Computer Graphics

9 - Orientation & Rotation

Yoonsang Lee
Hanyang University

Spring 2023

Outline

- Basic Concepts
  - Orientation vs. Rotation
  - Degrees of Freedom
  - Euler's Rotation Theorem

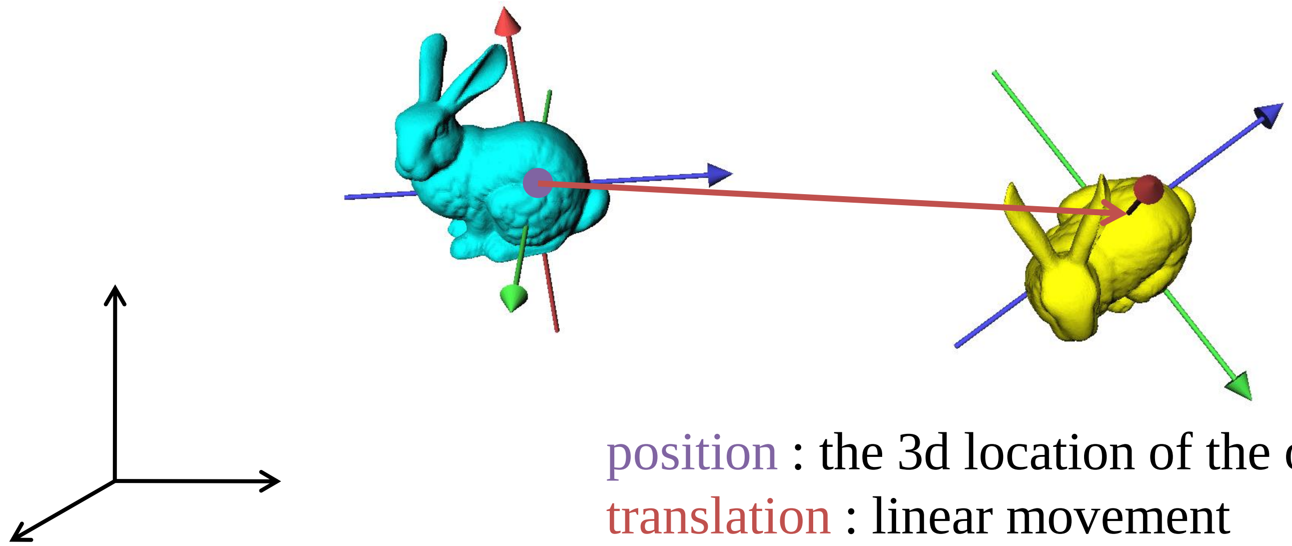
- 3D Orientation & Rotation Representations
  - Euler Angles
  - Rotation Vector (Axis-Angle)
  - Rotation Matrix
  - Unit Quaternion

- Which Representation to Use?
  - Consideration from Several Perspectives
  - Interpolation of 3D Orientation / Rotations
  - Pros & Cons of Each Representation

Basic Concepts

State vs. Movement

- (position : translation)

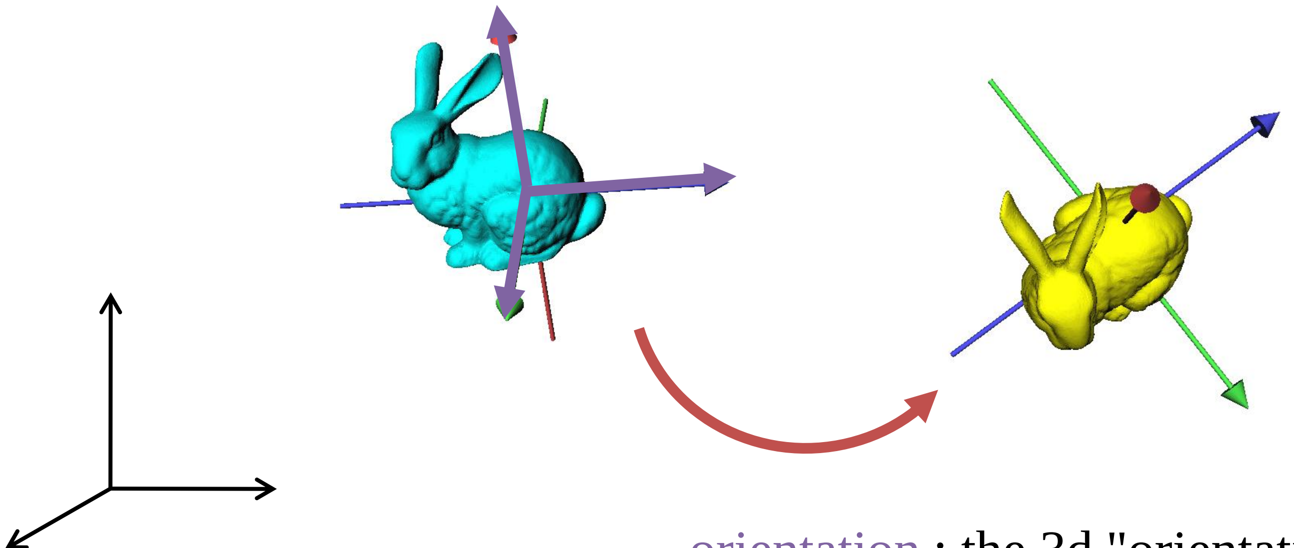


position : the 3d location of the object
translation : linear movement

Some reference frame

State vs. Movement

- (position : translation)
- (orientation : rotation)
- → (state : movement)



Some reference frame

orientation : the 3d "orientation" of the object
rotation : angular movement

Orientation vs. Rotation (and Position vs. Translation)

- **Orientation** & Position - *state*
  - Position: The state of being located.
    - Given a coordinate system, the position of an object can be represented **as a translation from a reference position.**
  - **Orientation**: The state of being oriented. (angular position)
    - Given a coordinate system, the orientation of an object can be represented **as a rotation from a reference orientation.**

- **Rotation** & Translation - *movement*
  - Translation: Linear movement (difference btwn. positions)
  - **Rotation**: Angular movement (difference btwn. orientations)

- This relationship is analogous to *point* vs. *vector*.
  - point: position
  - vector: difference between two points

Similarity in Operations

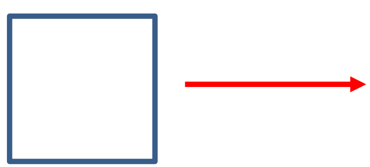
- Point & vector
  - (point) + (point) → (UNDEFINED)
  - (vector) ± (vector) → (vector)
  - (point) ± (vector) → (point)
  - (point) - (point) → (vector)

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

Not vector addition & subtraction

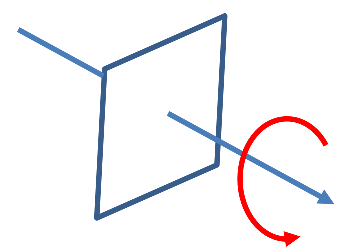
Degrees of Freedom (DOF)

