Rotations and Projective Space

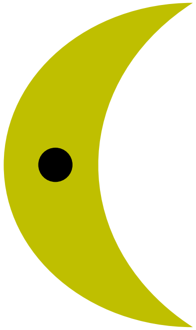
Rotations (in 2D)

Spins **points** about origin; **vectors** about their tails

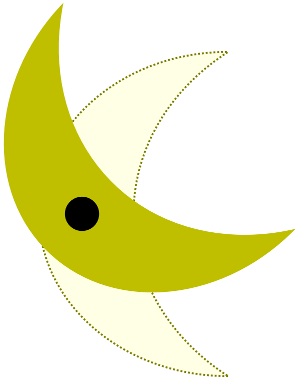
Rotation vs Orientation

Rotation is a **transformation**

Rotation of **rest pose** is **orientation**

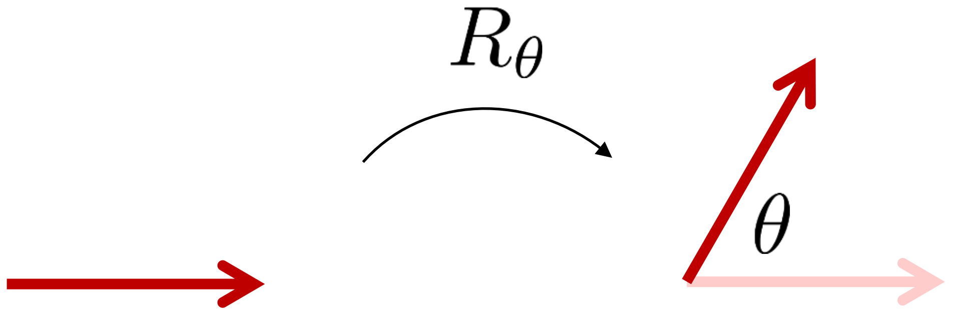


rest pose



Rotations (in 2D)

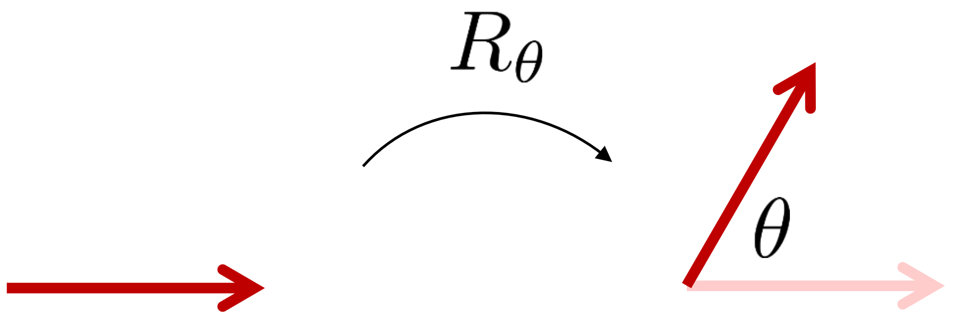
In 2D rotations are simple: parameterized by single **angle** θ



Rotations (in 2D)

In 2D rotations are simple: parameterized by single **angle** θ

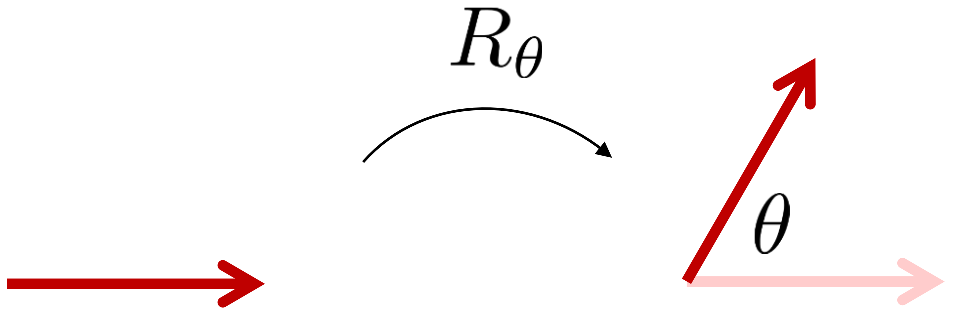
- "one dimensional"



Rotations (in 2D)

In 2D rotations are simple: parameterized by single **angle** θ

- "one dimensional"
- periodic

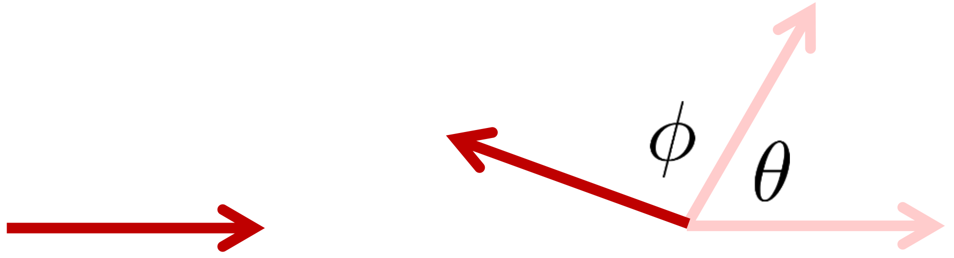


Rotation Group

2D rotations form a **group** called SO(2)

- compositions of rotations are rotations

$$R_\phi R_\theta = R_{\phi+\theta}$$



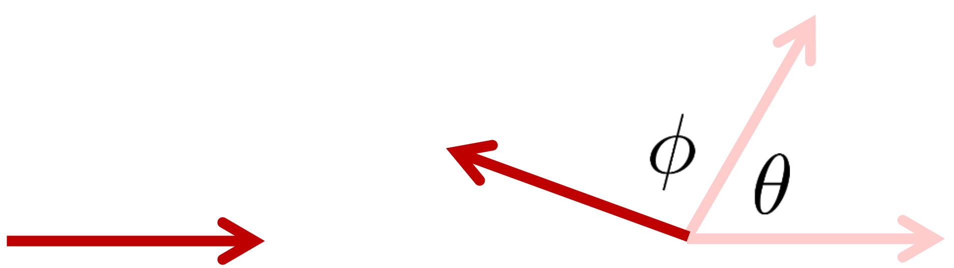
Rotation Group

2D rotations form a **group** called SO(2)

- compositions of rotations are rotations

$$R_\phi R_\theta = R_{\phi+\theta}$$

- rotations have inverses $R_{-\theta}$



Rotation Matrices (in 2D)

Rotations are linear (why)?

Rotation Matrices (in 2D)

Rotations are linear (why)?

Rotations can be represented by matrix (why)?

Rotation Matrices (in 2D)

Rotations are linear (why)?

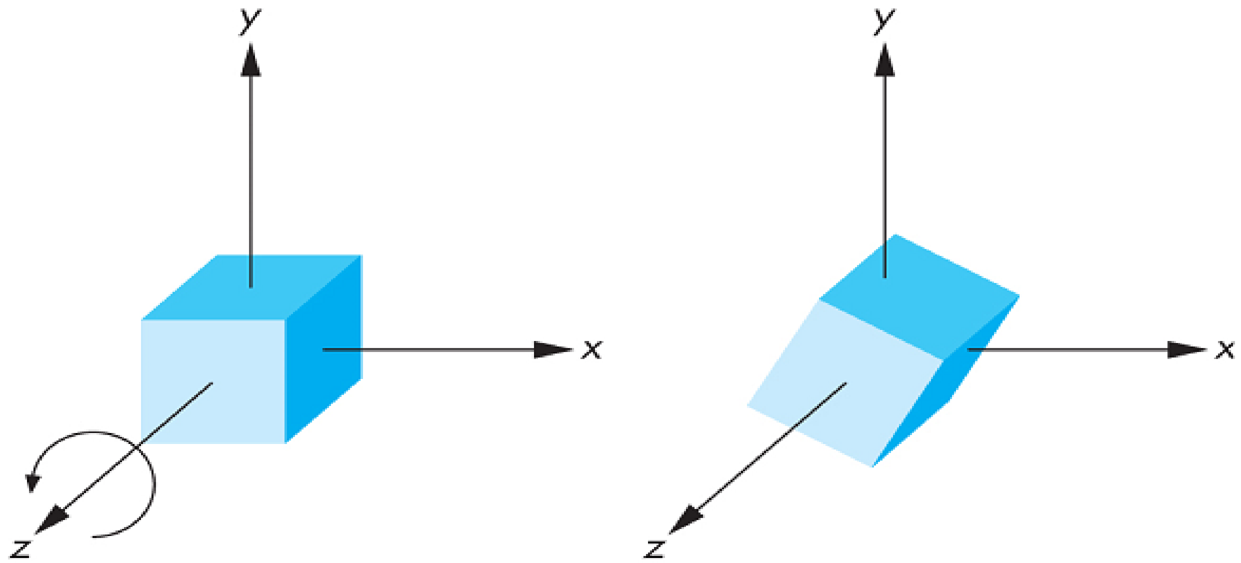
Rotations can be represented by matrix (why)?

Formula: $R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

3D Rotations and Orientations

3D rotations still linear, still form group SO(3)

- **rotations don't commute!!**

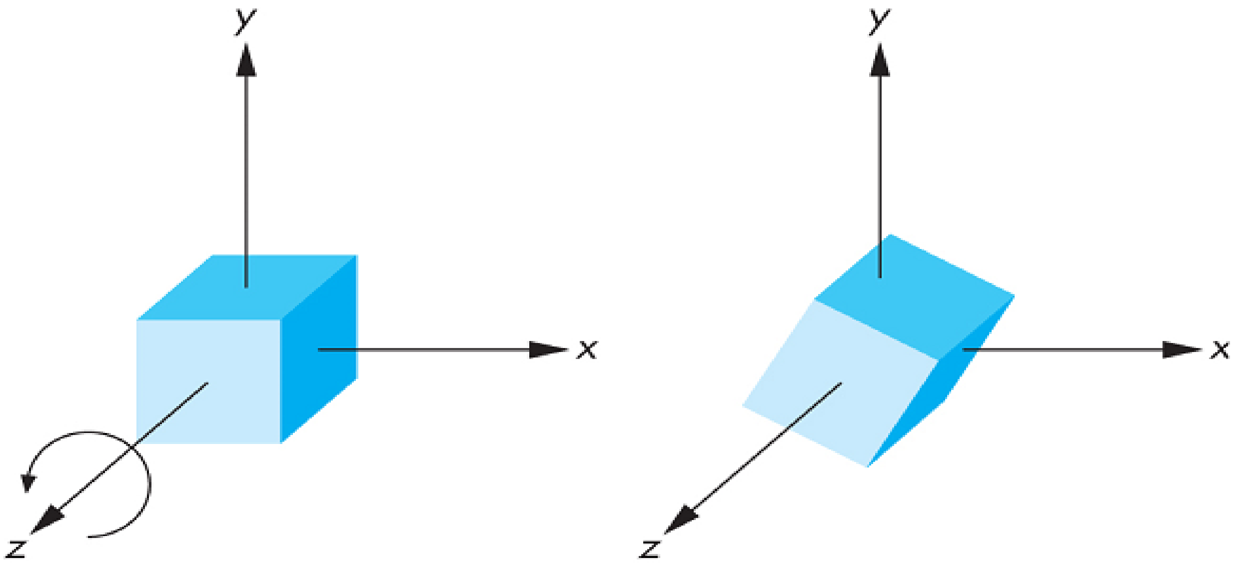


3D Rotations and Orientations

3D rotations still linear, still form group SO(3)

