# Computer Graphics

# 9 – Orientation & Rotation

Yoonsang Lee
Spring 2019

# What we've done so far

- Lecture 3 – 5 (Transformation)
- *: Movement & placement*

- Lecture 5 – 6 (Vertex Processing)
- *: Mapping to 2D screen*

- Lecture 7 – 8 (Mesh, Lighting & Shading)
- *: Appearance*

- **Lecture 9 – 10 (Orientation & Rotation, Animation)**
- ***: Movement & placement***

# Topics Covered

- Orientation vs. Rotation
- Degrees of freedom
- 2D orientation & rotation representations
  - Using 1D angle
  - Rotation matrices (2x2)
- 3D orientation & rotation representations
  - Euler angles
  - Axis-angle (Rotation vector)
  - **Rotation matrices**
  - Unit quaternions

# Orientation vs. Rotation, Degrees of freedom

# Orientation vs. Rotation

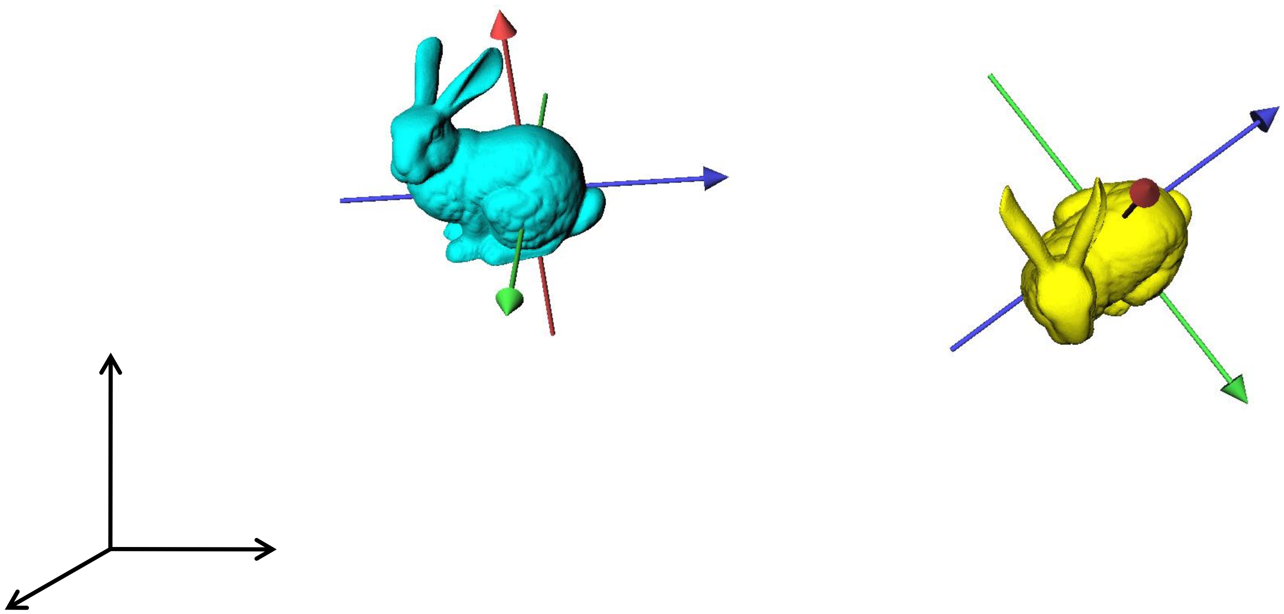
- ***Rotation***
  - Circular movement

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

# Analogy

(point : vector) is similar to (orientation : rotation)

Both represent a sort of (state : movement)



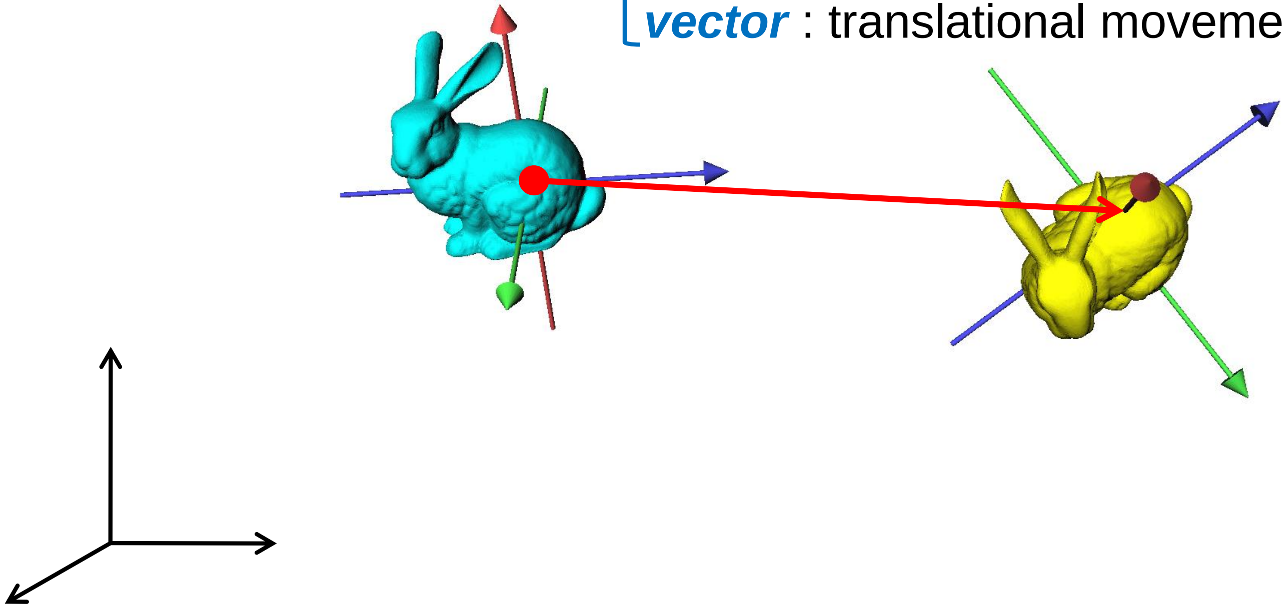
Reference coordinate system

# Analogy

(point : vector) is similar to (orientation : rotation)

Both represent a sort of (state : movement)

***point*** : the 3d location of the bunny
***vector*** : translational movement



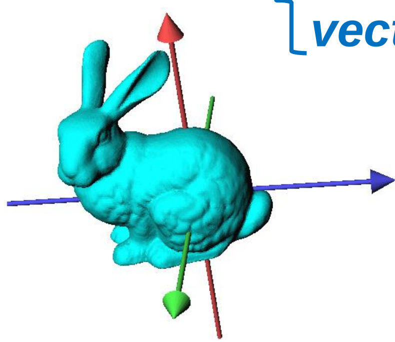
Reference coordinate system

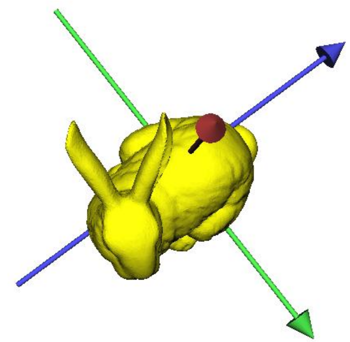
# Analogy

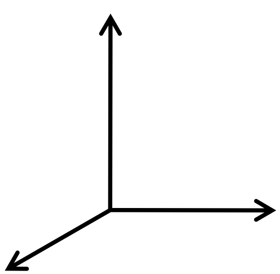
(point : vector) is similar to (orientation : rotation)

Both represent a sort of (state : movement)

***point*** : the 3d location of the bunny
***vector*** : translational movement







Reference coordinate system

***orientation*** : the 3d orientation of the bunny
***rotation*** : circular movement

# Analogy

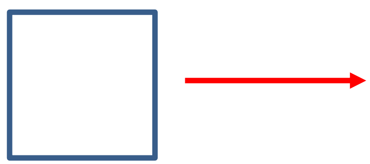
- Point & vector
  - (point) + (point) → (UNDEFINED)
  - (vector) ± (vector) → (vector)
  - (point) ± (vector) → (point)
  - (point) - (point) → (vector)

- Orientation & rotation
  - (orientation) (+) (orientation) → (UNDEFINED)
  - (rotation) (±) (rotation) → (rotation)
  - (orientation) (±) (rotation) → (orientation)
  - (orientation) (-) (orientation) → (rotation)

Not vector addition & subtraction

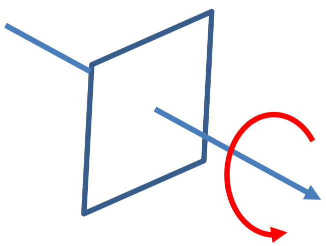
# Degrees of Freedom (DOF)

