

CSC 433

Computer Graphics

Alon Efrat
Credit: Joshua Levine

Lecture 09

Ray Tracing

Today's Agenda

- Reminders:
 - A02 due!
 - A03 posted!
- Goals for today:
 - Begin our discussing of 3D rendering with Ray Tracing

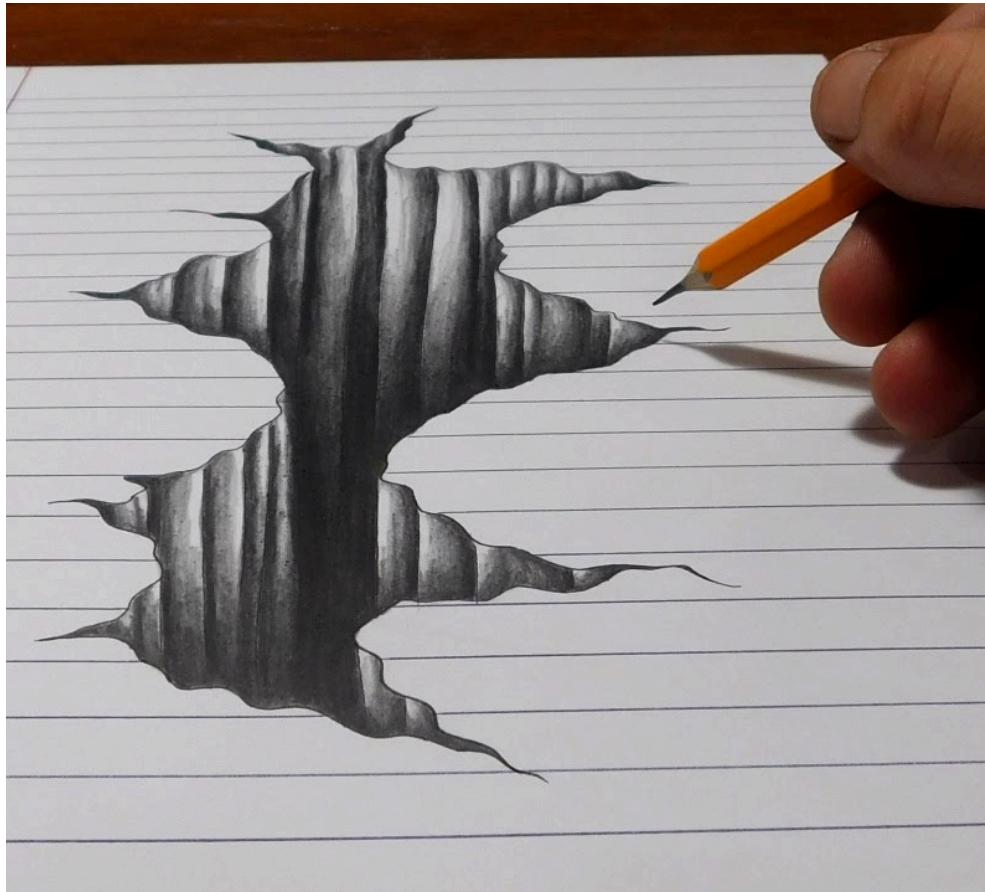
Rendering

What is Rendering?

“Rendering is the task of taking three-dimensional objects and producing a 2D image that shows the objects as viewed from a particular viewpoint”

Two Ways to Think About How We Make Images

- Drawing
- Photography



Two Ways to Think About Rendering

- Object-Ordered
 - Decide, for every object in the scene, its contribution to the image
- Image-Ordered
 - Decide, for every pixel in the image, its contribution from every object

Two Ways to Think About Rendering

- Object-Ordered or
Rasterization

```
for each object {  
    for each image pixel {  
        if (object affects pixel)  
        {  
            do something  
        }  
    }  
}
```

- Image-Ordered or
Ray Tracing

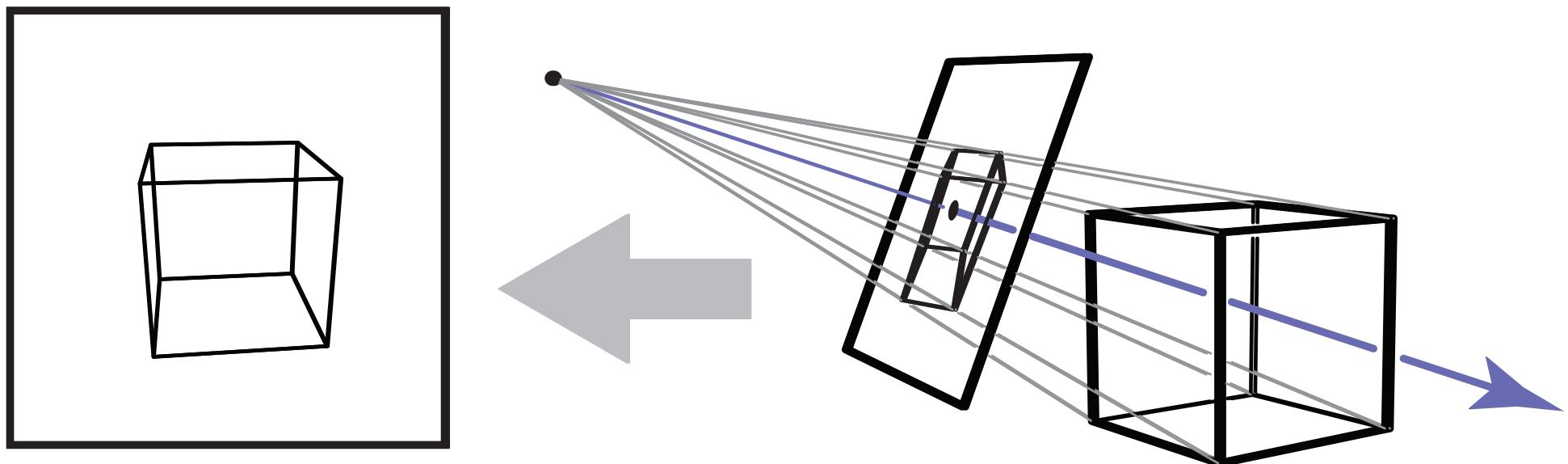
```
for each image pixel {  
    for each object {  
        if (object affects pixel)  
        {  
            do something  
        }  
    }  
}
```

TODAY

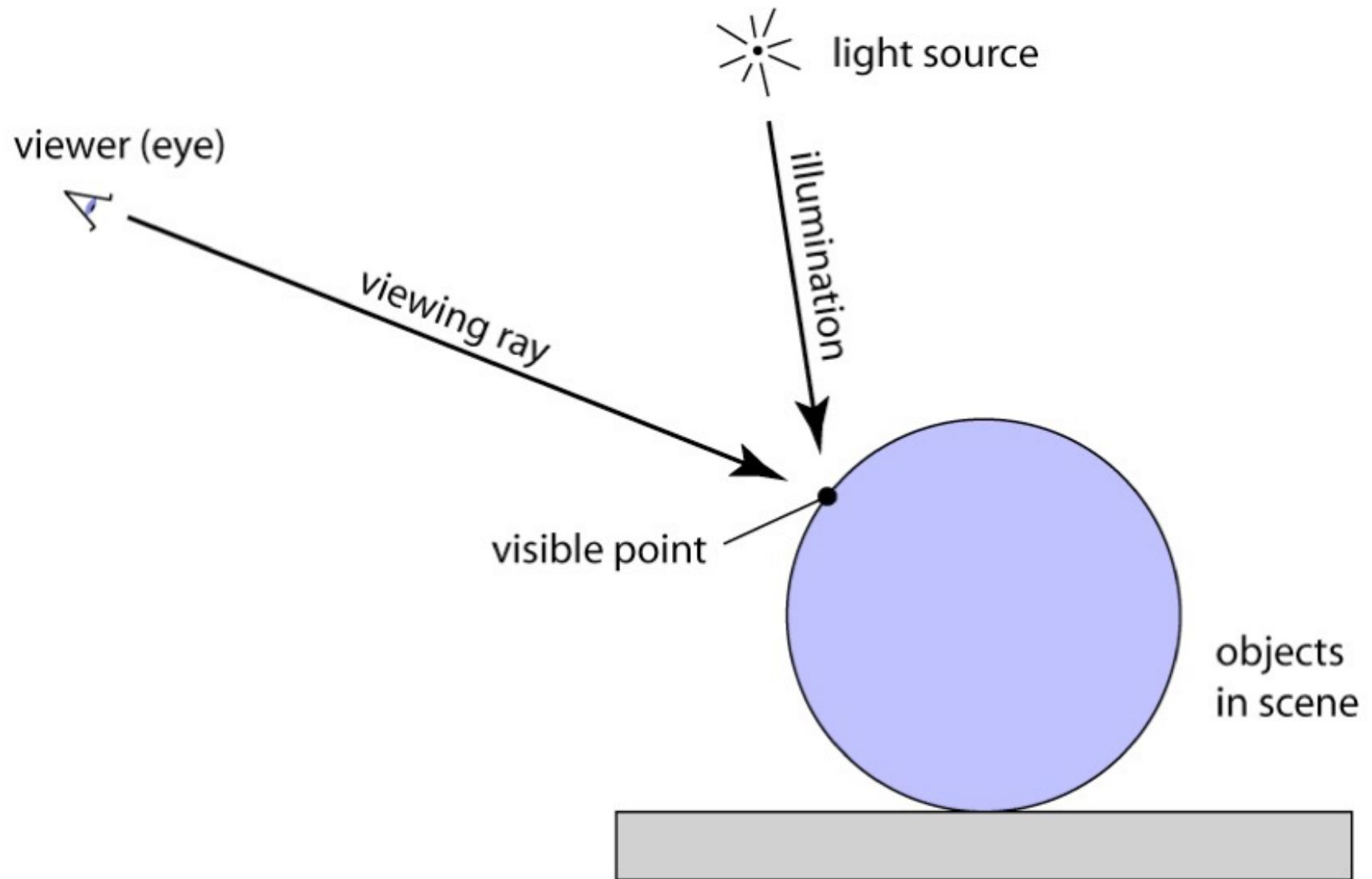
Basics of Ray Tracing

Idea of Ray Tracing

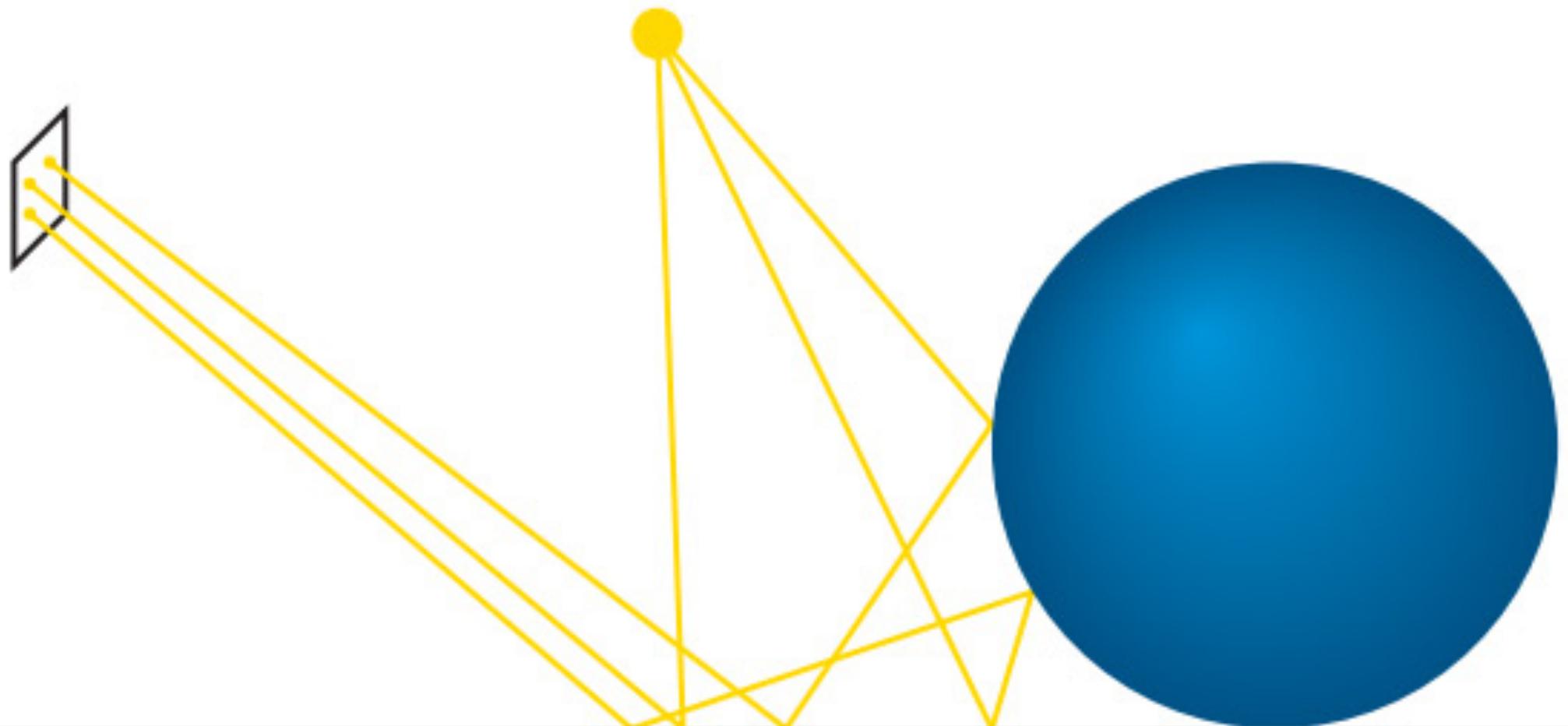
- Ask first, for each pixel: what belongs at that pixel?
- Answer: The set of objects that are visible if we were standing on one side of the image looking into the scene



