CSC 433/533
Computer Graphics
Algebra and Ray Shooting

Alon Efrat
Credit: Joshua Levine

What is a Vector?

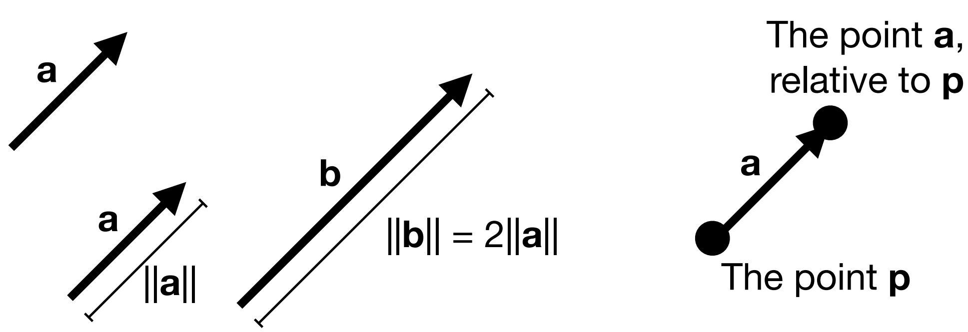
- A **vector** describes a length and a direction
- A vector is also a tuple of numbers
  - But, it often makes more sense to think in terms of the length/direction than the coordinates/numbers
  - And, especially in code, we want to manipulate vectors as objects and abstract the low-level operations
  - 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. (other notation: $\bar{a}$, $\overrightarrow{a}$)
- A vector's **length** is denoted with || ||, (sometimes we just denote . When **a** =(x,y), then $|\mathbf{a}| = \sqrt{a\,.\,x^2 + a\,.\,y^2}$
  - e.g. the length of **a** is ||**a**||
- A **unit vector** has length one
- The **zero vector** has length zero, and undefined direction

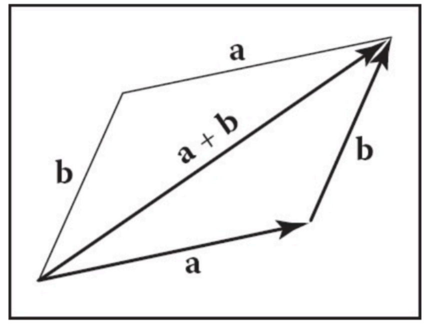
Vectors in Pictures

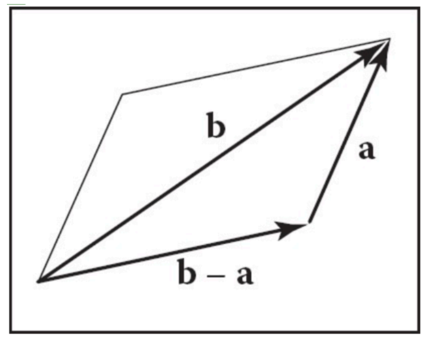
- We often use an arrow to represent a vector
  - The length of the arrow indicates the length of the vector, the direction of the arrow indicates the direction of the vector.
- The position of the arrow is irrelevant!
  - However, we can use vectors to represent positions by describing displacements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



Vector Operations

- Vectors can be added, e.g. for vectors **a**,**b**, there exists a vector **c** = **a**+**b**
  $\mathbf{a} + \mathbf{b} = (a\,.\,x + b\,.\,x,\quad a\,.\,y + b\,.\,y)$
- 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
  $\beta\mathbf{a} = (\beta a\,.\,x,\quad \beta a\,.\,y)$

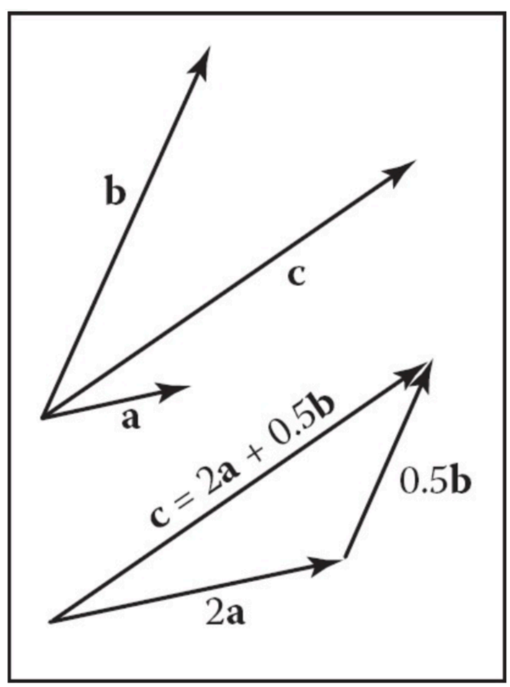




Vectors Decomposition

- 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**

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



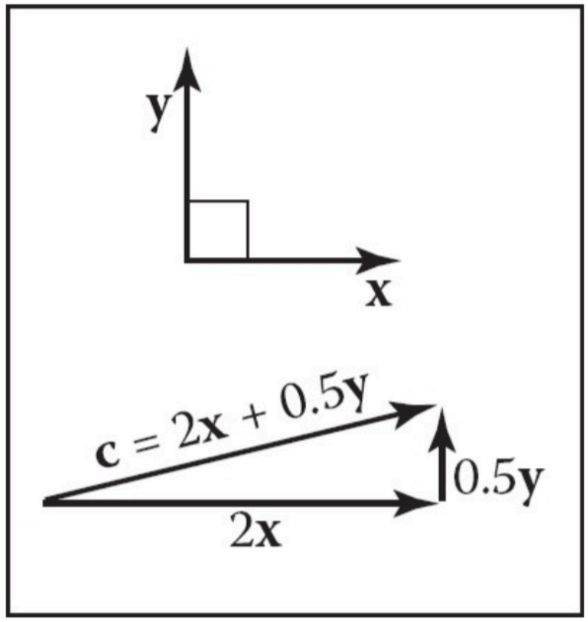
Canonical (Cartesian) Basis

- 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:

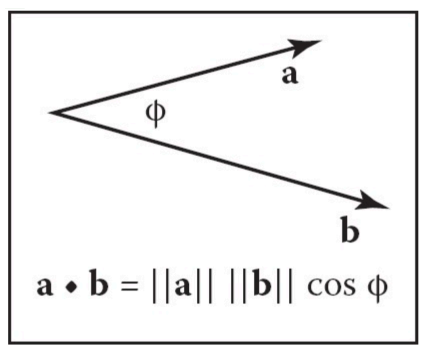
$$\mathbf{a} \cdot \mathbf{b} = (a.x \cdot b.x \;+\; a.y \cdot b.y)$$

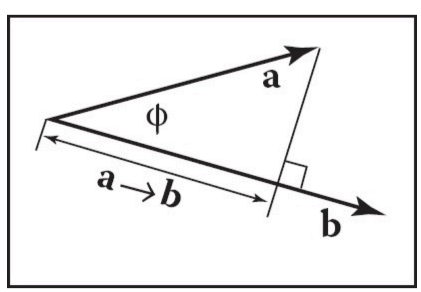
$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}||\;||\mathbf{b}|| \cos \phi$$

  - 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**

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





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:

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

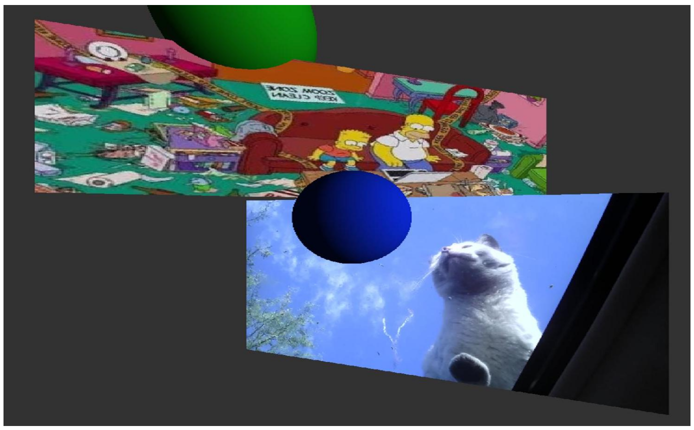
3D Vectors

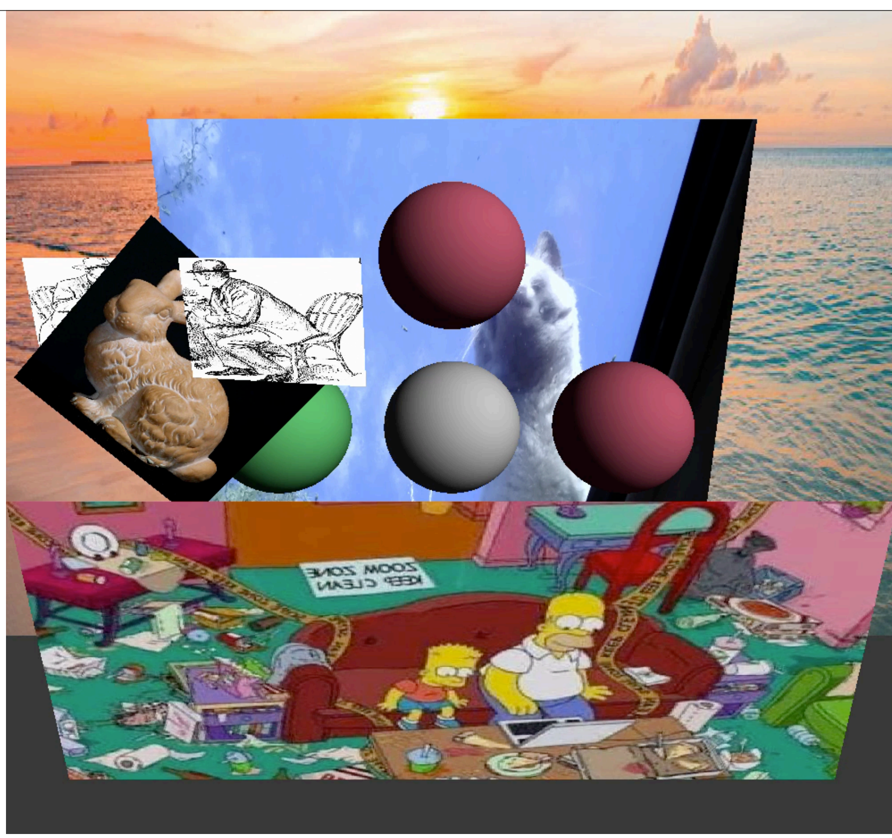
- Same idea as 2D, except these vectors are defined typically with a basis of three vectors
  - Still just a direction and a magnitude
  - But, useful for describing objects in three-dimensional space
- Most operations exactly the same, e.g. dot products:

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

Assignment 3. Balls and Billboards

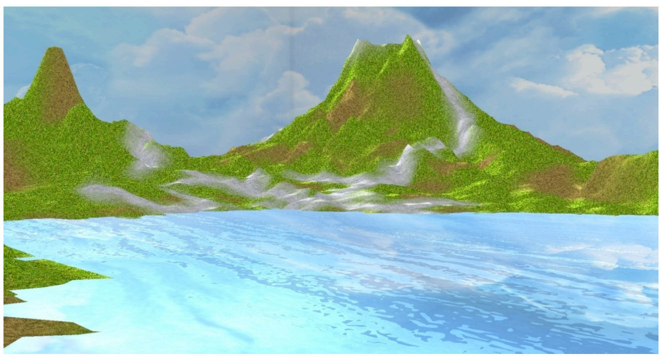
Input: JSON file describing locations of billboards and spheres.
Images placed on the billboards.
Output: scene showing what a viewer could see, and
A video showing camera movement





Billboards are extremely important for interactive computer graphics

- They could use as texture
- They could use as “imposer” of a very detailed huge geometric scene (e.g. the mountains at the background)
- The user could move (slightly) and not notice that the background mountains don’t move properly. Very small errors.

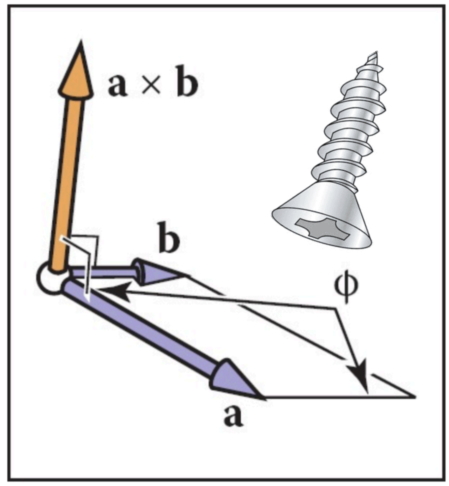


Each tree is its own billboard

- But if we render a tree on a billboard, why are the billboard not occluding each other ?
- We store at the data base a set of 2D images. Each shows the tree from a different directions.
- If the camera moves slightly, Small errors are not noticeable. Sometimes we need to switch with image with another

Cross Products

- In 3D, another way to “multiply” two vectors is the **cross product**, **a** × **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 **a** × **b** is always in the direction perpendicular (two possible answers).
- A screw turned from **a** to **b** will progress in the direction **a** × **b**
- Cross products distribute, but order matters:



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

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

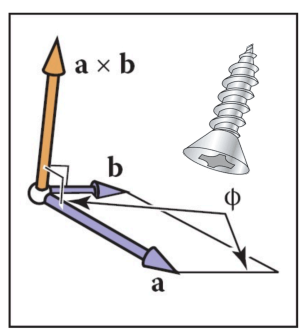
$$\mathbf{a} \times \mathbf{b} = (\underbrace{y_a z_b - z_a y_b}_{\textbf{x component}},\ z_a x_b - x_a z_b,\ \underbrace{x_a y_b - y_a x_b}_{\textbf{z component}})$$

Cross Products

- Since the cross product is always orthogonal to the pair of vectors, we can define our 3D Cartesian coordinate space with it:
- In practice though (and the book derives this), we use the following to compute cross products:

$$\begin{aligned} \mathbf{x} &= (1,0,0) \\ \mathbf{y} &= (0,1,0) \\ \mathbf{z} &= (0,0,1) \end{aligned}$$

$$\begin{aligned} \mathbf{x} \times \mathbf{y} &= +\mathbf{z}, \\ \mathbf{y} \times \mathbf{x} &= -\mathbf{z}, \\ \mathbf{y} \times \mathbf{z} &= +\mathbf{x}, \\ \mathbf{z} \times \mathbf{y} &= -\mathbf{x}, \\ \mathbf{z} \times \mathbf{x} &= +\mathbf{y}, \\ \mathbf{x} \times \mathbf{z} &= -\mathbf{y}. \end{aligned}$$



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

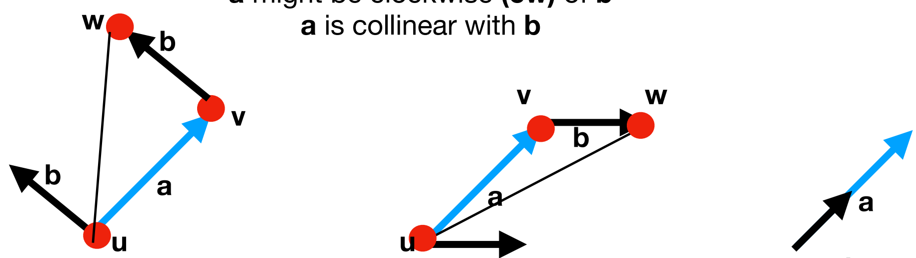
Checking orientation

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**



$x_a y_b - y_a x_b > 0$ | $x_a y_b - y_a x_b < 0$ | $x_a y_b - y_a x_b = 0$

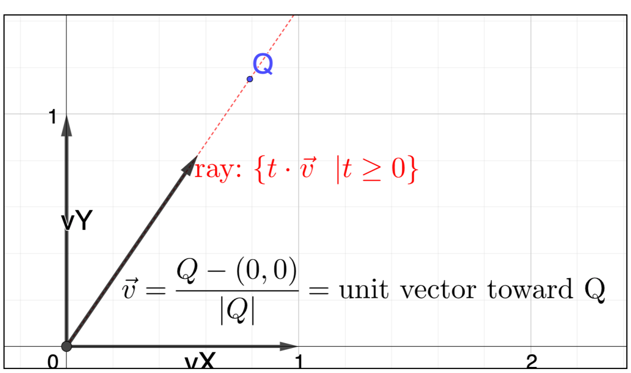
a is counter-clockwise **(ccw)** of **b**

