# CMP205: Computer Graphics


## Lecture 10: Ray Tracing I

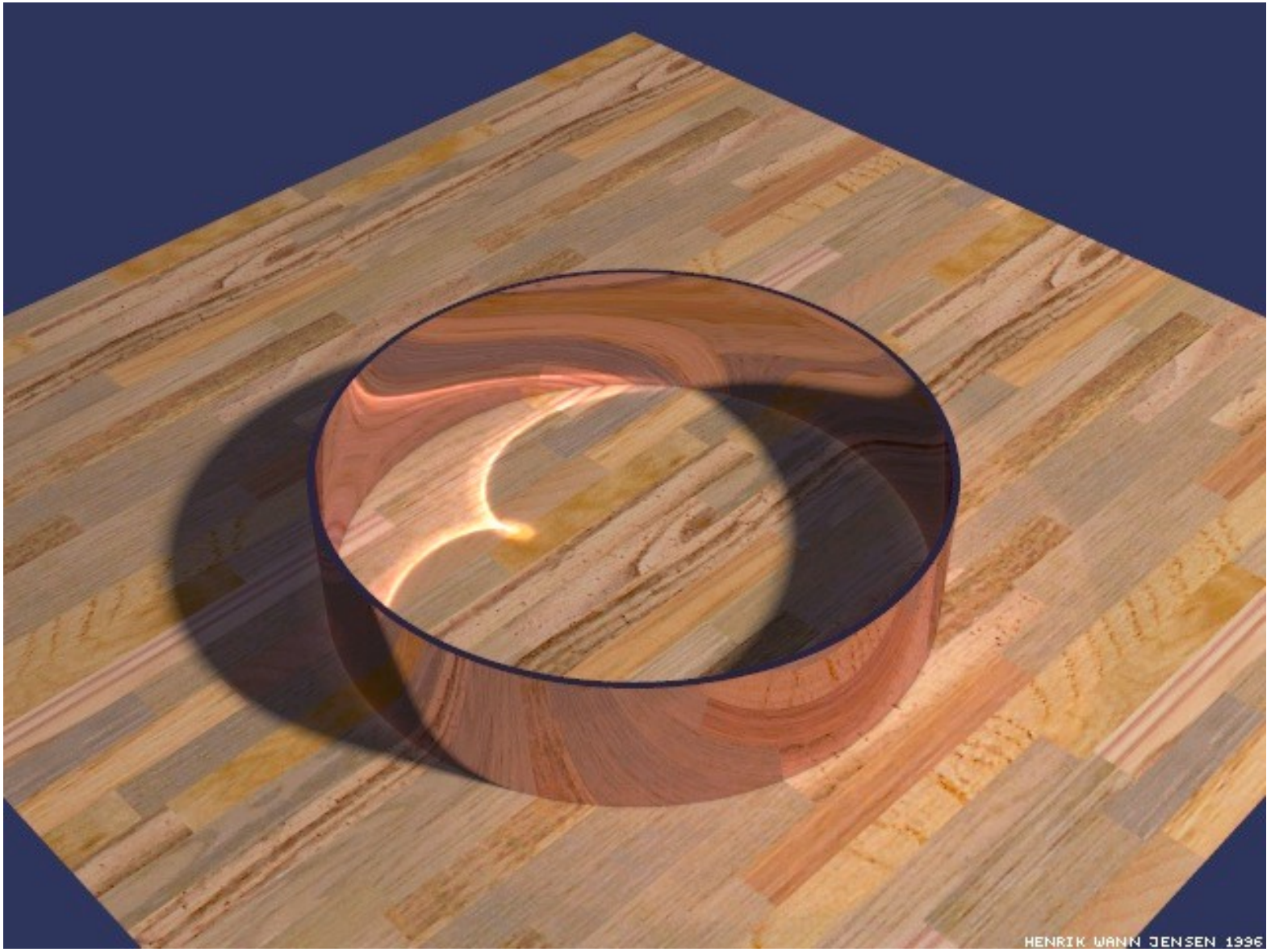
Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Fall 2013

# Agenda

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

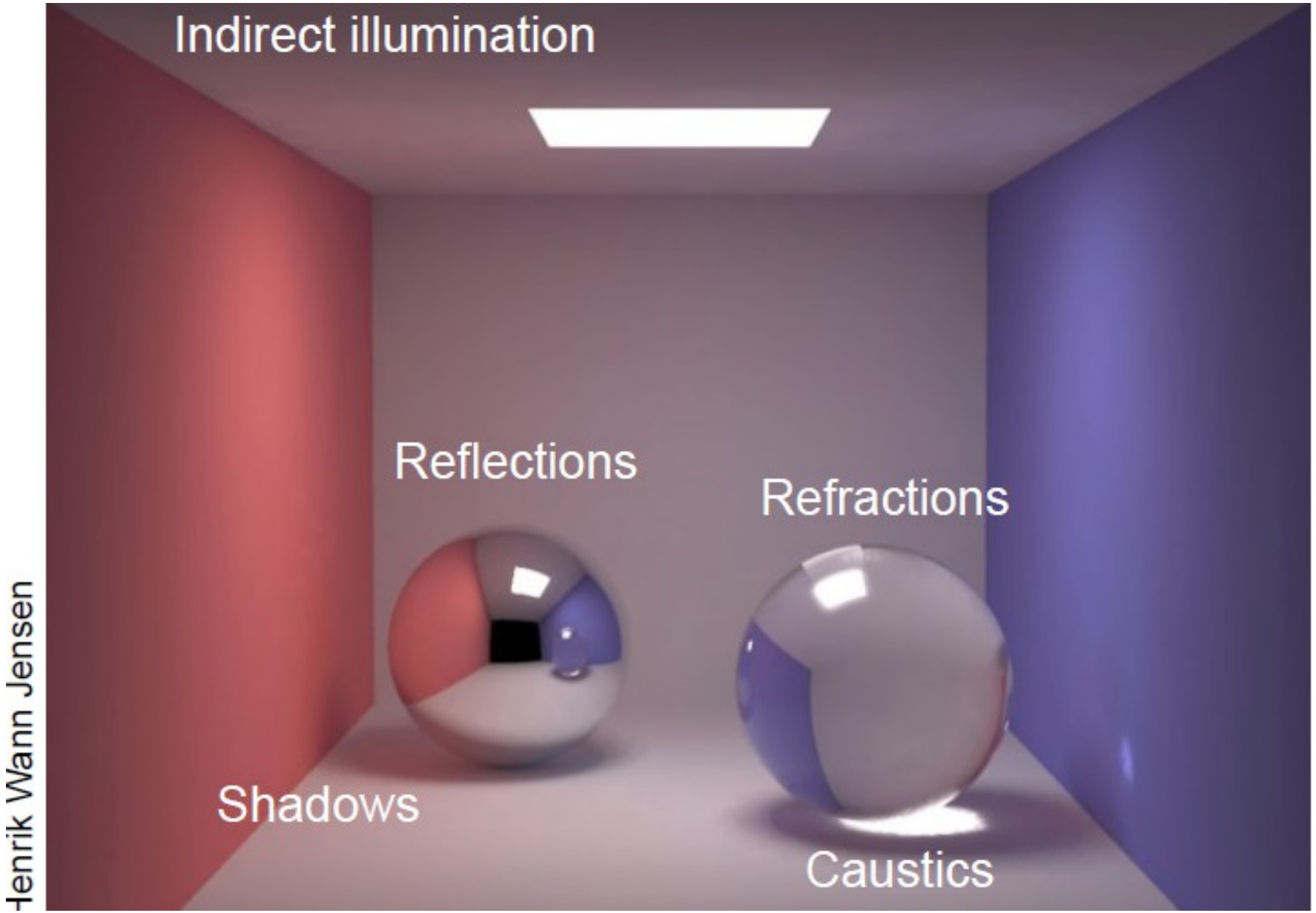
**Acknowledgment**: Some slides adapted from Steve Marschner, Maneesh Agrawala, and Fredo Durand

# What's Ray Tracing?



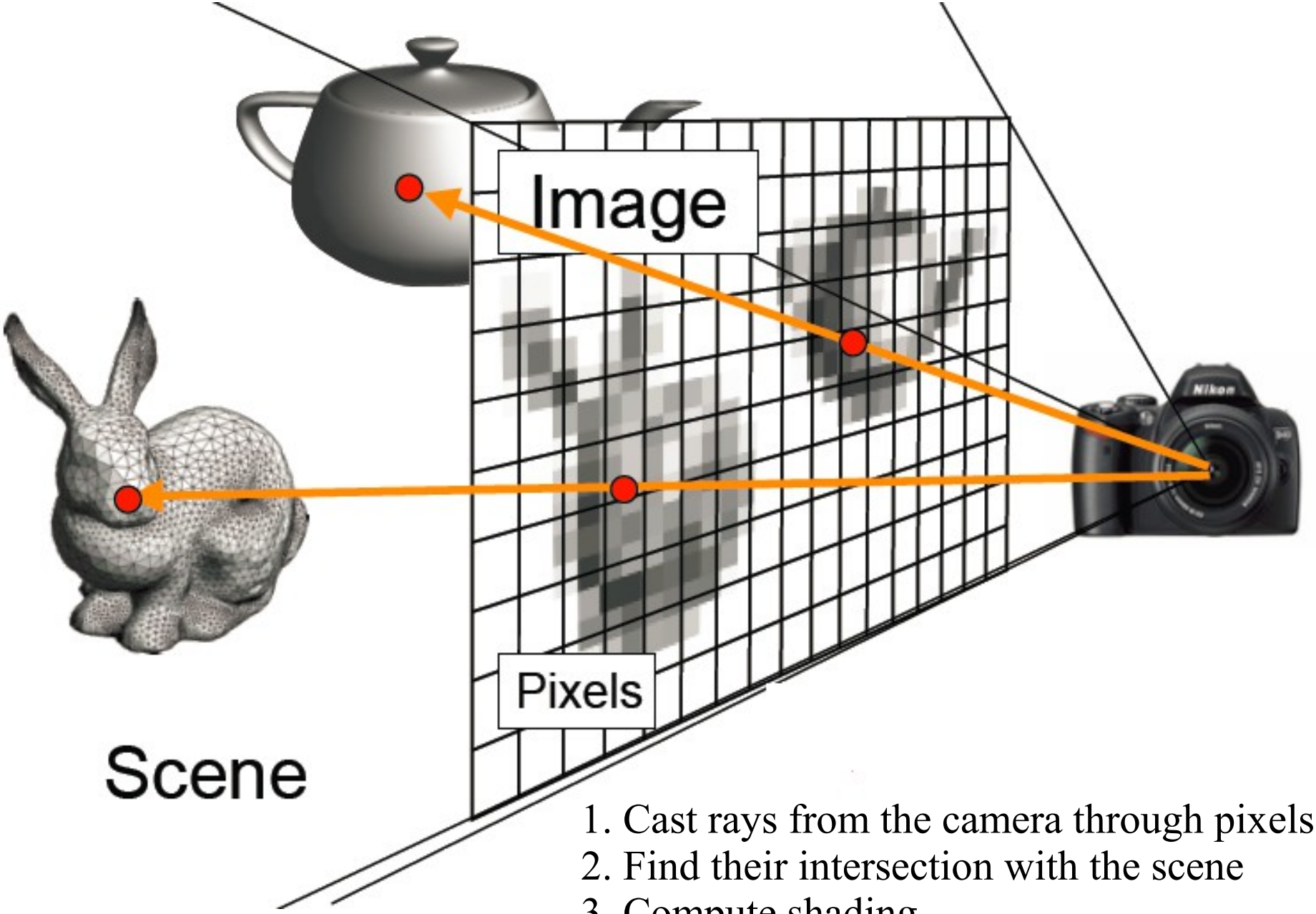

A rendering method that produces *more* realistic images

# What's Ray Tracing?




Naturally handles reflections, shadows, refractions, … etc.