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



Translation along one direction

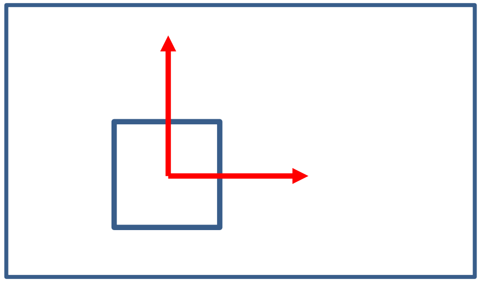
: 1 DOF



Rotation about an axis

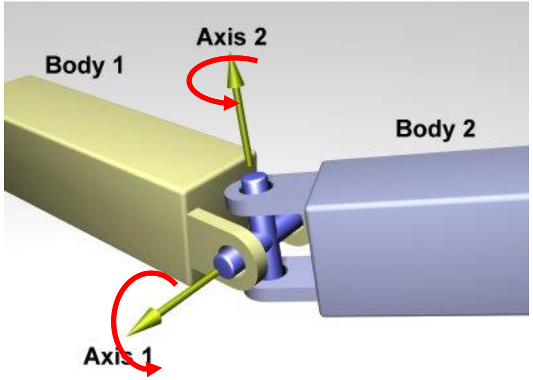
: 1 DOF

Degrees of Freedom (DOF)



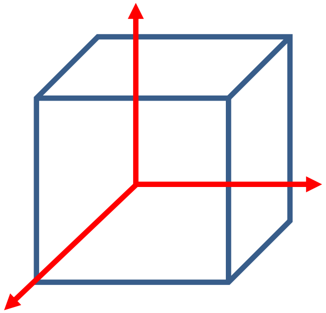
Translation on a plane

: 2 DOFs



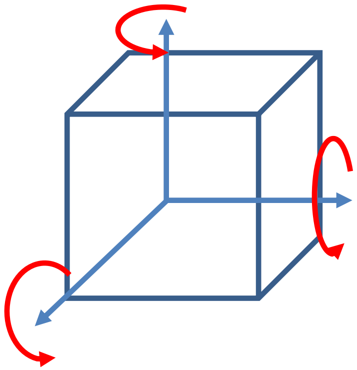
Rotation about two axes

: 2 DOFs



Translation in 3D space

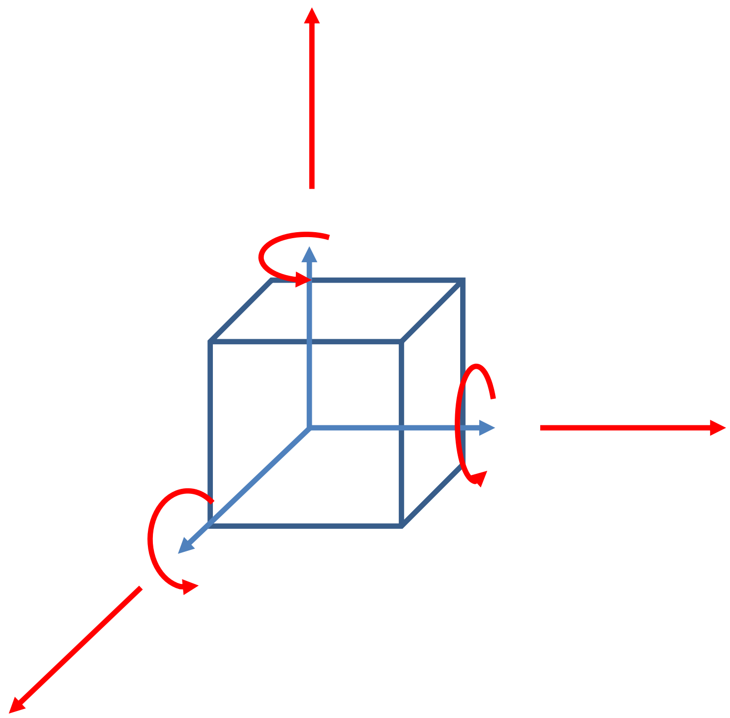
: 3 DOFs



Rotation in 3D space

: 3 DOFs

Degrees of Freedom (DOF)



Any rigid motion in 3D space

: 6 DOFs

Euler's Rotation Theorem

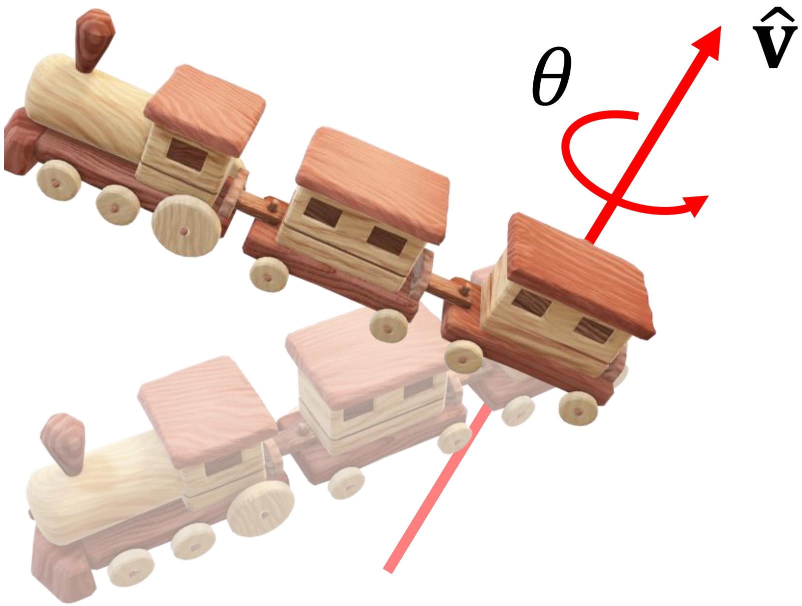
- ***Theorem.*** *When a sphere is moved around its centre it is always possible to find a diameter whose direction in the displaced position is the same as in the initial position.*

  - Leonhard Euler (1707-1783)

- → In 3D space, <u>any displacement of a rigid body such that a point on the rigid body remains fixed</u>, is equivalent to a single <u>rotation about some axis</u> that runs through the fixed point.

Euler's Rotation Theorem

- → For any 3D rotation (any movement with one point fixed), we can always find a fixed **axis** of rotation and an **angle** about the axis.



3D Orientation & Rotation Representations

Describing 3D Rotation & Orientation

- Describing 3D rotation & orientation is not as intuitive as the 2D case.