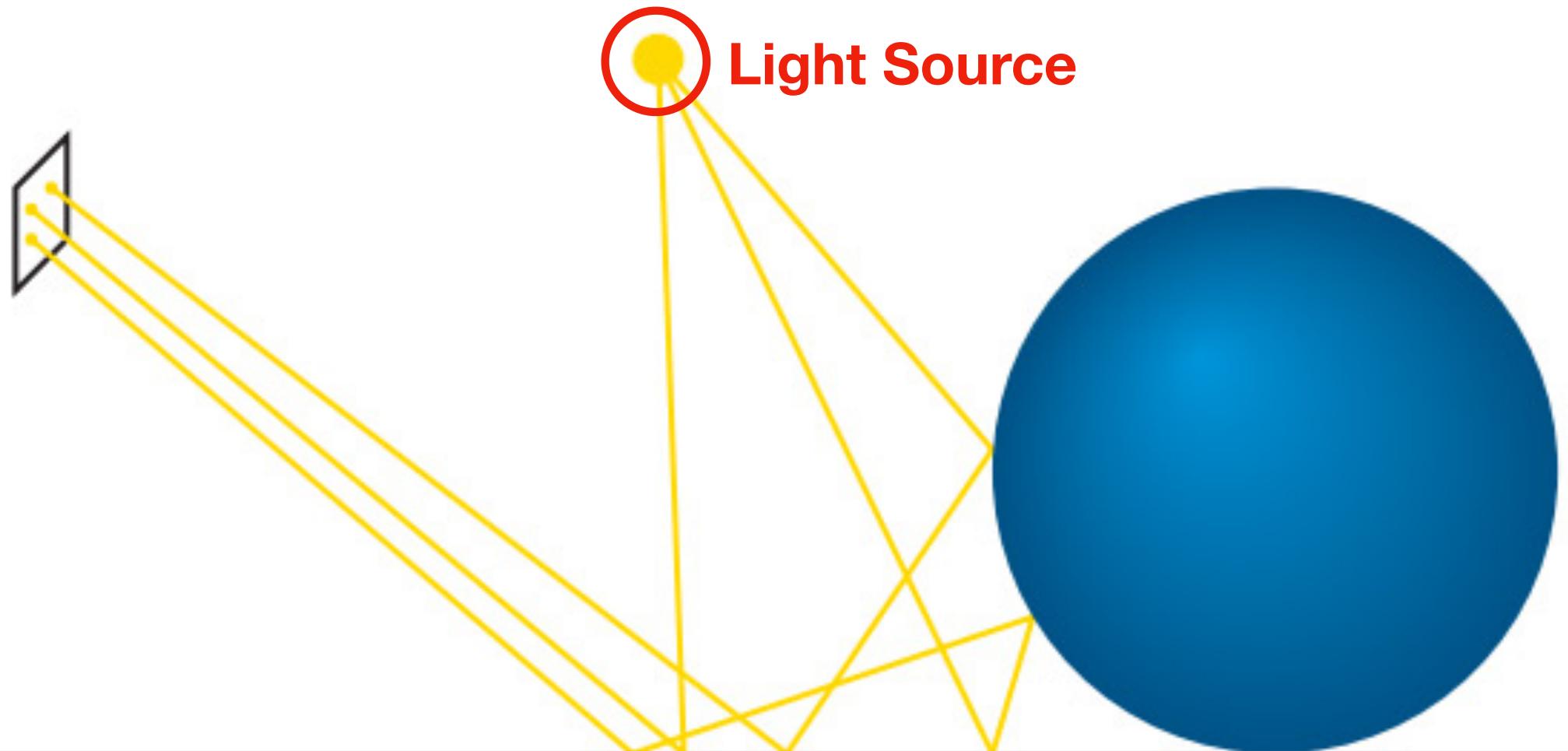
Key Concepts, in Diagram



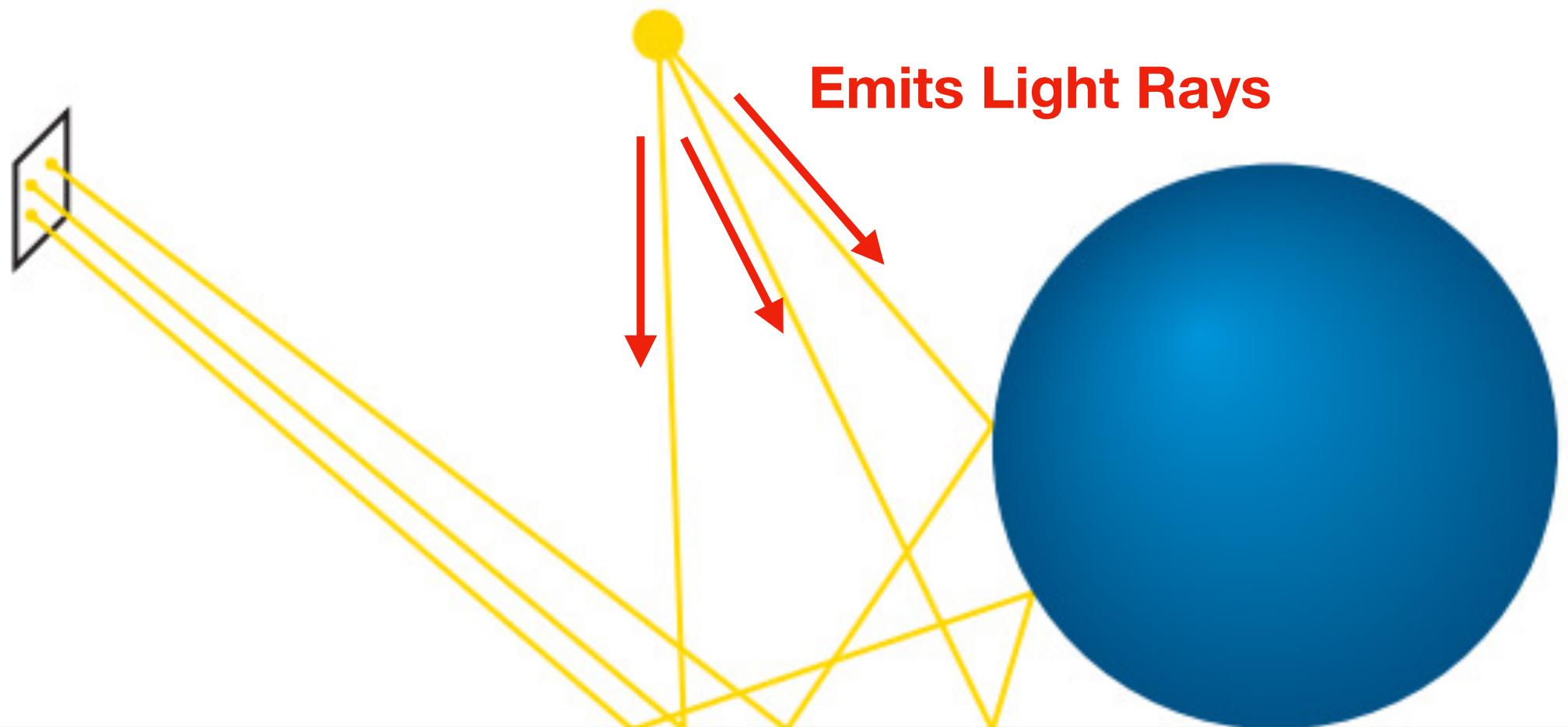
Idea: Using Paths of Light to Model Visibility



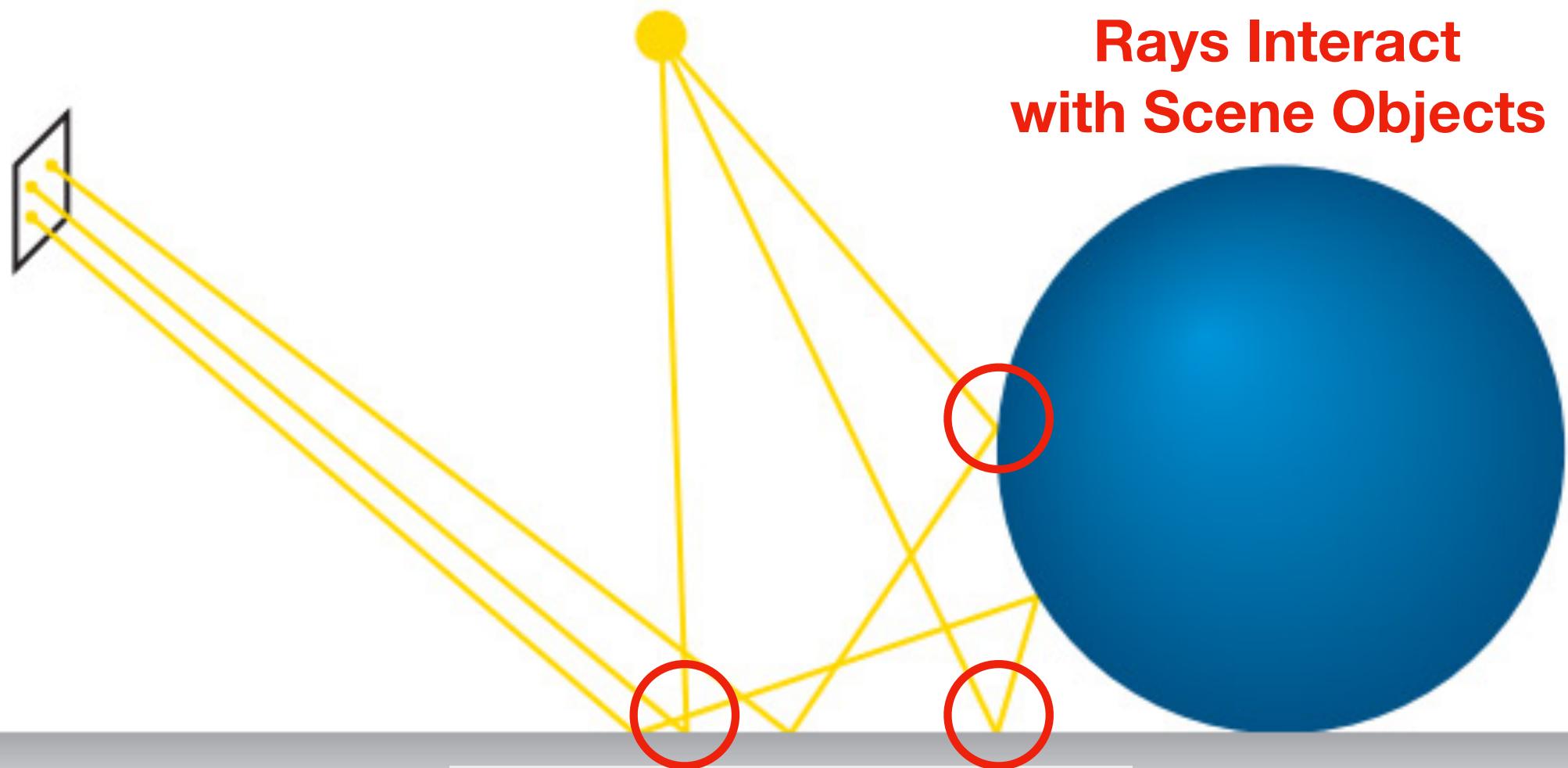
Using Paths of Light to Model Visibility



Using Paths of Light to Model Visibility

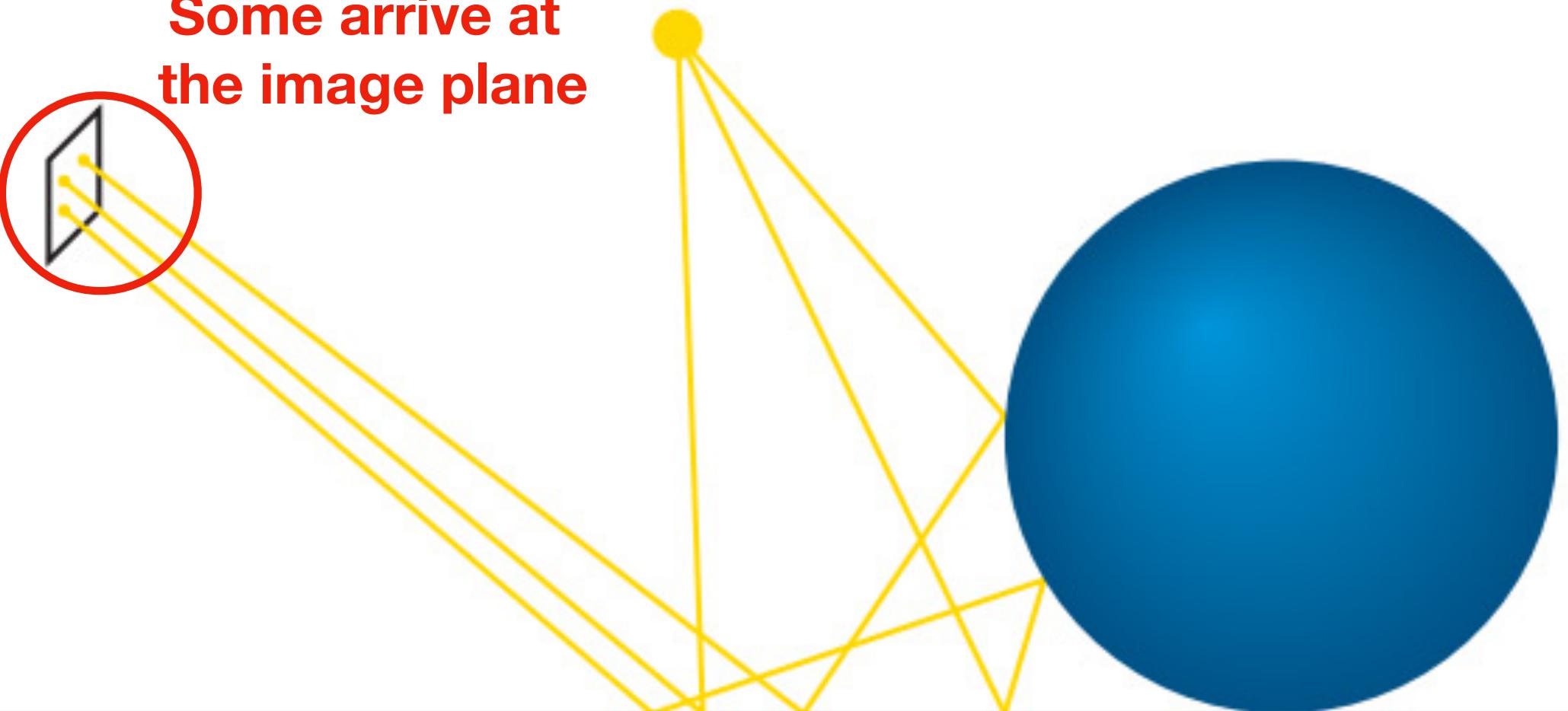


Using Paths of Light to Model Visibility



Using Paths of Light to Model Visibility

Some arrive at
the image plane



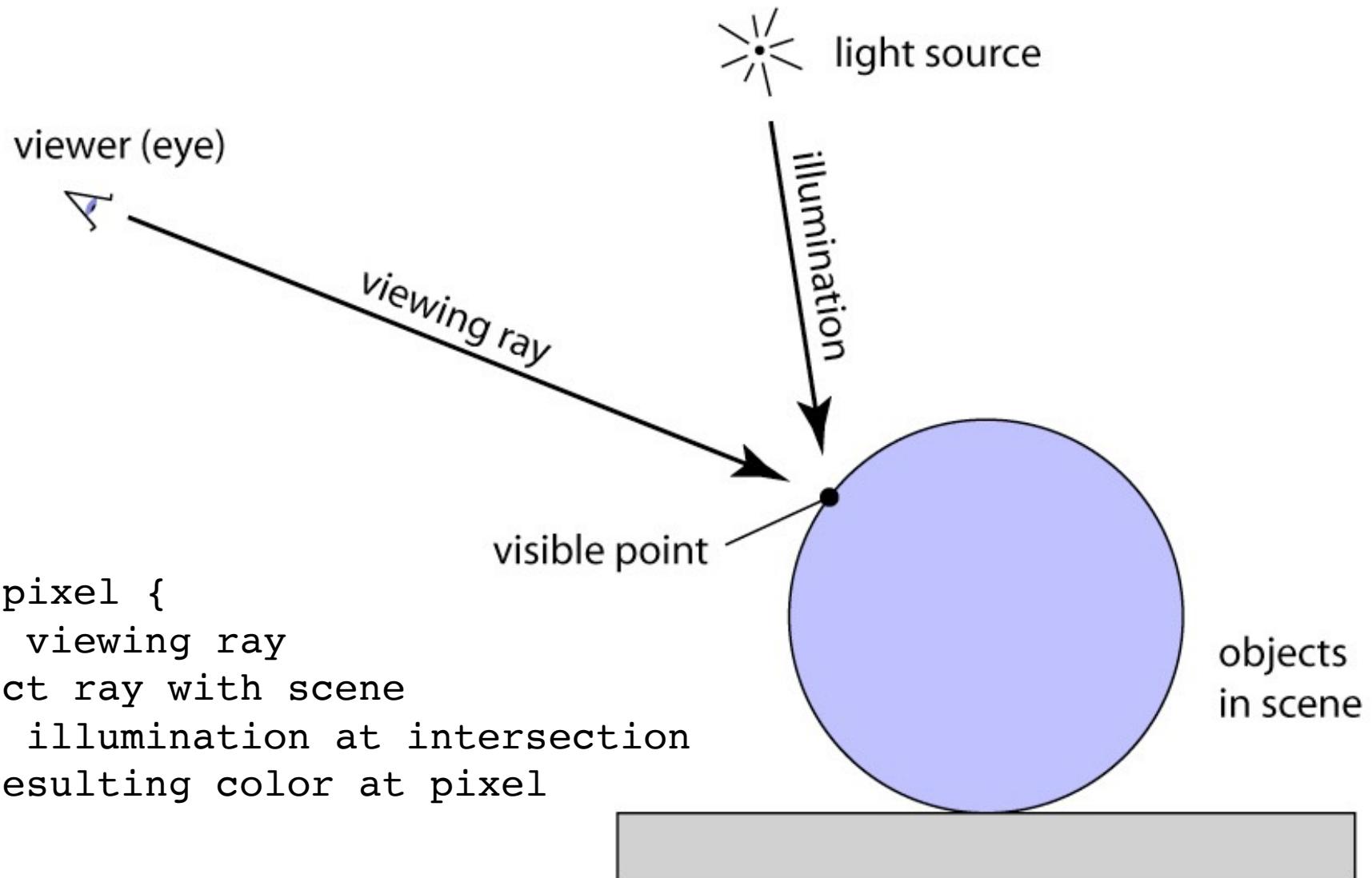
Using Paths of Light to Model Visibility



