Dien Aly  
Computer Engineering Department  
Cairo University  
Fall 2012

# Agenda

- What is Ray Tracing?
- Ray Tracing Vs Rasterization
- Ray Tracing Basics
  - Ray Generation
  - Ray Intersection
- Ray Tracing Program

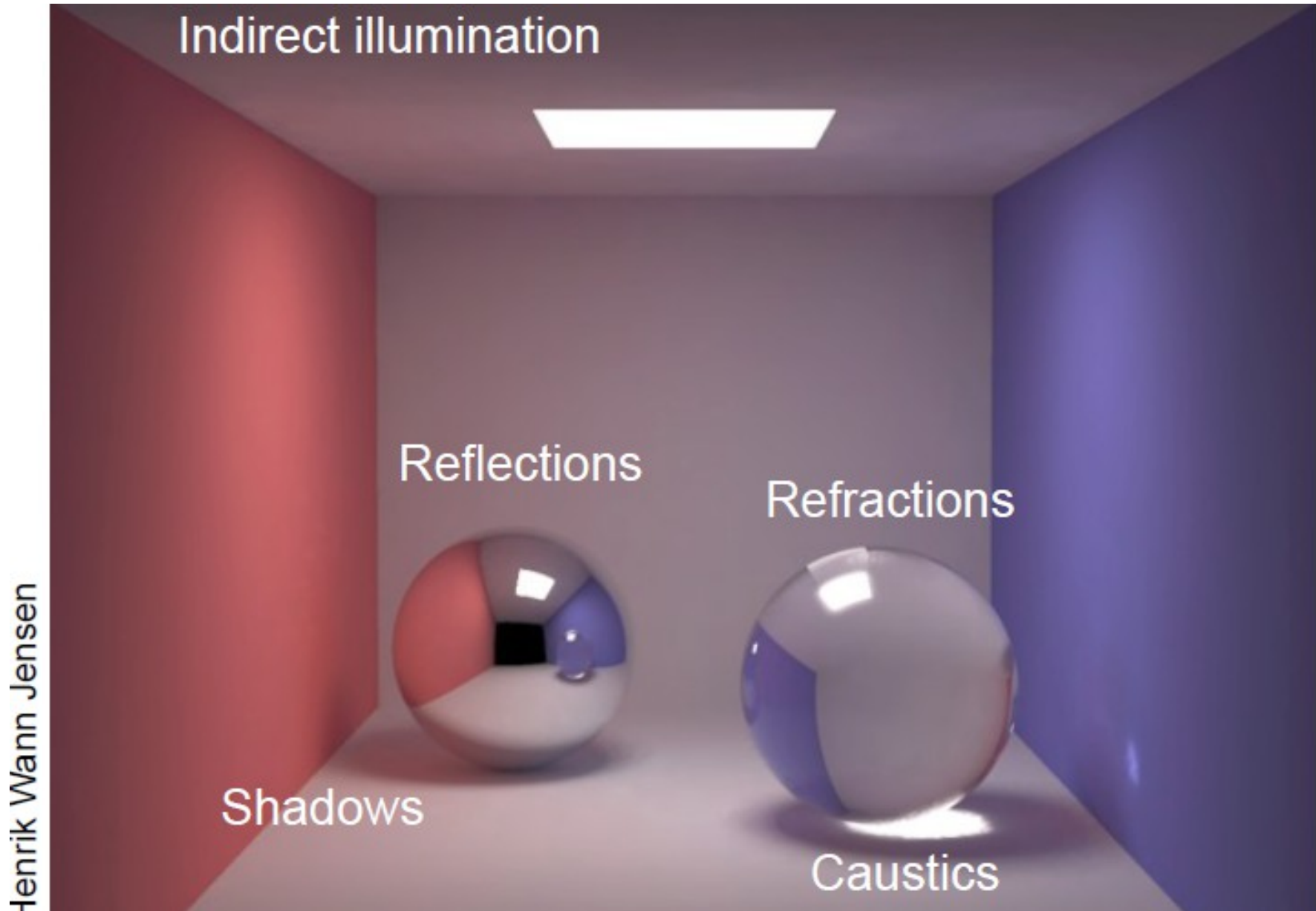
**Acknowledgment:** Some slides adapted from Steve Marschner, Maneesh Agrawala, and Fredo Durand

# What's Ray Tracing?



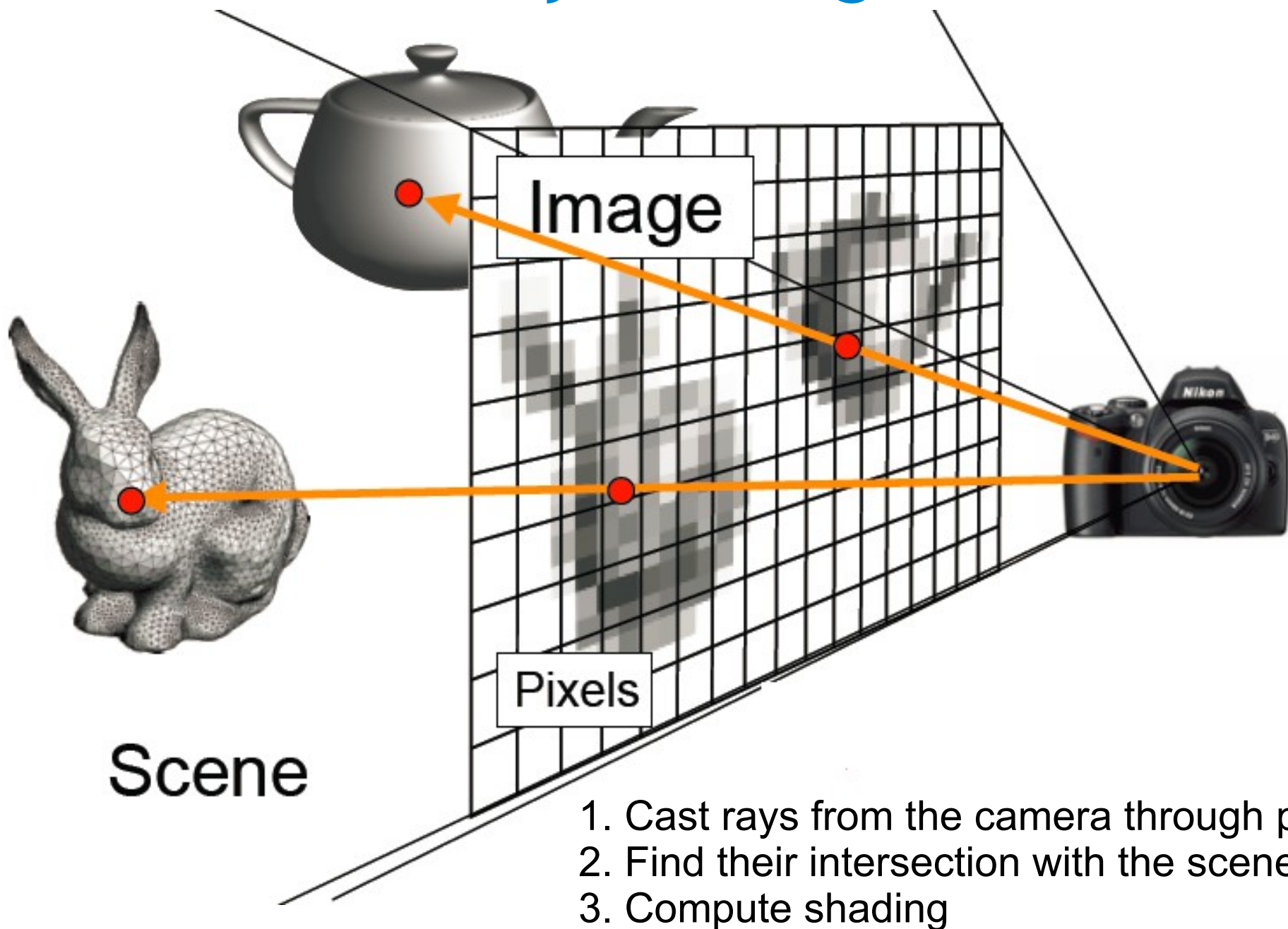
A rendering method that produces realistic images

# What's Ray Tracing?



Naturally handles reflections, shadows, refractions, ... etc.

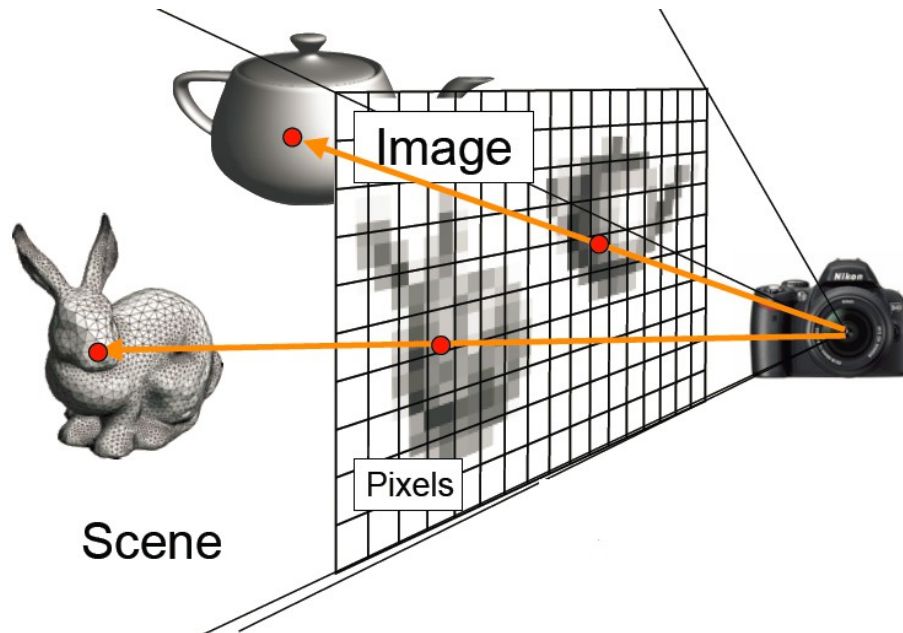
# Ray Tracing



# Ray Tracing Vs Rasterization

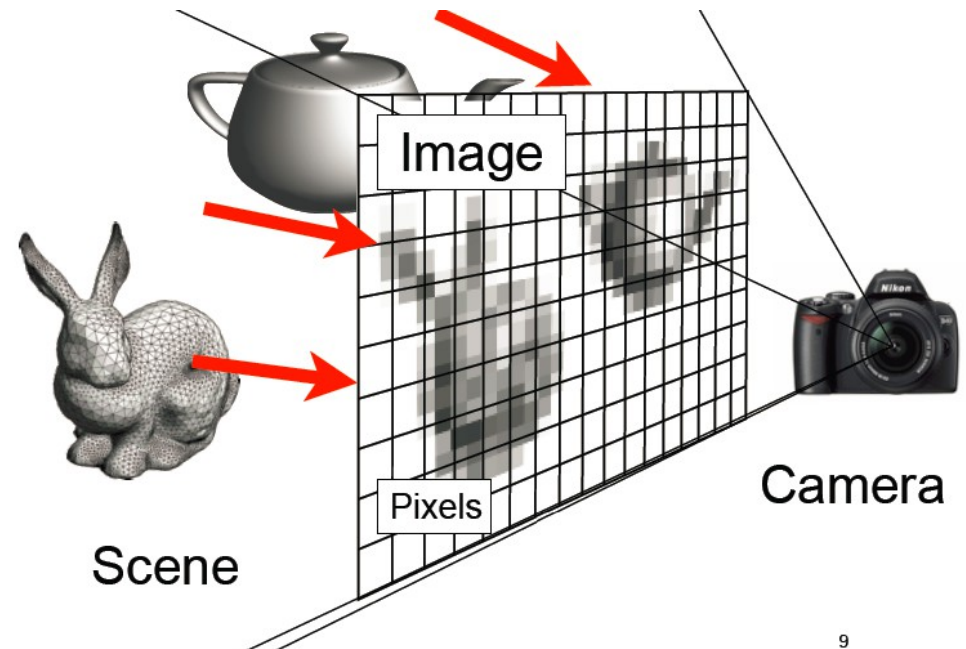
## Ray Tracing

```
For each pixel
  For each triangle
    Does ray hit triangle?
    Keep closest hit
  Compute shading
```



## Rasterization

```
For each triangle
  For each pixel
    Does triangle cover pixel?
    Keep closest hit
  Compute shading
```



# Ray Tracing Vs Rasterization

## Ray Tracing

```
For each pixel
  For each triangle
    Does ray hit triangle?
    Keep closest hit
    Compute shading
```

### Pros

- Can render anything that can be intersected with a ray
- Naturally handles shadows, transparency, reflection ... etc. using recursion

### Cons

- Hard to implement in hardware
- Traditionally too slow for interactive applications
- Becoming faster and faster!

