CSC 433/533
Computer Graphics

Alon Efrat
Credit: Joshua Levine

Vector Math + Coding

Today's Agenda

- Reminders:
  - A02 questions?
- Goals for today:
  - Introduce some mathematics and connect it to code

Vectors

What is a Vector?

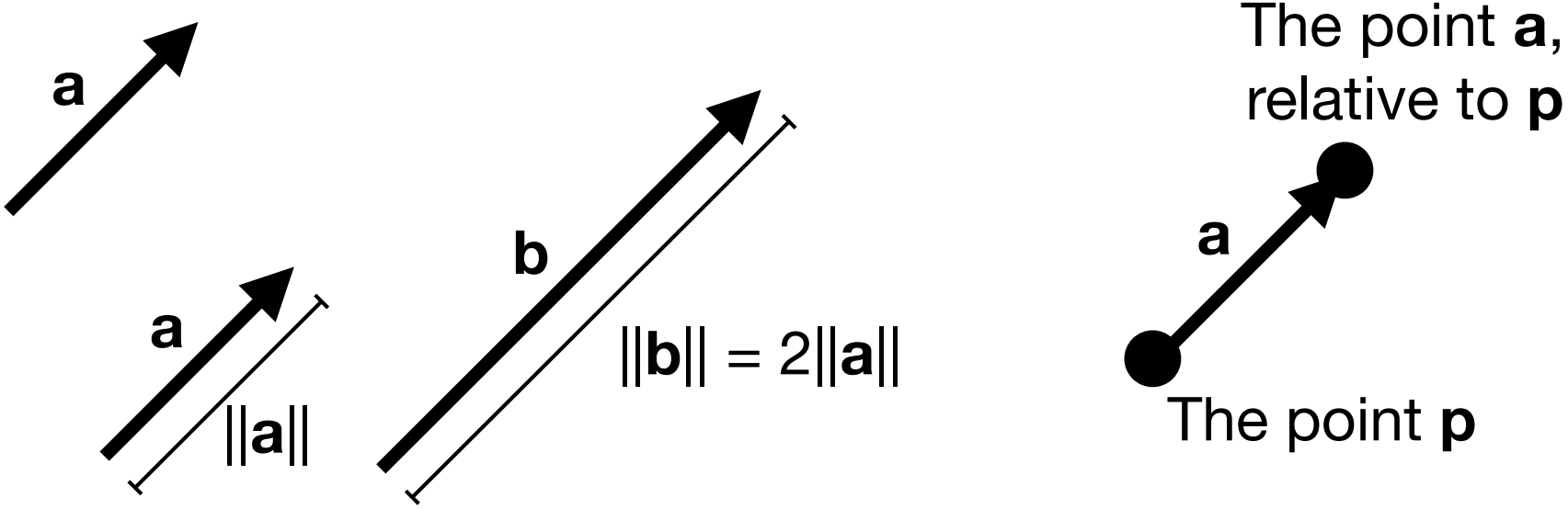
- A **vector** describes a length and a direction
- A vector is also a tuple of numbers
  - But, it often makes more sense to think in terms of the length/direction than the coordinates/numbers
  - And, especially in code, we want to manipulate vectors as objects and abstract the low-level operations
  - Compare with a **scalar**, or just a single number

Properties

- Two vectors, **a** and **b**, are the same (written **a** = **b**) if they have the same length and direction.
- A vector's **length** is denoted with || ||,
  - e.g. the length of **a** is ||**a**||
- A **unit vector** has length one
- The **zero vector** has length zero, and undefined direction

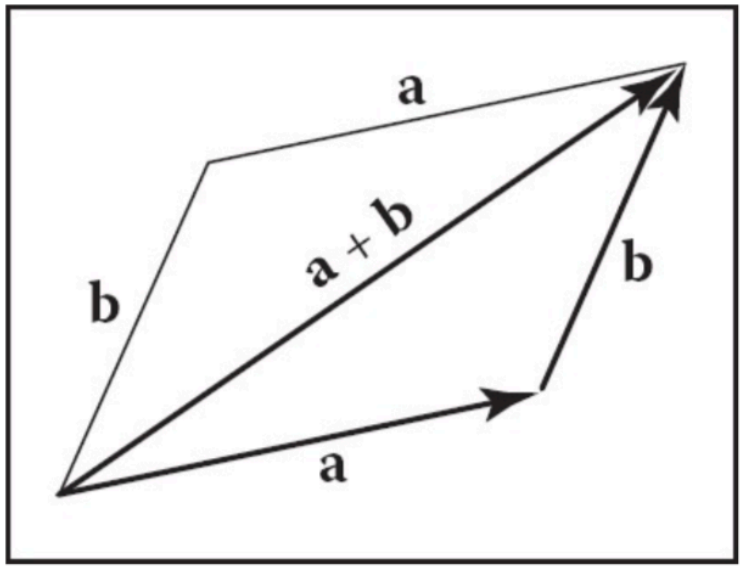
Vectors in Pictures

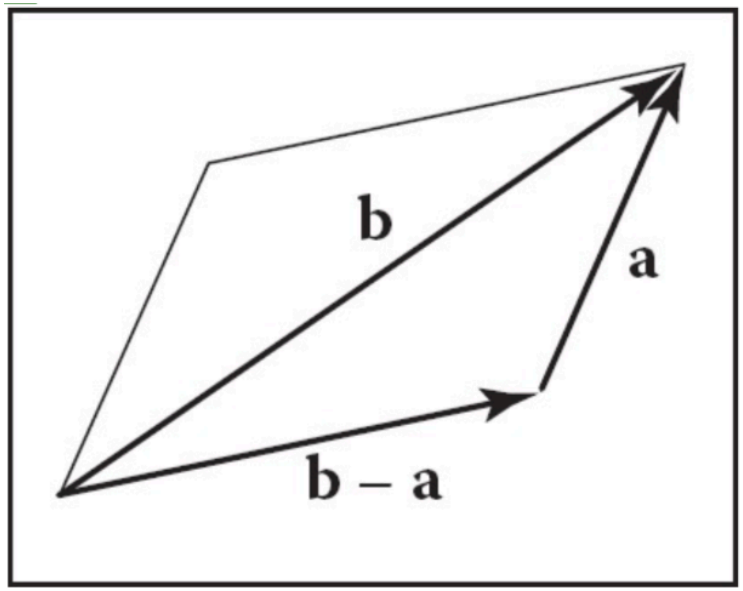
- We often use an arrow to represent a vector
  - The length of the arrow indicates the length of the vector, the direction of the arrow indicates the direction of the vector.
- The position of the arrow is irrelevant!
  - However, we can use vectors to represent positions by describing displacements from a common point



Vector Operations

- Vectors can be added, e.g. for vectors **a**,**b**, there exists a vector **c** = **a**+**b**
- Defined using the parallelogram rule: idea is to trace out the displacements and produced the combined effect
- Vectors can be negated (flip tail and head), and thus can be subtracted
- Vectors can be multiplied by a scalar, which scales the length but not the direction



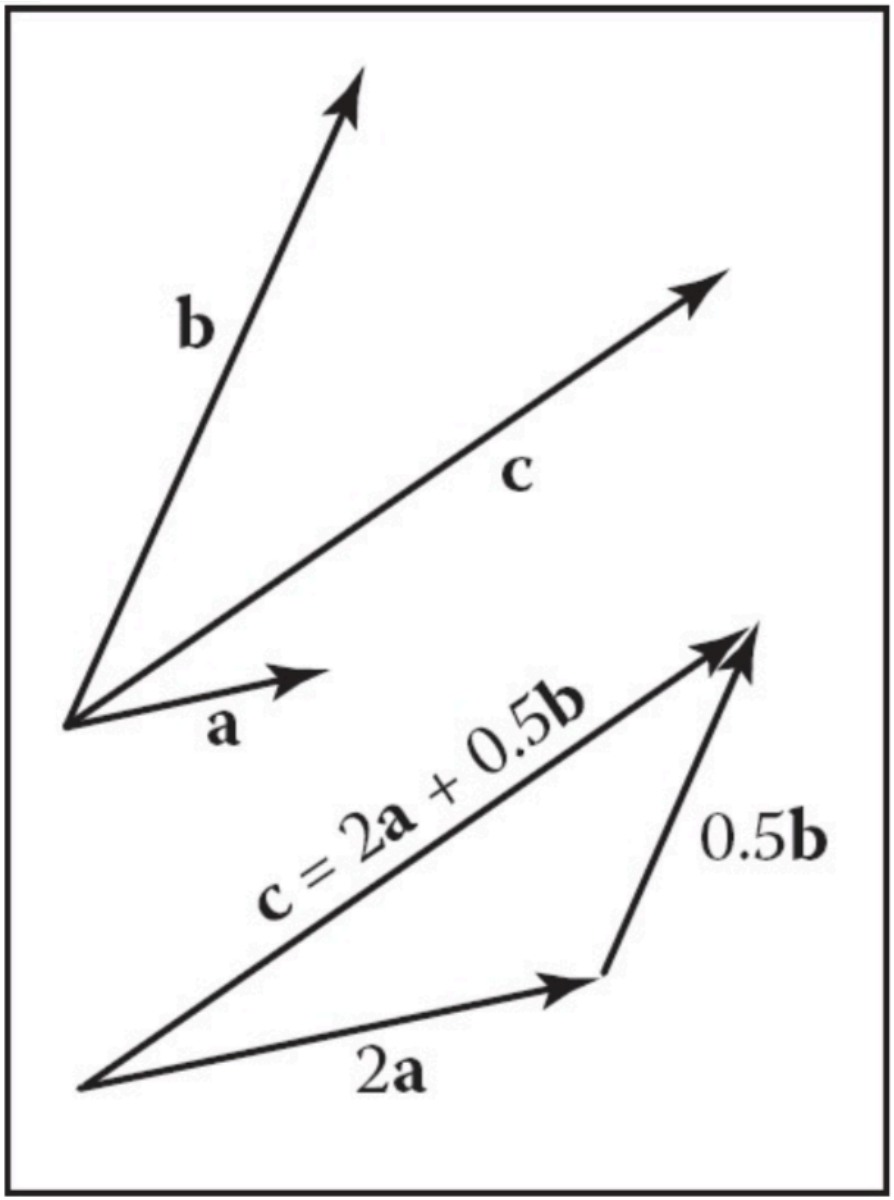


Vectors Decomposition

- By linear independence, any 2D vector can be written as a combination of any two nonzero, nonparallel vectors

- Such a pair of vectors is called a **2D basis**

$$\mathbf{c} = a_c\mathbf{a} + b_c\mathbf{b}$$



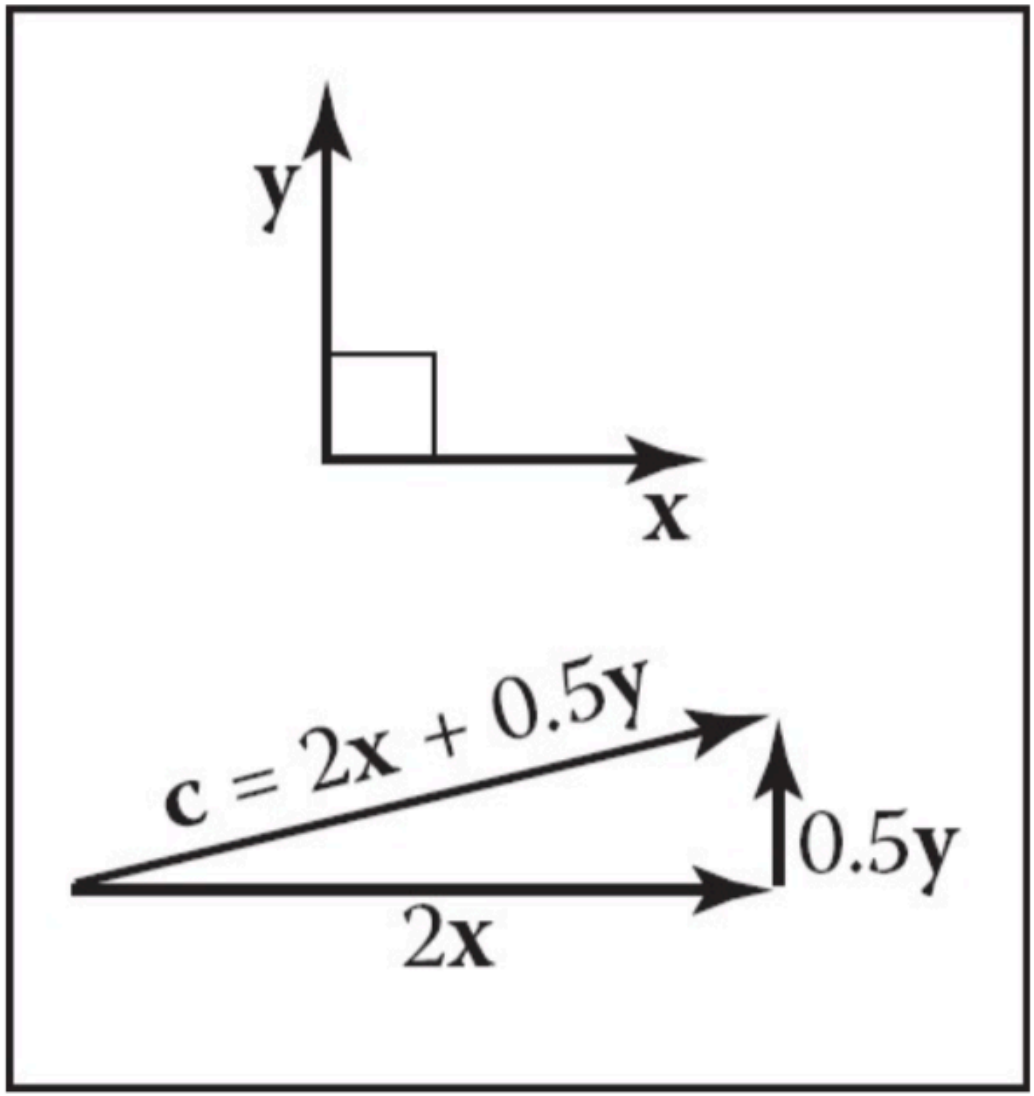
Canonical (Cartesian) Basis

- Often, we pick two perpendicular vectors, **x** and **y**, to define a common **basis**
- Notationally the same,

$$\mathbf{a} = x_a\mathbf{x} + y_a\mathbf{y}$$

- But we often don't bother to mention the basis vectors, and write the vector as **a** = (x_a,y_a), or

$$\mathbf{a} = \begin{bmatrix} x_a \\ y_a \end{bmatrix}$$

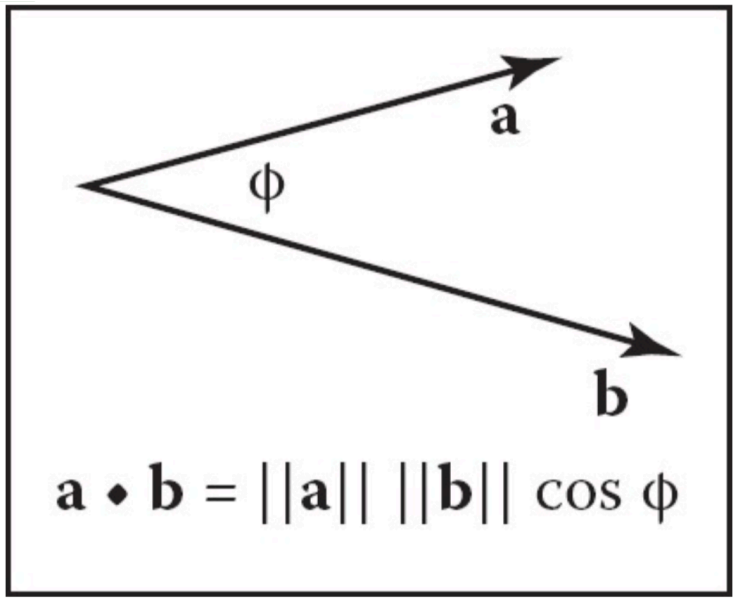


Vector Multiplication: Dot Products

- Given two vectors **a** and **b**, the **dot product**, relates the lengths of **a** and **b** with the angle ϕ between them:

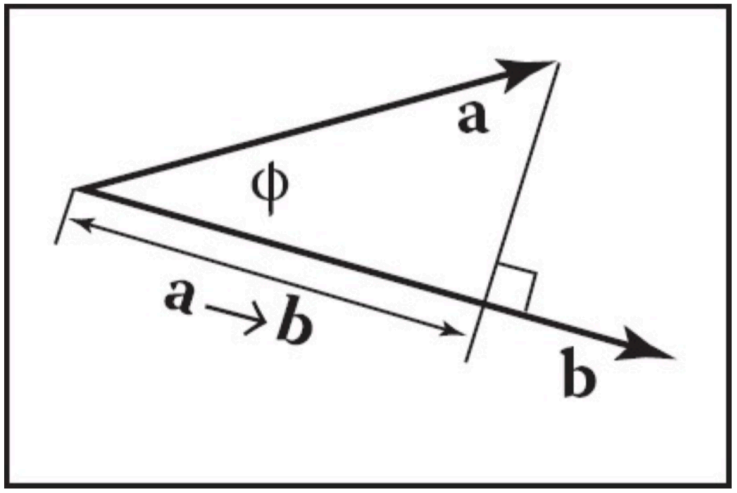
$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}|| \, ||\mathbf{b}|| \cos \phi$$

  - Sometimes called the scalar product, as it produces a scalar value



- Also can be used to produce the **projection**, **a**→**b**, of **a** onto **b**

$$\mathbf{a} \rightarrow \mathbf{b} = ||\mathbf{a}|| \cos \phi = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{b}||}$$



Dot Products are Associative and Distributive

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a},$$
$$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c},$$
$$(k\mathbf{a}) \cdot \mathbf{b} = \mathbf{a} \cdot (k\mathbf{b}) = k\mathbf{a} \cdot \mathbf{b}$$

- And, we can also define them directly if **a** and **b** are expressed in Cartesian coordinates:

$$\mathbf{a} \cdot \mathbf{b} = x_a x_b + y_a y_b$$

3D Vectors

- Same idea as 2D, except these vectors are defined typically with a basis of three vectors
  - Still just a direction and a magnitude
  - But, useful for describing objects in three-dimensional space
- Most operations exactly the same, e.g. dot products:

$$\mathbf{a} \cdot \mathbf{b} = x_a x_b + y_a y_b + z_a z_b$$

Cross Products

- In 3D, another way to "multiply" two vectors is the **cross product**, $\mathbf{a} \times \mathbf{b}$:

$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \, \|\mathbf{b}\| \sin \phi$$

- $\|\mathbf{a} \times \mathbf{b}\|$ is always the area of the parallelogram formed by **a** and **b**, and $\mathbf{a} \times \mathbf{b}$ is always in the direction perpendicular (two possible answers).

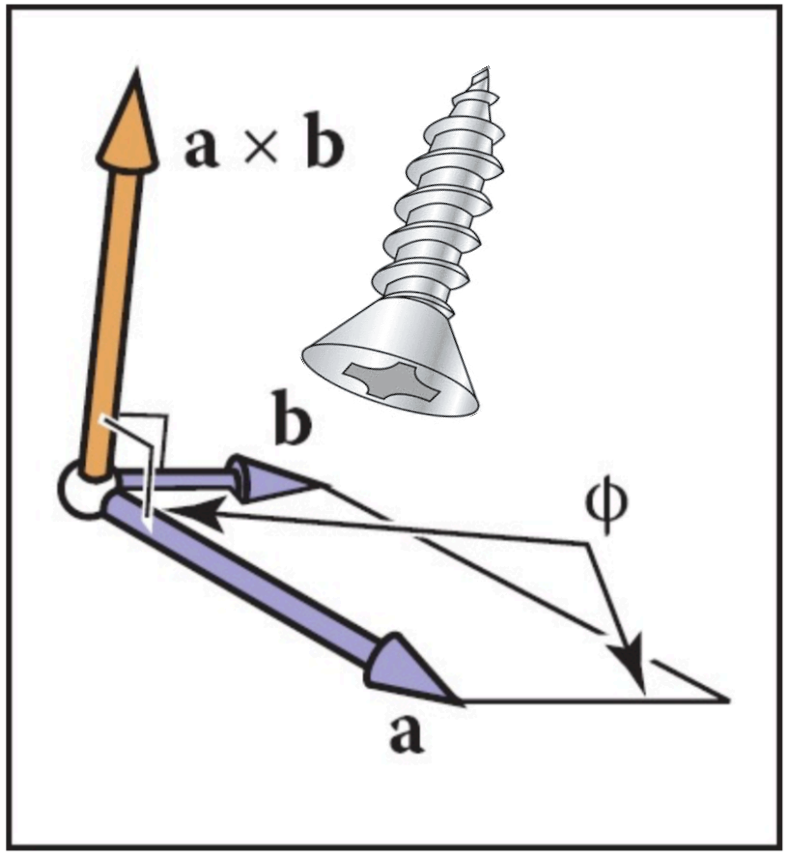
- A screw turned from **a** to **b** will progress in the direction $\mathbf{a} \times \mathbf{b}$

- Cross products distribute, but order matters:

$$\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$$

$$\mathbf{a} \times (k\mathbf{b}) = k(\mathbf{a} \times \mathbf{b})$$

$$\mathbf{a} \times \mathbf{b} = -(\mathbf{b} \times \mathbf{a})$$



Cross Products

- Since the cross product is always orthogonal to the pair of vectors, we can define our 3D Cartesian coordinate space with it:

$$\begin{aligned} \mathbf{x} &= (1,0,0) \\ \mathbf{y} &= (0,1,0) \\ \mathbf{z} &= (0,0,1) \end{aligned} \qquad \begin{aligned} \mathbf{x} \times \mathbf{y} &= +\mathbf{z}, \\ \mathbf{y} \times \mathbf{x} &= -\mathbf{z}, \\ \mathbf{y} \times \mathbf{z} &= +\mathbf{x}, \\ \mathbf{z} \times \mathbf{y} &= -\mathbf{x}, \\ \mathbf{z} \times \mathbf{x} &= +\mathbf{y}, \\ \mathbf{x} \times \mathbf{z} &= -\mathbf{y}. \end{aligned}$$

- In practice though (and the book derives this), we use the following to compute cross products:

$$\mathbf{a} \times \mathbf{b} = (y_a z_b - z_a y_b,\ z_a x_b - x_a z_b,\ x_a y_b - y_a x_b)$$

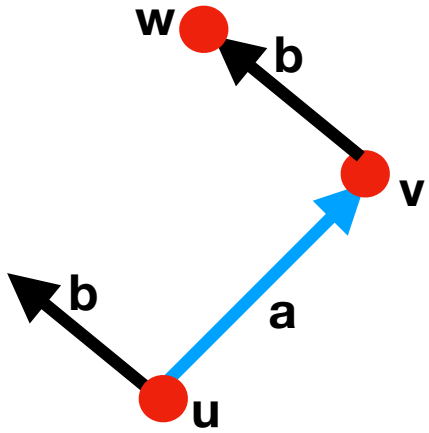
Checking orientation

Assume **a, b are in 2D (z=0)**. There are 3 possible scenarios.

a might be counter-clockwise **(ccw)** of **b**

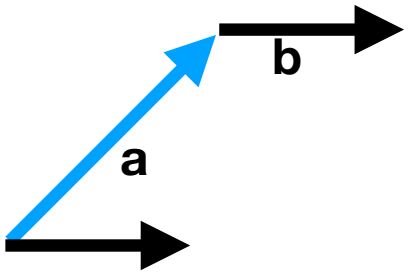
a might be clockwise **(cw)** of **b**

a is collinear with **b**



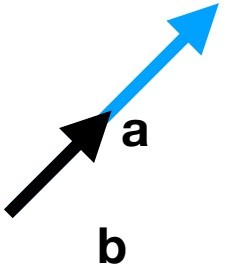
$$x_a y_b - y_a x_b > 0$$

a is counter-clockwise **(ccw)** of **b**



$$x_a y_b - y_a x_b < 0$$

a is clockwise **(cw)** of **b**



$$x_a y_b - y_a x_b = 0$$

a, b collinear

This will provide a convenient way to check if a triangle with vertices u,v,w is CCW or CW

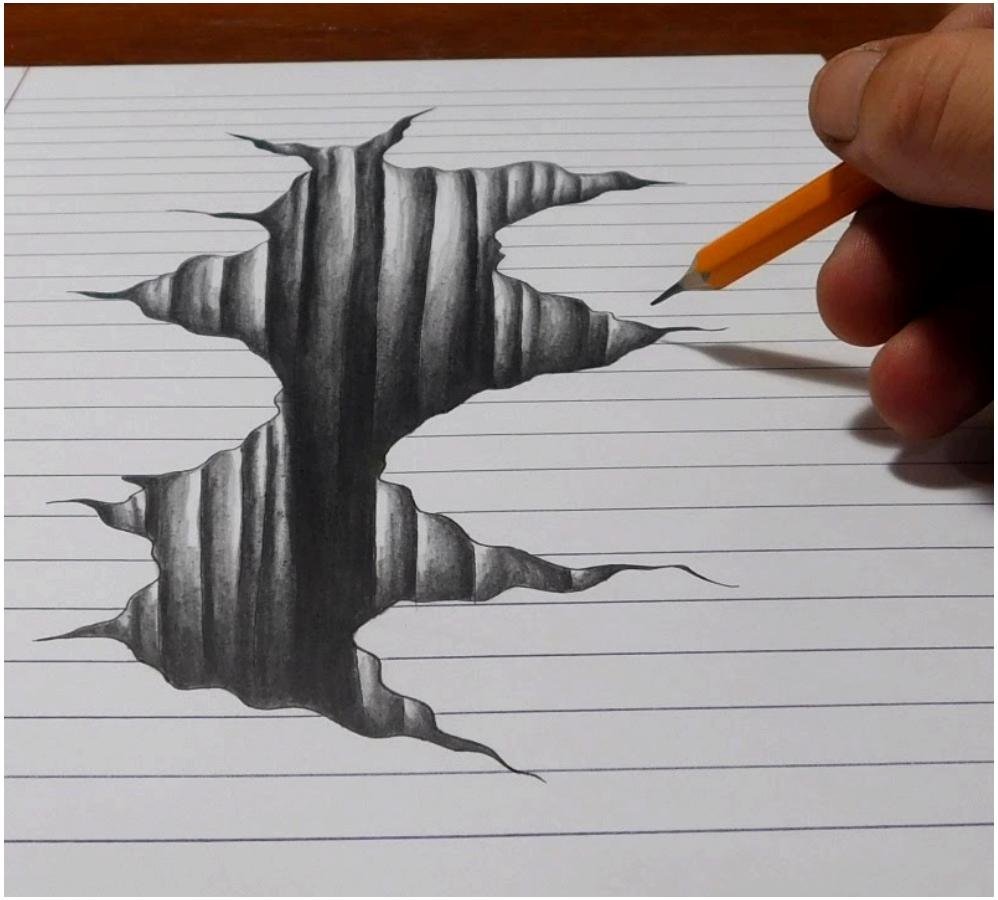
Rendering

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**

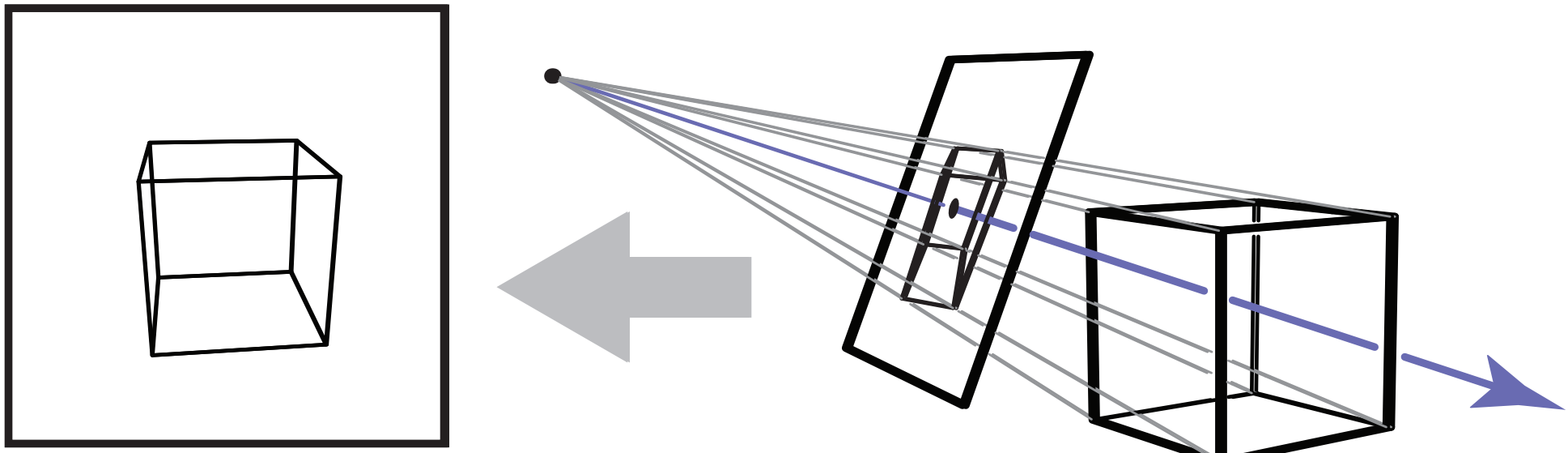
```
for each image pixel {
  for each object {
    if (object affects pixel)
    {
      do something
    }
  }
}
```

