

---

# Computer Graphics

## 9 - Orientation & Rotation

Yoonsang Lee  
Spring 2021

# Topics Covered

---

- Orientation vs. Rotation
- Degrees of Freedom
- 3D Orientation & Rotation Representations
  - Euler angles
  - Axis-angle (Rotation vector)
  - **Rotation matrices**
  - Unit quaternions
- 3D Orientation Interpolation

---

# **Orientation vs. Rotation, Degrees of Freedom**

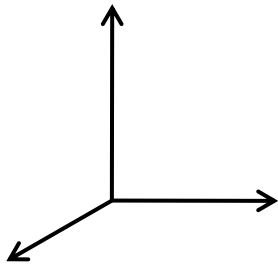
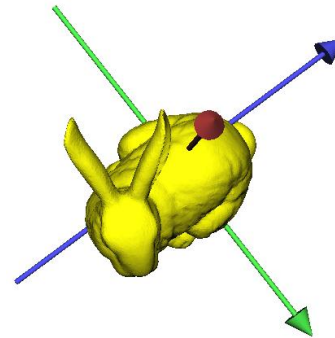
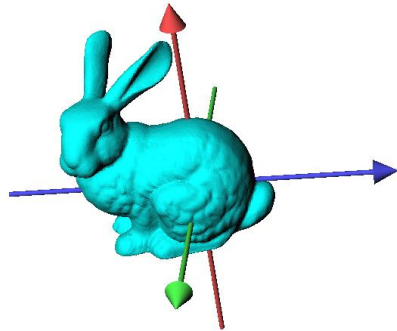
# Orientation vs. Rotation

---

- Rotation
  - Circular movement
  
- Orientation
  - The state of being oriented
  - Given a coordinate system, the orientation of an object can be represented **as a rotation from a reference orientation**

# Analogy

- (point : vector) is similar to (orientation : rotation)
  - Both represent a sort of (state : movement)

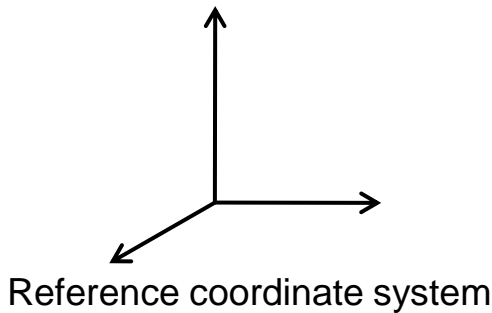
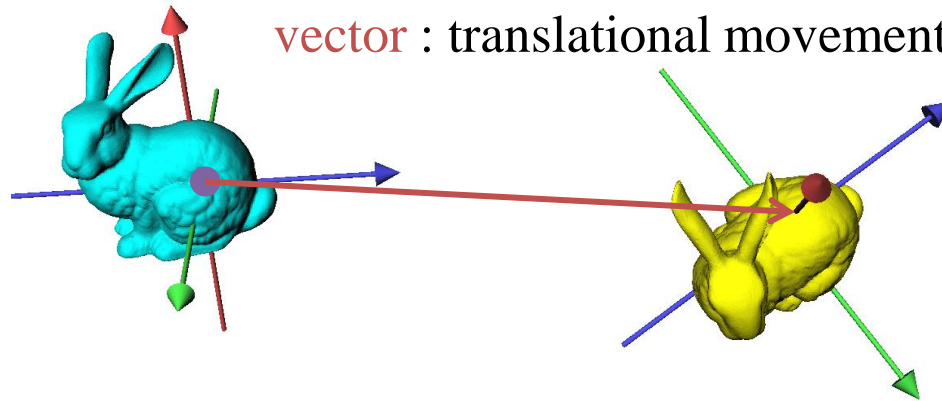


Reference coordinate system

# Analogy

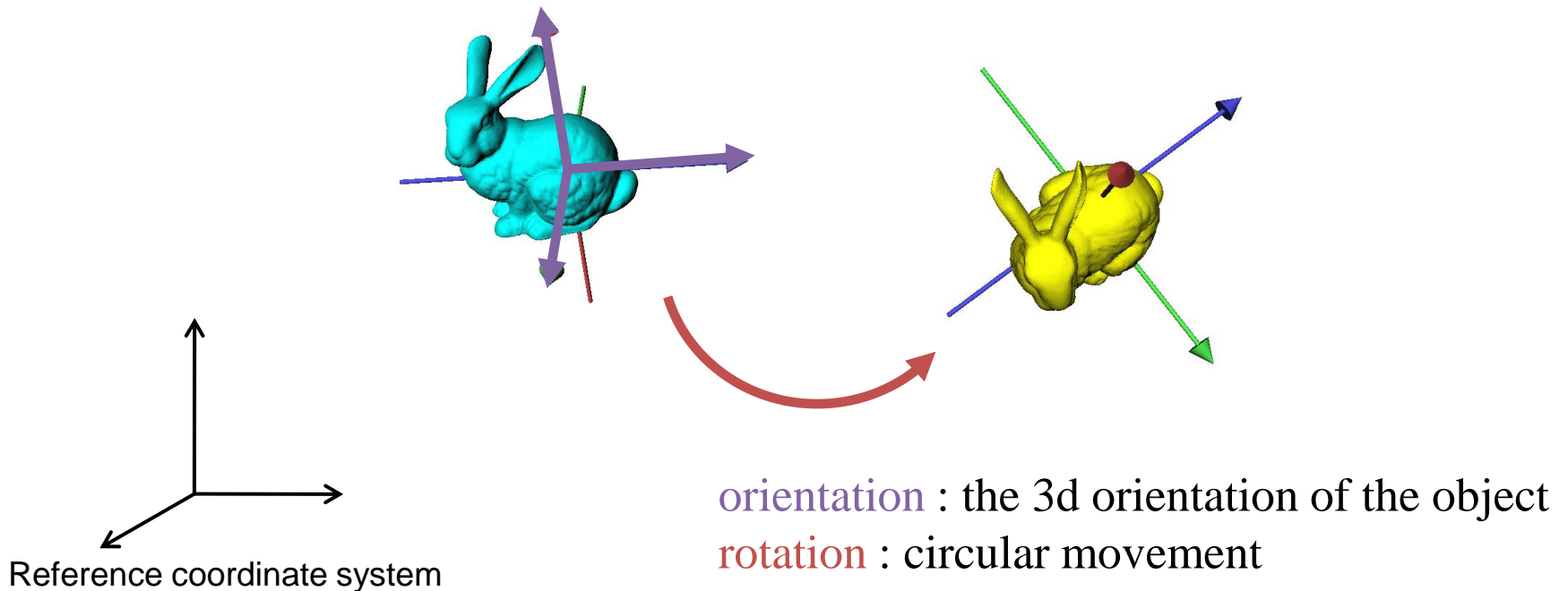
- (point : vector) is similar to (orientation : rotation)
  - Both represent a sort of (state : movement)

point : the 3d location of the object  
vector : translational movement



# Analogy

- (point : vector) is similar to (orientation : rotation)
  - Both represent a sort of (state : movement)



# Analogy

---

- Point & vector

- (point) + (point)  $\rightarrow$  (UNDEFINED)
- (vector)  $\pm$  (vector)  $\rightarrow$  (vector)
- (point)  $\pm$  (vector)  $\rightarrow$  (point)
- (point) - (point)  $\rightarrow$  (vector)

- Orientation & rotation

- (orientation) (+) (orientation)  $\rightarrow$  (UNDEFINED)
- (rotation) ( $\pm$ ) (rotation)  $\rightarrow$  (rotation)
- (orientation) ( $\pm$ ) (rotation)  $\rightarrow$  (orientation)
- (orientation) (-) (orientation)  $\rightarrow$  (rotation)

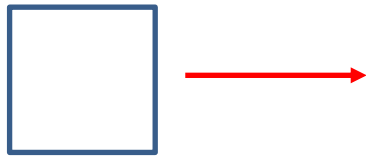
Not vector addition & subtraction



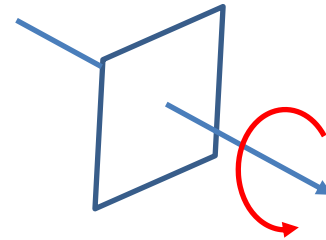
# Degrees of Freedom (DOF)

---

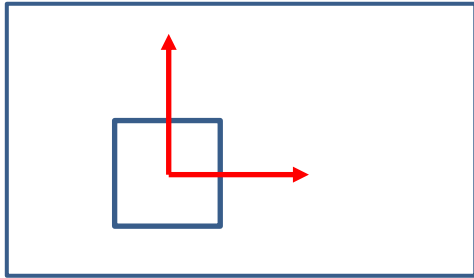
- The number of **independent parameters** that define a **unique configuration**