- **rotations don't commute!!**

But compared to 2D, very tricky



What We Want To Do With Rots

1. Write them down (represent them)

What We Want To Do With Rots

1. Write them down (represent them)

2. Use them to rotate points/vectors

What We Want To Do With Rots

1. Write them down (represent them)

2. Use them to rotate points/vectors

3. Compose them

What We Want To Do With Rots

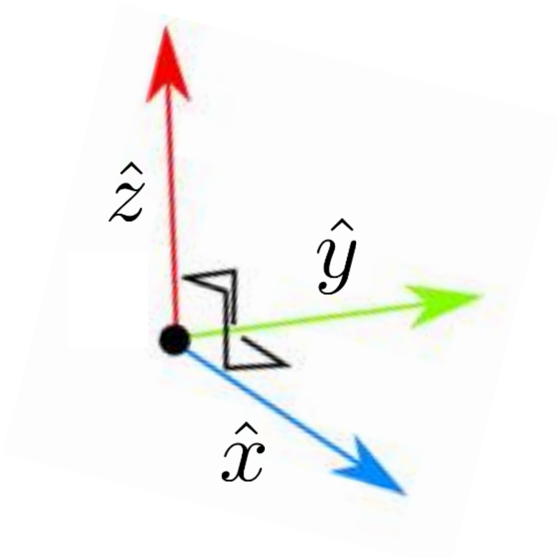
1. Write them down (represent them)

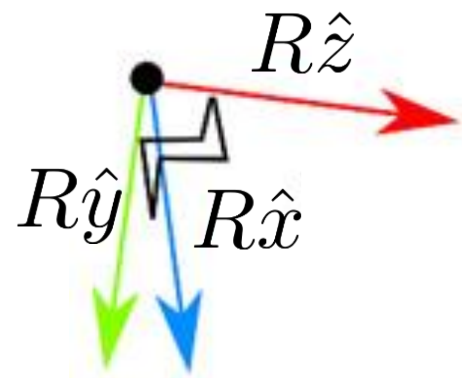
2. Use them to rotate points/vectors

3. Compose them

4. Interpolate them
    - interpolate: smoothly blend

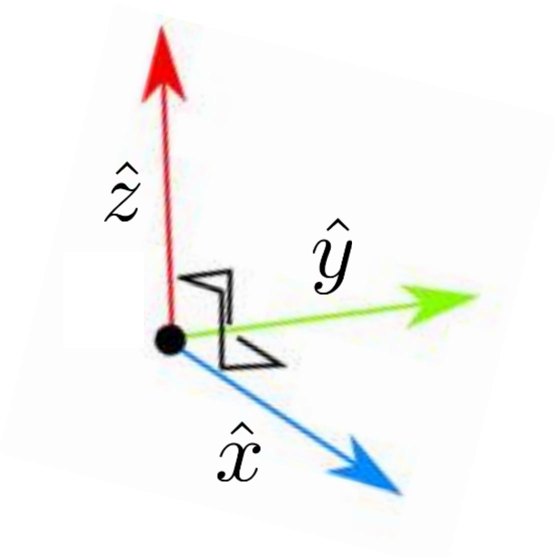
Representation 1: Frames

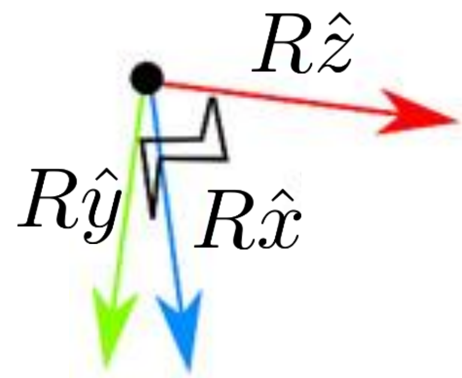




Three **orthogonal**, **unit** vectors $R\hat{x}, R\hat{y}, R\hat{z}$

Representation 1: Frames





Three **orthogonal**, **unit** vectors $R\hat{x}, R\hat{y}, R\hat{z}$

Actually only need two: $R\hat{z} = R\hat{x} \times R\hat{y}$

Representation 1: Frames

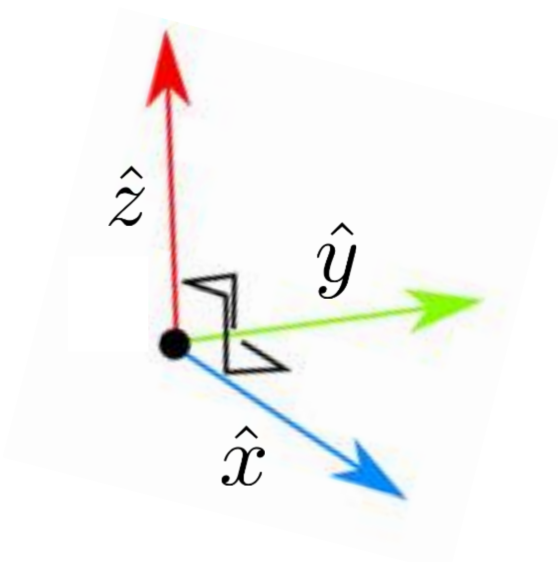
Pros:

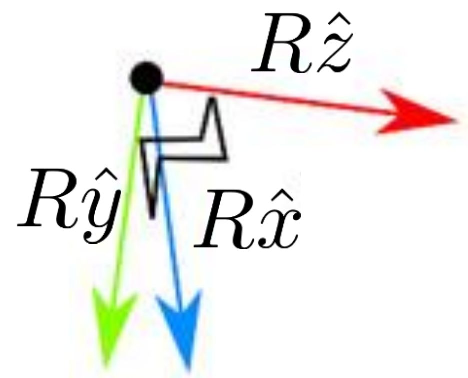
- intuitive

Cons:

- needs six numbers
- complicated orthonormality condition
- composition: unclear
- interpolation: no chance

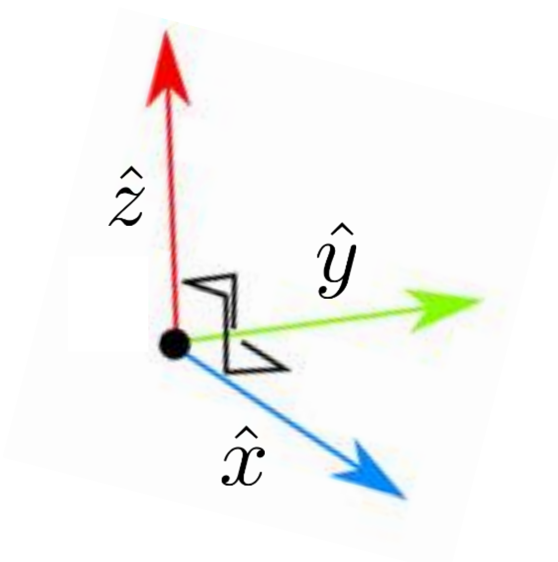
Representation 2: Matrices

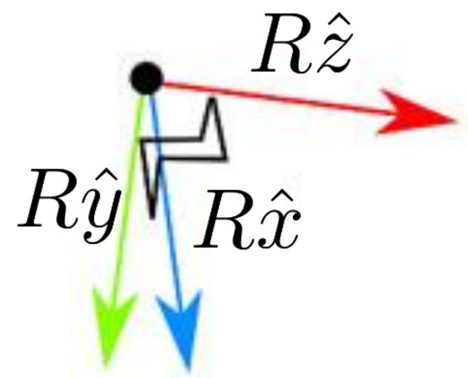




$$R = \left[\begin{array}{c|c|c} R\hat{x} & R\hat{y} & R\hat{z} \end{array} \right]$$

Representation 2: Matrices



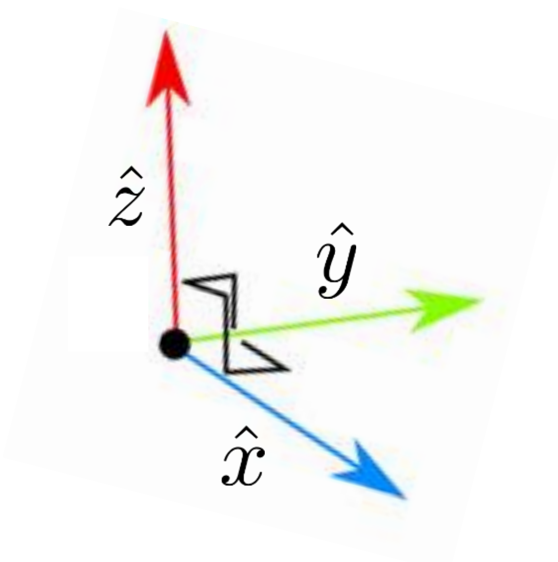


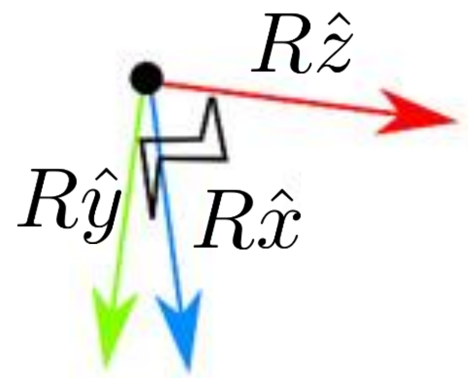
$$R = \left[\begin{array}{c|c|c} R\hat{x} & R\hat{y} & R\hat{z} \end{array}\right]$$

columns unit vectors

columns orthogonal

Representation 2: Matrices





$$R = \left[\begin{array}{c|c|c} R\hat{x} & R\hat{y} & R\hat{z} \end{array} \right]$$