TODAY

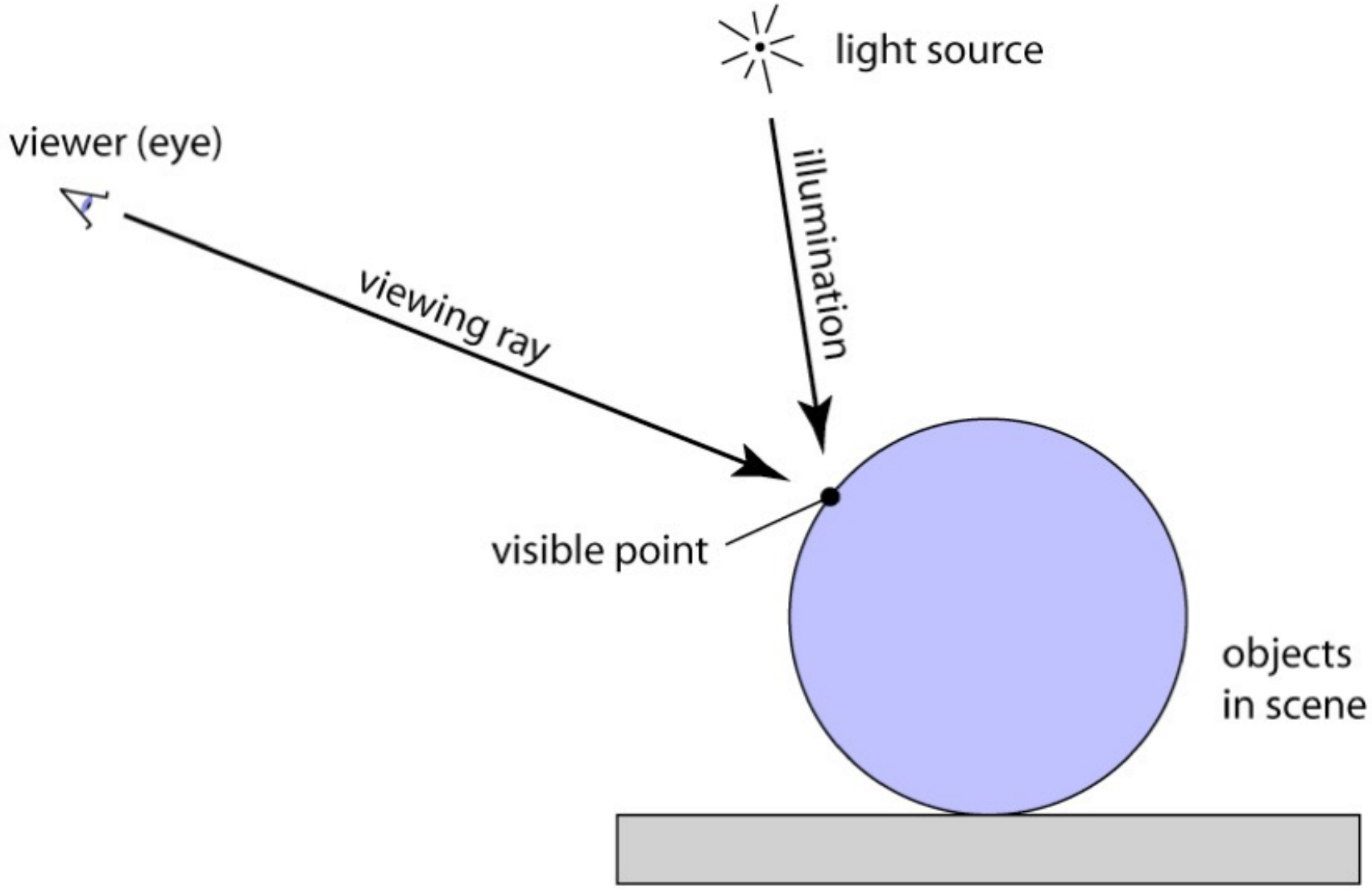
Basics of Ray Tracing

Idea of Ray Tracing

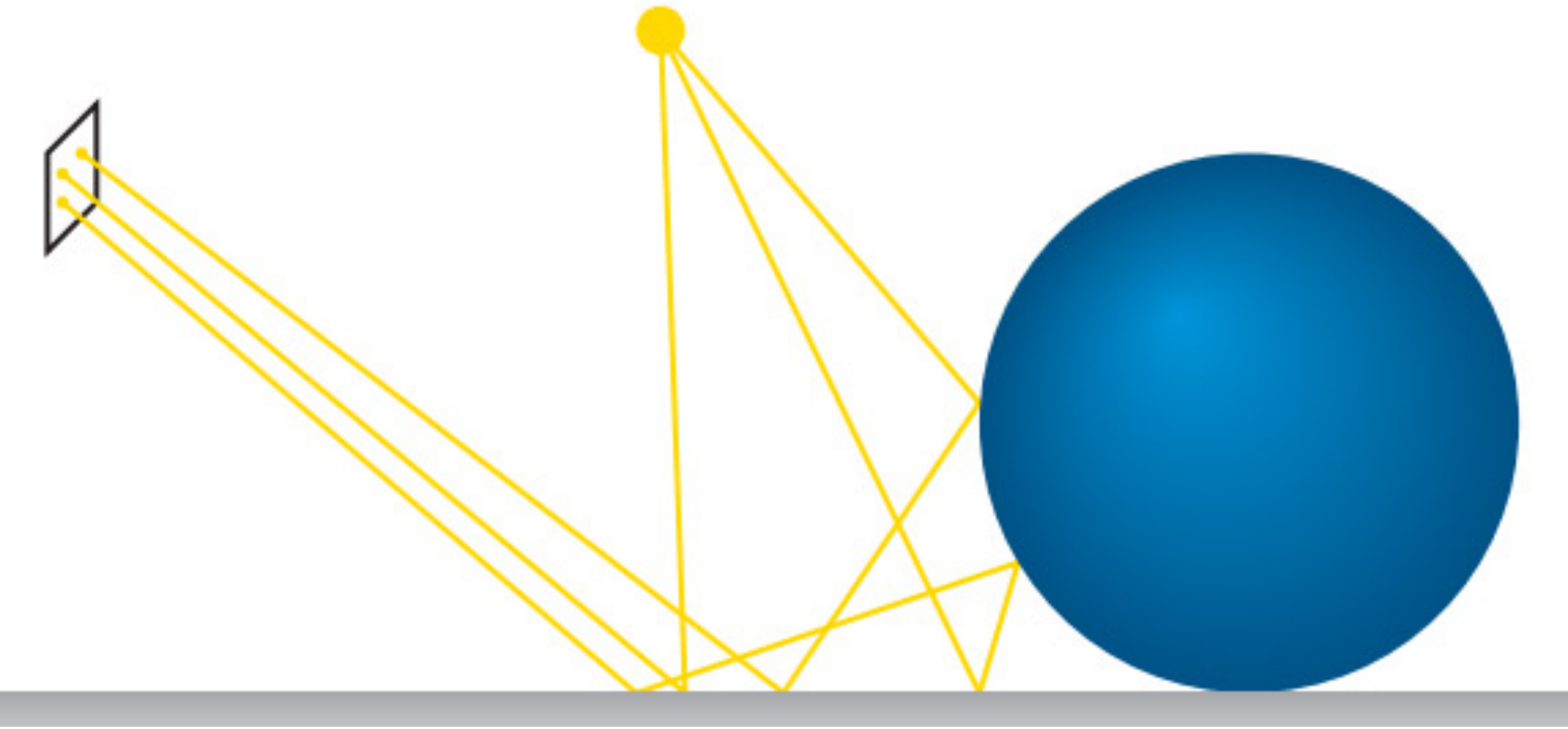
- Ask first, for each pixel: what belongs at that pixel?
- Answer: The set of objects that are visible if we were standing on one side of the image looking into the scene



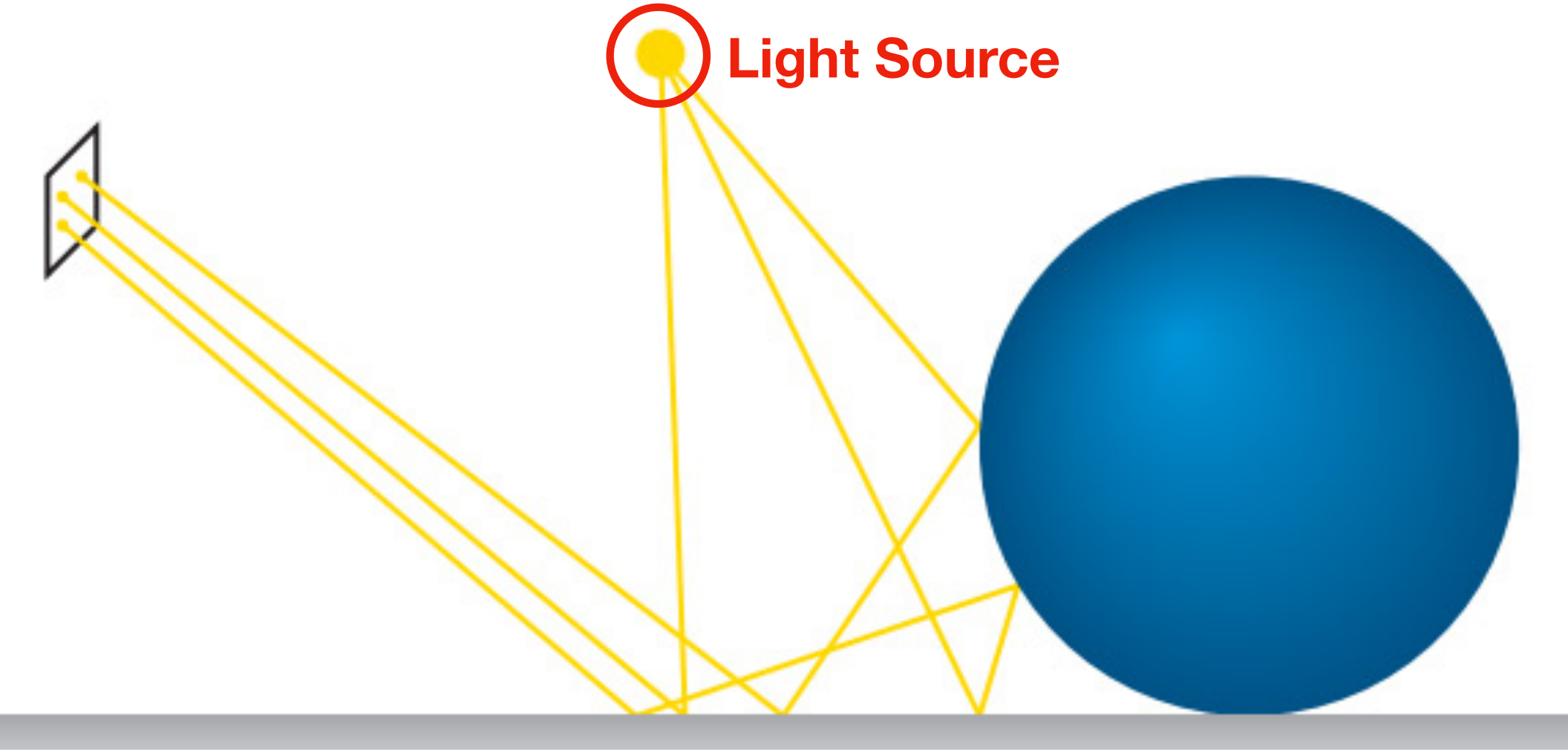
Key Concepts, in Diagram



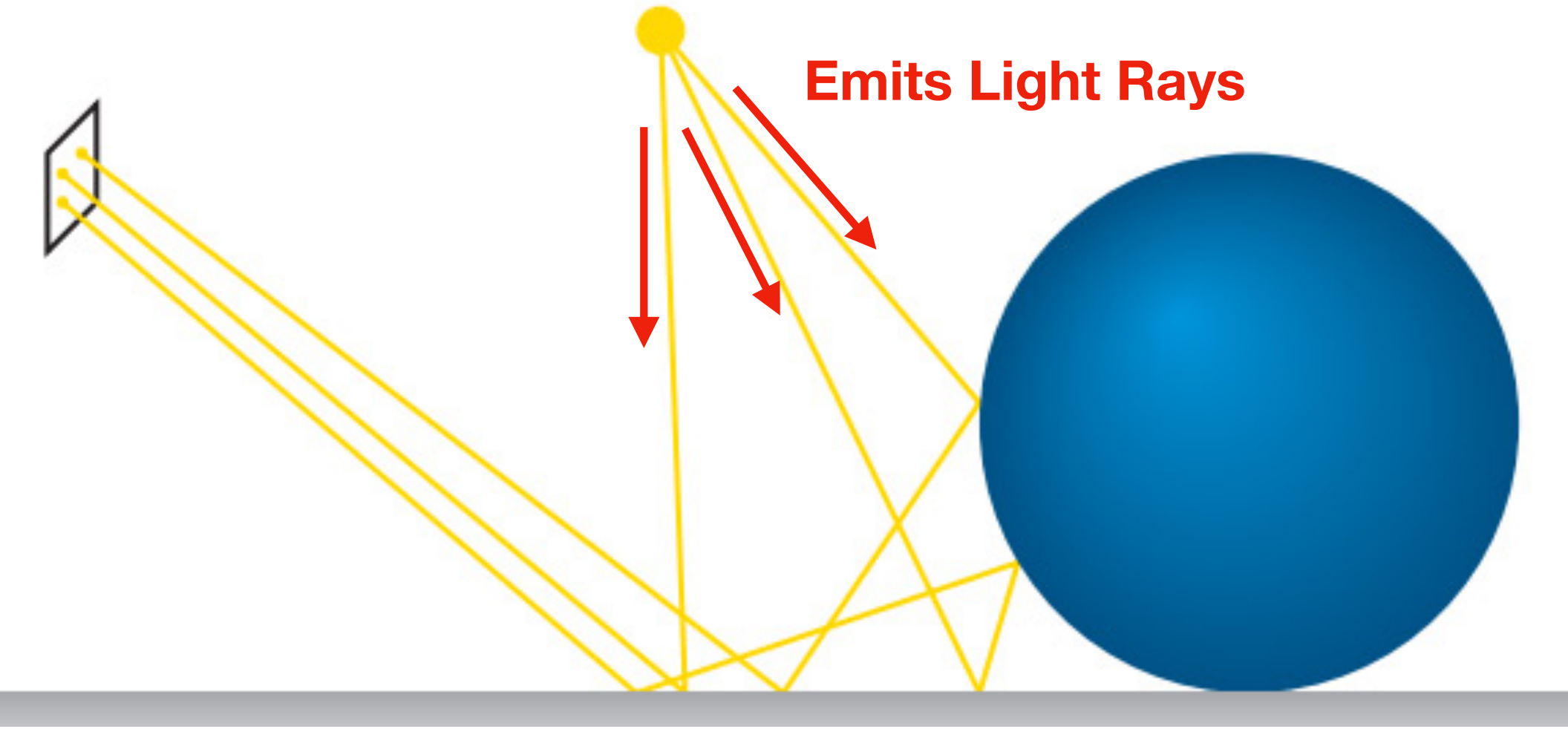
Idea: Using Paths of Light to Model Visibility