Translation along one  
direction  
: 1 DOF

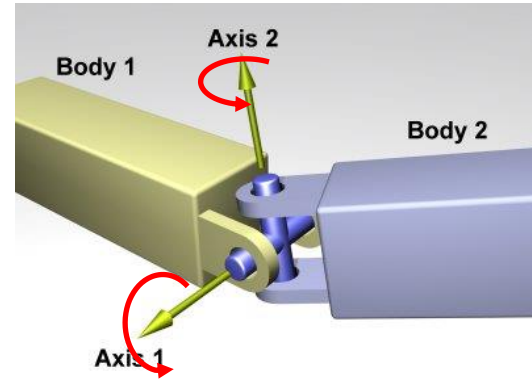


Rotation about an axis  
: 1 DOF



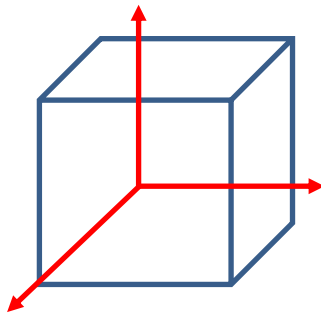
Translation on a plane

: 2 DOFs



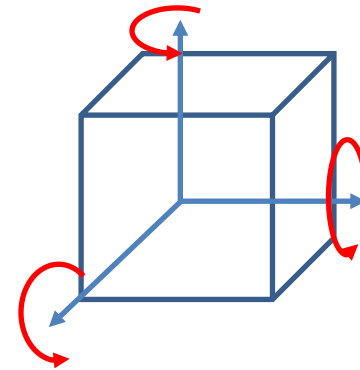
Rotation about two axes

: 2 DOFs



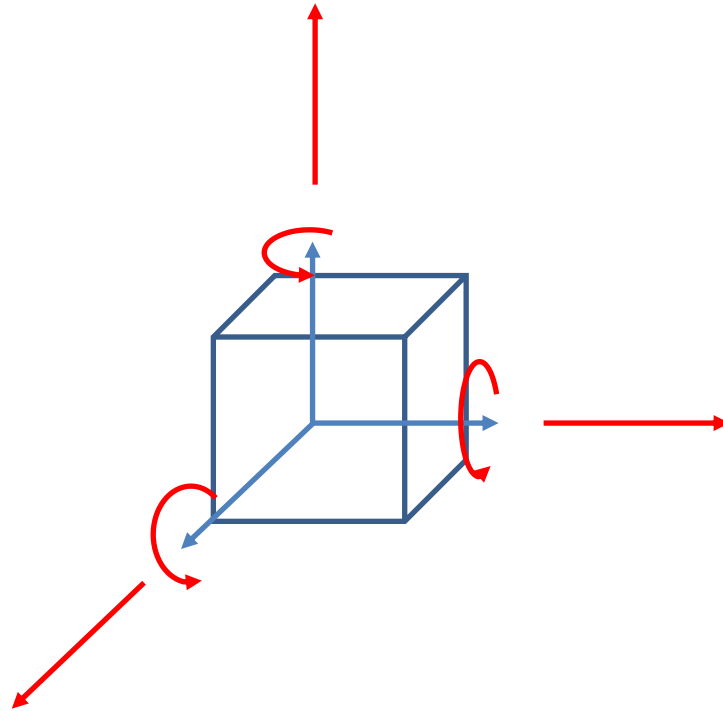
Translation in 3D space

: 3 DOFs



Rotation in 3D space

: 3 DOFs



Any rigid motion in 3D  
space

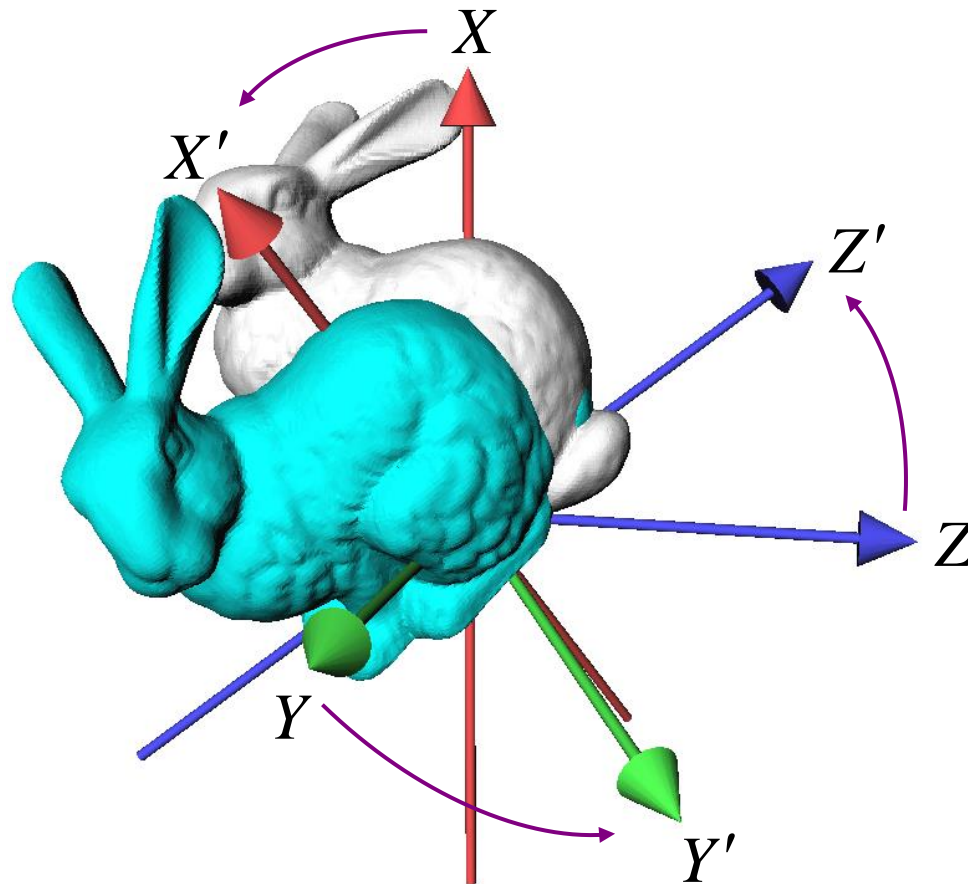
: 6 DOF

---

# **3D Orientation & Rotation Representations**

# 3D Rotation

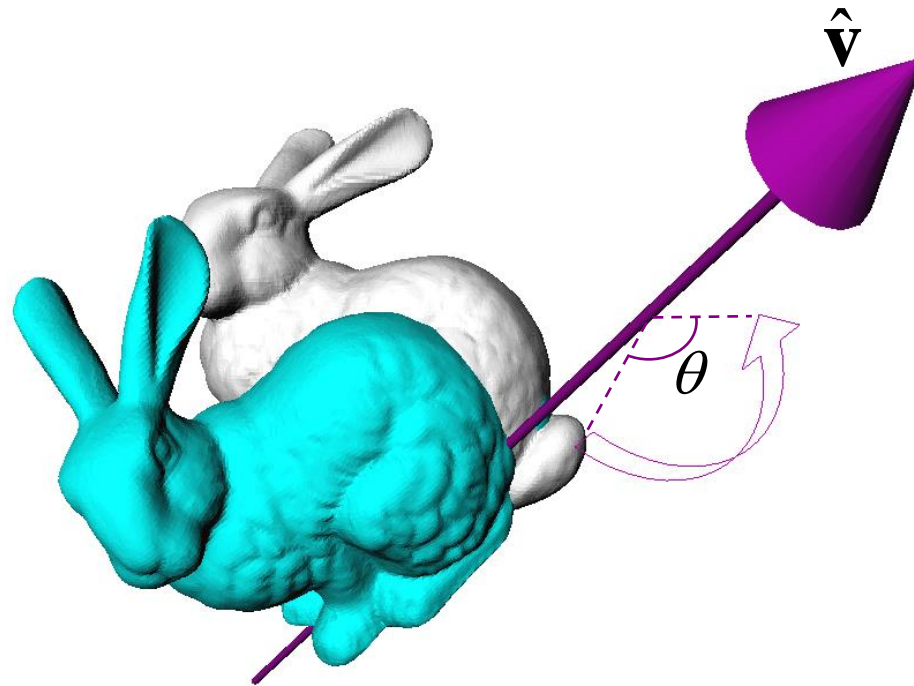
- Given two arbitrary orientations of a rigid object,



# 3D Rotation

---

- We can always find a fixed axis of rotation and an angle about the axis



# Euler's Rotation Theorem

---

**The general displacement of a rigid body with one point fixed is a rotation about some axis**

Leonhard Euler (1707-1783)

In other words,

- Arbitrary 3D rotation equals to one rotation around an axis
- Any 3D rotation leaves one vector unchanged

# Describing 3D Rotation & Orientation

---

- Several ways to describe 3D rotation and orientation
  - Euler angles
  - Rotation vector (Axis-angle)
  - Rotation matrices
  - Unit quaternions

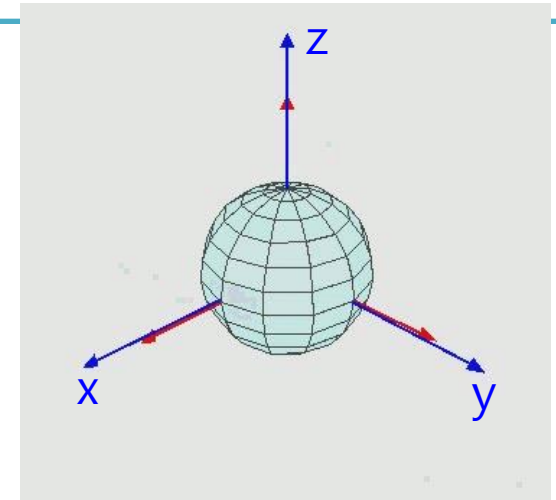
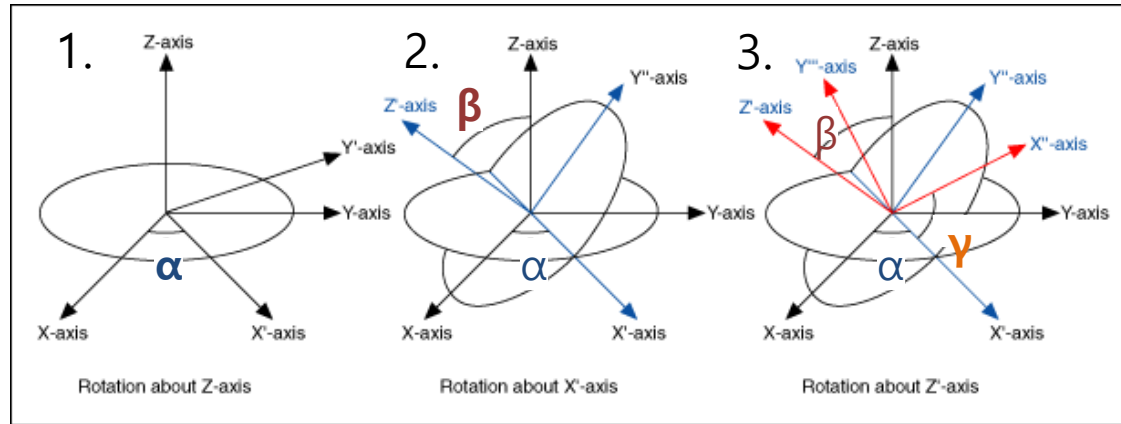