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



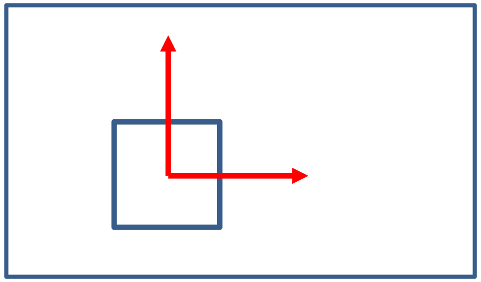
Translation along one direction

: 1 DOF



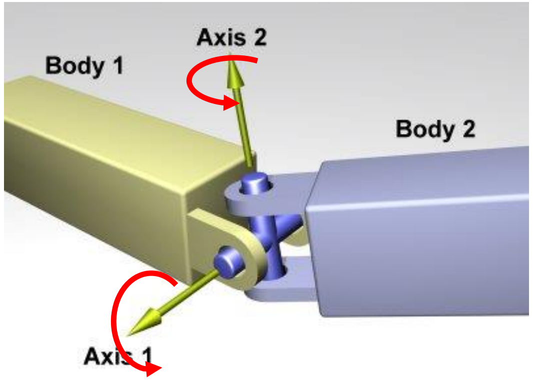
Rotation about an axis

: 1 DOF



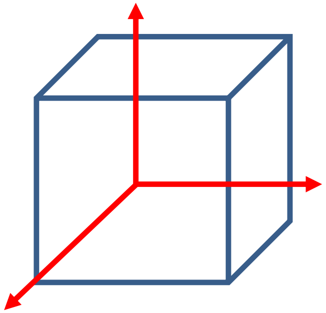
Translation on a plane

: 2 DOF



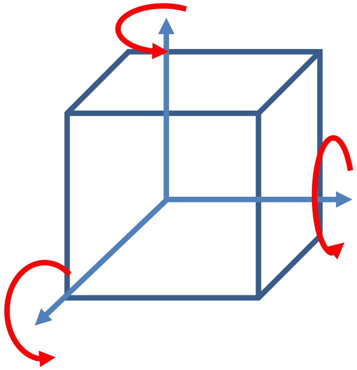

Rotation about two axes

: 2 DOF



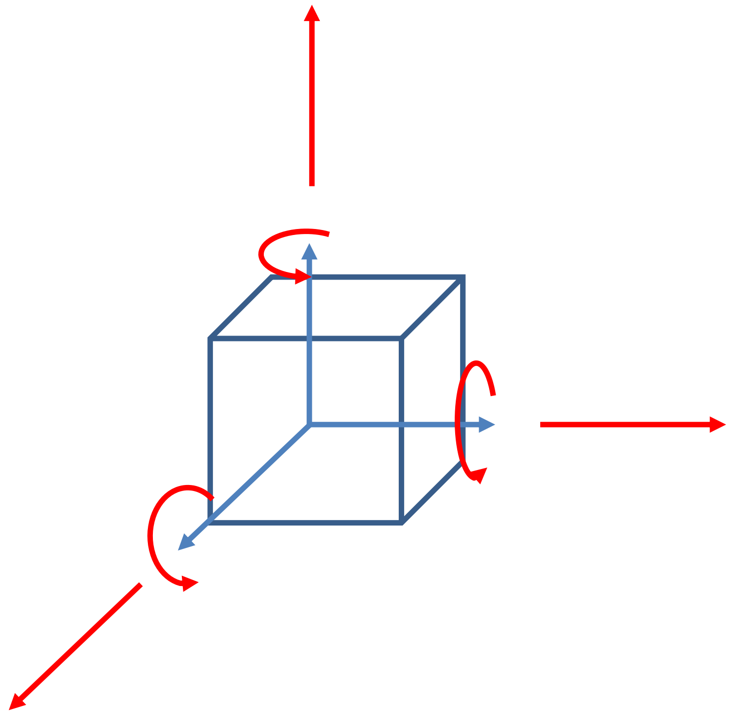
Translation in 3D space

: 3 DOF



Rotation in 3D space

: 3 DOF



Any rigid motion in 3D space

: 6 DOF

# Quiz #1

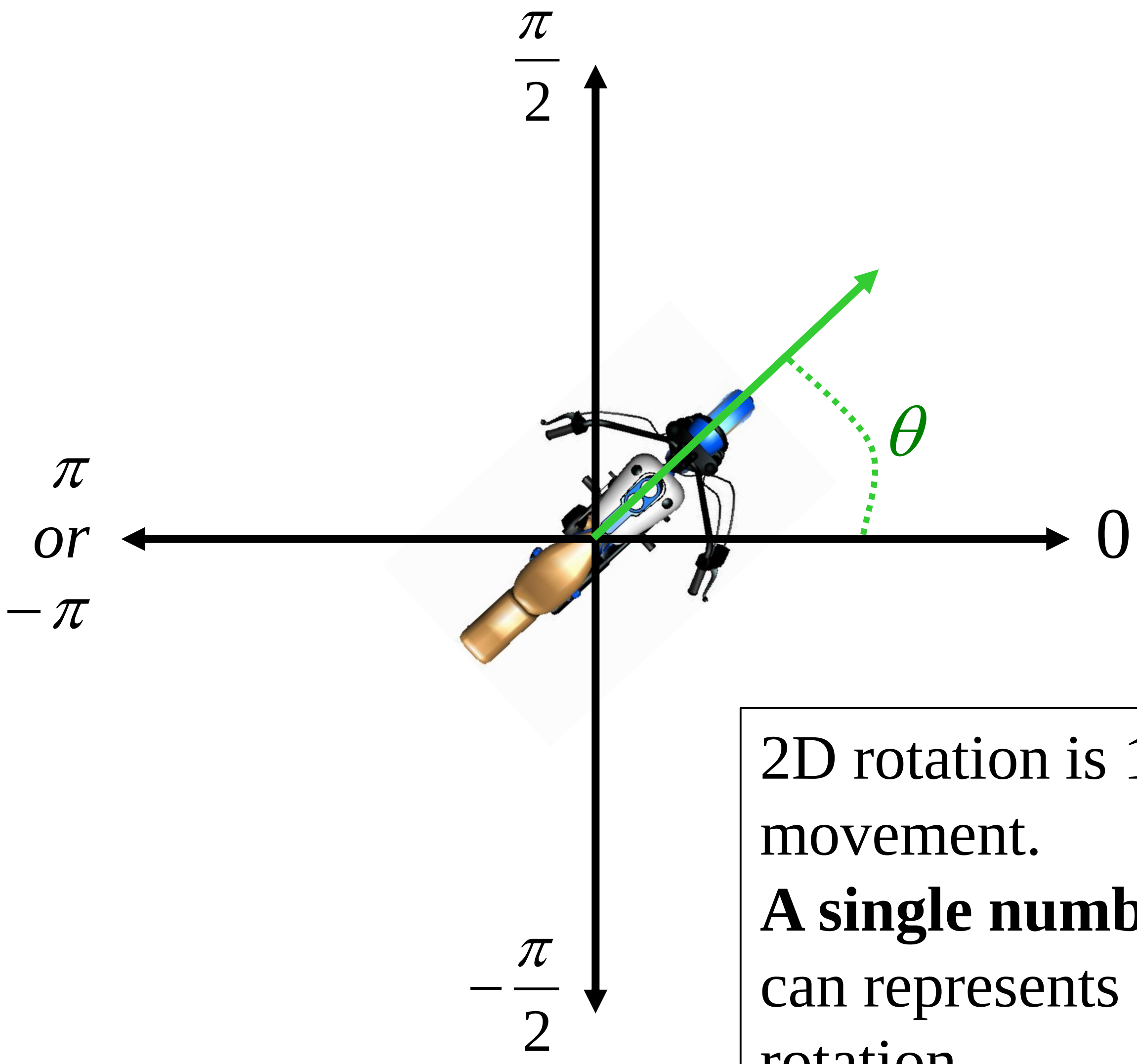
- Go to https://www.slido.com/
- Join #cg-hyu
- Click “Polls”

- 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”.

