Ray Tracing

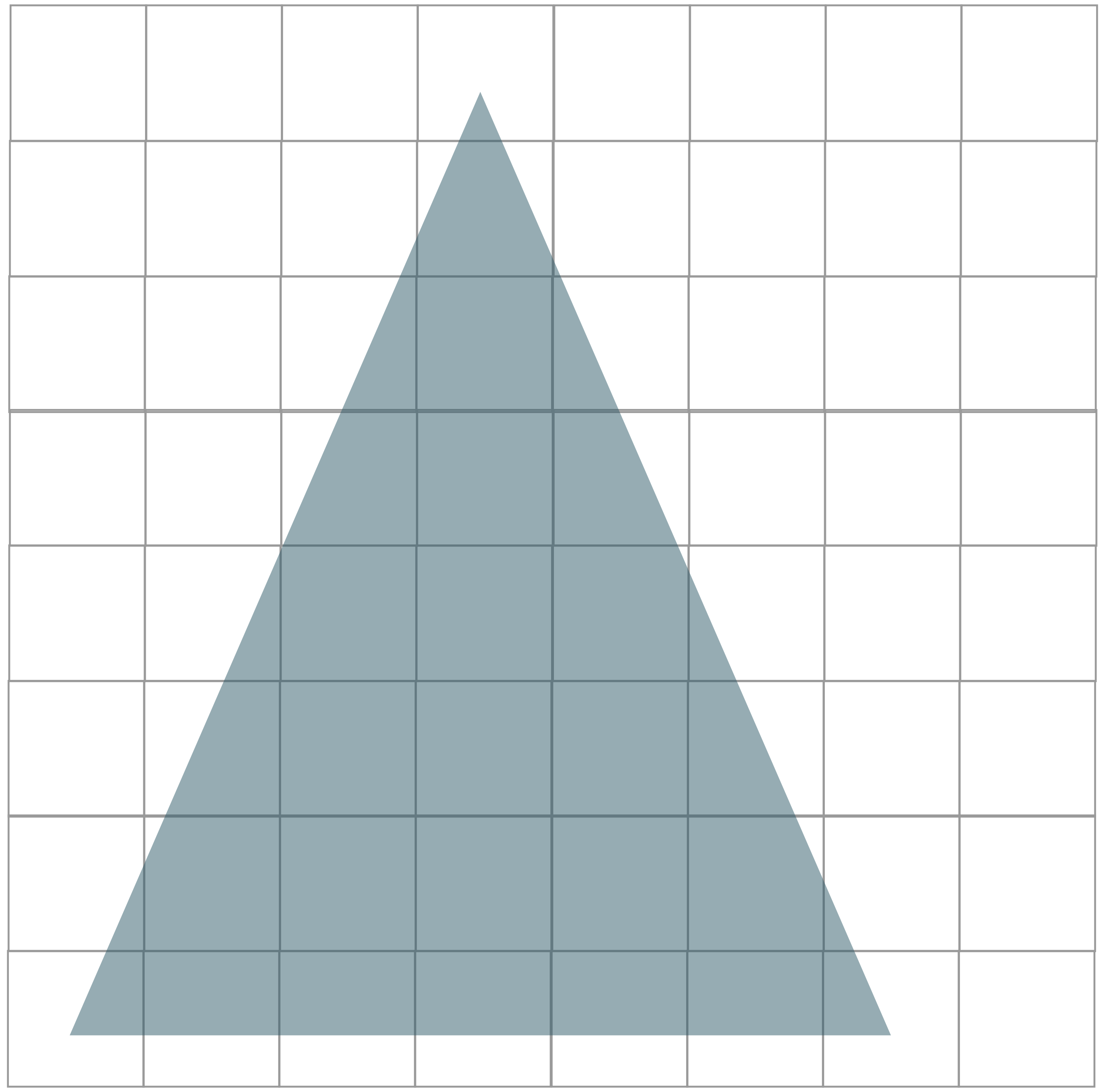
CS 385 - Class 27
3 May 2022

Background

Rasterization

- This is what we've been studying all semester
- Determine which pixels are affected by each geometric primitive
  - shade only those pixels
- Use some scheme to determine which color owns the pixel
  - painter's algorithm (last color wins)
  - depth buffering
  - blending

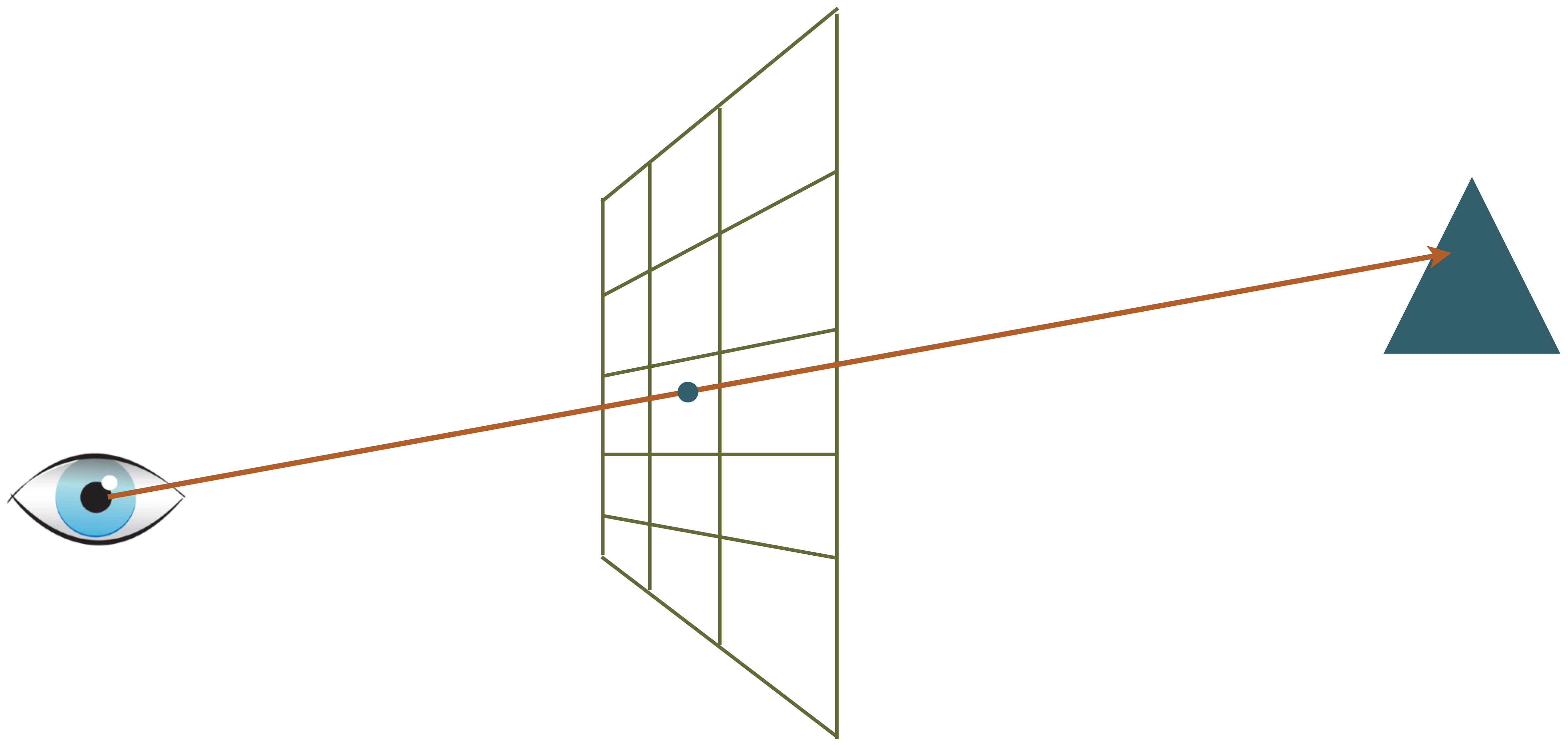
```
foreach ( primitive in the scene ) {
  foreach ( fragment in primitive ) {
    shade( fragment );
  }
}
```