# Euler Angles

---

- Express any arbitrary 3D rotation using **three rotation angles about three principle axes**
  - x, y, z axes

# Example: ZXZ Euler Angles



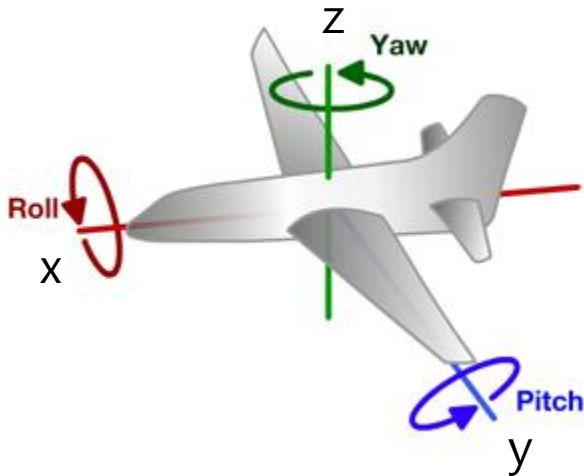
- 1. Rotate about Z-axis by  $\alpha$
- 2. Rotate about X-axis of the new frame by  $\beta$
- 3. Rotate about Z-axis of the new frame by  $\gamma$

<https://commons.wikimedia.org/wiki/File:Euler2a.gif>

$$R = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \beta & -\sin \beta \\ 0 & \sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_Z(\alpha) R_{X'}(\beta) R_Z(\gamma)$$

# Example: Yaw-Pitch-Roll Convention (ZYX Euler Angles)



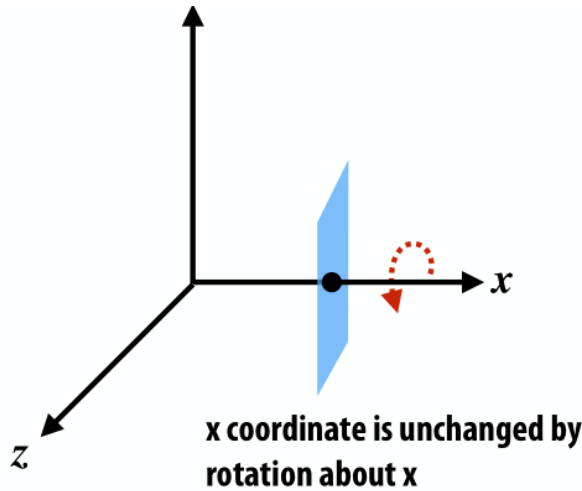
- Common for describing the orientation of aircrafts
- 1. Rotate about Z-axis by **yaw** angle
- 2. Rotate about Y-axis of the new frame by **pitch** angle
- 3. Rotate about X-axis of the new frame by **roll** angle

$$R = R_z(\text{yaw}) R_y(\text{pitch}) R_x(\text{roll})$$

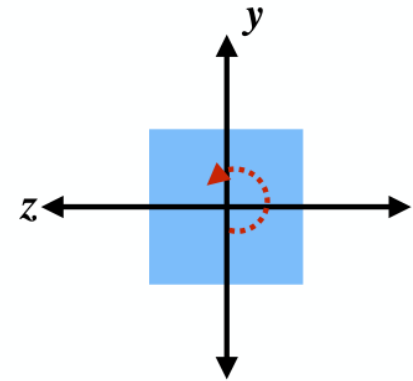
# Recall: Rotation Matrix in 3D

## Rotation about x axis:

$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$



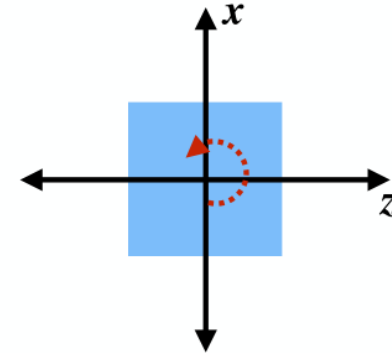
View looking down -x axis:



## Rotation about y axis:

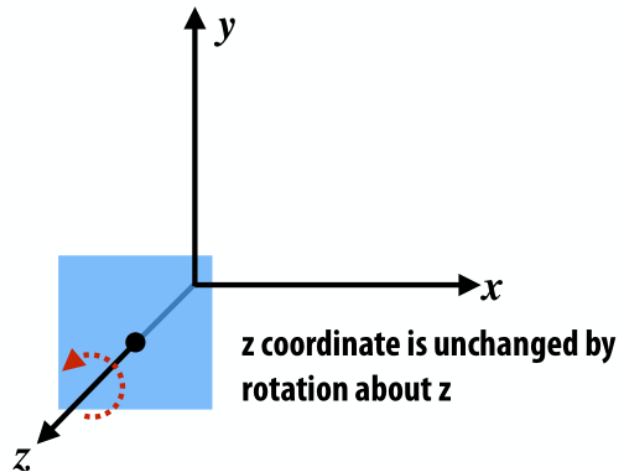
$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix}$$

View looking down -y axis:



## Rotation about z axis:

$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

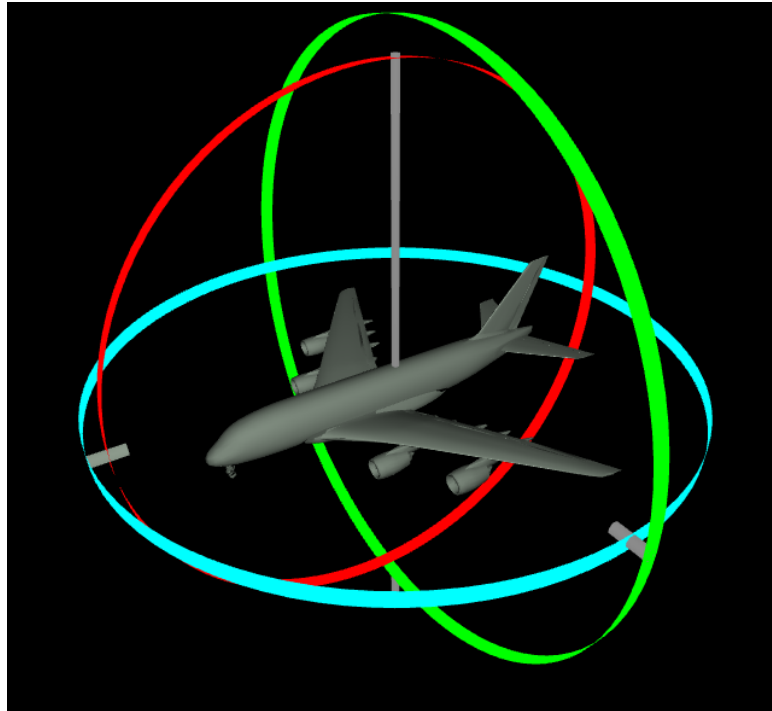


# Euler Angles

---

- Possible 12 combinations
  - XYZ, XYX, XZY, XZX
  - YZX, YZY, YXZ, YXY
  - ZXY, ZXZ, ZYX, ZYZ
- (Combination is possible as long as the same axis does not appear consecutively.)

# [Practice] Euler Angles Online Demo

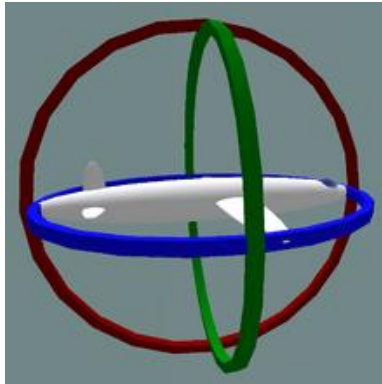


<http://www.ctralie.com/Teaching/COMPS/CI290/Materials/EulerAnglesViz/>

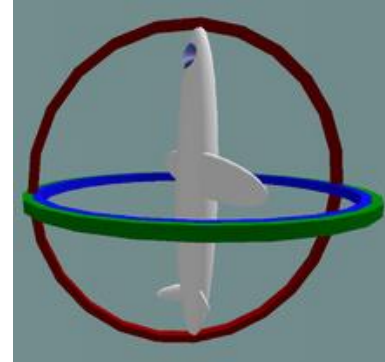
- Try to change yaw, pitch, roll angles

# Gimbal Lock

- One potential problem that Euler angles can suffer from is 'gimbal lock'
- This results when two axes effectively line up, resulting in a temporary **loss of a degree of freedom**



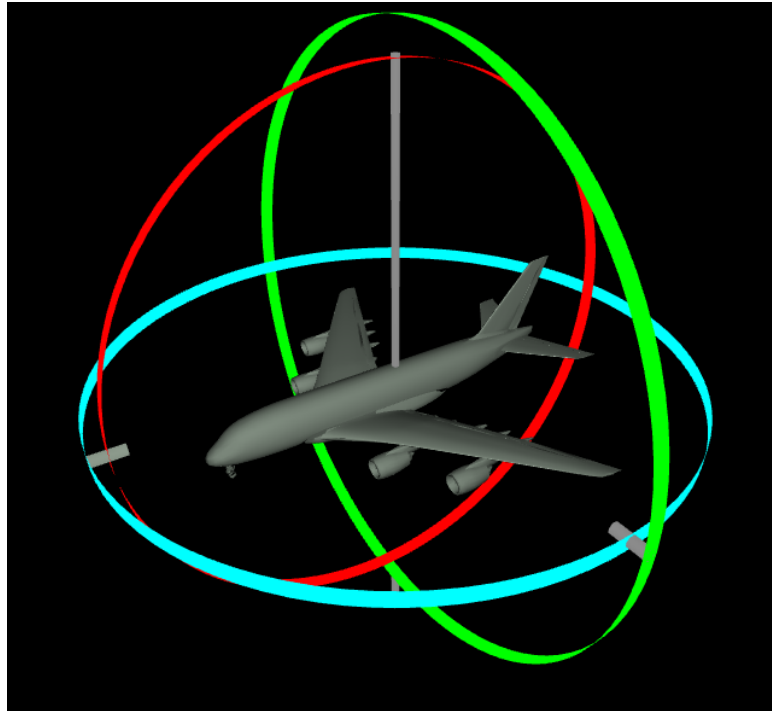
Normal situation.  
The plane can rotate  
in any directions