# Ray Tracing




1. Cast rays from the camera through pixels
2. Find their intersection with the scene
3. Compute shading

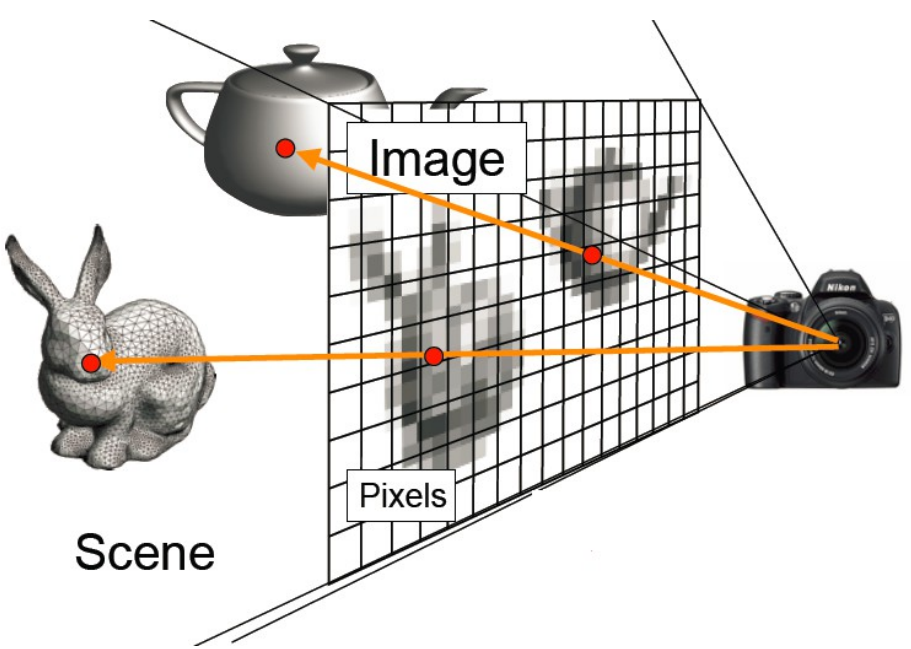
# Ray Tracing Vs Rasterization

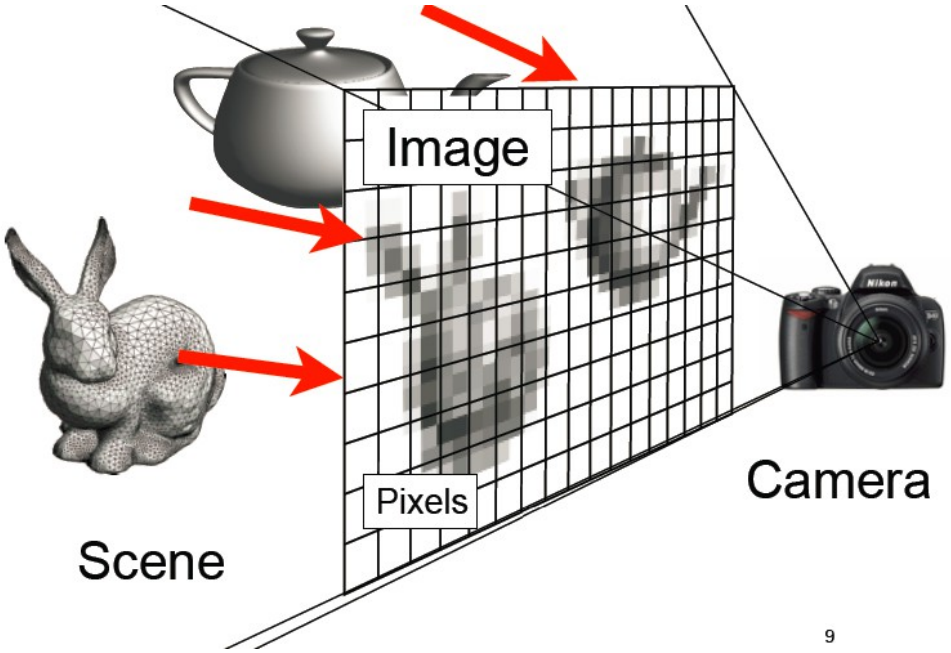
Ray Tracing

```
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
```

Rasterization

```
For each triangle
  For each pixel
    Does triangle cover pixel?
    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**

- Harder to implement in hardware
- Traditionally too slow for interactive applications
- But becoming faster and faster!

**Pros**

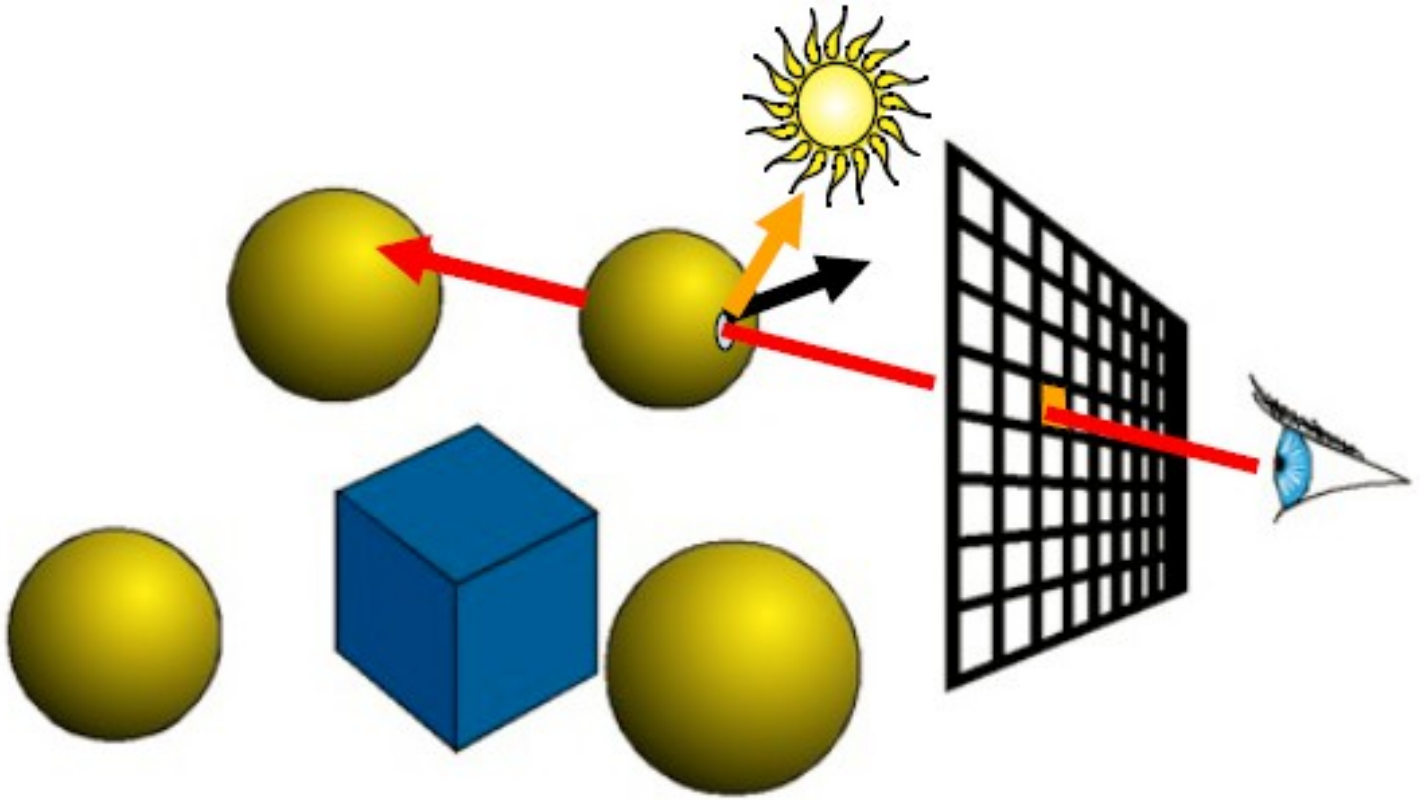
- Much much faster
- Readily available in GPUs
- Parallelizable

**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
    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
```