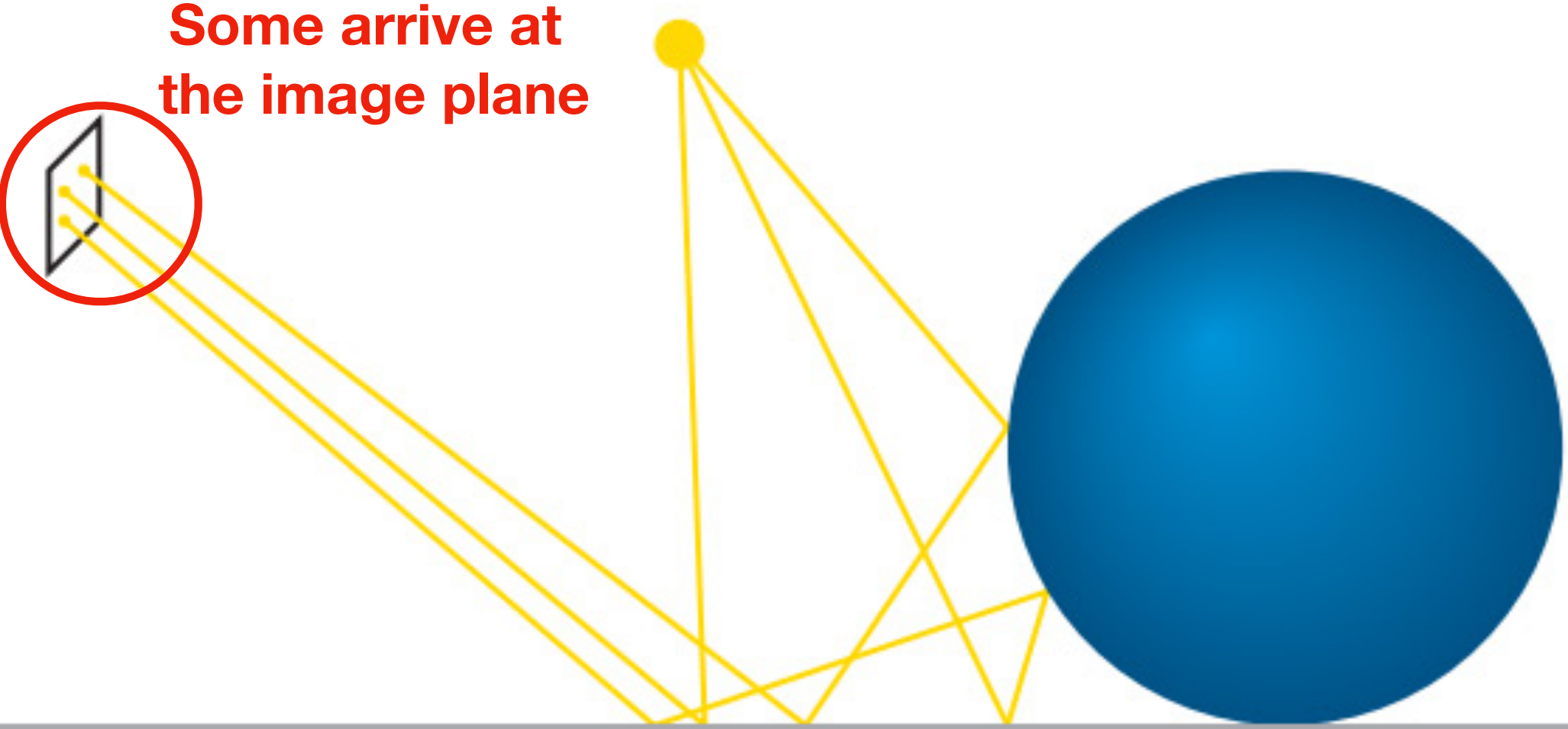
Using Paths of Light to Model Visibility



Using Paths of Light to Model Visibility



Using Paths of Light to Model Visibility



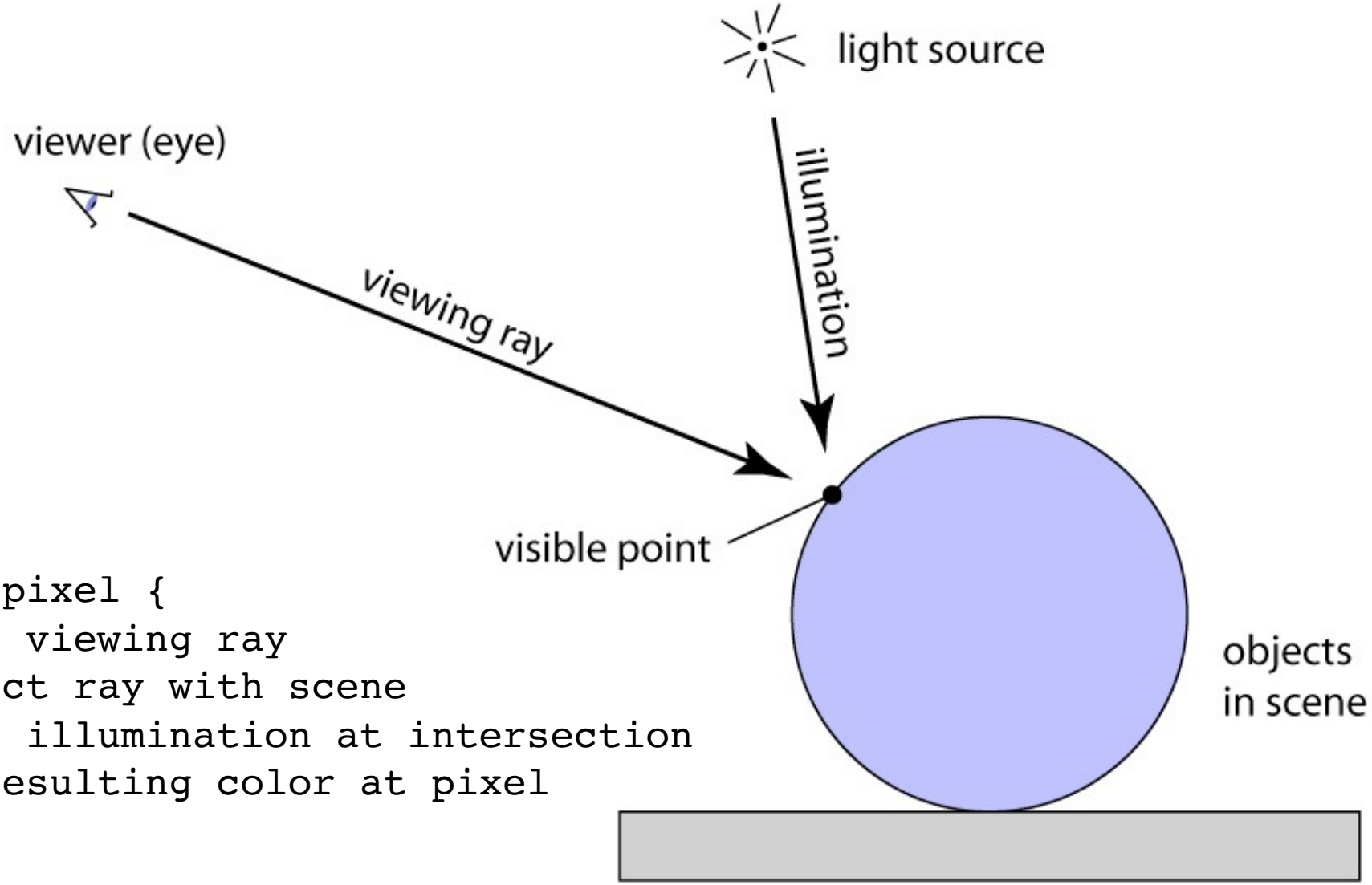
https://software.intel.com/file/37491

Using Paths of Light to Model Visibility

Forwarding vs Backward Tracing

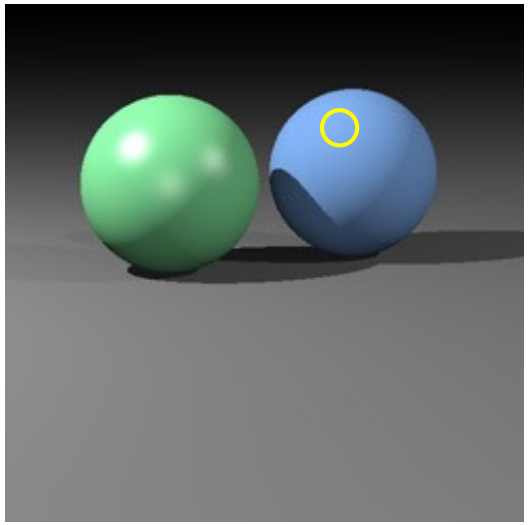
- Idea: Trace rays from light source to image
  - This is slow!
- Better idea: Trace rays **from** image **to** light source

Ray Tracing Algorithm




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

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

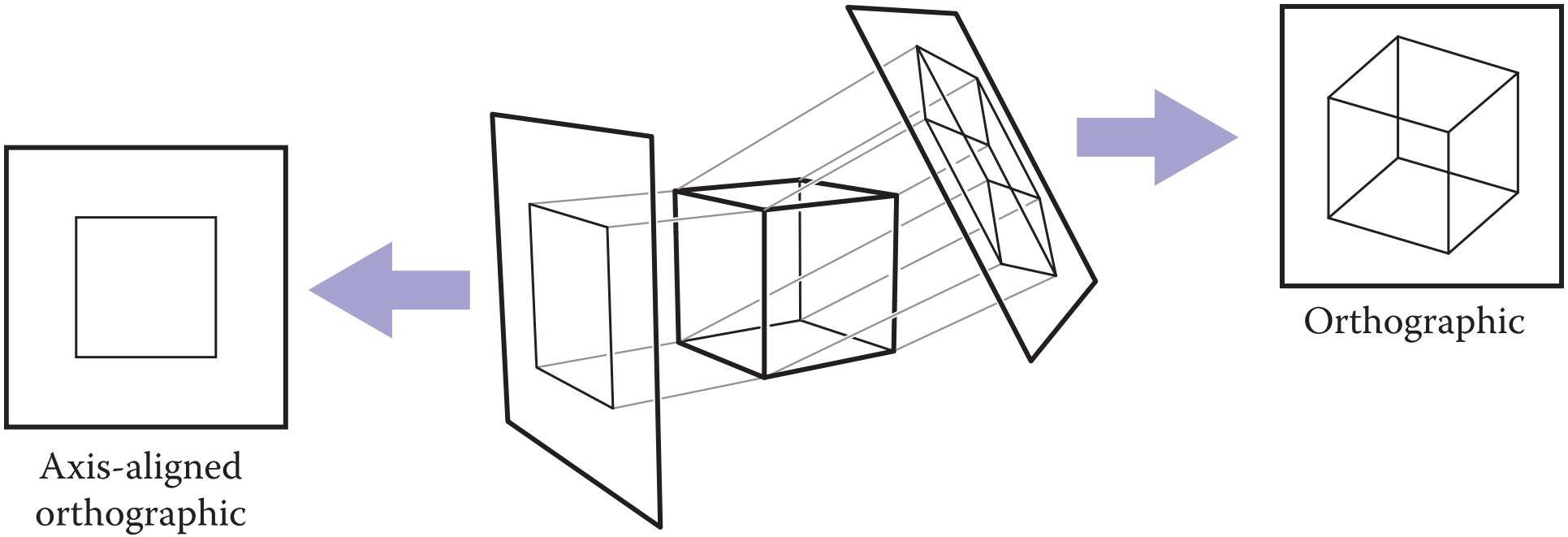
```
for each pixel {
  compute viewing ray
  intersect ray with scene
  compute illumination at intersection
  store resulting color at pixel
}
```

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

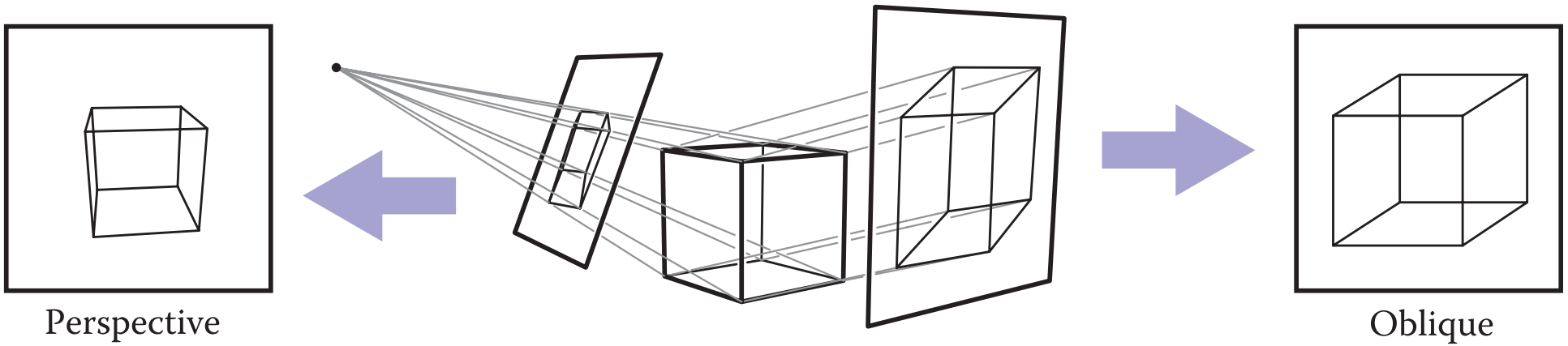
Orthographic Views

- 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

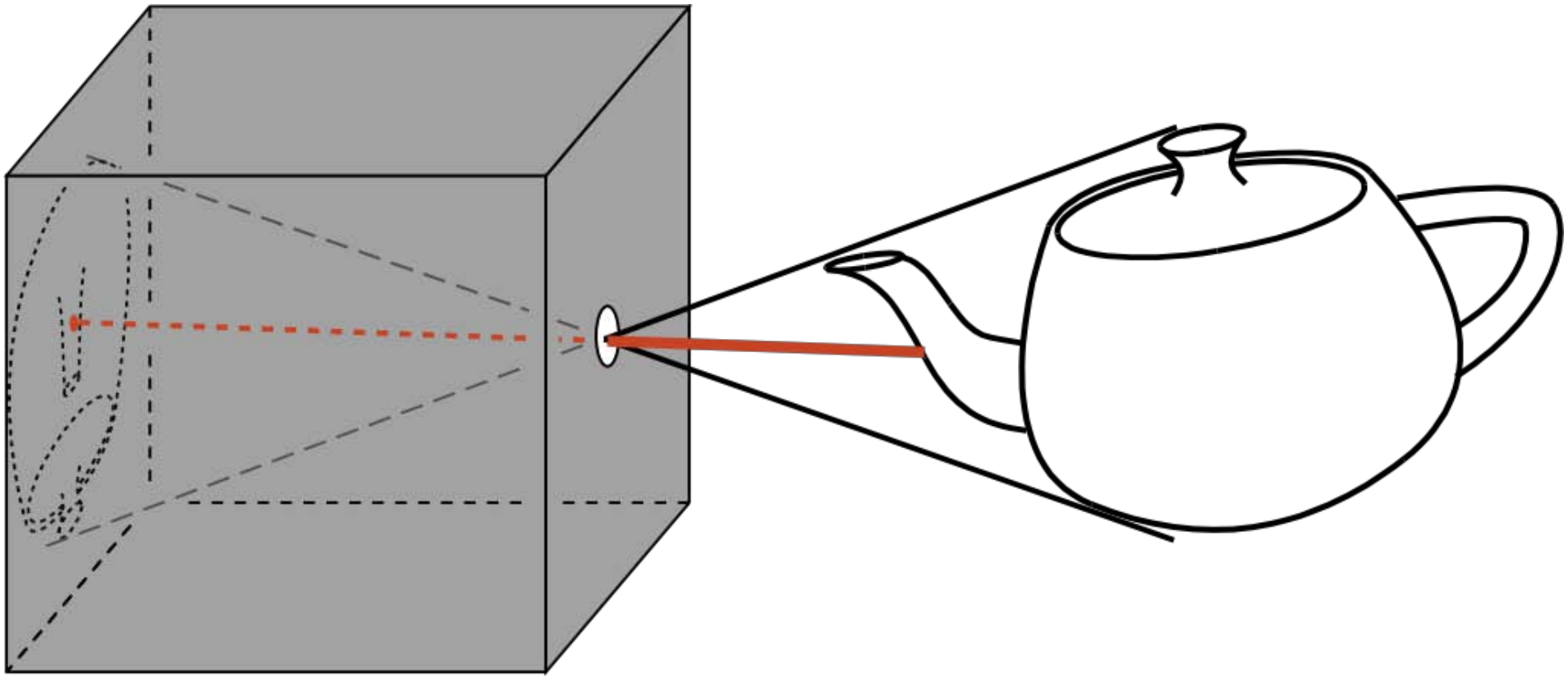
- But, objects that are further away should look smaller!
- Instead, we can project objects through a single viewpoint and record where they hit the plane.
- Lines which are paper in 3D might be non-parallel in the view