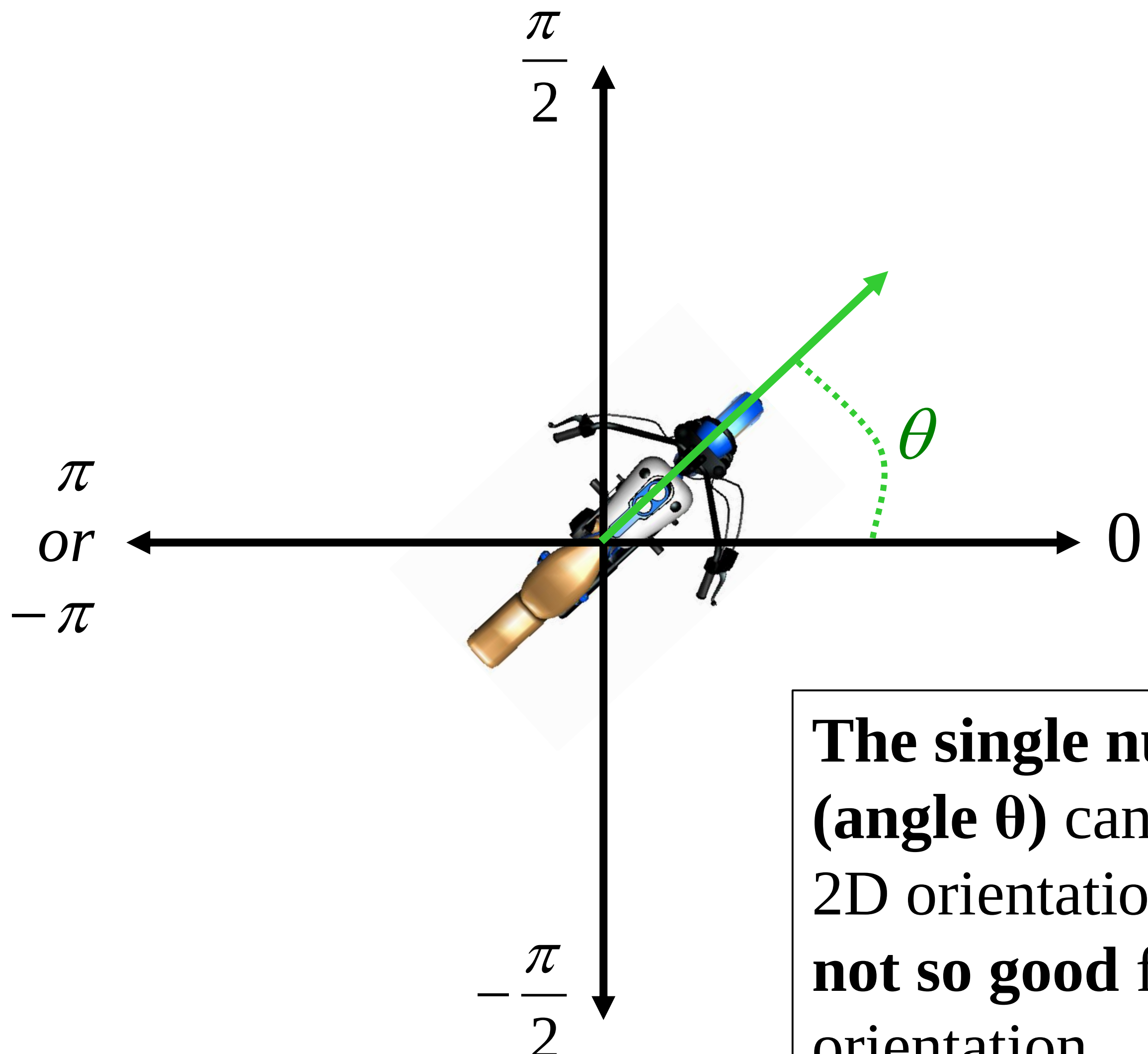
# 2D & 3D orientation & rotation representations

# 2D Rotation



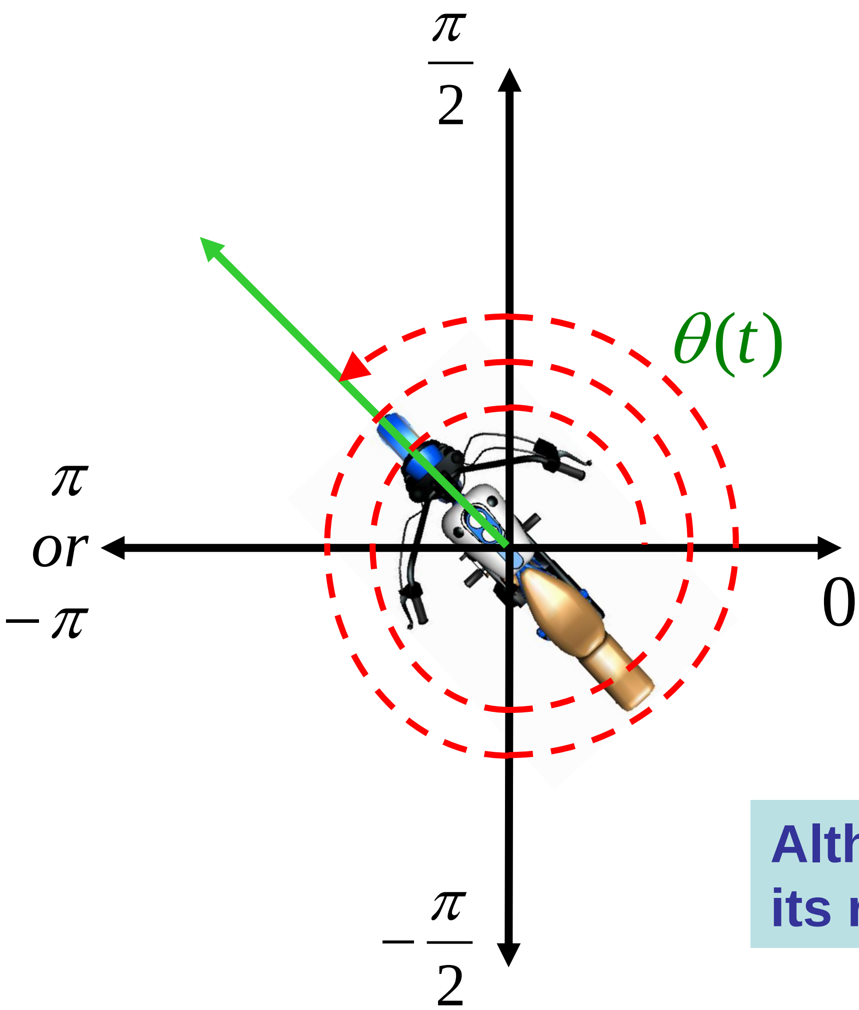

2D rotation is 1 DOF movement.
**A single number (angle θ)** can represents any 2D rotation.

