Ray Tracing
Part 1: Introduction

Computer Graphics, CSCD18

Fall 2007

Instructor: Leonid Sigal

Motivation

- Shading models we have considered so far
  - Gouraud, Phong, Flat Shading are all “local models”
  - Only account for direct lighting (i.e. light that originates at the point light source hits the surface and then is projected towards the eye/camera)
- Global models can also account for
  - Shadows
  - Secondary illumination (e.g. color bleeding from object to object)
  - Arbitrary reflections (e.g. mirrors)
  - Refractions (e.g. transparencies, sub-surface scattering)
- All global effects are captured via ambient term in the Phong lighting equation (**very** crude)

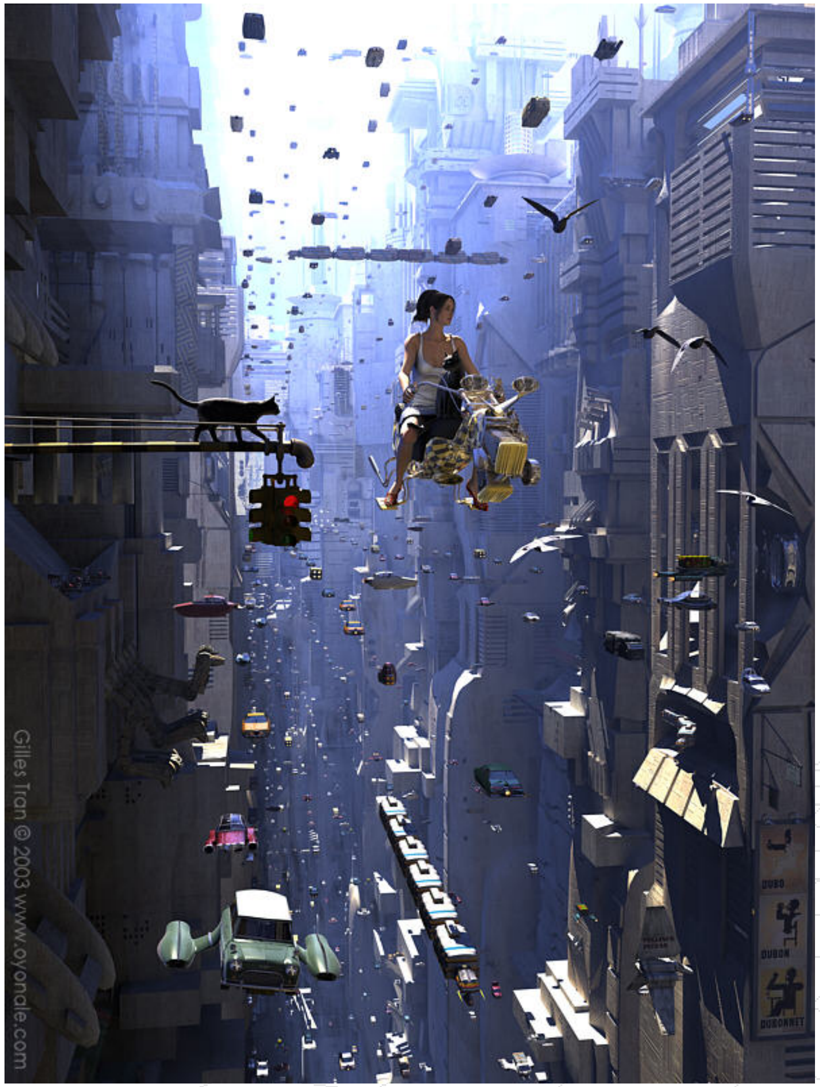
Ray Tracing

- Ray tracing was developed to help model properties of global illumination (that cannot be modeled using the models we considered thus far).

- Benefits of ray tracing
  - This is a state of the art algorithm
    - Entertainment (Animations, Movies, Commercials)
    - Games (PlayStation 3)
    - Simulation (e.g. architectural)
    - Art
  - It's customizable, so one can make rendering engines that are arbitrarily simple/complex
  - It can be implemented in real time on some of the latest hardware (with moderate assumptions on resolution and scene geometry)

Ray Tracing

See: http://www.povray.org/community/hof/



Gilles Tran (c) 2000 www.oyonale.com
CoolCows

"Main street (blue)" © Gilles Tran (2003) *"The Cool Cows" © Gilles Tran (2000)*

Ray Tracing

See: http://www.povray.org/community/hof/

Gilles Tran © 2002 www.oyonale.com

"The Dark Side of the Trees" © Gilles Tran (2002)

"Capriccio" © Gena Obukhov, Ib Rasmussen, Jim Charter, Txemi Jendrix, Peter Hertel, Matti Karnaattu, Bob Hughes, Christoph Hormann (2003)