Pinhole Cameras

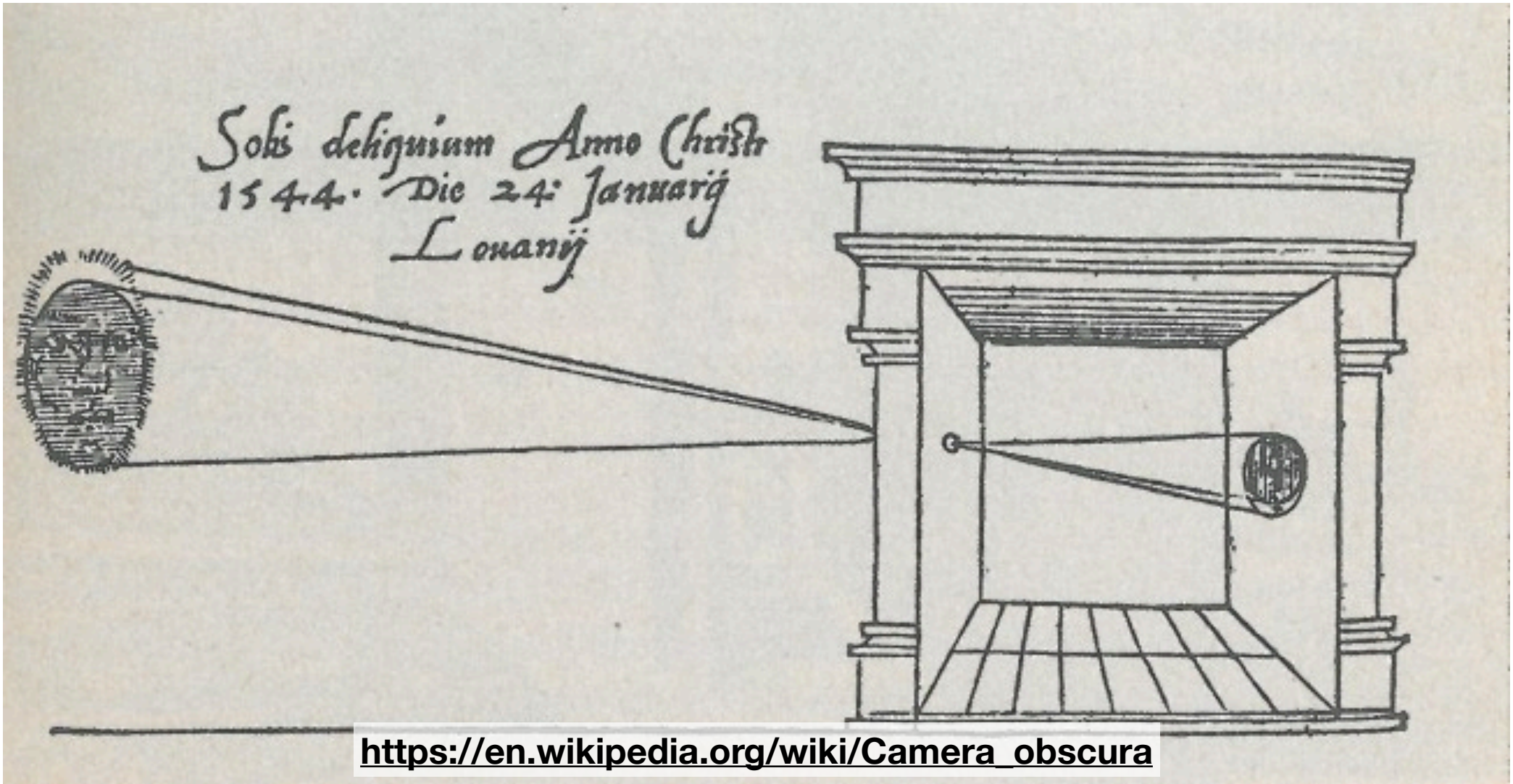
- Idea: Consider a box with a tiny hole. All light that passes through this hole will hit the opposite side

- Produced image inverts



Camera Obscura

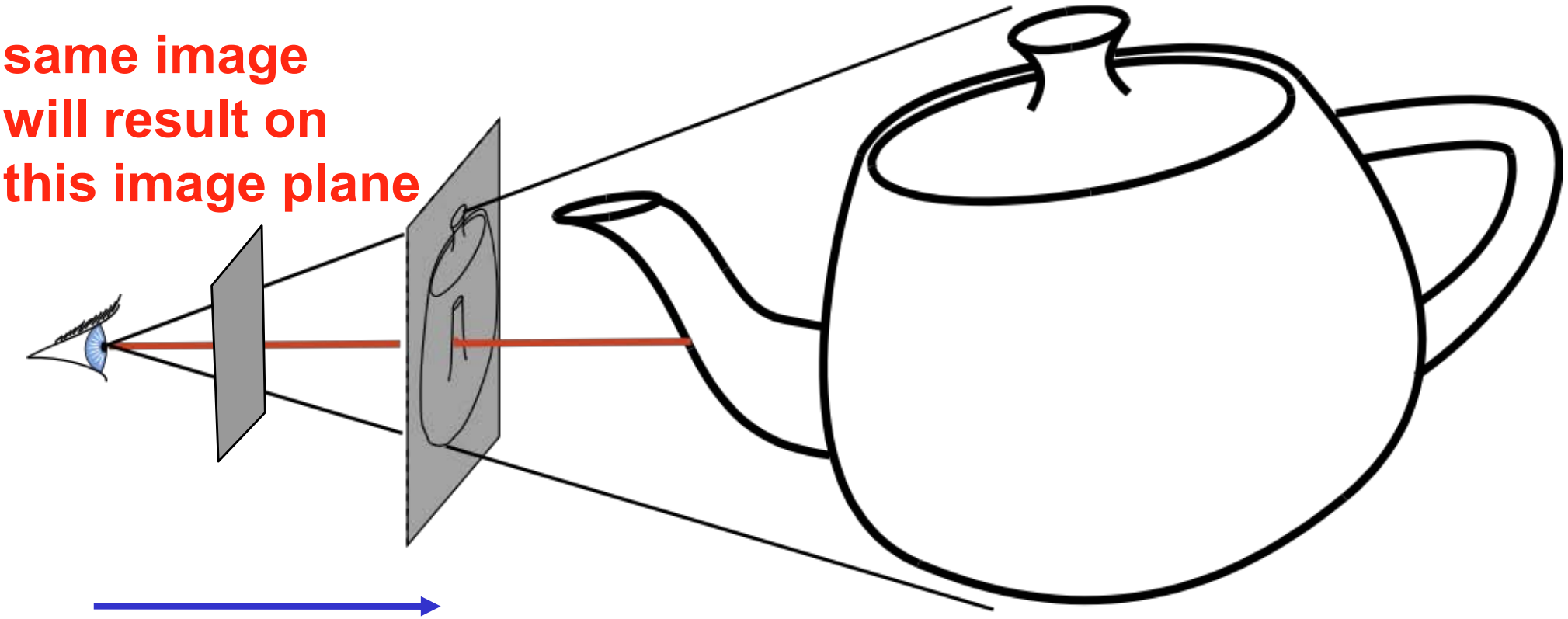
- Gemma Frisius, 16th century



https://en.wikipedia.org/wiki/Camera_obscura

Simplified Pinhole Cameras

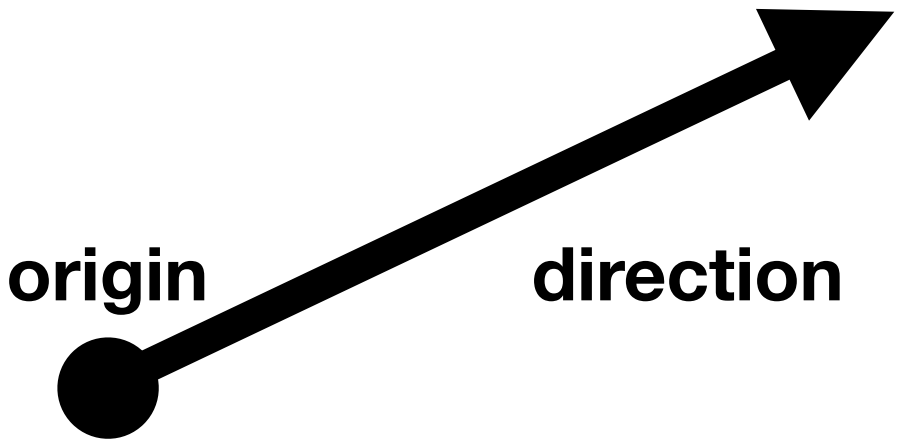
- Instead, we can place the eye at the pinhole and consider the eye-image pyramid (sometimes called **view frustum**)