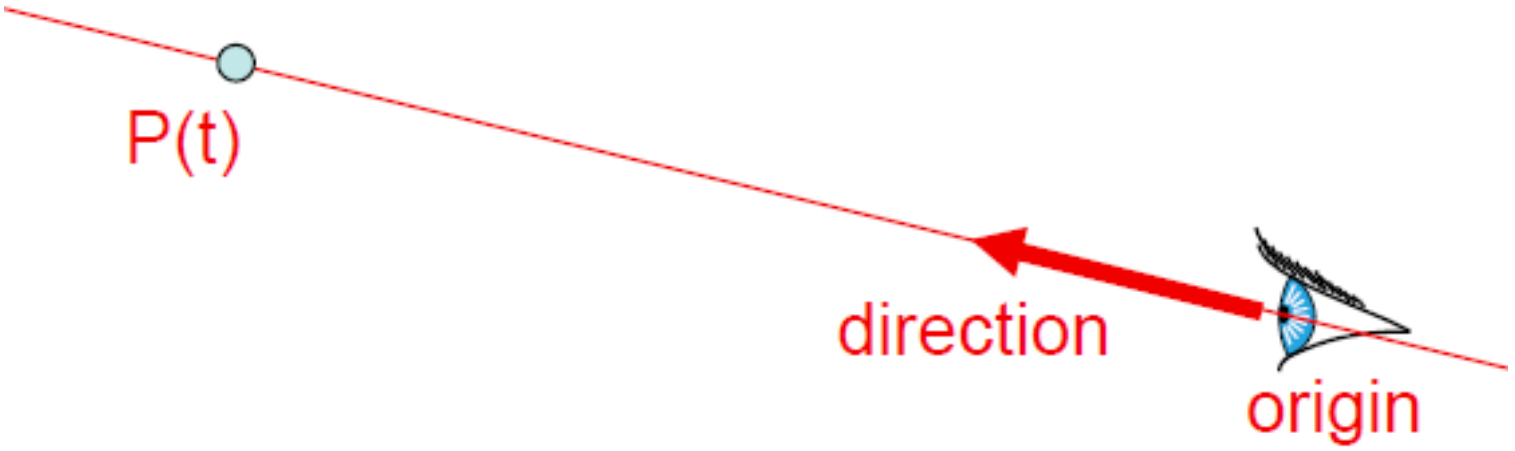
# Ray Generation

A ray is composed of:
- Starting point ***e***
- Direction vector ***d***

Parametric equation: $\boldsymbol{p}(t) = \boldsymbol{e} + t\,\boldsymbol{d}$

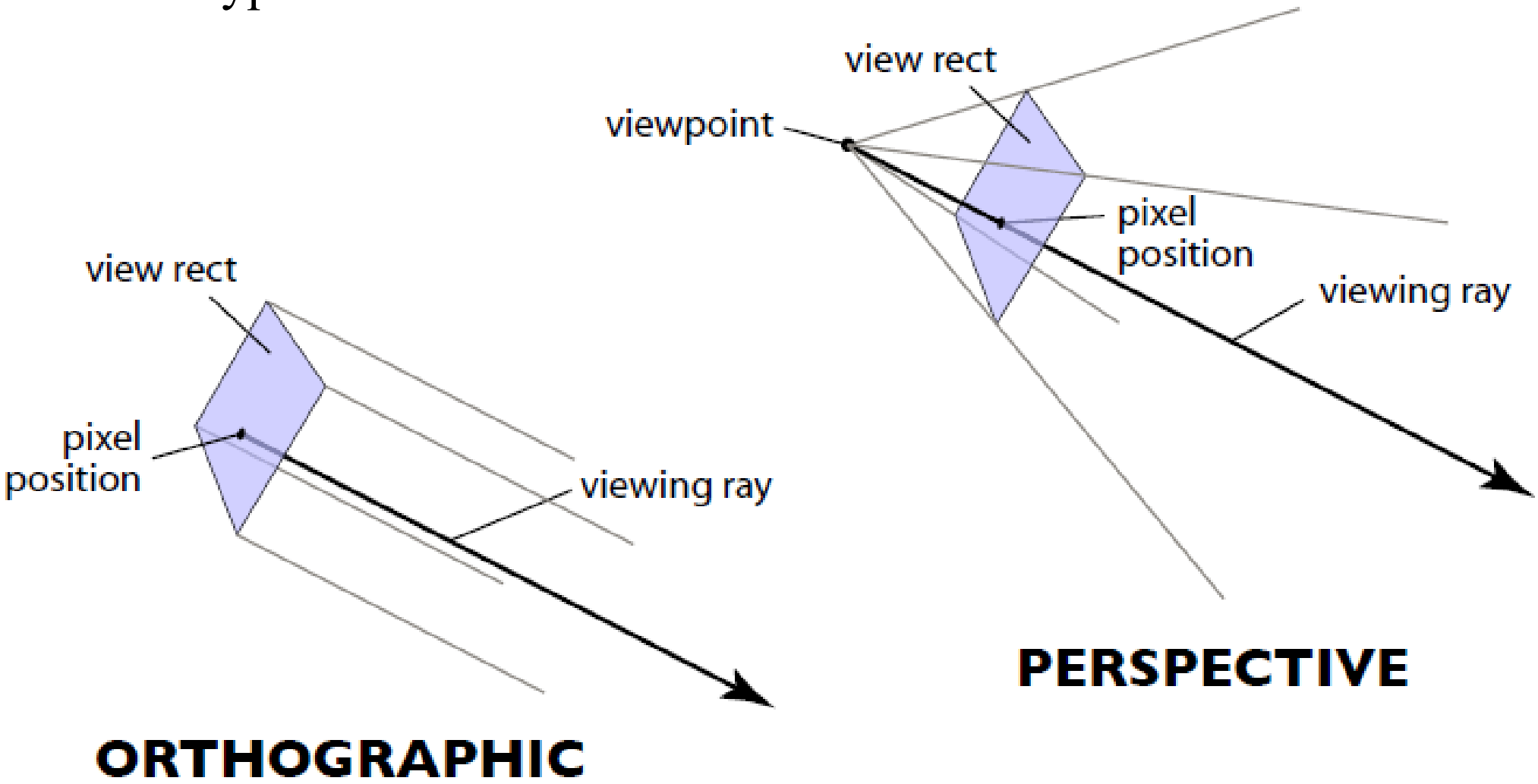
$t = 0 \rightarrow \boldsymbol{p}(t) = \boldsymbol{e}$

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



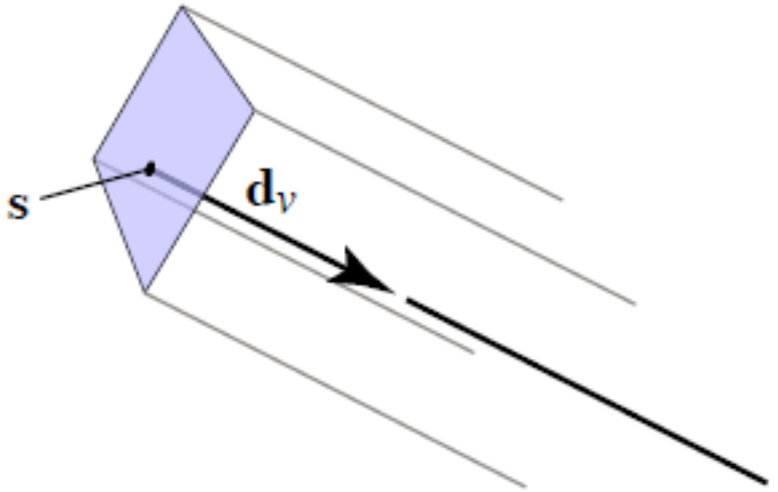
# Ray Generation

Two types of cameras



# Ray Generation: Orthographic

All rays are in the direction of $\mathbf{d}_v$

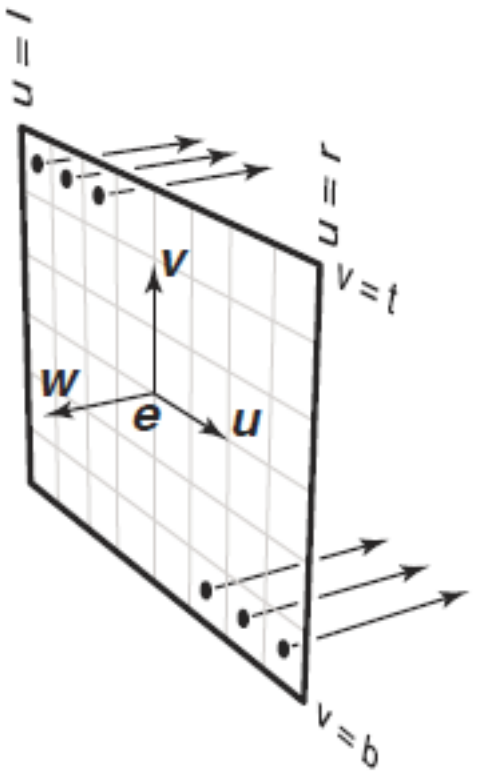



$$p(t) = s + t\, d_v$$

Where is the viewing rectangle in World Coordinates?

# Ray Generation: Orthographic

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

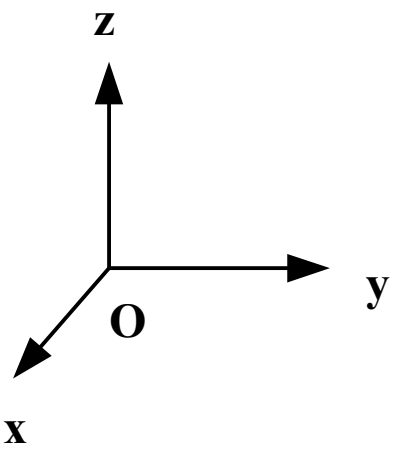



Screen point (in world space): $\boldsymbol{s} = \boldsymbol{e} + u_s \boldsymbol{u} + v_s \boldsymbol{v}$

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

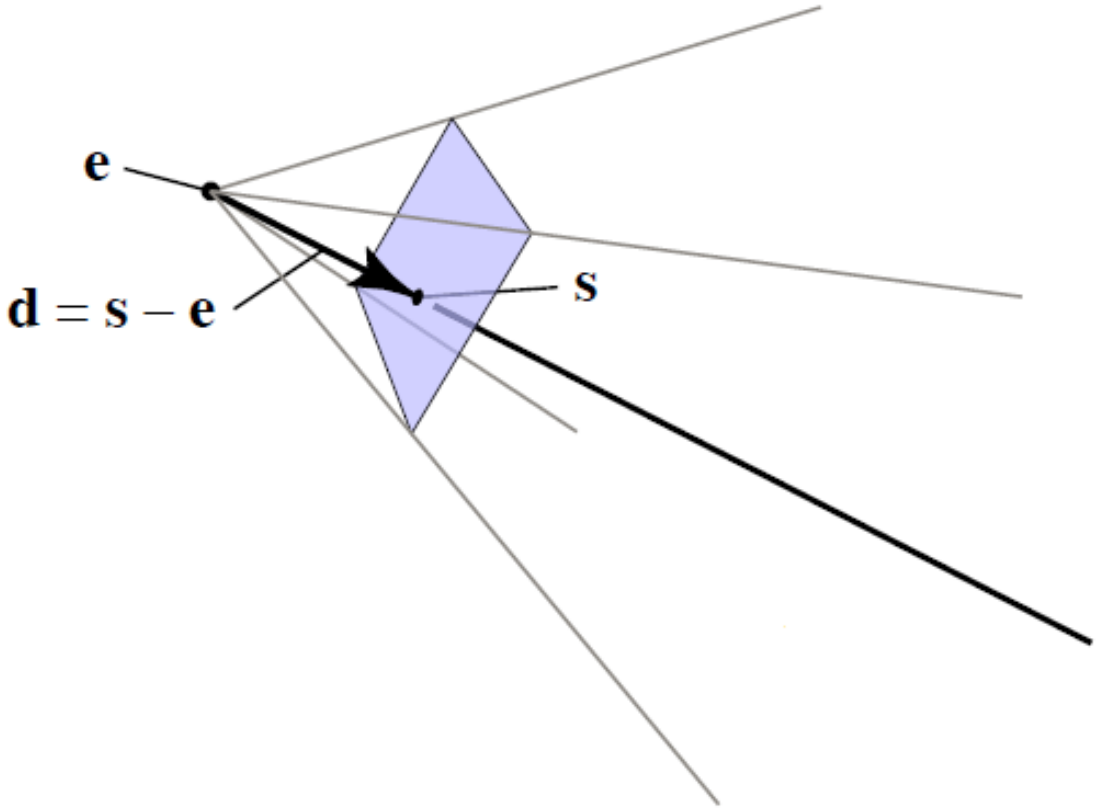
Starting Point: $\boldsymbol{s}$

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



# Ray Generation: Perspective

All rays pass through the camera center **e**




$$\boldsymbol{p}(t) = \boldsymbol{e} + t\,\boldsymbol{d}$$

Where is the viewing rectangle in World Coordinates?