Gimbal lock:  
two out of the three  
gimbals are in the same  
plane, one DoF is lost

- Euler angles have **singularities**, i.e., it loses DoFs (can't move in a certain direction) at some configurations

# [Practice] Gimbal Lock



<http://www.ctralie.com/Teaching/COMPS/CI290/Materials/EulerAnglesViz/>

- Make gimbal lock by aligning two of three rotation axes
  - Set pitch to 90 degrees



# [Practice] Euler Angles in OpenGL

---

- Start with the practice code from the lecture “7 - Lighting & Shading”,
- Just replace render() function

```

def render():
    global gCamAng, gCamHeight

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)

    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, 1, 1,10)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    gluLookAt(5*np.sin(gCamAng),gCamHeight,5*np.cos(gCamAng), 0,0,0, 0,1,0)

    # draw global frame
    drawFrame()

    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glEnable(GL_RESCALE_NORMAL)

    # set light properties
    lightPos = (4.,5.,6.,1.)
    glLightfv(GL_LIGHT0, GL_POSITION,
lightPos)

    ambientLightColor = (.1,.1,.1,1.)
    diffuseLightColor = (1.,1.,1.,1.)
    specularLightColor = (1.,1.,1.,1.)
    glLightfv(GL_LIGHT0, GL_AMBIENT,
ambientLightColor)
    glLightfv(GL_LIGHT0, GL_DIFFUSE,
diffuseLightColor)
    glLightfv(GL_LIGHT0, GL_SPECULAR,
specularLightColor)

```

```

# ZYX Euler angles
t = glfw.get_time()
xang = t
yang = np.radians(30)
zang = np.radians(30)
M = np.identity(4)
Rx = np.array([[1,0,0],
               [0, np.cos(xang), -np.sin(xang)],
               [0, np.sin(xang), np.cos(xang)]])
Ry = np.array([[np.cos(yang), 0, np.sin(yang)],
               [0,1,0],
               [-np.sin(yang), 0, np.cos(yang)]])
Rz = np.array([[np.cos(zang), -np.sin(zang), 0],
               [np.sin(zang), np.cos(zang), 0],
               [0,0,1]])
M[:3,:3] = Rz @ Ry @ Rx
glMultMatrixf(M.T)

# # The same ZYX Euler angles with OpenGL functions
# glRotate(30, 0,0,1)
# glRotate(30, 0,1,0)
# glRotate(np.degrees(xang), 1,0,0)

glScalef(.25,.25,.25)

# draw cubes
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, (.5,.5,.5,1.))
drawCube_glDrawArray()

glTranslatef(2.5,0,0)
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, (1.,0.,0.,1.))
drawCube_glDrawArray()

glTranslatef(-2.5,2.5,0)
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, (0.,1.,0.,1.))
drawCube_glDrawArray()

glTranslatef(0,-2.5,2.5)
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, (0.,0.,1.,1.))
drawCube_glDrawArray()
glDisable(GL_LIGHTING)

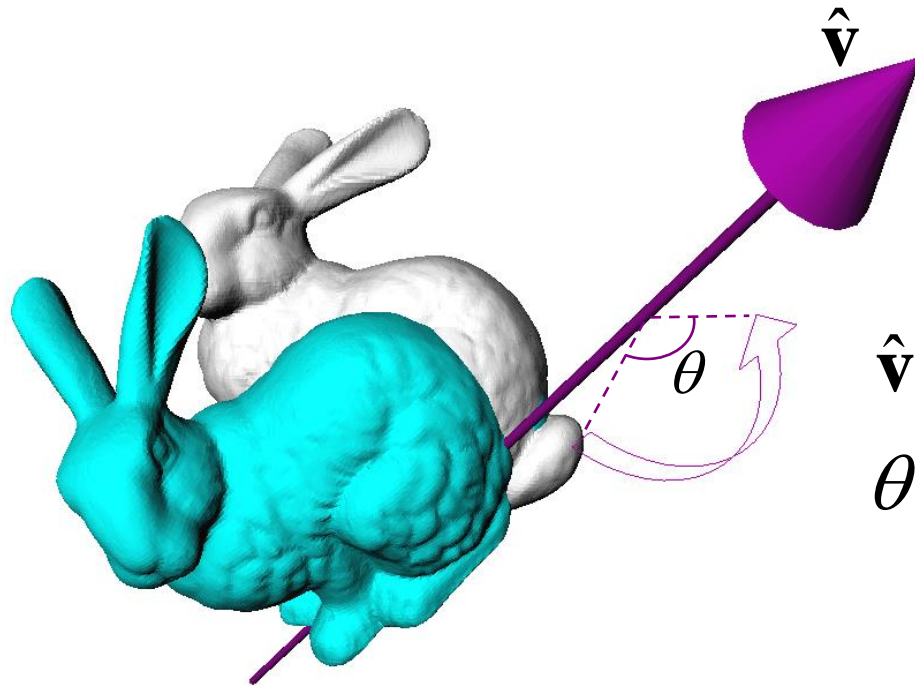
```

# Quiz #1

---

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click “Polls”
  
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - e.g. **2017123456: 4)**
  
- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

# Rotation Vector (Axis-Angle)



$\hat{\mathbf{v}}$  : rotation axis (unit vector)

$\theta$  : scalar angle

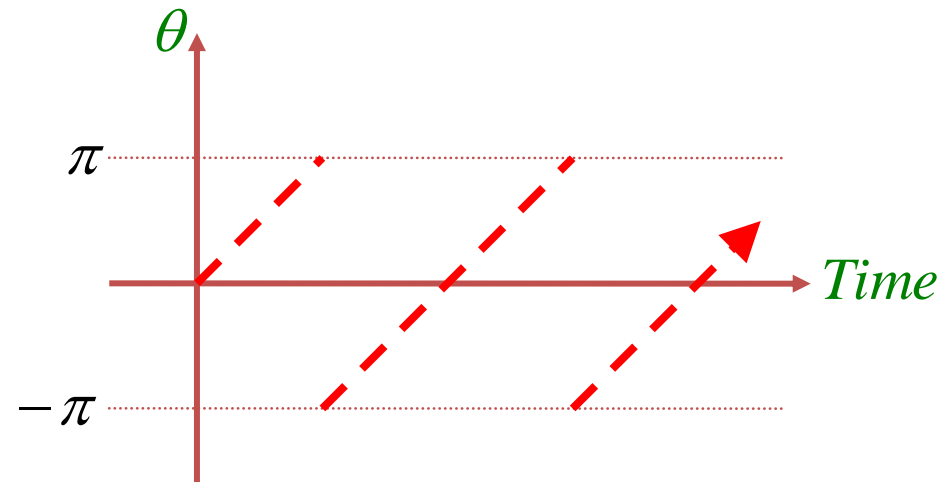
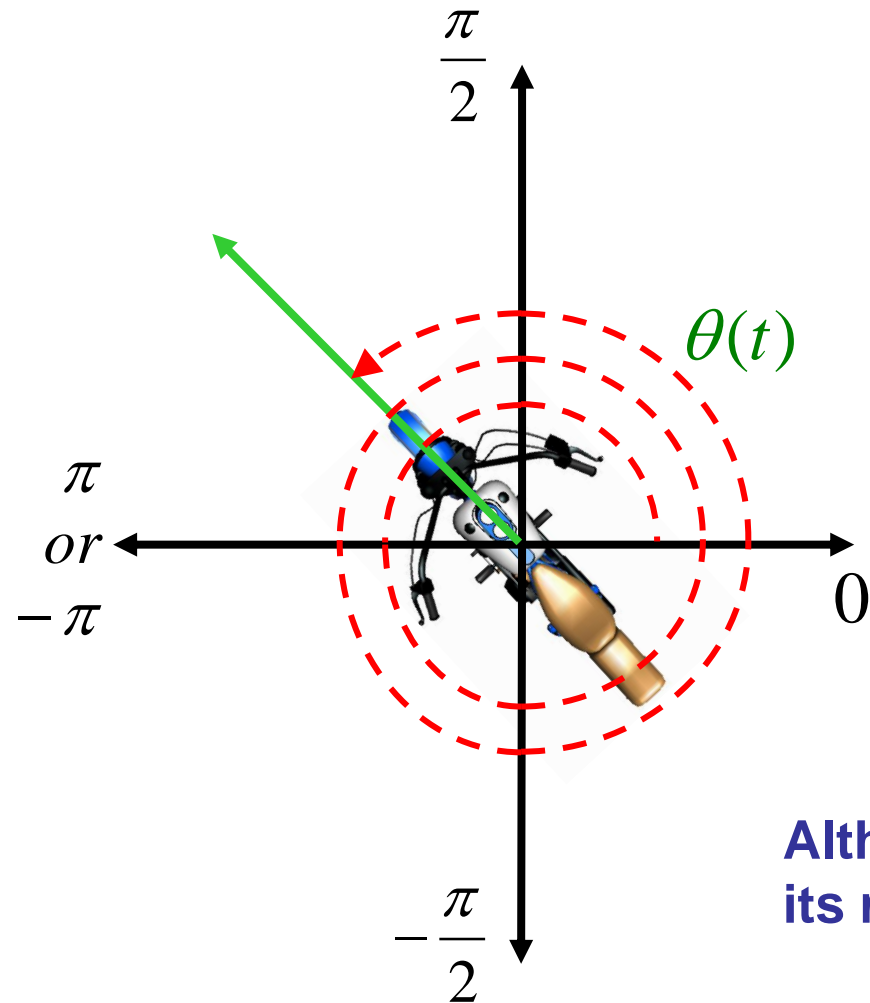
- Rotation vector  $\mathbf{v} = \theta \hat{\mathbf{v}} = (x, y, z)$
- Axis-Angle  $(\theta, \hat{\mathbf{v}})$