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

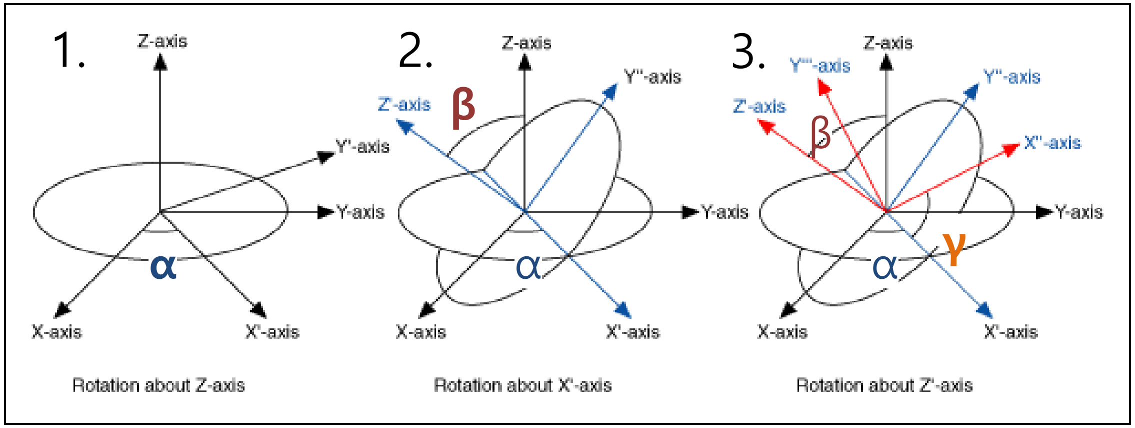
Euler Angles

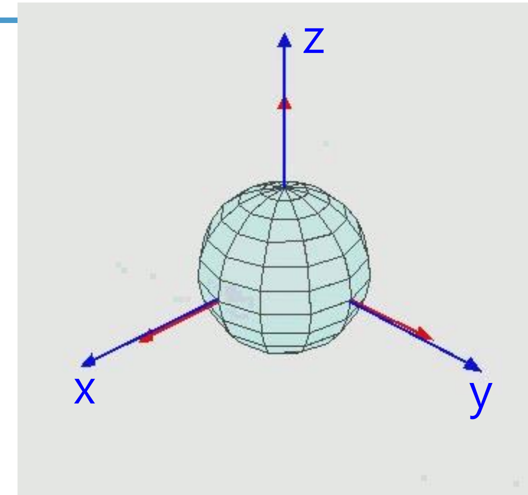
- Express any arbitrary 3D rotation using **three rotation angles about three principle axes.**

- 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.)

Example: ZXZ Euler Angles





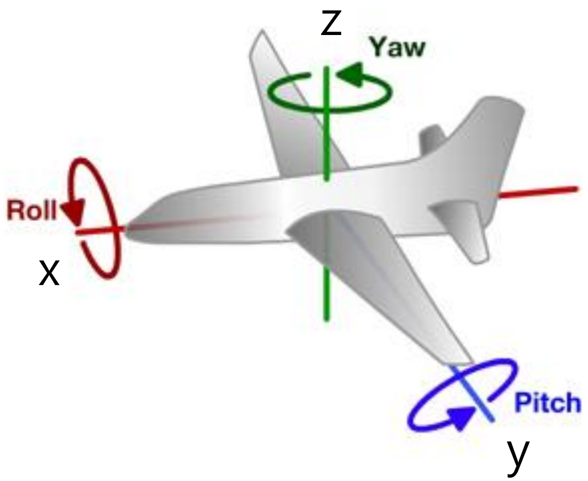
https://commons.wikimedia.org/wiki/File:Euler2a.gif

- 1. Rotate about Z-axis by α
- 2. Rotate about X-axis of the new frame by β
- 3. Rotate about Z-axis of the new frame by γ

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

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

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



- Common for describing the orientation of aircrafts

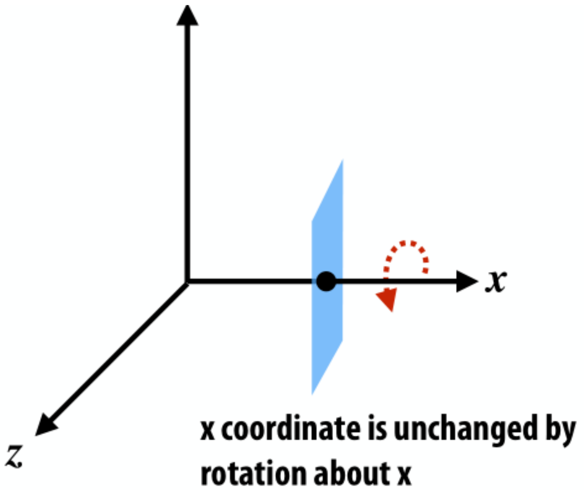
- 1. Rotate about Z-axis by yaw angle
- 2. Rotate about Y-axis of the new frame by pitch angle
- 3. Rotate about X-axis of the new frame by roll angle

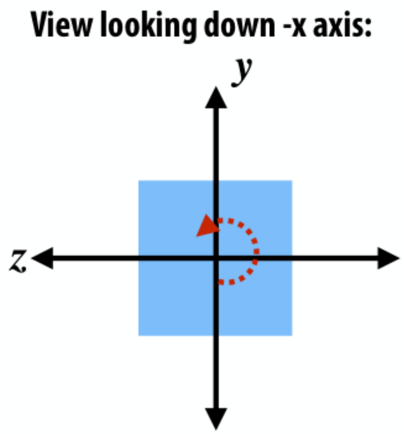
$$R = R_z(\text{yaw})\, R_y(\text{pitch})\, R_x(\text{roll})$$

Recall: 3D Rotation Matrix about Principle Axes

Rotation about x axis:

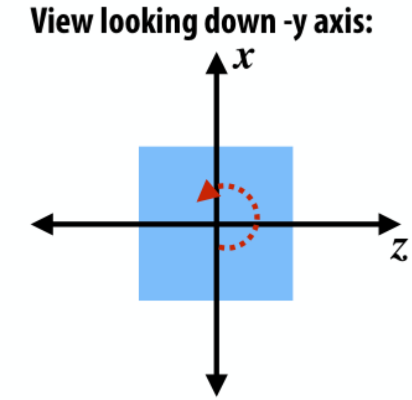
$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$





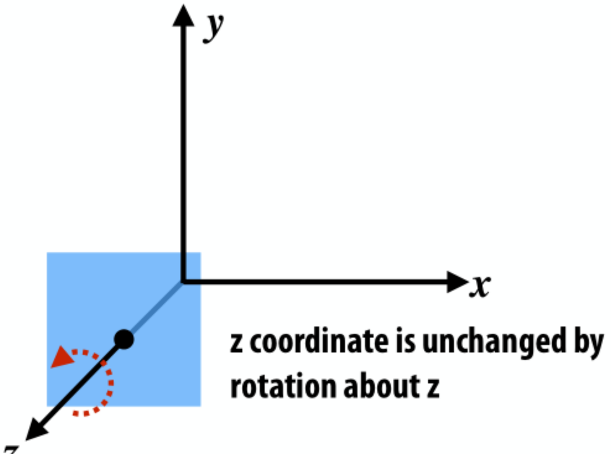
Rotation about y axis:

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



Rotation about z axis:

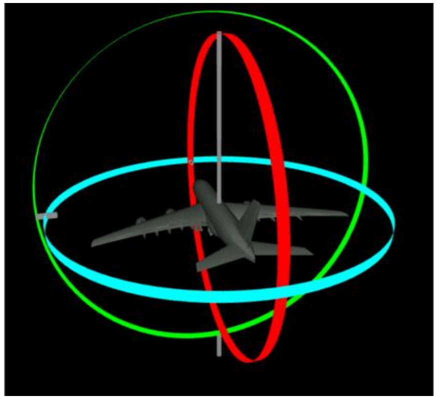
$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