# Ray Generation: Perspective

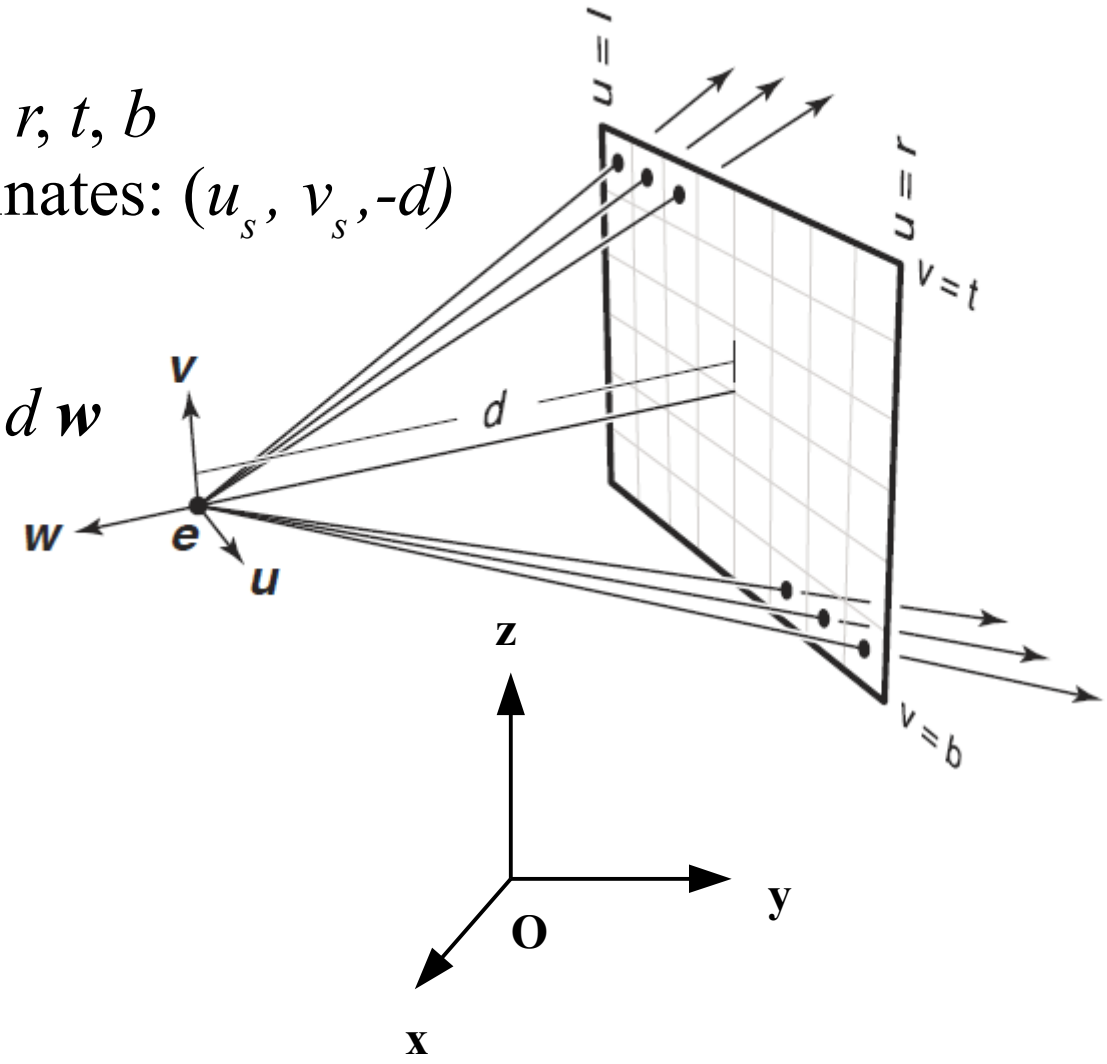
- Camera basis: **u**, **v**, **w**
- Camera position: **e**
- View rectangle specified by *l*, *r*, *t*, *b*
- Screen point in camera coordinates: $(u_s, v_s, -d)$

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

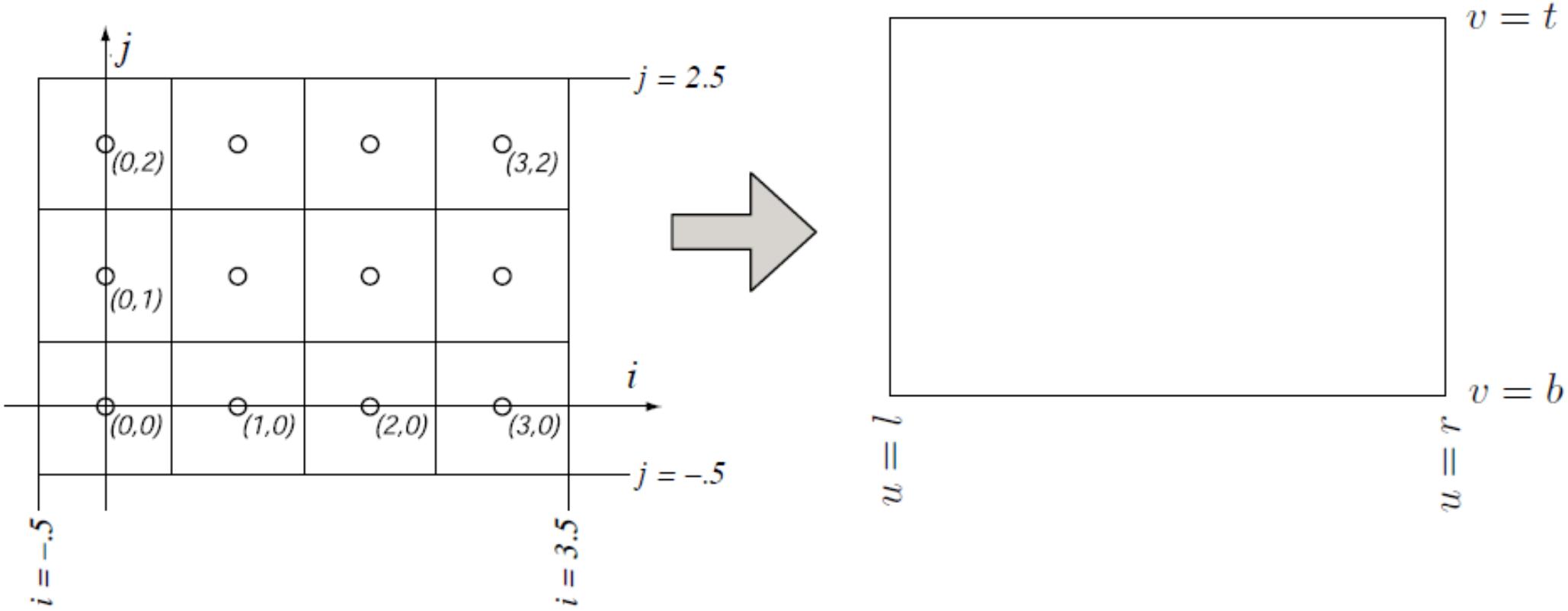
Direction: $\boldsymbol{d} = \boldsymbol{s} - \boldsymbol{e}$

Starting Point: $\boldsymbol{e}$

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



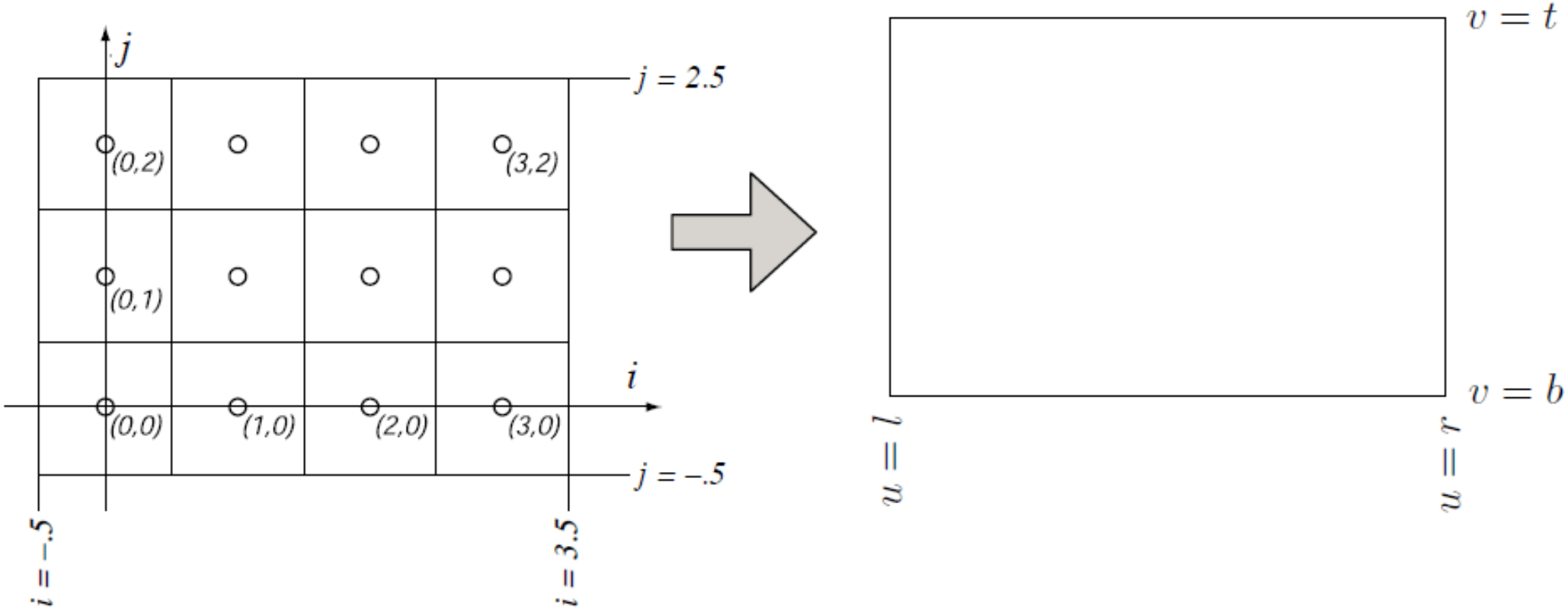
# Ray Generation: Pixel-to-Image




$[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$ $\qquad$ $[l\,, r] \times [b\,, t]$