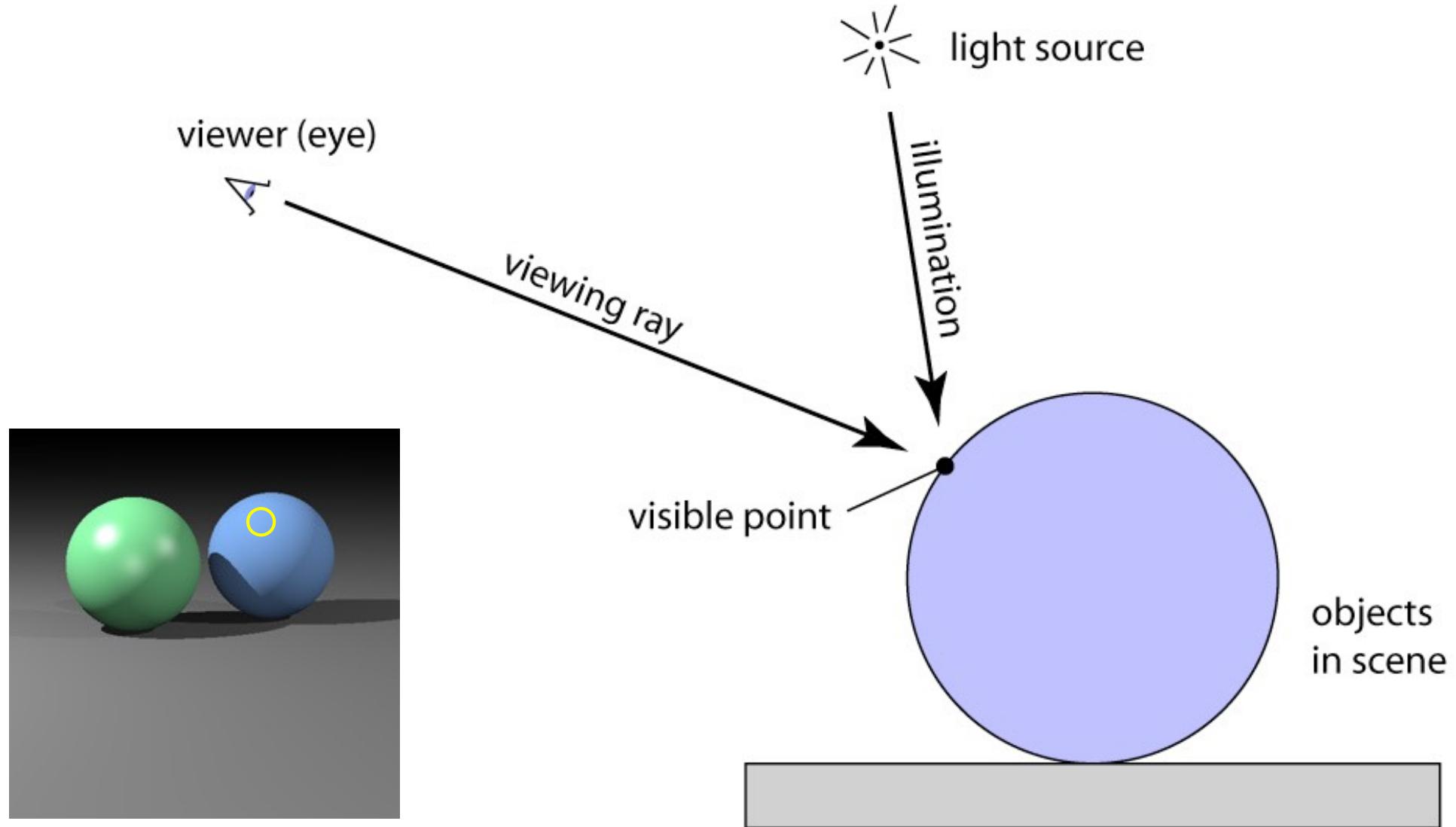
Forwarding vs Backward Tracing

- Idea: Trace rays from light source to image
 - This is slow!
- Better idea: Trace rays from image to light source

Ray Tracing Algorithm



Ray Tracing Algorithm



Cameras and Perspective

If illumination is uniform and directional-free (ambient light):

```
for each pixel {  
    compute viewing ray  
    intersect ray with scene  
    copy the color of the object at this point to this pixel.  
}
```

Commonly, we need slightly more involved

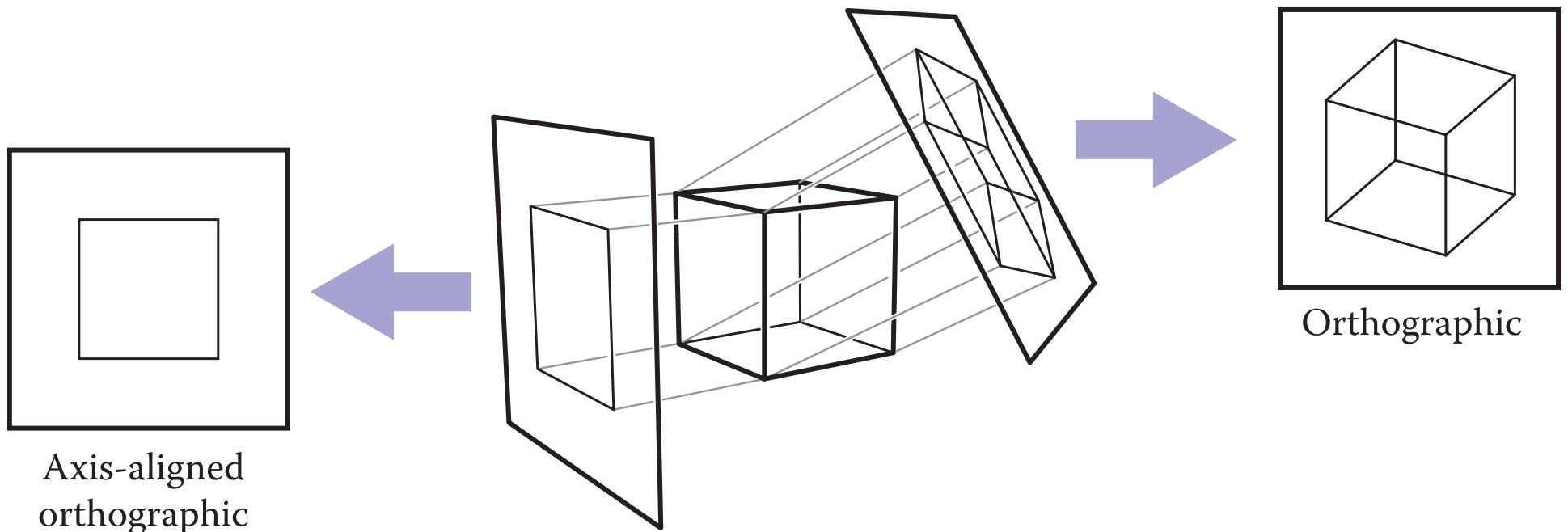
```
for each pixel {  
    compute viewing ray  
    intersect ray with scene  
    compute illumination at intersection  
    store resulting color at pixel  
}
```

Linear Perspective

- Standard approach is to project objects to an image plane so that straight lines in the scene stay straight lines on the image
- Two approaches:
 - Parallel projection: Results in **orthographic** views
 - Perspective projection: Results in **perspective** views

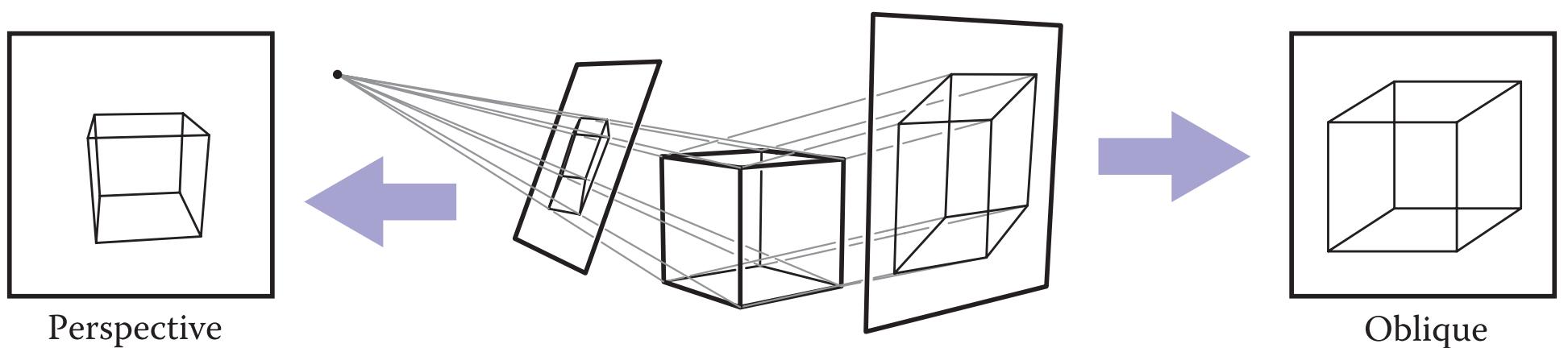
Orthographic Views

- Points in 3D are moved along parallel lines to the image plane.
- Resulting view determined solely by choice of projection direction and orientation/position of image plane



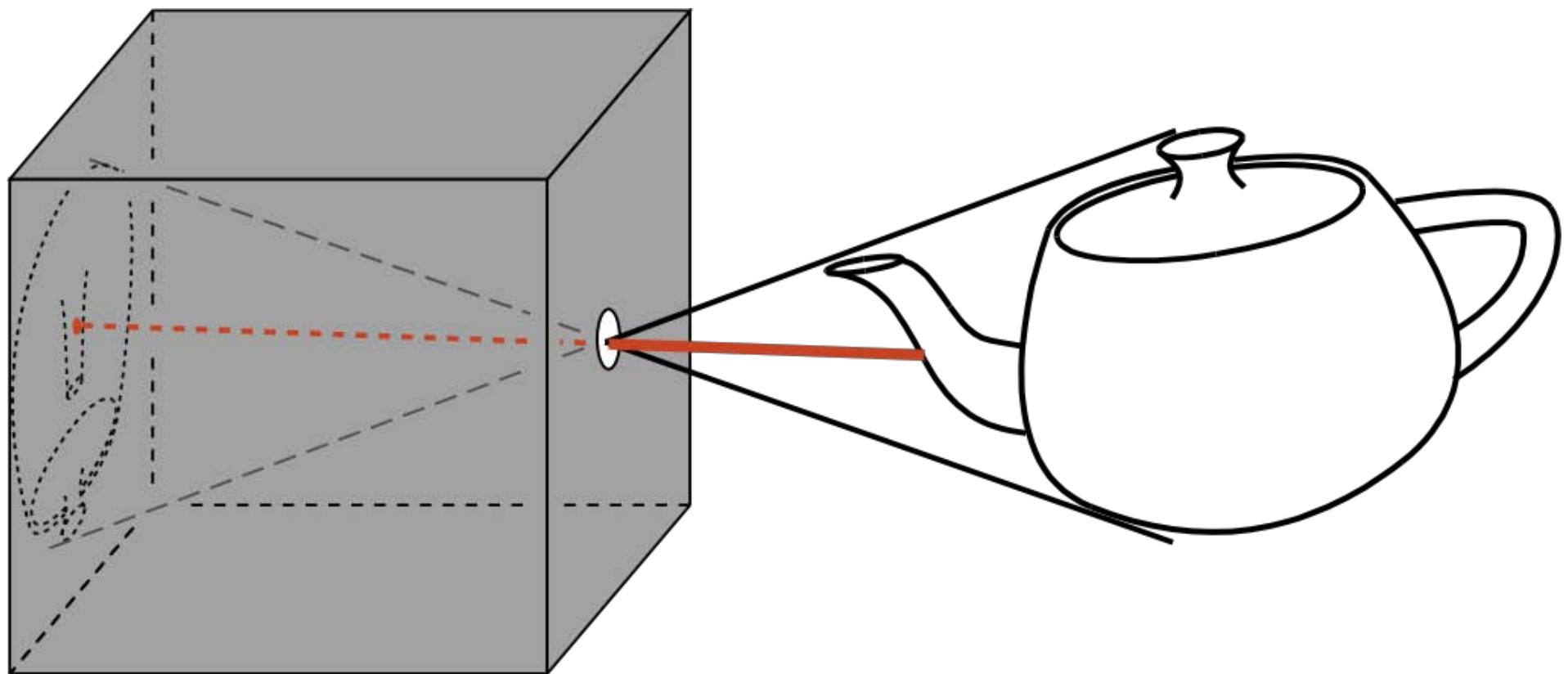
Perspective Views

- But, objects that are further away should look smaller!
- Instead, we can project objects through a single viewpoint and record where they hit the plane.