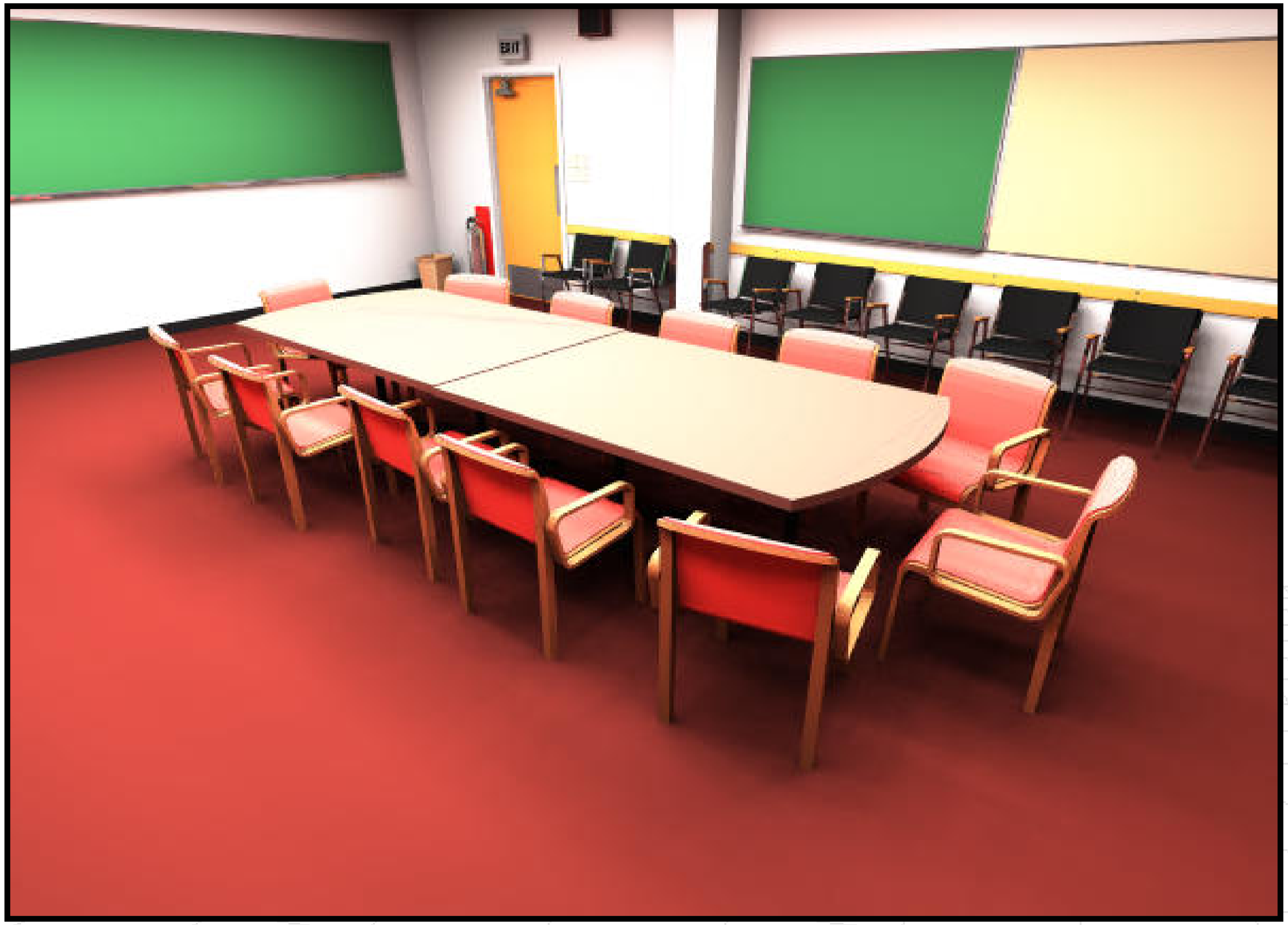
Ray Tracing

- Ray tracing competitions have been held on-line for past 10 years
  - Take a look at: ***http://www.irtc.org/stills/index.html#s2006***

MARCO LUCINI 1997

Real-time Rendering

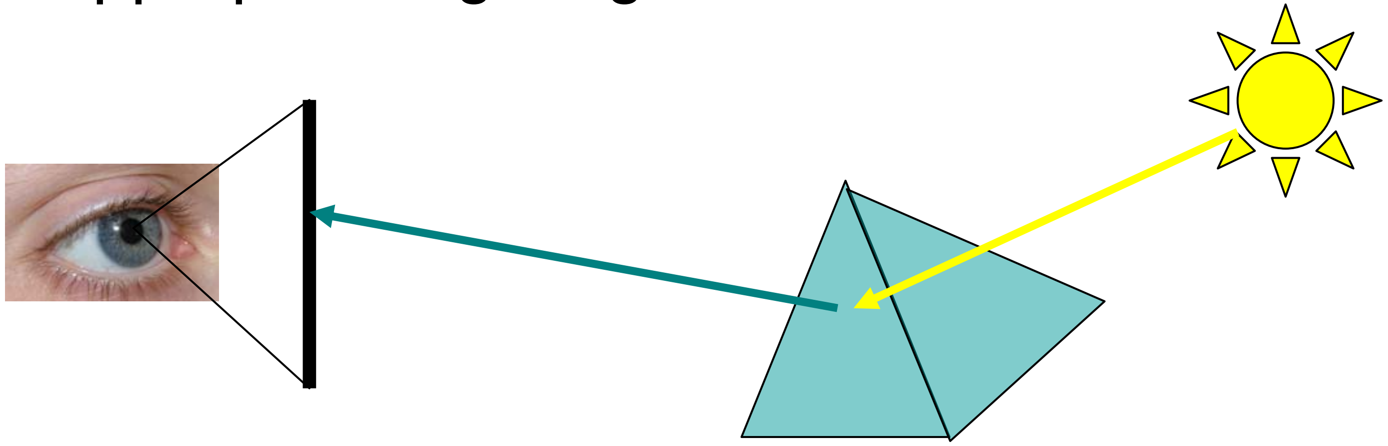
From Siggraph 2005 course by Philipp Slusallek, Peter Shirley, Bill Mark, Gordon Stoll, Ingo Wald



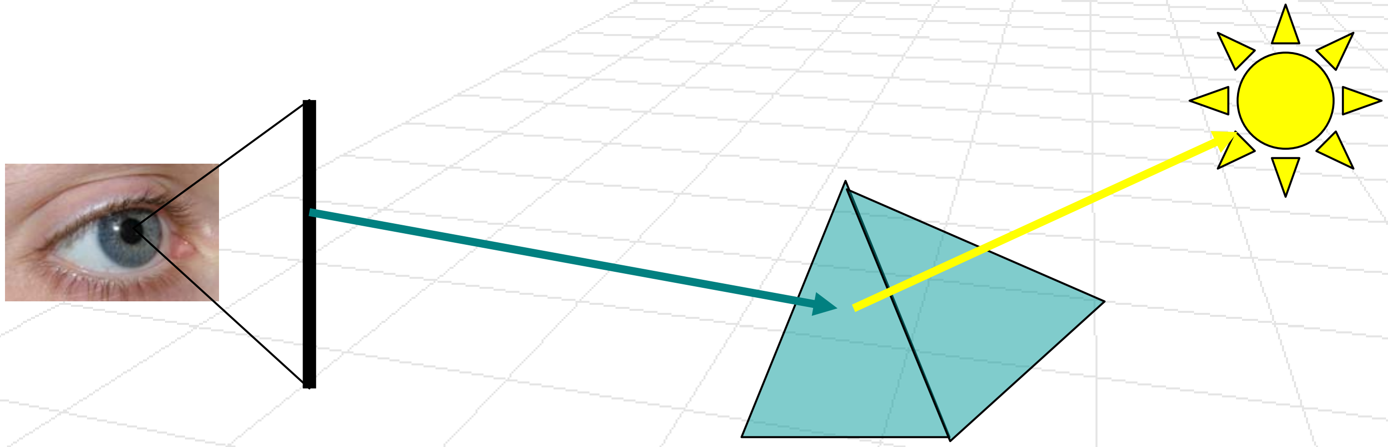
Conference room (380 000 triangles, 104 lights) with full global illumination in realtime

Rendering in Computer Graphics

- **Resterization:** Project the geometry into the image plane under appropriate lighting



- **Ray tracing:** Project image samples (pixels) backwards



Ray Tracing

- **Idea:** Trace paths of light backwards from the camera (image pixel) to surface (or multiple surfaces) to light(s).