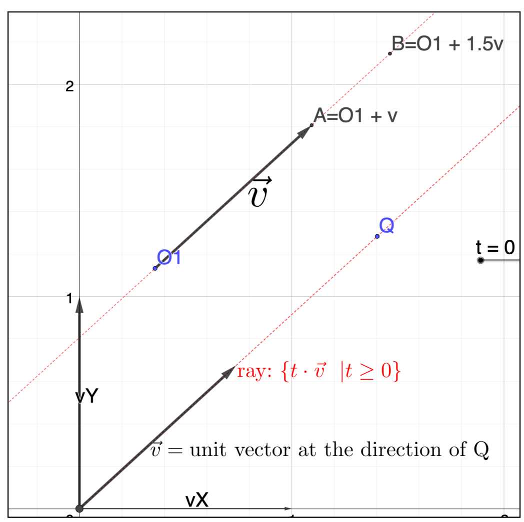
a is clockwise **(cw)** of **b**

a, b collinear

This will provide a convenient way to check if a triangle with vertices u,v,w (when vertices are given to us in this order) is CCW or CW

Rays, lines, Orthogonal Projections





The ray $\{t \cdot \overrightarrow{v} \mid t \geq 0\}$

The line that $\overrightarrow{v}$ **defines is**

$\ell = \{t \cdot \overrightarrow{v} \mid v \in \mathbb{R}\}$

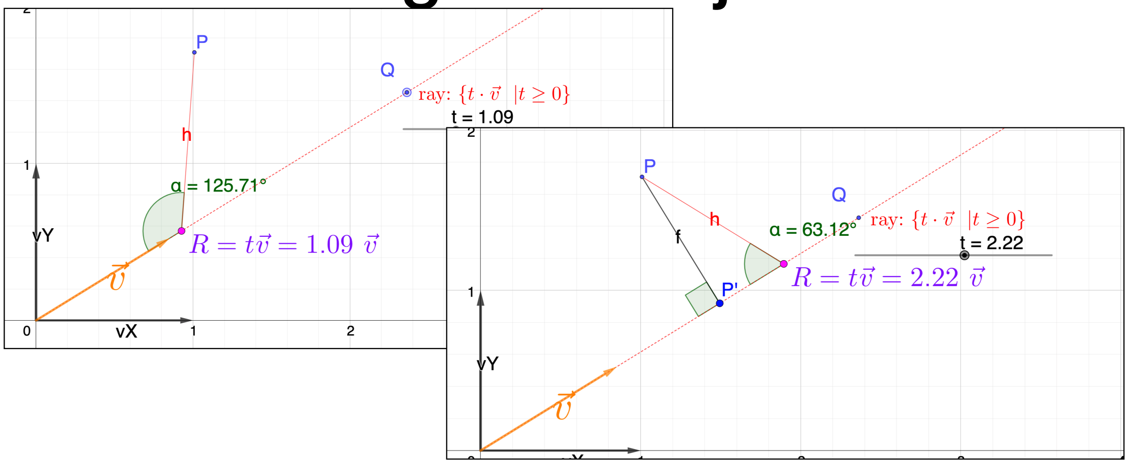
(that is, t **is any real value**

The ray $\{O1 + t \cdot \overrightarrow{v} \mid t \geq 0\}$

This is the same ray, shifted by $O1$

That is, the ray emerges from $O1$

Orthogonal Projections



- Let P be a point not on the ray
- Need to find: The point P' which is the orthogonal projection of P on $\ell = \{t\overrightarrow{v} \,|\, t \in \mathbb{R}\}$
- P' is the closest point on ℓ to P
- Assume t start at zero, and slowly increases. Let $R = t \cdot \overrightarrow{v}$. Monitor the angle $\angle(O, R, P')$. At some time t_0, this angle is 0, and R and P' coincide, and $\angle(O, R, P') = 0$. This means:

$$(t_0 \cdot \overrightarrow{v}) \perp (P - t_0\overrightarrow{v}) \Rightarrow$$
$$(t_0 \cdot \overrightarrow{v}) \cdot (P - t_0\overrightarrow{v}) = 0$$
$$t_0 v \cdot P = t_0^2(\overrightarrow{v} \cdot \overrightarrow{v})$$
$$\Leftrightarrow t_0 = P \cdot \overrightarrow{v}$$
$$\Rightarrow P' = (P\overrightarrow{v})\overrightarrow{v}$$

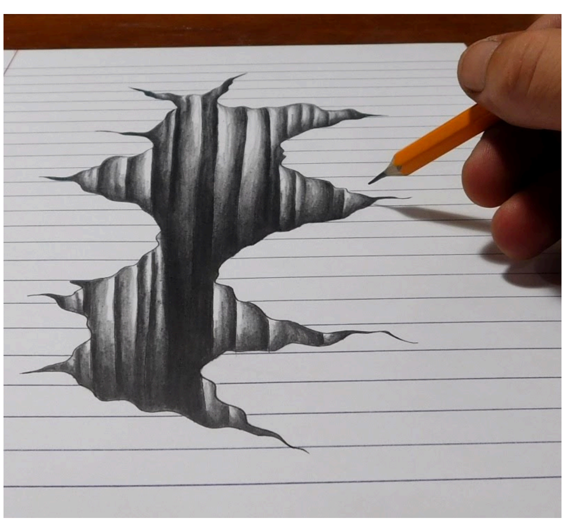
Rendering

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**

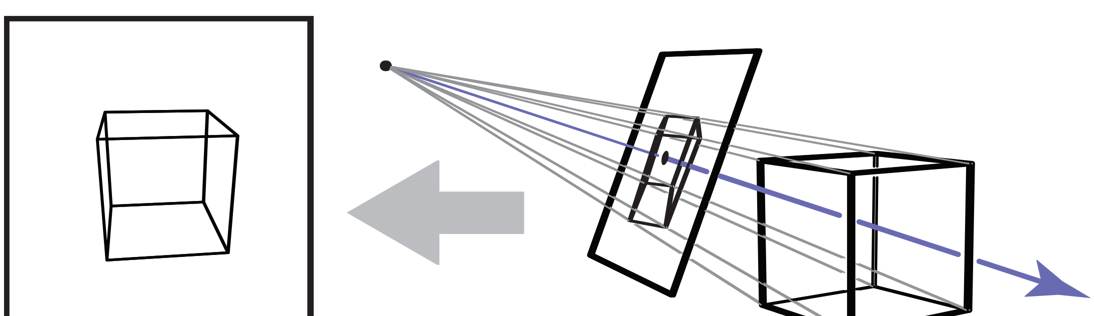
```
for each image pixel {
  for each object {
    if (object affects pixel)
    {
      do something
    }
  }
}
```