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

# Ray Generation: Pixel-to-Image




$[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$ $\qquad$ $[l, r] \times [b, t]$

Windowing Transformation: Translate, Scale, Translate

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

# 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: $\boldsymbol{p}(t)=\boldsymbol{e}+t\,\boldsymbol{d}$

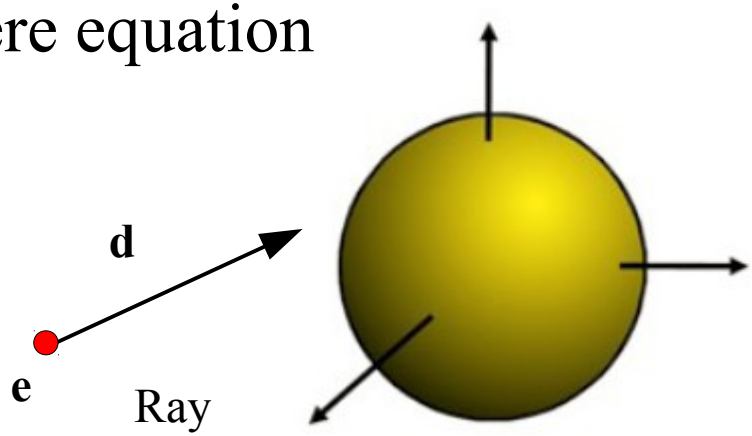
Sphere implicit equation: $\|\boldsymbol{p}-\boldsymbol{c}\|^2-r^2=0$ for center $\boldsymbol{c}$ & radius $r$

Intersect → Substitute ray equation into sphere equation and solve for $t$

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

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

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




Quadratic Equation in $t$ !

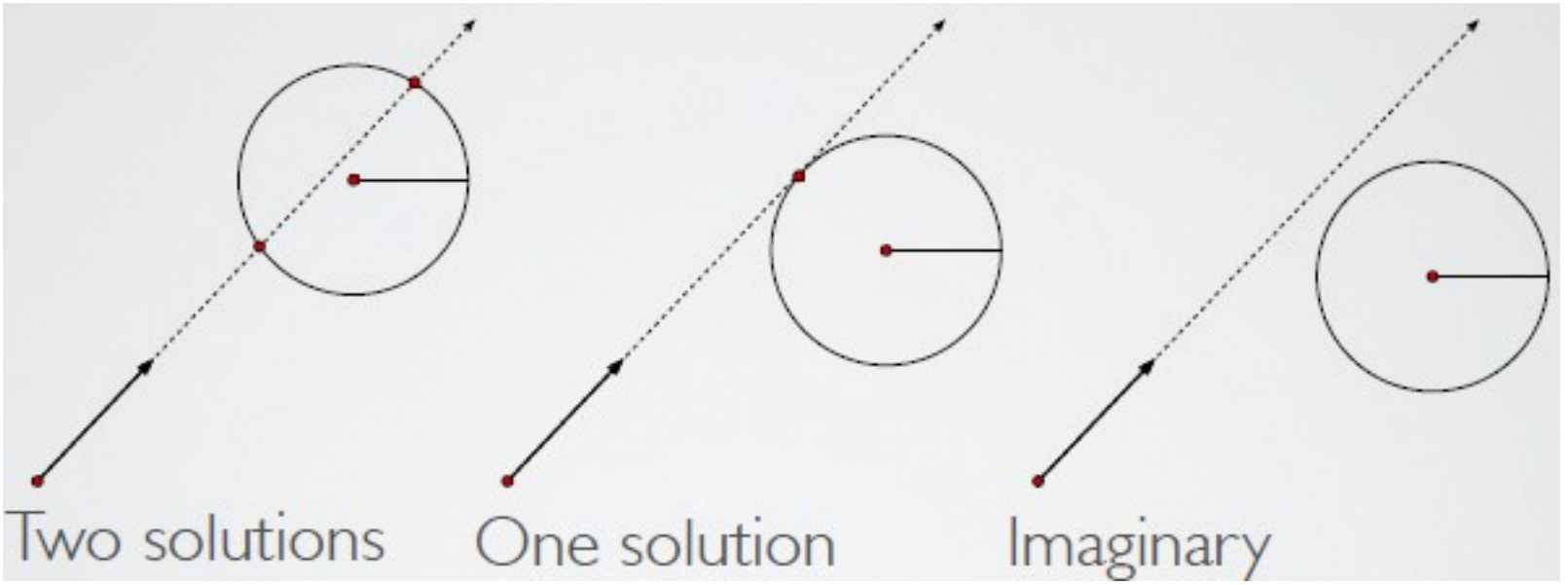
# Ray Intersection: Sphere

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

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

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

$D>0$ $D=0$ $D<0$

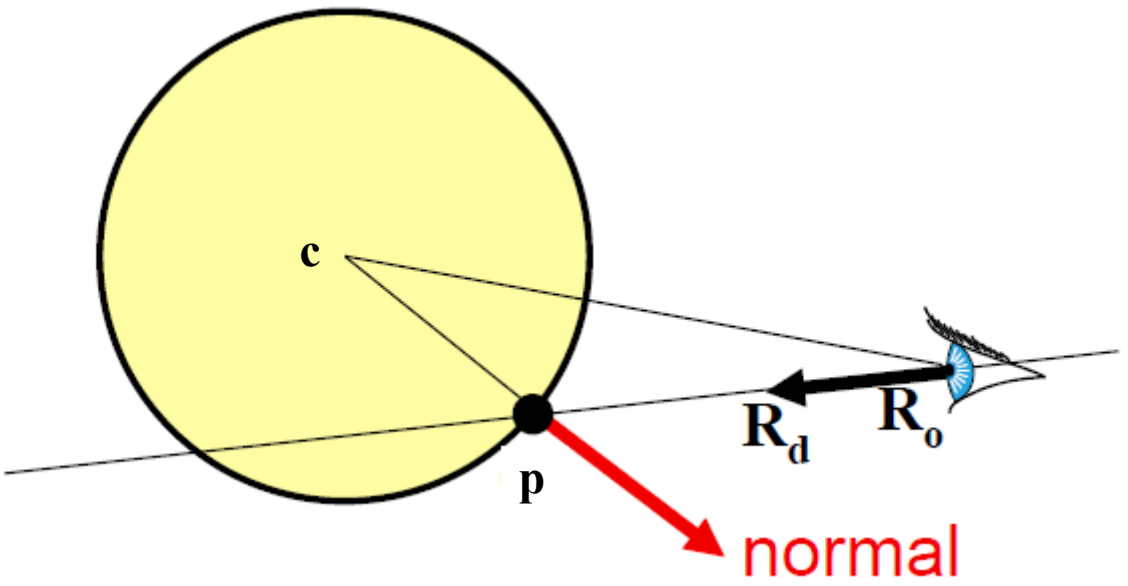



Which $t$ to choose? Smallest $t > t_{min}$

# Ray Intersection: Sphere

What about surface normal?

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



# Ray Intersection: Plane

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

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

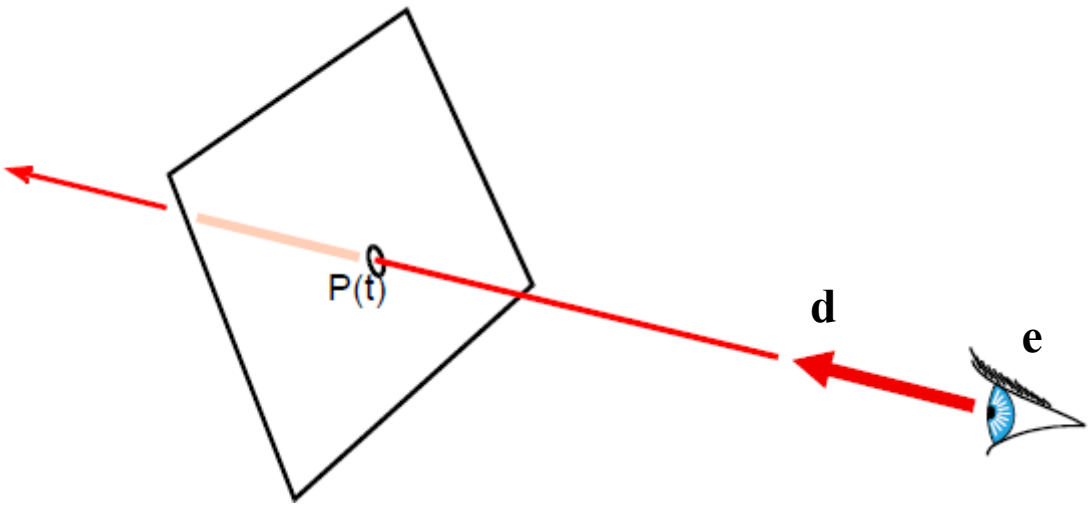
Intersect → Substitute ray equation into plane equation and solve for $t$

$$\boldsymbol{n}^T\left(\boldsymbol{e}+t\,\boldsymbol{d}\right)+D=0$$

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

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

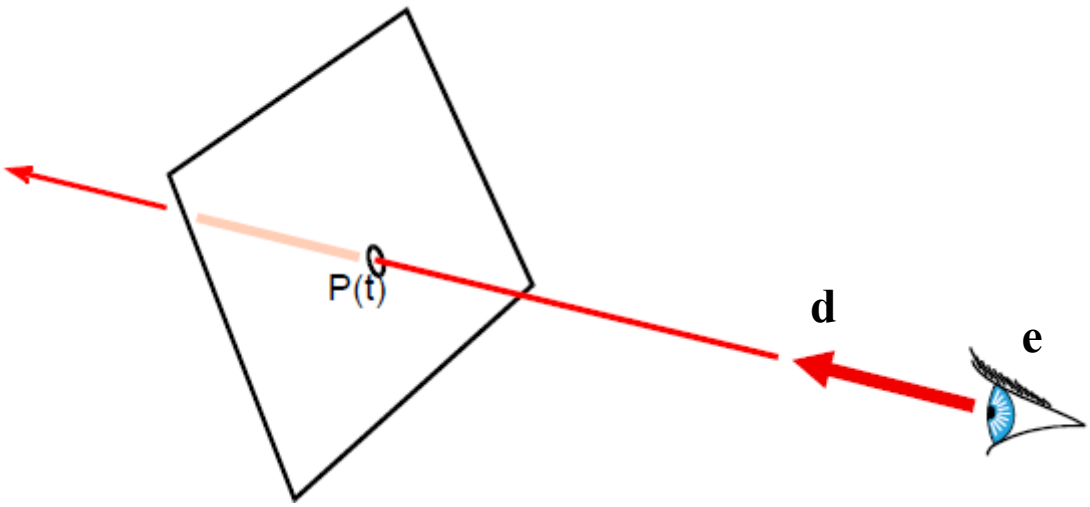
Ray parallel to Plane !