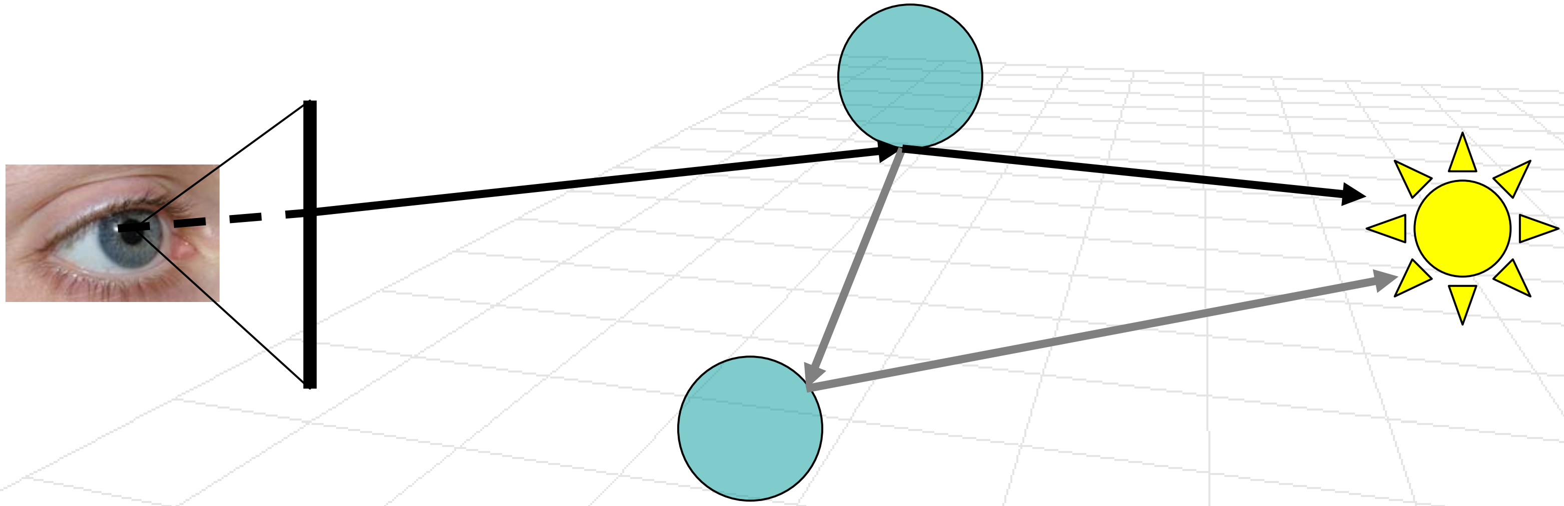
- Recursive algorithm
- For each pixel
  - Cast a ray (hence ray tracing) from eye of camera through pixel and find 1st surface hit by that ray
  - Determine surface radiance at the surface with combination of local and global models

    (To estimate global component, cast rays from surface point to possible incident directions to determine how much light comes from each direction)

Ray Tracing

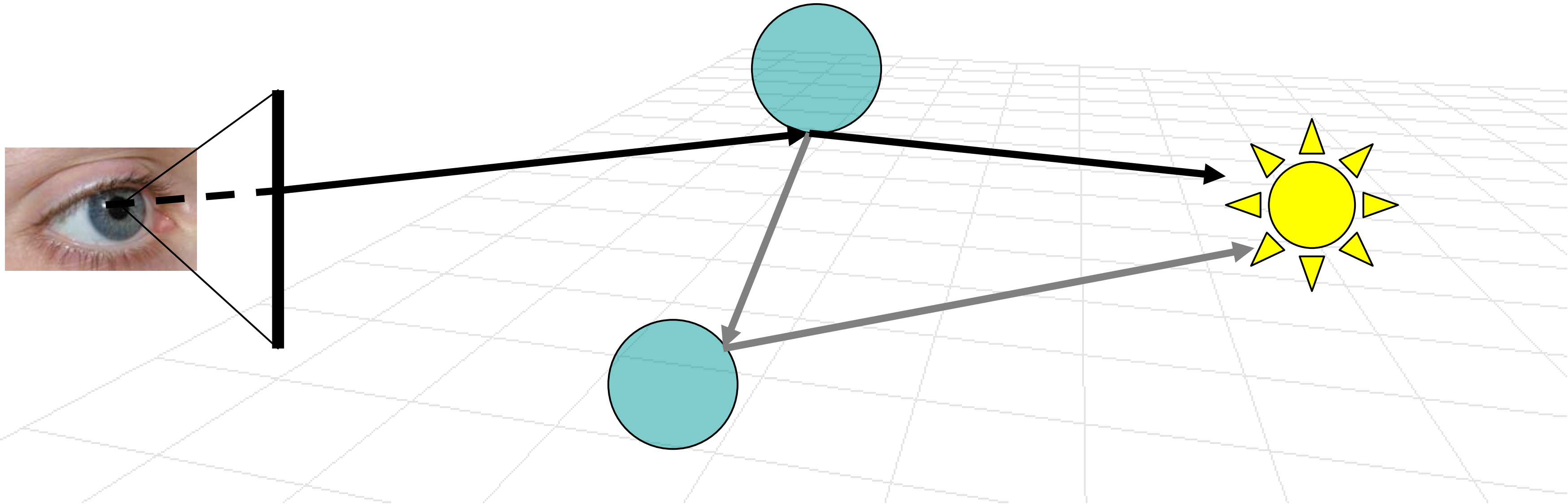
- For each pixel
  - Cast a ray (hence ray tracing) from eye of camera through pixel and find 1st surface hit by that ray
  - Determine surface radiance at the surface with combination of local and global models