## Rasterization

```
For each triangle
  For each pixel
    Does triangle cover pixel?
    Keep closest hit
    Compute shading
```

### Pros

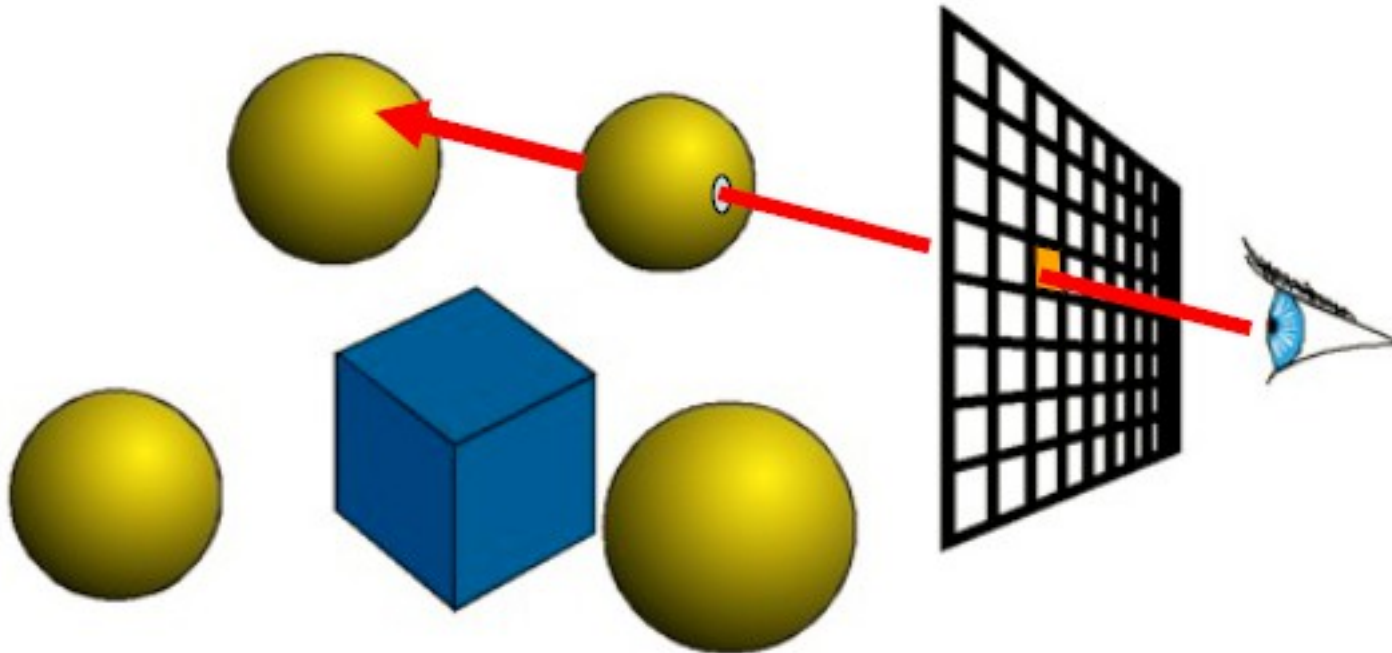
- Much much faster
- Readily available in GPUs
- Parallelizable

### Cons

- Limited to certain primitives, esp. triangles
- Faceting and shading artifacts
- No unified handling of shadows, reflection, transparency (only approx.)

# Ray Tracing

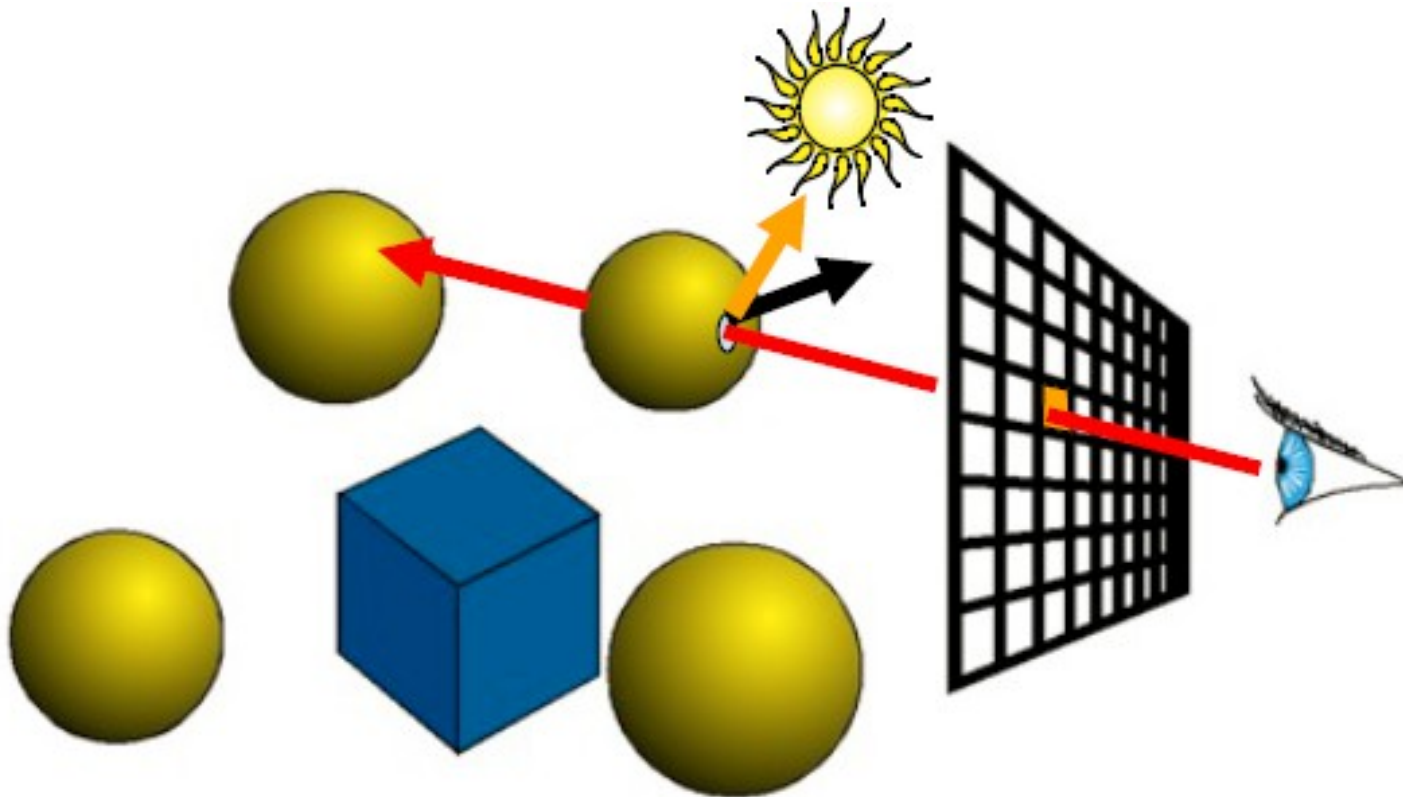
```
For each pixel  
  Construct a ray from the eye  
  For each object in the scene  
    Find intersection point (and surface normal)  
  Keep if closest
```





# Ray Tracing

```
For each pixel
  Construct a ray from the eye
  For each object in the scene
    Find intersection point (and surface normal)
  Keep if closest
  Compute Shading
```



# Ray Generation

```
For each pixel
```

```
  Construct a ray from the eye
```

```
  For each object in the scene
```

```
    Find intersection point (and surface normal)
```

```
    Keep if closest
```

```
    Compute Shading
```

# Ray Generation

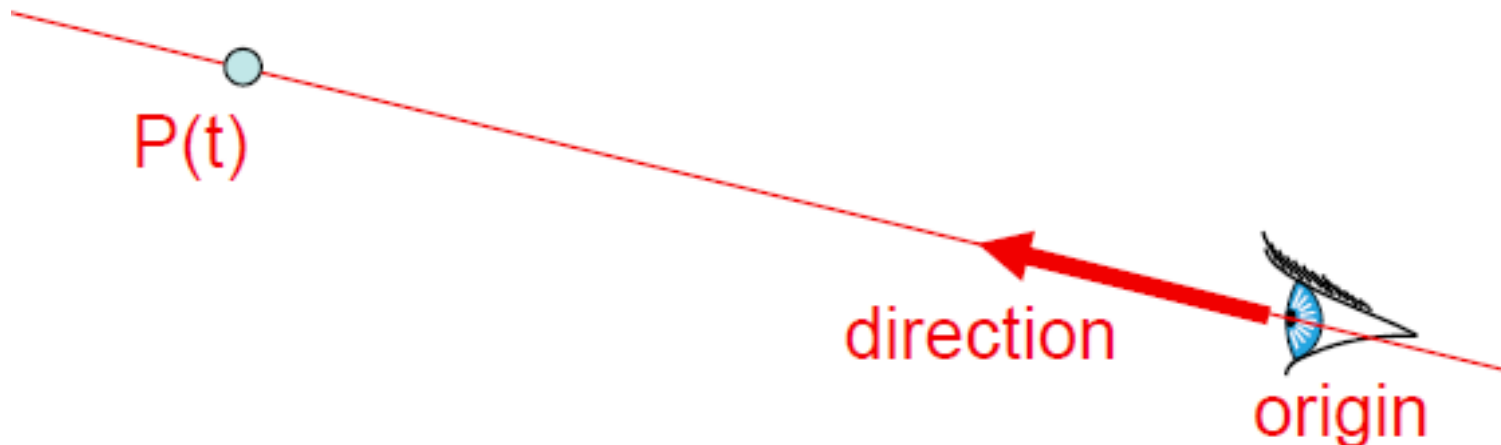
A ray is composed of:

- Starting point  $e$
- Direction vector  $d$

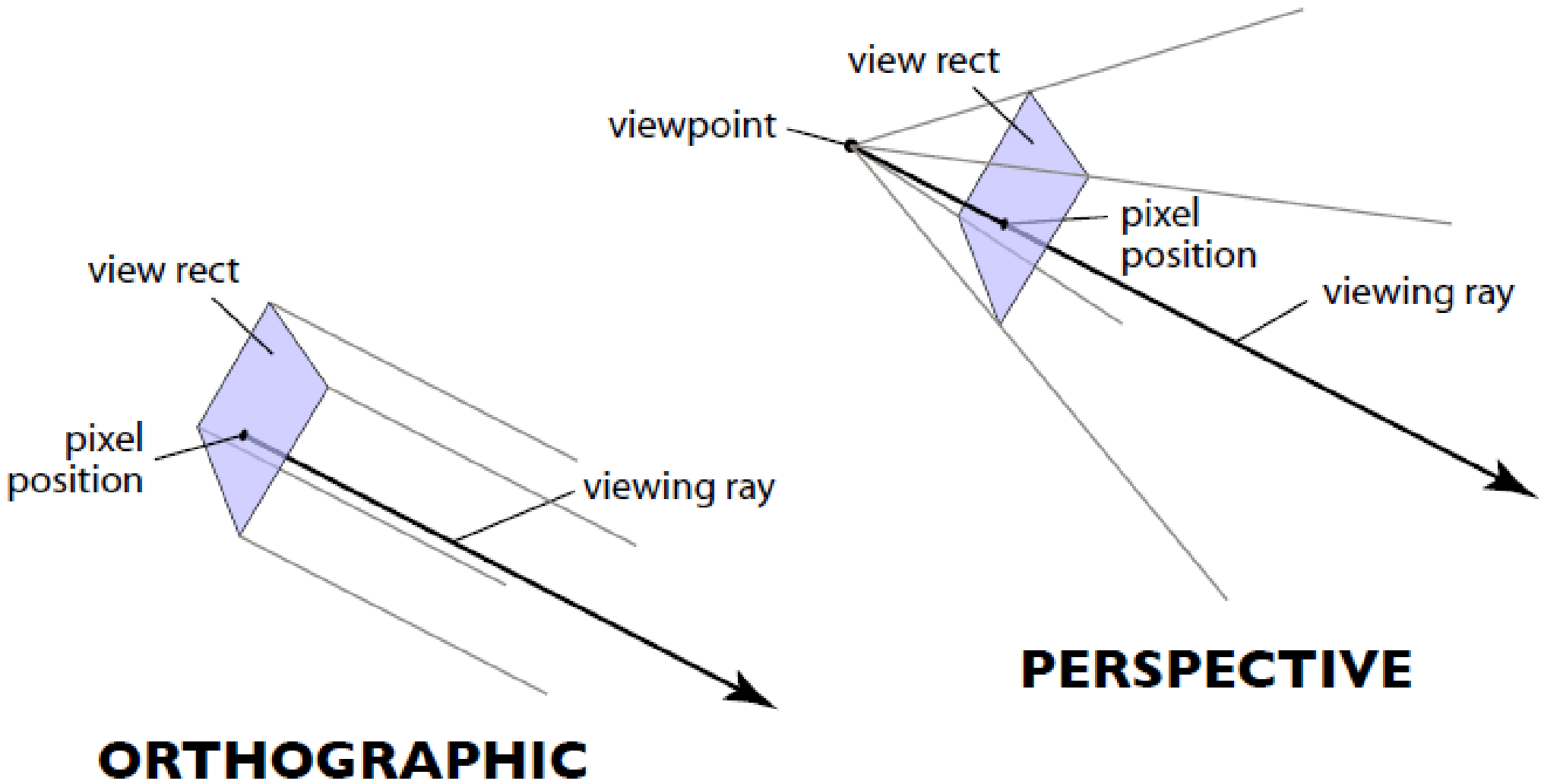
Parametric equation:  $p(t) = e + t d$

$$t = 0 \rightarrow p(t) = e$$

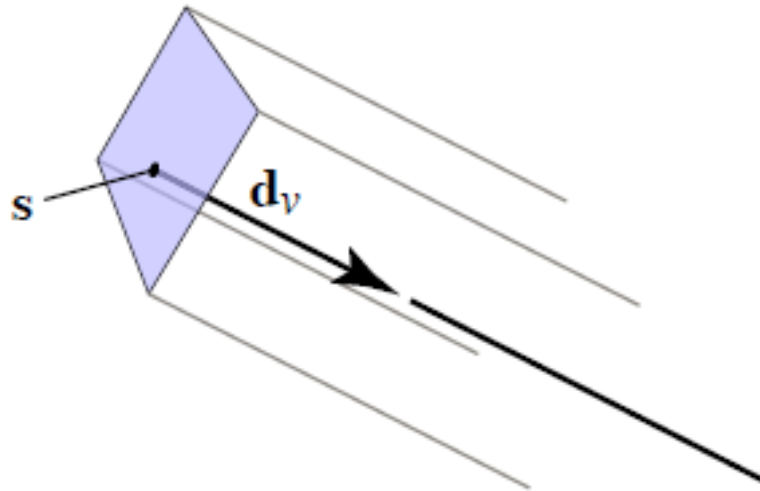
Find smallest  $t > 0$  such that  $p(t)$  lies on a surface!



# Ray Generation



# Ray Generation: Orthographic



$$p(t) = s + t d_v$$

Where is the viewing rectangle in World Coordinates?

# Ray Generation: Orthographic

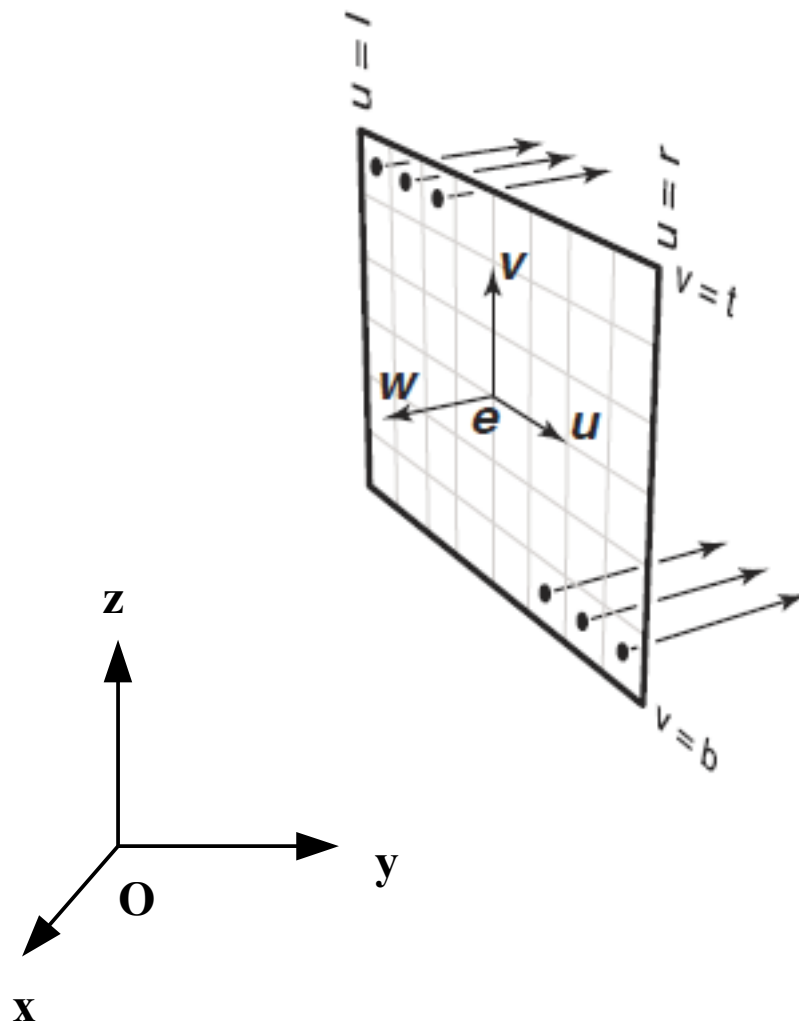
- Camera basis:  $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$
- Camera position:  $\mathbf{e}$
- View rectangle specified by  $l$ ,  $r$ ,  $t$ ,  $b$
- Screen point in  $uv$ -plane:  $(u_s, v_s)$

Screen point:  $\mathbf{s} = \mathbf{e} + u_s \mathbf{u} + v_s \mathbf{v}$

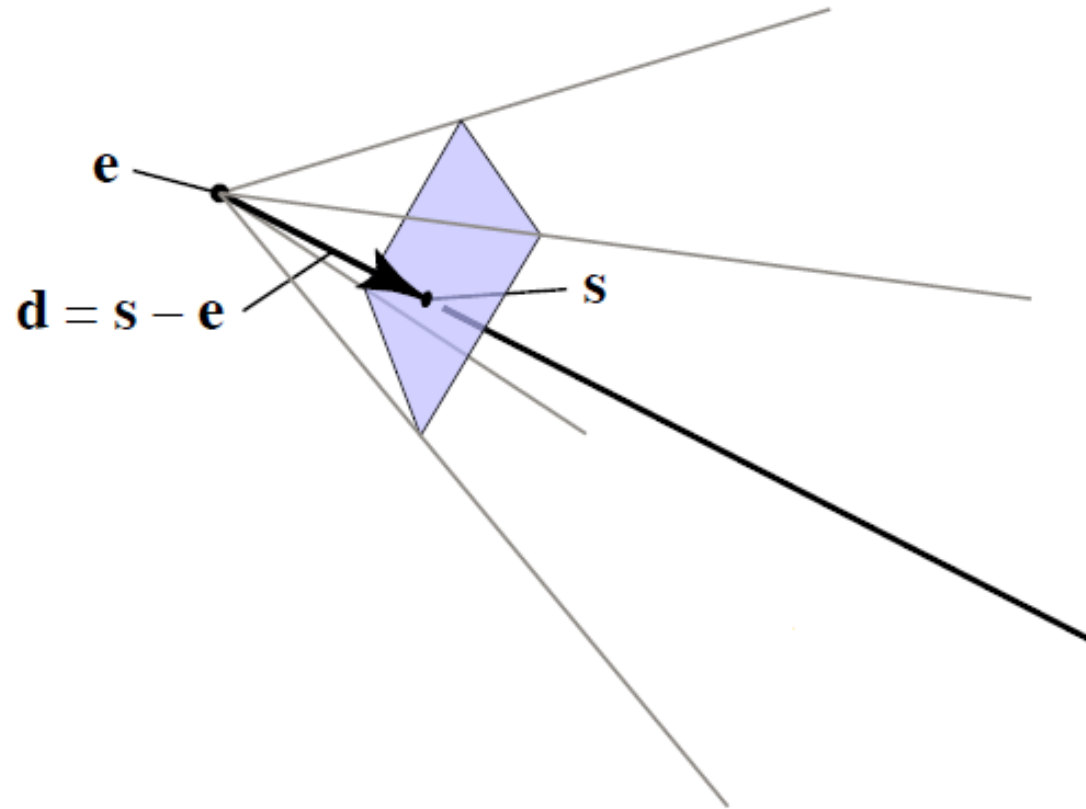
Direction:  $\mathbf{d} = -\mathbf{w}$

Starting Point:  $\mathbf{s}$

Ray:  $\mathbf{p}(t) = \mathbf{s} + t \mathbf{d}$



# Ray Generation: Perspective



$$p(t) = e + t d$$

Where is the viewing rectangle in World Coordinates?