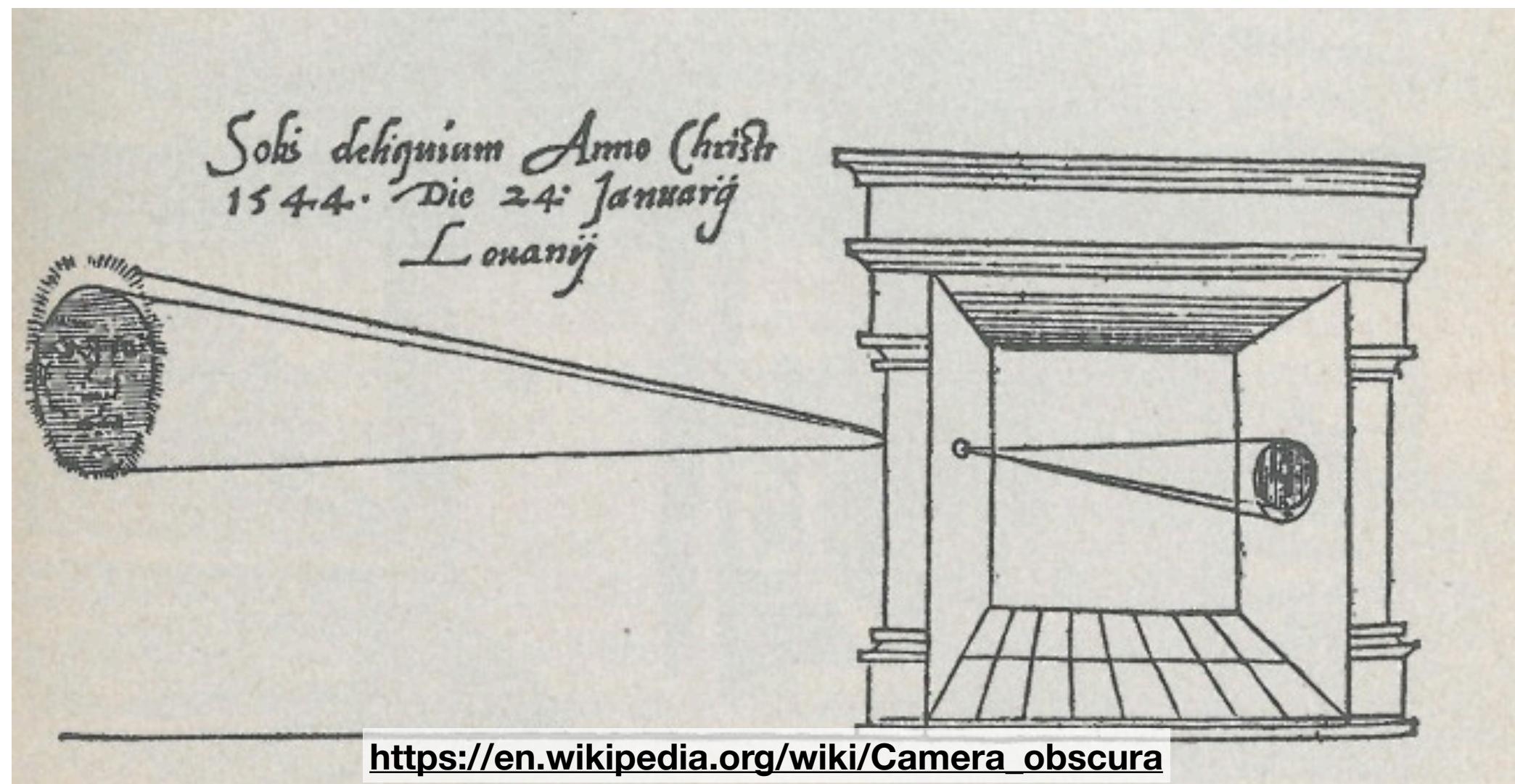
Pinhole Cameras

- Idea: Consider a box with a tiny hole. All light that passes through this hole will hit the opposite side
- Produced image inverts



Camera Obscura

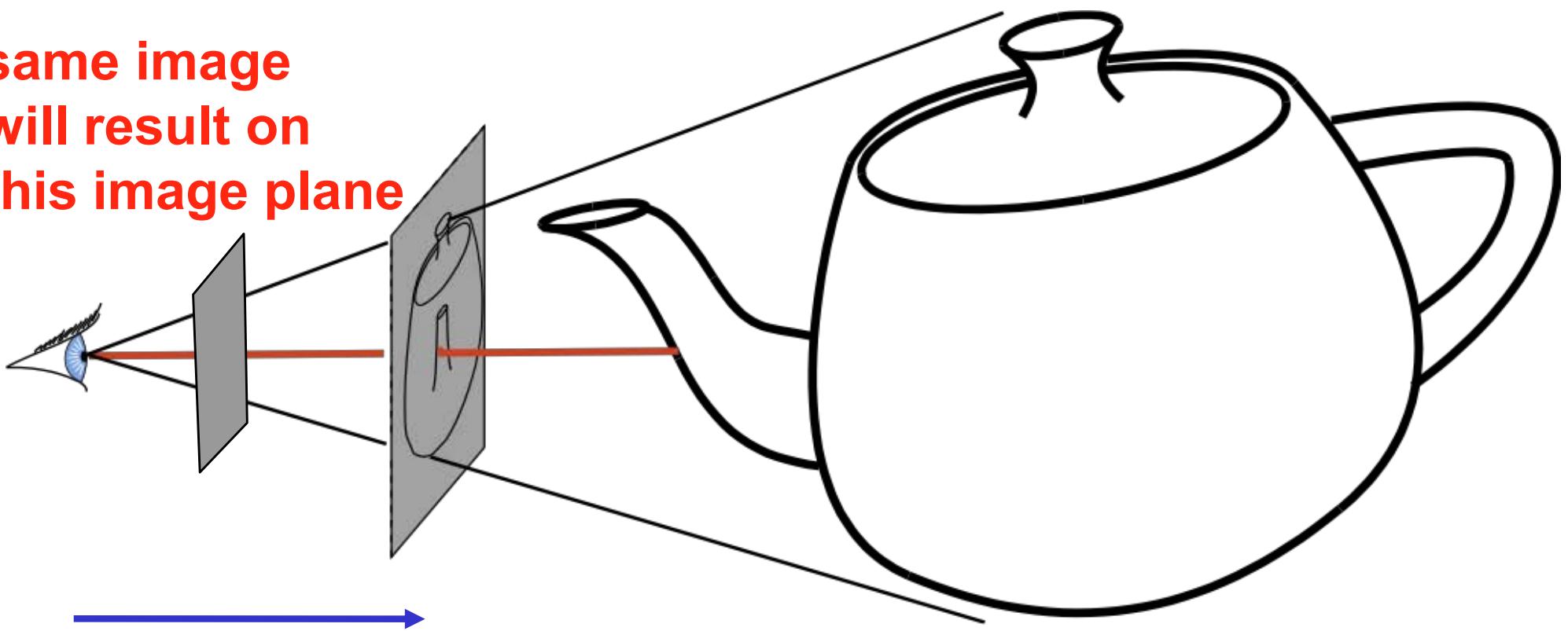
- Gemma Frisius, 16th century



Simplified Pinhole Cameras

- Instead, we can place the eye at the pinhole and consider the eye-image pyramid (sometimes called **view frustum**)

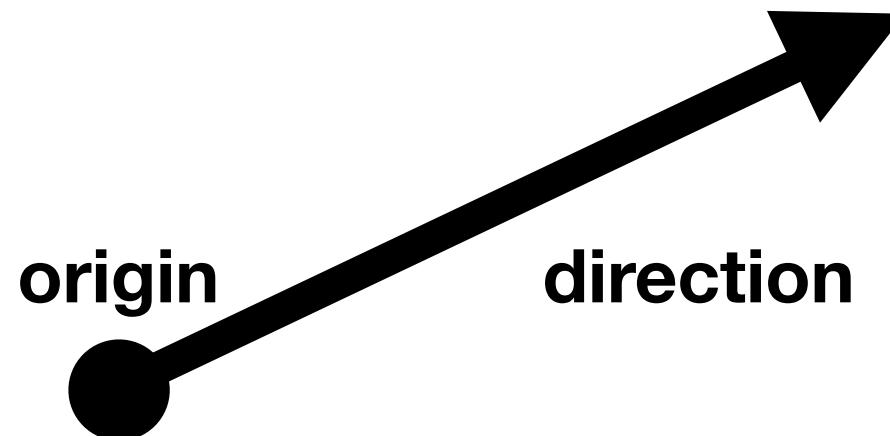
same image
will result on
this image plane



Defining Rays

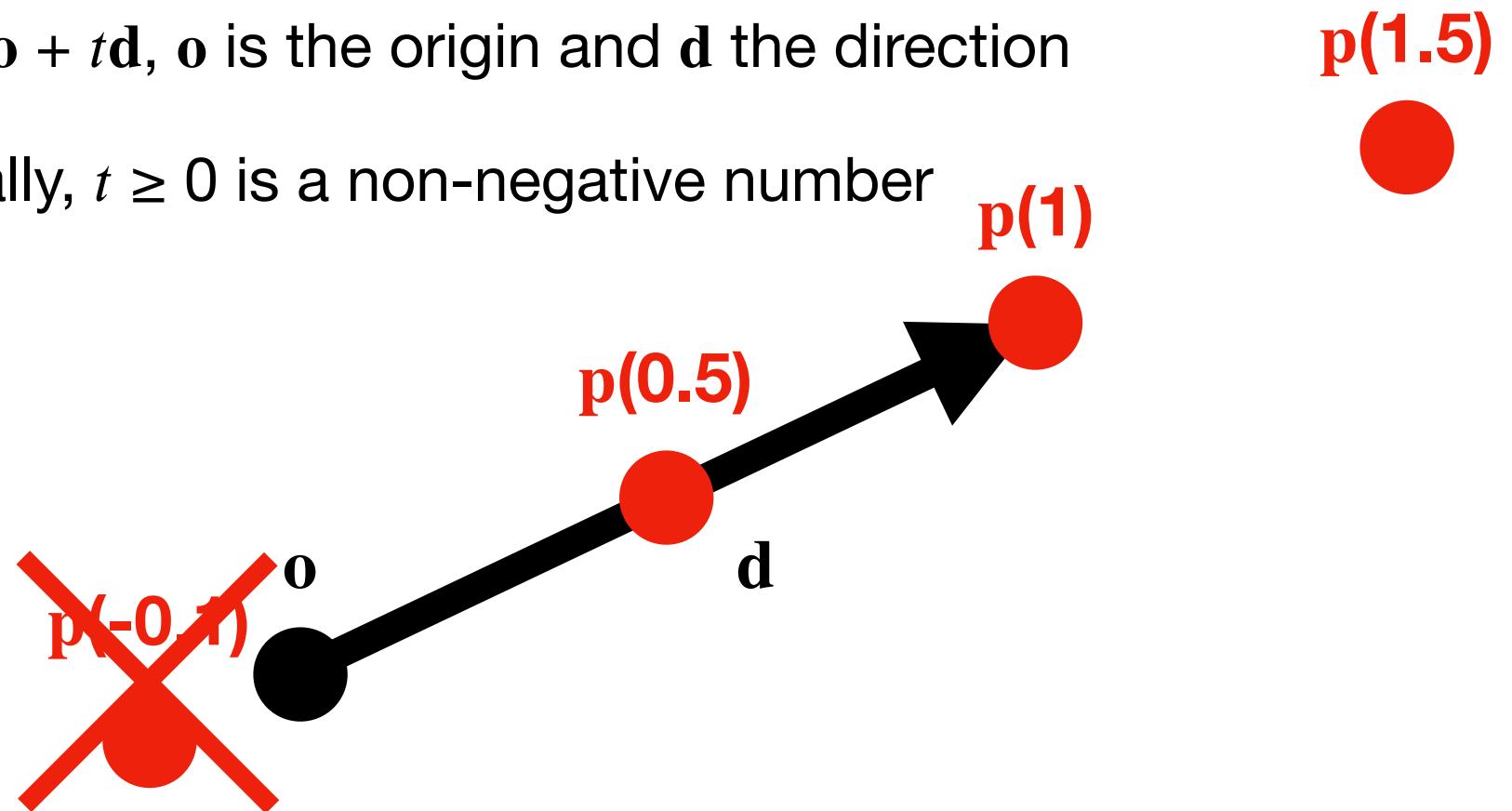
Mathematical Description of a Ray

- Two components:
 - An **origin**, or a position that the ray starts from
 - A **direction**, or a vector pointing in the direction the ray travels
 - Not necessarily unit length, but it's sometimes helpful to think of these as normalized

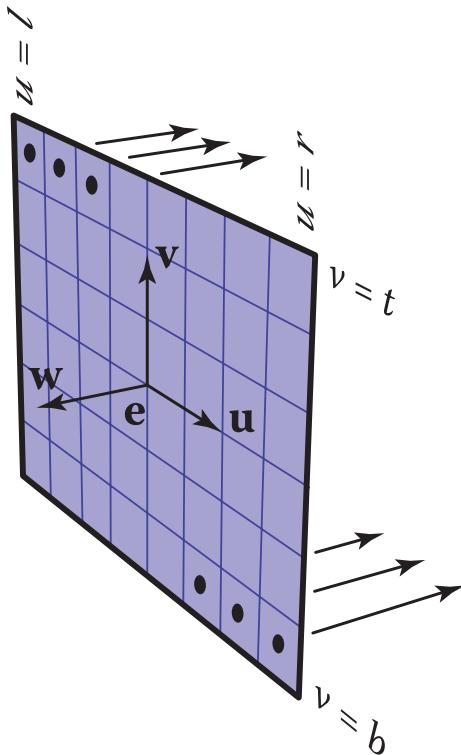


Mathematical Description of a Ray

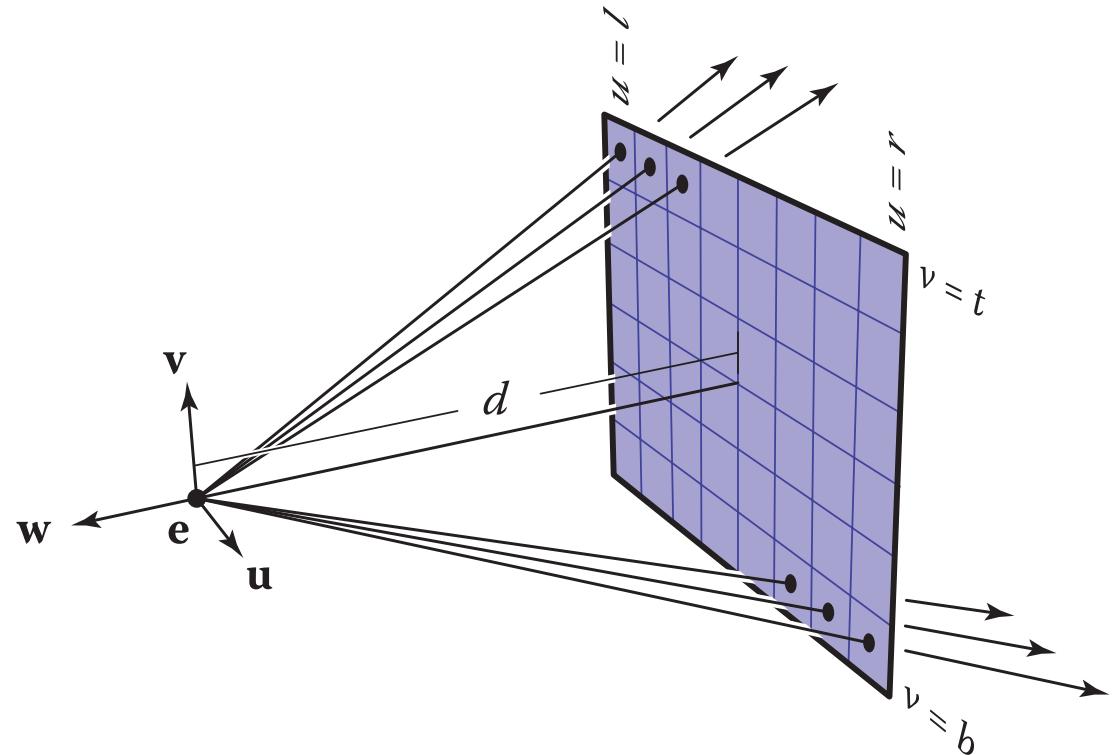
- Rays define a family of points, $\mathbf{p}(t)$, using a **parametric** definition
- $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$, \mathbf{o} is the origin and \mathbf{d} the direction
- Typically, $t \geq 0$ is a non-negative number