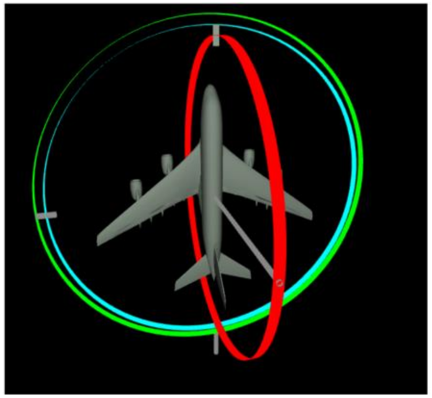
* This slide is from the slides of Prof. Kayvon Fatahalian and Prof. Keenan Crane (CMU): http://15462.courses.cs.cmu.edu/fall2015/

Gimbal Lock

- Euler angles temporarily lose a DOF when the two axes are aligned.

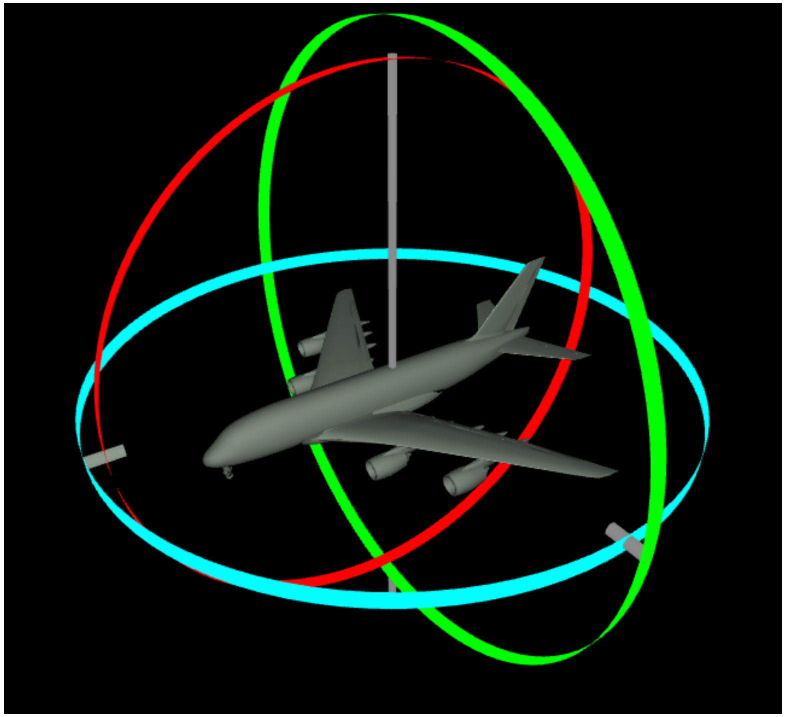


Normal configuration:
The object can rotate freely.



Gimbal lock configuration:
The object can not rotate in one direction.

[Demo] Euler Angles



https://compsci290-s2016.github.io/CoursePage/Materials/EulerAnglesViz/index.html

- Try changing yaw, pitch, roll angles.
- Try making gimbal lock by aligning two of three rotation axes.
  - ex) setting pitch to 90 degreess

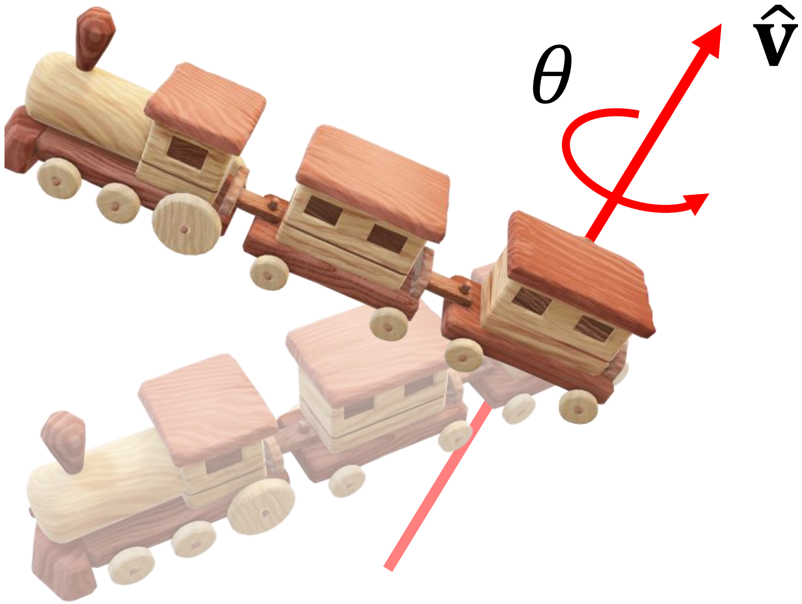
Quiz 1

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click "Polls"

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2021123456: 4.0**

- Note that your quiz answer must be submitted **in the above format** to receive a quiz score!

Rotation Vector (Axis-Angle)



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

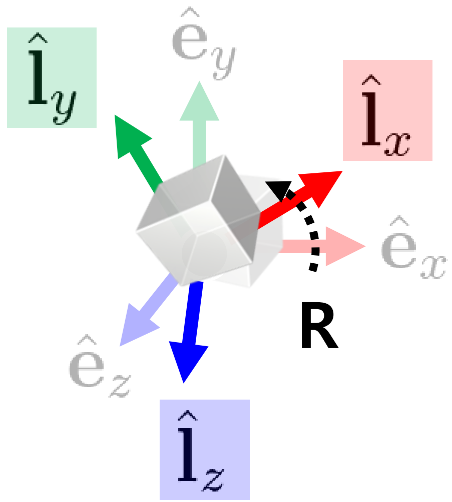
θ: scalar angle

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

Rotation Vector (Axis-Angle)

- Rotation vector is also known as *exponential coordinates* for rotation.
  - If you are curious about why it was called that way, please refer:
  - Section 3.2.3 of "Modern Robotics": http://hades.mech.northwestern.edu/images/2/25/MR-v2.pdf

Rotation Matrix



$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\hat{\mathbf{i}}_x \quad \hat{\mathbf{i}}_y \quad \hat{\mathbf{i}}_z$$

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

Rotation Matrix

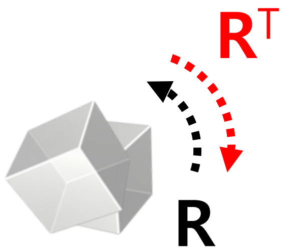
- A square matrix **R** is a rotation matrix if and only if