Ray Tracing

- Exactly the opposite of rasterization
- Determine which primitives affect each pixel
- Generate a ray from the viewing through the pixel of interest
  - see if that ray hits any objects in the scene

```
foreach ( pixel in the frame ) {
  initialize( ray );
  foreach ( primitive in the scene ) {
    intersect( ray, primitive );
  }
}
```



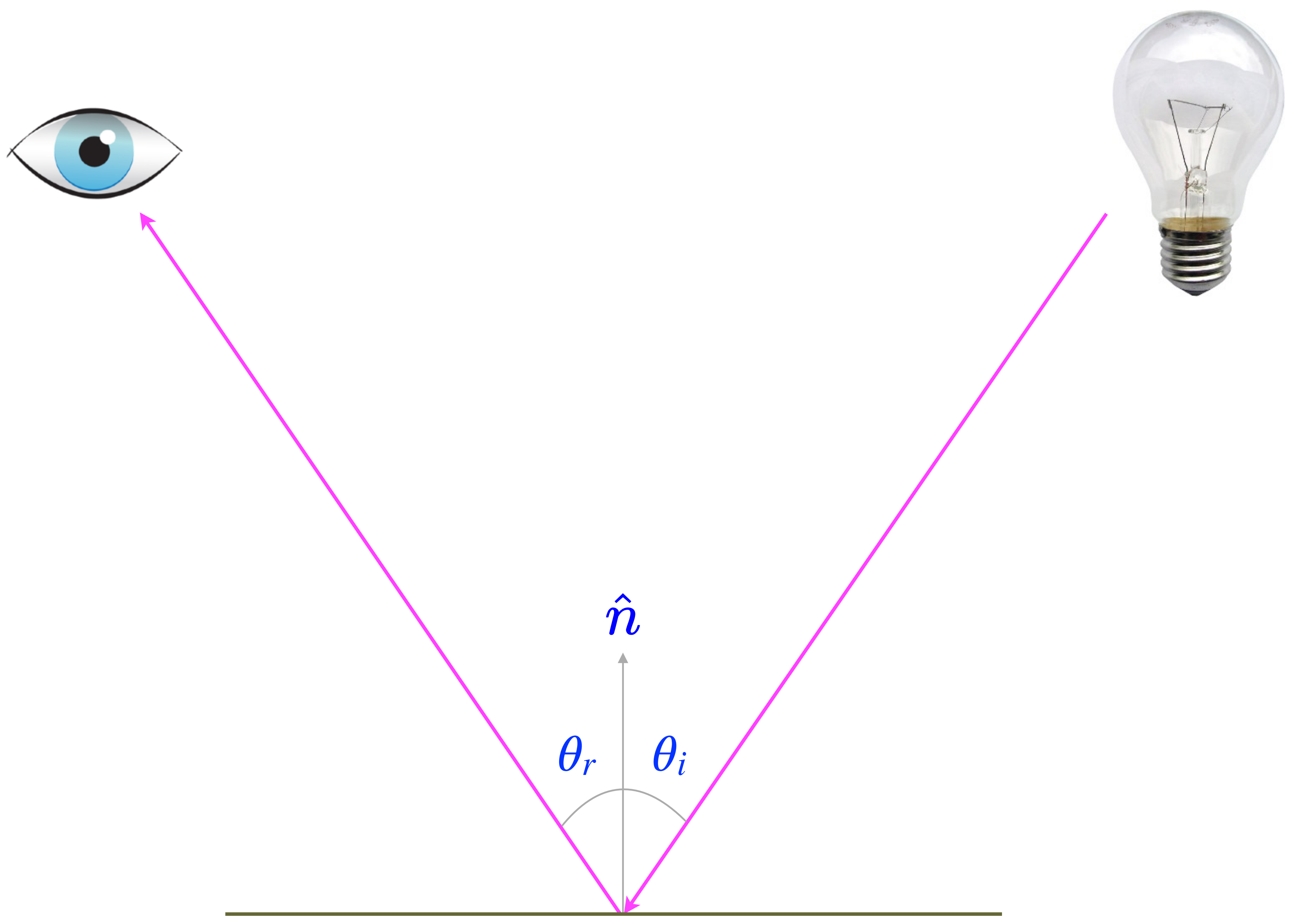
The Basic Idea

- Trace rays of light form a light source to the eye, reflecting them around the scene
- Employ the *law of reflection*