# Ray Intersection: Plane

What about surface normal?

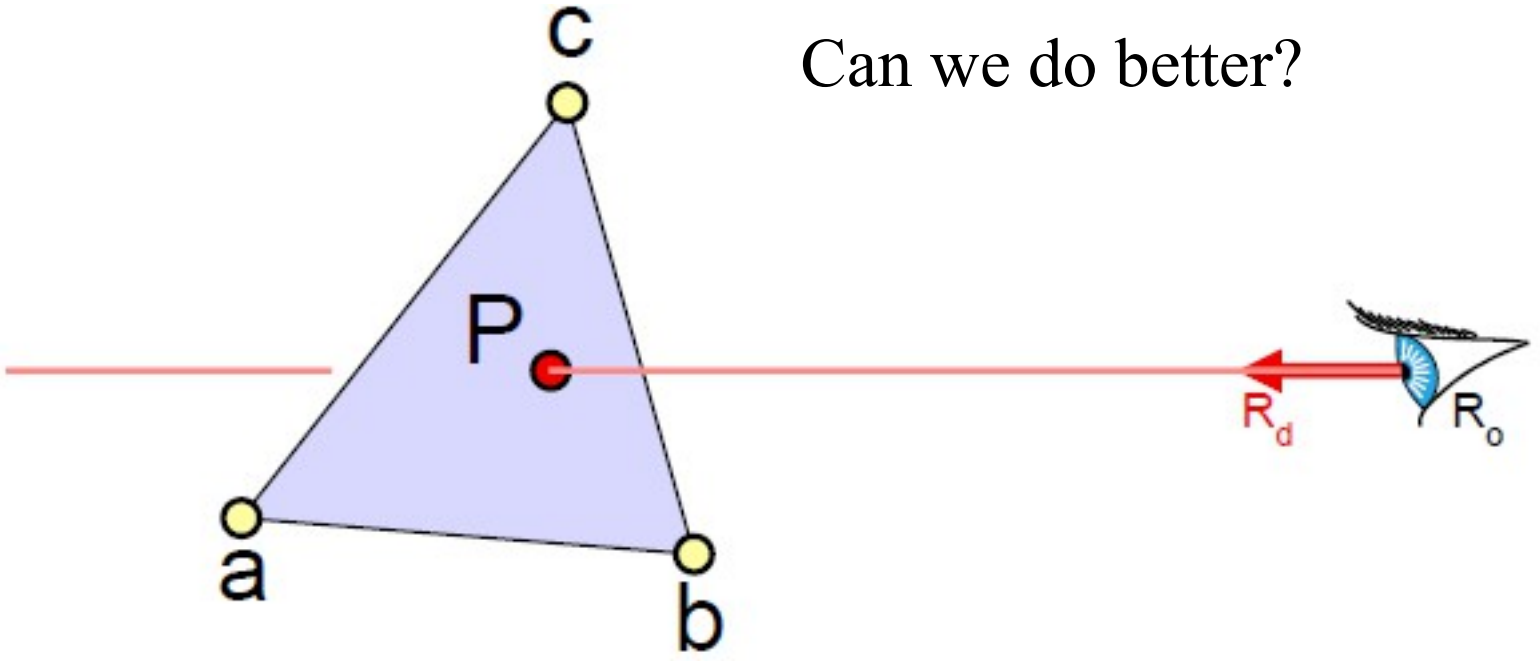
Already given !



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




Can we do better?

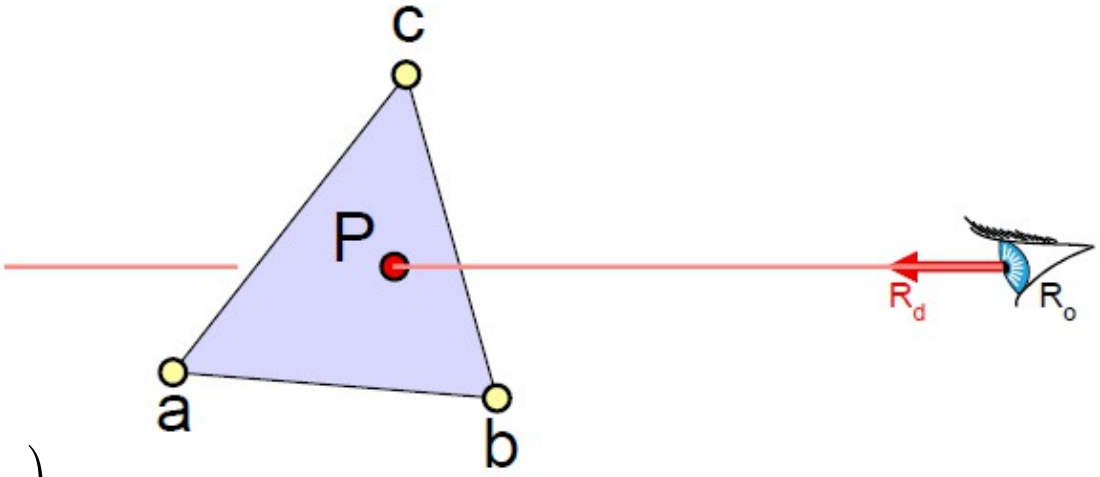
# Ray Intersection: Triangle

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

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

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

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



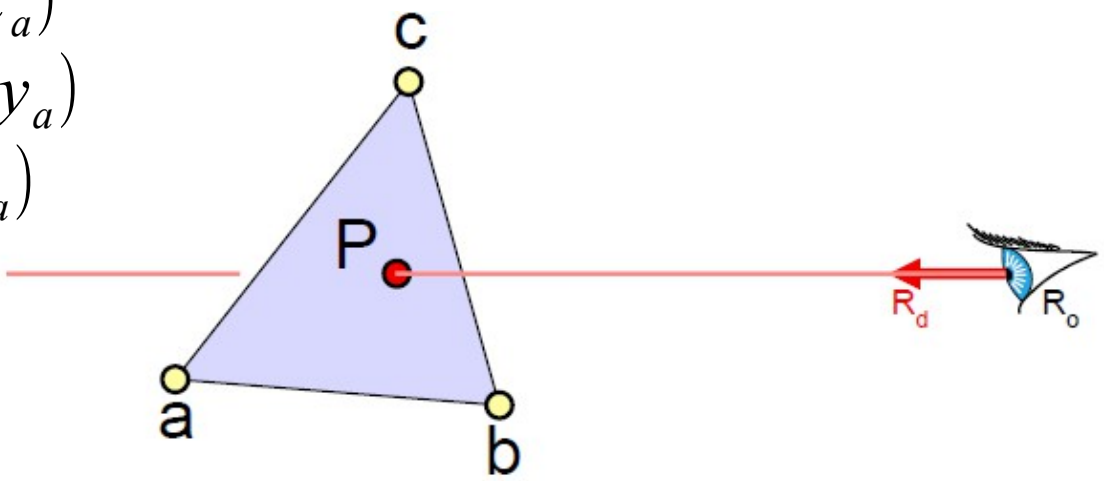

Three equations in three unknowns

$$x_e+t\,x_d=x_a+\beta(x_b-x_a)+\gamma(x_c-x_a)$$
$$y_e+t\,y_d=y_a+\beta(y_b-y_a)+\gamma(y_c-y_a)$$
$$z_e+t\,z_d=z_a+\beta(z_b-z_a)+\gamma(z_c-z_a)$$

# Ray Intersection: Triangle

Three equations in three unknowns

$$x_e + t\,x_d = x_a + \beta(x_b - x_a) + \gamma(x_c - x_a)$$
$$y_e + t\,y_d = y_a + \beta(y_b - y_a) + \gamma(y_c - y_a)$$
$$z_e + t\,z_d = z_a + \beta(z_b - z_a) + \gamma(z_c - z_a)$$