# Ray Generation: Perspective

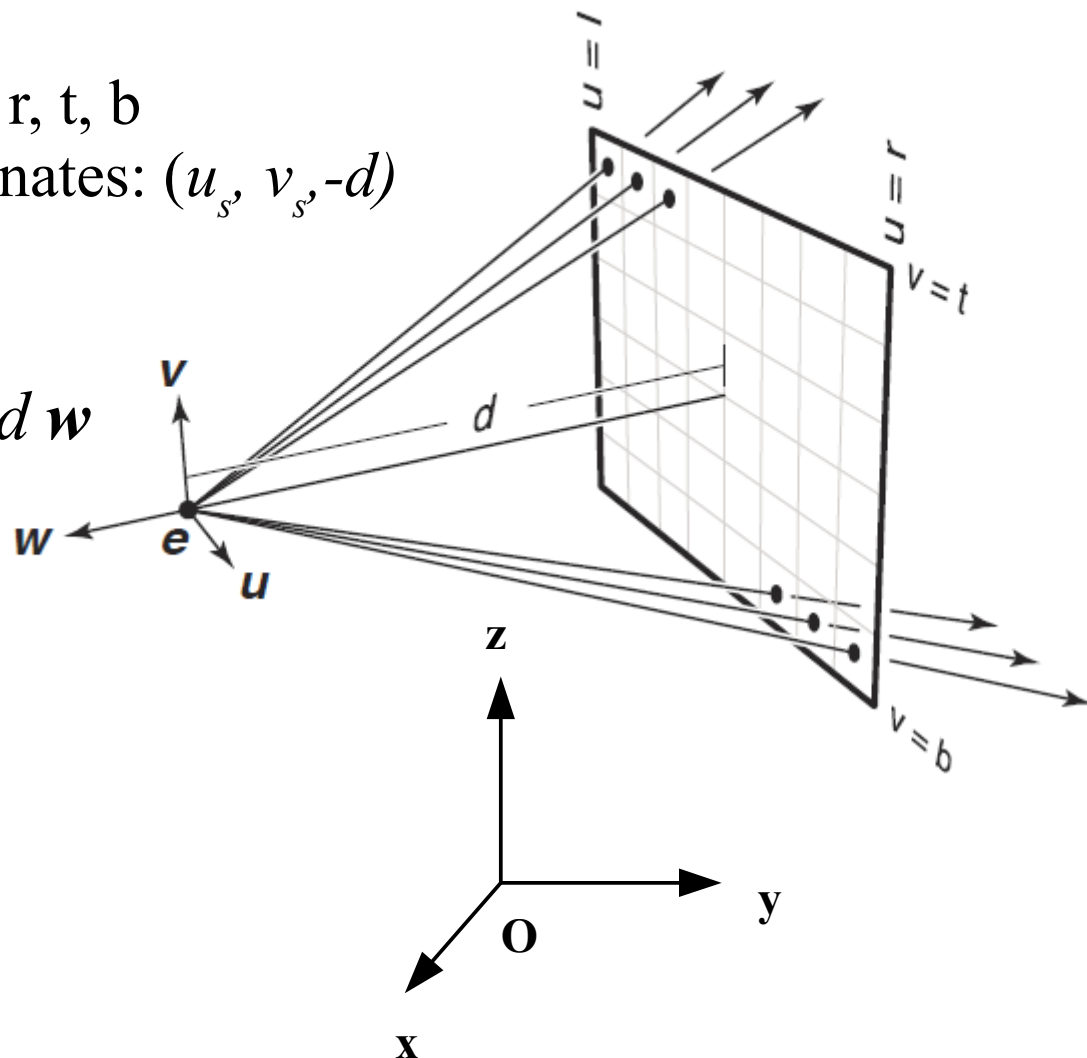
- Camera basis:  $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$
- Camera position:  $\mathbf{e}$
- View rectangle specified by  $l$ ,  $r$ ,  $t$ ,  $b$
- Screen point in camera coordinates:  $(u_s, v_s, -d)$

Screen point:  $\mathbf{s} = \mathbf{e} + u_s \mathbf{u} + v_s \mathbf{v} - d \mathbf{w}$

Direction:  $\mathbf{d} = \mathbf{s} - \mathbf{e}$

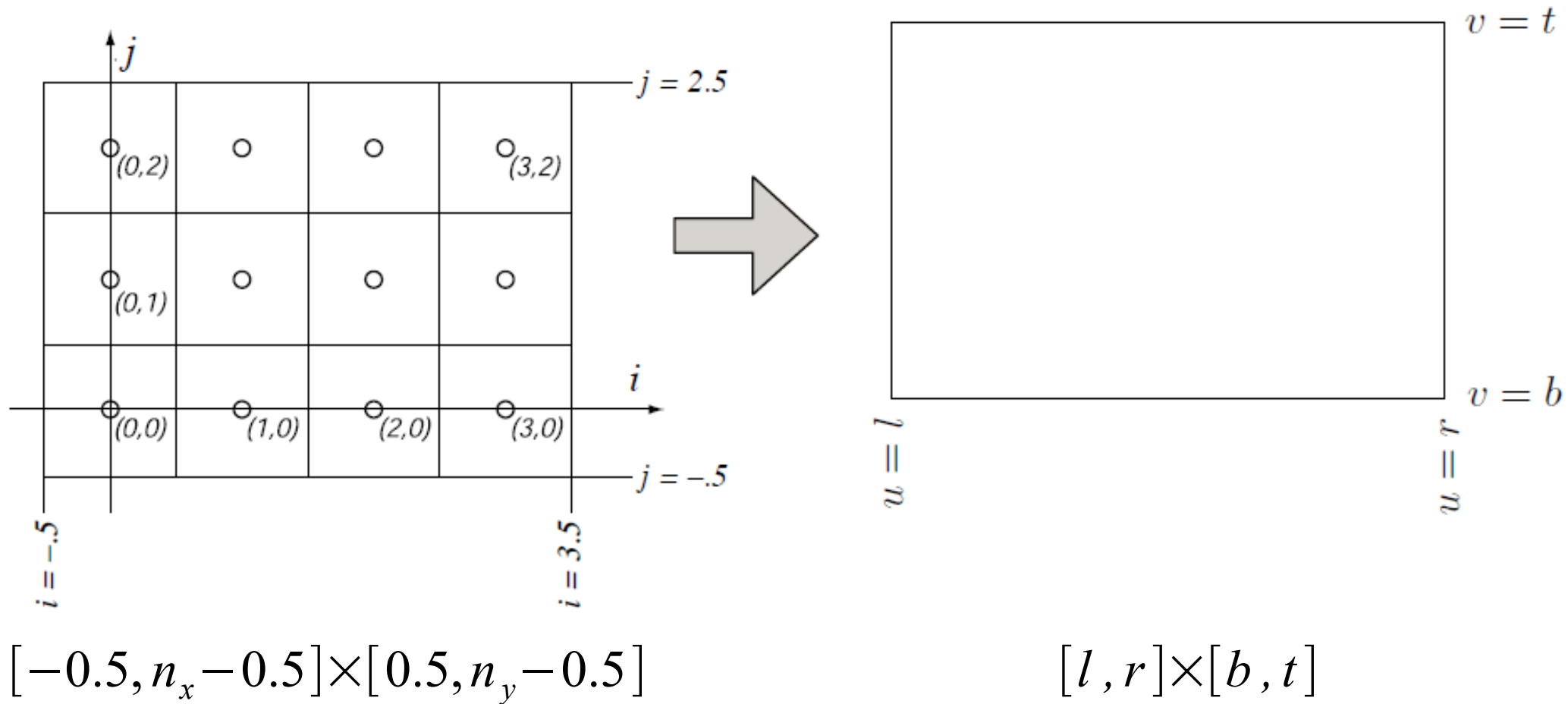
Starting Point:  $\mathbf{e}$

Ray:  $\mathbf{p}(t) = \mathbf{e} + t \mathbf{d}$



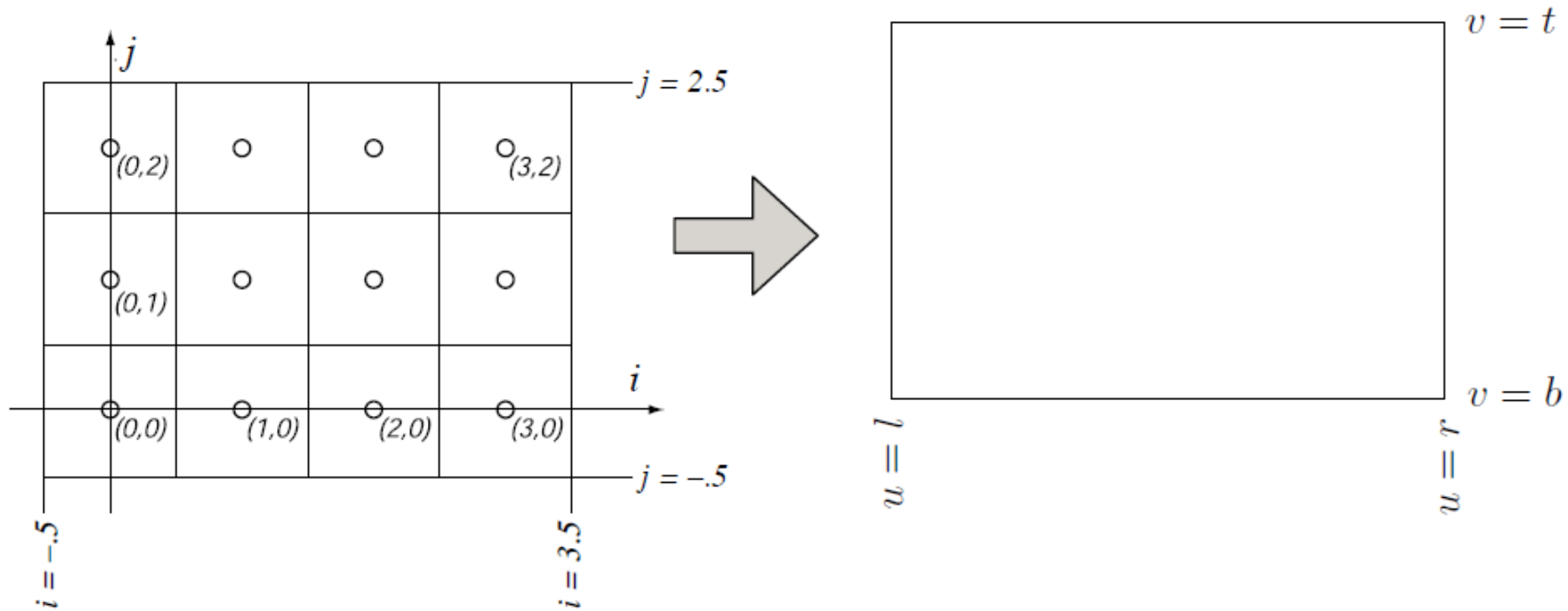


# Ray Generation: Pixel-to-Image



How to convert from pixel coordinates  $(i, j)$  to  $uv$  coordinates  $(u_s, v_s)$ ?