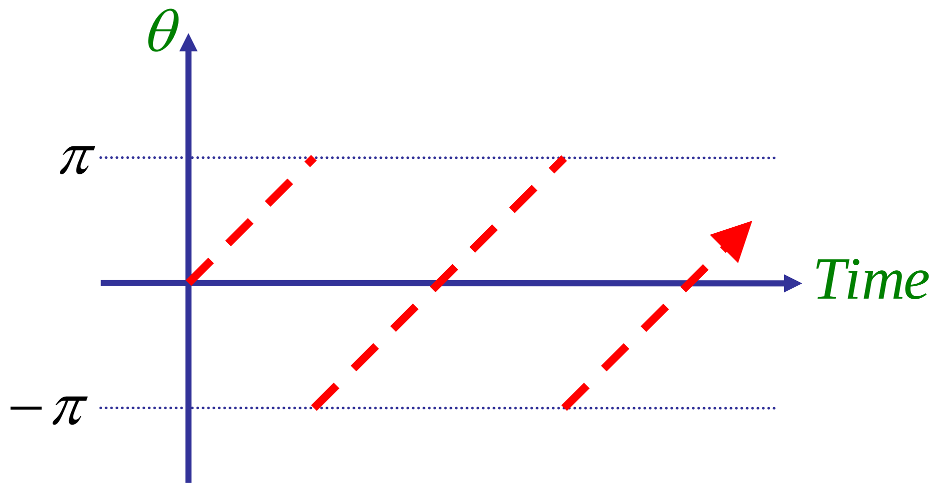
# 2D Orientation




**The single number (angle θ)** can represent 2D orientation, but it's **not so good** for orientation.

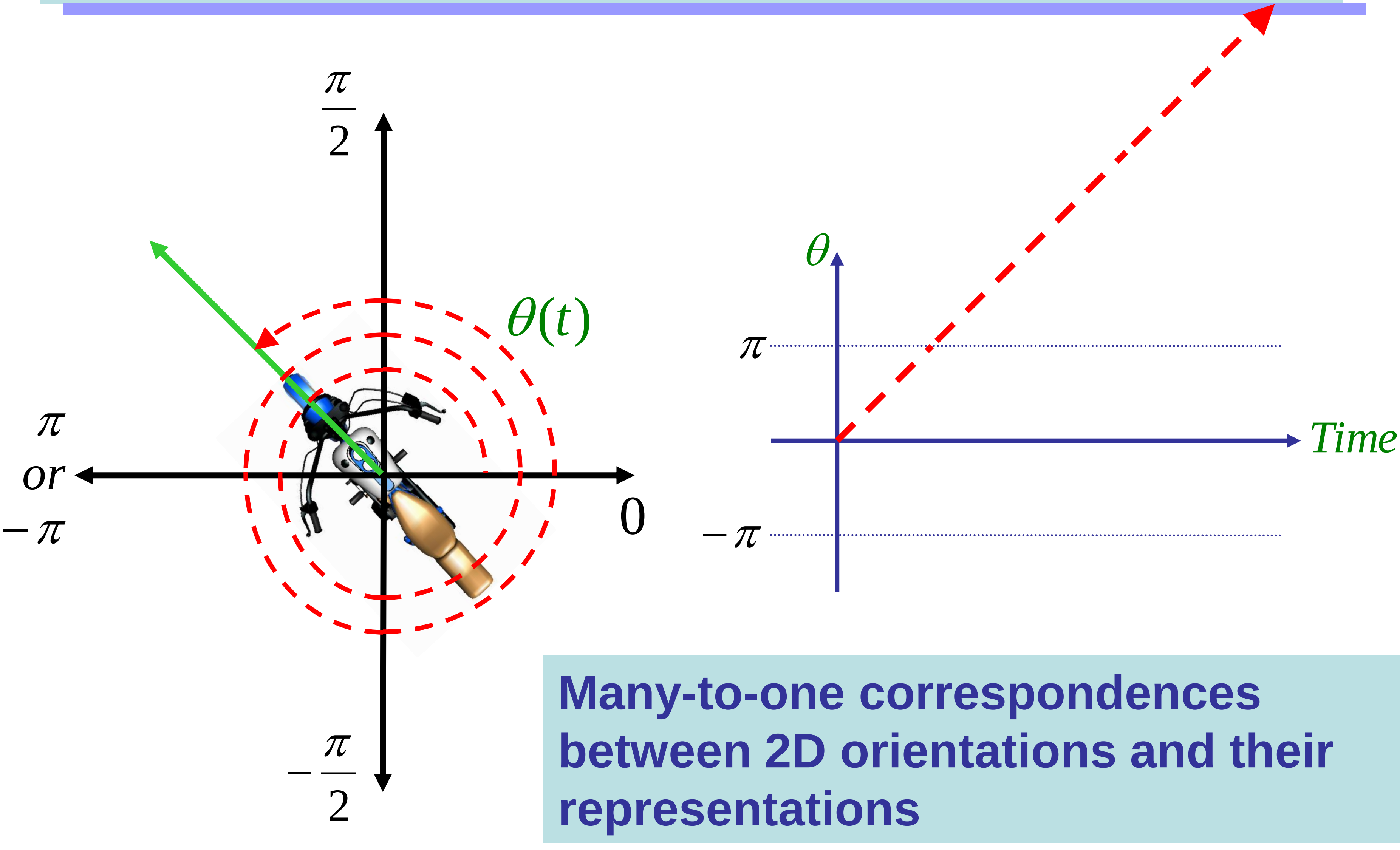
# 2D Orientation




**Although the motion is continuous, its representation could be discontinuous**

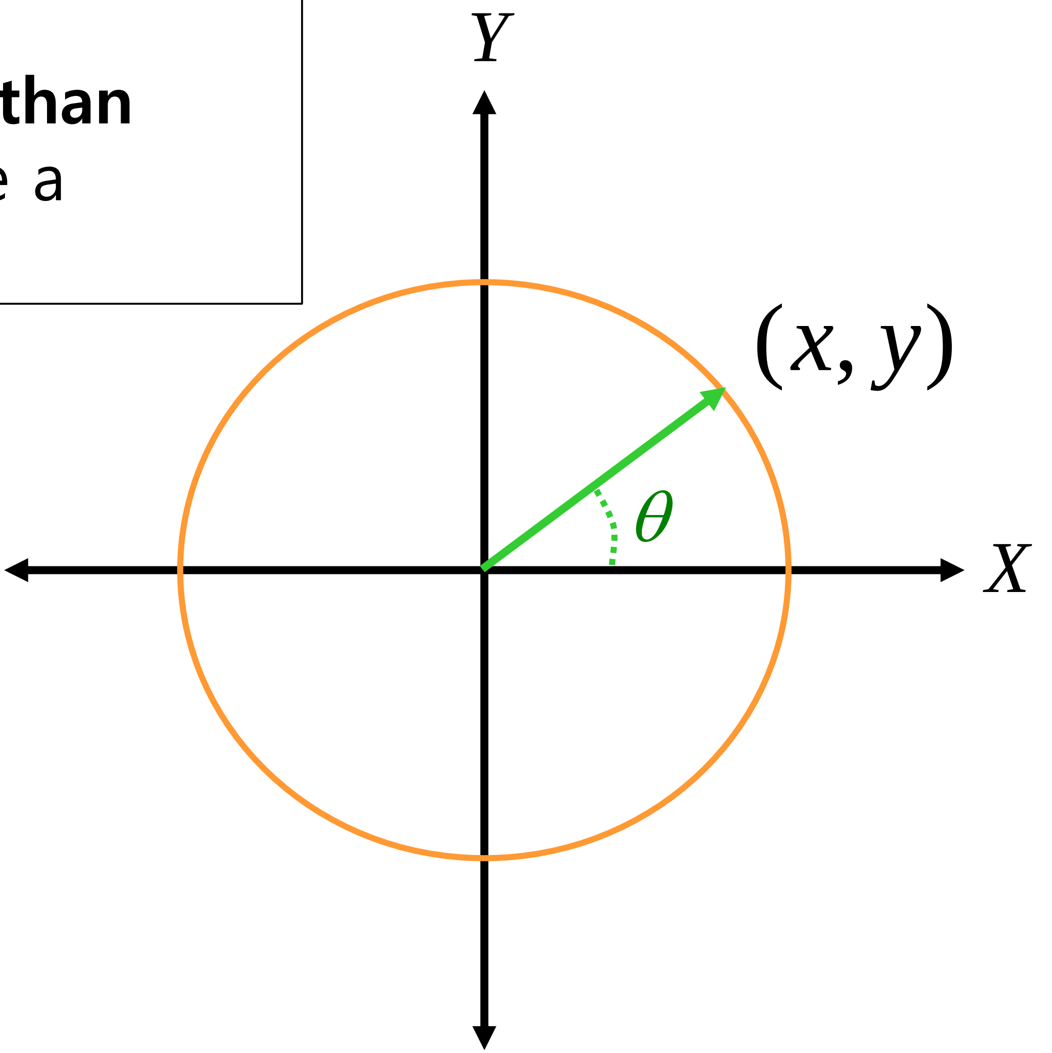
# 2D Orientation




**Many-to-one correspondences between 2D orientations and their representations**

# Extra Parameter

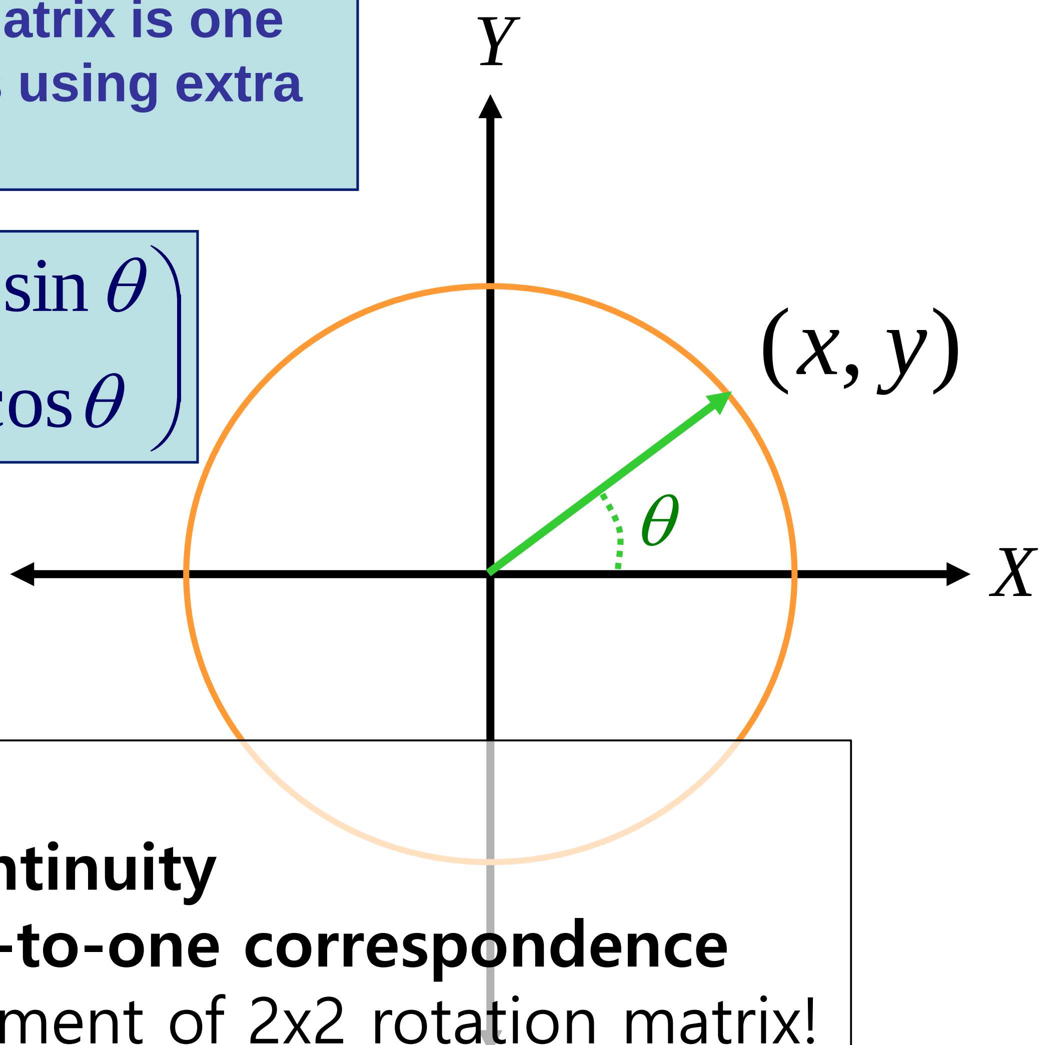
**Using more parameters than DOFs** can be a solution.