columns unit vectors

columns orthogonal

$$R^T R = I$$

Representation 2: Matrices

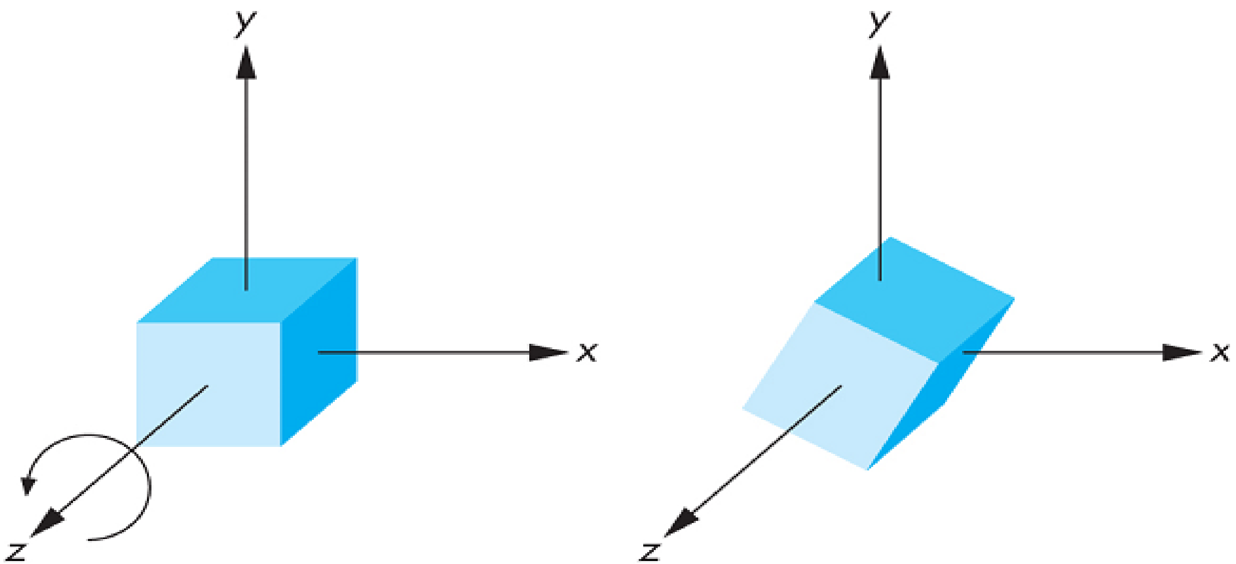
Pros

- composition easy
- applying rotation easy

Cons

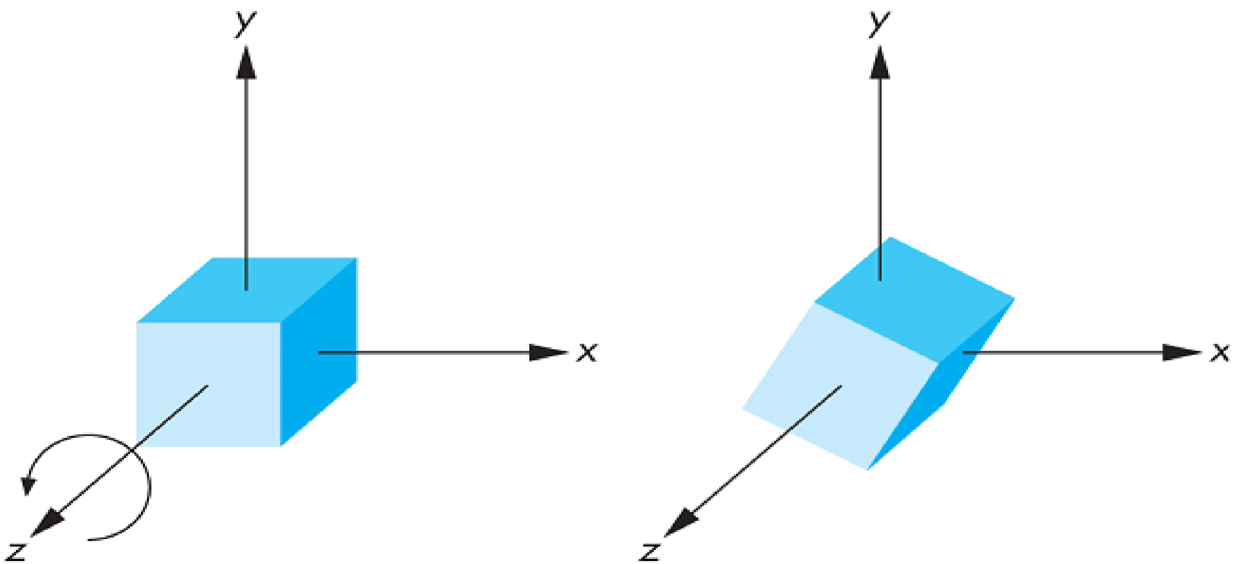
- needs **nine** numbers
- complicated condition $R^T R = I$
- interpolation: no chance

Special Case: Rotation about Axis



$$R_\theta^z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

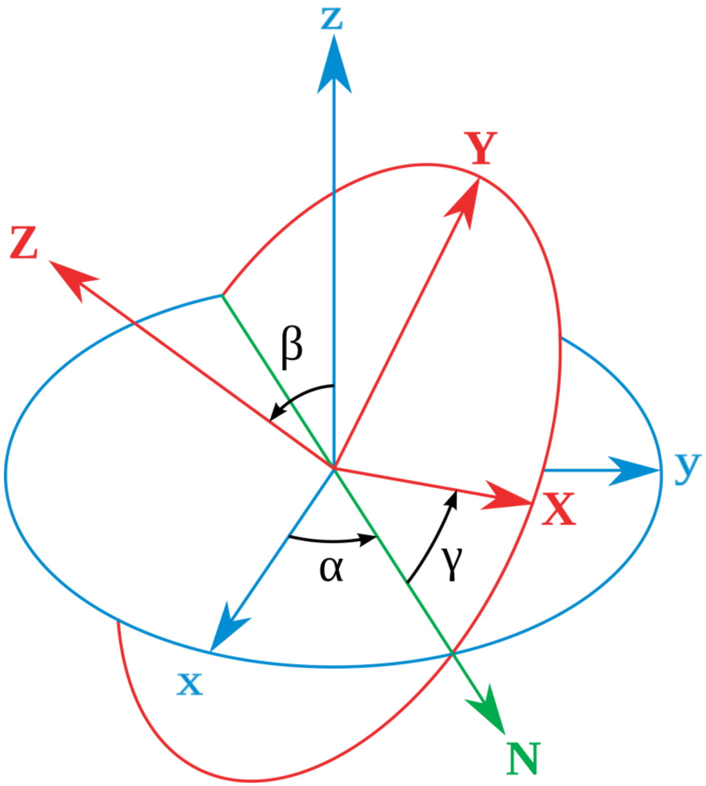
Special Case: Rotation about Axis



$$R_\theta^z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

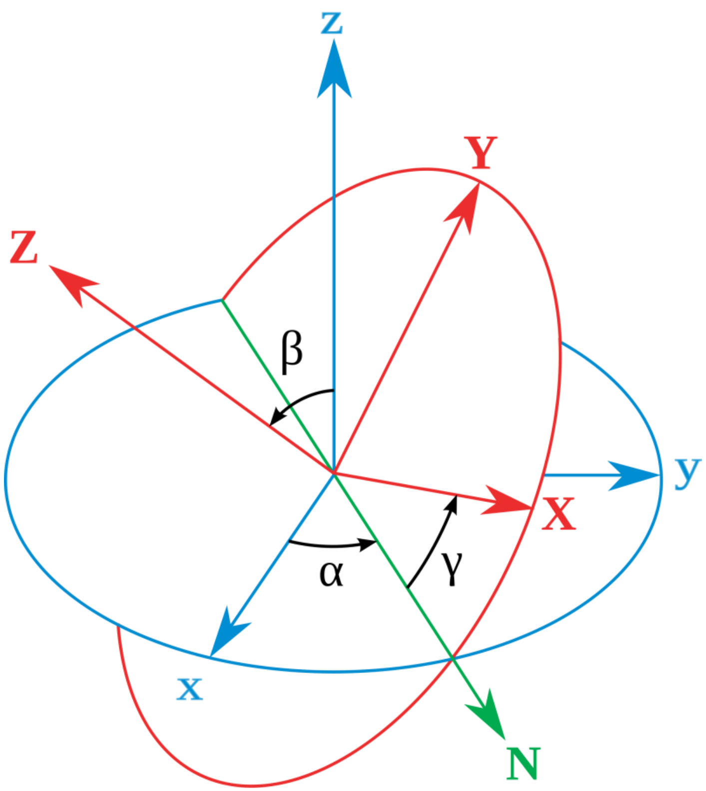
Idea: $R = R_\alpha^x R_\beta^y R_\gamma^z$

Representation 3: Euler Angles

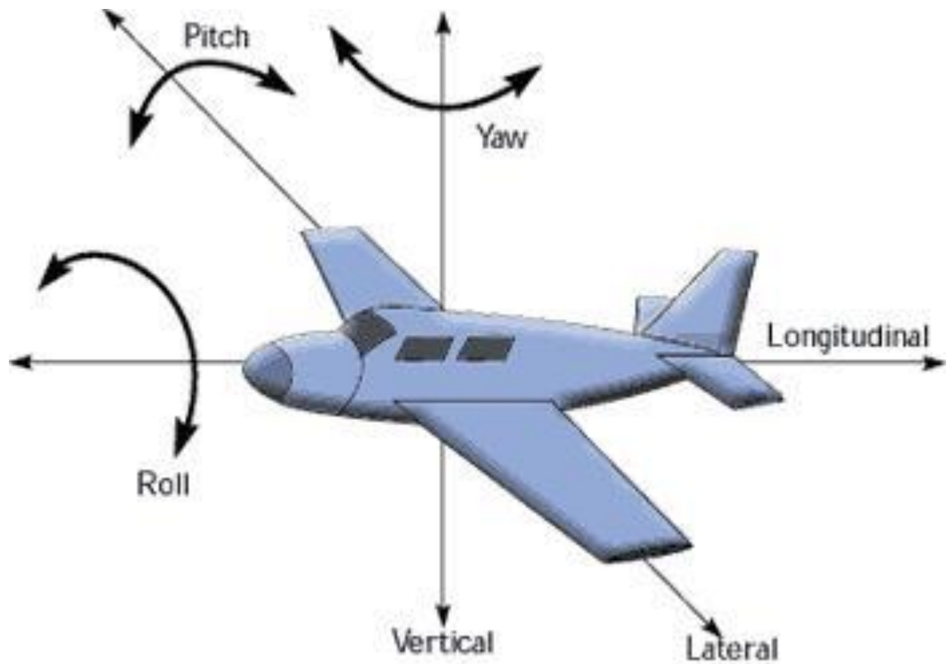


$$R = R^z_\alpha R^x_\beta R^z_\gamma$$

Representation 3: Euler Angles



$$R = R^z_\alpha R^x_\beta R^z_\gamma$$



common in aviation

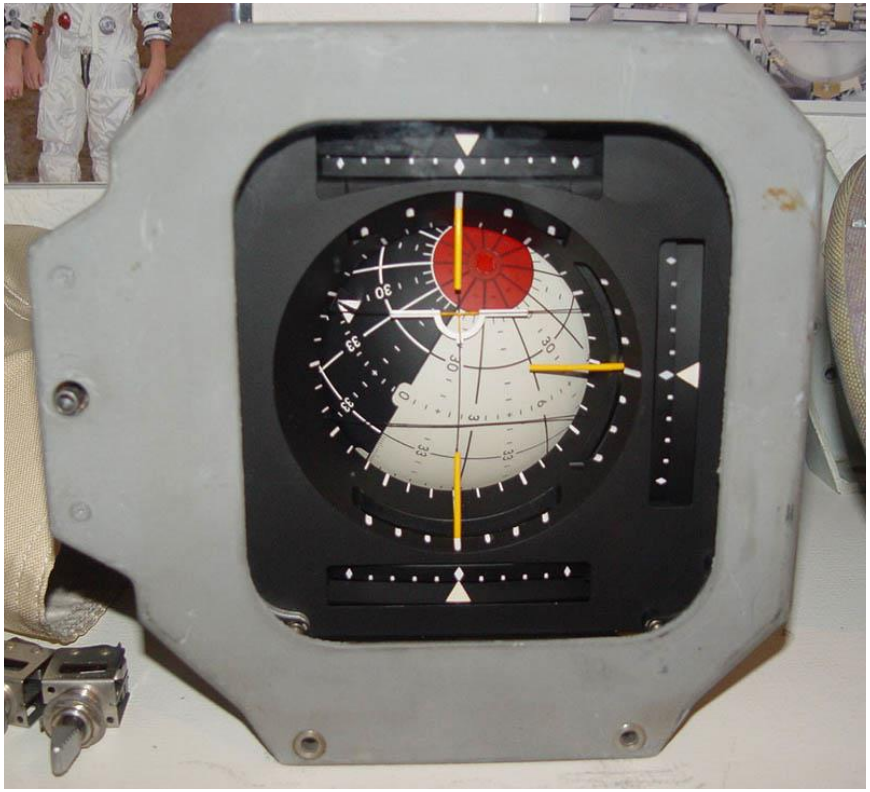
order matters!