# Ray Generation: Pixel-to-Image



$$[-0.5, n_x - 0.5] \times [0.5, n_y - 0.5]$$

$$[l, r] \times [b, t]$$

Windowing Transformation: Translate, Scale, Translate

$$u_s = l + \frac{r-l}{n_x} (i+0.5) \quad \& \quad v_s = b + \frac{t-b}{n_y} (j+0.5)$$

# Ray Intersection

```
For each pixel
  Construct a ray from the eye
  For each object in the scene
    Intersection (ray, t0, t1)
  Keep if closest
  Compute Shading
```

Finds the intersection (and surface normal) for  $t \geq t_0$  and  $t \leq t_1$

# Ray Intersection: Sphere

Ray parametric equation:  $\mathbf{p}(t) = \mathbf{e} + t \mathbf{d}$

Sphere implicit equation:  $\|\mathbf{p} - \mathbf{c}\|^2 - r^2 = 0$  for center  $\mathbf{c}$  & radius  $r$

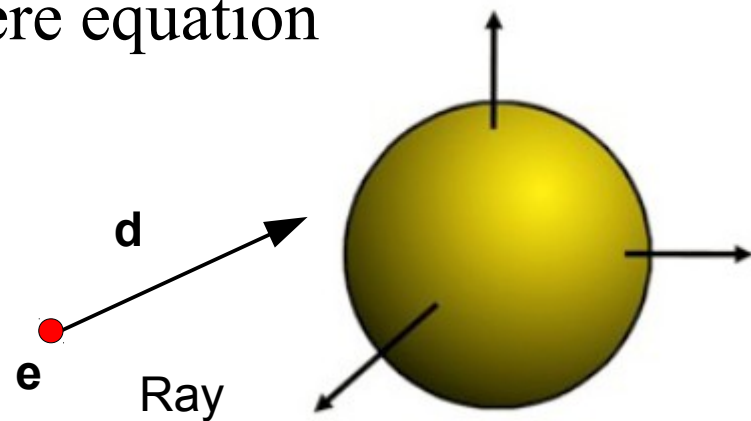
Intersect  $\rightarrow$  Substitute ray equation into sphere equation and solve for  $t$

$$\|\mathbf{e} + t \mathbf{d} - \mathbf{c}\|^2 - r^2 = 0$$

$$(\mathbf{e} + t \mathbf{d} - \mathbf{c})^T (\mathbf{e} + t \mathbf{d} - \mathbf{c}) - r^2 = 0$$

$$(\mathbf{d}^T \mathbf{d}) t^2 + 2 \mathbf{d}^T (\mathbf{e} - \mathbf{c}) t + (\mathbf{e} - \mathbf{c})^T (\mathbf{e} - \mathbf{c}) - r^2 = 0$$

Quadratic Equation in  $t$  !



# Ray Intersection: Sphere

Quadratic Equation in  $t$ :  $At^2 + Bt + C = 0$

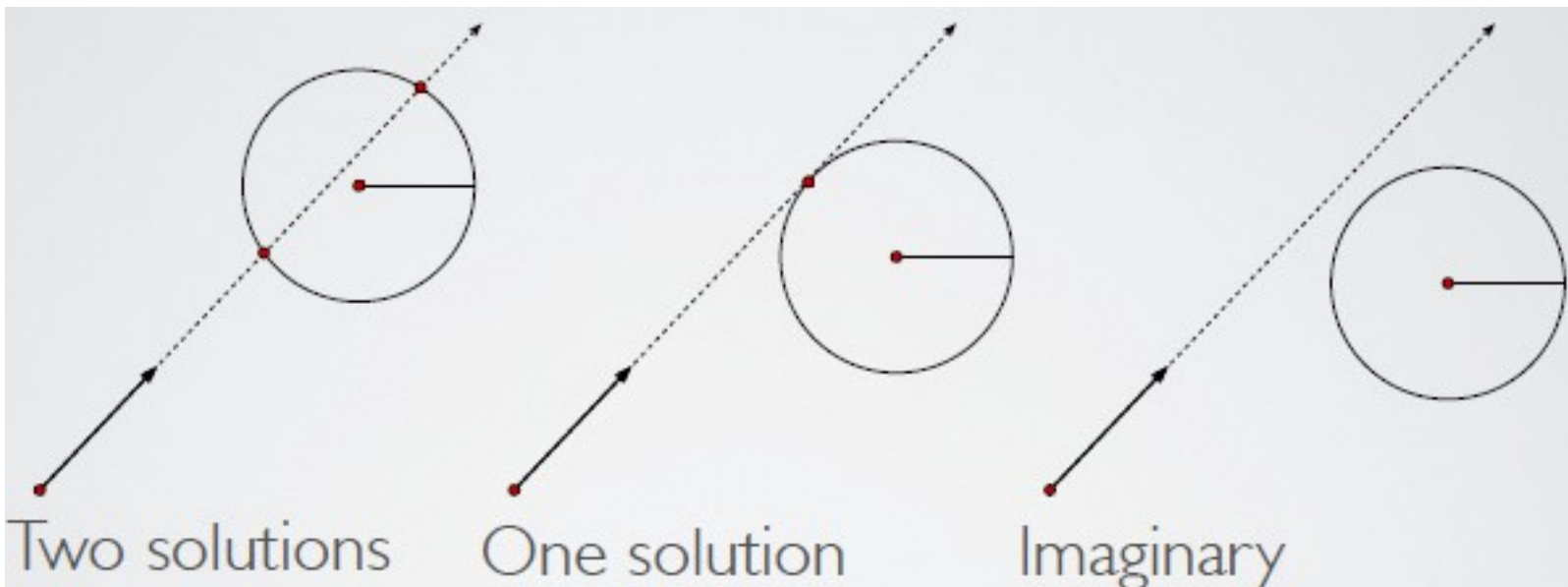
$$t = \frac{-B \pm \sqrt{D}}{2A}$$

Discriminant:  $D = B^2 - 4AC$

$D > 0$

$D = 0$

$D < 0$



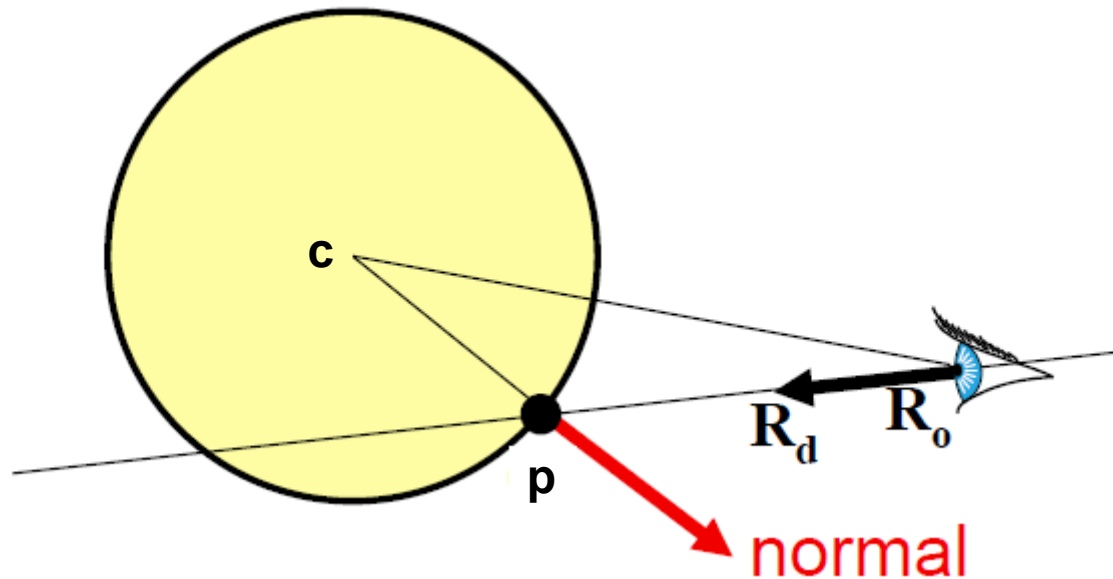
Which  $t$  to choose?

Smallest  $t > t_{min}$

# Ray Intersection: Sphere

What about surface normal?

$$\mathbf{n} = \frac{\mathbf{p} - \mathbf{c}}{\|\mathbf{p} - \mathbf{c}\|}$$



# Ray Intersection: Plane

Ray parametric equation:  $\mathbf{p}(t) = \mathbf{e} + t \mathbf{d}$

Plane equation:  $\mathbf{n}^T \mathbf{p} + D = 0$  where  $D = -\mathbf{n}^T \mathbf{p}_0$

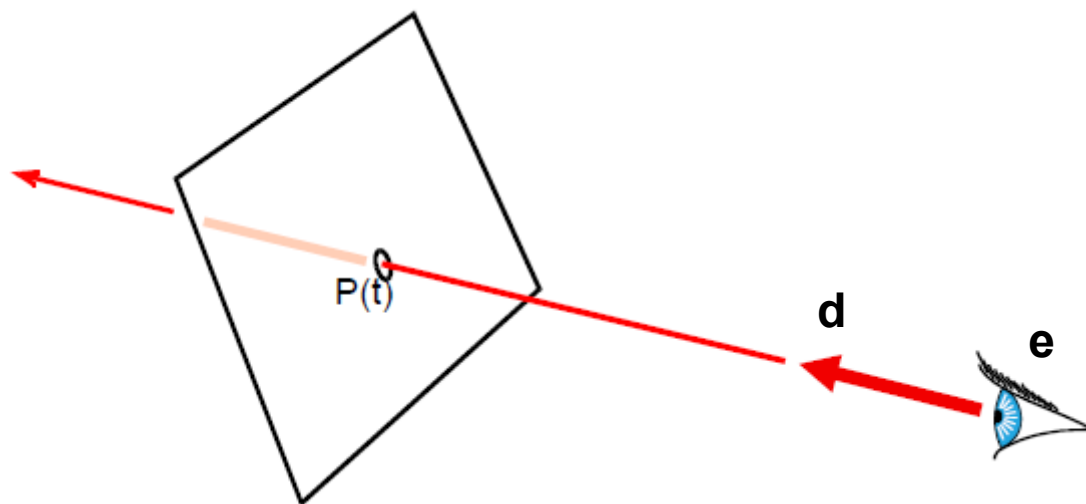
Intersect  $\rightarrow$  Substitute ray equation into plane equation and solve for  $t$

$$\mathbf{n}^T (\mathbf{e} + t \mathbf{d}) + D = 0$$

$$t = -\frac{D + \mathbf{n}^T \mathbf{e}}{\mathbf{n}^T \mathbf{d}}$$

What if  $\mathbf{n}^T \mathbf{d} = 0$ ?