# Extra Parameter

**2x2 Rotation matrix is one of the methods using extra parameters**

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$




There are

- **No discontinuity**
- **No many-to-one correspondence**

for each element of 2x2 rotation matrix!

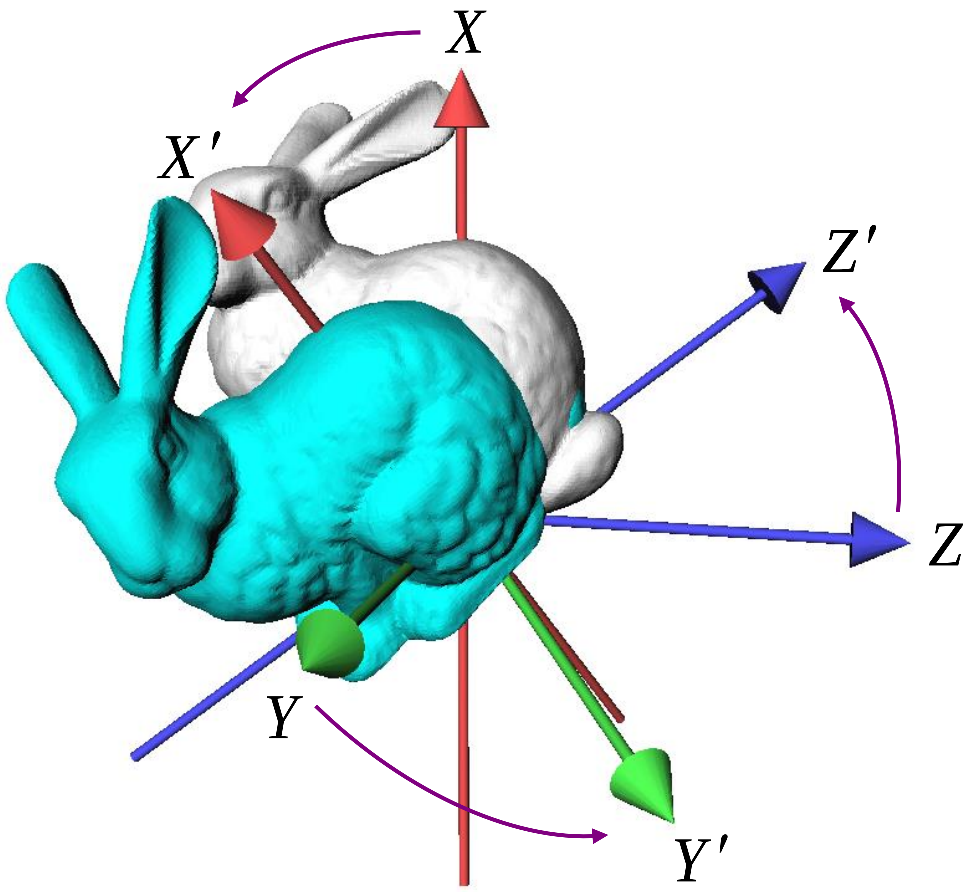
# 2D Rotation and Orientation

- ***2D Rotation***
  - The consequence of any ***2D rotational*** movement can be uniquely represented by a turning angle

- ***2D Orientation***
  - The non-singular parameterization of ***2D orientations*** requires extra parameters
    - E.g.) 2x2 rotation matrices

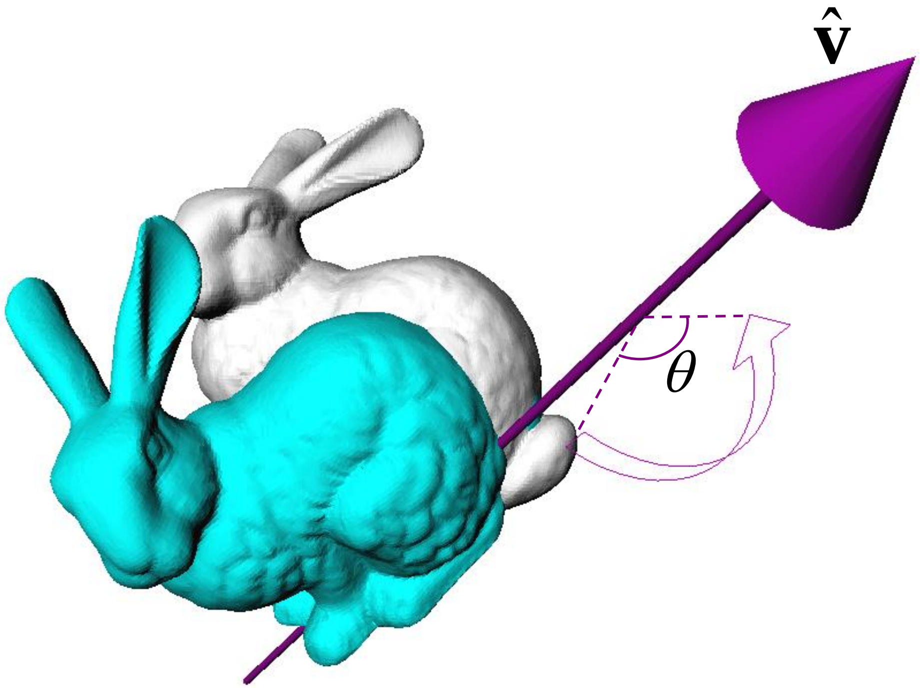
# 3D Rotation

- 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

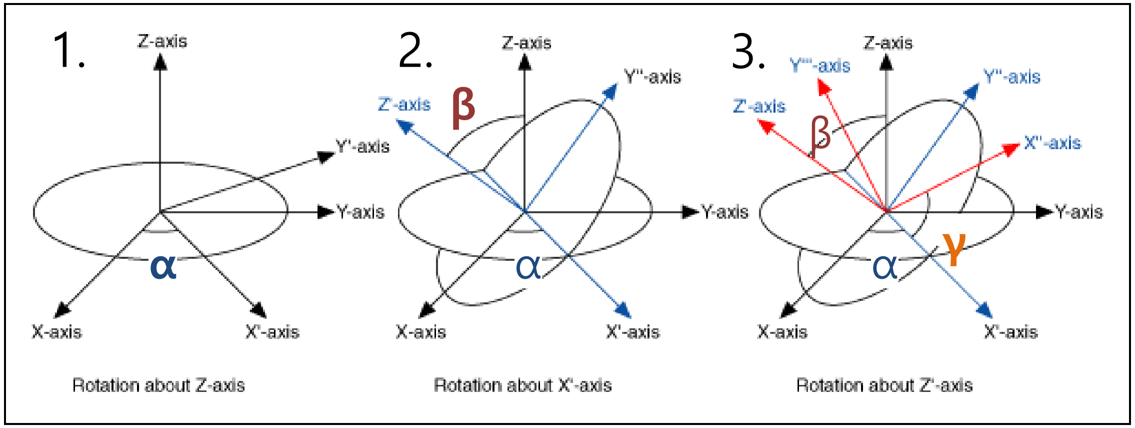
- Describing 3D rotation & orientation is more complicated than 2D.

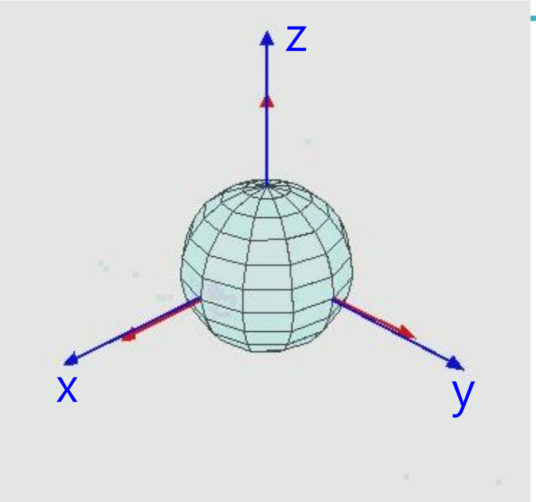
- Many ways to do it
  - Euler angles
  - Axis-angle (Rotation vector)
  - Rotation matrices
  - Unit quaternions