Defining Rays

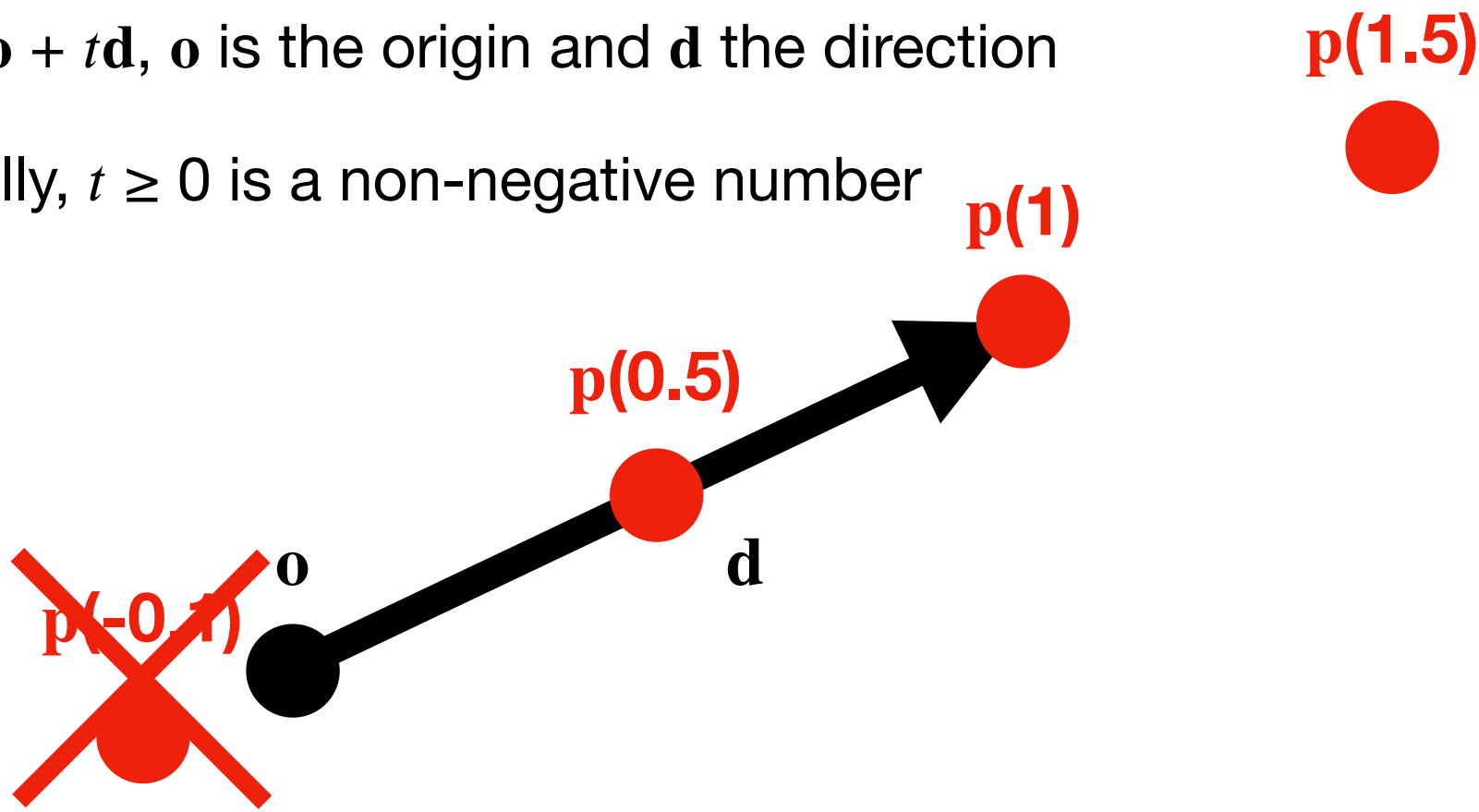
Mathematical Description of a Ray

- 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

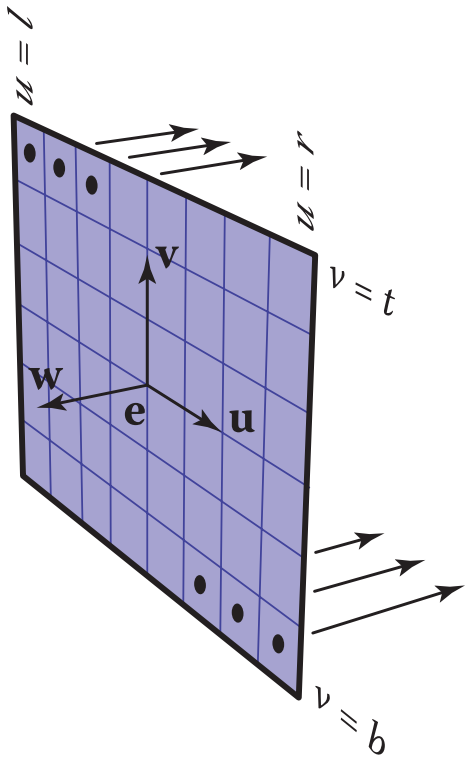


Mathematical Description of a Ray

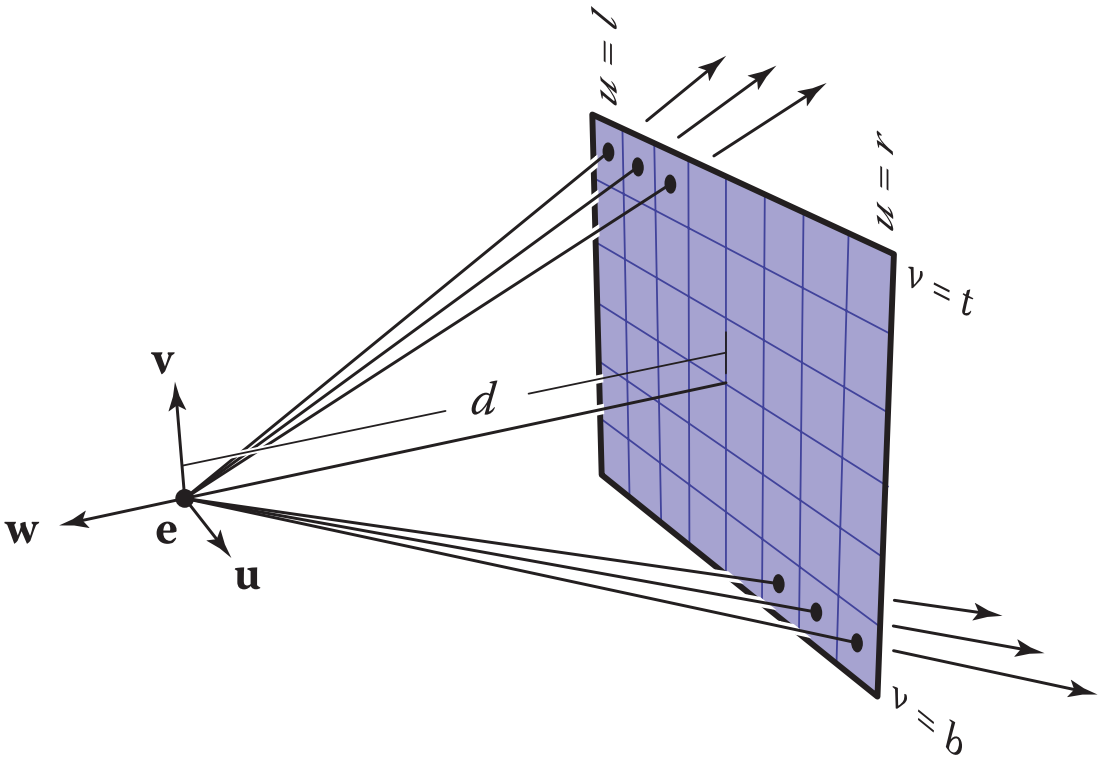
- Rays define a family of points, $\mathbf{p}(t)$, using a **parametric** definition
- $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$, $\mathbf{o}$ is the origin and $\mathbf{d}$ the direction
- Typically, $t \geq 0$ is a non-negative number



Orthographic vs. Perspective Rays



Parallel projection
same direction, different origins



Perspective projection
same origin, different directions

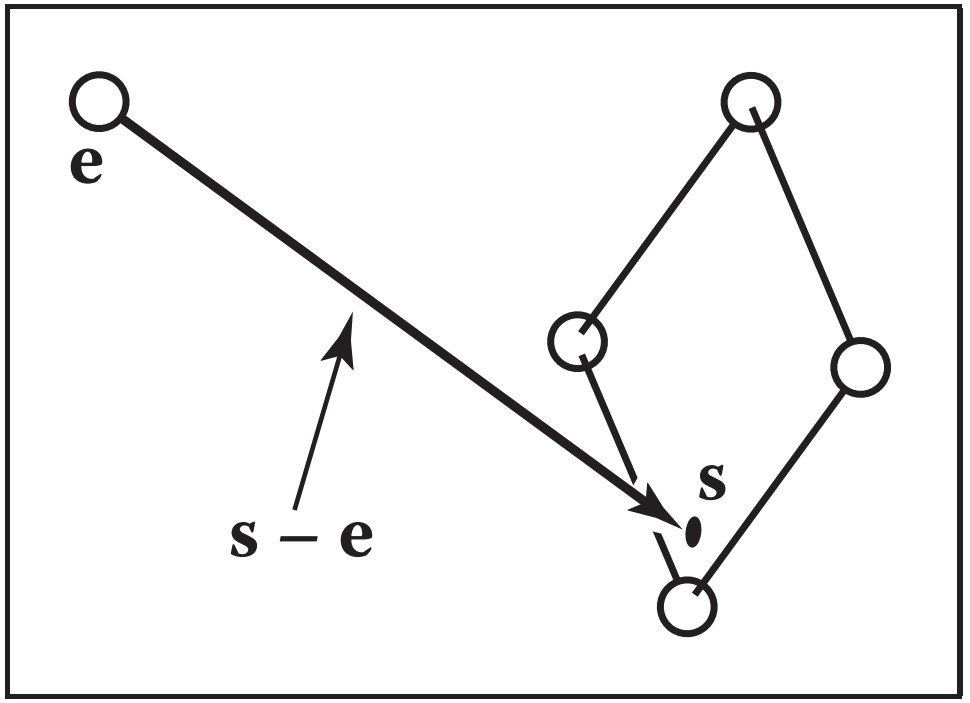
Defining o and d in Perspective Projection

- Given a viewpoint, $\mathbf{e}$, and a position on the image plane, $\mathbf{s}$

  $\mathbf{o} = \mathbf{e}$

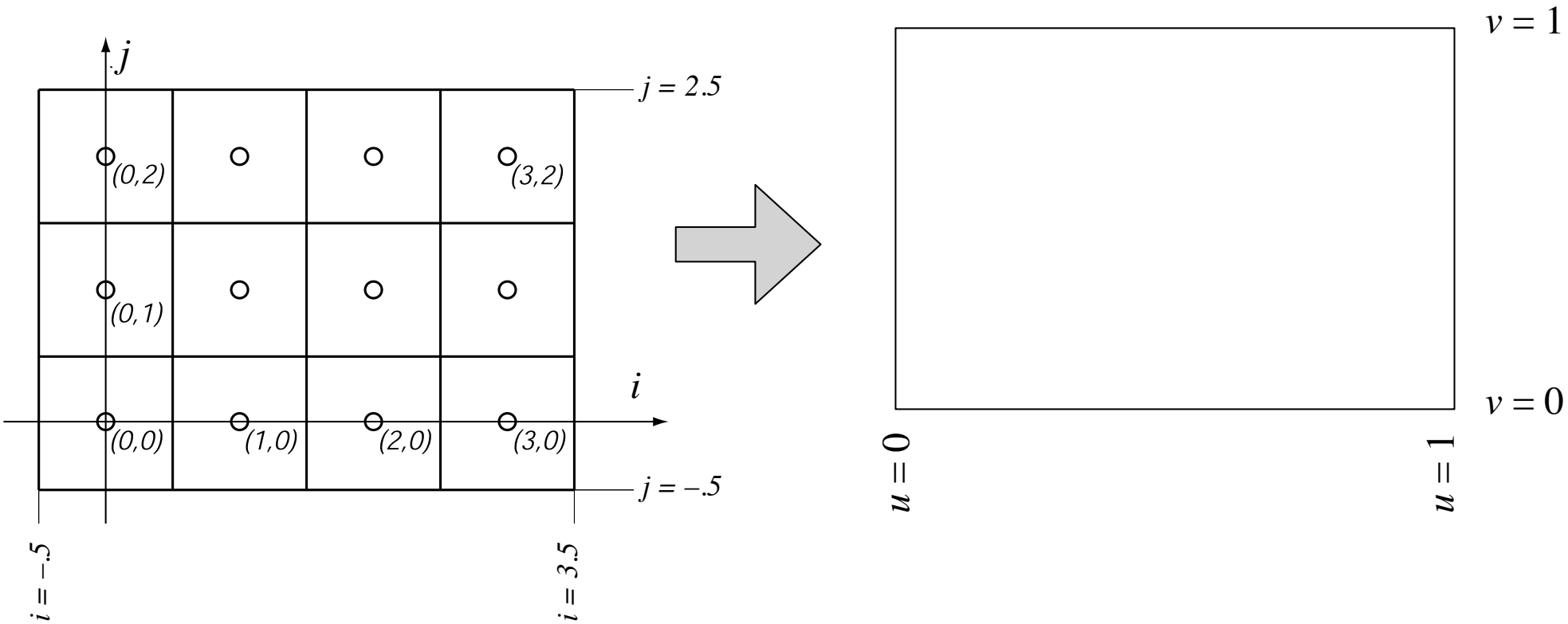
  $\mathbf{d} = \mathbf{s} - \mathbf{e}$

- And thus $\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$



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?