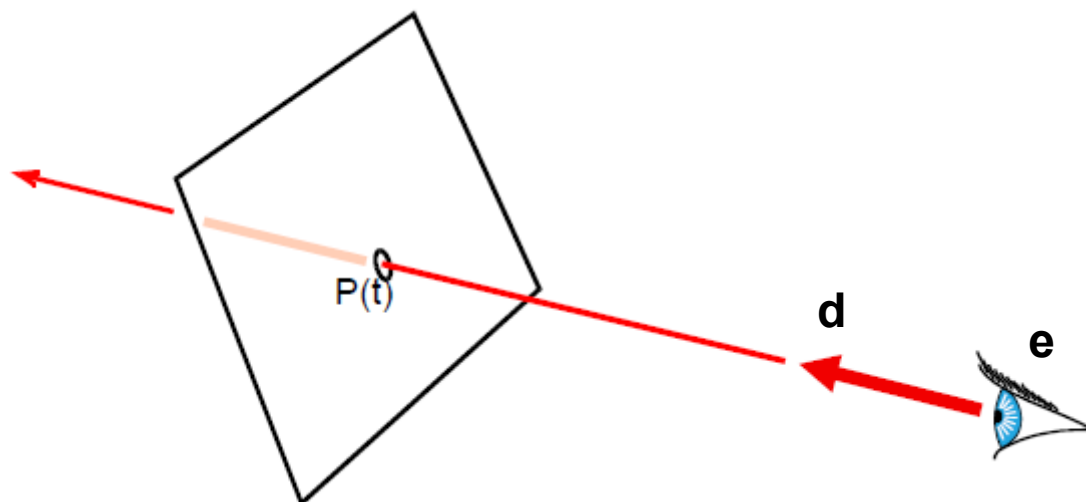
Ray parallel to Plane !



# Ray Intersection: Plane

What about surface normal?

Already given !



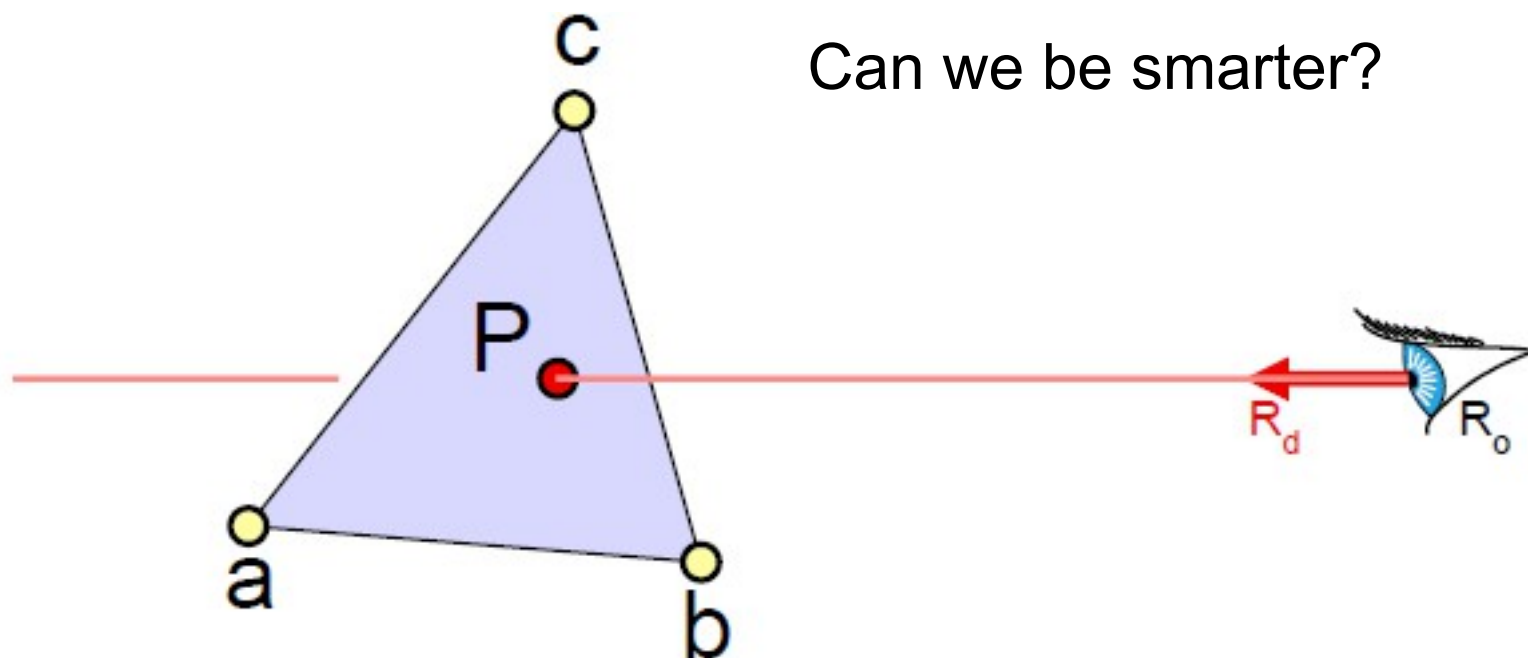


# Ray Intersection: Triangle

Straightforward approach ?

1. Intersect ray with triangle's plane
2. Find Barycentric coordinates of intersection point
3. Decide whether inside or outside triangle

$$0 \leq \alpha, \beta, \gamma \leq 1$$



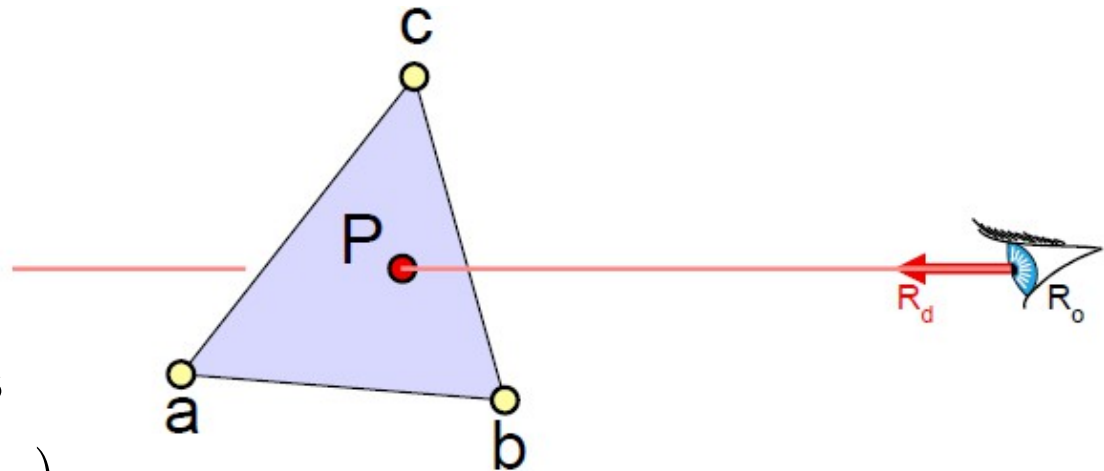
# Ray Intersection: Triangle

Ray parametric equation:  $\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$

Triangle equation:  $\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$

Intersect  $\rightarrow$  Substitute ray equation into triangle equation and solve for  $t$ ,  $\beta$ , and  $\gamma$ . Inside if  $\beta + \gamma < 1$  and  $\beta \& \gamma > 0$

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



Three equations in three unknowns

$$x_e + t x_d = x_a + \beta(x_b - x_a) + \gamma(x_c - x_a)$$

$$y_e + t y_d = y_a + \beta(y_b - y_a) + \gamma(y_c - y_a)$$

$$z_e + t z_d = z_a + \beta(z_b - z_a) + \gamma(z_c - z_a)$$

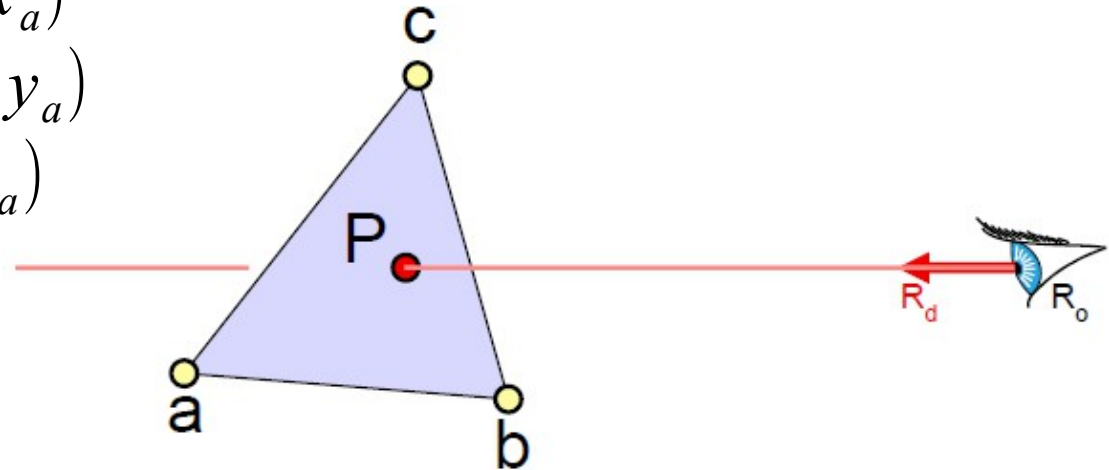
# Ray Intersection: Triangle

Three equations in three unknowns

$$x_e + t x_d = x_a + \beta(x_b - x_a) + \gamma(x_c - x_a)$$

$$y_e + t y_d = y_a + \beta(y_b - y_a) + \gamma(y_c - y_a)$$

$$z_e + t z_d = z_a + \beta(z_b - z_a) + \gamma(z_c - z_a)$$



$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

Solve for  $t$ ,  $\beta$ , and  $\gamma$

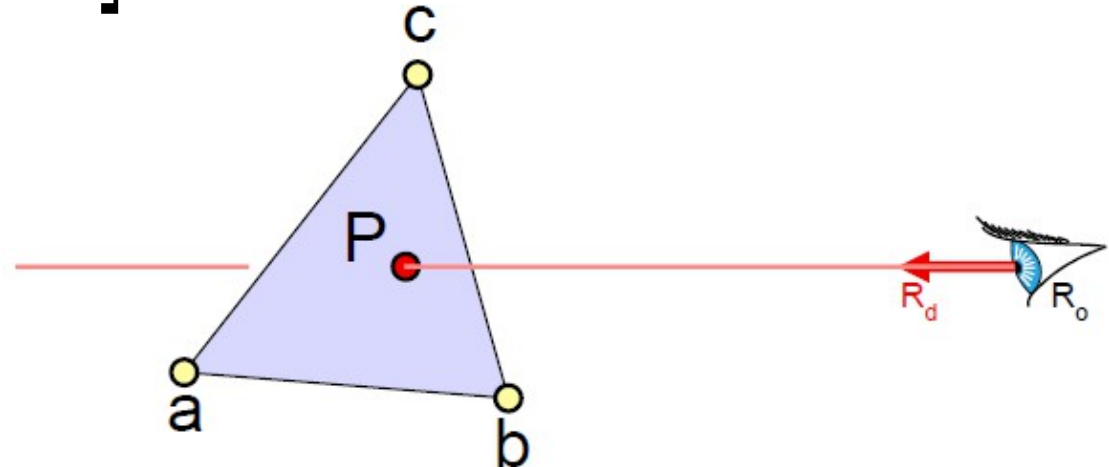
# Ray Intersection: Triangle

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

Solve for  $t$ ,  $\beta$ , and  $\gamma$

$$\beta = \frac{\begin{vmatrix} x_a - x_e & x_a - x_c & x_d \\ y_a - y_e & y_a - y_c & y_d \\ z_a - z_e & z_a - z_c & z_d \end{vmatrix}}{|A|}$$

$$\gamma = \frac{\begin{vmatrix} x_a - x_b & x_a - x_e & x_d \\ y_a - y_b & y_a - y_e & y_d \\ z_a - z_b & z_a - z_e & z_d \end{vmatrix}}{|A|}$$



$$t = \frac{\begin{vmatrix} x_a - x_b & x_a - x_c & x_a - x_e \\ y_a - y_b & y_a - y_c & y_a - y_e \\ z_a - z_b & z_a - z_c & z_a - z_e \end{vmatrix}}{|A|}$$