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

- 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 in 3D, **SO(3)***

Geometric Properties of Rotation Matrix

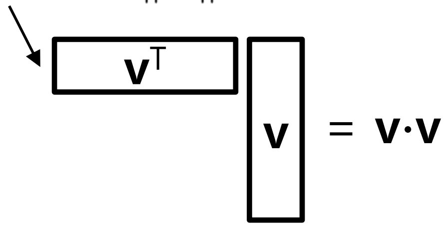
- $\mathbf{R}^T$ is an inverse rotation of $\mathbf{R}$.
  - Because, $\mathbf{R}\mathbf{R}^T = \mathbf{I} \iff \mathbf{R}^{-1} = \mathbf{R}^T$



- $\mathbf{R}_1\mathbf{R}_2$ is a rotation matrix as well (composite rotation).
  - proof) $(\mathbf{R}_1\mathbf{R}_2)^T(\mathbf{R}_1\mathbf{R}_2) = \mathbf{R}_2^T\mathbf{R}_1^T\mathbf{R}_1\mathbf{R}_2 = \mathbf{R}_2^T\mathbf{R}_2 = \mathbf{I}$

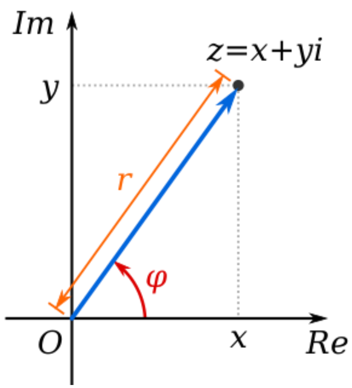
    and $\det(\mathbf{R}_1\mathbf{R}_2) = \det(\mathbf{R}_1)\cdot\det(\mathbf{R}_2) = 1$

- The length of vector $\mathbf{v}$ is not changed after applying a rotation matrix $\mathbf{R}$.
  - proof) $\|\mathbf{R}\mathbf{v}\|^2 = (\mathbf{R}\mathbf{v})^T(\mathbf{R}\mathbf{v}) = \mathbf{v}^T\mathbf{R}^T\mathbf{R}\mathbf{v} = \mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2$



Quaternions

- Complex numbers can be used to represent 2D rotations.
- $z = x + yi$, where $i^2 = -1$
  - $z = x + yi = r\cos\varphi + i\,r\sin\varphi$



- Basic idea: Quaternion is its extension to 3D space.
- $q = w + xi + yj + zk$
- , where $i^2 = j^2 = k^2 = ijk = -1$

$$ij = k, \quad jk = i, \quad ki = j$$
$$ji = -k, \; kj = -i, \; ik = -j$$

Quaternions

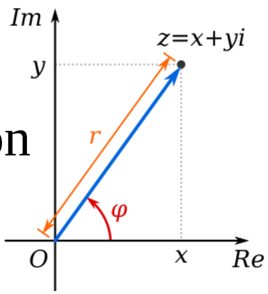
- $q = w + xi + yj + zk$
  - w is called a *real part* (or *scalar part*).
  - $xi + yj + zk$ is called an *imaginary part* (or *vector part*).

- Notation:

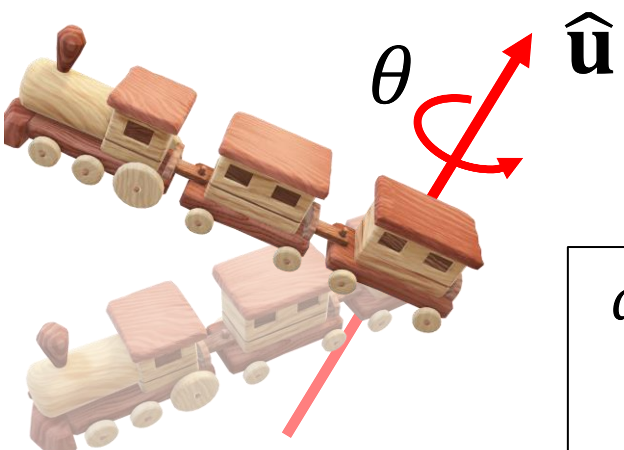
$$
\begin{aligned}
q &= w + xi + yj + zk \\
&= (w, x, y, z) \\
&= (w, \mathbf{v})
\end{aligned}
$$

Unit Quaternions

- Unit quaternions represent 3D rotations:
  - $q = w + ix + jy + kz,$
  - , where $w^2 + x^2 + y^2 + z^2 = 1$
  - Just like $z = x + yi$, where $x^2 + y^2 = 1$ represents 2D rotation ($z = \cos\varphi + i\ \sin\varphi$).



- Rotation about axis $\hat{\mathbf{u}}$ by angle θ:



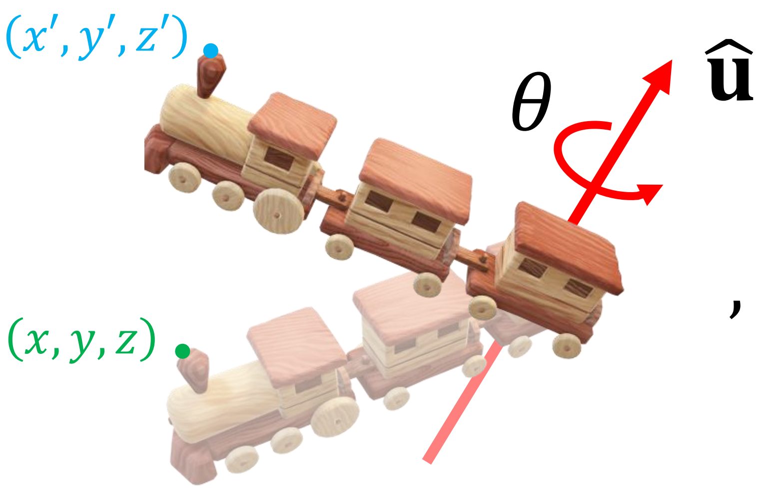
$$\mathbf{q} = \left(\cos\frac{\theta}{2}, \hat{\mathbf{u}}\sin\frac{\theta}{2}\right)$$

$$\begin{aligned} q &= w + xi + yj + zk \\ &= (w, x, y, z) \\ &= (w, \mathbf{v}) \end{aligned}$$

Unit Quaternions

- A 3D position (x, y, z) is represented as a *pure imaginary quaternion* $(0, x, y, z)$.

- If $\mathbf{p} = (0, x, y, z)$ is rotated about axis $\hat{\mathbf{u}}$ by angle θ, then the rotated position $\mathbf{p}' = (0, x', y', z')$ is:



$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$$

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

Unit Quaternions

Identity	$\mathbf{q} = (1,0,0,0)$
Multiplication	$\mathbf{q}_1\mathbf{q}_2 = (w_1, \mathbf{v}_1)(w_2, \mathbf{v}_2)$ $= (w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$
Inverse (Conjugate)	$\mathbf{q}^{-1} = (w, -x, -y, -z)$

- $\mathbf{q}_1\mathbf{q}_2$: rotate by $\mathbf{q}_1$ then $\mathbf{q}_2$ w.r.t. body frame

  or rotate by $\mathbf{q}_2$ then $\mathbf{q}_1$ w.r.t. world frame

- $\mathbf{p}' = \mathbf{q}_1\mathbf{q}_2\mathbf{p}(\mathbf{q}_1\mathbf{q}_2)^{-1} = \mathbf{q}_1(\mathbf{q}_2\mathbf{p}\mathbf{q}_2{}^{-1})\mathbf{q}_1{}^{-1}$

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. 2021123456: 4.0**

- Note that your quiz answer must be submitted **in the above format** to receive a quiz score!