    (To estimate global component, cast rays from surface point to possible incident directions to determine how much light comes from each direction)



Computational Issues to Consider

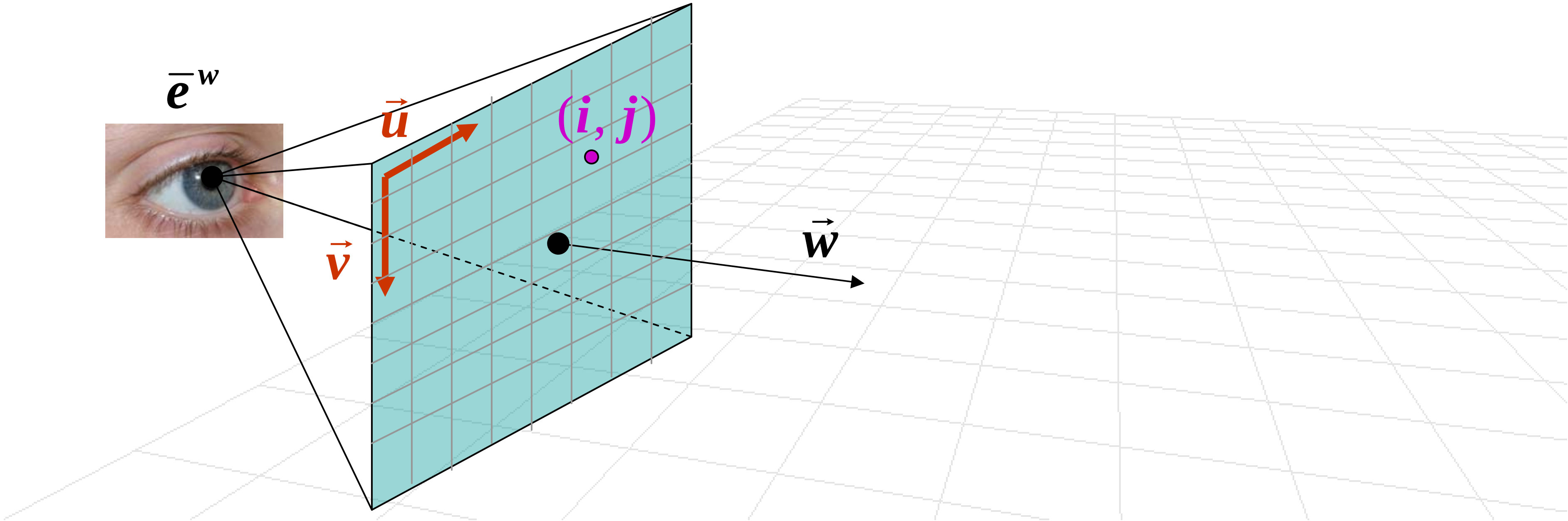
- Form rays (a.k.a. ray casting)
- Find intersection of rays with objects
- Find closest object intersection (there could be multiple object intersections for any given ray)
- Find normal at the closest intersection point (a.k.a hit point)
- Evaluate reflectance model at the hit point



Ray Casting

- We want ray through pixel (***i***, ***j***)
- Camera model

$\bar{e}^w$	**- Origin of camera coordinate frame**
$\vec{u}, \vec{v}, \vec{w}$	**- Directions of X,Y,Z in camera coordinate frame**
$(\bar{p} - \bar{r}) \cdot \vec{w} = 0$	**- Image plane, where** $\bar{r} = \bar{e}^w + f\vec{w}$



Ray Casting

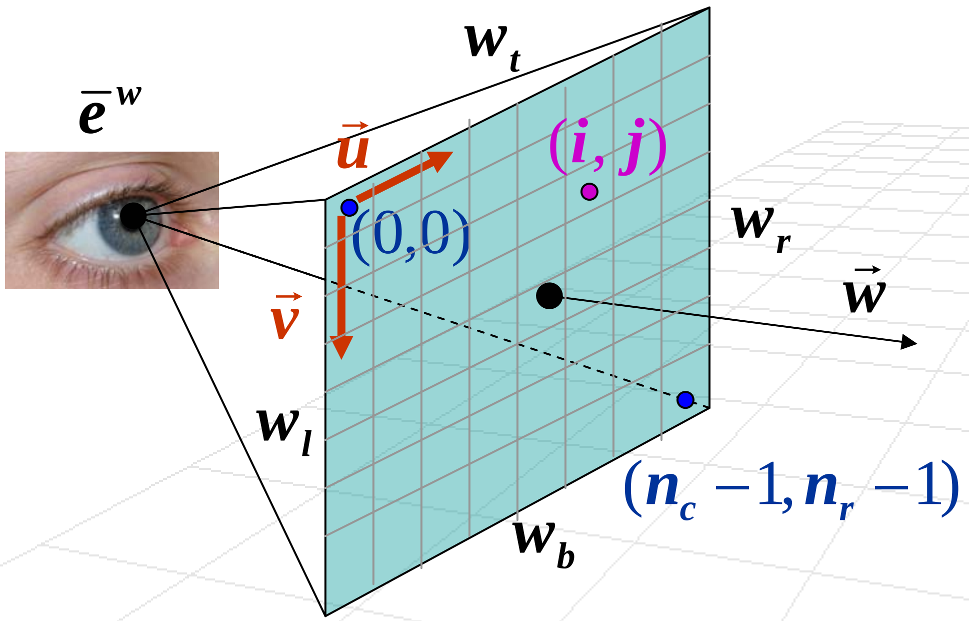
- We want ray through pixel (***i***, ***j***)
- Viewport