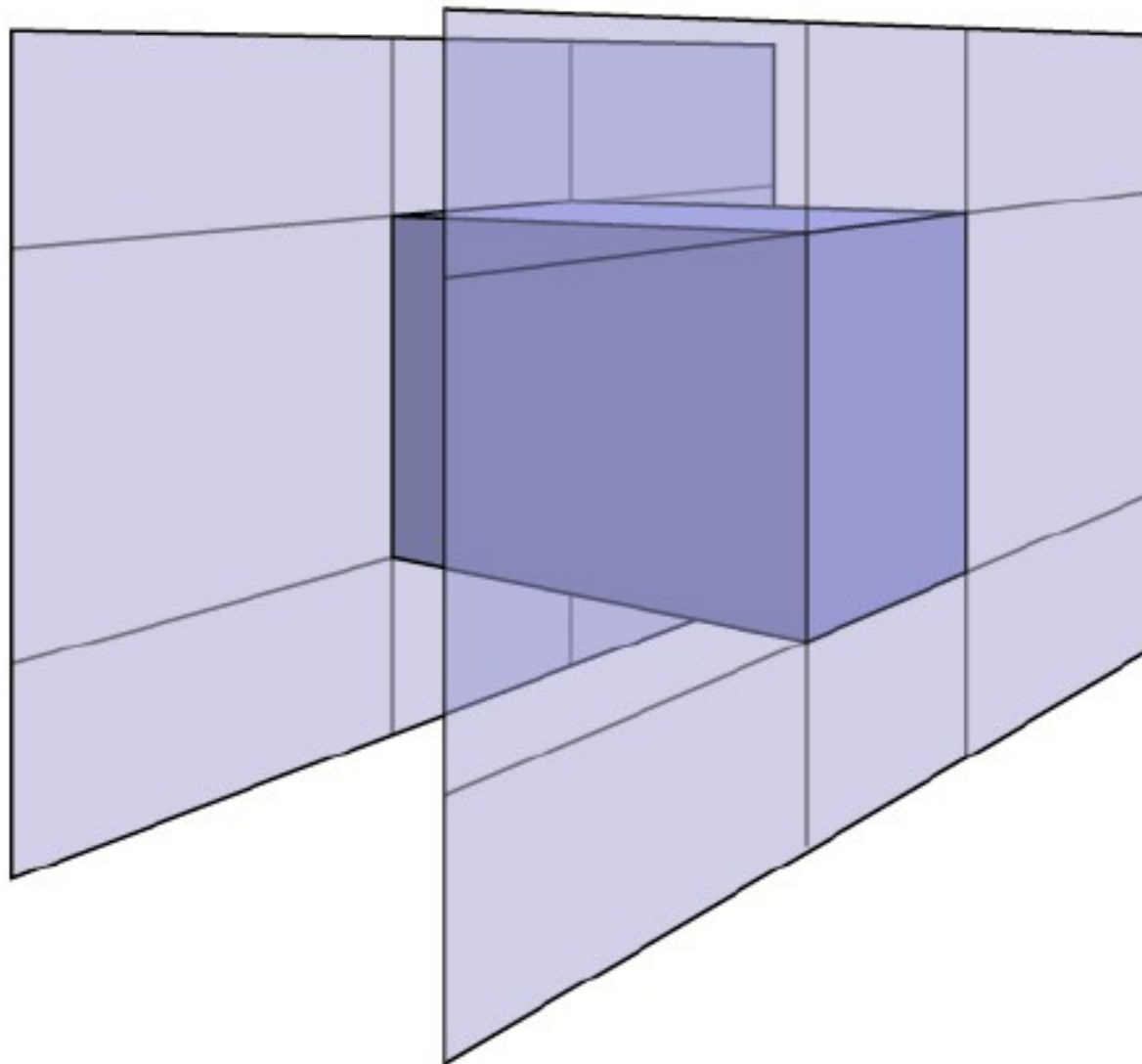
# Ray Intersection: Triangle

- Advantages?
  - Efficient
  - No need to store plane equations
  - Compute Barycentric coordinates and check in one step!

# Ray Intersection: Box

Want the intersection of the ray  
with a 3D box.

Do it for 2D first !



# Ray Intersection: Box

Ray equation:  $\mathbf{p}(t) = \mathbf{e} + t \mathbf{d}$

2D Box:  $x_p = x_{min}, x_p = x_{max}, y_p = y_{min}, y_p = y_{max}$

$$x_e + t_{xmin} x_d = x_{min}$$

$$\rightarrow t_{xmin} = (x_{min} - x_e) / x_d$$

$$x_e + t_{xmax} x_d = x_{max}$$

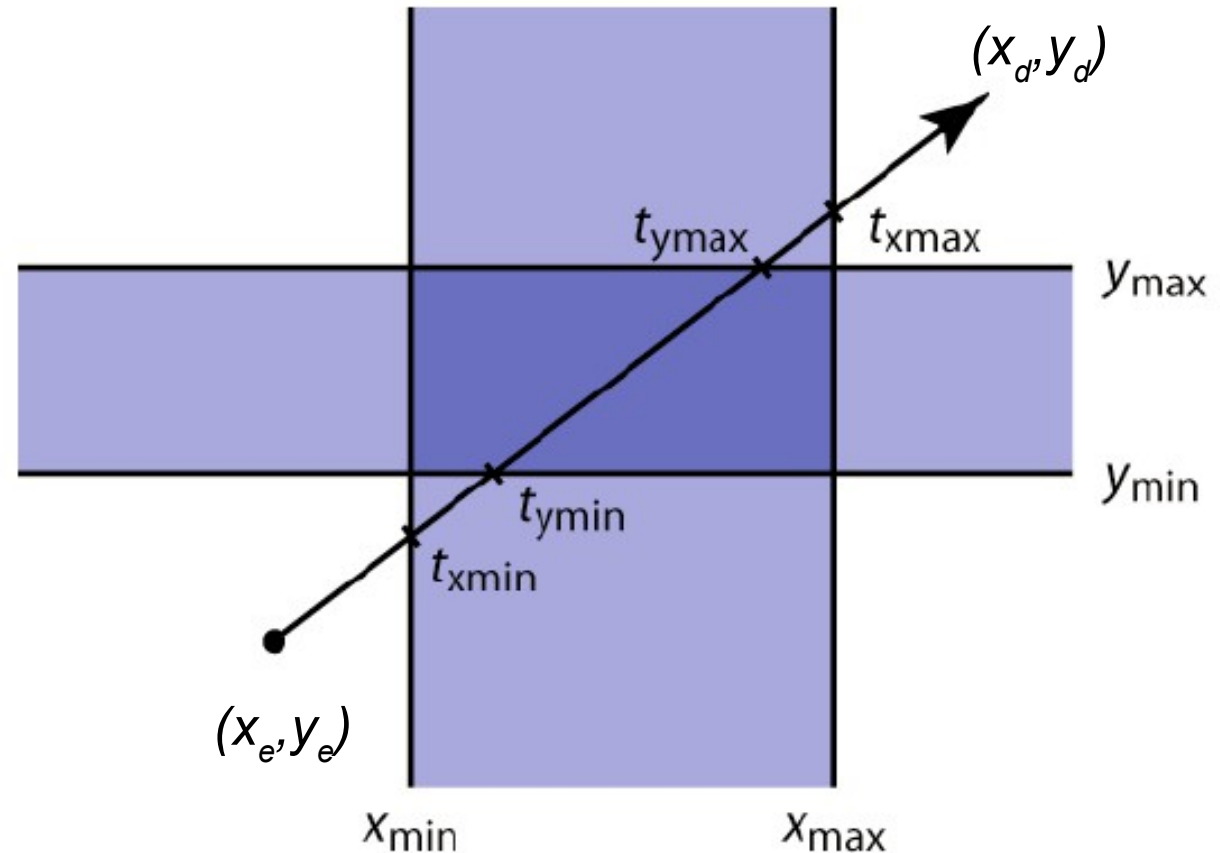
$$\rightarrow t_{xmax} = (x_{max} - x_e) / x_d$$

$$y_e + t_{ymin} y_d = y_{min}$$

$$\rightarrow t_{ymin} = (y_{min} - y_e) / y_d$$

$$y_e + t_{ymax} y_d = y_{max}$$

$$\rightarrow t_{ymax} = (y_{max} - y_e) / y_d$$



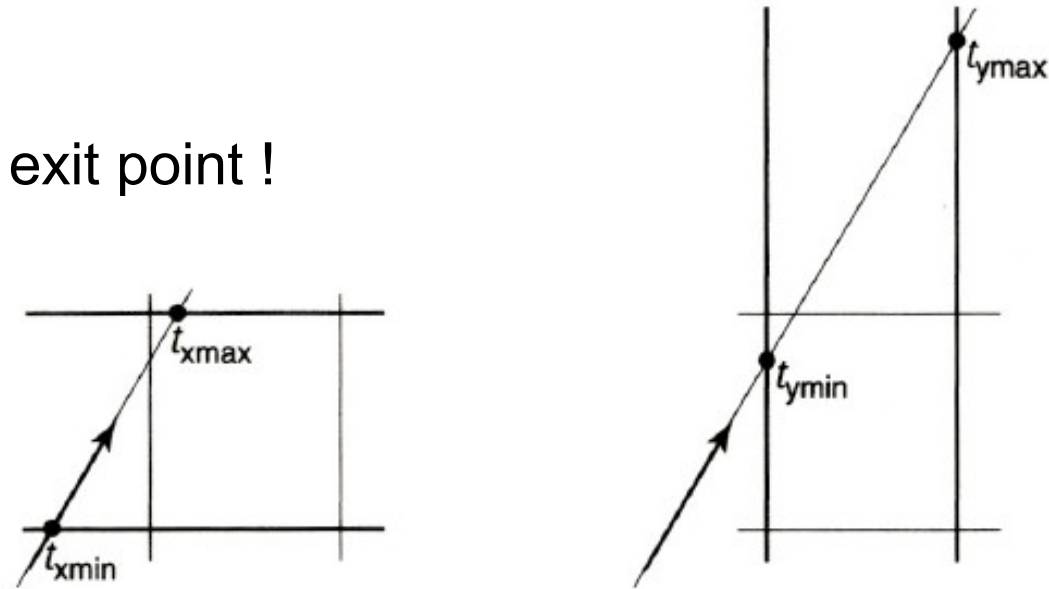
# Ray Intersection: Box

Each intersection gives an interval

We want the last entry point and first exit point !

$$t_{min} = \max(t_{xmin}, t_{ymin})$$

$$t_{max} = \min(t_{xmax}, t_{ymax})$$



Intersection?

$$\rightarrow t_{min} < t_{max}$$

Intersection point?