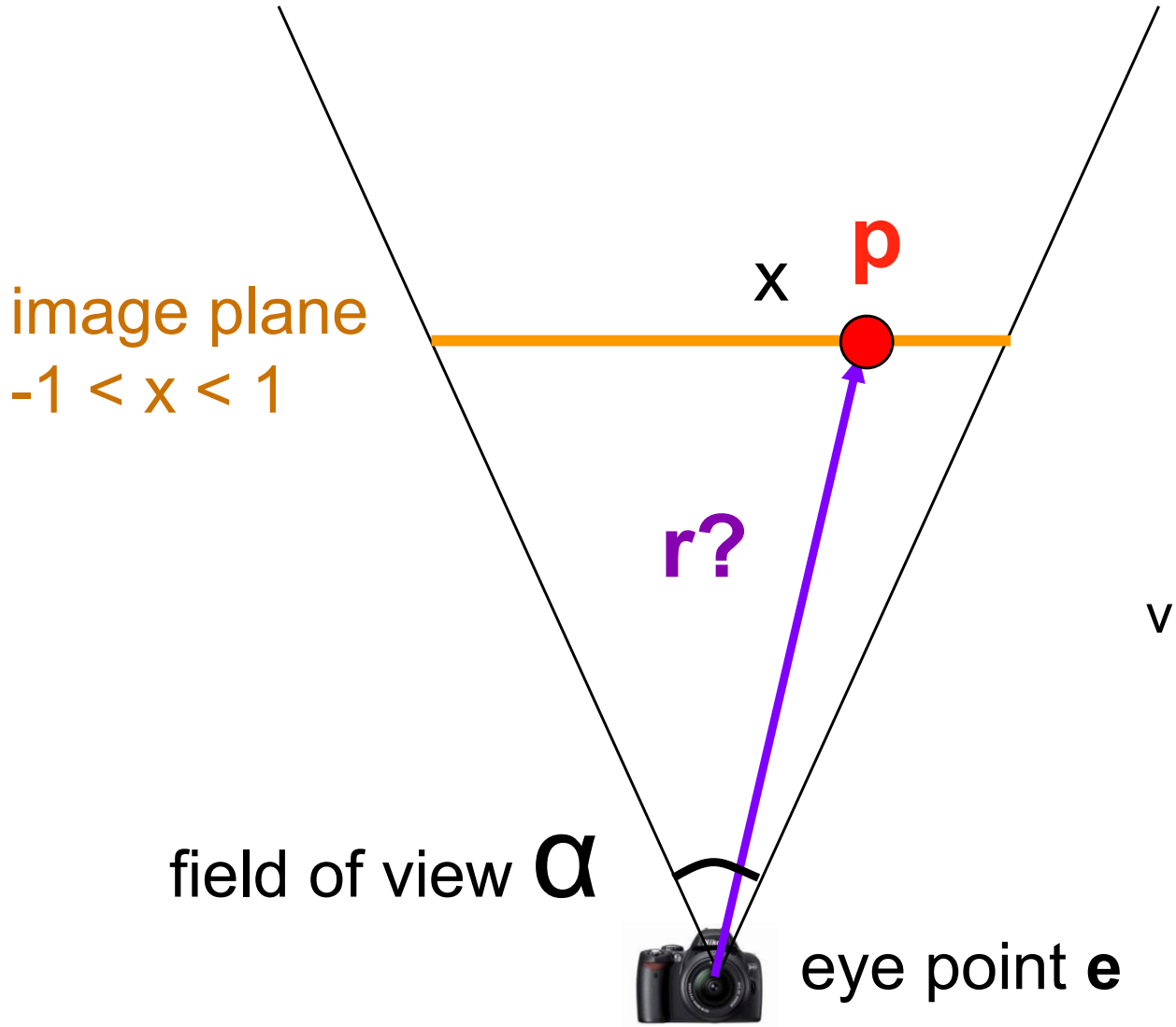
$$u = (i + 0.5)/n_x$$

$$v = (j + 0.5)/n_y$$

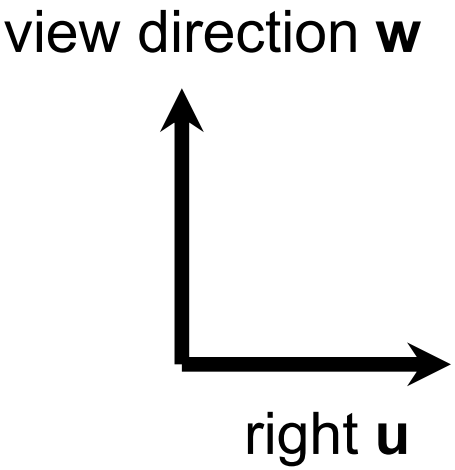
Camera Components

- 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

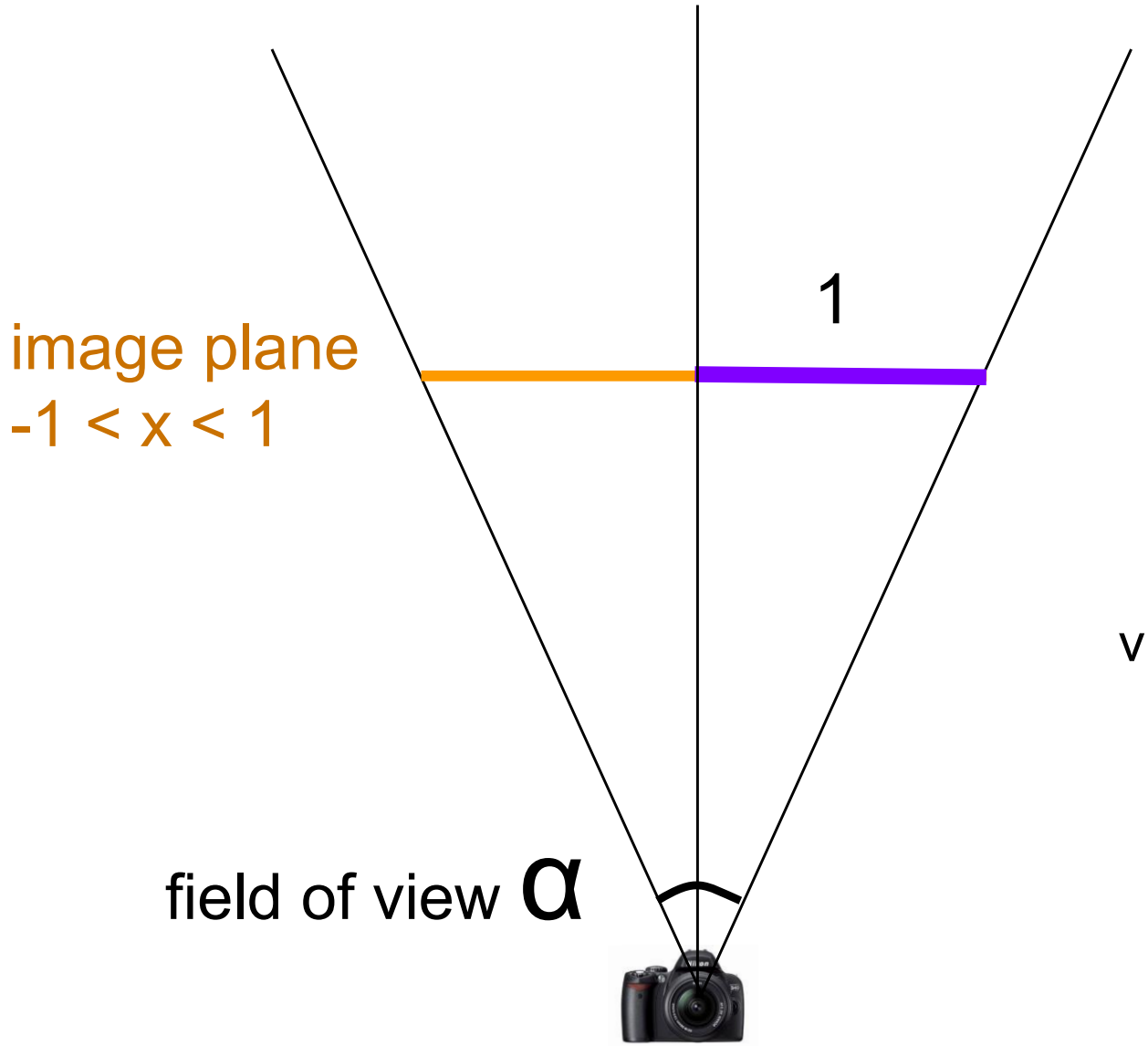
Ray Generation in 2D



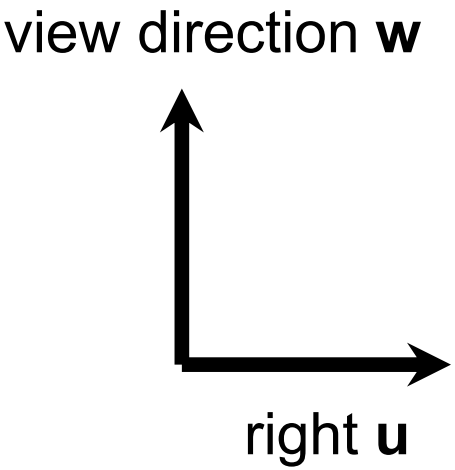
p is point on image plane at coordinate x, we want to know the direction of the ray **r**



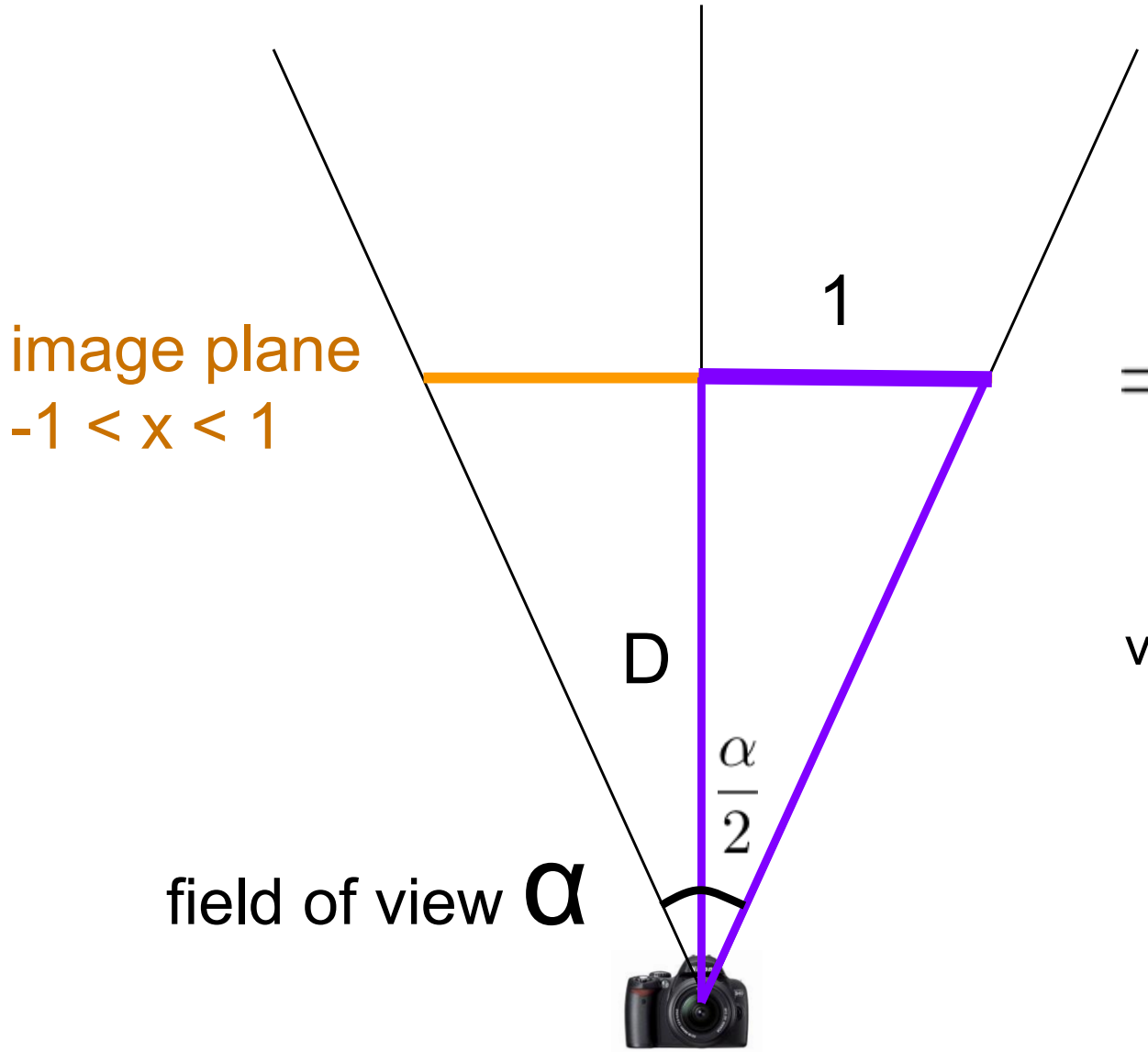
Ray Generation in 2D



What is the distance D to the screen so that the normalized coordinates go to 1?

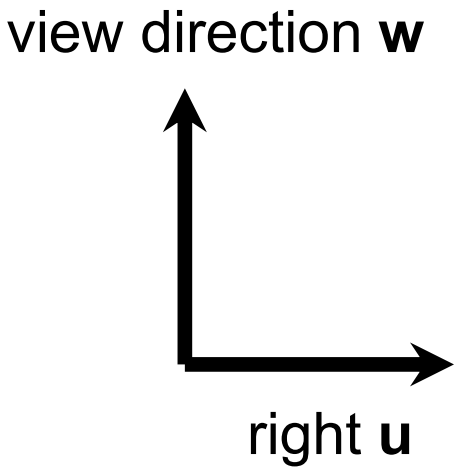


Ray Generation in 2D

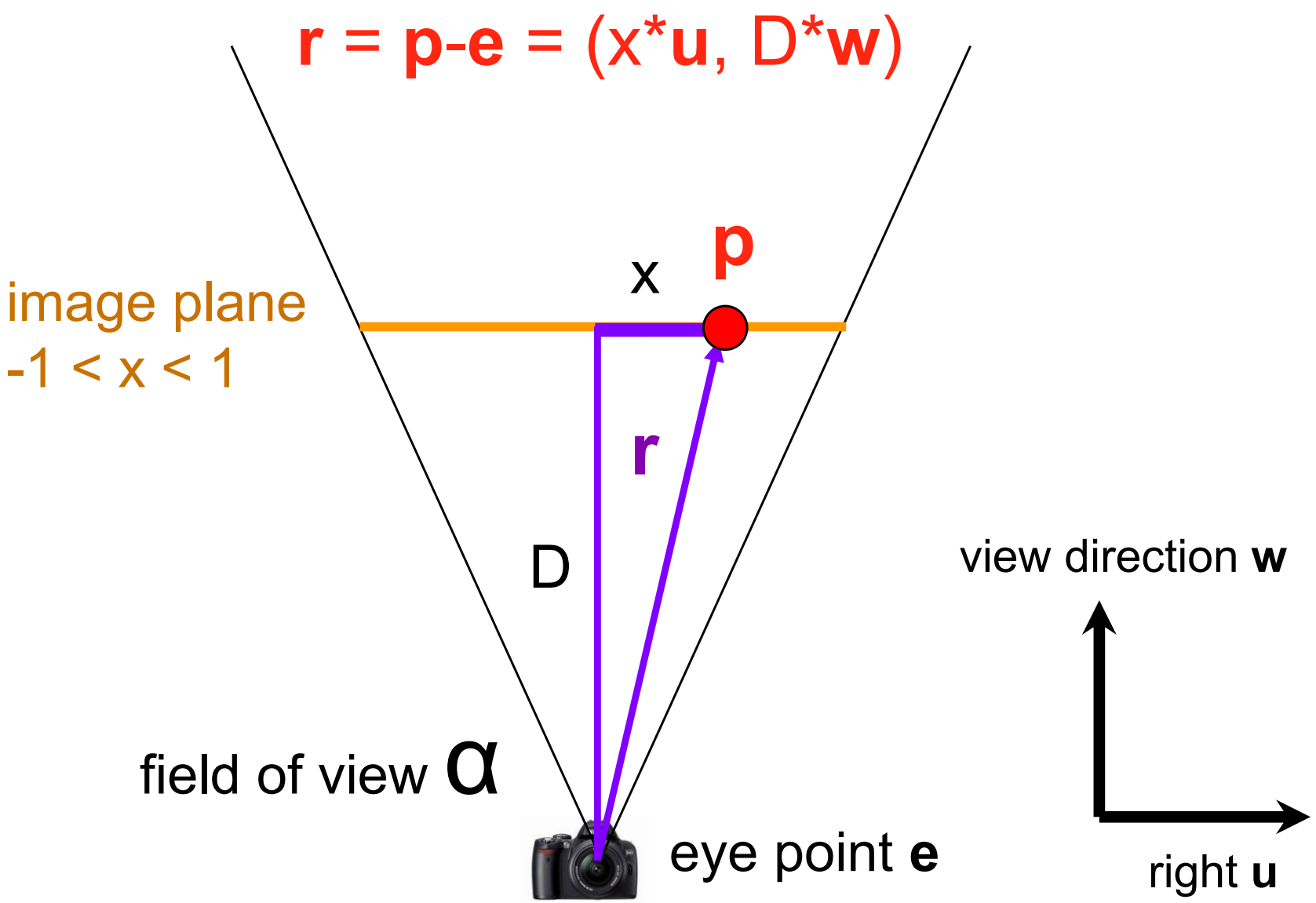


$$\tan\frac{\alpha}{2} = \frac{1}{D}$$

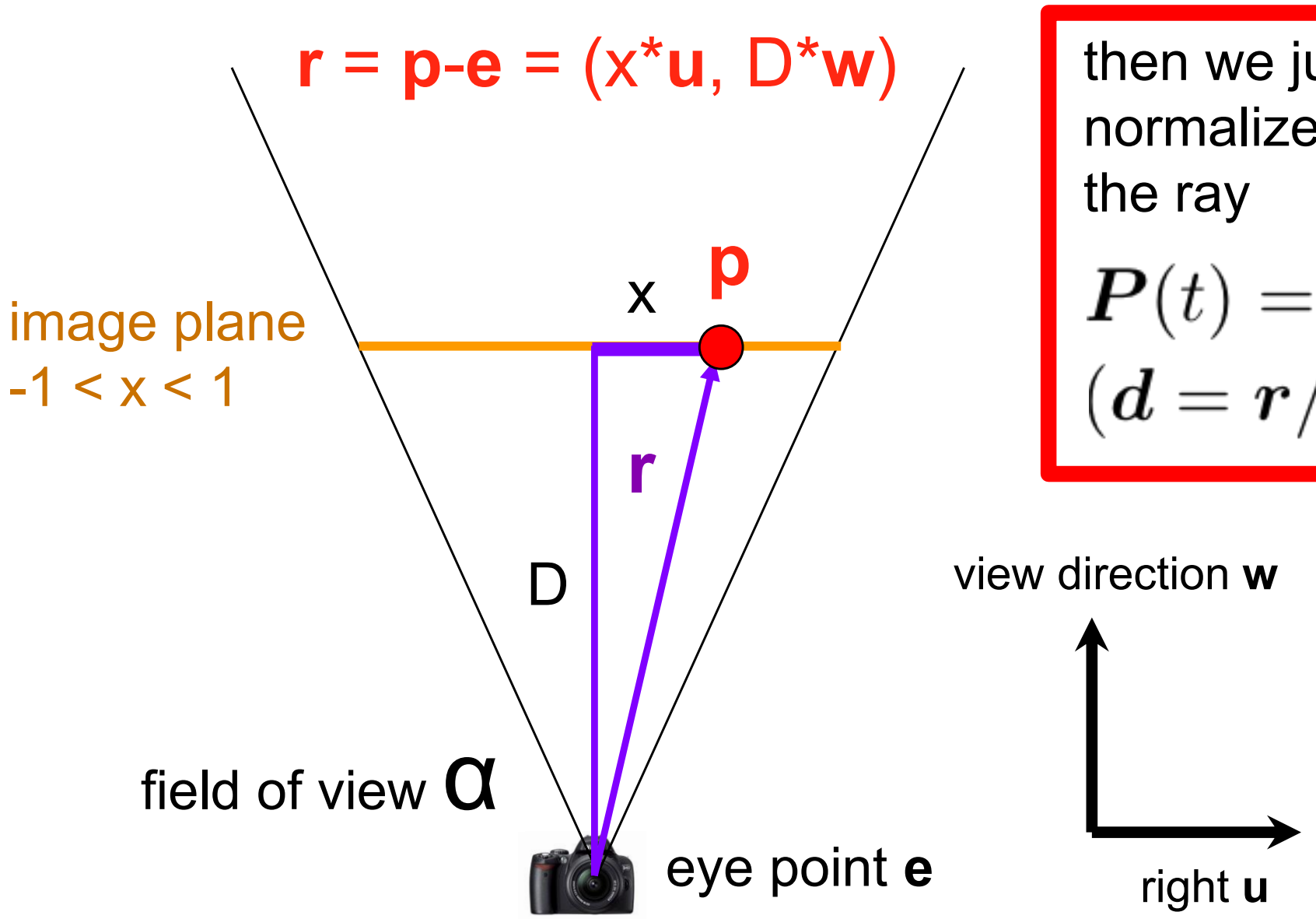
$$\Rightarrow D = \frac{1}{\tan(\alpha/2)}$$