# Euler Angles

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

# Example: ZXZ Euler Angles



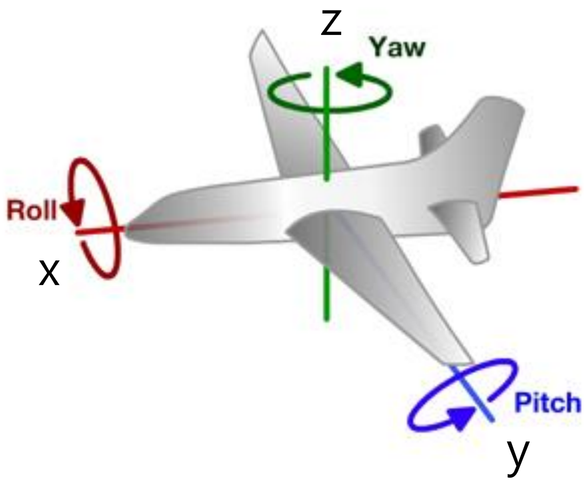

https://commons.wikimedia.org/wiki/File:Euler2a.gif

- 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 $\gamma$

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

$$R= \quad R_z(\alpha) \quad R_x(\beta) \quad R_z(\gamma)$$

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




- Common for describing the orientation of aircrafts

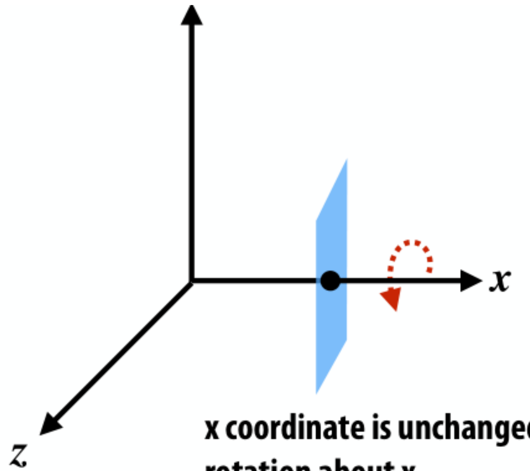
- 1. Rotate about Z-axis by yaw angle
- 2. Rotate about Y-axis of the new frame by pitch angle
- 3. Rotate about X-axis of the new frame by roll angle

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

# Recall: Rotation Matrix in 3D

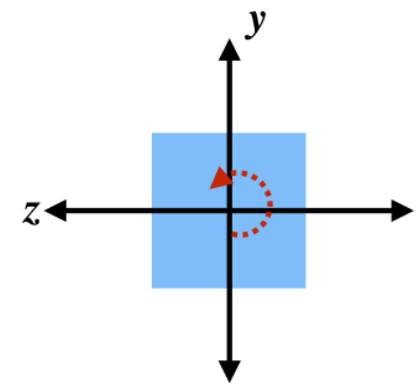
**Rotation about x axis:**

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




x coordinate is unchanged by rotation about x

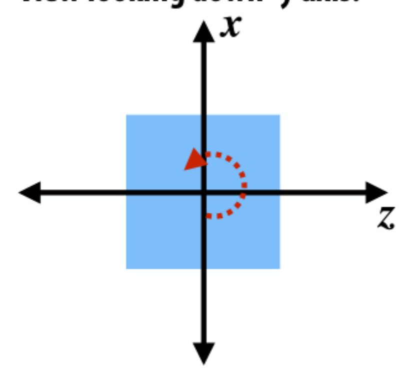
**View looking down -x axis:**




**Rotation about y axis:**

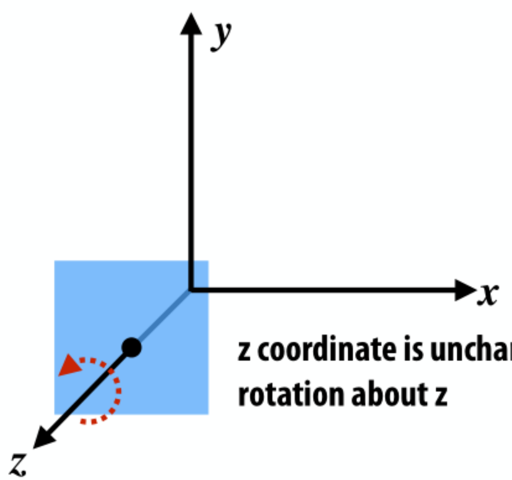
$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ \boxed{-}\sin\theta & 0 & \cos\theta \end{bmatrix}$$

**View looking down -y axis:**




**Rotation about z axis:**

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

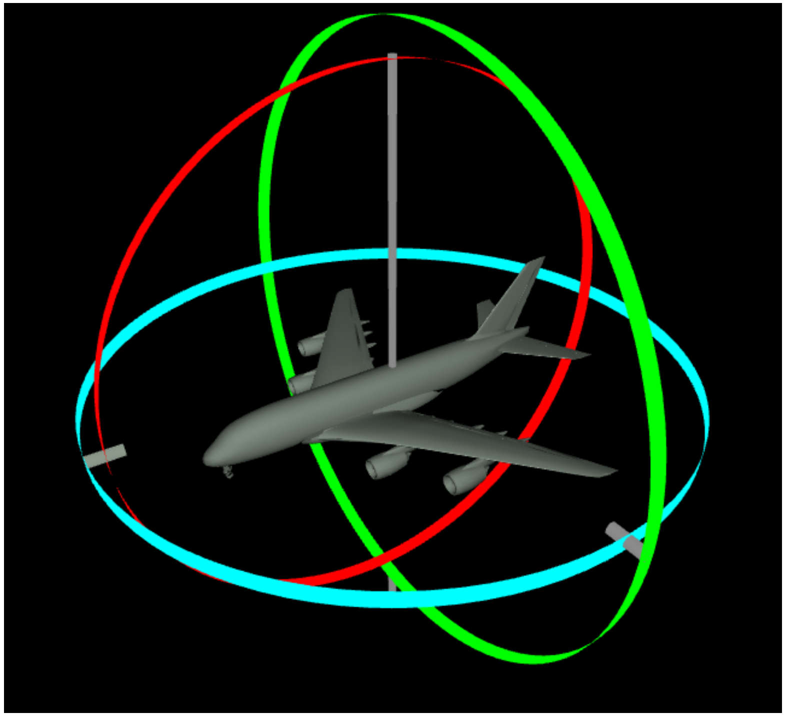



z coordinate is unchanged by rotation about z

# Euler Angles

- Possible 12 combinations
  - XYZ, XYX, XZY, XZX
  - YZX, YZY, YXZ, YXY
  - ZXY, ZXZ, ZYX, ZYZ

# [Practice] Euler Angles Online Demo



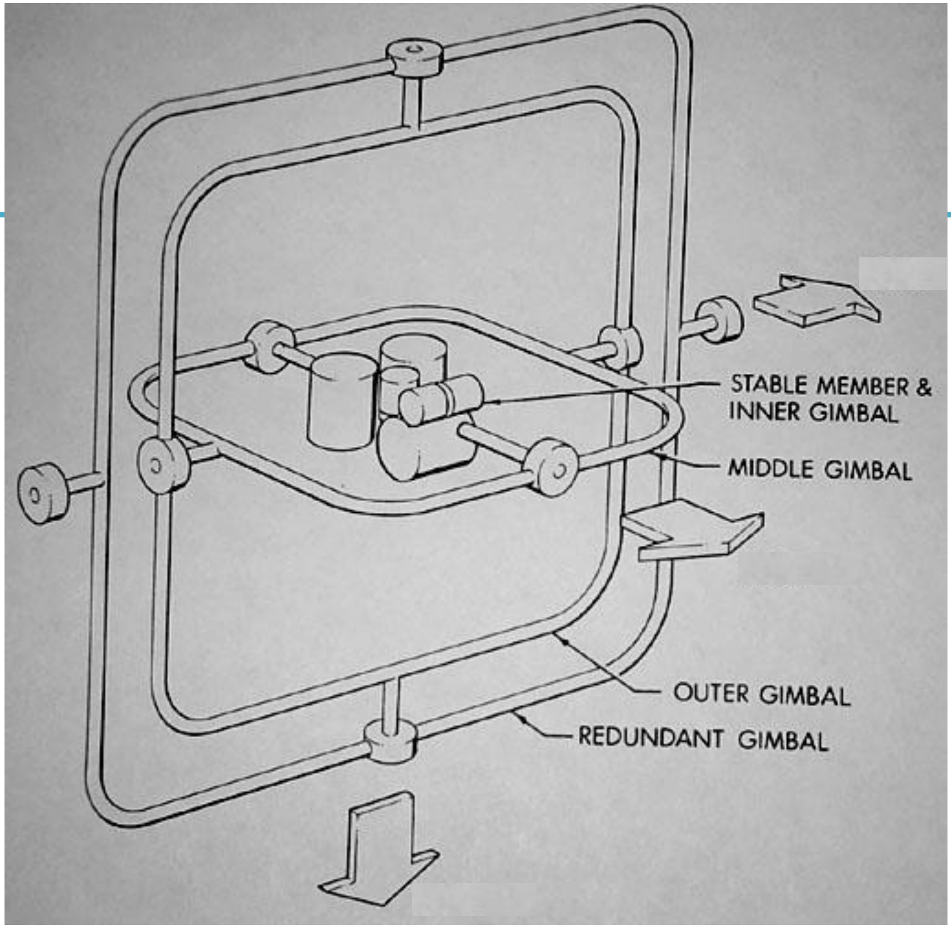
http://www.ctralie.com/Teaching/COMPSCI290/Materials/EulerAnglesViz/