Orthographic vs. Perspective Rays



Parallel projection
same direction, different origins



Perspective projection
same origin, different directions

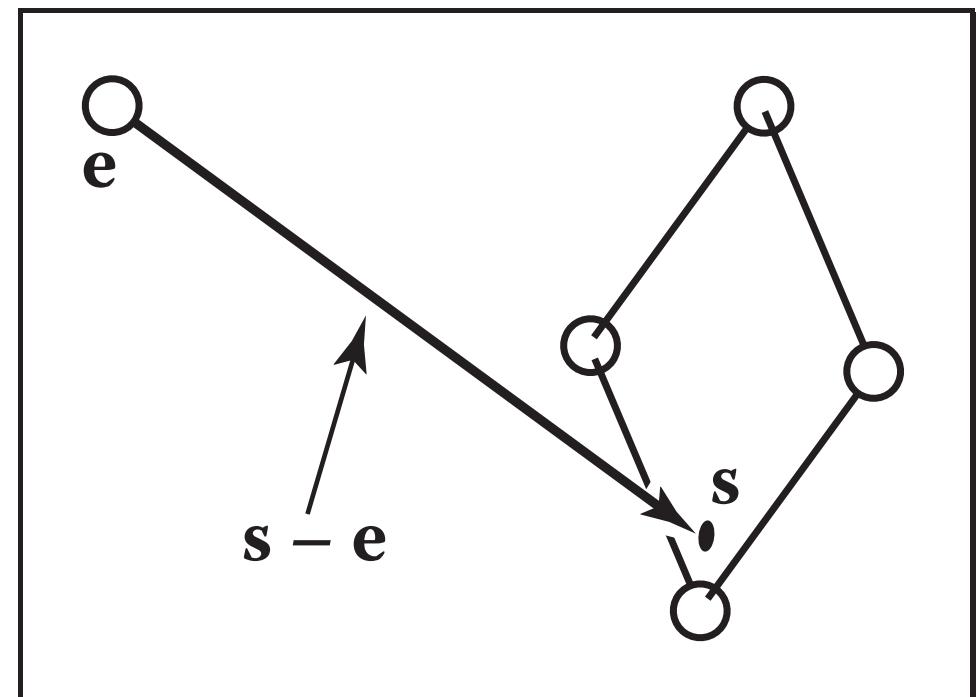
Defining o and d in Perspective Projection

- Given a viewpoint, e , and a position on the image plane, s

$$o = e$$

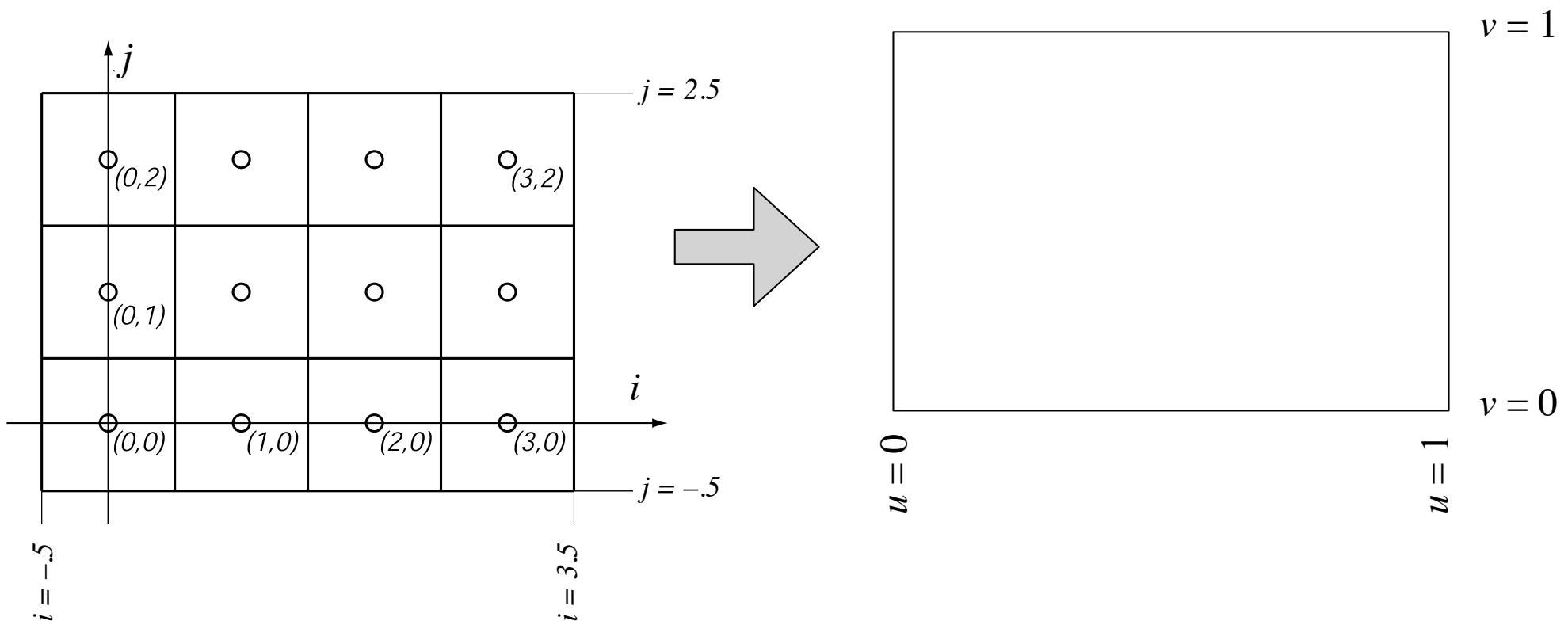
$$d = s - e$$

- And thus $p(t) = e + t(s - e)$

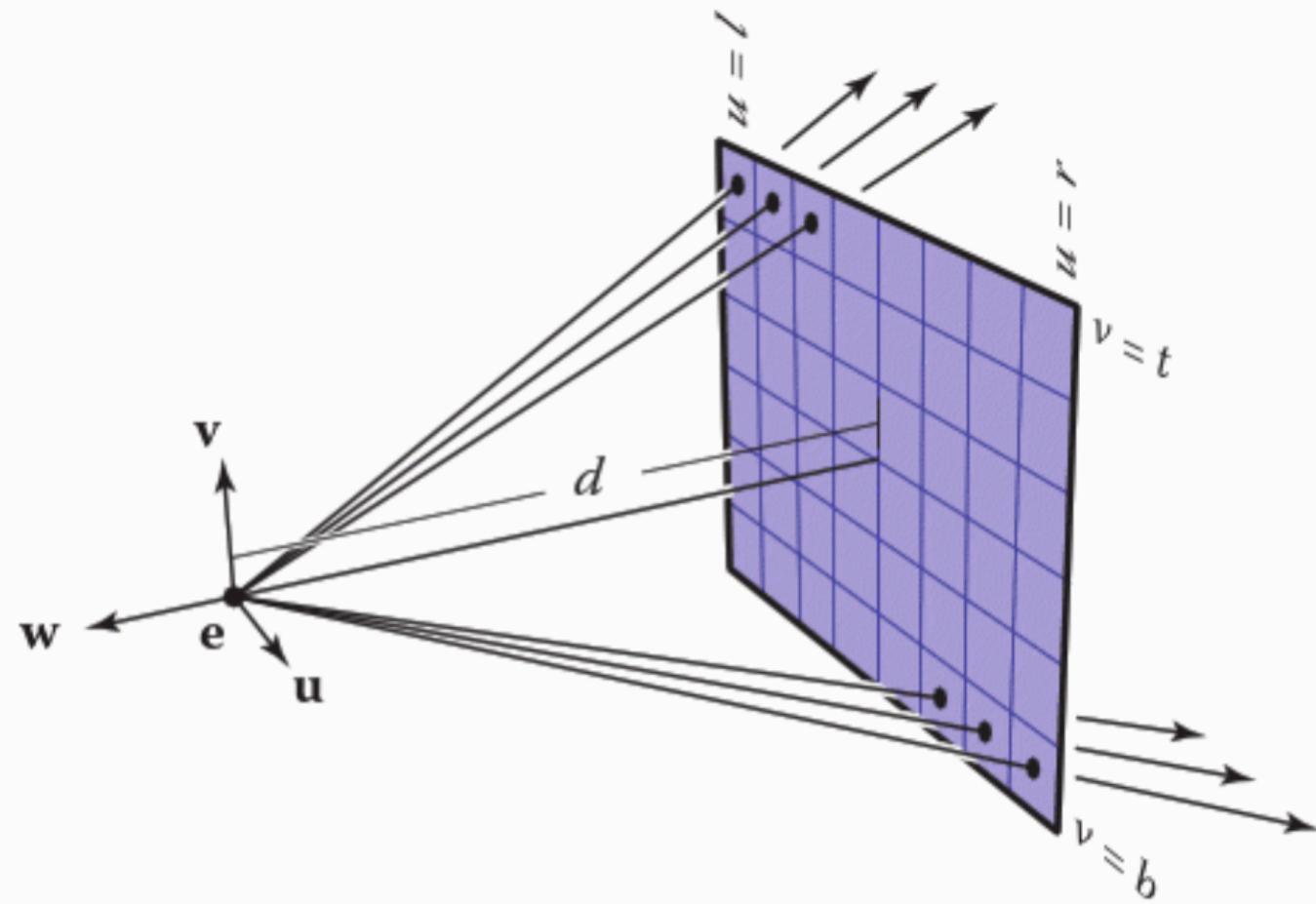


Pixel-to-Image Mapping

- Exactly where are pixels located? Must convert from pixel coordinates (i,j) to positions in 3D space (u,v,w)
- What should w be?



$$u = (i + 0.5)/n_x$$
$$v = (j + 0.5)/n_y$$



Camera Components

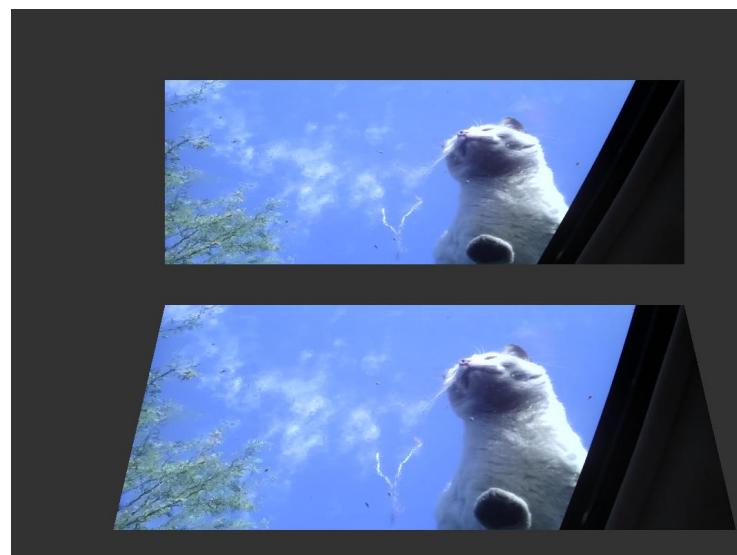
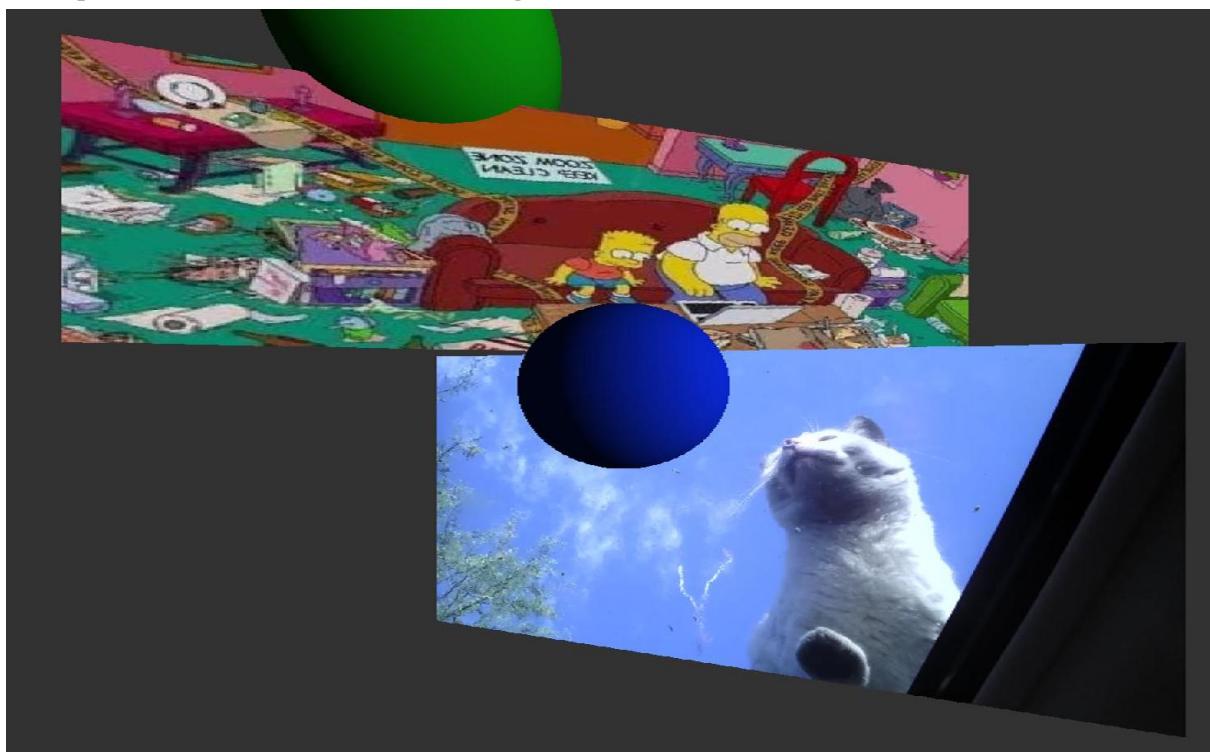
- Definition of an image plane
 - Both in terms of pixel resolution AND position in 3D space or more frequently in **field of view** and/or **distance**
- Viewpoint
- View direction
- Up vector

Assignment 3. Balls and Billboards

Input: JSON file describing locations of billboards and spheres.

Images placed on the billboards.