Gimbal Lock

When first rotation aligns other axes

Rotations get “stuck”

Apollo 11 Gimbal Lock



Representation 3: Euler Angles

Pros:

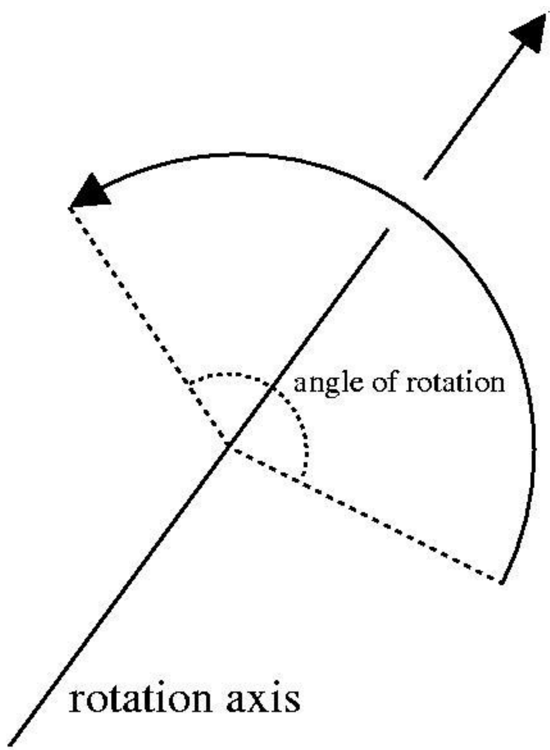
- just three numbers
- easy to convert to matrix

Cons

- complicated to apply/compose
- interpolation: gimbal lock

Representation 4: Axis-angle

Geometry fact: every rotation fixes an axis

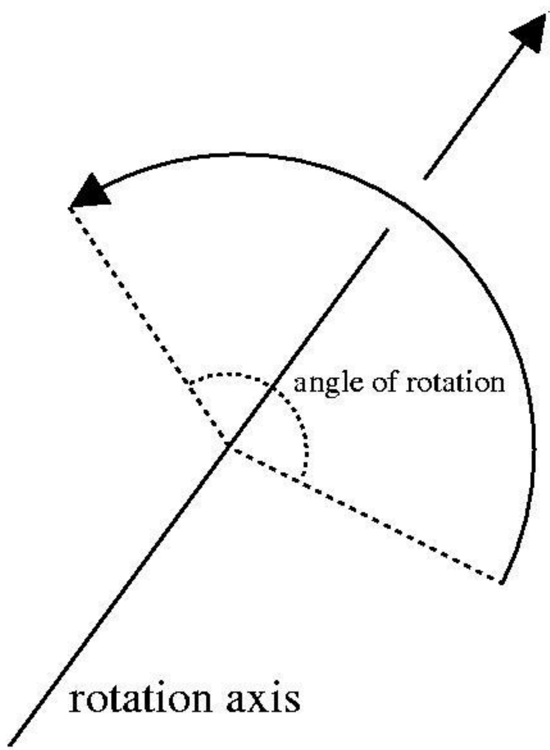


represent rotation as:

- axis $\hat{a}$
- angle θ

Representation 4: Axis-angle

Geometry fact: every rotation fixes an axis



represent rotation as:

- axis $\hat{a}$
- angle θ

…or “axis-angle” $\vec{\theta} = \theta\hat{a}$

Representation 4: Axis-angle

Converting to matrix **not** simple

Rodrigues Rotation Formula:

$$R = I + \sin\theta \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} + (1 - \cos\theta) \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}^2$$

Representation 4: Axis-angle

Pros:

- very intuitive
- only three numbers

Cons:

- complicated + **slow** to compose/apply
- interpolation: complicated

Representation 4: Quaternions

What is a quaternion?

- like complex numbers, but **three** imaginary dimensions

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$
$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$$

Representation 4: Quaternions

What is a quaternion?

- like complex numbers, but **three** imaginary dimensions

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$
$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$$

- cyclic relationship

$$\mathbf{ij} = \mathbf{k}$$
$$\mathbf{jk} = \mathbf{i}$$
$$\mathbf{ki} = \mathbf{j}$$

Representation 4: Quaternions

Turns out rotations can be represented by **unit** quaternions

$$a^2 + b^2 + c^2 + d^2 = 1$$

Rotation about axis (v_x, v_y, v_z):

$$q = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\left(v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}\right)$$

Representation 4: Quaternions

Turns out rotations can be represented by **unit** quaternions

$$a^2 + b^2 + c^2 + d^2 = 1$$

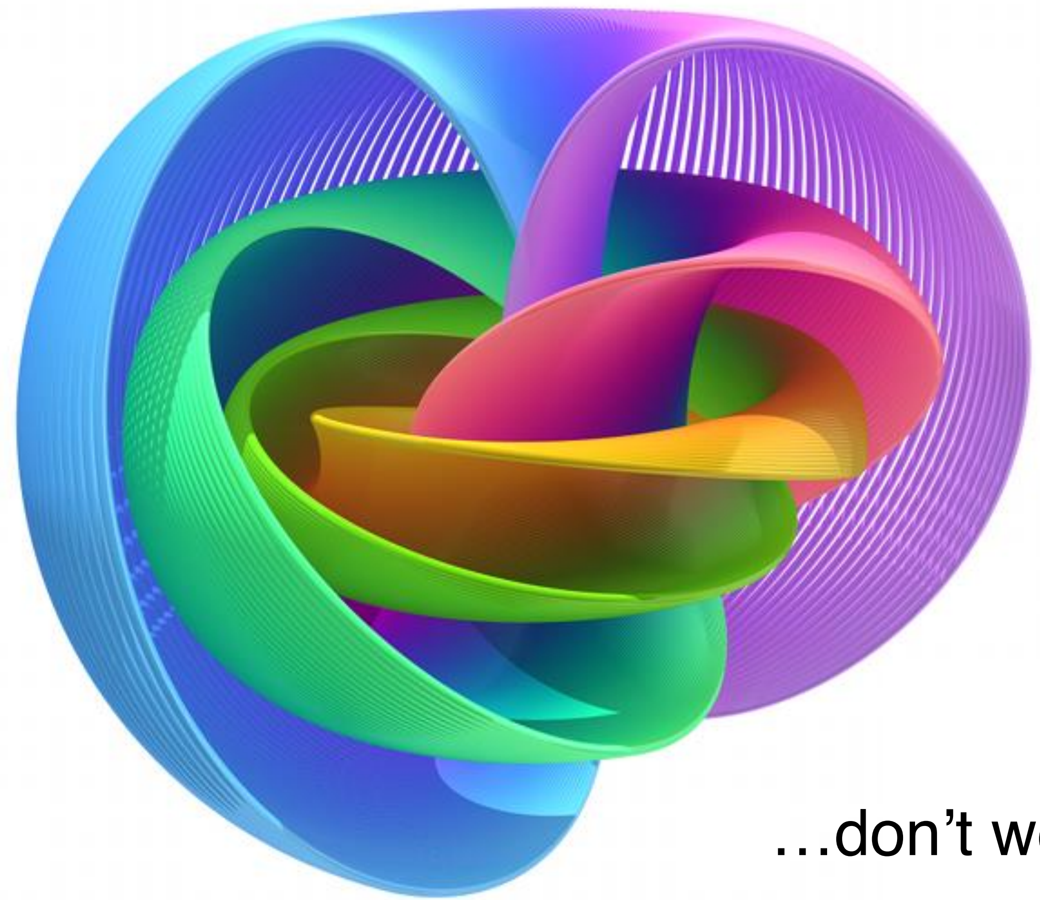
Rotation about axis (v_x, v_y, v_z)

$$q = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\left(v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}\right)$$

Acts on points/vectors by conjugation:

$$q\left(p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}\right)\bar{q}$$

Why Are Quaternions Rotations?

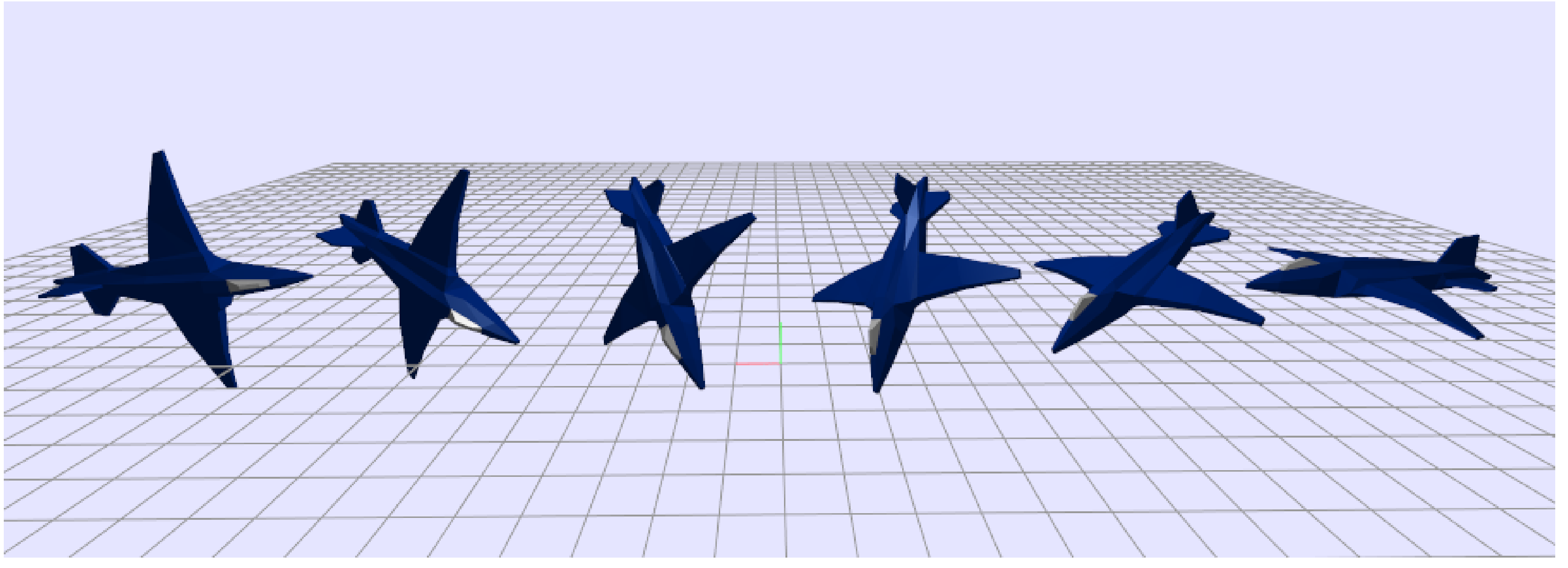


…don’t worry about it.

SLERP

Spherical **L**inear Int**erp**olation

- smoothly blends two rotations



SLERP

Spherical **L**inear Int**erp**olation

- smoothly blends two rotations

Why not interpolate rotations?

$$R(t) = (1 - t)R_1 + tR_2$$

SLERP

Spherical **L**inear Int**erp**olation

- smoothly blends two rotations

Turns out easy to do with quaternions

$$(q_2\bar{q}_1)^t q_1$$

- don't worry about the mechanics

Representation 4: Quaternions

Pros:

- just four numbers
- easy to compose
- easy to interpolate

Cons:

- unintuitive

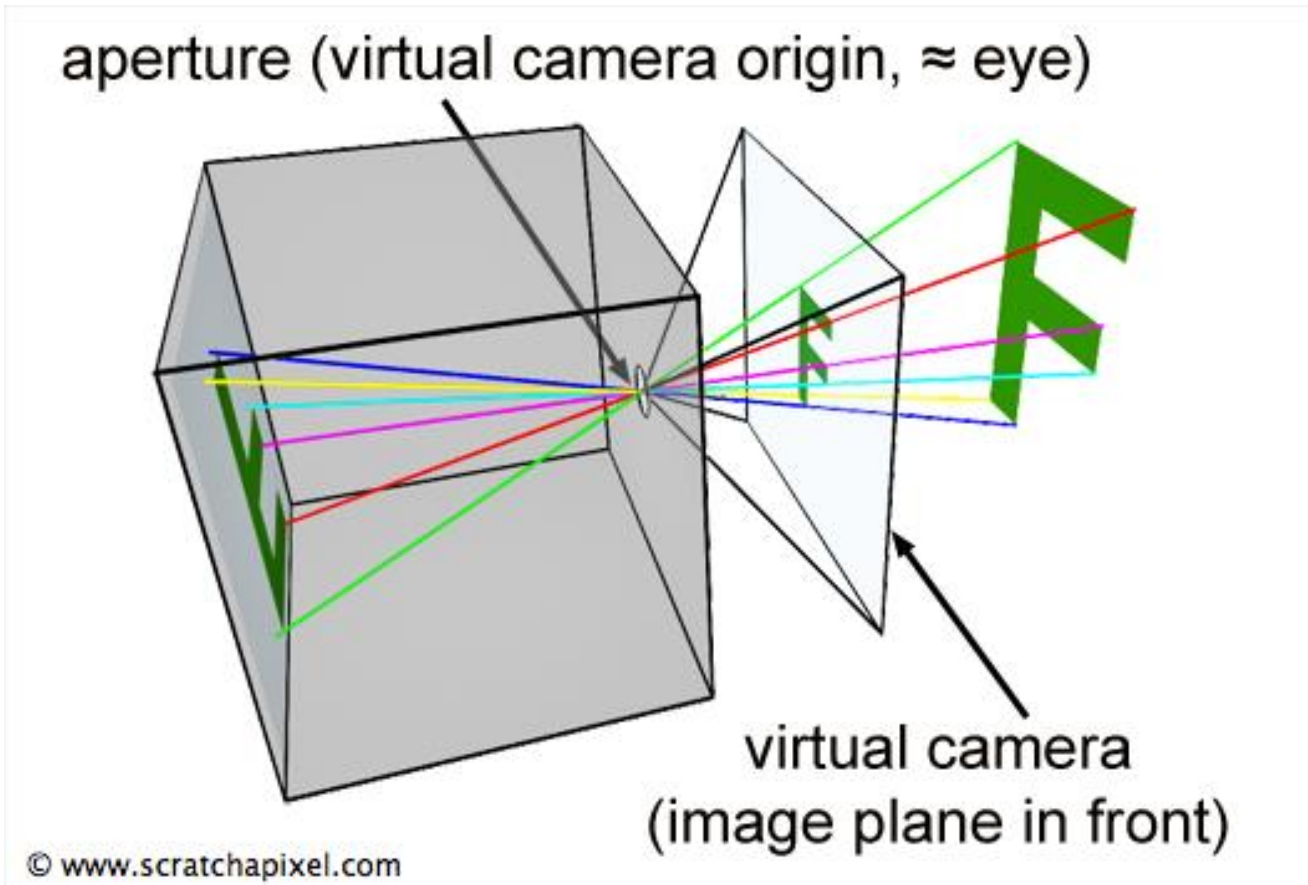
Rotations in Practice

Shaders work with **matrices** at end of day

High-performance intermediate computation is done with **quaternions**

Axis-angle most intuitive for camera controls and physics

Recall: Pinhole Camera



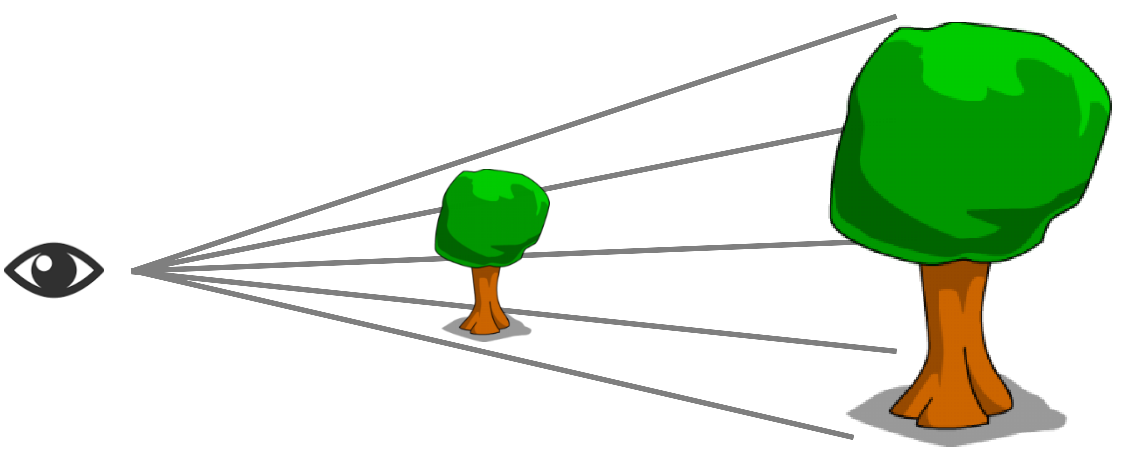
Pinhole Camera: Consequences

We see only **projection** of the world

Pinhole Camera: Consequences

We see only **projection** of the world

- many-to-one mapping
- distance vs scale ambiguity



Distance vs Scale Ambiguity

How we work around it:

- “deep learning”
- depth cues

Distance vs Scale Ambiguity

How we work around it:

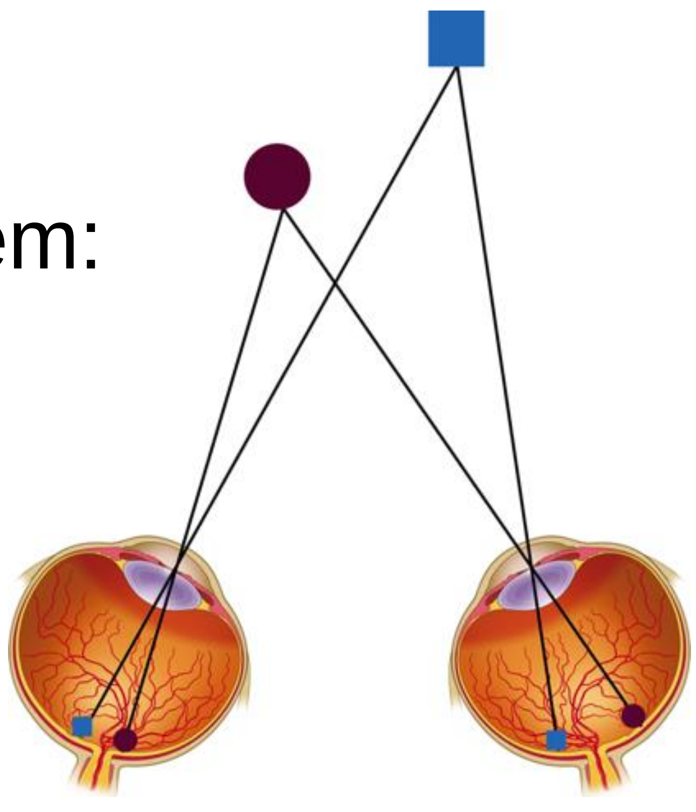
- “deep learning”
- depth cues

we assume:
similar objects have
similar shapes & sizes

Distance vs Scale Ambiguity

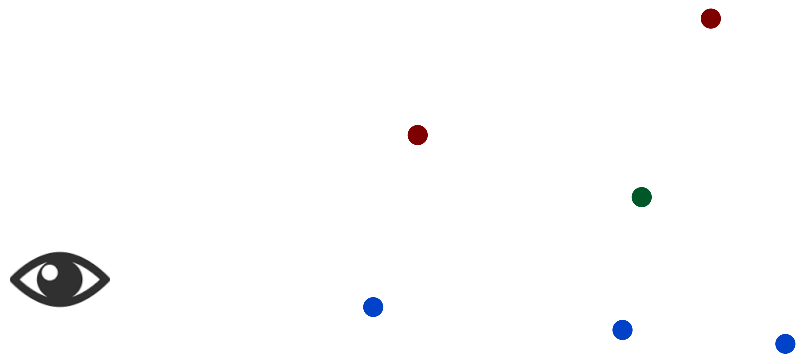
How we work around it:

- “deep learning”
- depth cues
- throw hardware at problem: binocular vision



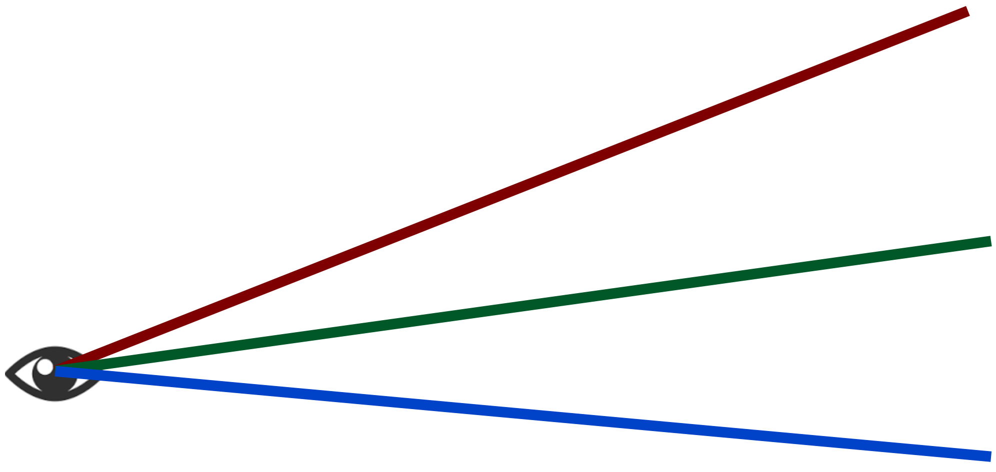
Monocular Vision Many-to-One

Points in space wrong abstraction



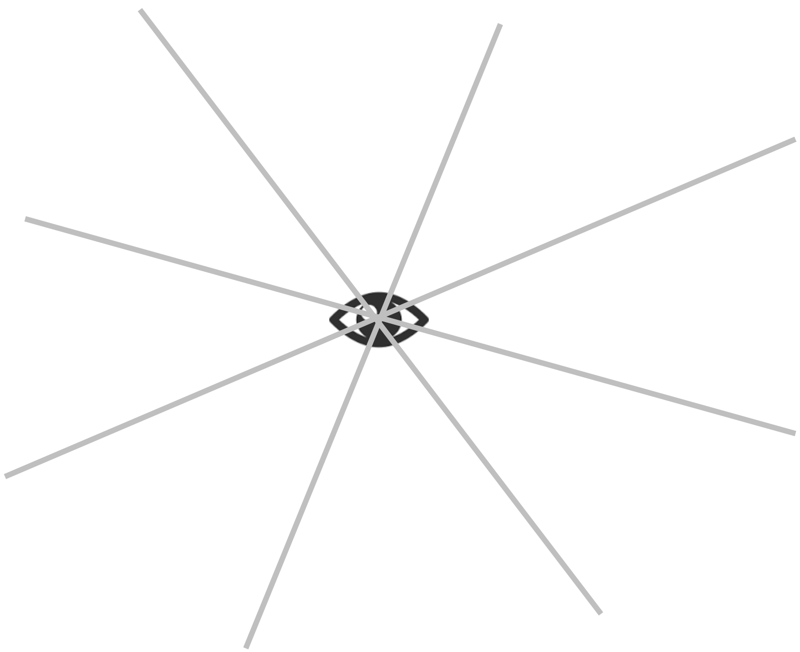
1D Projective Space

Space of **lines through origin**



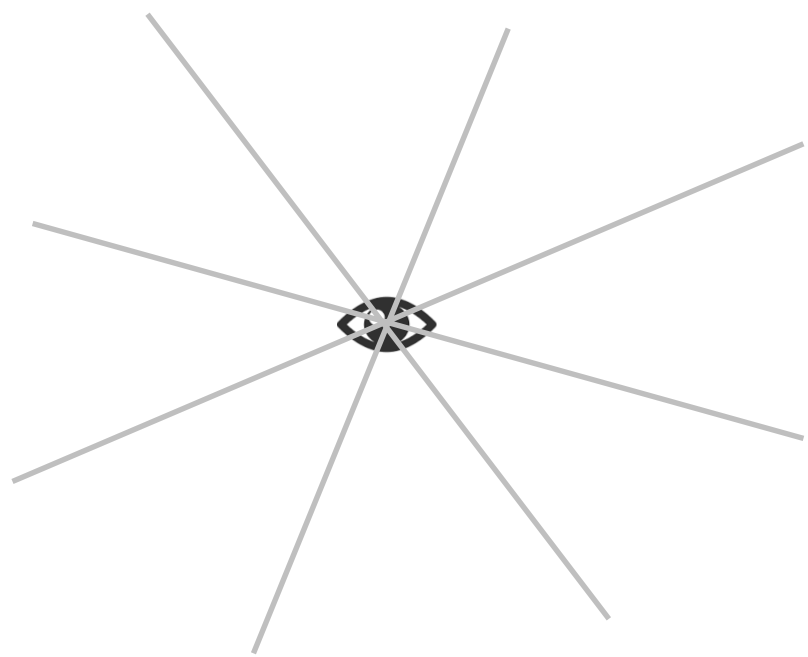
1D Projective Space Reps.

1. All lines through the origin



1D Projective Space Reps.