$$\theta_i = \theta_r$$

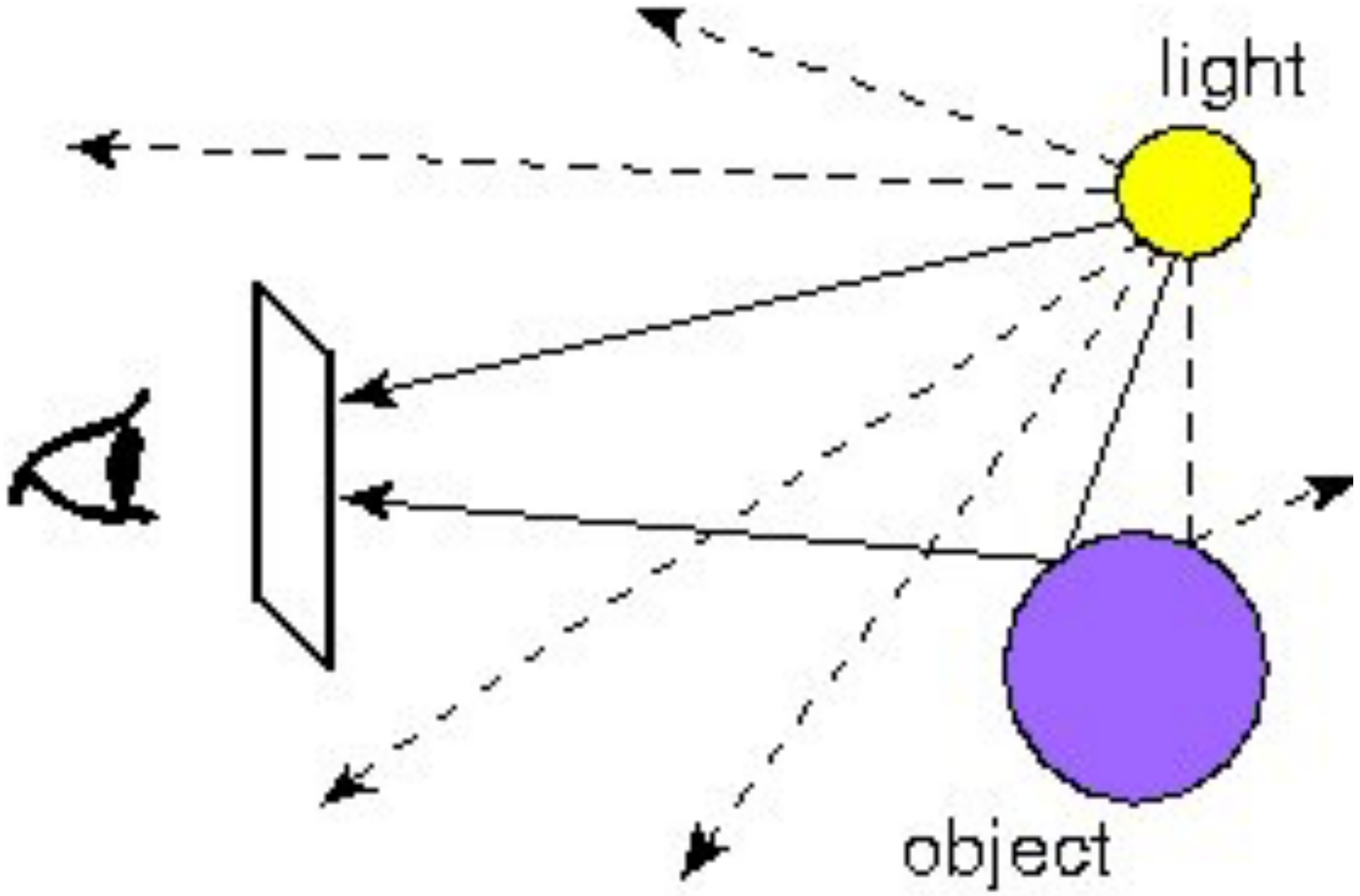
  - around the surface normal $\hat{n}$

- Determine a color at each intersection



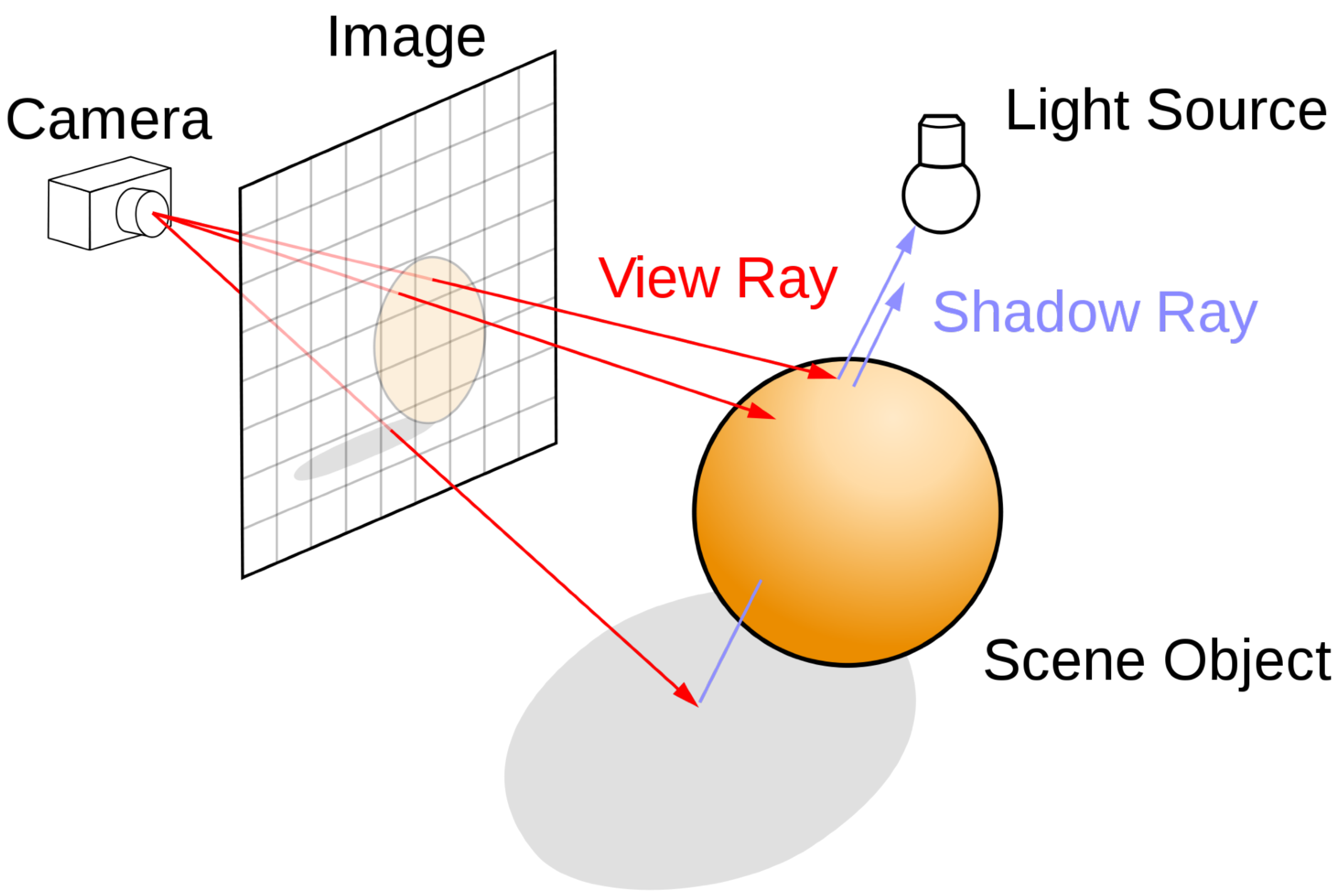
Forward Ray Tracing

- Trace every ray from the light source, and see if it intersects the imaging plane
- Lots of work with little reward
- *This* is not the way to do ray tracing!



Backward Ray Tracing

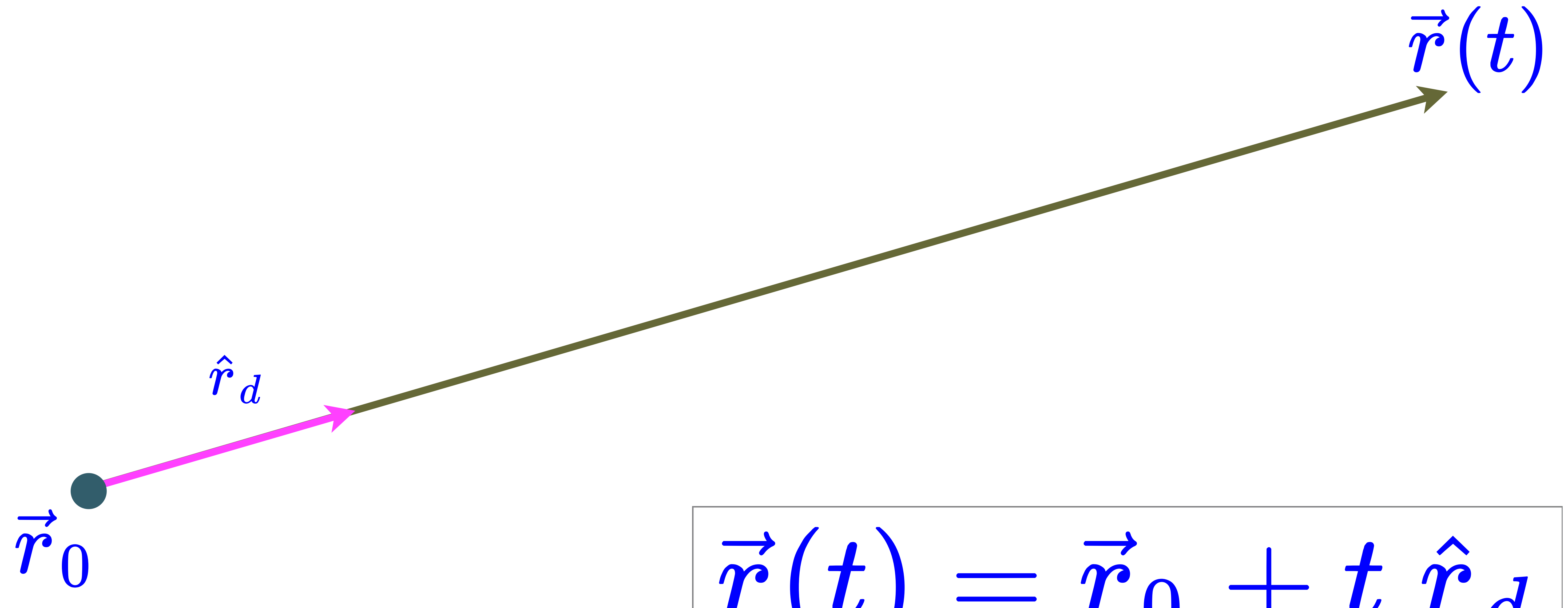
- Trace rays from the eye into the scene
- Much more reward for work done
- *This* is what is generally meant when people say "ray tracing"



Fundamentals

What's a Ray?

- Simply, a vector with an anchor point



$$\boxed{\vec{r}(t) = \vec{r}_0 + t\,\hat{r}_d}$$

Generating Rays for Tracing

- Each ray for our scene starts at the same point: the eye
- Ray intersects the image plane at a application-selected point
- Location and orientation of the image plane is also controlled by the application
  - it's just like setting the camera in applications