TODAY

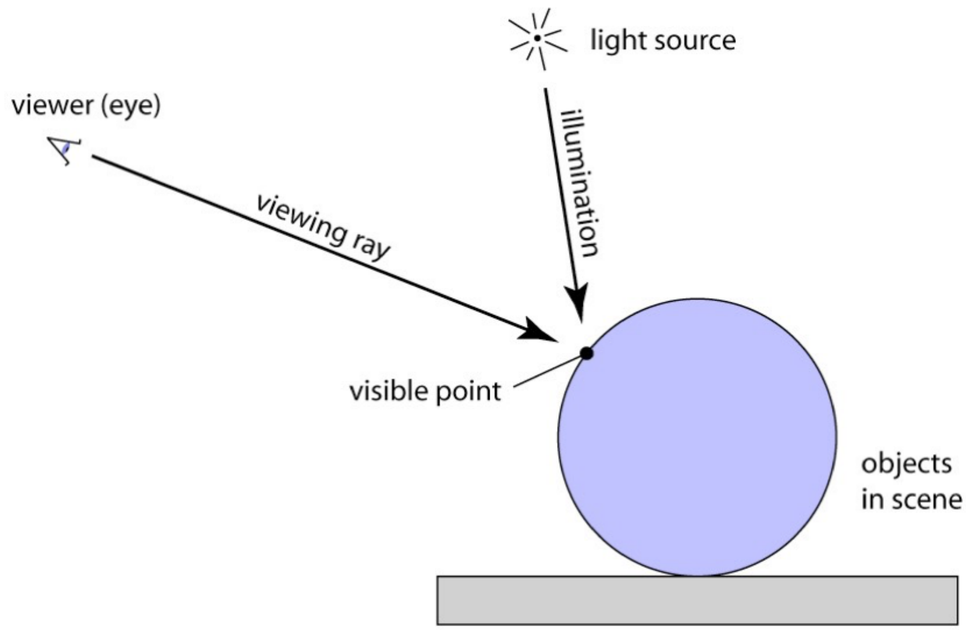
Basics of Ray Tracing

Idea of Ray Tracing

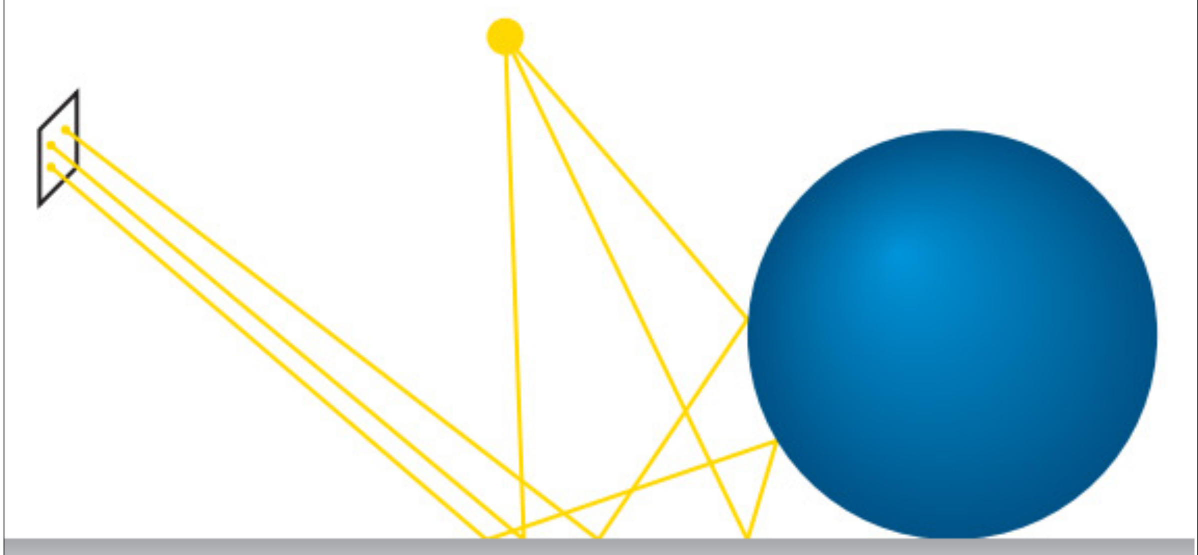
- Ask first, for each pixel: what belongs at that pixel?
- Answer: The set of objects that are visible if we were standing on one side of the image looking into the scene



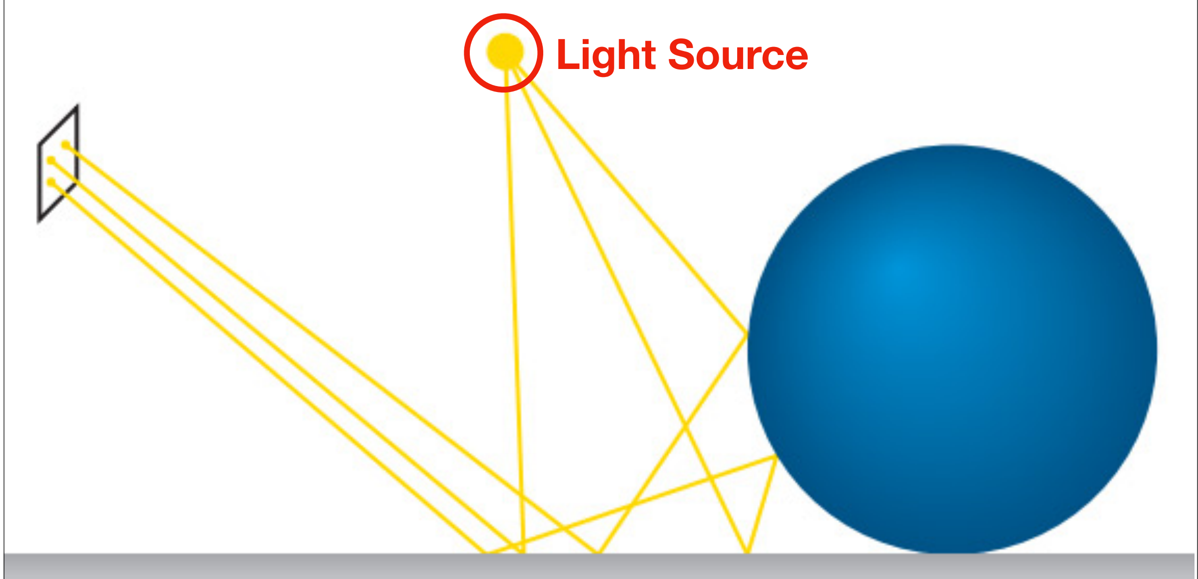
Key Concepts, in Diagram



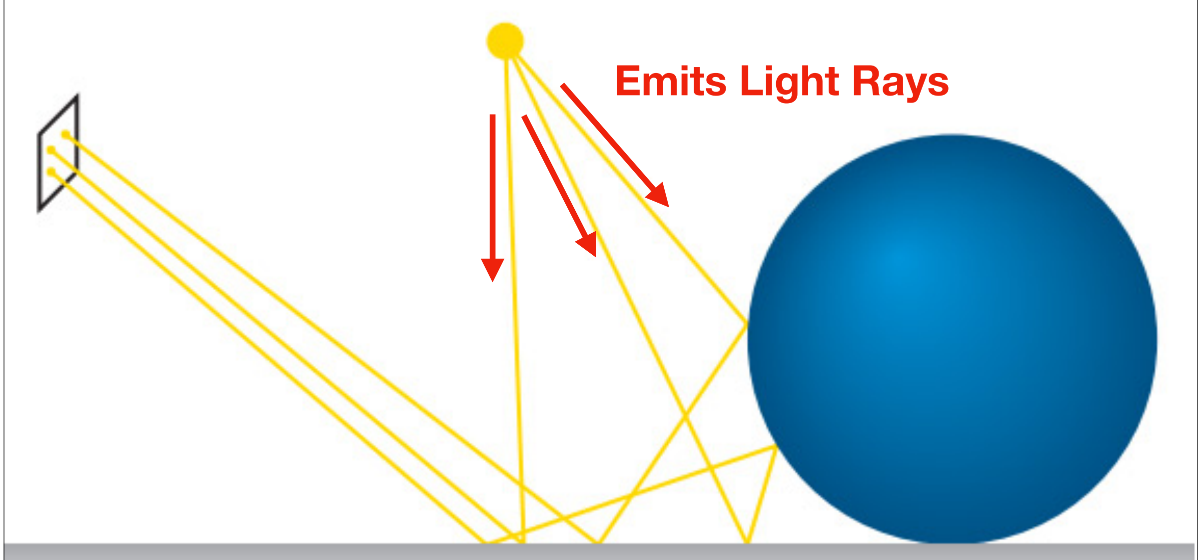
Idea: Using Paths of Light to Model Visibility



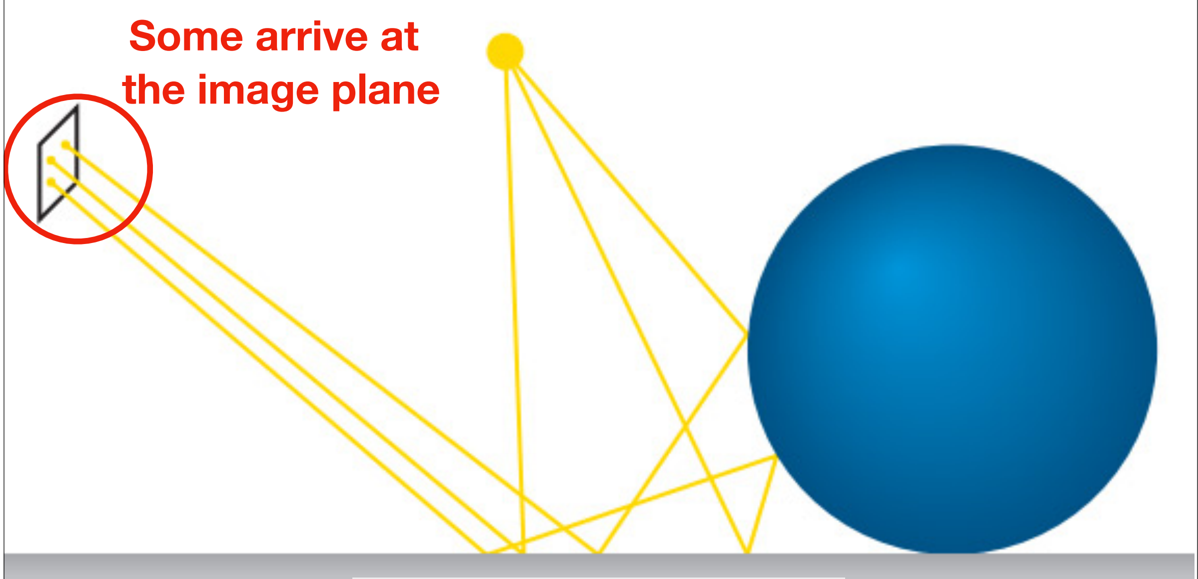
Using Paths of Light to Model Visibility