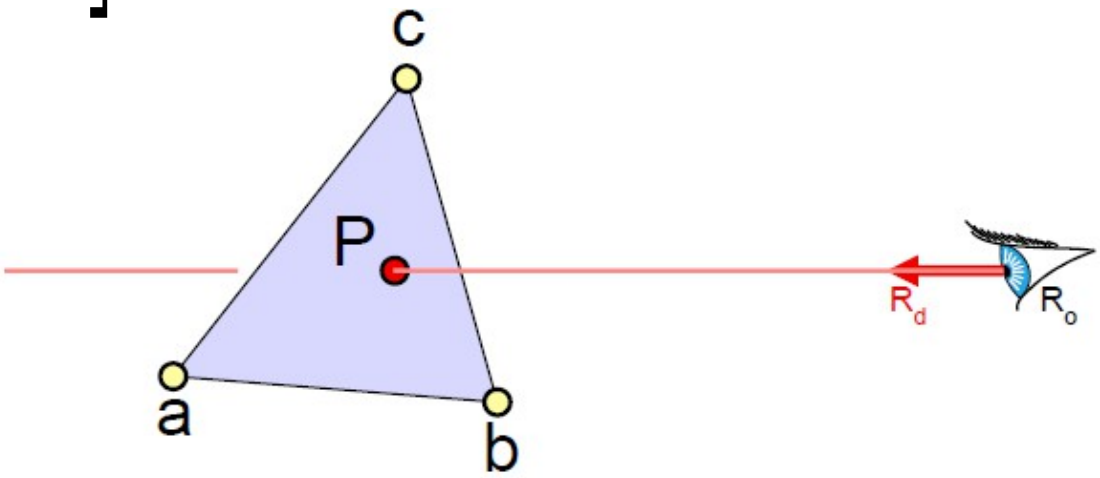

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

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 $\gamma$




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

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

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

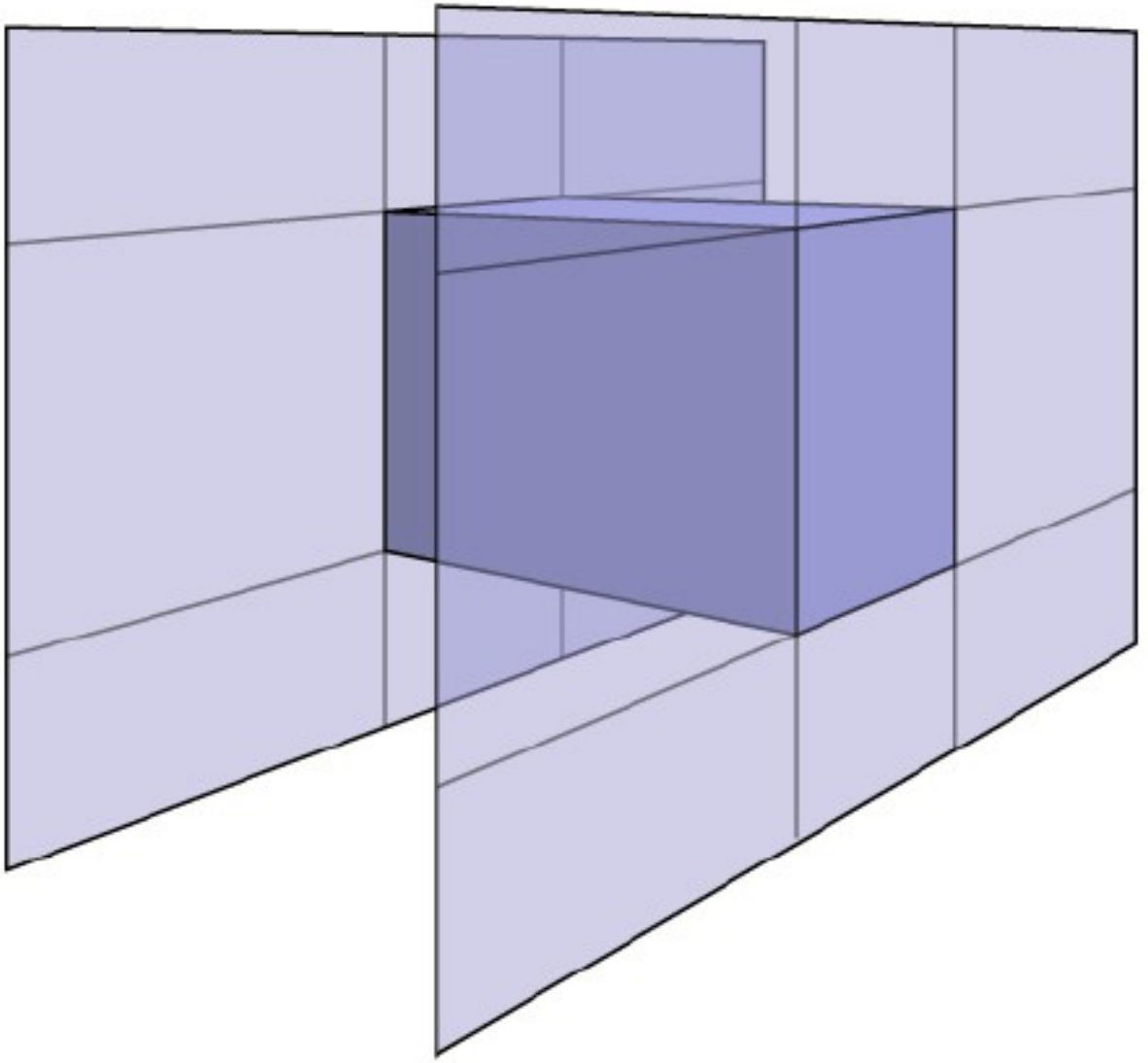
# Ray Intersection: Triangle

- 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: $\boldsymbol{p}(t) = \boldsymbol{e} + t\,\boldsymbol{d}$

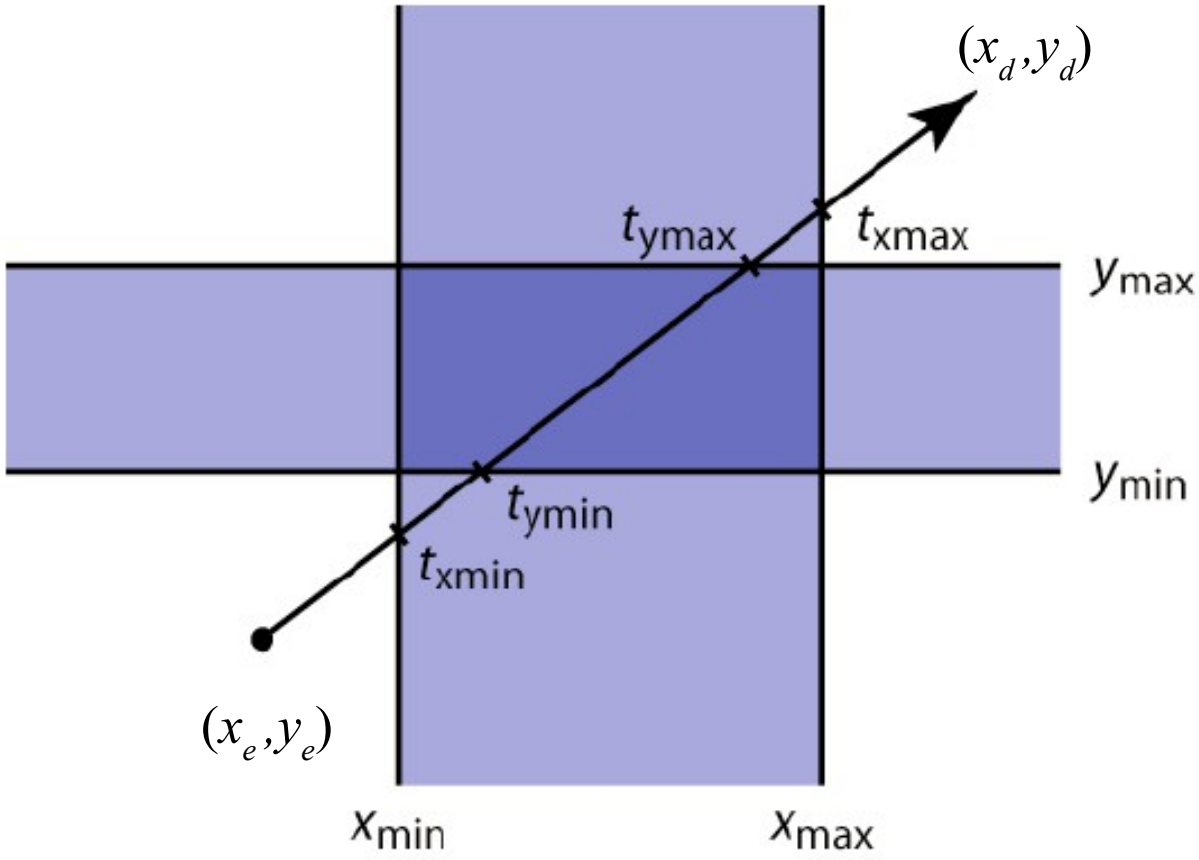
2D Box: $x_p = x_{min}, x_p = x_{max}, y_p = y_{min}, y_p = y_{max}$

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

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

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

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



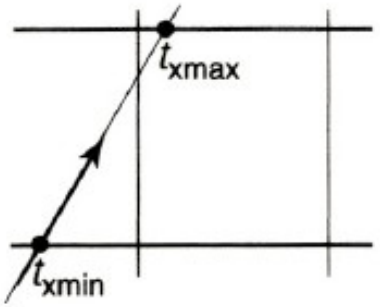
# Ray Intersection: Box

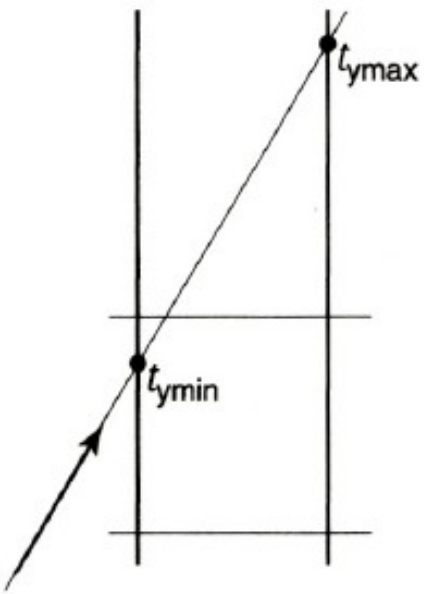
Each intersection gives an interval

We want the last entry point and first exit point !