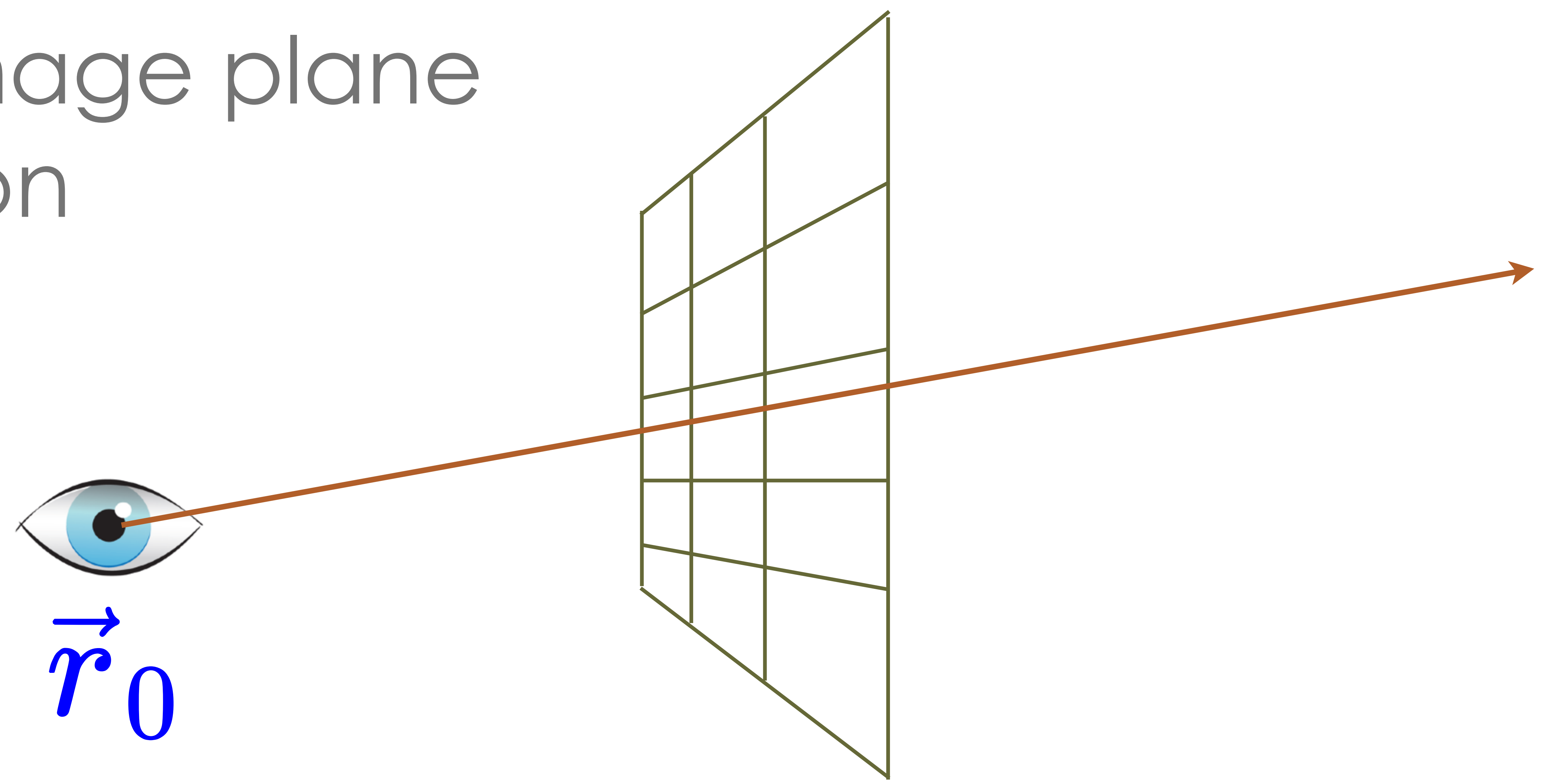
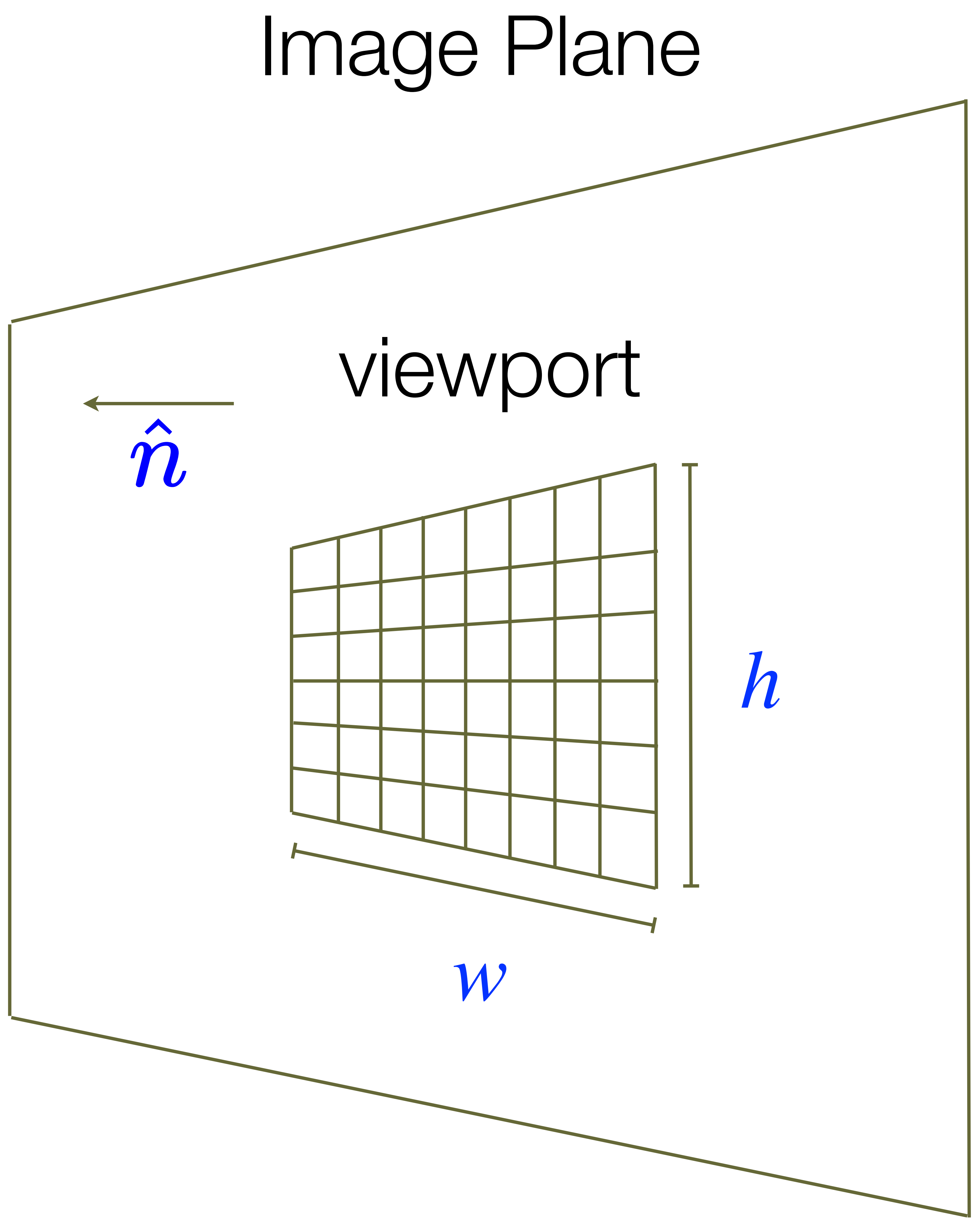