- Try to change yaw, pitch, roll angles

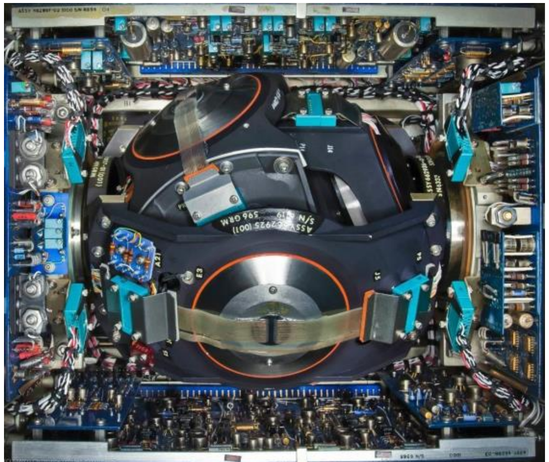
# Gimbal

- Hardware implementation of Euler angles

- Used in
  – Camera systems: to stabilize the camera movement
  – Inertial navigation systems (INS): to get the current orientation of aircrafts or ships

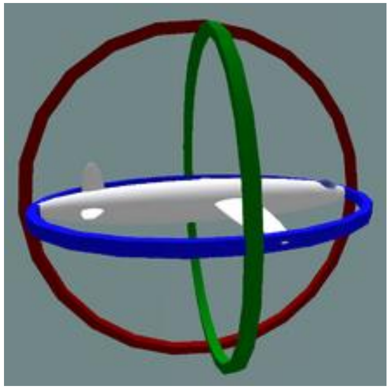




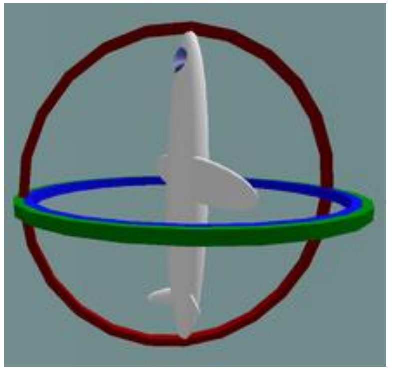


# Gimbal Lock

- 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**



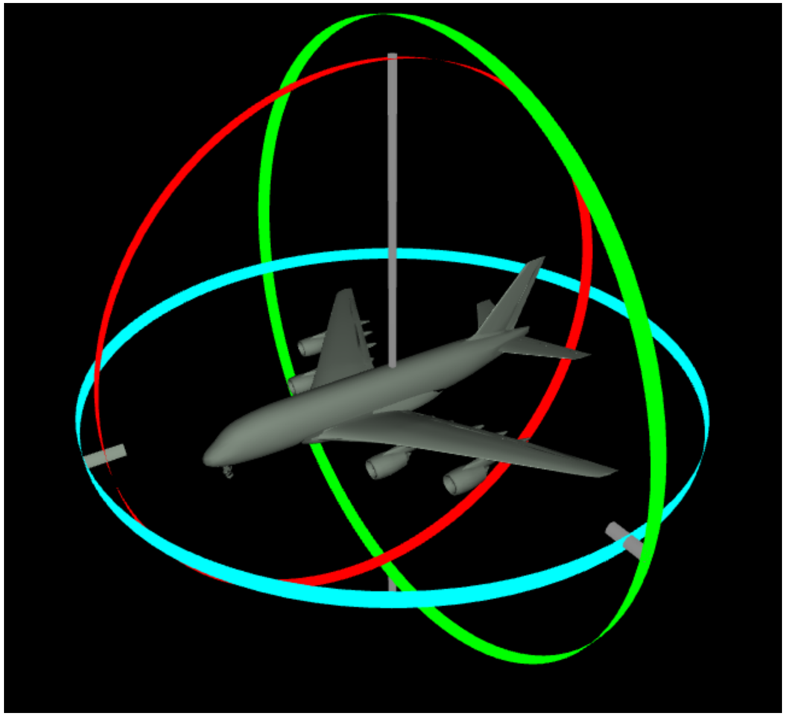
Normal situation.
The plane can rotate in any directions



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/COMPSCI290/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 previous lecture (8-Lighting&Shading).

- Just replace render() function

```python
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)
```

```python
# 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)
```

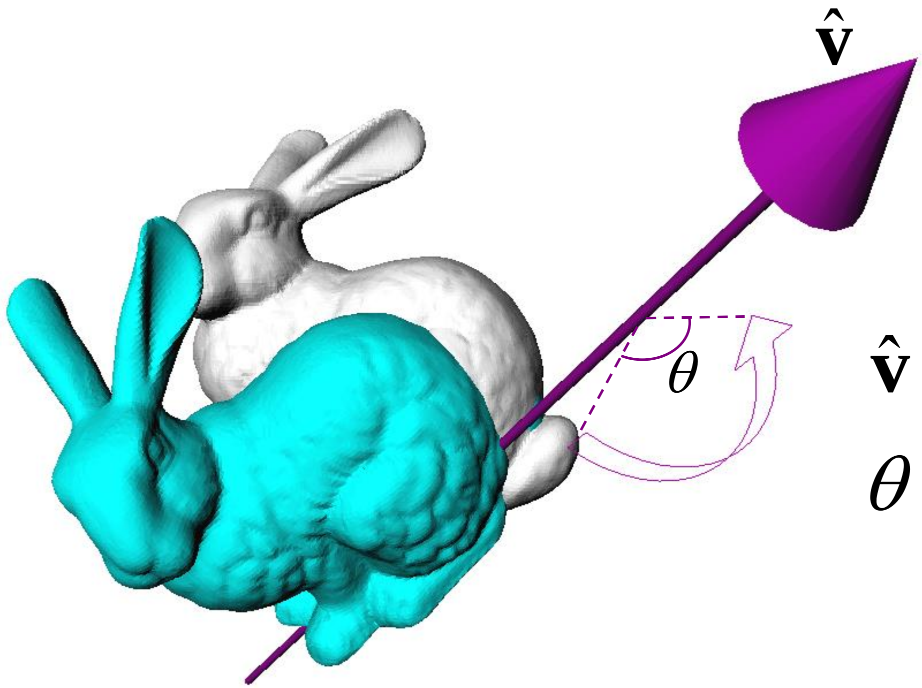
# Quiz #2

- Go to https://www.slido.com/
- Join #cg-hyu
- 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”.

# Axis-Angle (Rotation Vector)




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

$\theta$ : scalar angle

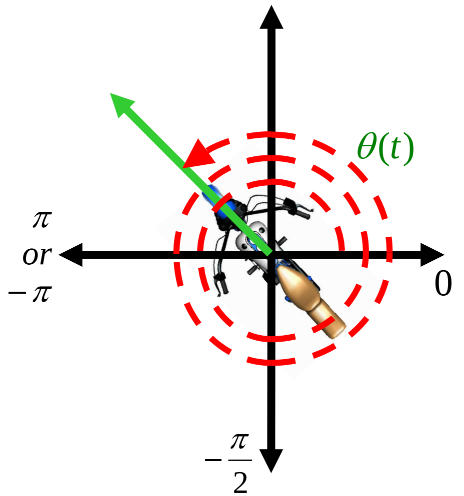
- Rotation vector (3 parameters) $\mathbf{v} = \theta\,\hat{\mathbf{v}} = (x, y, z)$
- Axis-Angle (1+2 parameters) $(\theta, \hat{\mathbf{v}})$

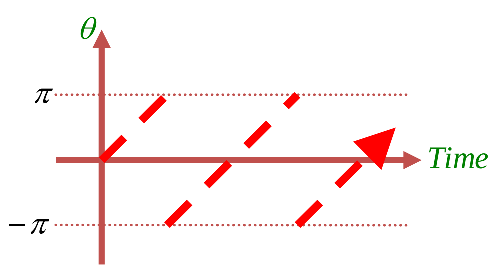
# 3D Orientation

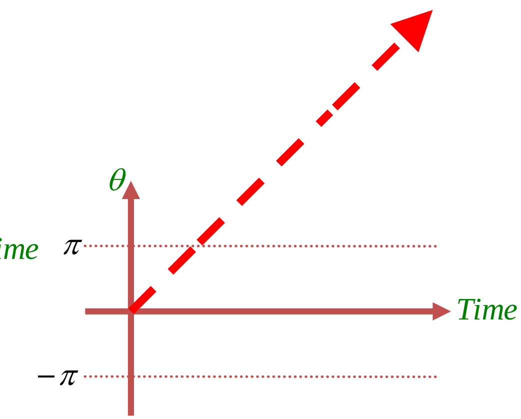
- Euler angles and axis-angle use 3 parameters.