Output: scene showing what a viewer could see



In our trivia corner today: Line-Scan cameras



\$1300

Model No.	MV-CL020-40GC
Type	2k pixels line scan GigE camera
Pixel Size	7 µm
Horizontal Frequency	1 Hz to 51 kHz



**Takes 51000 images /second
But each image is 1D**

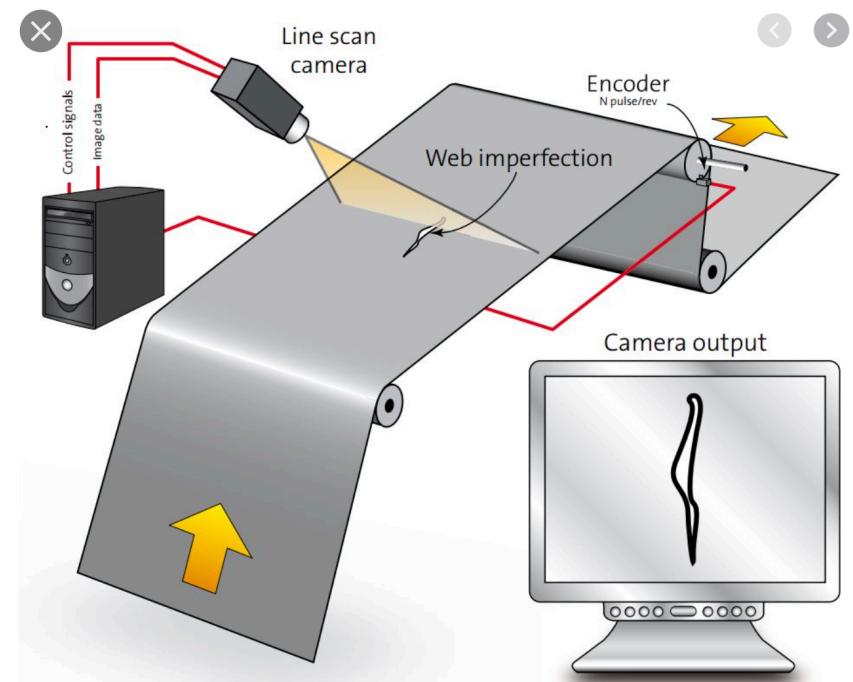
Other line scan cameras could reach 12000 pixels per scan

Why are they used

- Road monitoring



- **Better handling noise,
optical distortions**
Better dynamic range

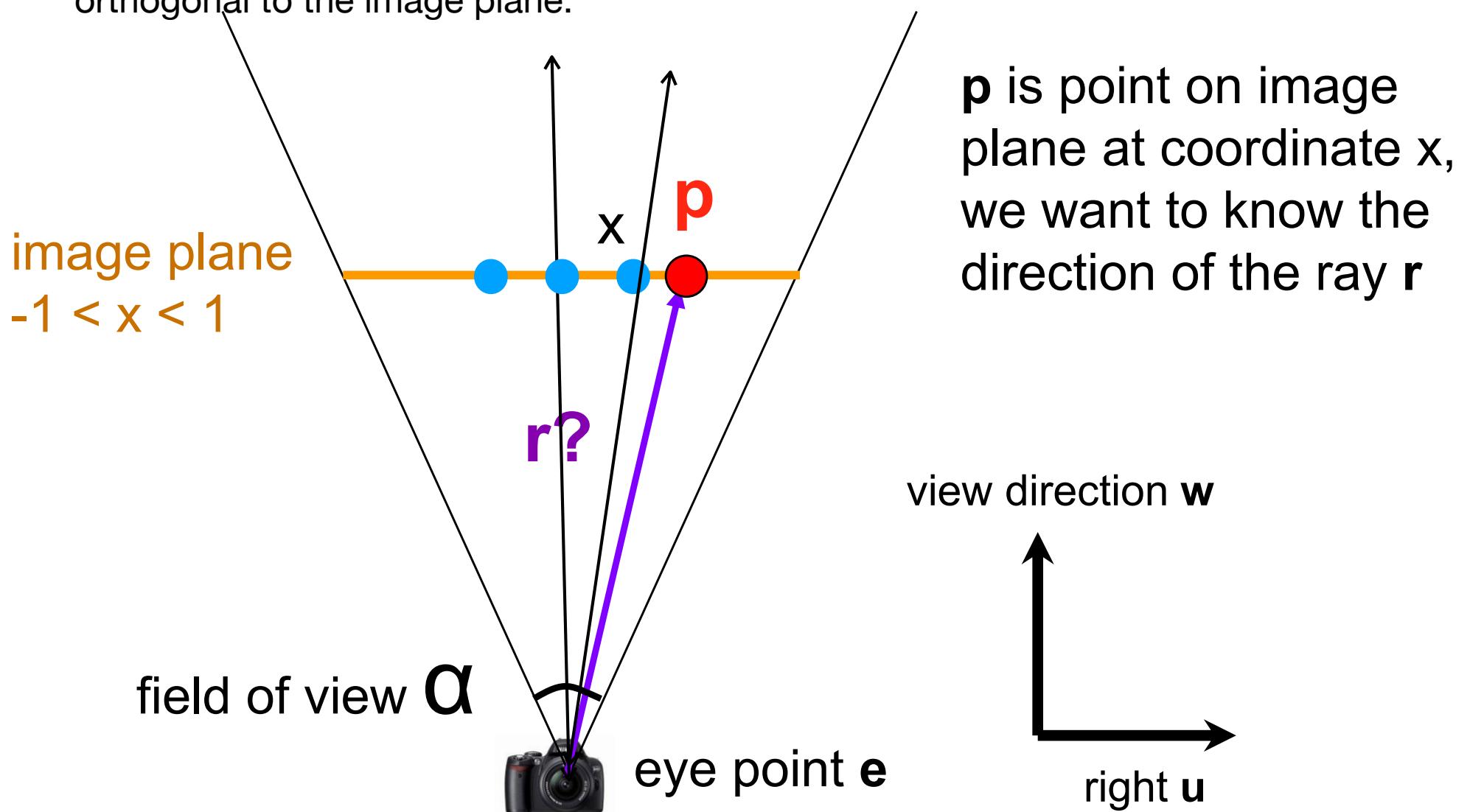


Having the line-scan cameras as a excuse,

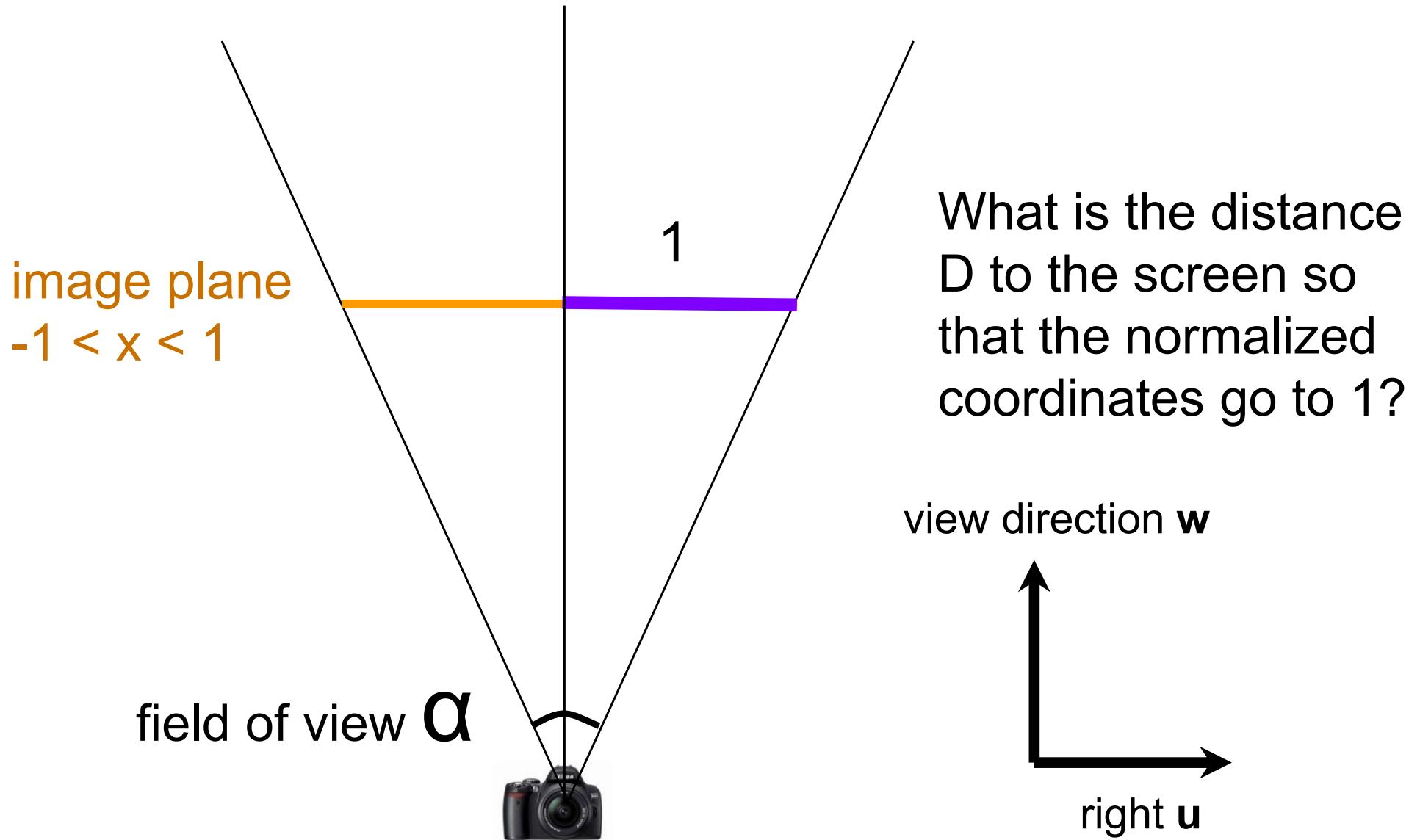
Lets study ray-tracing in the 2D world

Ray Generation in 2D

- The image plane (should actually call it “image line”)
- Our algorithm will assign a color to each pixel, by tracing a ray through the pixel and check the color of the object it hits
- User determined the location e of the camera, the direction w through, and orthogonal to the image plane.

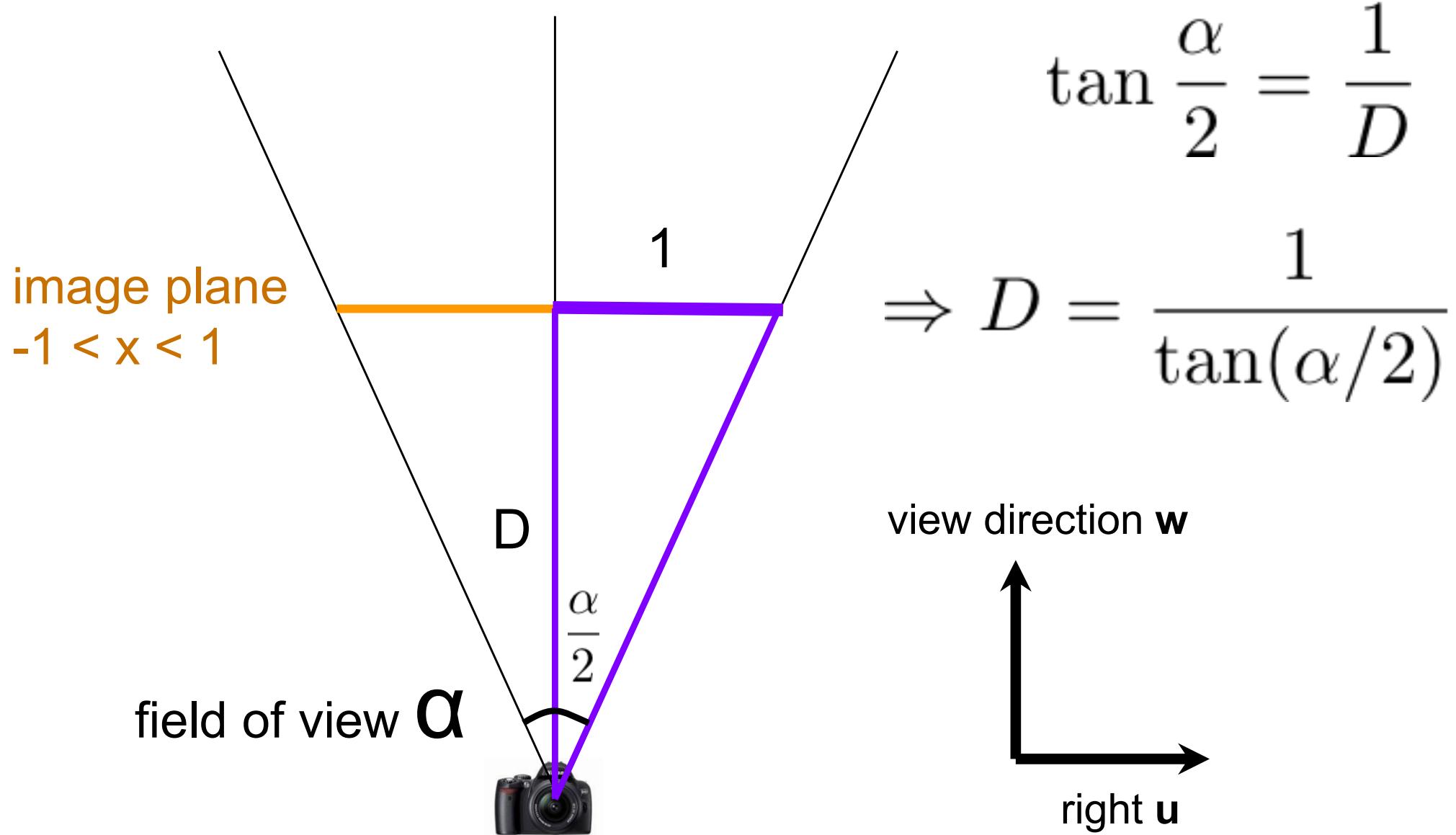


Ray Generation in 2D

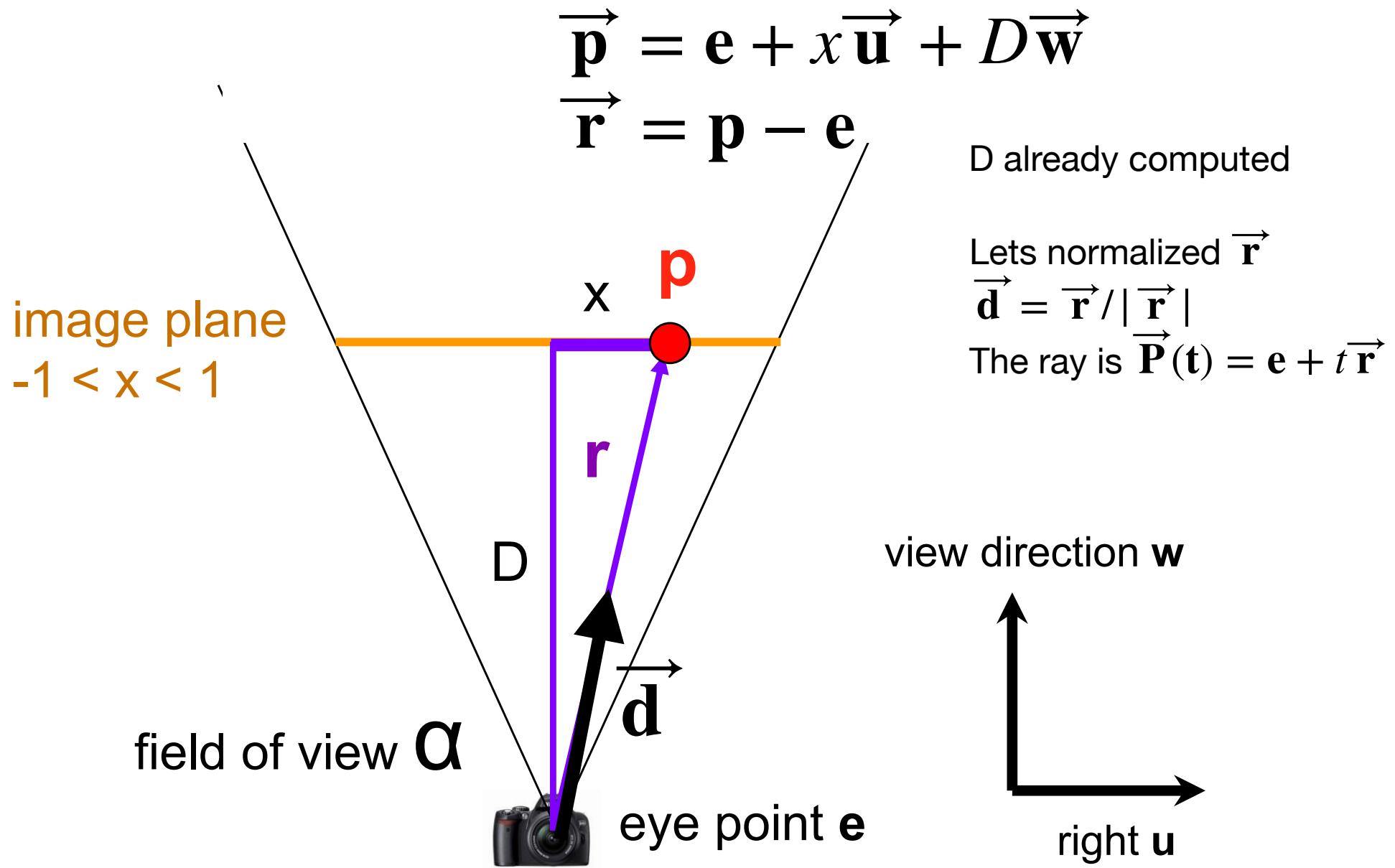


The user specifies the field of view angle `fov_angle`.