w_l **- Left edge of the view volume**

w_r **- Right edge of the view volume**

w_t **- Top edge of the view volume**

w_b **- Bottom edge of the view volume**



Ray Casting

- We want ray through pixel (***i***, ***j***)
- Viewport

w_l **- Left edge of the view volume**

w_r **- Right edge of the view volume**

w_t **- Top edge of the view volume**

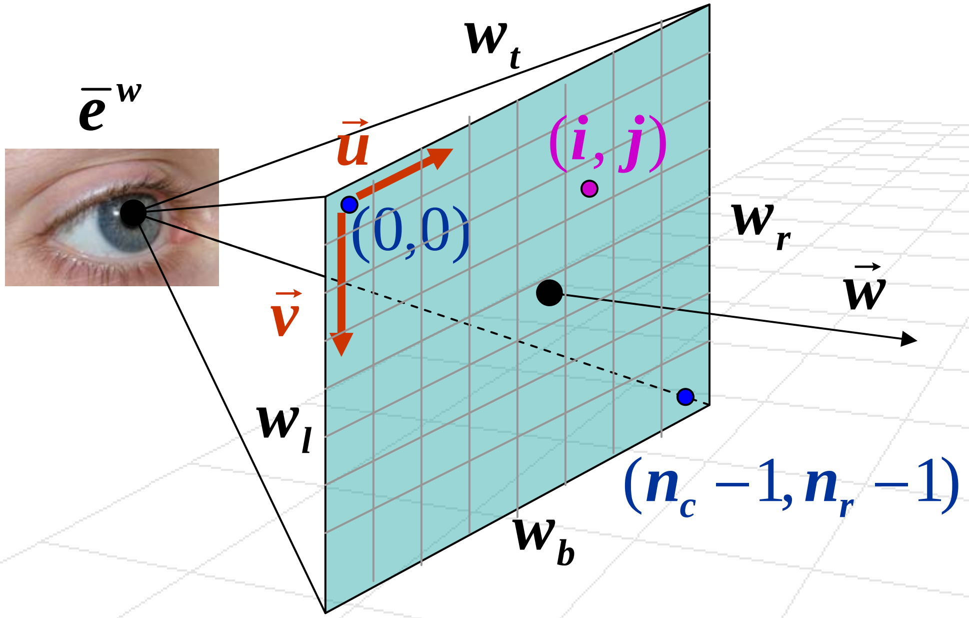
w_b **- Bottom edge of the view volume**

In camera coordinates

$$\bar{p}^c_{i,j} = (w_l + i\Delta u, w_t + j\Delta v, f)$$

$$\Delta u = \frac{w_r - w_l}{n_c - 1}$$

$$\Delta v = \frac{w_b - w_t}{n_r - 1}$$



In world coordinates

$$\bar{p}^w_{i,j} = \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ \vec{u} & \vec{v} & \vec{w} \\ \downarrow & \downarrow & \downarrow \end{bmatrix} \bar{p}^c_{i,j} + \bar{e}^w$$

Ray Casting

- We want ray through pixel (***i***, ***j***)

The ray

$$\bar{r}(\lambda) = \bar{p}^{w}_{i,j} + \lambda\,\vec{d}_{i,j}, \quad \lambda > 0$$

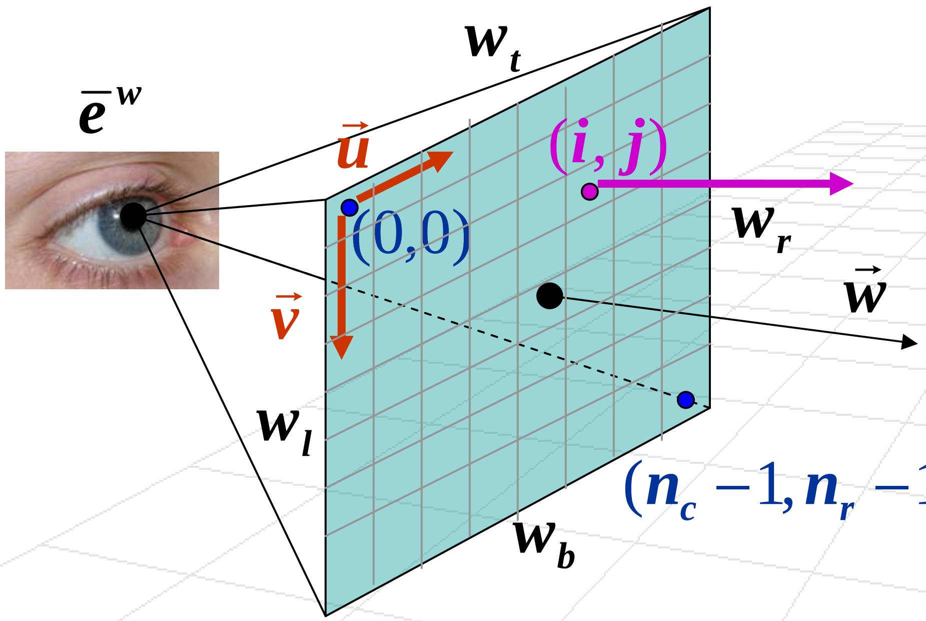
$$\vec{d}_{i,j} = \bar{p}^{w}_{i,j} - \bar{e}^{w}$$