Which Representation to Use?

Which Representation to Use?

- Let's consider each representation from the following four perspectives:

- 1) "Addition" of rotations
- 2) "Subtraction" of rotations
- 3) Interpolation of rotations
- 4) Continuity / correspondence in each representation

1) "Addition" of Rotations

- ✓ Rotation matrix, Unit quaternion:
  - $\mathbf{R}_1\mathbf{R}_2$, $\mathbf{q}_1\mathbf{q}_2$
  - Rotate by $\mathbf{R}_1$(or $\mathbf{q}_1$), then by $\mathbf{R}_2$(or $\mathbf{q}_2$) w.r.t. (current) body frame.
  - (Element-wise addition does NOT even produce a rotation matrix or unit quaternion.)

- ✗ Euler angles:
  - $(\alpha_1, \beta_1, \gamma_1) + (\alpha_2, \beta_2, \gamma_2) = (\alpha_1 + \alpha_2, \beta_1 + \beta_2, \gamma_1 + \gamma_2)$?
  - Does **NOT** mean rotate by $(\alpha_1, \beta_1, \gamma_1)$, then by $(\alpha_2, \beta_2, \gamma_2)$.

- ✗ Rotation vector:
  - $\mathbf{v}_1 + \mathbf{v}_2$?
  - Does **NOT** mean rotate by $\mathbf{v}_1$, then by $\mathbf{v}_2$.

2) "Subtraction" of Rotations

- ✓ Rotation matrix, Unit quaternion:
  - $\mathbf{R}_1{}^T\mathbf{R}_2$, $\mathbf{q}_1{}^{-1}\mathbf{q}_2$
  - Rotational difference: A rotation matrix that rotate a frame $\mathbf{R}_1$(or $\mathbf{q}_1$) to be coincident with the frame $\mathbf{R}_2$(or $\mathbf{q}_2$) when applied w.r.t. the frame $\mathbf{R}_1$(or $\mathbf{q}_1$)
  - Because $\mathbf{R}_1(\mathbf{R}_1{}^T\mathbf{R}_2) = \mathbf{R}_2$
  - (Element-wise subtraction does NOT even produce a rotation matrix or unit quaternion.)

- ✗ Euler angles:
  - $(\alpha_2, \beta_2, \gamma_2) - (\alpha_1, \beta_1, \gamma_1) = (\alpha_2 - \alpha_1, \beta_2 - \beta_1, \gamma_2 - \gamma_1)$?
  - Does **NOT** mean difference between rotation $(\alpha_1, \beta_1, \gamma_1)$ and $(\alpha_2, \beta_2, \gamma_2)$.

- ✗ Rotation vector:
  - $\mathbf{v}_2 - \mathbf{v}_1$?
  - Does **NOT** mean difference between rotation $\mathbf{v}_1$ and $\mathbf{v}_2$.

3) Interpolation of Rotations

- Can we just linear interpolate each element of
  - Euler angles
  - Rotation vector
  - Rotation matrix
  - Unit quaternion
- ?

- → No!

Interpolating Each Element of Rotation Matrices / Unit Quaternions?

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

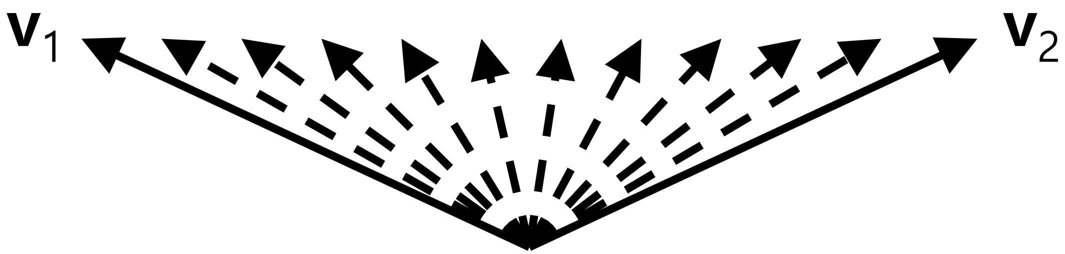
$$\text{lerp}(\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}, \begin{bmatrix}1 & 0 & 0\\0 & 0 & -1\\0 & 1 & 0\end{bmatrix}, 0.5) = \begin{bmatrix}1 & 0 & 0\\0 & 0.5 & -0.5\\0 & 0.5 & 0.5\end{bmatrix}$$

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 Vectors?

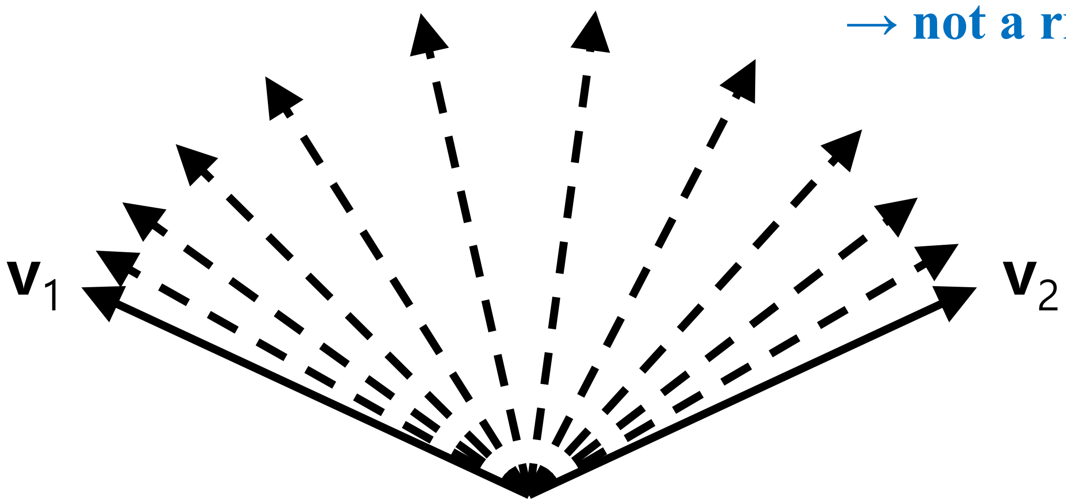
- Let's say we have two rotation vectors $\mathbf{v}_1$ & $\mathbf{v}_2$ of the same length
- Linear interpolation of $\mathbf{v}_1$ & $\mathbf{v}_2$ produces even spacing



Interpolating Rotation Vectors?

- 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!