# Discontinuity (or Many-to-one Correspondences) Problem

---

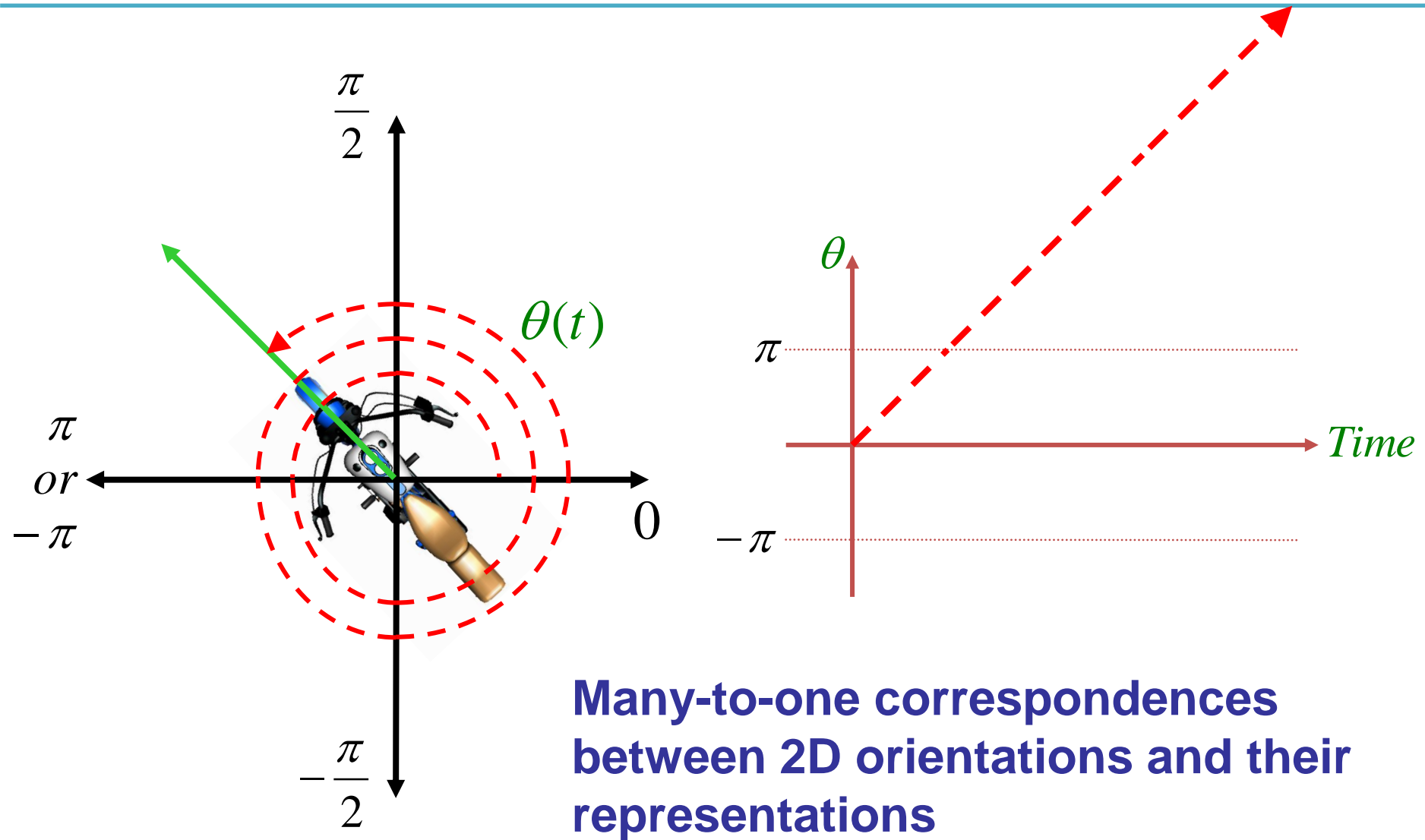
- Euler angles and rotation vector use 3 parameters.
- But, expressing 3D orientation using 3 parameters has problems:
- Euler angles
  - Discontinuity (or many-to-one correspondences)
  - **Gimbal lock**
- Rotation Vector (Axis-Angle)
  - Discontinuity (or many-to-one correspondences)

# Problem of Representing 2D Orientation by Angle $\theta$ : Discontinuity



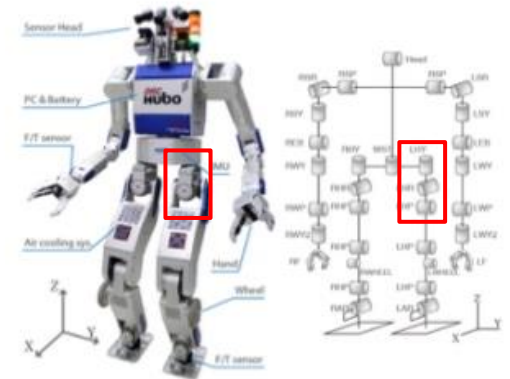
Although the motion is continuous, its representation could be discontinuous

# Problem of Representing 2D Orientation by Angle $\theta$ : Many-to-one Correspondence



# Discontinuity (or Many-to-one Correspondences) Problem

- To avoid these problems, we need more parameters than DOFs
  - Rotation matrices
  - Unit quaternions
- But Euler angles is still meaningful because
  - It's the most common way to implement actuated 3 DOF rotational joints in real world.





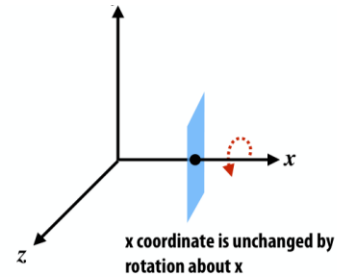
# Rotation Matrices

- Rotation in 3D space can be represented as 3x3 matrix:

## Rotation matrix about x, y, z axis

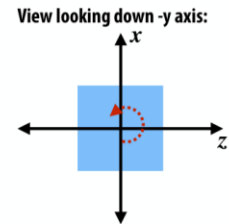
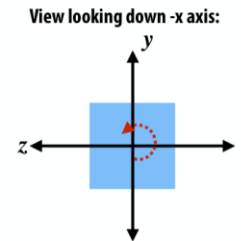
Rotation about x axis:

$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$



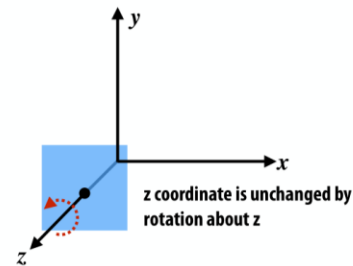
Rotation about y axis:

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$



Rotation about z axis:

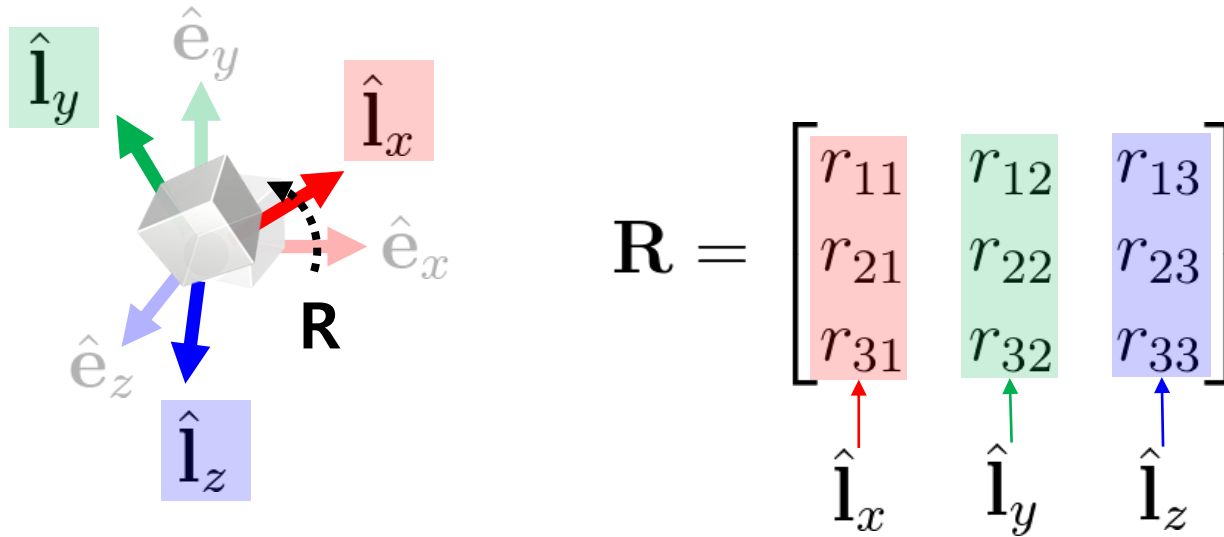
$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Rotation matrix from ZXZ Euler angles

$$\mathbf{R} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \beta & -\sin \beta \\ 0 & \sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Meaning of Rotation Matrix



- A rotation matrix defines
  - **Orientation** of new rotated frame or,
  - **Rotation** from a global frame to be that rotated frame

# Mathematical Properties of Rotation Matrix

---

- A square matrix  $\mathbf{R}$  is a rotation matrix if and only if

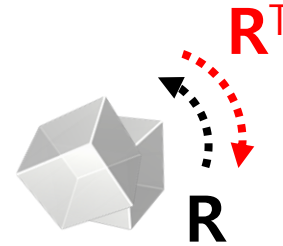
$$\boxed{1. \mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}} \ \&\amp; \ \boxed{2. \det(\mathbf{R}) = 1}$$

- For details, see *9 - reference-rotmat-properties.pdf*
- A rotation matrix is an **orthogonal matrix with determinant 1**
  - Sometimes it is called *special orthogonal matrix*
  - A set of rotation matrices of size 3 forms a *special orthogonal group,  $SO(3)$*

# Geometric Properties of Rotation Matrix

- $\mathbf{R}^T$  is an inverse rotation of  $\mathbf{R}$

– Because,  $\mathbf{R}\mathbf{R}^T = \mathbf{I} \iff \mathbf{R}^{-1} = \mathbf{R}^T$