In camera coordinates

$$\bar{p}^{c}_{i,j} = (w_l + i\Delta u, w_t + j\Delta v, f)$$

$$\Delta u = \frac{w_r - w_l}{n_c - 1}$$

$$\Delta v = \frac{w_b - w_t}{n_r - 1}$$

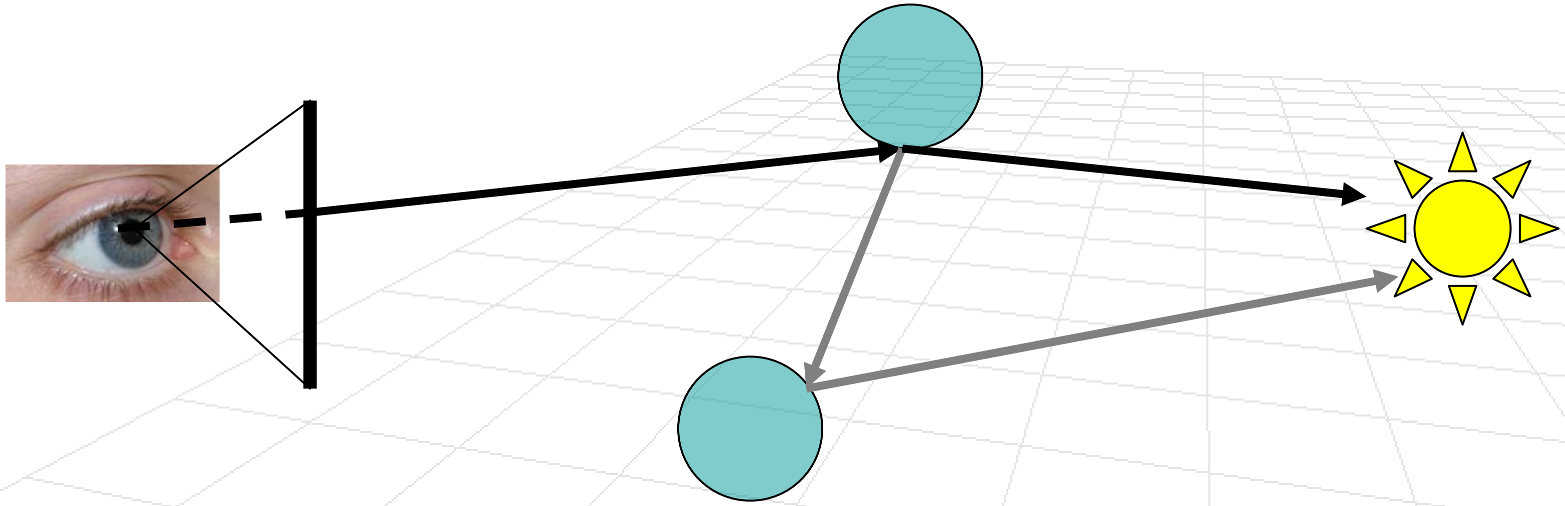


In world coordinates

$$\bar{p}^{w}_{i,j} = \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ \vec{u} & \vec{v} & \vec{w} \\ \downarrow & \downarrow & \downarrow \end{bmatrix} \bar{p}^{c}_{i,j} + \bar{e}^{w}$$

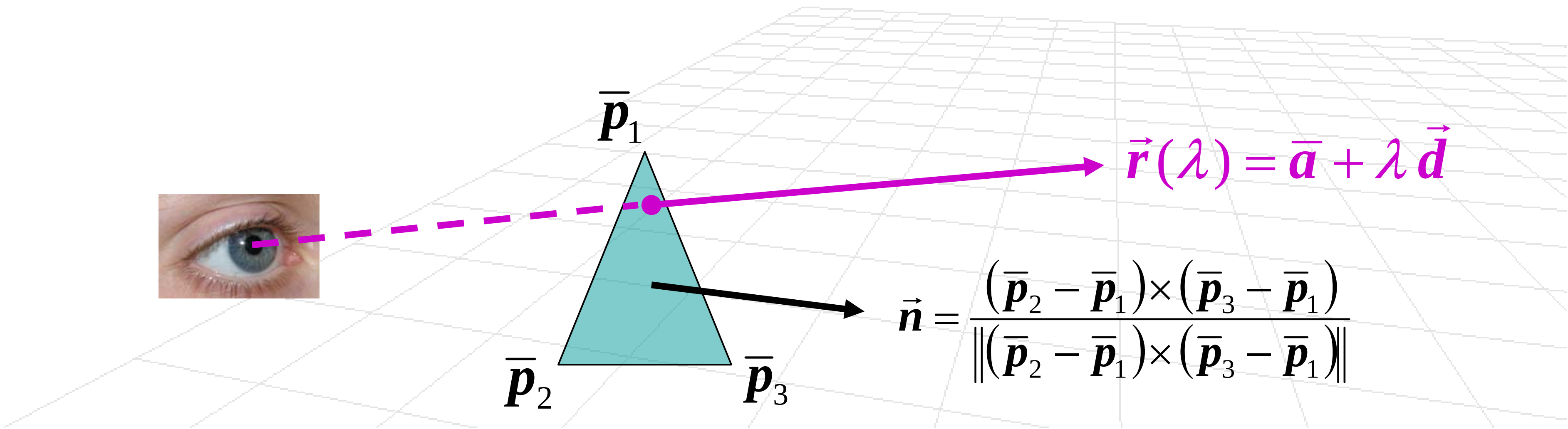
Computational Issues to Consider

- Form rays (a.k.a. ray casting)
- Find intersection of rays with objects
- Find closest object intersection (there could be multiple object intersections for any given ray)
- Find normal at the closest intersection point (a.k.a hit point)
- Evaluate reflectance model at the hit point



Finding Intersections (for triangle)

- Assume we have a triangle with vertices $\bar{\boldsymbol{p}}_1$, $\bar{\boldsymbol{p}}_2$, $\bar{\boldsymbol{p}}_3$ in counter clockwise order
- And we have a ray $\vec{\boldsymbol{r}}(\lambda) = \bar{\boldsymbol{a}} + \lambda \vec{\boldsymbol{d}}$
- There are two ways to solve for the ray-triangle intersection


$\bar{\boldsymbol{p}}_1$
$\bar{\boldsymbol{p}}_2$
$\bar{\boldsymbol{p}}_3$
$\vec{\boldsymbol{r}}(\lambda) = \bar{\boldsymbol{a}} + \lambda \vec{\boldsymbol{d}}$
$\vec{\boldsymbol{n}} = \frac{(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1)\times(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)}{\|(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1)\times(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)\|}$

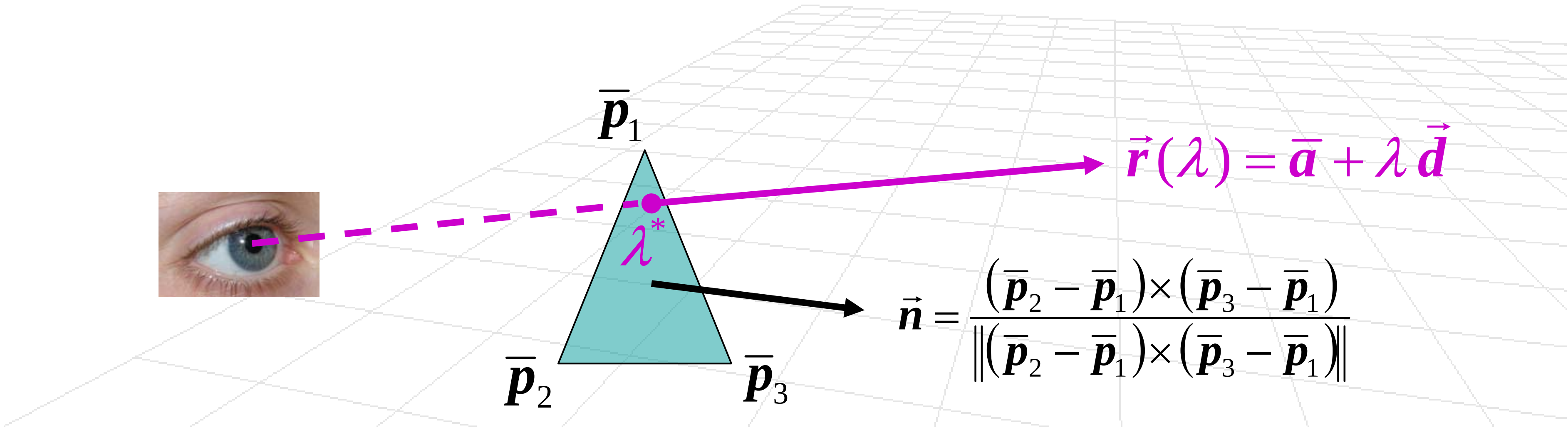
Finding Intersections (for triangle)