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

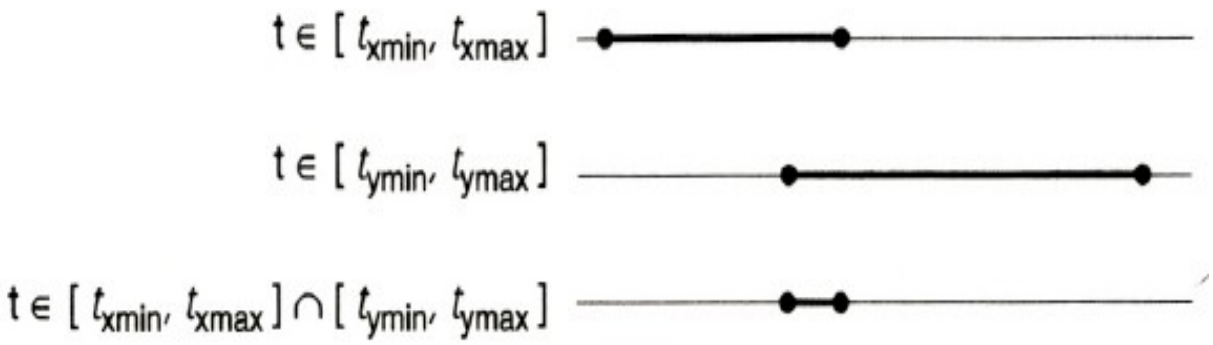
$$t_{max} = min(t_{xmax}, t_{ymax})$$




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

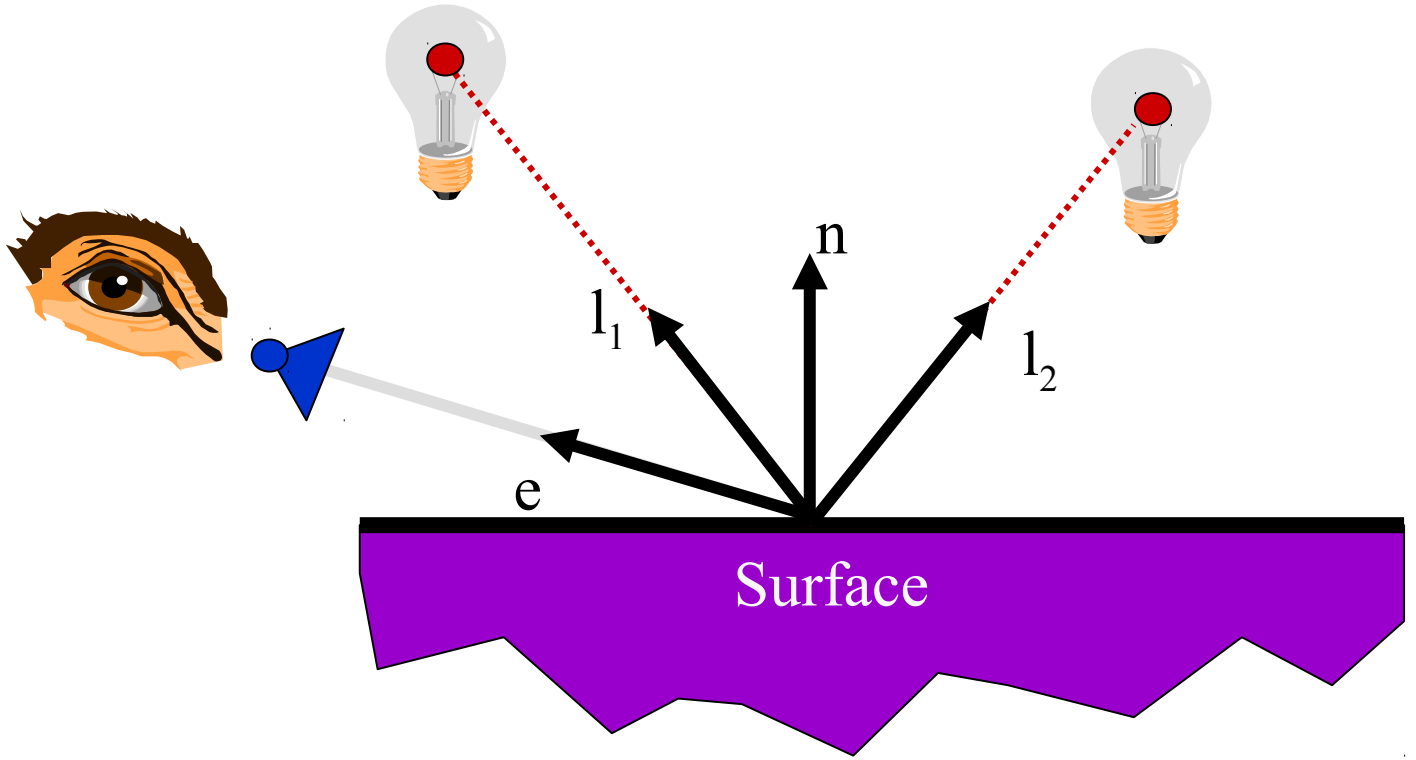
Intersection point?
$\rightarrow p(t_{min})$



# 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
```

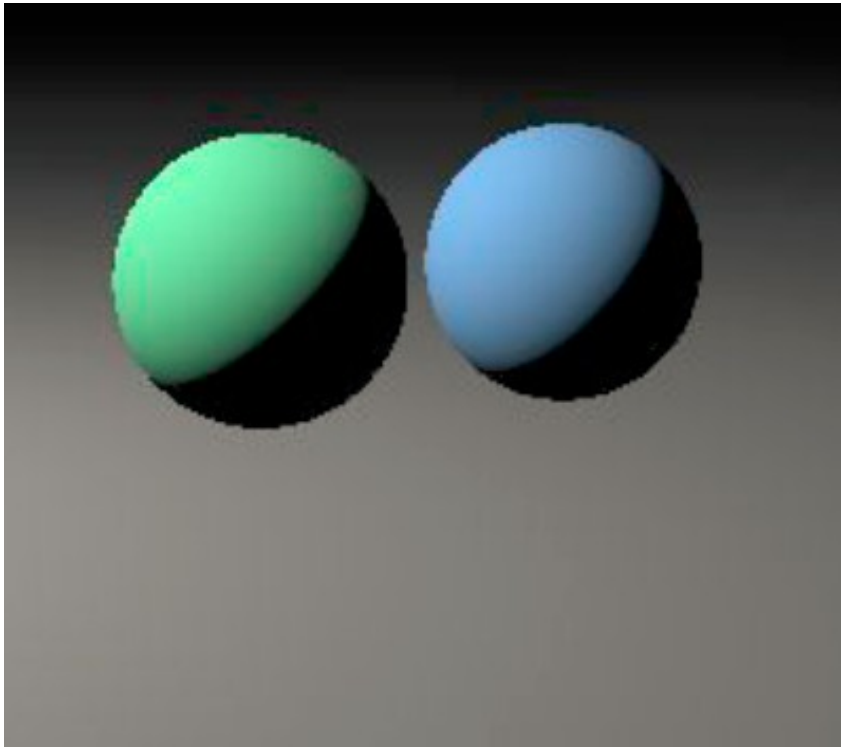
# Shading




$$R = k_a I_a + \sum_i k_d I_i \max\left(0, \boldsymbol{l_i} \cdot \boldsymbol{n}\right) + k_s I_i \max\left(0, \boldsymbol{e} \cdot \boldsymbol{r_i}\right)^p$$

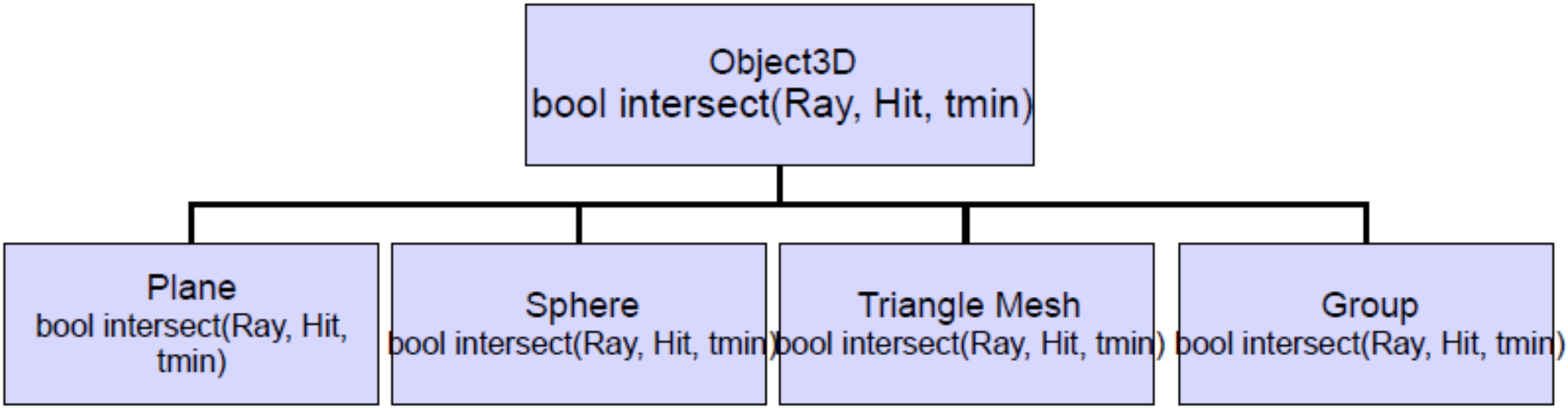
# Ray Tracing Program

```
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
```



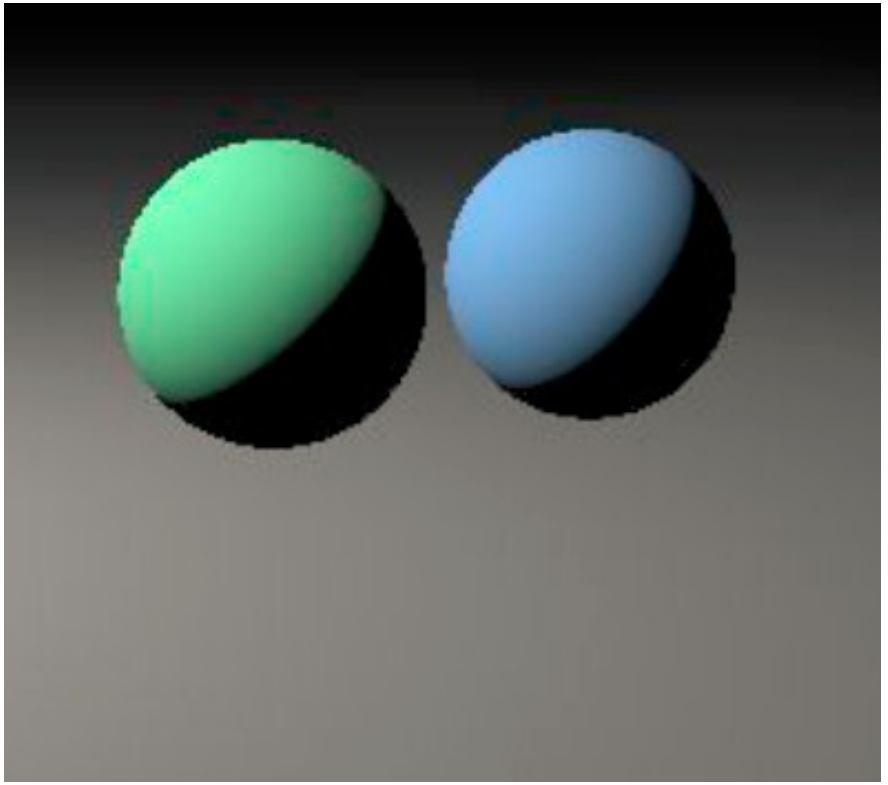
# Ray Tracing Program

- 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