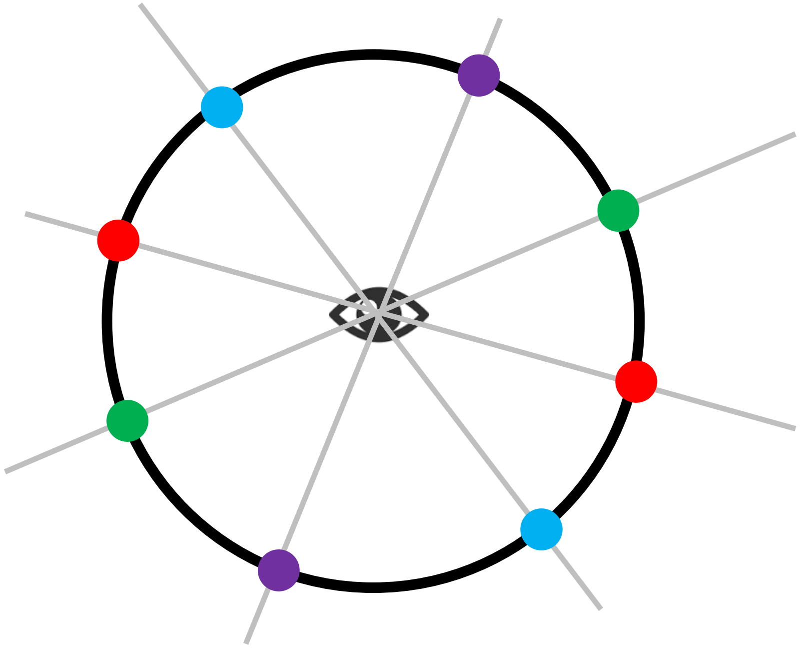
1. All lines through the origin



Note:
no notion of sign

1D Projective Space Reps.

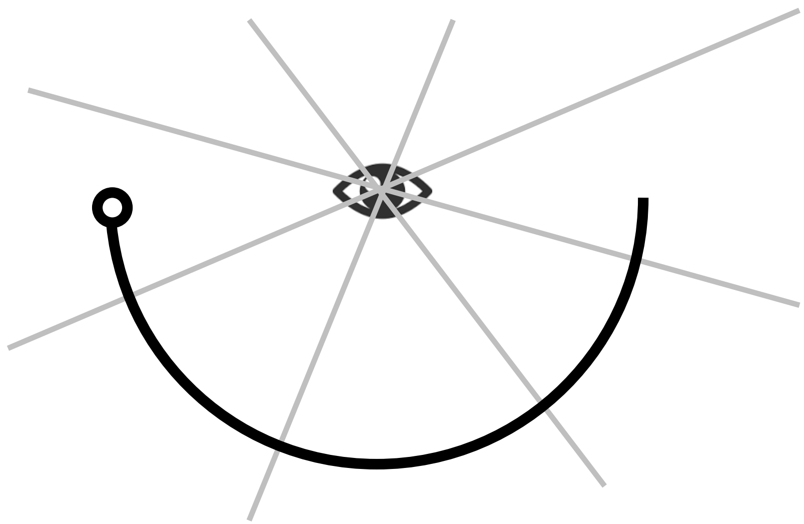
1. All lines through the origin
2. Circle with **antipodal points glued**



Note:
no notion of sign

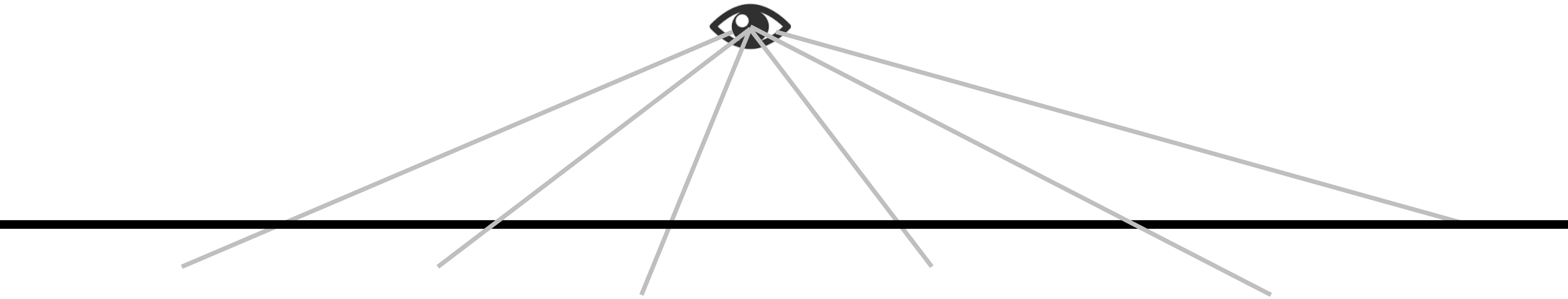
1D Projective Space Reps.

1. All lines through the origin
2. Circle with **antipodal points glued**
3. Interval $[0, \pi]$ with **boundary glued**



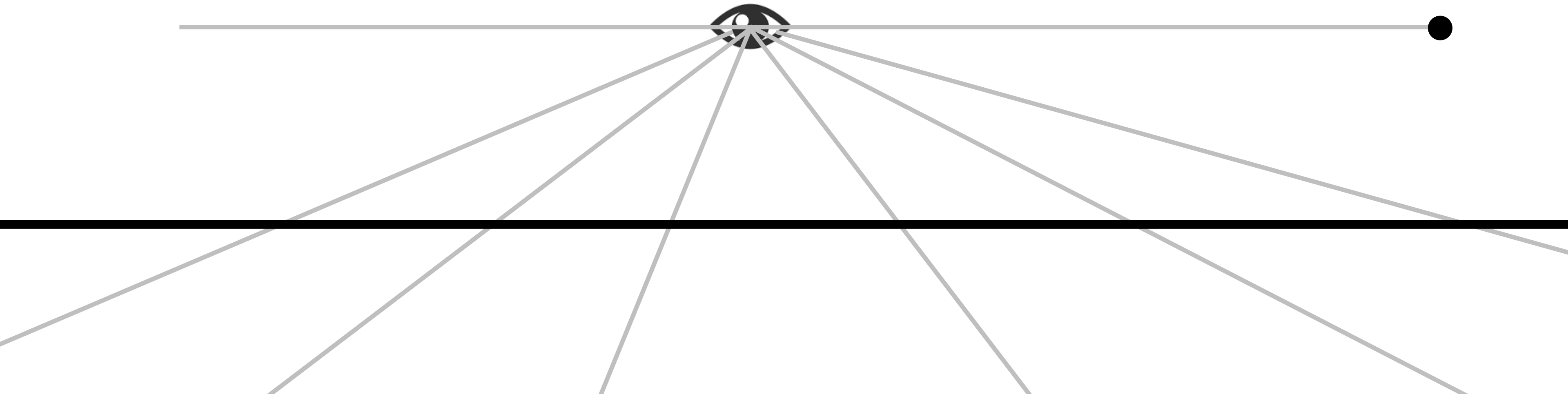
1D Projective Space Reps.

1. All lines through the origin
2. Circle with **antipodal points glued**
3. Interval $[0, \pi]$ with **boundary glued**
4. The real line…



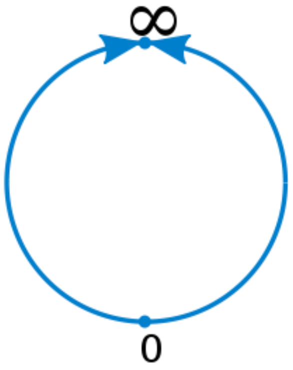
1D Projective Space Reps.

1. All lines through the origin
2. Circle with **antipodal points glued**
3. Interval $[0, \pi]$ with **boundary glued**
4. The real line… plus “point at infinity”



1D Projective Space Reps.

1. All lines through the origin
2. Circle with **antipodal points glued**
3. Interval $[0, \pi]$ with **boundary glued**
4. The real line… plus “point at infinity”
    - “ends” of line meet at infinity

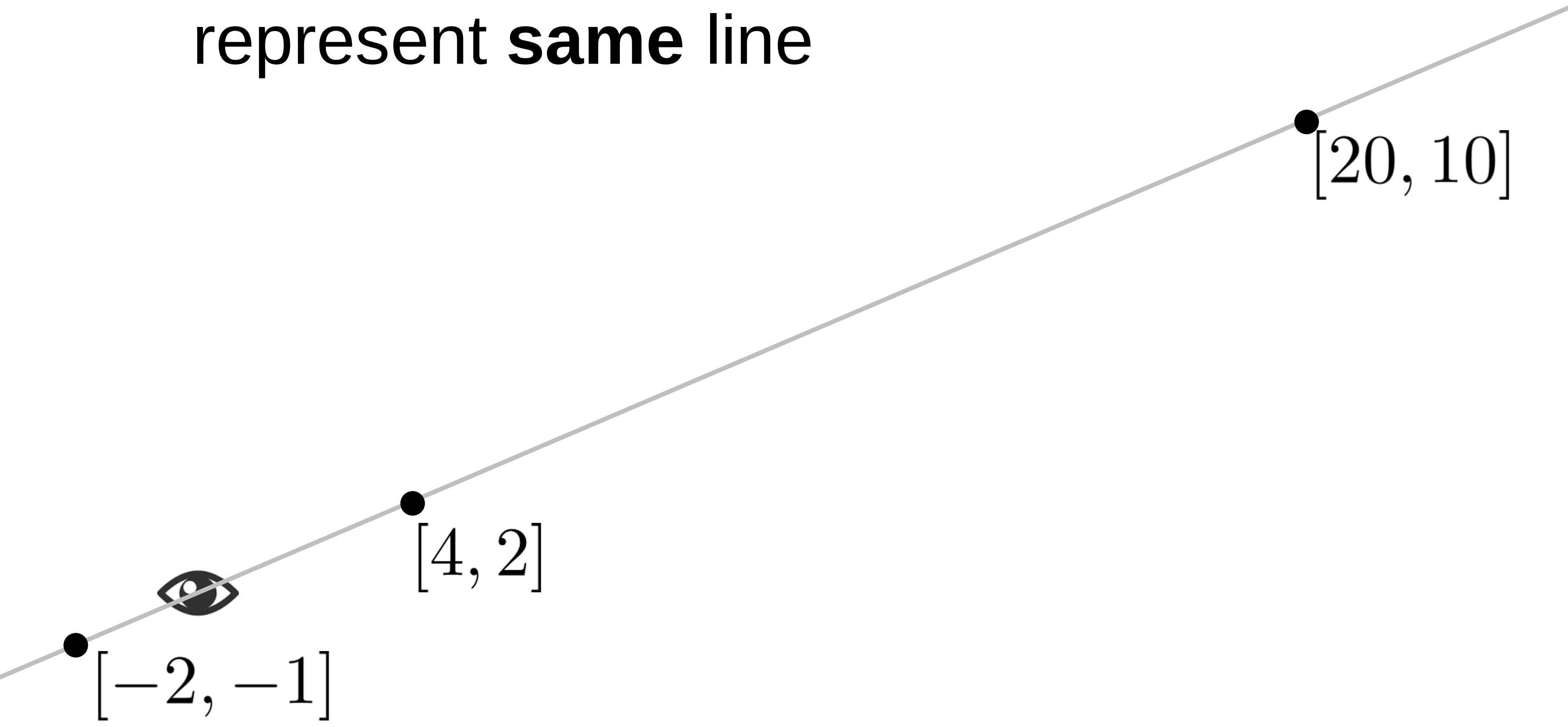


1D Projective Space Reps.