Using Paths of Light to Model Visibility



Using Paths of Light to Model Visibility



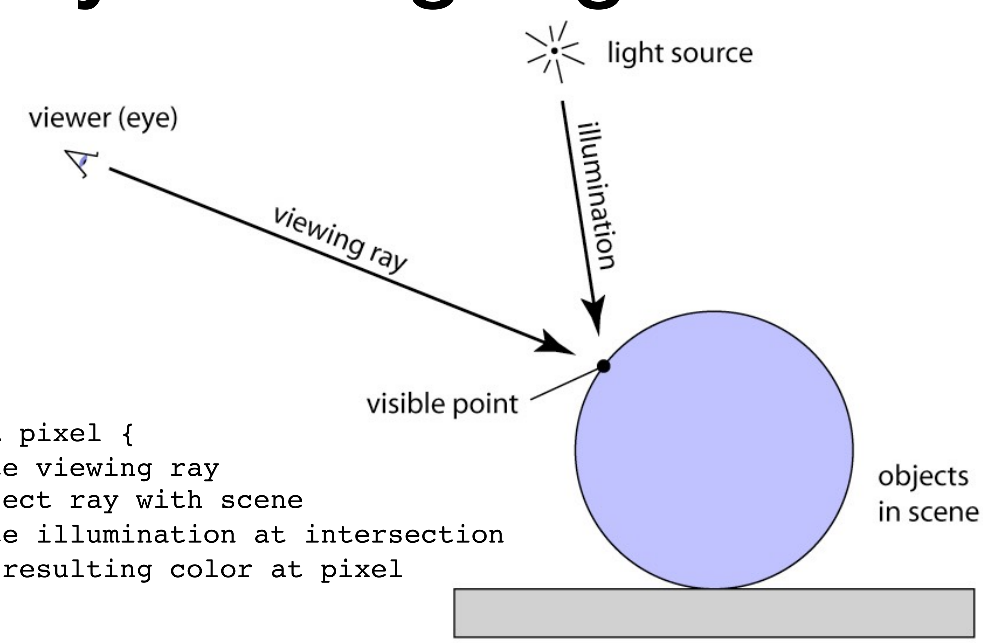
https://software.intel.com/file/37491

Using Paths of Light to Model Visibility

Forwarding vs Backward Tracing

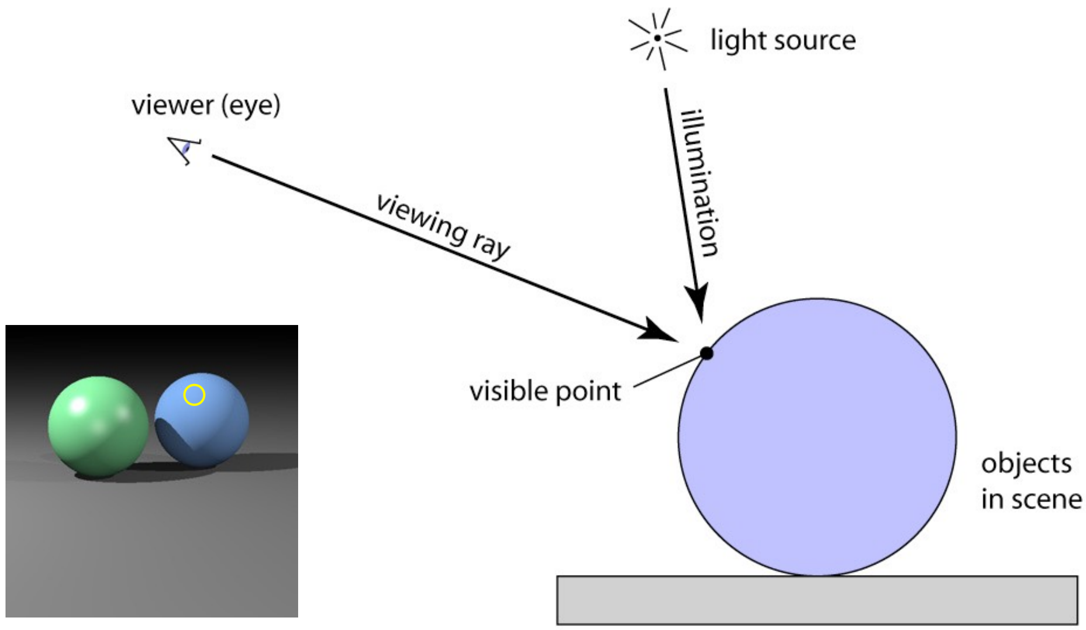
- Idea: Trace rays from light source to image
  - This is slow!
- Better idea: Trace rays **<u>from</u>** image **<u>to</u>** light source

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

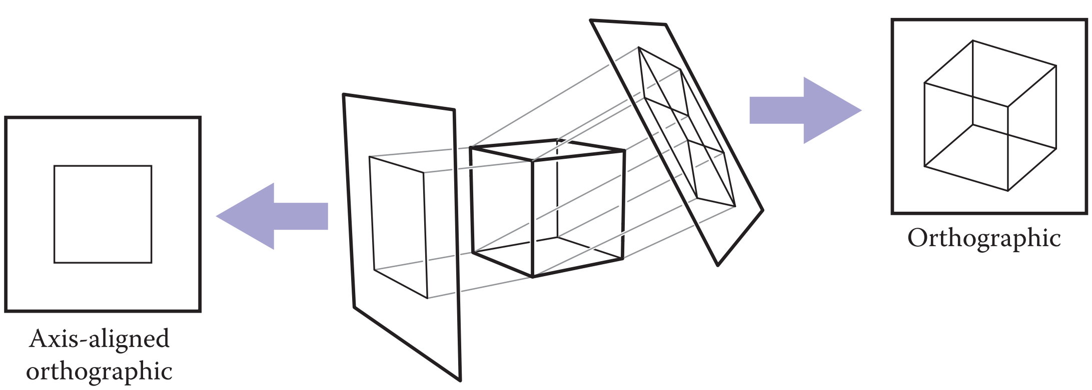
```
for each pixel {
  compute viewing ray
  intersect ray with scene
  compute illumination at intersection
  store resulting color at pixel
}
```