- Algorithm 1
  - Intersect the ray with the plane, $(\bar{\boldsymbol{p}} - \bar{\boldsymbol{p}}_1) \cdot \vec{\boldsymbol{n}} = 0$, in which the triangle lies

$$(\bar{\boldsymbol{a}} + \lambda^* \vec{\boldsymbol{d}} - \bar{\boldsymbol{p}}_1) \cdot \vec{\boldsymbol{n}} = 0$$

$$\lambda^* = \frac{(\bar{\boldsymbol{p}}_1 - \bar{\boldsymbol{a}}) \cdot \vec{\boldsymbol{n}}}{\vec{\boldsymbol{d}} \cdot \vec{\boldsymbol{n}}}$$

  - Test if the intersection is within the triangle bounds (using half-plane tests)


$\bar{\boldsymbol{p}}_1$
$\vec{\boldsymbol{r}}(\lambda) = \bar{\boldsymbol{a}} + \lambda \vec{\boldsymbol{d}}$
λ^*
$\vec{\boldsymbol{n}} = \frac{(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1) \times (\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)}{\|(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1) \times (\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)\|}$
$\bar{\boldsymbol{p}}_2$
$\bar{\boldsymbol{p}}_3$

Finding Intersections (for triangle)

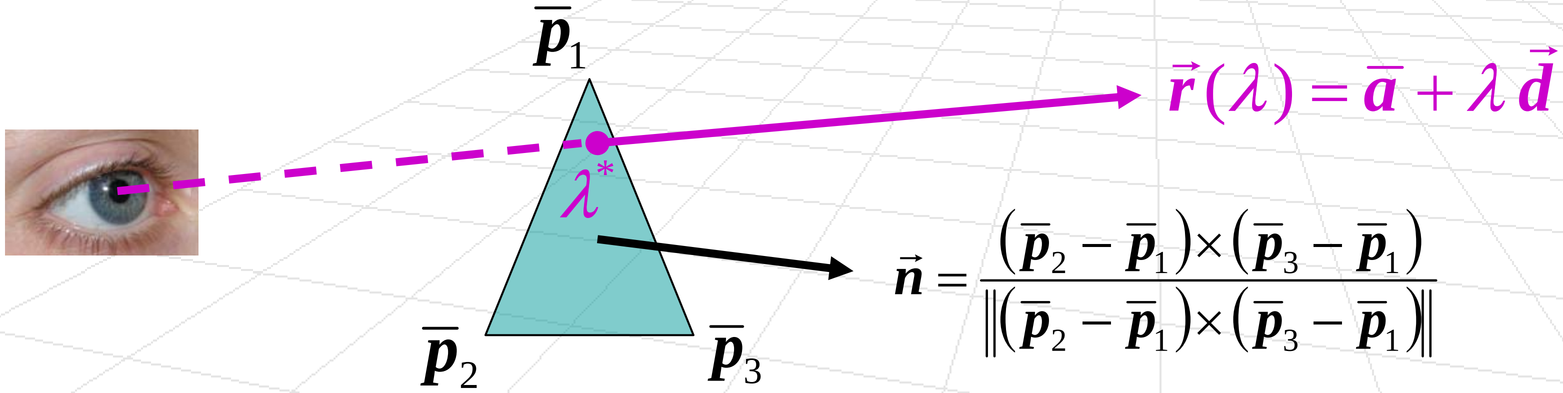
- Algorithm 2
  - Use an alternative parameterization for the plane

$$\boldsymbol{p}(\alpha,\beta) = \boldsymbol{p}_1 + \alpha(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1) + \beta(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)$$

and solve for the parameters α, β , which leads 3x3 system

$$\begin{bmatrix} \uparrow & \uparrow & \uparrow \\ -(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1) & -(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1) & \vec{\boldsymbol{d}} \\ \downarrow & \downarrow & \downarrow \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \lambda^* \end{bmatrix} = (\bar{\boldsymbol{p}}_1 - \bar{\boldsymbol{a}})$$

  - If intersection is in the triangle then the all following conditions will hold: $\alpha \geq 0, \quad \beta \geq 0, \quad \alpha + \beta \leq 1$


$\bar{\boldsymbol{p}}_1$
$\vec{\boldsymbol{r}}(\lambda) = \bar{\boldsymbol{a}} + \lambda\vec{\boldsymbol{d}}$
λ^*
$\bar{\boldsymbol{p}}_2$
$\bar{\boldsymbol{p}}_3$

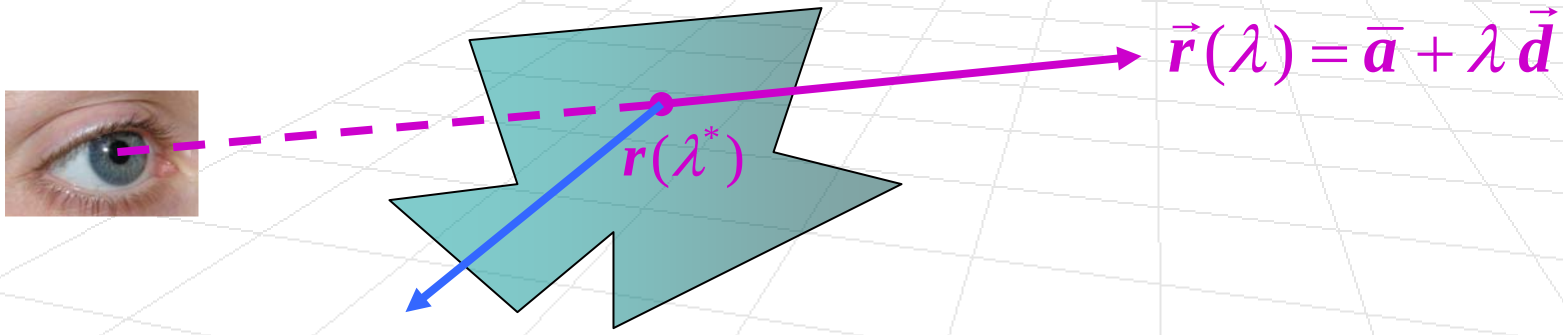
$$\vec{\boldsymbol{n}} = \frac{(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1)\times(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)}{\left\|(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1)\times(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)\right\|}$$