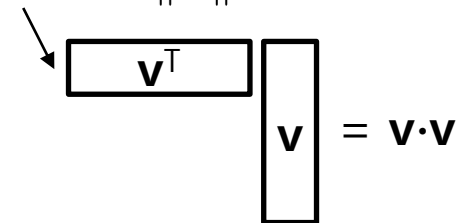
- $\mathbf{R}_1\mathbf{R}_2$  is a rotation matrix as well (composite rotation)

– proof)  $(\mathbf{R}_1\mathbf{R}_2)^T(\mathbf{R}_1\mathbf{R}_2) = \mathbf{R}_2^T\mathbf{R}_1^T\mathbf{R}_1\mathbf{R}_2 = \mathbf{R}_2^T\mathbf{R}_2 = \mathbf{I}$

and  $\det(\mathbf{R}_1\mathbf{R}_2) = \det(\mathbf{R}_1) \cdot \det(\mathbf{R}_2) = 1$

- The length of vector  $\mathbf{v}$  is not changed after applying a rotation matrix  $\mathbf{R}$

– proof)  $\|\mathbf{R}\mathbf{v}\|^2 = (\mathbf{R}\mathbf{v})^T(\mathbf{R}\mathbf{v}) = \mathbf{v}^T\mathbf{R}^T\mathbf{R}\mathbf{v} = \mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2$


$$\boxed{\mathbf{v}^T} \boxed{\mathbf{v}} = \mathbf{v} \cdot \mathbf{v}$$

# [Practice] Properties of Rotation Matrix

---

- Start with the previous practice code
- Just replace `render()` function

```

def render():
    global gCamAng, gCamHeight
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, 1, 1,10)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

gluLookAt(5*np.sin(gCamAng),gCamHeight,5*np.cos(gCamAng),
0,0,0, 0,1,0)
    drawFrame() # draw global frame

    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glEnable(GL_RESCALE_NORMAL) # rescale normal

    glLightfv(GL_LIGHT0, GL_POSITION, (1.,2.,3.,1.))
    glLightfv(GL_LIGHT0, GL_AMBIENT, (.1,.1,.1,1.))
    glLightfv(GL_LIGHT0, GL_DIFFUSE, (1.,1.,1.,1.))
    glLightfv(GL_LIGHT0, GL_SPECULAR, (1.,1.,1.,1.))

    # ZYX Euler angles
    t = glfw.get_time()
    xang = t
    yang = np.radians(30)
    zang = np.radians(30)
    M = np.identity(4)
    Rx = np.array([[1,0,0],
                   [0, np.cos(xang), -np.sin(xang)],
                   [0, np.sin(xang), np.cos(xang)]])
    Ry = np.array([[np.cos(yang), 0, np.sin(yang)],
                   [0,1,0],
                   [-np.sin(yang), 0, np.cos(yang)]])
    Rz = np.array([[np.cos(zang), -np.sin(zang), 0],
                   [np.sin(zang), np.cos(zang), 0],
                   [0,0,1]])

```

```

R = Rz @ Ry @ Rx
## check inverse rotation
# R = Rz @ Ry @ Rx.T

## check R @ R.T
# print(R @ R.T)

## check determinant
# print(np.linalg.det(R))

M[:3,:3] = R
glMultMatrixf(M.T)

glScalef(.25,.25,.25)

# draw cubes
glMaterialfv(GL_FRONT,
GL_AMBIENT_AND_DIFFUSE, (.5,.5,.5,1.))
drawCube_glDrawArray()

glTranslatef(2.5,0,0)
glMaterialfv(GL_FRONT,
GL_AMBIENT_AND_DIFFUSE, (1.,0.,0.,1.))
drawCube_glDrawArray()

glTranslatef(-2.5,2.5,0)
glMaterialfv(GL_FRONT,
GL_AMBIENT_AND_DIFFUSE, (0.,1.,0.,1.))
drawCube_glDrawArray()

glTranslatef(0,-2.5,2.5)
glMaterialfv(GL_FRONT,
GL_AMBIENT_AND_DIFFUSE, (0.,0.,1.,1.))
drawCube_glDrawArray()

glDisable(GL_LIGHTING)

```

# Rotation Matrix for Rotation about an Arbitrary Axis

---

- Euler's Rotation Theorem tells us that an arbitrary 3D rotation equals to one rotation around an axis.
- Then, how to compute the rotation matrix for given axis vector  $u=(u_x, u_y, u_z)$  by angle  $\theta$ ?
- A naive, inefficient method:
  - Step 1: rotate the axis  $u$  so that it is aligned with the Z-axis
  - Step 2: rotate about the Z-axis by the angle  $\theta$
  - Step 3: rotate the Z-axis back to the original axis
  - For details, see *9 - reference-naive-rotvec2rotmat.pdf*

# Rotation Matrix for Rotation about an Arbitrary Axis

---

- More efficient solution: Rodrigues' rotation formula
- Rotation about a normalized axis vector  $\mathbf{u}=(u_x, u_y, u_z)$  by angle  $\theta$ :

$$R = \begin{bmatrix} \cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_z u_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2 (1 - \cos \theta) \end{bmatrix}$$

(You do not have to memorize this)



# Quiz #2

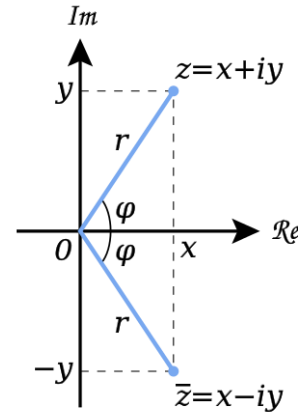
---

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click “Polls”
  
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - e.g. **2017123456: 4)**
  
- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

# Quaternions

- Complex numbers can be used to represent 2D rotations

$$z = x + iy \quad \text{where} \quad i^2 = -1$$



- Basic idea: Quaternion is its extension to 3D space

$$q = w + ix + jy + kz \quad \text{where}$$

$$\begin{aligned} i^2 &= j^2 = k^2 = ijk = -1 \\ ij &= k, \quad jk = i, \quad ki = j \\ ji &= -k, \quad kj = -i, \quad ik = -j \end{aligned}$$

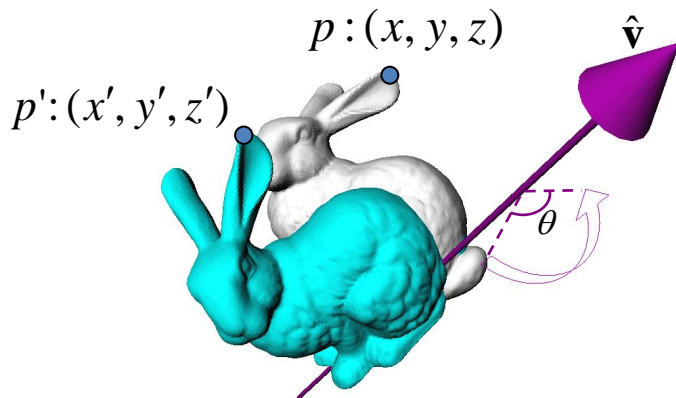
# Unit Quaternions

- Unit quaternions represent 3D rotations

$$\begin{aligned}\mathbf{q} &= w + ix + jy + kz \\ &= (w, x, y, z) \\ &= (w, \mathbf{v})\end{aligned}$$

$$w^2 + x^2 + y^2 + z^2 = 1$$

- Rotation about axis  $\hat{\mathbf{v}}$  by angle  $\theta$



$$\mathbf{q} = \left( \cos \frac{\theta}{2}, \hat{\mathbf{v}} \sin \frac{\theta}{2} \right)$$

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1} \quad \text{where} \quad \mathbf{p} = (0, x, y, z)$$

For details, see *9 - reference-quaternions.pdf*

# Which Representation to Use?

---

- 3D orientation & rotation representation
  - Euler angles
  - Rotation Vector (Axis-Angle)
  - Rotation matrices
  - Unit quaternions
- Which one to use?
- General recommendation: **rotation matrices** or **unit quaternions**.
  - Reason: Euler angles and rotation vector have gimbal lock or discontinuity (many-to-one correspondences) problems.
- But you may need other representations depending on the context.
  - Euler angles are useful for hardware implementation of ball joints.

# Which Representation to Use?

---

- Rotation matrices and unit quaternions do not have discontinuity or gimbal lock problems
  - Because they use more parameters (rotation matrix: 9, unit quaternion: 4) than DOFs of 3D orientation/rotation (3)
- Rotation matrices vs. Unit quaternions ?

# Rotation Matrix vs. Unit Quaternion

---

- Equivalent in many aspects
  - Redundant
  - No singularity
  - Can be converted from & to axis-angle representation
- Why quaternions ?
  - Fewer parameters
  - Simpler algebra
  - Easy to fix numerical error
- Why rotation matrices ?
  - One-to-one correspondence (quaternion: 2-to-1 correspondence)
  - Handle rotation and translation in a uniform way
    - Eg) 4x4 homogeneous matrices

# Conversion Between Representations

---

- **Rotation vector** → **Rotation matrix**
  - Rodrigues' rotation formula, ...
- **Rotation matrix** → **Rotation vector**
  - We'll see one of the methods soon.
- **Euler angles** → **Rotation matrix**
  - Building canonical rotation matrices ( $\mathbf{R}_x$ ,  $\mathbf{R}_y$ ,  $\mathbf{R}_z$ ) and composing them
- **Rotation matrix** → **Euler angles**
  - Not covered in this class (but you can easily google it)
- **Unit quaternion** ↔ **Rotation matrix**
  - Not covered in this class (but you can easily google it)