Image Planes

- Similar to our near-clipping planes
- Application specifies a rectangular region in a plane in space
- That region is partitioned into (usually) equal-sized pixels
- Usually, the line-of-sight is parallel with the image plane's normal
- Our familiar eye coordinates simplify the math
    - although, any plane in space can be the image plane



Computing a Ray's Values

- Assume we're in our favorite eye coordinates
  - eye at the origin
  - image plane located at some point down the $-z$ axis
- Further suppose:
  - viewport extends from [-1, 1] in x & y
  - viewport partitioned into 100×100 pixels

$$\vec{r}_0 = \vec{0} \quad \text{eye position}$$

$$\vec{c}_p = (0, 0, -z) \quad \text{center point on plane}$$

$$\Delta x = \frac{2}{100}, \Delta y = \frac{2}{100} \quad \text{size of a pixel}$$

$$\vec{o} = (n\Delta x, m\Delta y, 0) \quad \text{pixel offset in plane}$$

$$\begin{aligned} \vec{p} &= \vec{c}_p + \vec{o} \\ &= (n\Delta x, m\Delta y, -z) \end{aligned} \quad \text{pixel location on plane}$$

$$\vec{r}_d = \vec{p} - \vec{r}_0 \quad \text{ray direction}$$

$$\hat{r}_d = ||\vec{r}_d|| \quad \text{normalized ray direction}$$

$$\vec{r}(t) = \vec{r}_0 + t\,\hat{r}_d \quad \text{final parameterized ray}$$

Intersections

Intersecting a Sphere

$$x^2 + y^2 + z^2 = a^2 \qquad \text{explicit form}$$
$$\vec{r} = (x, y, z)$$
$$\vec{r} \cdot \vec{r} - a^2 = 0 \qquad \text{implicit form}$$

$$\begin{aligned}
\vec{r}(t) \cdot \vec{r}(t) - a^2 &= 0 \\
(\vec{r}_0 + t\hat{r}_d) \cdot (\vec{r}_0 + t\hat{r}_d) - a^2 &= \\
\vec{r}_0 \cdot \vec{r}_0 + 2t\,\vec{r}_0 \cdot \hat{r}_d + t^2 \hat{r}_d \cdot \hat{r}_d - a^2 &= \\
t^2 \hat{r}_d \cdot \hat{r}_d + 2t\,\vec{r}_0 \cdot \hat{r}_d + \vec{r}_0 \cdot \vec{r}_0 - a^2 &= \\
\text{however, } \hat{n} \cdot \hat{n} = 1 & \\
t^2 + t(2\,\vec{r}_0 \cdot \hat{r}_d) + (\vec{r}_0 \cdot \vec{r}_0 - a^2) &= 0
\end{aligned}$$

Ever heard of the quadratic equation?

$$ax^2 + bx + c = 0 \implies x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$t^2 + t\underbrace{(2\,\vec{r}_0 \cdot \hat{r}_d)}_{b} + \underbrace{(\vec{r}_0 \cdot \vec{r}_0 - a^2)}_{c} = 0$$

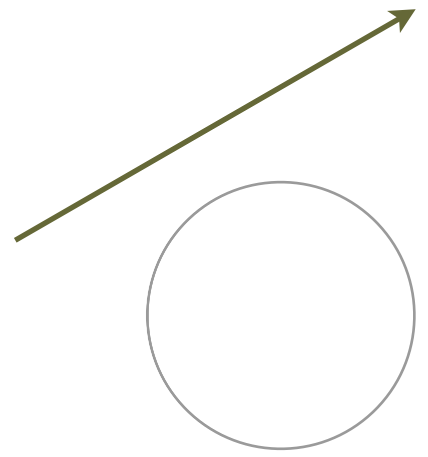
$$\boxed{t = -(\vec{r}_0 \cdot \hat{r}_d) \pm \sqrt{(\vec{r}_0 \cdot \hat{r}_d)^2 - (\vec{r}_0 \cdot \vec{r}_0 - a^2)}}$$

Using the Discriminant

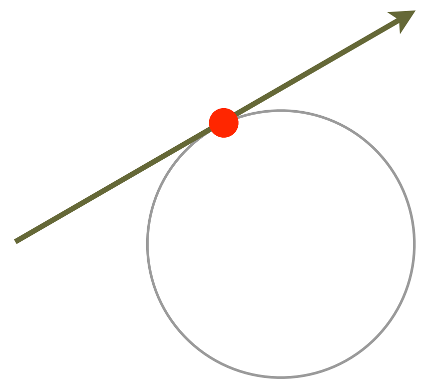
- The discriminant is the value under the square root
  - its value tells how the sphere and ray intersected

$$d = \sqrt{(\vec{r}_0 \cdot \hat{r}_d)^2 - (\vec{r}_0 \cdot \vec{r}_0 - a^2)}$$

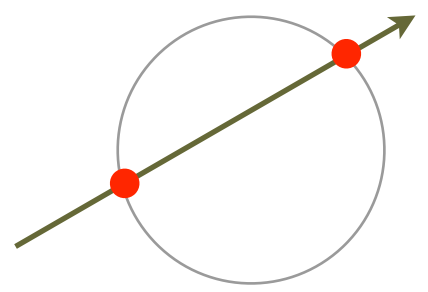
$d < 0$



$d = 0$



$d > 0$



choose the smaller t value
for the closer intersection.