Ray Generation in 2D



Ray Generation in 2D



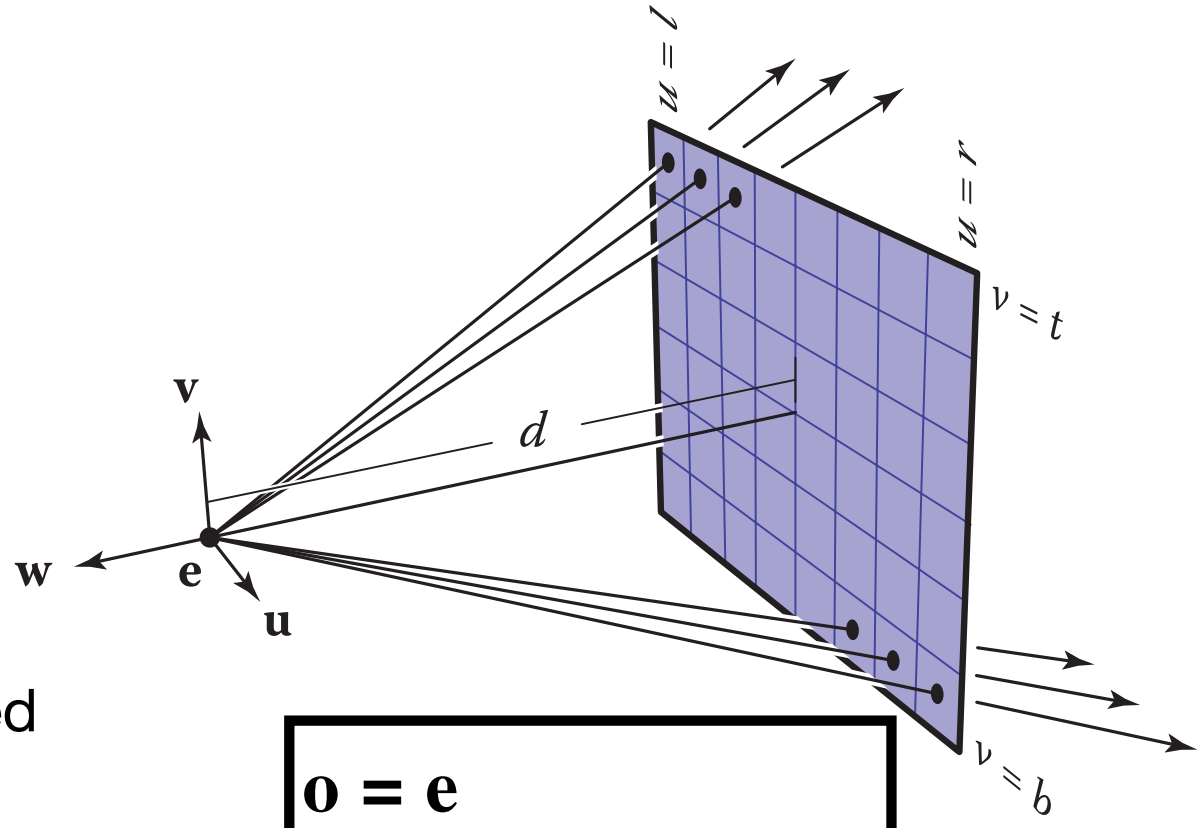
then we just normalize **r** to get the ray

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

$$(\boldsymbol{d} = \boldsymbol{r}/\|\boldsymbol{r}\|)$$

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
- 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**

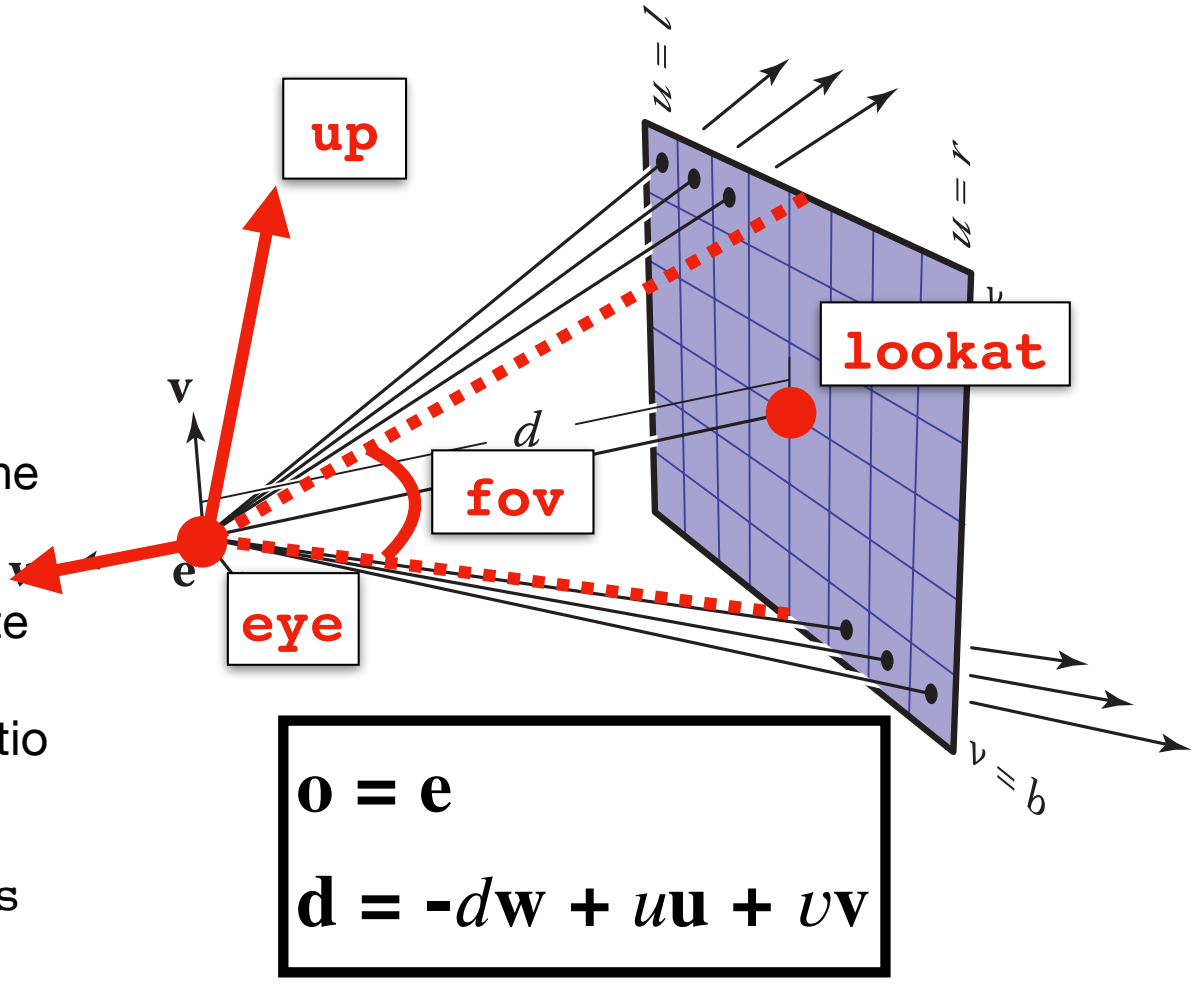


$$\mathbf{o} = \mathbf{e}$$

$$\mathbf{d} = -d\mathbf{w} + u\mathbf{u} + v\mathbf{v}$$

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**!
- A `fov_angle` for the vertical FOV
  - The FOV defines the **height** of the image plane in world space
  - 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



Intersecting Objects

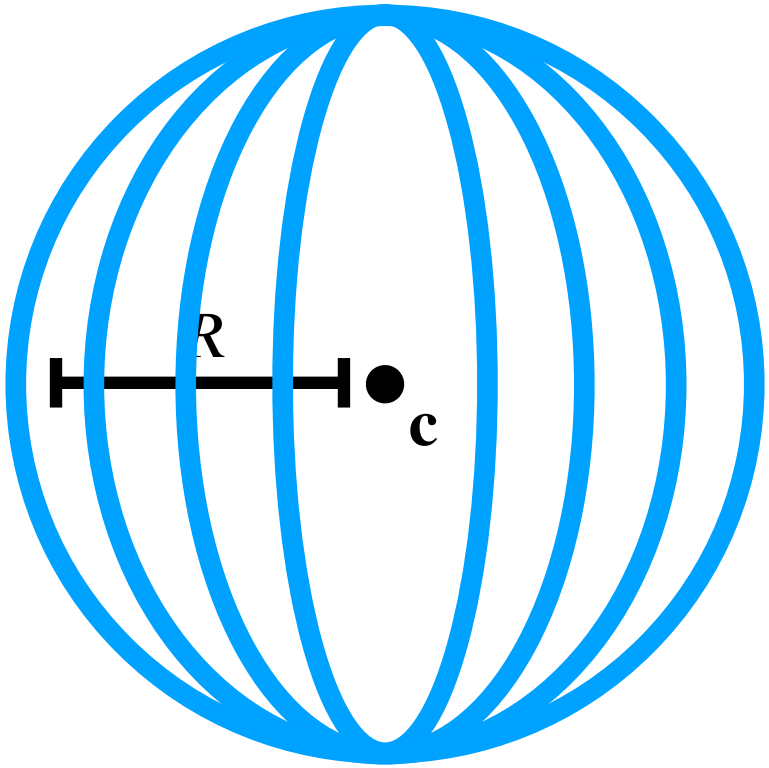
```
for each pixel {
  compute viewing ray
  intersect ray with scene
  compute illumination at intersection
  store resulting color at pixel
}
```

Defining a Sphere

- We can define a sphere of radius R, centered at position $\mathbf{c}$, using the implicit form

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

- Any point $\mathbf{p}$ that satisfies the above lives on the sphere



Ray-Sphere Intersection

- Two conditions must be satisfied:
  - Must be on a ray: $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$
  - 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 $f(\mathbf{p}(t))$:

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

- Solving for t is a quadratic equation

Ray-Sphere Intersection

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

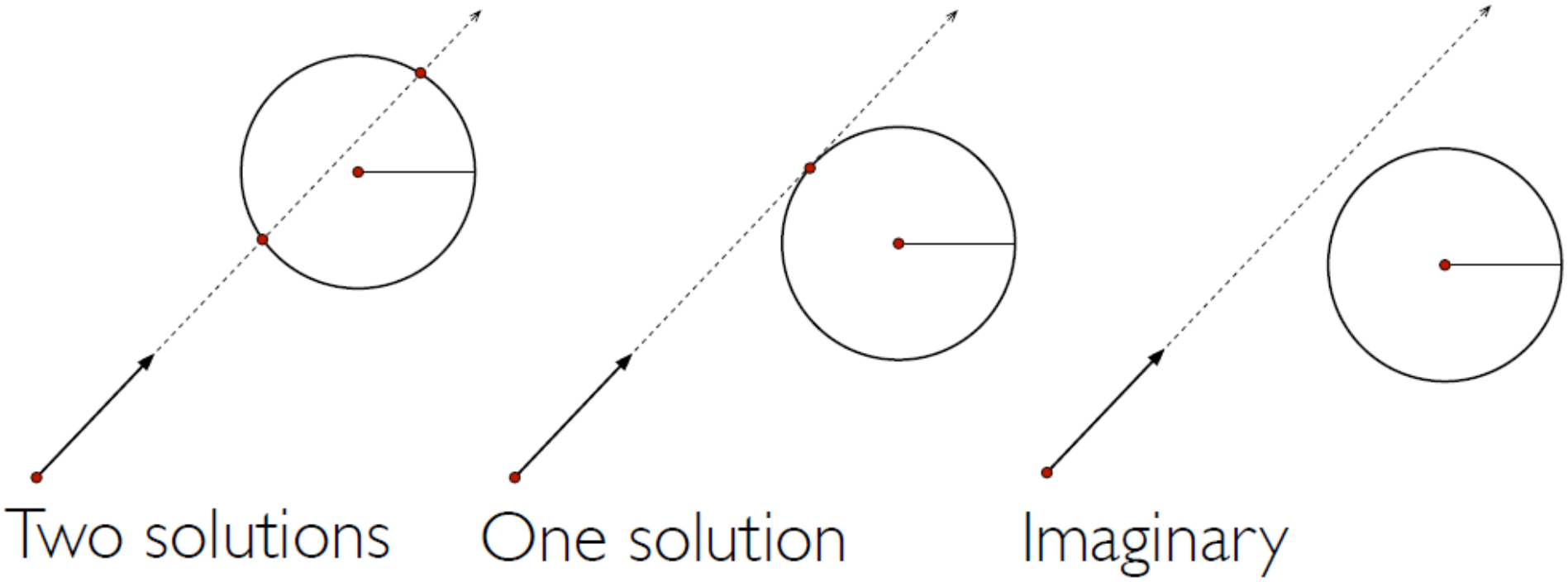
$$(\mathbf{d} \cdot \mathbf{d})t^2 + (2\mathbf{d} \cdot (\mathbf{o} - \mathbf{c}))t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2 = 0$$

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

Discriminant, $D = B^2-4^*A^*C$
Solutions must satisfy:
$t = (-B \pm \sqrt{(D)}) / 2A$

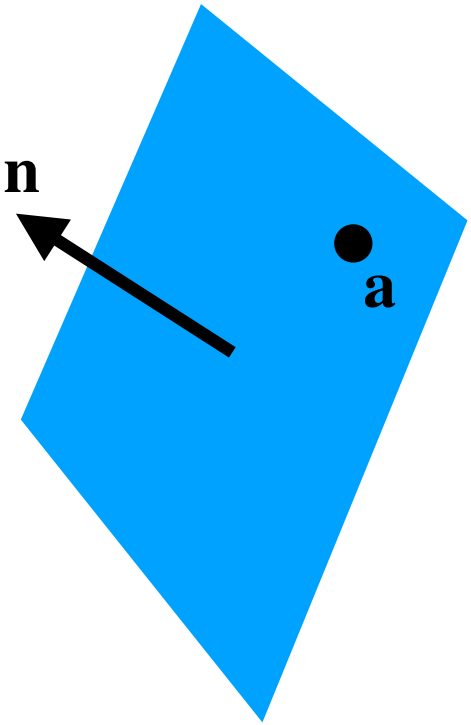
Ray-Sphere Intersection

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



Defining a Plane

- A point $\mathbf{p}$ that satisfies the following implicit form lives on a plane through point $\mathbf{a}$ that has normal $\mathbf{n}$



$$f(\mathbf{p}) = (\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0$$

- $f(\mathbf{p}) > 0$ lives on the “front” side of the plane (in the direction pointed to by the normal
- $f(\mathbf{p}) < 0$ lives on the “back” side

Ray-Plane Intersection

- 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 $t = ((\mathbf{a} - \mathbf{o}) \cdot \mathbf{n}) / (\mathbf{d} \cdot \mathbf{n})$