---

# **3D Orientation Interpolation**

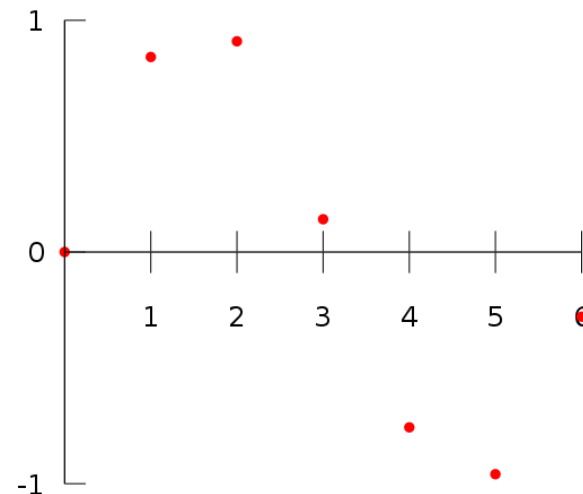


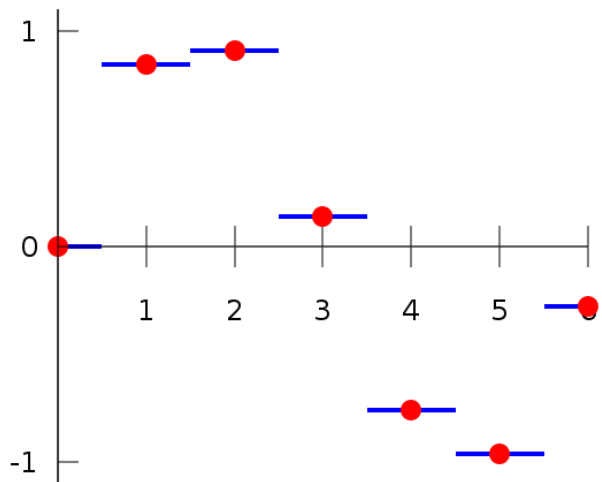
# Interpolation

- A method of constructing new data points within the range of a discrete set of known data points.
- In other words, guessing unknown function  $f(x)$  from known data points  $(x_i, f(x_i))$ .

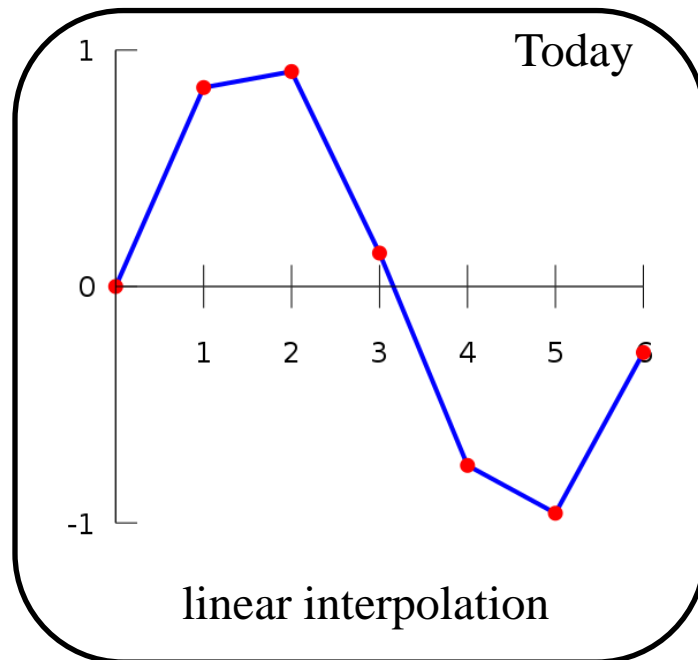
Ex) Known data points

x	f(x)
0	0
1	0.8415
2	0.9093
3	0.1411
4	-0.7568
5	-0.9589
6	-0.2794

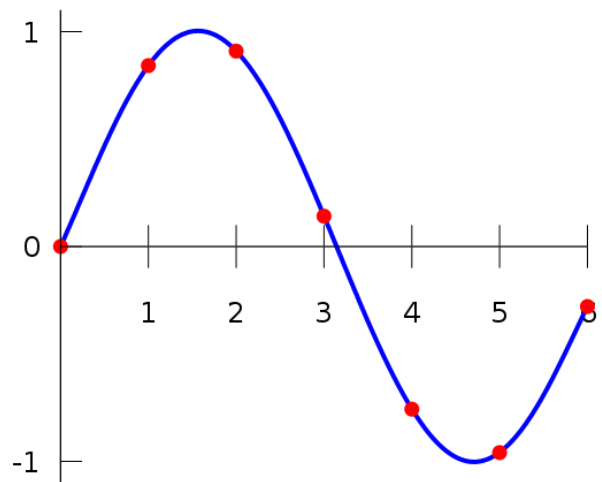




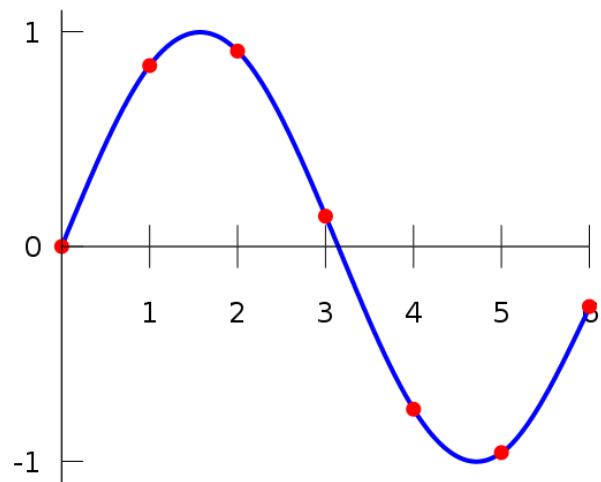
nearest-neighbor interpolation



linear interpolation

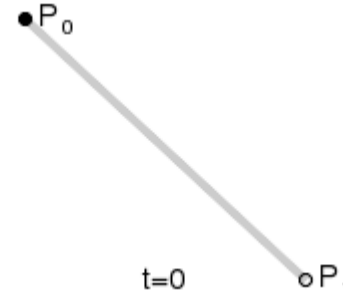
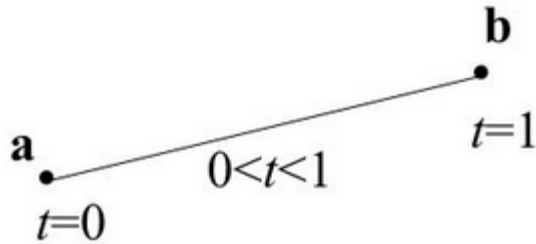


polynomial interpolation



spline interpolation

# Linear Interpolation



[https://upload.wikimedia.org/wikipedia/commons/0/00/B%C3%A9zier\\_1\\_big.gif](https://upload.wikimedia.org/wikipedia/commons/0/00/B%C3%A9zier_1_big.gif)

$$\text{lerp}(\mathbf{a}, \mathbf{b}, t) = (1 - t)\mathbf{a} + t\mathbf{b}$$

```
float lerp(float v0, float v1, float t)
{
    return (1 - t) * v0 + t * v1;
}
```

- A straight line between two points
- This is fine for translations

# Linear Interpolation for 3D Orientations?

---

- Recall: 3D orientation & rotation representation
  - Euler angles
  - Rotation vector
  - Rotation matrices
  - Unit quaternions
- How to linearly interpolate **two orientations** in these representations?

# Interpolating Each Element of Rotation Matrix?

- Let's try to interpolate  $\mathbf{R}_0$ (identity) and  $\mathbf{R}_1$ (rotation by  $90^\circ$  about x-axis)

$$\text{lerp}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, 0.5\right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & -0.5 \\ 0 & 0.5 & 0.5 \end{bmatrix}$$

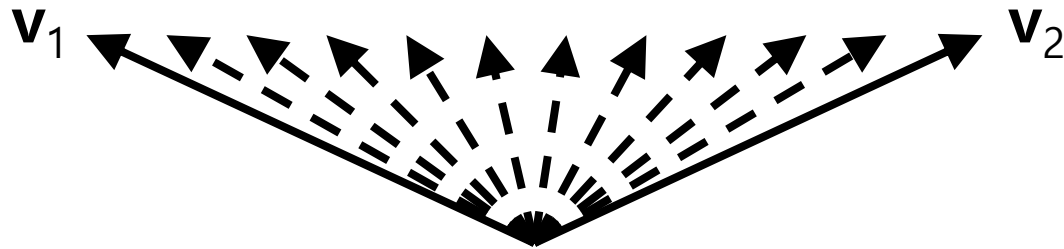
 **is not a rotation matrix!  
does not make sense at all!**

- Similarly, interpolating each number (w, x, y, z) in unit quaternions does not make sense.

# Interpolating Rotation Vector?

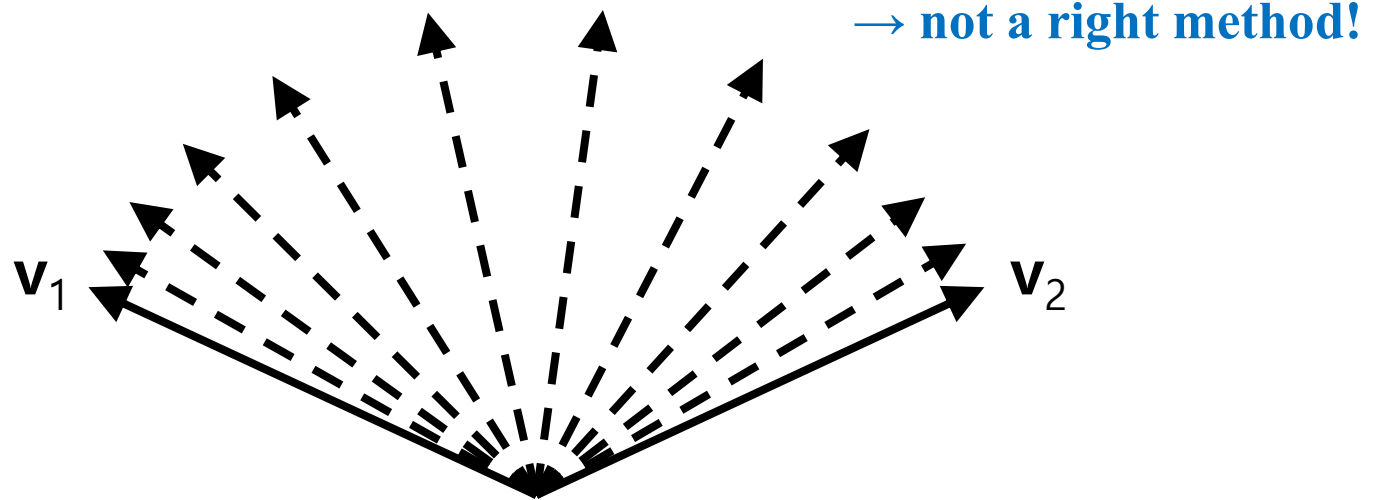
---

- Let's say we have two rotation vectors  $\mathbf{v}_1$  &  $\mathbf{v}_2$  of the same length
- Linear interpolation of  $\mathbf{v}_1$  &  $\mathbf{v}_2$  produces even spacing



# Interpolating Rotation Vector?

- Let's say we have two rotation vectors  $\mathbf{v}_1$  &  $\mathbf{v}_2$  of the same length
- Linear interpolation of  $\mathbf{v}_1$  &  $\mathbf{v}_2$  produces even spacing
- But it's not evenly spaced in terms of orientation!