$$\rightarrow p(t_{min})$$

$$t \in [t_{xmin}, t_{xmax}]$$

A horizontal number line with a solid black segment between two points, representing the interval  $t \in [t_{xmin}, t_{xmax}]$ .

$$t \in [t_{ymin}, t_{ymax}]$$

A horizontal number line with a solid black segment between two points, representing the interval  $t \in [t_{ymin}, t_{ymax}]$ .

$$t \in [t_{xmin}, t_{xmax}] \cap [t_{ymin}, t_{ymax}]$$

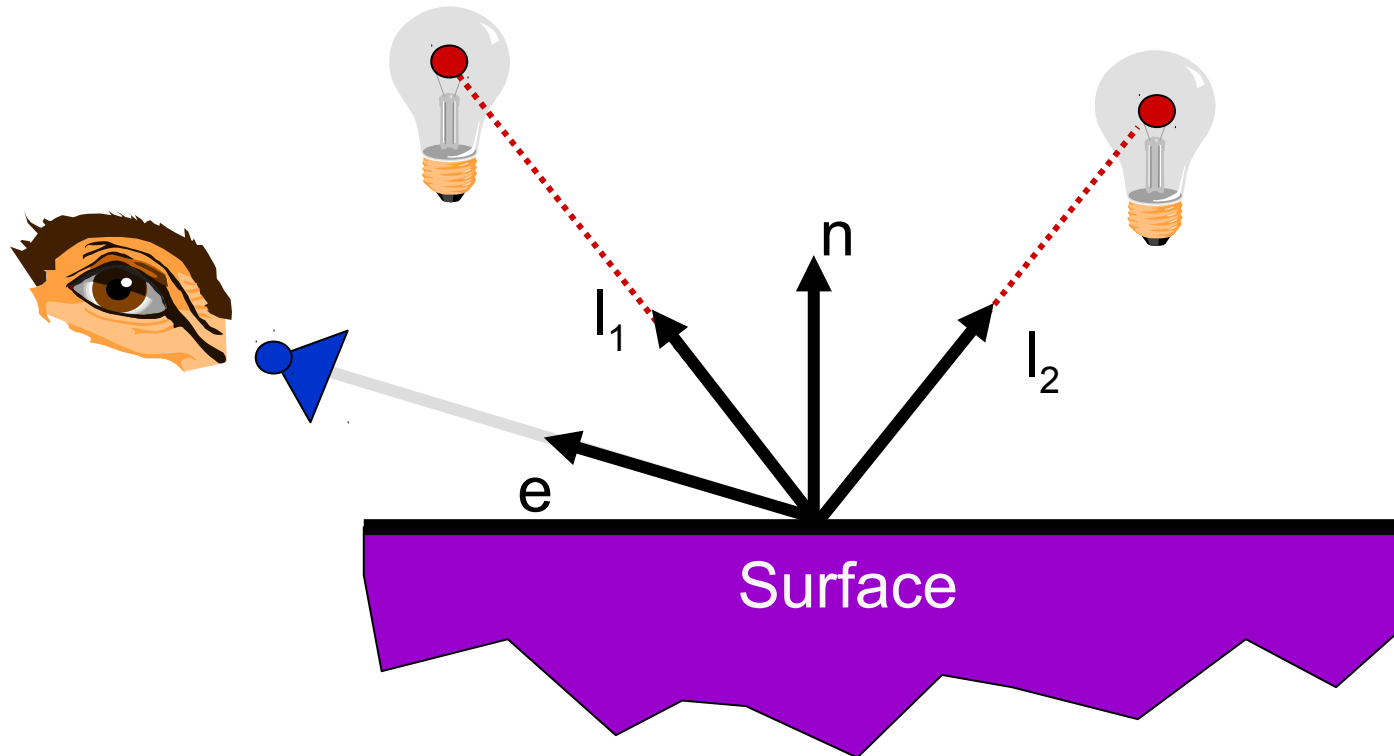
A horizontal number line with a solid black segment between two points, representing the intersection of the two intervals  $t \in [t_{xmin}, t_{xmax}] \cap [t_{ymin}, t_{ymax}]$ .



# Shading

```
For each pixel
  Construct a ray from the eye
  For each object in the scene
    Find intersection point (and surface normal)
    Keep if closest
  Compute Shading
```

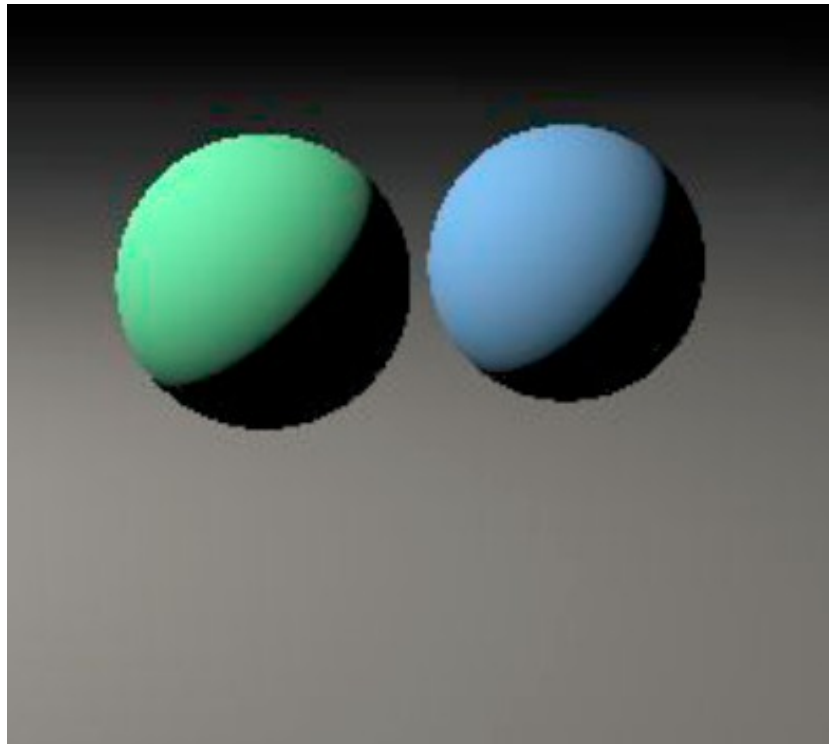
# Shading



$$R = k_a I_a + \sum_i k_d I_i \max(0, \mathbf{l}_i \cdot \mathbf{n}) + k_s I_i \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$$

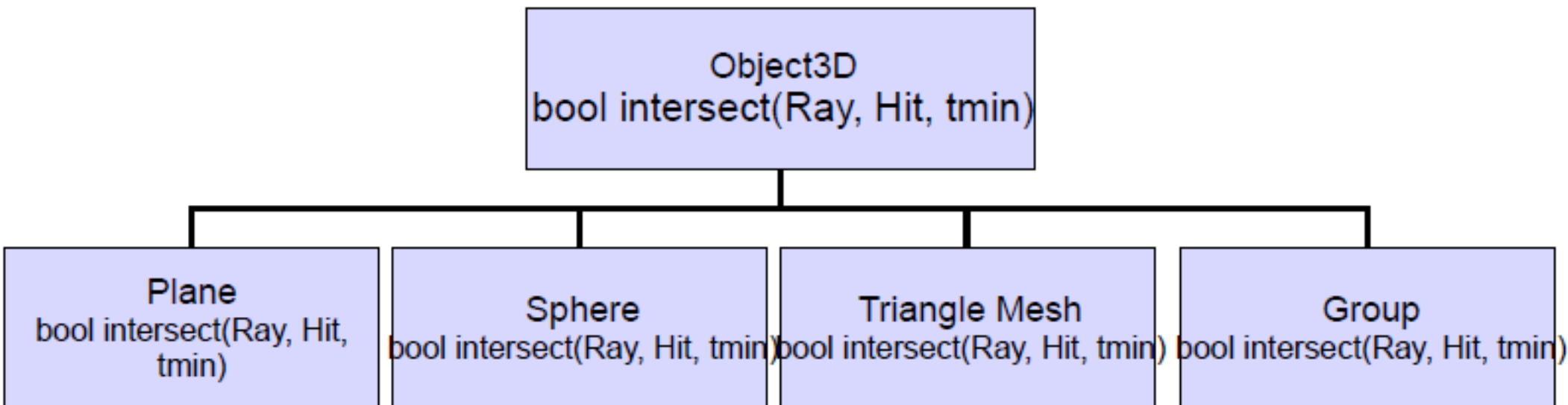
# Ray Tracing Program

```
function ComputeShading(ray, t0, t1)
  Get intersection of ray with scene
  If intersection != null
    Color = ambient
    Get n, h, l
    Color += kd * max(0, <n,l>) + ks * <h, n>p
  Else
    Color = background
```



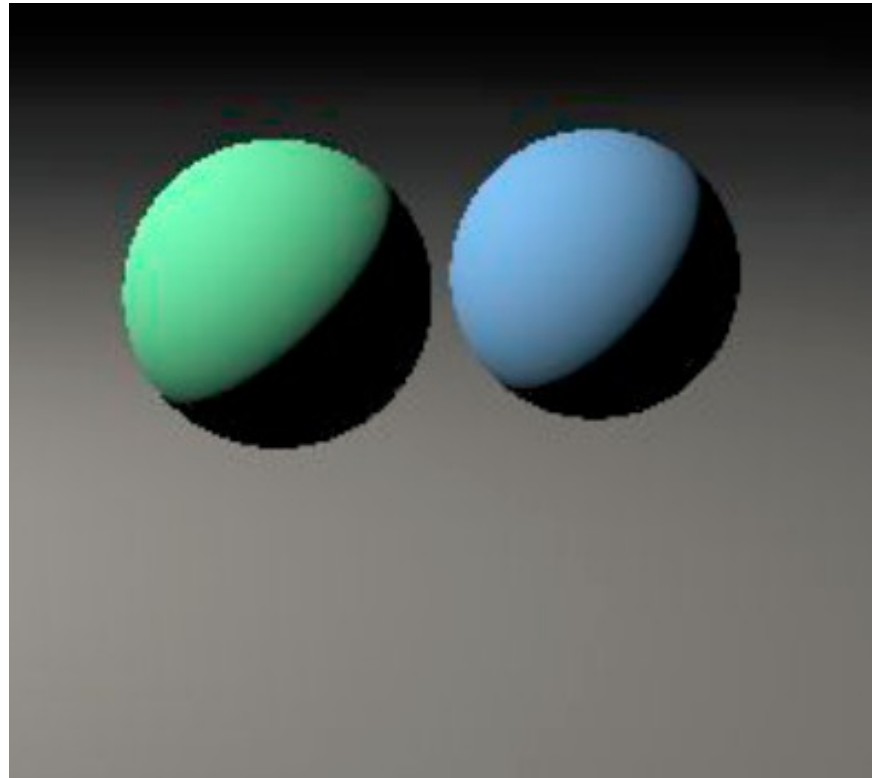
# Ray Tracing Program

- Usually Object Oriented Design
- Objects (and Scene!) derive from base class
- Cameras (orthographic and perspective) derive from base class (for ray generation)
- Virtual methods do the trick!



# Ray Tracing Program

- Similar to rasterization pipeline seen so far
- Will see later how to deal with shadows, reflections, transparency, ...



# Recap

- What is Ray tracing
- Ray Tracing basics
- Ray Generation
- Ray Intersection
- Ray Tracing Program