1. All lines through the origin
2. Circle with **antipodal points glued**
3. Interval $[0, \pi]$ with **boundary glued**
4. The real line… plus “point at infinity”
5. Homogeneous coordinates $[x, w]$

Homogeneous Coordinates

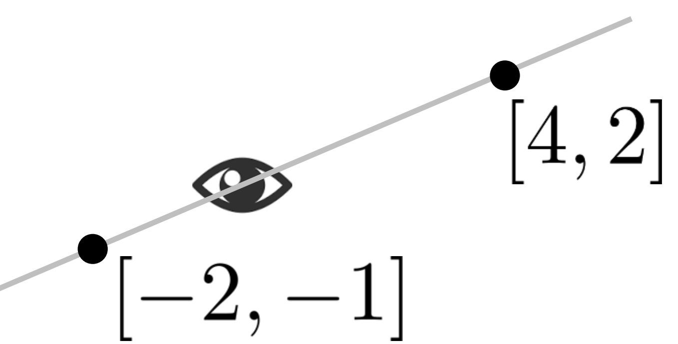
All homogeneous coordinates $[\alpha x, \alpha w]$ represent **same** line



Homogeneous Coordinates

All homogeneous coordinates $[\alpha x, \alpha w]$ represent **same** line

- say $[x, w] \sim [\alpha x, \alpha w]$ ("are **equivalent**")

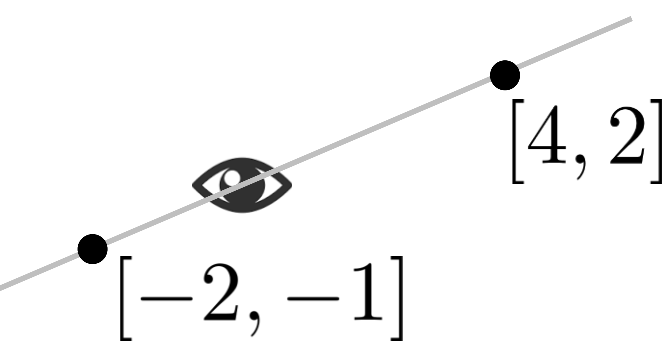


Homogeneous Coordinates

All homogeneous coordinates $[\alpha x, \alpha w]$ represent **same** line

- say $[x, w] \sim [\alpha x, \alpha w]$ ("are **equivalent**")

All points in 1D projective space equiv to:



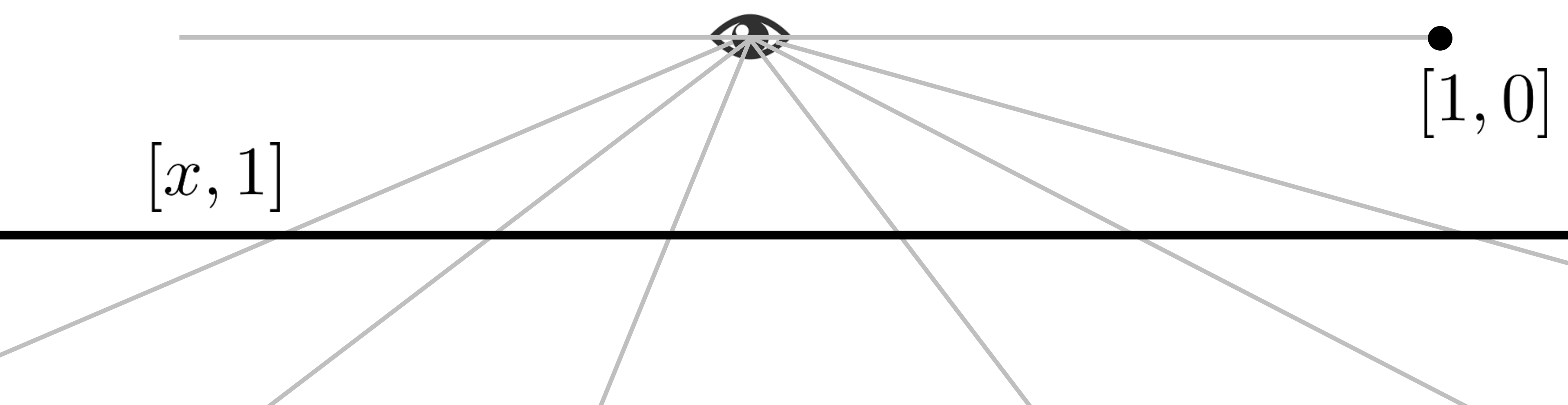
1. $[x, 1]$
2. $[1, 0]$

Homogeneous Coordinates

All homogeneous coordinates $[\alpha x, \alpha w]$ represent **same** line

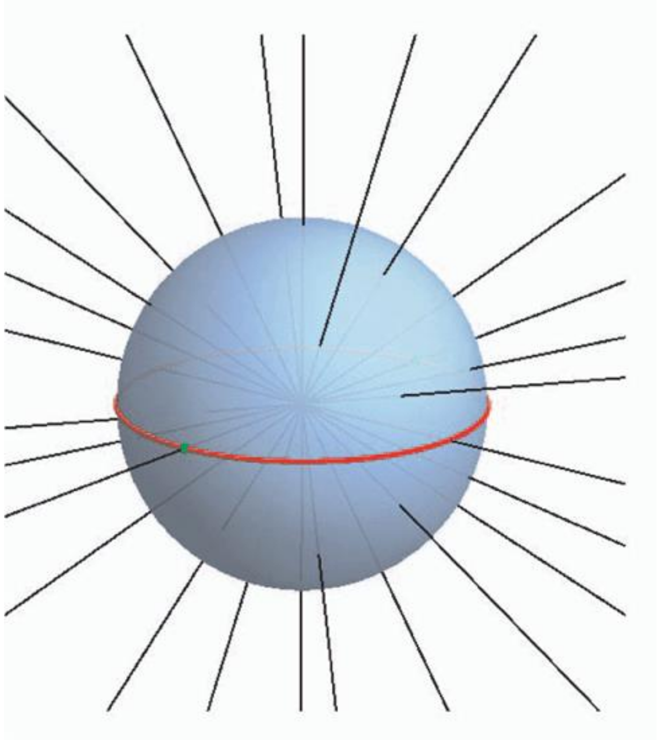
- say $[x, w] \sim [\alpha x, \alpha w]$ ("are **equivalent**")

All points in 1D projective space equiv to:



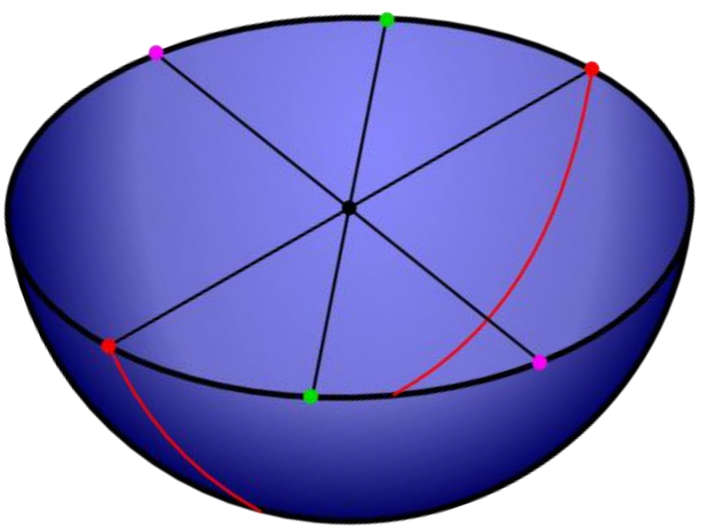
2D Projective Plane

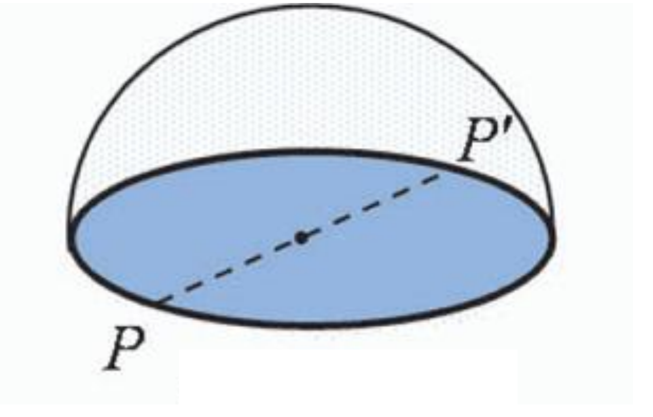
1. All lines through the origin
2. Sphere with **antipodal points glued**



2D Projective Plane

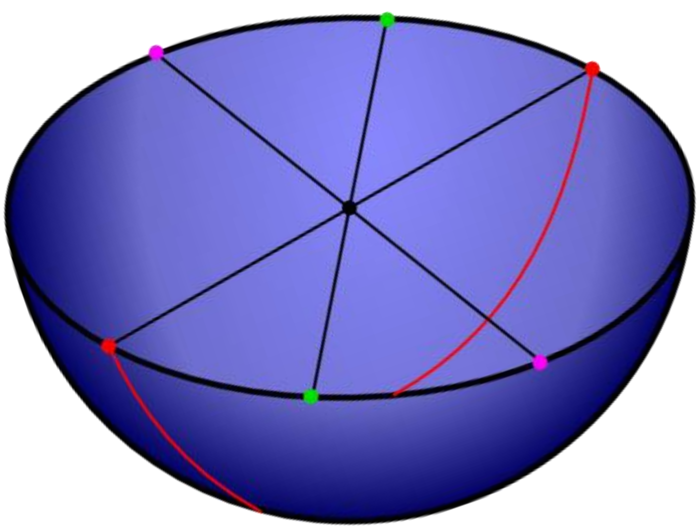
1. All lines through the origin
2. Sphere with **antipodal points glued**
3. Radius π disk with **boundary glued**

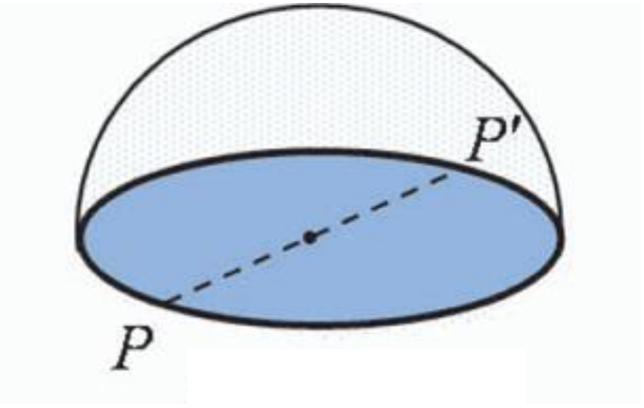




2D Projective Plane

1. All lines through the origin
2. Sphere with **antipodal points glued**
3. Radius π disk with **boundary glued**



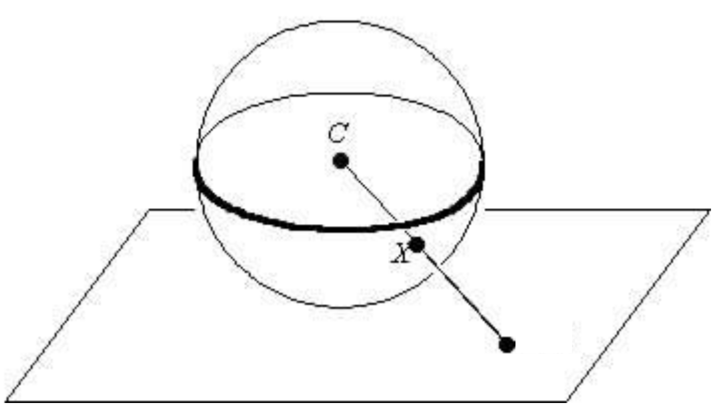


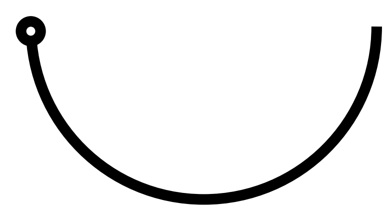
what does this look like?