# Interpolating Euler Angles?

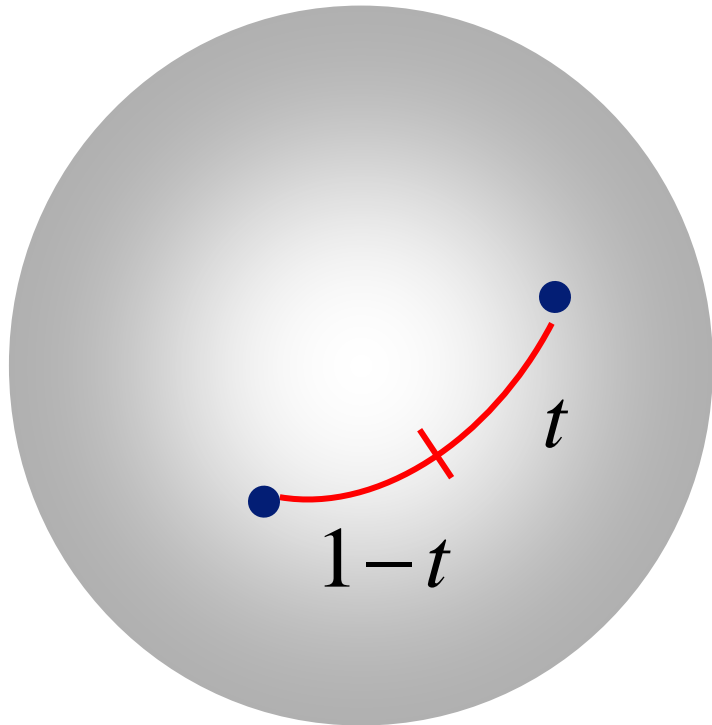
---

- Interpolating two tuples of Euler angles does not make correct result
  - + angular velocity is not constant
  - + still suffer from gimbal lock: jerky movement occurs near gimbal lock configuration



# Slerp

- The right answer: **Slerp** [Shoemake 1985]
  - Spherical linear interpolation
  - Linear interpolation of two orientations



“t” refers power, not transpose

$$\begin{aligned}\text{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) &= \mathbf{R}_1 (\mathbf{R}_1^T \mathbf{R}_2)^t \\ &= \mathbf{R}_1 \exp(t \cdot \log(\mathbf{R}_1^T \mathbf{R}_2))\end{aligned}$$

# Slerp

---

$$\begin{aligned}\text{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) &= \mathbf{R}_1 (\mathbf{R}_1^T \mathbf{R}_2)^t \\ &= \mathbf{R}_1 \exp(t \cdot \log(\mathbf{R}_1^T \mathbf{R}_2))\end{aligned}$$

- $\exp()$ : **rotation vector to rotation matrix**
- $\log()$ : **rotation matrix to rotation vector**
  
- Implication
  - $\mathbf{R}_1^T \mathbf{R}_2$ : difference between orientation  $\mathbf{R}_1$  and  $\mathbf{R}_2$  ( $\mathbf{R}_2(-)\mathbf{R}_1$ )
  - $\mathbf{R}^t$ : scaling rotation (scaling rotation angle)
  - $\mathbf{R}_a \mathbf{R}_b$ : add rotation  $\mathbf{R}_b$  to orientation  $\mathbf{R}_a$  ( $\mathbf{R}_a(+)\mathbf{R}_b$ )

# Exp & Log

- **Exp (exponential): rotation vector to rotation matrix**

- Given normalized rotation axis  $\mathbf{u}=(u_x, u_y, u_z)$ , rotation angle  $\theta$

$$R = \begin{bmatrix} \cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_z u_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2 (1 - \cos \theta) \end{bmatrix}$$

(Rodrigues' rotation formula)

- **Log (logarithm): rotation matrix to rotation vector**

Given rotation matrix  $\mathbf{R}$ , compute axis  $\mathbf{v}$  and angle  $\theta$

$$\theta = \cos^{-1}((R_{11} + R_{22} + R_{33} - 1)/2)$$

$$v_1 = (R_{32} - R_{23})/(2 \sin \theta)$$

$$v_2 = (R_{13} - R_{31})/(2 \sin \theta)$$

$$v_3 = (R_{21} - R_{12})/(2 \sin \theta)$$

→ But this formula has a singularity at  $\theta=k\pi$ , where  $k$  is an integer.

# Algorithm for Log

## Algorithm for Computing the Logarithm of a Rotation Matrix

**Objective:** Given  $R \in SO(3)$ , find  $\omega \in \mathbb{R}^3$ ,  $\|\omega\| = 1$ , and  $\theta \in [0, \pi]$  such that

$$R = e^{[\omega]\theta} = I + \sin \theta [\omega] + (1 - \cos \theta)[\omega]^2. \quad (3.62)$$

(i) If  $\text{tr } R = 3$ , then set  $\omega = 0$ ,  $\theta = 0$ .

(ii) If  $\text{tr } R = -1$ , then set  $\theta = \pi$ , and  $\omega$  to any of the three following vectors that is nonzero:

$$\omega = \frac{1}{\sqrt{2(1 + r_{33})}} \begin{bmatrix} r_{13} \\ r_{23} \\ 1 + r_{33} \end{bmatrix} \quad (3.63)$$

or

$$\omega = \frac{1}{\sqrt{2(1 + r_{22})}} \begin{bmatrix} r_{12} \\ 1 + r_{22} \\ r_{32} \end{bmatrix} \quad (3.64)$$

or

$$\omega = \frac{1}{\sqrt{2(1 + r_{11})}} \begin{bmatrix} 1 + r_{11} \\ r_{21} \\ r_{31} \end{bmatrix}. \quad (3.65)$$

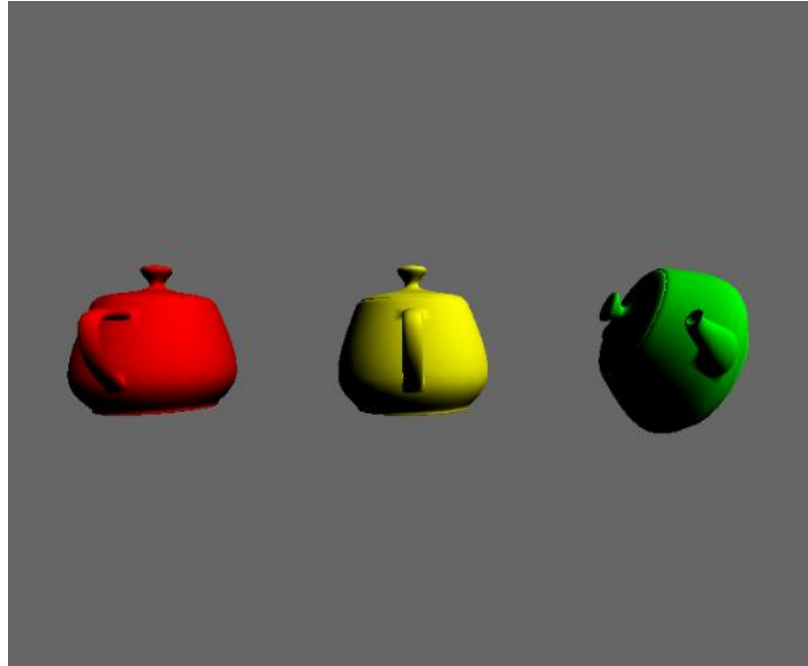
(iii) Otherwise set  $\theta = \cos^{-1} \left( \frac{\text{tr } R - 1}{2} \right) \in [0, \pi)$  and  $[\omega] = \frac{1}{2 \sin \theta} (R - R^T)$ .

See section 3.1.3 of INTRODUCTION TO ROBOTICS for detail:

[http://robotics.snu.ac.kr/fcp/files/\\_pdf\\_files\\_publications/\\_a\\_first\\_coruse\\_in\\_robot\\_mechanics.pdf](http://robotics.snu.ac.kr/fcp/files/_pdf_files_publications/_a_first_coruse_in_robot_mechanics.pdf)

# [Practice] Slerp Online Demo

---



<https://nccastaff.bournemouth.ac.uk/jmacey/WebGL/QuatSlerp/>

- Change “Start Rotation” & “End Rotation”
- Move “Interpolate” slider

# Quiz #3

---

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click “Polls”
  
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - e.g. **2017123456: 4)**
  
- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

# Next Time

---

- Lab in this week:
  - Lab assignment 9
  
- Next lecture:
  - 10 - Kinematics & Animation
  
- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Jehee Lee, SNU, [http://mrl.snu.ac.kr/courses/CourseGraphics/index\\_2017spring.html](http://mrl.snu.ac.kr/courses/CourseGraphics/index_2017spring.html)
  - Prof. Taesoo Kwon, Hanyang Univ., <http://calab.hanyang.ac.kr/cgi-bin/cg.cgi>
  - Prof. Kayvon Fatahalian and Prof. Keenan Crane, CMU, <http://15462.courses.cs.cmu.edu/fall2015/>
  - Prof. Sung-Hee Lee, KAIST