Intersecting a Plane

$$Ax + By + Cz = D \qquad \text{explicit form}$$

$$\vec{r} = (x, y, z) \;\text{ and }\; \hat{n} = (A, B, C)$$

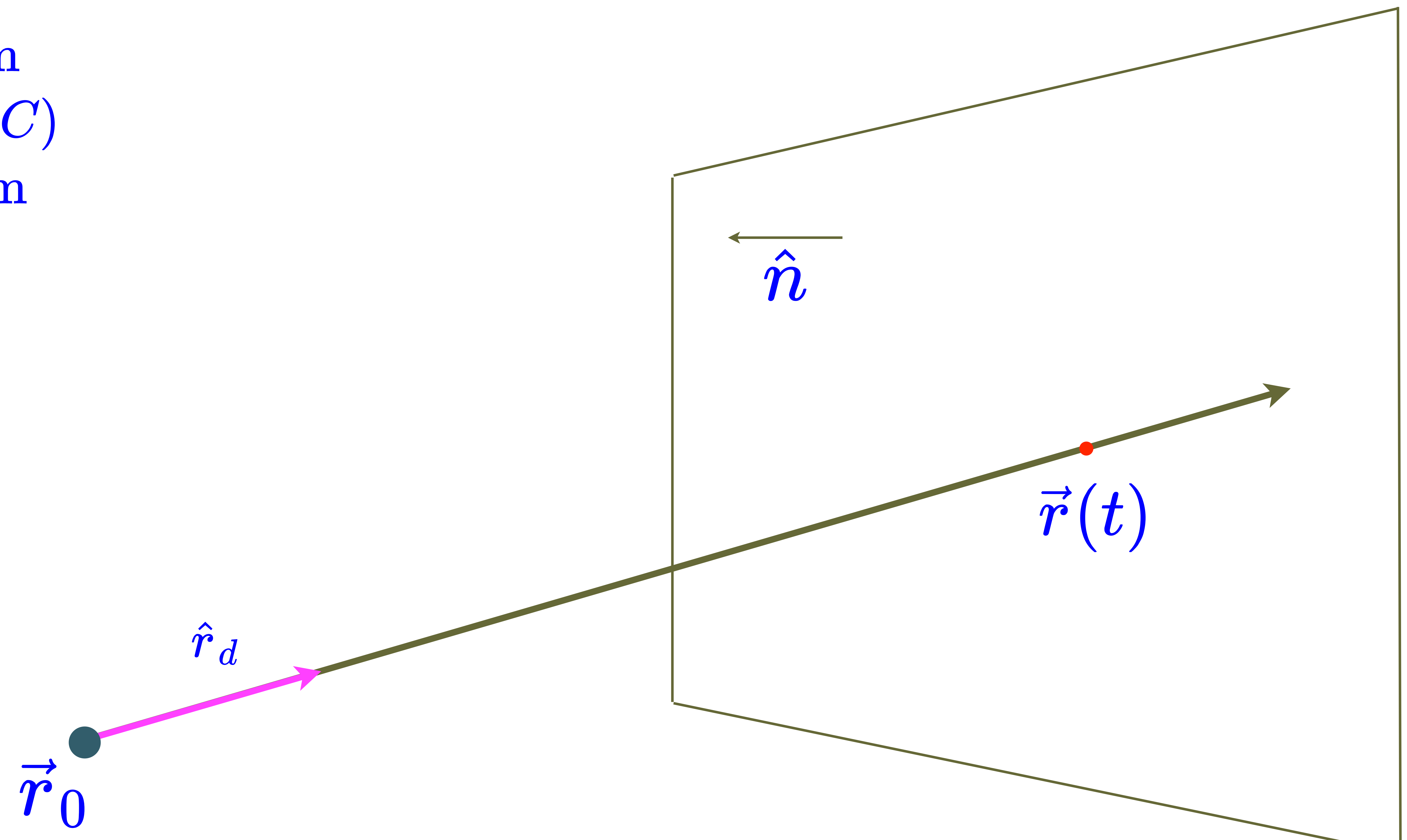
$$\vec{r} \cdot \hat{n} - D = 0 \qquad \text{implicit form}$$

$$\vec{r}(t) \cdot \hat{n} - D = 0$$

$$(\vec{r}_0 + t\hat{r}_d) \cdot \hat{n} - D =$$

$$(\vec{r}_0 \cdot \hat{n}) + t(\hat{r}_d \cdot \hat{n}) - D =$$

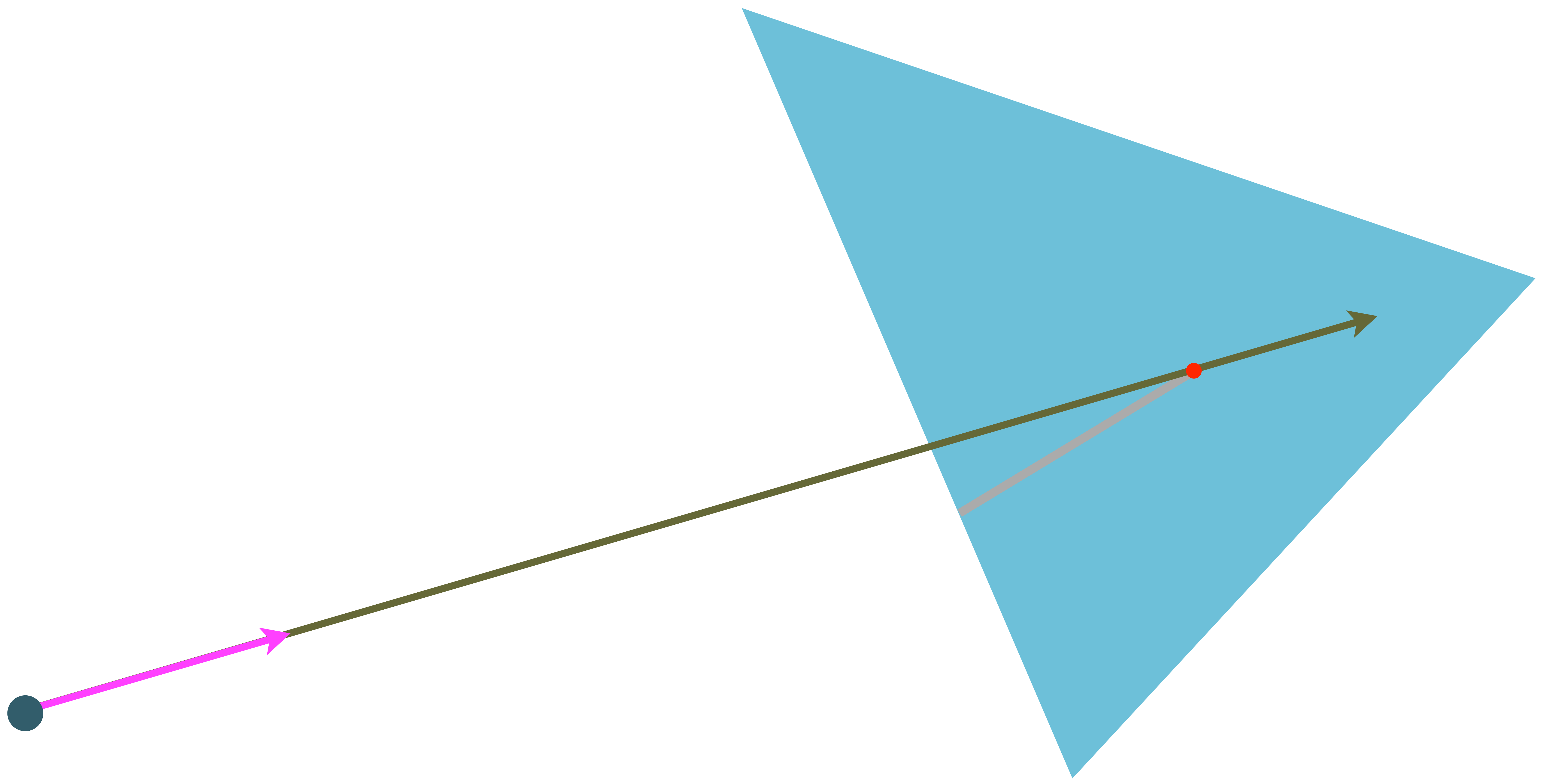
$$\boxed{t = \frac{D - \vec{r}_0 \cdot \hat{n}}{\hat{r}_d \cdot \hat{n}}}$$