Boy’s Surface

2D Projective Plane

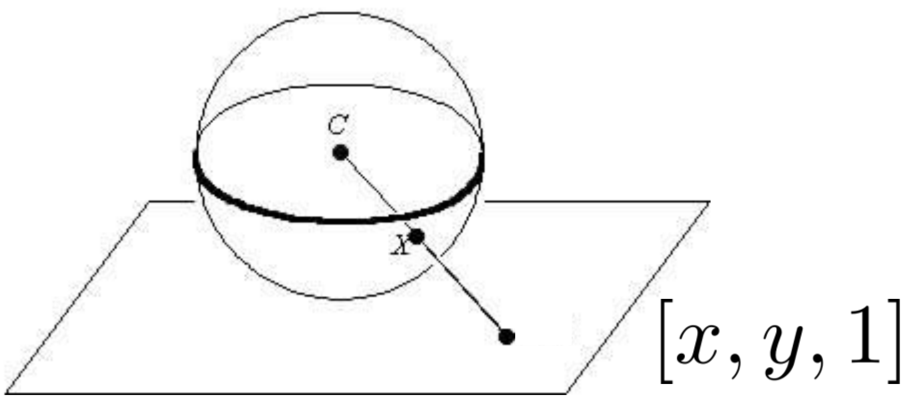
1. All lines through the origin
2. Sphere with **antipodal points glued**
3. Radius π disk with **boundary glued**
4. Plane plus “**line at infinity**”

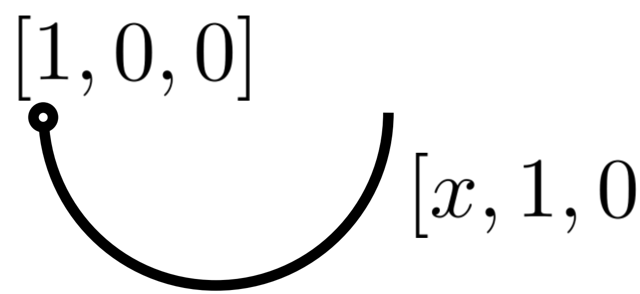




2D Projective Plane

1. All lines through the origin
2. Sphere with **antipodal points glued**
3. Radius π disk with **boundary glued**
4. Plane plus "**line at infinity**"
5. Homogeneous coordinates $[x, y, w]$

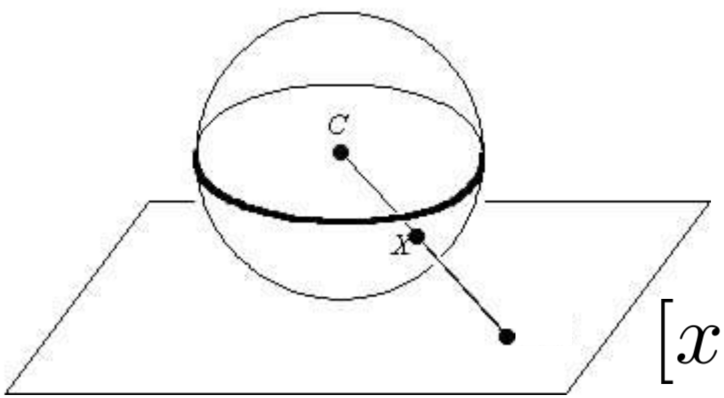




on projective plane, all lines intersect at one point

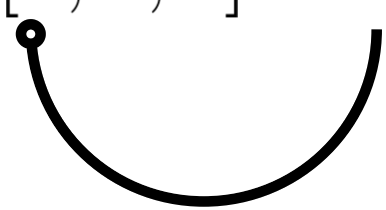
2D Projective Plane

1. All lines through the origin
2. Sphere with **antipodal points glued**
3. Radius π disk with **boundary glued**
4. Plane plus "**line at infinity**"
5. Homogeneous coordinates $[x, y, w]$



$[x, y, 1]$

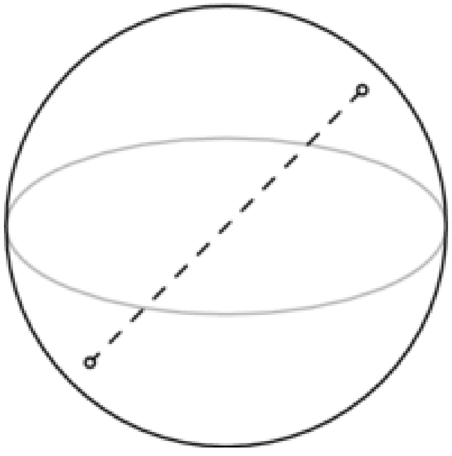
$[1, 0, 0]$



$[x, 1, 0]$

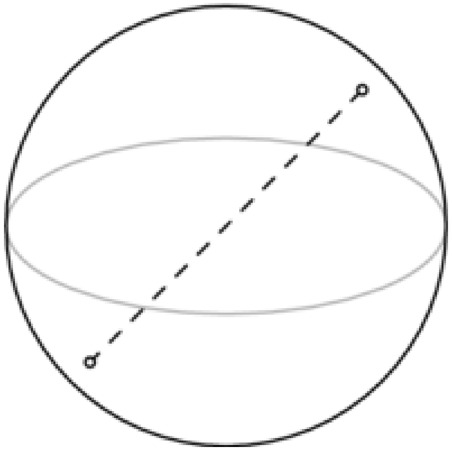
3D Projective Space

1. All lines through the origin
2. Hypersphere with **antipodal pts glued**
3. Radius π ball with **boundary glued**



3D Projective Space

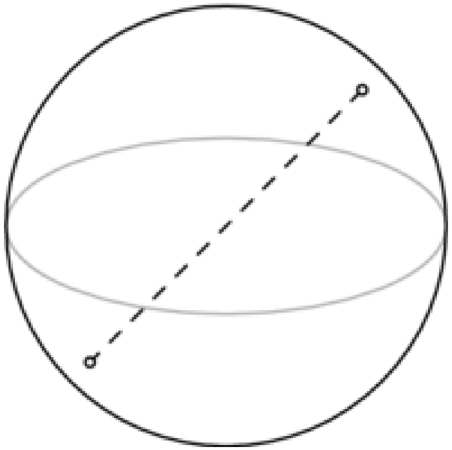
1. All lines through the origin
2. Hypersphere with **antipodal pts glued**
3. Radius π ball with **boundary glued**



remind you of anything?

3D Projective Space

1. All lines through the origin
2. Hypersphere with **antipodal pts glued**
3. Radius π ball with **boundary glued**
4. Rotations in 3D (axis-angle)!



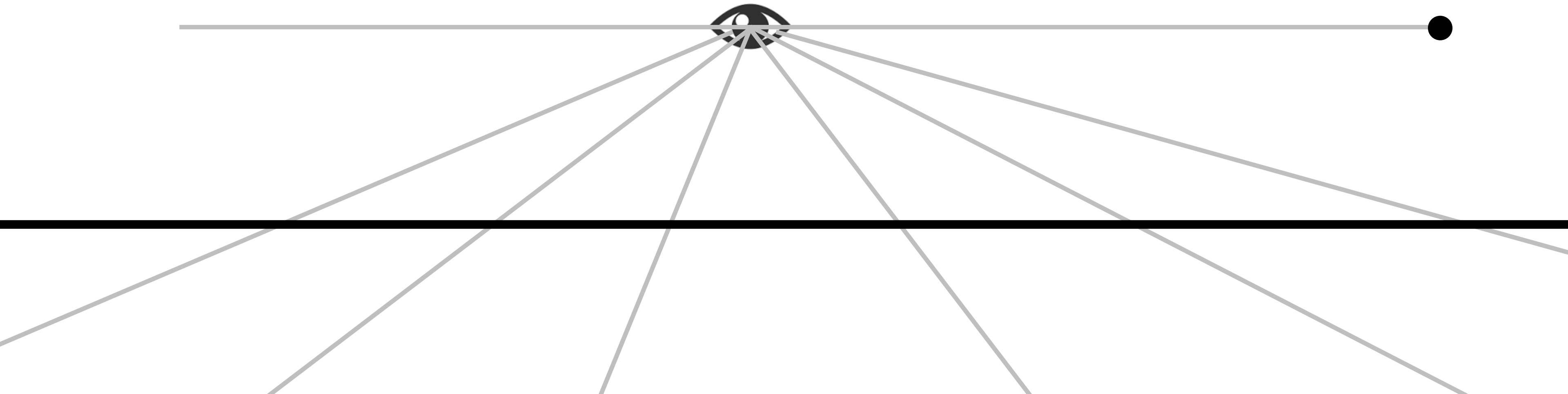
remind you of anything?

3D Projective Space

1. All lines through the origin
2. Hypersphere with **antipodal pts glued**
3. Radius π ball with **boundary glued**
4. Rotations in 3D (axis-angle)!
5. Homogeneous coordinates $[x, y, z, w]$

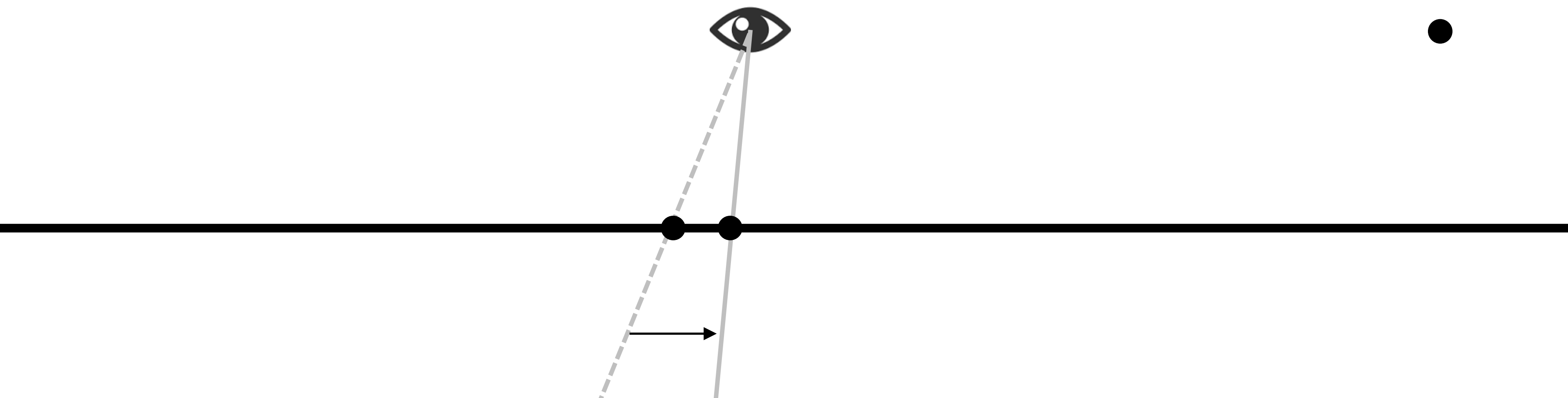
A Final Note

What does shear do in projective space?



A Final Note

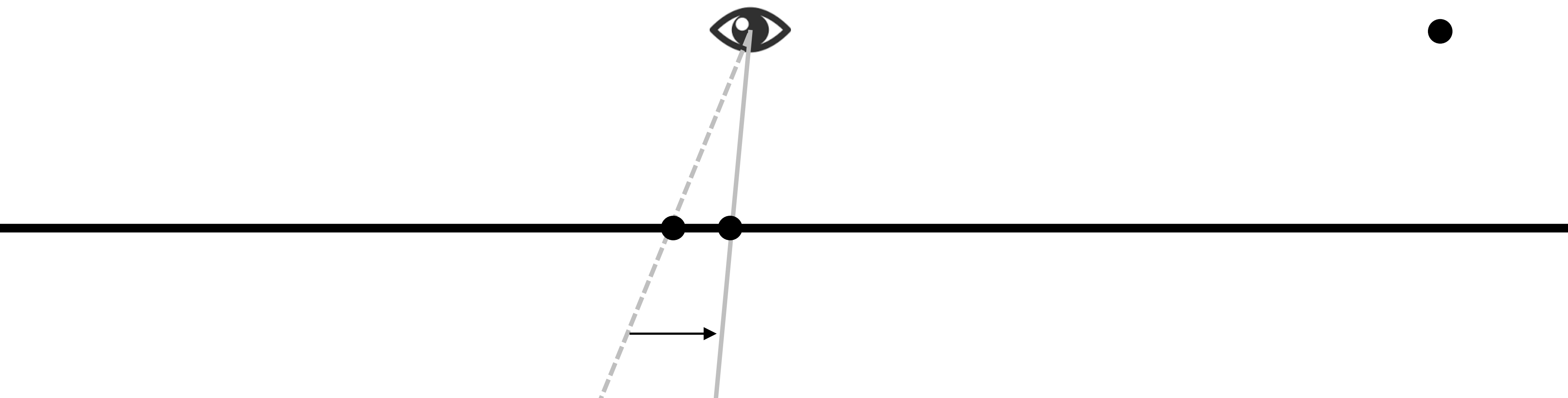
What does shear do in projective space?



A Final Note

What does shear do in projective space?

- points on line **translate**



A Final Note

What does shear do in projective space?

- points on line **translate**
- point at infinity untouched