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

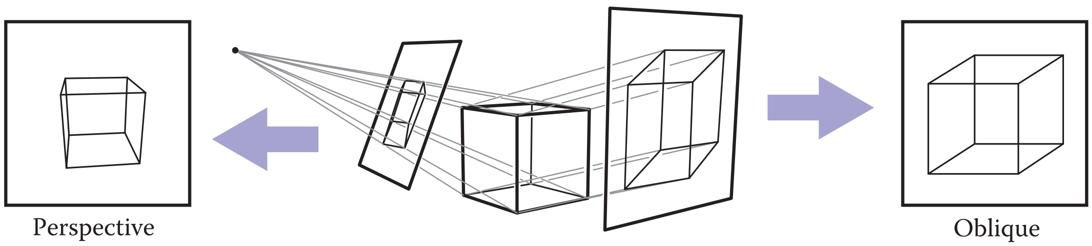
Orthographic Views

- 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



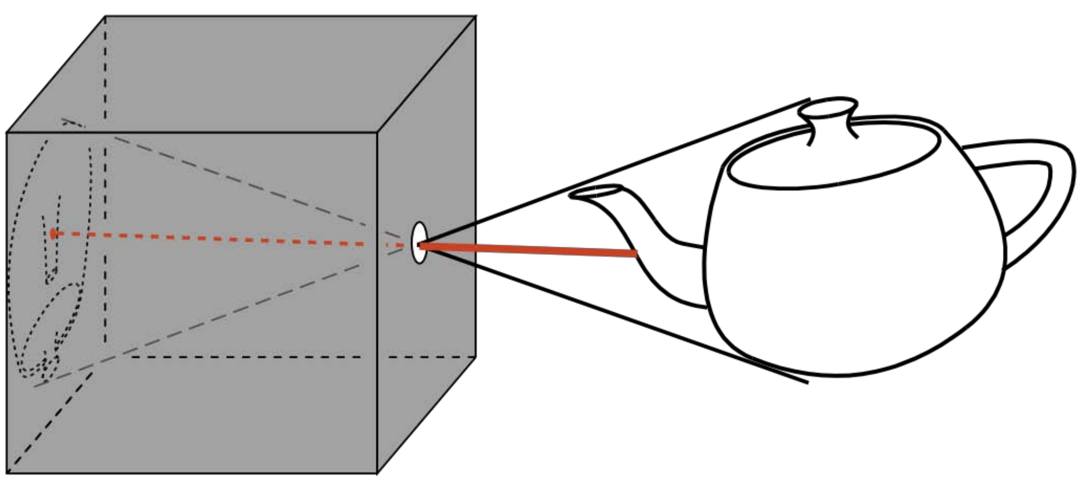
Perspective Views

- 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



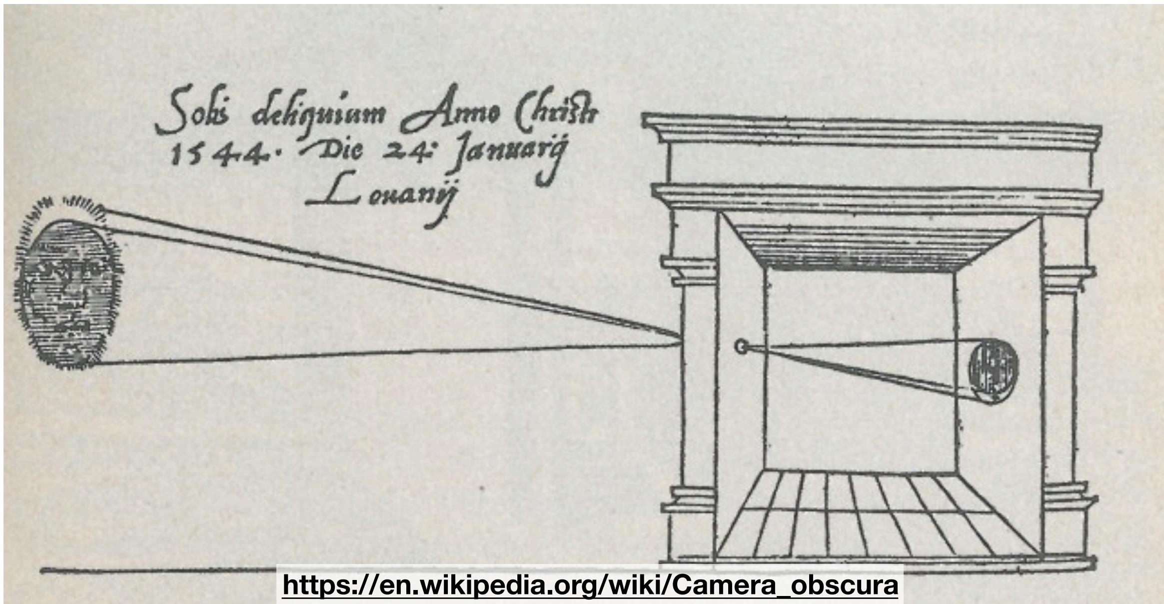
Pinhole Cameras

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



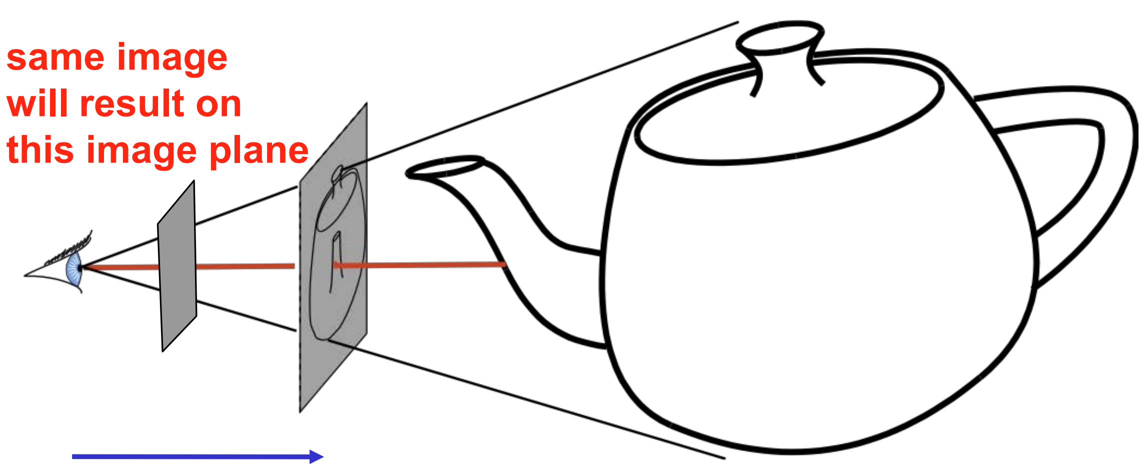
Camera Obscura

- Gemma Frisius, 16th century



Simplified Pinhole Cameras

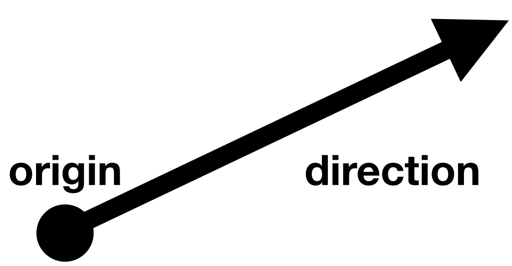
- Instead, we can place the eye at the pinhole and consider the eye-image pyramid (sometimes called **view frustum**)



Defining Rays

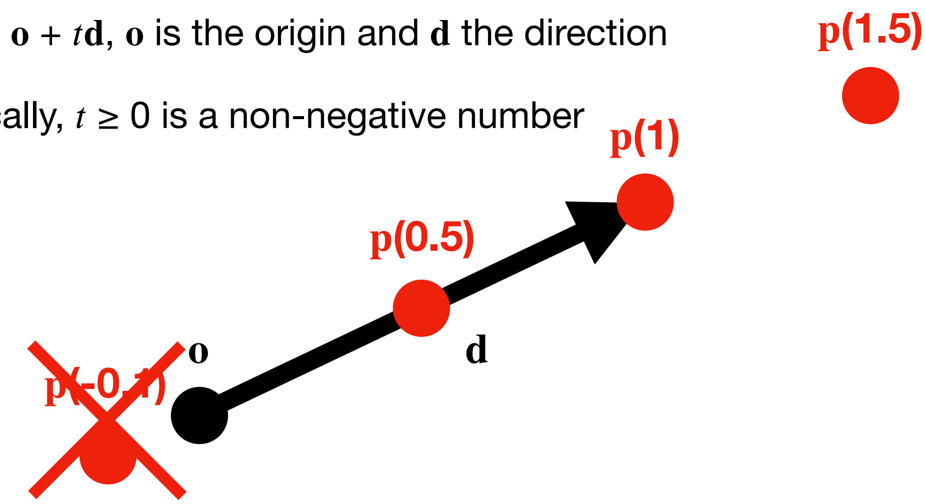
Mathematical Description of a Ray

- 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



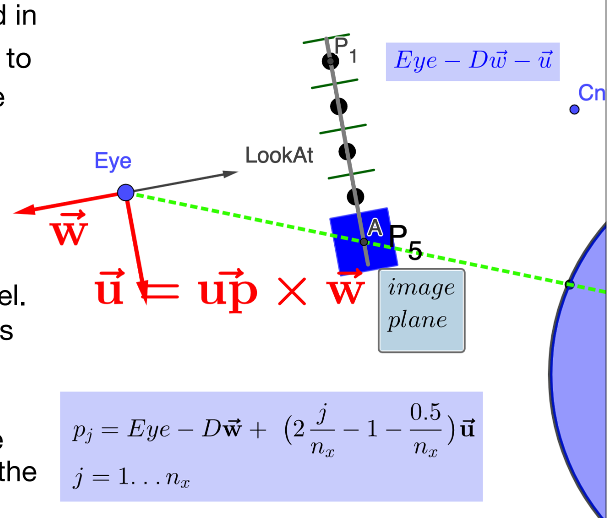
Mathematical Description of a Ray

- 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

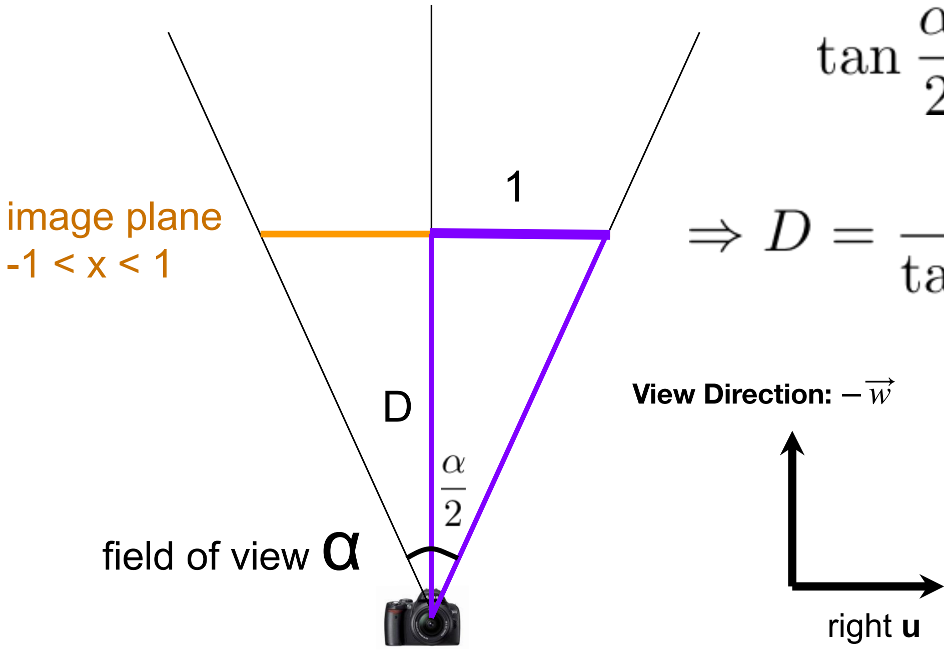


The Plan (high level)