Finding Intersections (for general planar polygons)

- Find the intersection with the general planar polygon
  - Same as Algorithm 2
- Construct a ray in the plane originating at $\boldsymbol{r}(\lambda^*)$ and count the number of intersections with polygon sides:

# of intersections is odd => point inside the polygon

# of intersections is even => point outside the polygon


$\vec{\boldsymbol{r}}(\lambda) = \bar{\boldsymbol{a}} + \lambda\,\vec{\boldsymbol{d}}$
$\boldsymbol{r}(\lambda^*)$

Finding Intersections (for spheres)

- Let's consider simple case, unit length sphere at a given point $\bar{p}$, $\|\bar{p}\|^2=1$, by first substituting:

$$(\bar{a}+\lambda^*\vec{d})\cdot(\bar{a}+\lambda^*\vec{d})-1=0$$

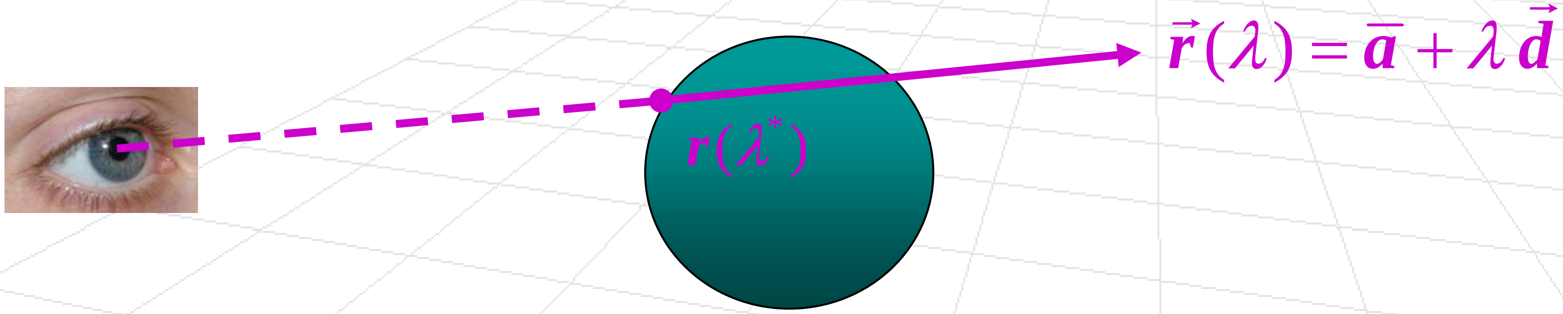
then expanding:

$$A(\lambda^*)^2+2B\lambda^*+C=0$$, where

$$A=\vec{d}\cdot\vec{d} \qquad C=\bar{a}\cdot\bar{a}-1$$
$$B=\bar{a}\cdot\vec{d} \qquad D=B^2-AC$$

Producing solution:

$$\lambda^*=\frac{-2B\pm\sqrt{4B^2-4AC}}{2A}=-\frac{B}{A}\pm\frac{\sqrt{D}}{A}$$



Finding Intersections (for spheres)

$$\lambda^* = \frac{-2B \pm \sqrt{4B^2 - 4AC}}{2A} = -\frac{B}{A} \pm \frac{\sqrt{D}}{A} \qquad A = \vec{d}\cdot\vec{d} \quad C = \bar{a}\cdot\bar{a} - 1$$

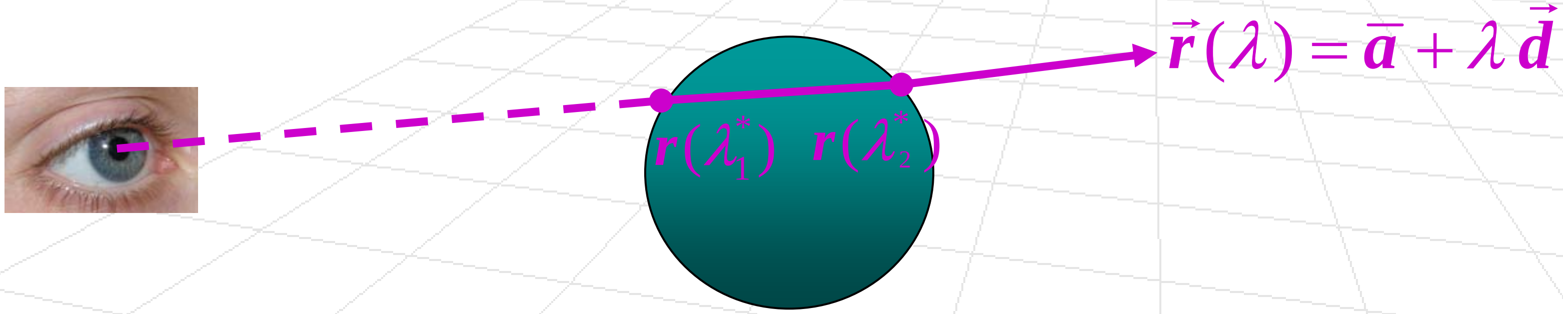
$$B = \bar{a}\cdot\vec{d} \quad D = B^2 - AC$$

- Cases
  - D < 0 no intersection
  - D = 0 1 hit (ray grazes the sphere)
  - D > 0 2 hits with at λ_1^*, λ_2^*

$\lambda_1^* < 0, \lambda_2^* < 0$ - Both hits are behind viewplane (not visible)

$\lambda_1^* > 0, \lambda_2^* < 0$ - Eye inside sphere, first hit is valid

$\lambda_1^* > \lambda_2^* > 0$ - Both hits are valid (in front og viewplane), second closer


$\vec{r}(\lambda) = \bar{a} + \lambda\vec{d}$
$r(\lambda_1^*)$
$r(\lambda_2^*)$