We need to calculate the distance D to the image plane

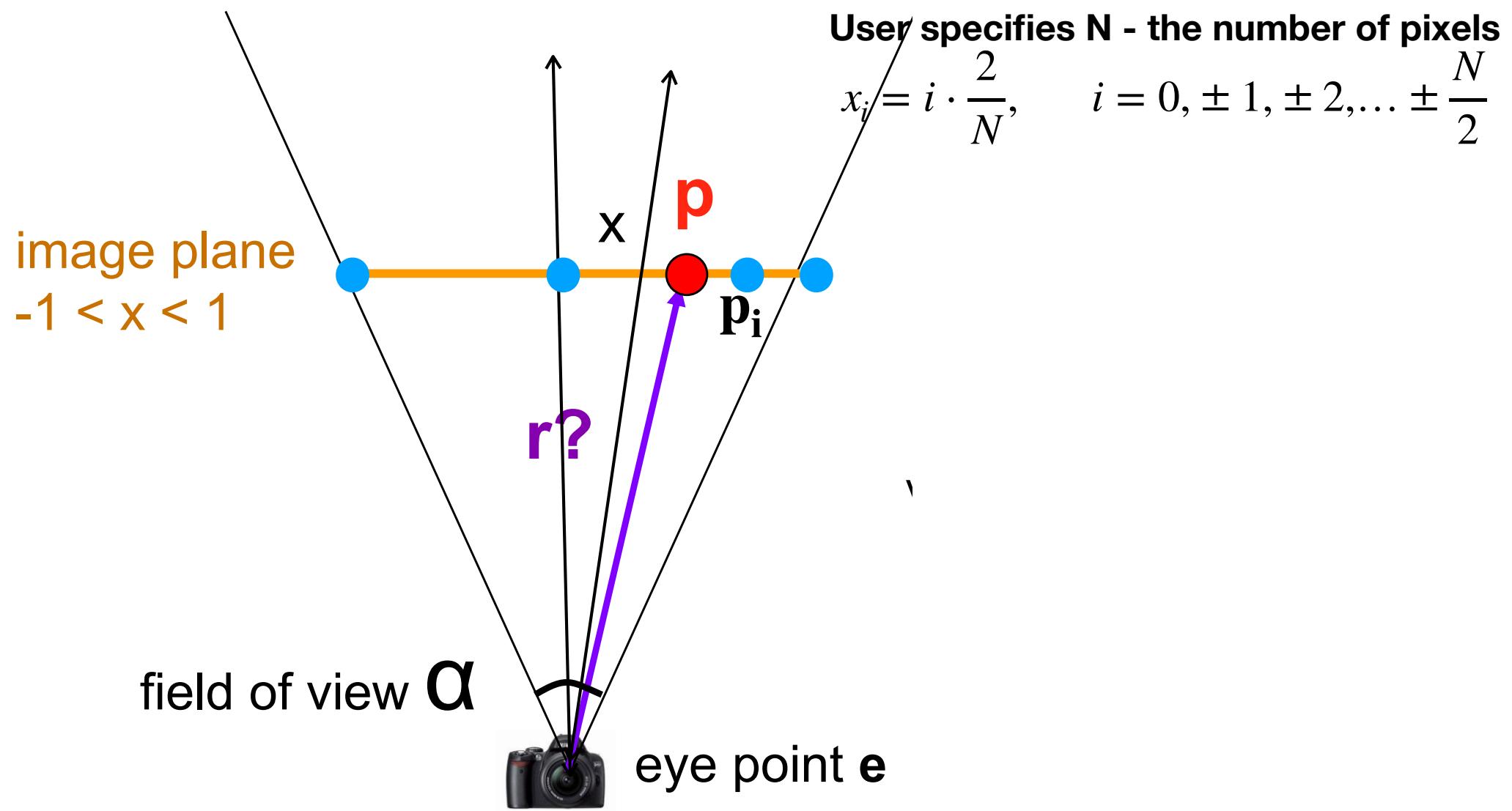


Ray Generation in 2D



Calculating all the rays

$$\vec{p}_i = \mathbf{e} + x_i \vec{\mathbf{u}} + D \vec{\mathbf{w}}$$

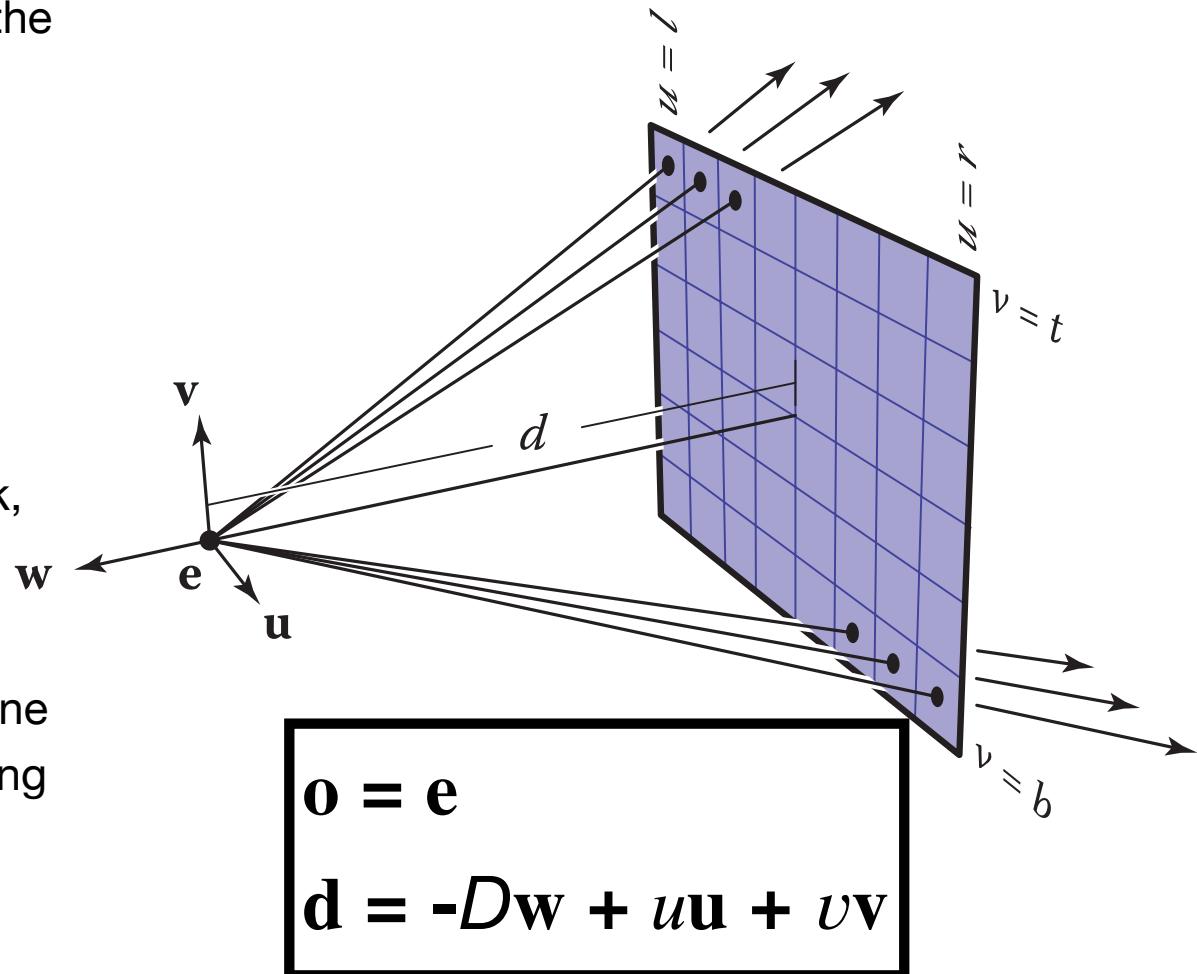


From 2D to 3D

- Moving from 2D to 3D is essentially the same thing once you can define the positions of pixels in uvw space

from now on, we use d to denote distance to image plane

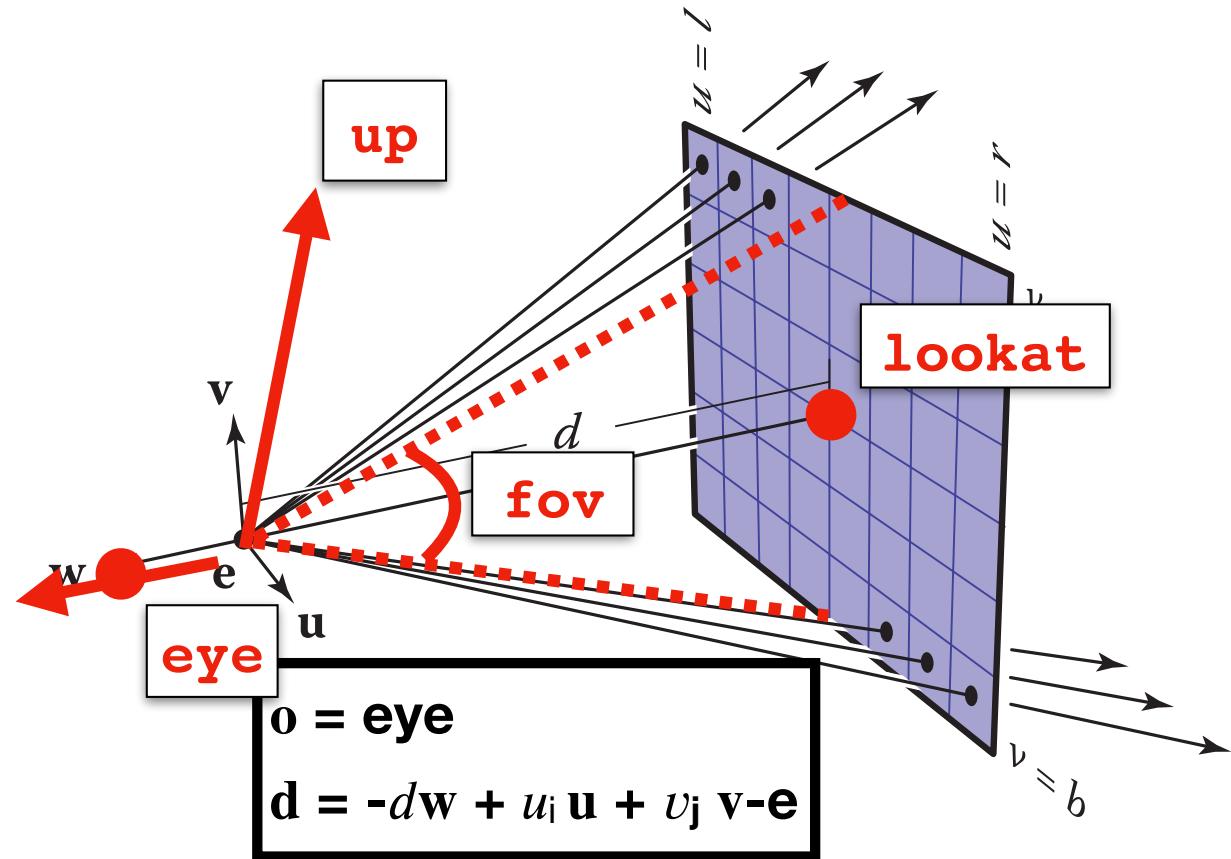
- Following the convention of the book, w is the negated viewpoint vector
- The up vector, v , can be used to define a local coordinate space by computing u



In the Assignment

- An eye position
- A position to lookat, which is centered in the image
 - **w** can be defined use eye and lookat as well as d
- An up vector, not necessarily **v**! (but the vectors v up w in the same plane)
- A `fov_angle` for the **vertical** FOV
- The FOV defines the **height** of the image plane in world space
- Each pixel is a square, but number of pixels rows vs columns might be different
- You can then use this to compute the **width** of the image plane in world space using the aspect ratio (rows/columns) of the image
- Using the number of rows/columns you can then sample u, v

We use a terminology that is very common.
Not always most intuitive



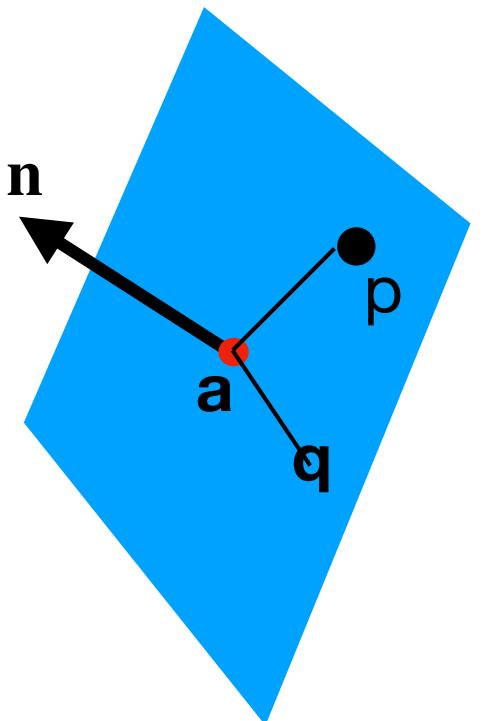
$$u_i = i \cdot 2/N_{cols}, \quad i = 0, \pm 1, \pm 2, \dots, \pm N_{cols}/2$$

Intersecting Objects

```
for each pixel {  
    compute viewing ray  
    intersect ray with scene  
    compute illumination at intersection  
    store resulting color at pixel  
}
```

Defining a Plane

- Let h be a plane with normal \mathbf{n} , and containing a point \mathbf{a} . Let p be some other point. Then p is on this plane if and only if (iff) $\mathbf{p} \cdot \mathbf{n} = \mathbf{a} \cdot \mathbf{n}$
- Proof. Consider the segment $p-a$. p is on the plane iff $p-a$ is orthogonal to n . Using the property of dot product $(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = |\mathbf{p} - \mathbf{a}| |\mathbf{n}| \cos \alpha$
- Here α is the angle between them. Now $\cos(90)=0$. So if p on this plane then $\mathbf{p} \cdot \mathbf{n} = \mathbf{a} \cdot \mathbf{n}$ implying
- If $\mathbf{p} \cdot \mathbf{n} > \mathbf{a} \cdot \mathbf{n}$ then \mathbf{p} lives on the “front” side of the plane (in the direction pointed to by the normal)
- $\mathbf{p} \cdot \mathbf{n} - \mathbf{a} \cdot \mathbf{n} < 0$ means that \mathbf{p} lives on the “back” side.
- Sometimes used as $f(\mathbf{p})=0$ iff “ \mathbf{p} on the plane”. So the function $f(\mathbf{p})$ is $f(\mathbf{p})=(\mathbf{p}-\mathbf{a}) \cdot \mathbf{n}$
- If we have 3 points a,p,q all on the plane, then we can compute a normal $\mathbf{n} = (\mathbf{p} - \mathbf{a}) \times (\mathbf{q} - \mathbf{a})$. (cross product).
- Warning: The term “normal” does not mean that it was normalized.