→ **not a right method!**

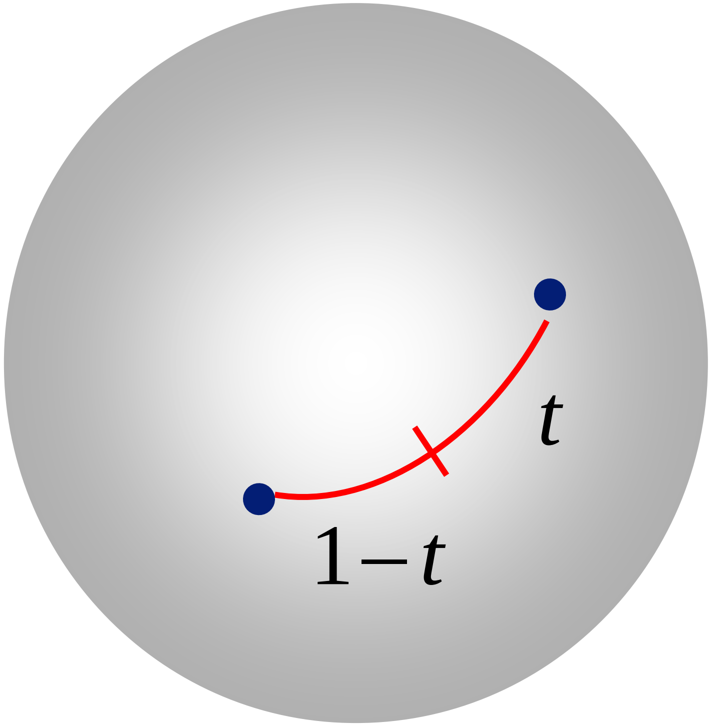


Interpolating Euler Angles?

- Interpolating two tuples of Euler angles does not provide 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]
  - **S**pherical **l**inear int**erp**olation
  - Linear interpolation of two orientations



"t" refers power, not transpose

$$\mathrm{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) = \mathbf{R}_1(\mathbf{R}_1^T\mathbf{R}_2)^t$$
$$= \mathbf{R}_1 \exp(\mathrm{t} \cdot \log(\mathbf{R}_1^T\mathbf{R}_2))$$

Slerp with Rotation Matrices

- $\text{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) = \mathbf{R}_1(\mathbf{R}_1^T\mathbf{R}_2)^t$
- 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 $\mathbf{R}_a$ ($\mathbf{R}_a(+)\mathbf{R}_b$)

- $\text{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) = \mathbf{R}_1(\mathbf{R}_1^T\mathbf{R}_2)^t$
  $= \mathbf{R}_1 \exp(\mathrm{t} \cdot \log(\mathbf{R}_1^T\mathbf{R}_2))$
  - exp(): **rotation vector** to **rotation matrix**
  - log(): **rotation matrix** to **rotation vector**

Exp & Log

- **Exp** (exponential): **rotation vector** to **rotation matrix**
  - Given normalized rotation axis u=(u_x,u_y,u_z), rotation angle θ

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

(Rodrigues' rotation formula)

- **Log** (logarithm): **rotation matrix** to **rotation vector**
  - Given rotation matrix **R**, compute axis **v** and angle θ:

$$\theta = \cos^{-1}((R_{11} + R_{22} + R_{33} - 1)/2)$$
$$v_1 = (R_{32} - R_{23})/(2\sin\theta)$$
$$v_2 = (R_{13} - R_{31})/(2\sin\theta)$$
$$v_3 = (R_{21} - R_{12})/(2\sin\theta)$$

+ some algorithm to avoid singularity at θ=kπ, where k is an integer.

- **No need to try to memorize these formulas!**
- For detail, please refer Section 3.2.3 of "Modern Robotics": http://hades.mech.northwestern.edu/images/2/25/MR-v2.pdf

Exp & Log

- Practical note:
  - For exp() and log(), you can use the functions provided by libraries such as pyglm, scipy (python), and Eigen (c++).
  - Today's lab uses pyglm for this.

  - You can implement your own exp() and log() if you wish. Implementation is not too difficult.

Slerp with Quaternions

- Quaternion slerp:
  - $\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) = \mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t$

- Geometric slerp (equivalent):
  - $\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) = \frac{\sin((1-t)\varphi)}{\sin\varphi}\mathbf{q}_1 + \frac{\sin(t\varphi)}{\sin\varphi}\mathbf{q}_2$
    - φ: the angle subtended by the arc ($\cos\varphi = \mathbf{q}_1 \cdot \mathbf{q}_2$)

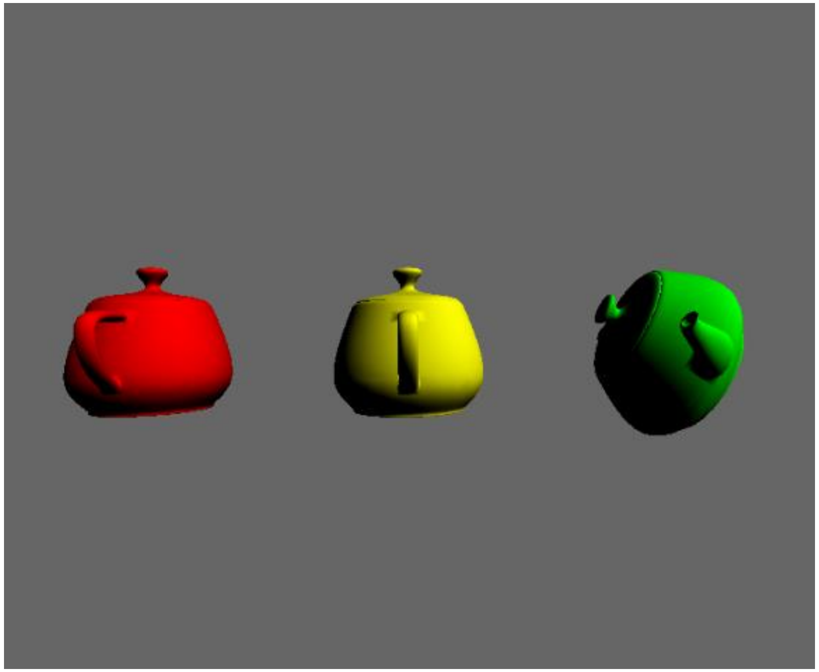
- No slerp for Euler angles or rotation vector representation!
  - They need to be converted to rotation matrices or unit quaternions to be *slerp*ed.

Comparison of Interpolation Methods

- Start orientation (ZYX Euler angles): Rz(-90) Ry(90) Rx(0)
- End orientation (ZYX Euler angles): Rz(0) Ry(0) Rx(90)

https://youtu.be/Y02MAWKmfGU

[Demo] Slerp



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

- Try changing “Start Rotation” & “End Rotation”
- Try moving “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. 2021123456: 4.0**

- Note that your quiz answer must be submitted **in the above format** to receive a quiz score!