- Given camera parameters (details later), and n_x, n_y , the number of pixels in a row, and in column, of the rendered image, we need to generate $n_x \times n_y$ rays, emerging from the camera.
- To create the rays, we will need a set of witness points $p_{i,j}$ All in the image plane. Each witness point is in a center of a pixel. Shoot a ray from the EYE to each witness point.
- For each ray, find what is the color of the first object it hits, and copy this color to the corresponding pixel.



Ray Generation in 2D



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

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

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 LookAt (in hw3, you are given a center that you are looking at. It is a point in the scene)
- Up vector (note that is not necessarily the “up” of the geometric scene

Building coordinates system

- $\overrightarrow{LookAt} = \dfrac{\overrightarrow{Center} - Eye}{\|\overrightarrow{Center} - Eye\|}$
- $\vec{\mathbf{w}} = -\overrightarrow{LookAt}$ - it is a unit vector pointing backward (toward the viewer)
- $\vec{u} = \overrightarrow{Up} \times \vec{w}$. Vector point right from the eye. Make sure to normalized
- $\vec{v} = \vec{w} \times \vec{u}$
- The segment $(Eye, Center)$ is orthogonal to the image plane, and pass via the middle of the image plane

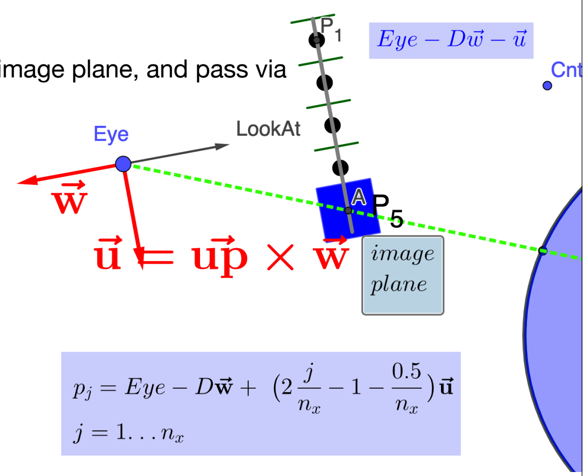
Where is the point $Eye - D\vec{\mathbf{w}}$ **?**

Witness points (first in 2D):

$$p_j = Eye - D\vec{\mathbf{w}} + \left(2\frac{j}{n_x} - 1 - \frac{0.5}{n_x}\right)\vec{\mathbf{u}}$$

$j = 1,2,\ldots$**#columns**

Ray r: $r = Eye + t(p_i - Eye)$



Now in 3D

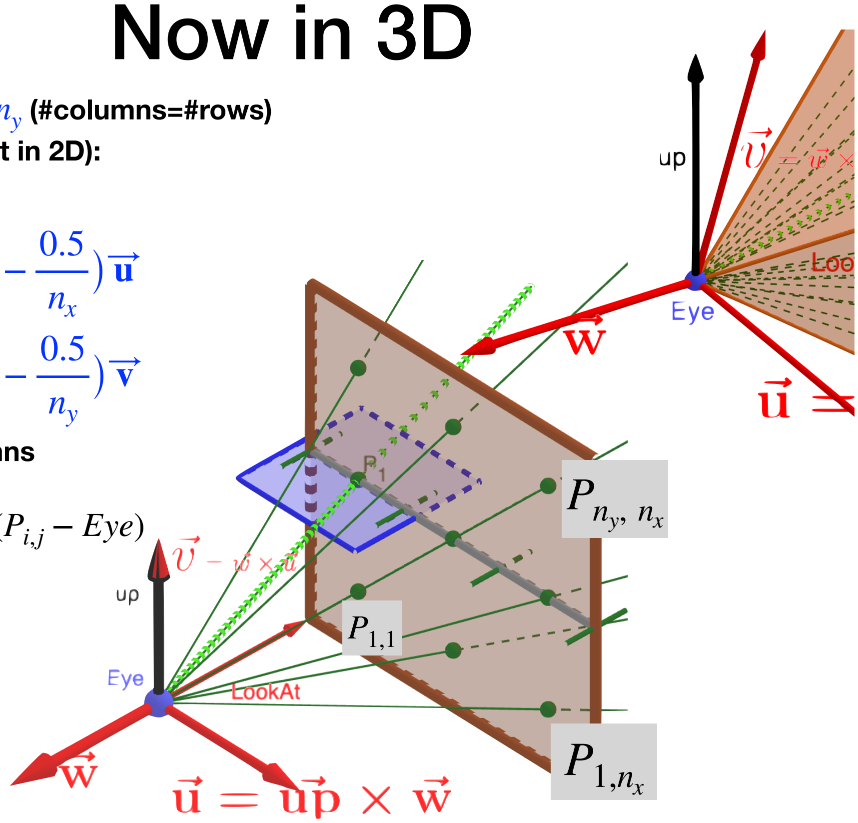
Assume first $n_x = n_y$ **(#columns=#rows)**

Witness points (first in 2D):

$$P_{i,j} = Eye - D\vec{\mathbf{w}} + \left(2\frac{j}{n_x} - 1 - \frac{0.5}{n_x}\right)\vec{\mathbf{u}} + \left(2\frac{i}{n_y} - 1 - \frac{0.5}{n_y}\right)\vec{\mathbf{v}}$$

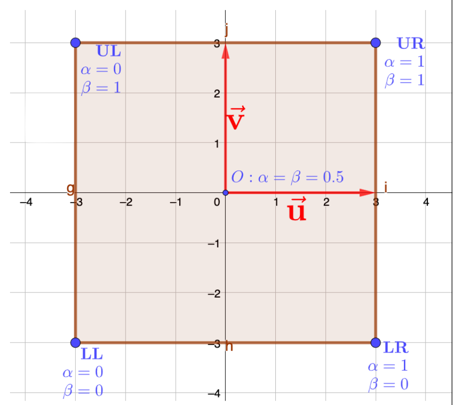
$i, j = 1,2,\ldots$**#columns**

Ray r: $r = Eye + t(P_{i,j} - Eye)$



Here is systematic way to develop these formulas (you will have multiple opportunities in this course to use similar tricks

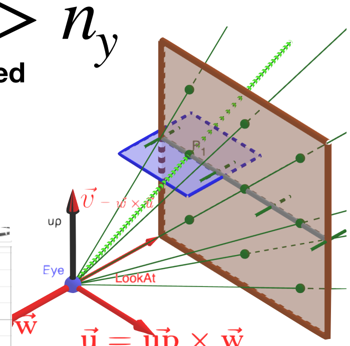
- Canonical representation:
- Each point in the image could be represented by coordinates (α, β). The lower left (LL) is $\alpha = \beta = 0$, That is $LL = O - \vec{u} = \vec{v}$
- And the lower right (LR) is $\alpha = 1,\ \beta = 0.$
- By linear interpolation $P(\alpha, \beta) = O + (2\alpha - 1)\vec{u} + (2\beta - 1)\vec{v}$
- Observe that $|\vec{u}| = |\vec{v}| = 1,$ and the size of a pixel is $\dfrac{2}{n_x} \times \dfrac{2}{n_y}$
- At this point, we remember that the image consists of $n_x \times n_y$ pixels. Referring to the LL corner of each pixel, we could transform the canonical representation to image representation by setting $\alpha = j/n_x,\ \ \beta = i/n_y$. Substitute, we obtain
- $P(i,j) = O + \left(\frac{2j}{n_x} - 1\right)\vec{u} + \left(\frac{2i}{n_y} - 1\right)\vec{v}$
- Finally, if you index the image $p_1, p_2 \ldots p_n$, then subtract half a pixel.