Ray-Plane Intersection

- A ray $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$
- Two conditions must be satisfied:
 - Must be on a ray: $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$
 - Must be on the plane: $f(\mathbf{p}) = (\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0$
- Can substitute the equations and solve for t in $f(\mathbf{p}(t))$:
$$(\mathbf{o} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} = 0$$
- This means that $t_{hit} = ((\mathbf{a} - \mathbf{o}) \cdot \mathbf{n}) / (\mathbf{d} \cdot \mathbf{n})$. The intersection point is $\mathbf{o} + t_{hit}\mathbf{d}$

Defining a Sphere

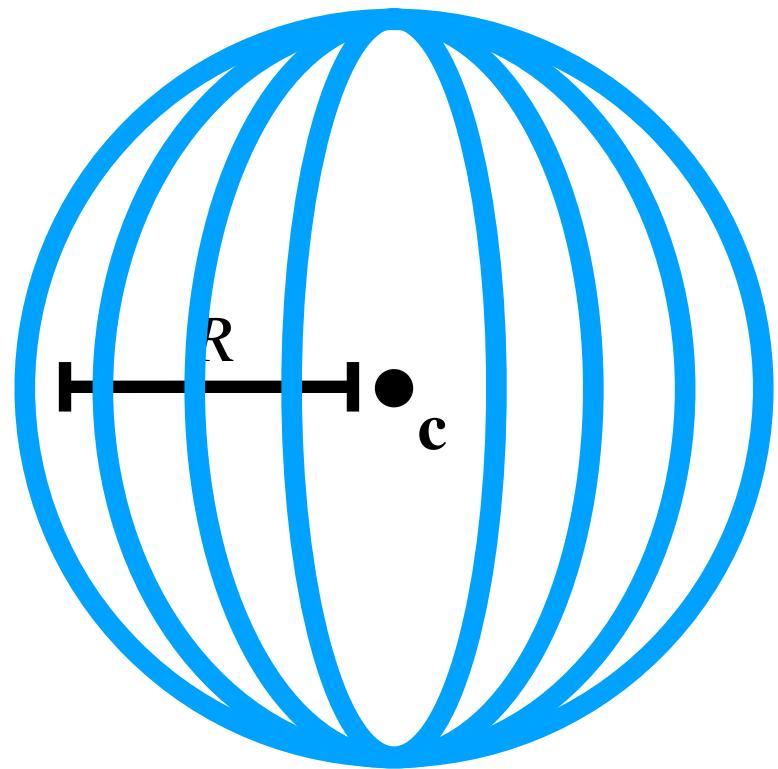
- For every vector v , the length of v is $|$

$$|v| = \sqrt{v \cdot v} = \sqrt{x^2 + y^2 + z^2}$$

- We can define a sphere of radius R , centered at position c , using the implicit form $|p-c|=R$, implying

$$f(p) = (p - c) \cdot (p - c) - R^2 = 0$$

- Any point p that satisfies the above lives on the sphere



Ray-Sphere Intersection

- Two conditions must be satisfied:
 - Must be on a ray: $\mathbf{p}(t) = \mathbf{o} + t\mathbf{r}$
 - Must be on a sphere: $f(\mathbf{p}) = (\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0$
- Can substitute the equations and solve for t in $f(\mathbf{p}(t))$:

$$(\mathbf{o} + t\mathbf{r} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{r} - \mathbf{c}) - R^2 = 0$$

- Solving for t is a quadratic equation

Ray-Sphere Intersection

- Solve $(\mathbf{o} + t\mathbf{r} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{r} - \mathbf{c}) - R^2 = 0$ for t :
- Rearrange terms:

$$(\mathbf{d} \cdot \mathbf{d})t^2 + (2\mathbf{d} \cdot (\mathbf{o} - \mathbf{c}))t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2 = 0$$

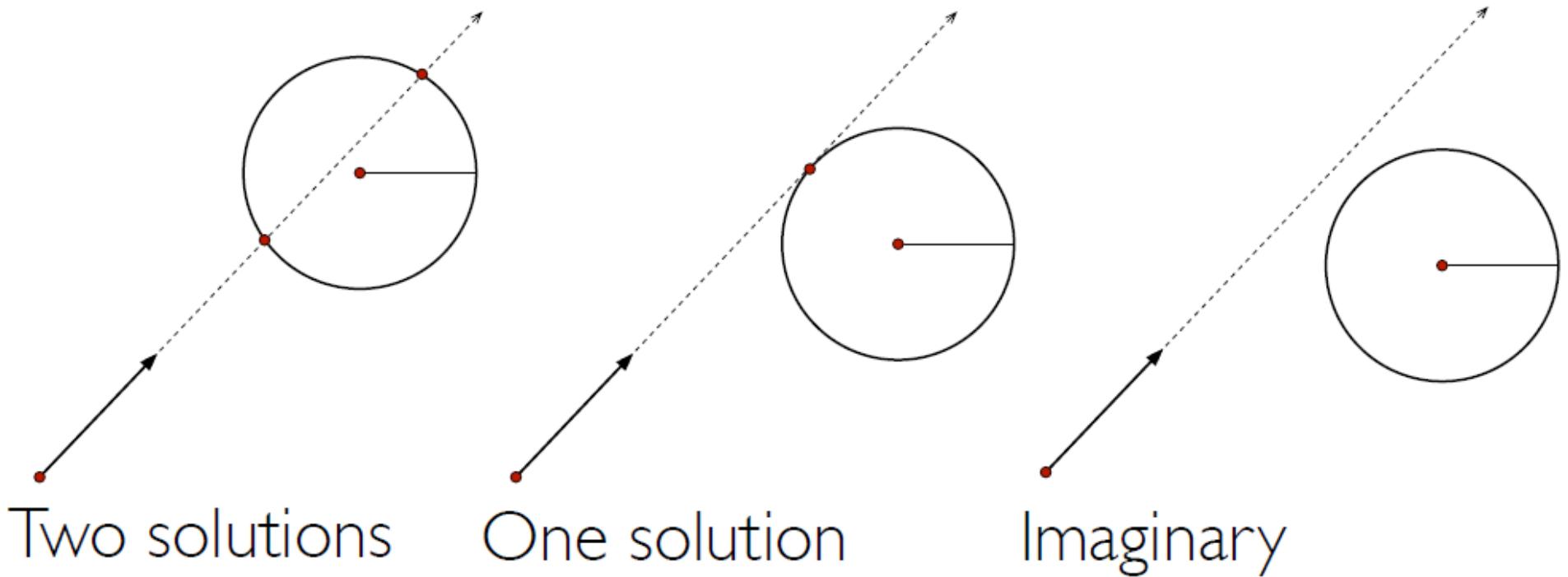
- Solve the quadratic equation $At^2 + Bt + C = 0$ where
 - $A = (\mathbf{r} \cdot \mathbf{r})$
 - $B = 2 * \mathbf{d} \cdot (\mathbf{o} - \mathbf{c})$
 - $C = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2$

Discriminant, $D = B^2 - 4 * A * C$
Solutions must satisfy:

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

Ray-Sphere Intersection

- Number of intersections dictated by the discriminant
- In the case of two solutions, prefer the one with lower t



Revisiting dot product: projections

Let \mathbf{u}, \mathbf{v} be orthonormal vectors (**orthogonal and unit length**). \mathbf{r} is another vector

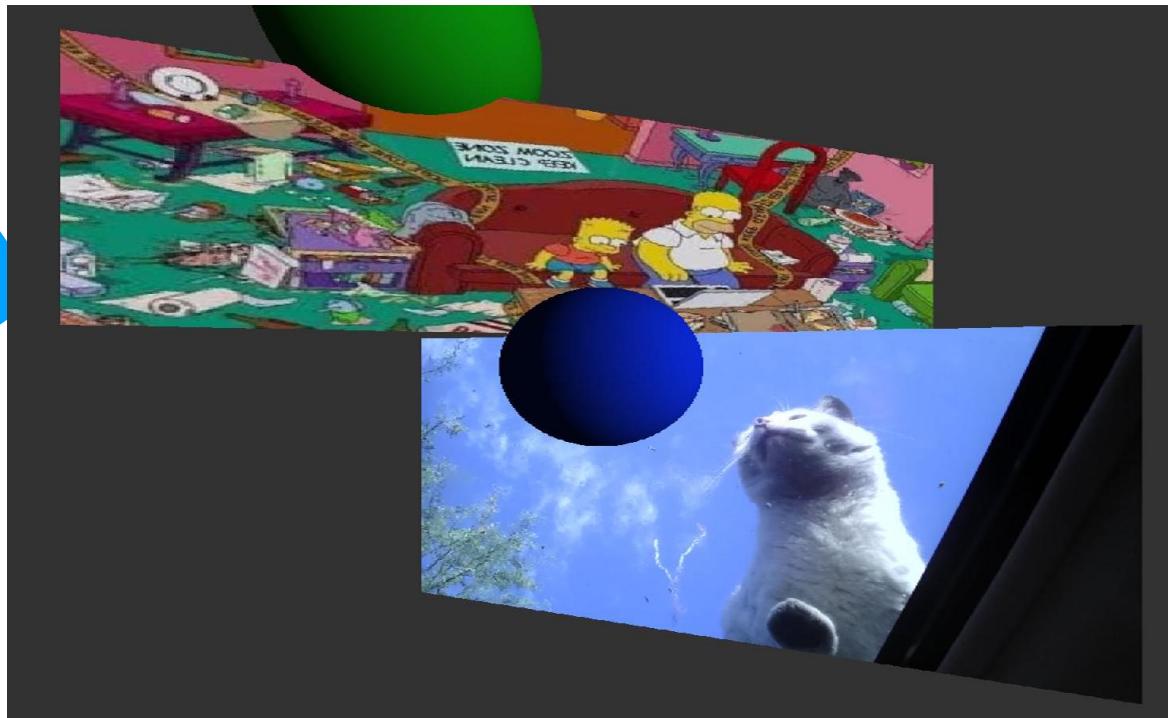
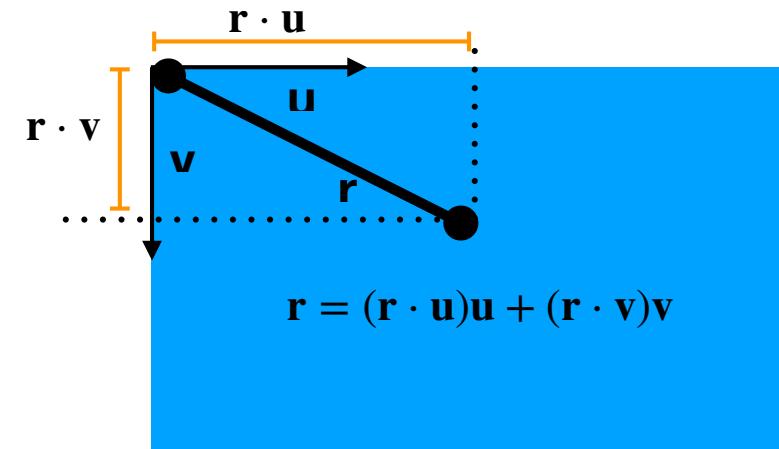
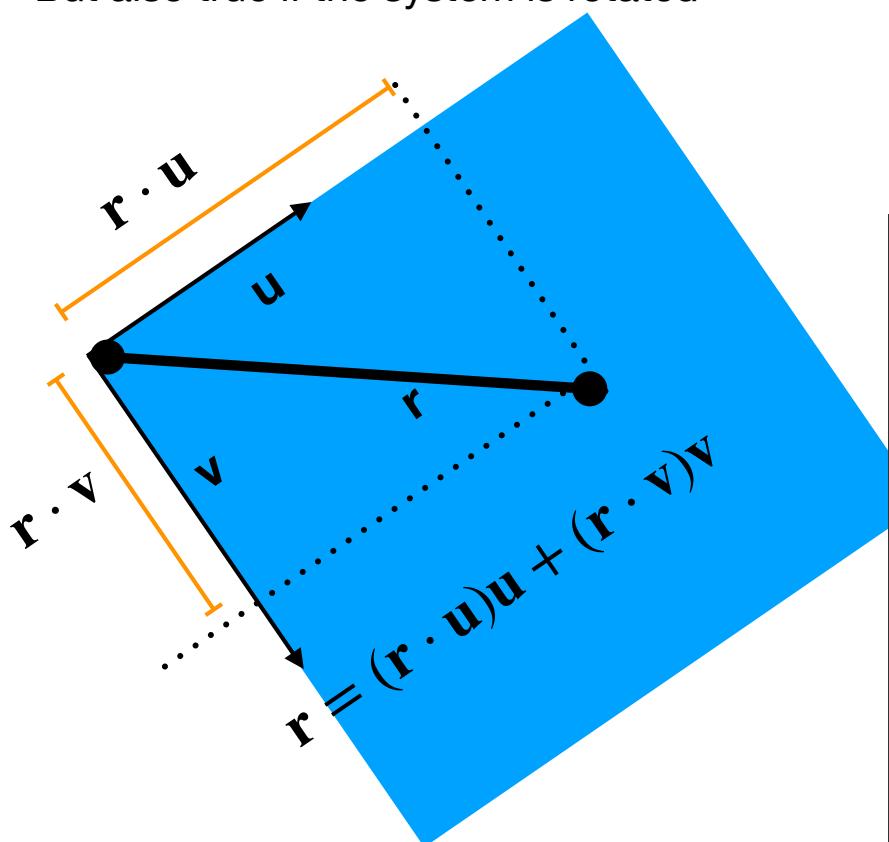
$$\mathbf{r} \cdot \mathbf{u} = |\mathbf{r}| |\mathbf{u}| \cos \alpha = |\mathbf{r}| \cos \alpha$$

is the projection of \mathbf{r} on the direction of \mathbf{u}

the length of the “shadow” that \mathbf{r} cast on the line containing \mathbf{u}

Sounds obvious when the coordinate system is xyz

But also true if the system is rotated



More Examples Of What Can Be Done With Ray Tracing

- <https://developer.nvidia.com/optix>
- <https://embree.github.io/gallery.html>

Next Lecture Required

Reading

- FOCG, Ch. 4.5-4.9

Assignment 3. Balls and Billboards

Input: JSON file describing locations of billboards and spheres.

Images placed on the billboards.

Output: scene showing what a viewer could see