ray is parallel to plane if $\hat{r}_d \cdot \hat{n} = 0$,
which means no intersection

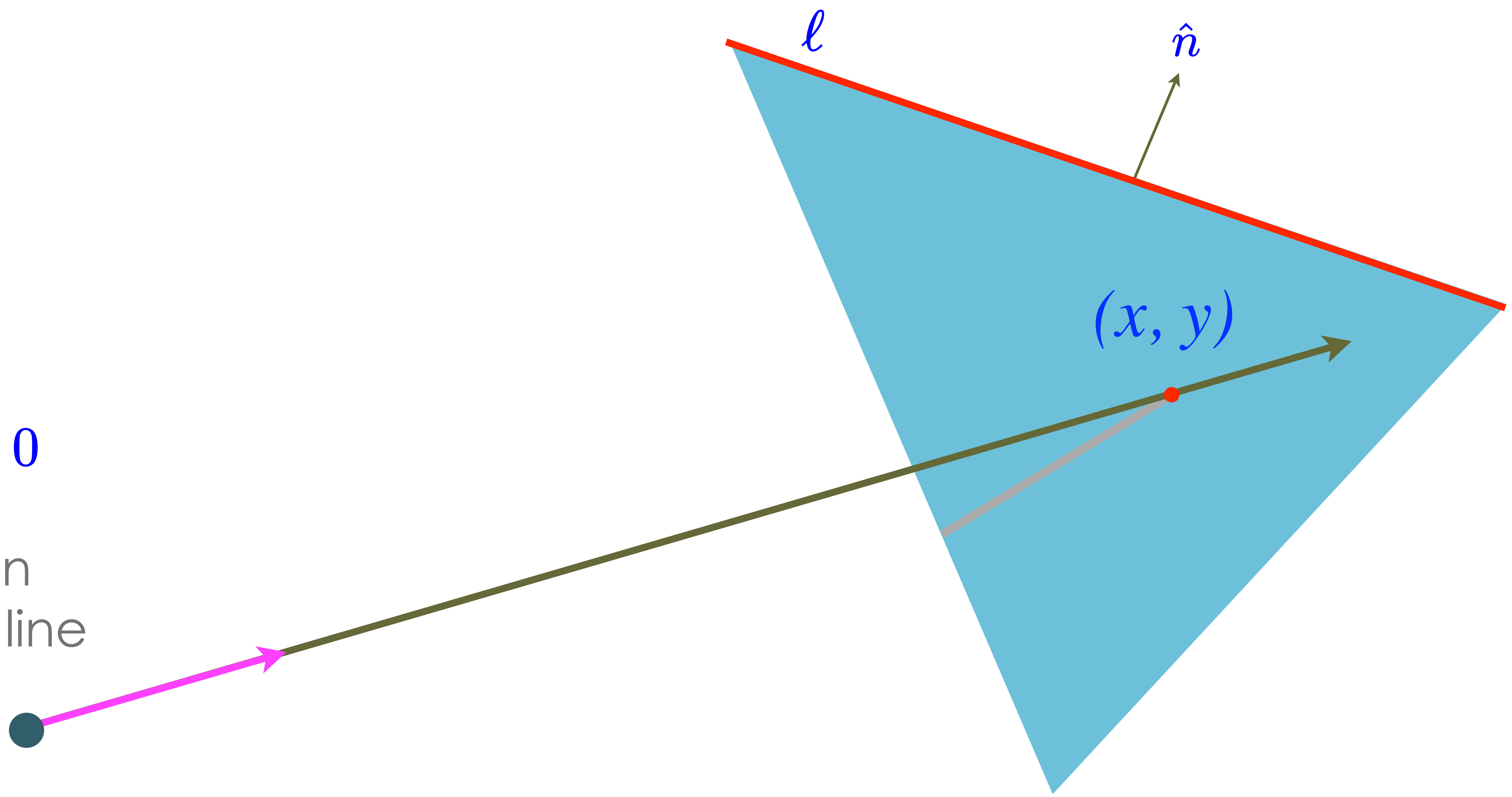
Intersecting a Triangle

- Intersect the plane of the triangle
- Then determine if the intersection is inside of the triangle



Intersecting a Triangle

- Intersect the plane of the triangle
- Then determine if the intersection is inside of the triangle
- Remember our old friend $y = mx + b$
  - not quite the right formula
- We need the *implicit form* $\hat{n} \cdot \vec{r} + c = 0$
- Recall that if $\hat{n} \cdot \vec{r} + c > 0$, is true, then the point is on the normal side of the line
- If the point is negative for all three edges, then it's inside of the triangle



What Happens when We Hit Something? (part 1)

- Two things:
  1. record the intersection point (we'll use it later when we compute a color)
  2. generate a new ray and repeat the process
- That "repeat the process" is code for *recursion*
- Ray tracing is inherently recursive
  - recursing down records all the intersections
  - unwinding up computes colors