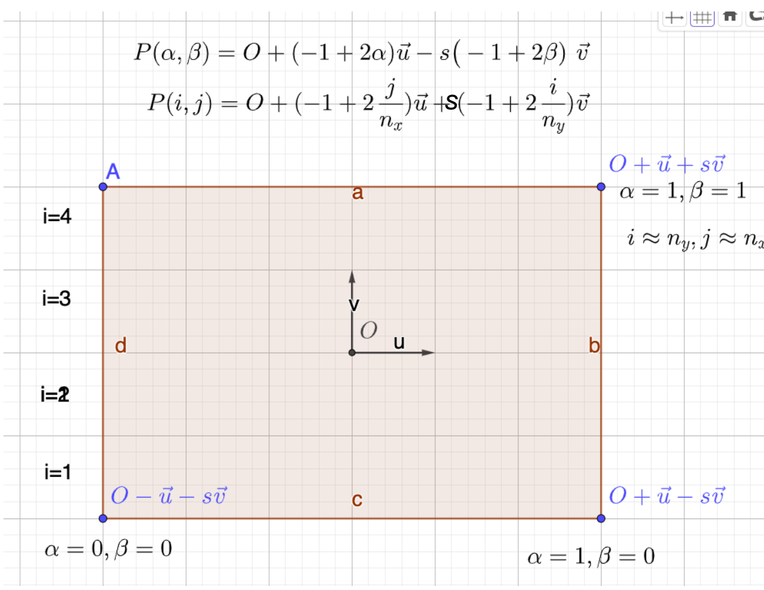


- Finally, if you index the image $p_1, p_2 \ldots p_n$, then subtract half a pixel. $P(i,j) = Eye - D\vec{w} + \left(\frac{2j-1}{n_x} - 1\right)\vec{u} + \left(\frac{2i-1}{n_y} - 1\right)\vec{v}$
- If you index $p_0, p_2 \ldots p_{n-1}$, the add a half a pixel $P(i,j) = Eye - D\vec{w} + \left(\frac{2j+1}{n_x} - 1\right)\vec{u} + \left(\frac{2i+1}{n_y} - 1\right)\vec{v}$

Now in 3D - the case $n_x > n_y$

Assume that each pixel is still a square. So the generated image, $width > height$**. Let** $s = n_y / n_x$





Intersecting Objects

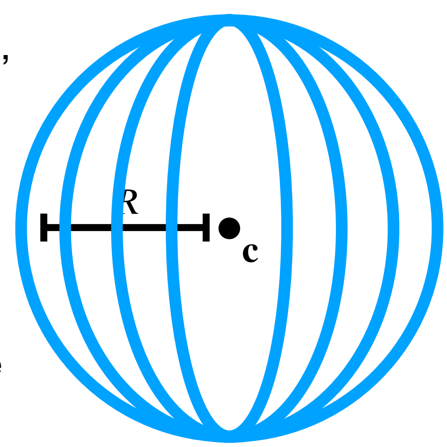
```
for each pixel {
  compute viewing ray
  intersect ray with scene
  compute illumination at intersection
  store resulting color at pixel
}
```

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:

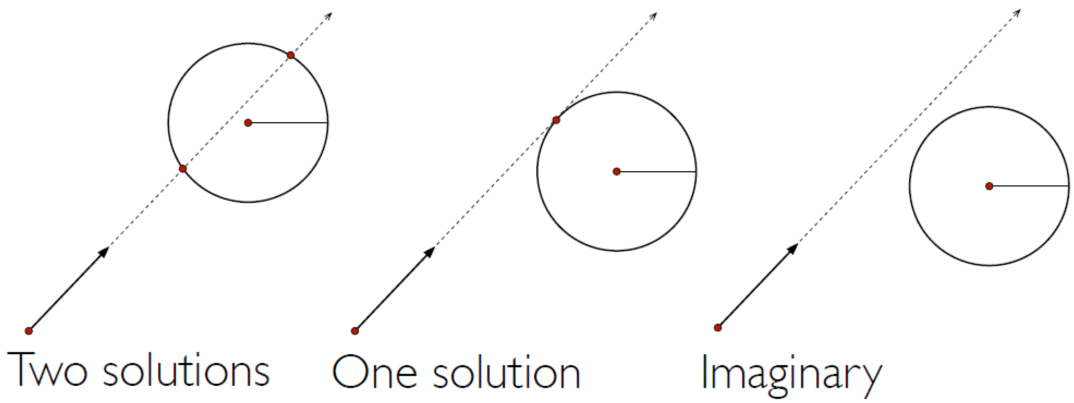
$$(\mathbf{d} \cdot \mathbf{d})t^2 + (2\mathbf{d} \cdot (\mathbf{o} - \mathbf{c}))t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2 = 0$$

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

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

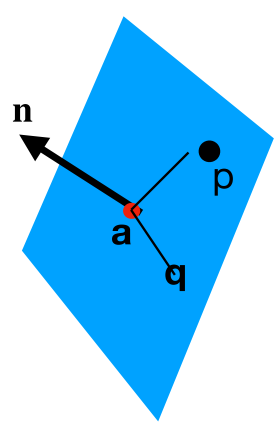
Ray-Sphere Intersection

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



Defining a Plane

- Let h be a plane with normal **n,** and containing a point **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 **p n** > **a n** then **p** lives on the "front" side of the plane (in the direction pointed to by the normal
- **p n-an** < 0 means that **p** lives on the "back" side.
- Sometimes used as **f(p)=0** iff ``p on the plane". So the function f(p) is **f(p)=(p-a)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 $f(\mathbf{p}(t))$:

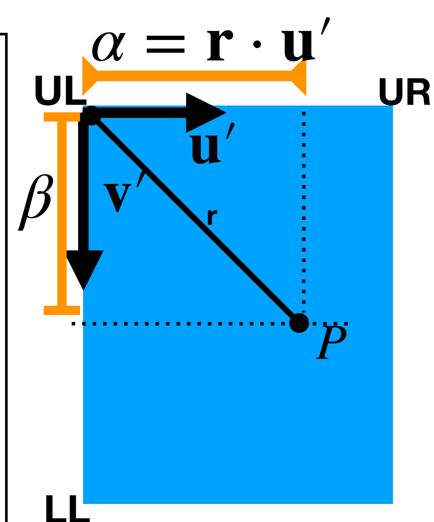
$$(\mathbf{o} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} = 0$$

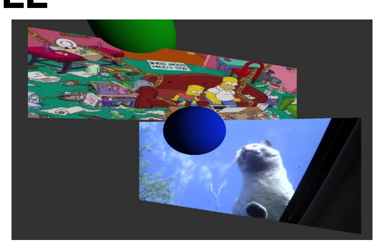
- This means that $t_{hit} = ((\mathbf{a} - \mathbf{o}) \cdot \mathbf{n}) / (\mathbf{d} \cdot \mathbf{n})$. The intersection point is $\mathbf{o} + t_{hit}\mathbf{d}$

Finding the color of a point on a billboard

- For each billboard, you will be given 3 corners (UL,UR,LL)
- Let **u',v'** be orthonormal vectors (orthogonal and unit length). $\mathbf{u}' = \frac{UR - UL}{||UR - UL||}$
- Let P be a point on the plane containing the billboard. Let $\mathbf{r} = P - UL$.
- **Let** $\alpha = \mathbf{r} \cdot \mathbf{u}'$
- α is the length of the projection of **r** on **u'** .
- Other words. "shadow" that **r** casts on the line containing **u'**
- We can use α, β to determine if P is in the billboard, (how), and if yes, find the pixel of the image of the billboard at P.

$$\mathbf{P} = \mathbf{UL} + \underbrace{(\mathbf{r} \cdot \mathbf{u}')}_{\alpha}\,\mathbf{u}' + \underbrace{(\mathbf{r} \cdot \mathbf{v}')}_{\beta}\,\mathbf{v}'$$





Constructing Orthonormal Bases from a Pair of Vectors

- Given two vectors **a** and **b**, which might not be orthonormal to begin with:

$$\begin{aligned} \mathbf{w} &= \frac{\mathbf{a}}{||\mathbf{a}||}, \\ \mathbf{u} &= \frac{\mathbf{b}\times\mathbf{w}}{||\mathbf{b}\times\mathbf{w}||}, \\ \mathbf{v} &= \mathbf{w}\times\mathbf{u}. \end{aligned}$$

- In this case, **w** will align with **a** and **v** will be the closest vector to **b** that is perpendicular to **w**