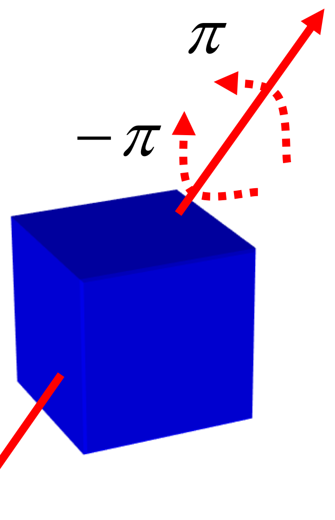
- Expressing 3D orientation using 3 parameters has problems:
- Euler angles
  - Discontinuity (or many-to-one correspondences)
  - **Gimbal lock**

- Axis-angle (Rotation vector)
  - Discontinuity (or many-to-one correspondences)



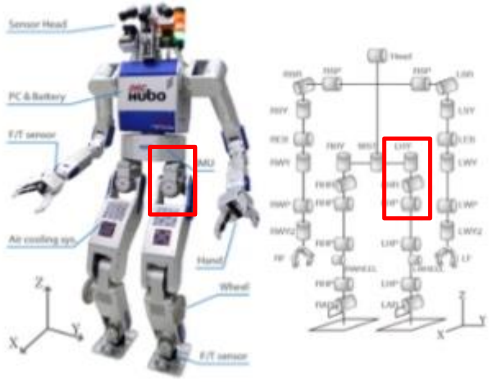



# 3D Orientation

- 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.
  - No need to "normalize" the numbers.



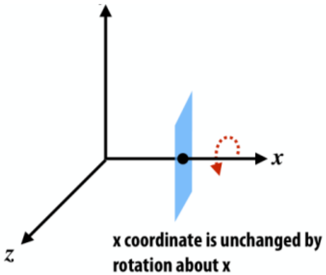
# Rotation Matrices

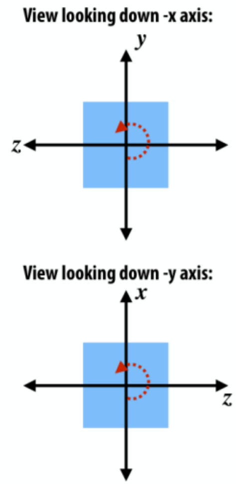
- Rotation in 3D space can be represented as 3x3 matrix:

**Rotation matrix about x, y, z axis**

**Rotation about x axis:**

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



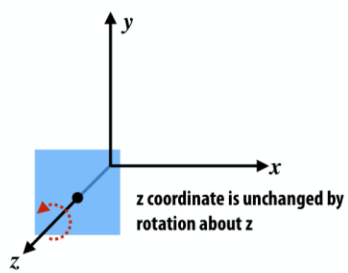




**Rotation about y axis:**

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

**Rotation about z axis:**

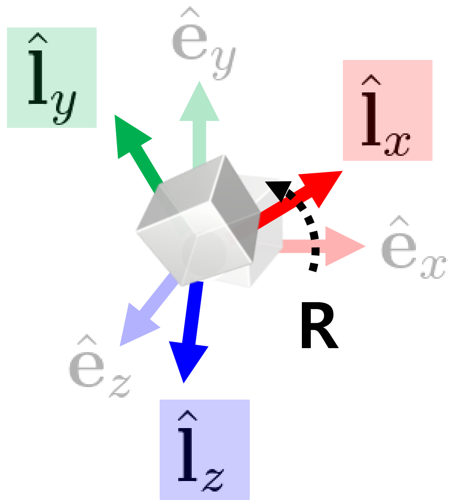
$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$




**Rotation matrix from ZXZ Euler angles**

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

# Meaning of Rotation Matrix




$$\mathbf{R} = \begin{array}{ccc} \left[\begin{matrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{matrix}\right] \\ \begin{matrix} \uparrow & \uparrow & \uparrow \\ \hat{\mathbf{i}}_x & \hat{\mathbf{i}}_y & \hat{\mathbf{i}}_z \end{matrix} \end{array}$$

- 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

$$\boxed{\textbf{1.}\ \ \mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}}$$

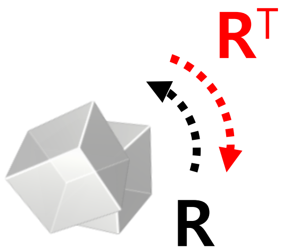
$$\boxed{\textbf{2.}\ \ \det(\mathbf{R}) = 1}$$

- 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

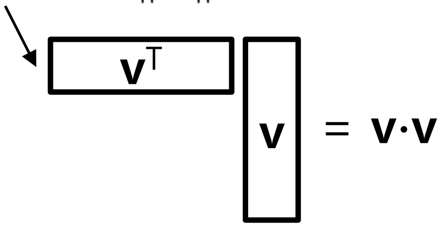
- $\mathbf{R}^{\mathrm{T}}$ is an inverse rotation of $\mathbf{R}$
  - Because, $\mathbf{R}\mathbf{R}^T = \mathbf{I} \iff \mathbf{R}^{-1} = \mathbf{R}^T$




- $\mathbf{R}_1\mathbf{R}_2$ 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) $\|\mathbf{R}\mathbf{v}\|^2 = (\mathbf{R}\mathbf{v})^T(\mathbf{R}\mathbf{v}) = \mathbf{v}^T\mathbf{R}^T\mathbf{R}\mathbf{v} = \mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2$



# [Practice] Properties of Rotation Matrix

- Start with the previous practice code

- Just replace render() function

```python
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]])
```

```python
    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

- Recall Euler's Rotation Theorem:
  - Arbitrary 3D rotation equals to one rotation around an axis
  - 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 $u=(u_x,u_y,u_z)$ by angle $\theta$:

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

(You do not have to memorize this)

# Quiz #3

- Go to https://www.slido.com/
- Join #cg-hyu
- Click “Polls”

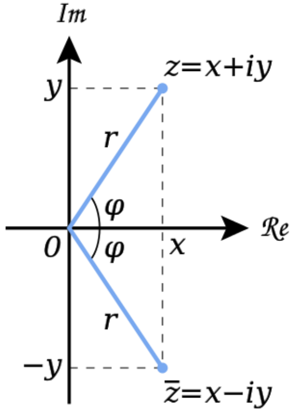
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- 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

$z = x + iy$ **where** $i^2 = -1$




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

$q = w + ix + jy + kz$ **where**

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = k, \quad jk = i, \quad ki = j$$
$$ji = -k, kj = -i, ik = -j$$

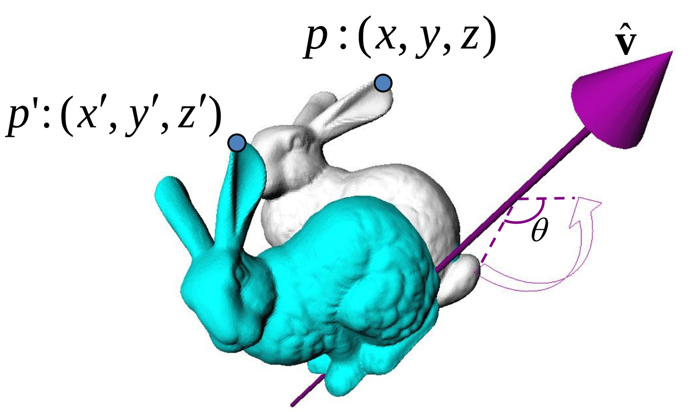
# Unit Quaternions

- Unit quaternions represent 3D rotations

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

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

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




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

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

# Unit Quaternions

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

- Antipodal equivalence
  - q and –q represent the same rotation
  - 2-to-1 mapping: Each individual rotation is represented by **two** quaternions

# Which Representation to Use?

- 3D orientation & rotation representation
  - Euler angles
  - Axis-angle (Rotation vector)
  - Rotation matrices
  - Unit quaternions

- Which one to use?

- General recommendation: **rotation matrices** or **unit quaternions**.
- But you may need other representations depending on the context.
  - Euler angles are useful for hardware implementation of ball joints.

# Which Representation to Use?

- Reason: Euler angles and axis-angle have problems

- Euler angles
  - Discontinuity (or many-to-one correspondences)
  - Gimbal lock

- Axis-angle (Rotation vector)
  - Discontinuity (or many-to-one correspondences)

# 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
  - Handle rotation and translation in a uniform way
    - Eg) 4x4 homogeneous matrices

# Conversion Between Representations

- **Axis-angle → Rotation matrix**
  - Rodrigues' rotation formula, ...
- **Rotation matrix → Axis-angle**
  - Several ways, we'll see one of them in next lecture.
- **Euler angles → Rotation matrix**
  - Building canonical rotation matrices ($\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$) and composing them
- **Rotation matrix → Euler angles**
  - Several ways, but not covered in this class
- **Unit quaternion ↔ Rotation matrix**
  - Several ways, but not covered in this class

# Next Time

- Lab in this week:
  - Lab assignment 9

- Next lecture:
  - 10 - Animation


- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Jehee Lee, SNU, 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