3) Interpolation of Rotations: Summary

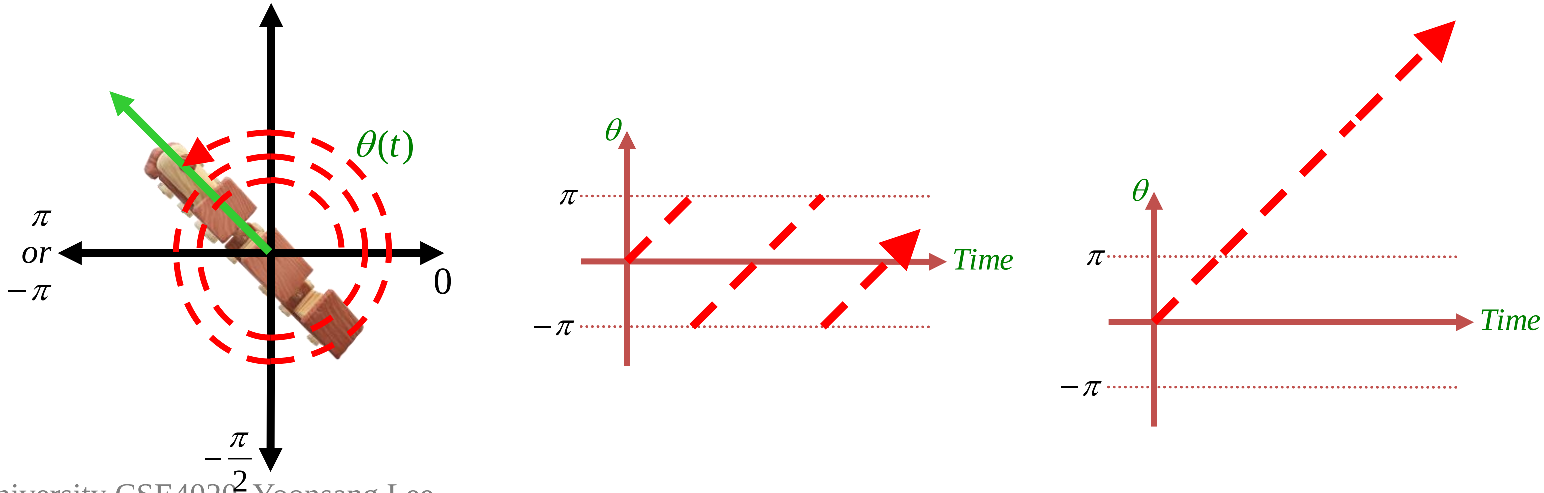
- ✓ Rotation matrix, Unit quaternion:
  - slerp($\mathbf{R}_1, \mathbf{R}_2, t$), slerp($\mathbf{q}_1, \mathbf{q}_2, t$)
  - (Element-wise interpolation does NOT even produce a rotation matrix or unit quaternion.)

- ✗ Euler angles:
  - lerp($(\alpha_1, \beta_1, \gamma_1), (\alpha_2, \beta_2, \gamma_2)$)?
  - Does **NOT** mean linear interpolation between two rotations

- ✗ Rotation vector:
  - lerp($\mathbf{v}_1, \mathbf{v}_2$)?
  - Does **NOT** mean linear interpolation between two rotations

4) Continuity / Correspondence

- ✗ Euler angles, Rotation vector:
  - Use 3 parameters to express 3 DOFs orientations / rotations.
  - Due to this characteristic, they have the following problems:
  - Discontinuity
    - The representation of continuous orientations may have discontinuities.
  - Many-to-one mapping
    - One orientation can be mapped to many representations



4) Continuity / Correspondence

- △ Unit quaternion:
  - Use 4 parameters
  - Continuous representation
  - Two-to-one mapping
    - Any **q** and **–q** always represent the same rotation.
    - This property is called *antipodal equivalence*.

- ✓ Rotation matrix:
  - Use 9 parameters
  - Continuous one-to-one mapping

Again: Which Representation to Use?

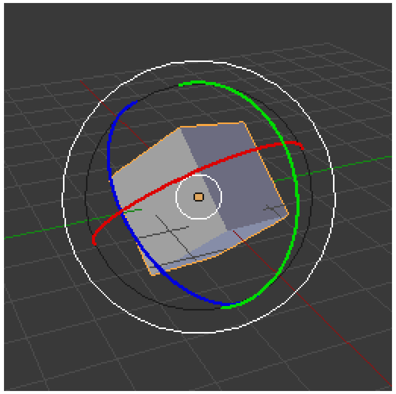
- General advice:
  - Use rotation matrices or unit quaternions.

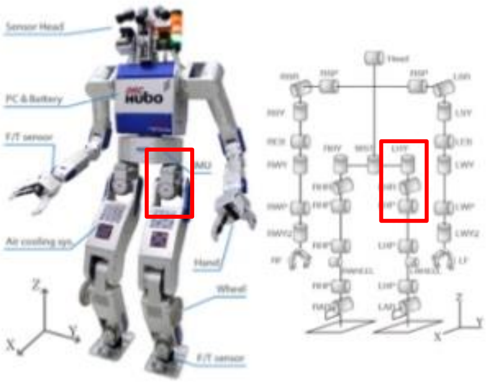
- More advanced advice:
  - Don't stick to just one representation. Each has its pros and cons.
  - To take advantage of a different representation, you can convert your rotational data to another representation and back to the original representation.

Pros & Cons of Euler Angles

- ▼ Not provide accurate "addition", "subtraction", and interpolation operations
- ▼ Discontinuity / Many-to-one correspondence
- ▼ Gimbal lock

- ▲ Intuitive way for manipulating orientations in 3D tools
- ▲ Can be used to represent the state of a hardware implementation of a ball joints using three perpendicular hinge joints





Pros & Cons of Rotation Vector

- ▼ Not provide accurate "addition", "subtraction", and interpolation operations
- ▼ Discontinuity / Many-to-one correspondence

- ▲ Good for visualizing a "rotation"
  - It gives a direct sense of rotation axis and angle.
- ▲ Most compact representation using 3 parameters
  - Euler angles too, but it has gimbal lock issues.

Pros & Cons of Rotation Matrix

- ▲ Provides accurate "addition", "subtraction", and interpolation operations
- ▲ Continuous one-to-one mapping (very nice)
- ▲ Good for visualizing "orientation"
  - You can easily visualize an orientation by drawing the frame with its x, y, z axes using the three columns of the rotation matrix.
- ▲ Can be easily extended to a 4x4 affine transformation matrix to handle rotation and translation in a uniform way

- ▼ Many (9) parameters (storage)
- ▼ Computationally slower and numerically less stable than unit quaternions (but not much)

Pros & Cons of Unit Quaternion

- ▲ Provides accurate "addition", "subtraction", and interpolation operations
- ▲ Continuous representation
- ▲ Only 4 parameters
- ▲ Computationally faster and numerically more stable than rotation matrices

- ▼ Two-to-one mapping (*antipodal equivalence*)
- ▼ Less intuitive number system

Conversion between Representations

- There are well-established theories for conversion between
  - Euler angles
  - Rotation vector
  - Rotation matrix
  - Unit quaternion.

- You can use libraries such as pyglm, scipy (python) or Eigen (c++) for conversion between these representation.
  - pyglm only provides some of these conversions.

- You can implement your own conversion code if you wish, referring to these theories (by gooling, etc).

Lab Session

- Now let's start the lab session.