

Introduction to Data-Driven Animation: Programming with Motion Capture

Jehee Lee

Seoul National University

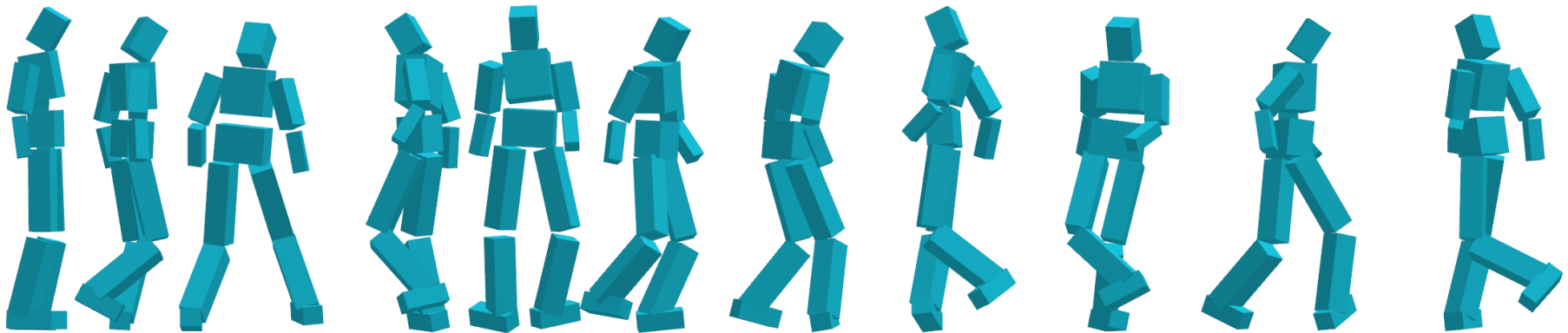


Data-Driven Animation with Motion Capture



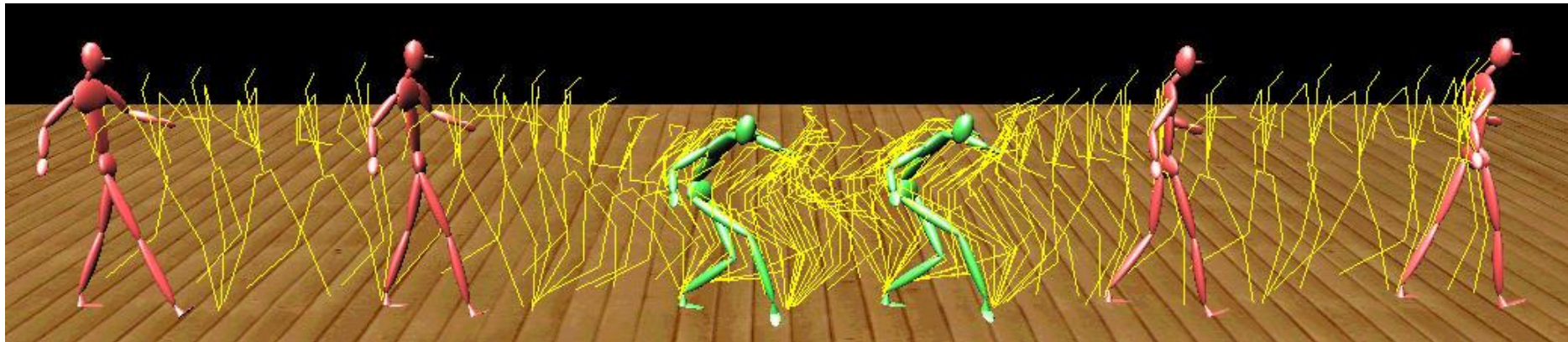
Programming with Motion Capture

- Why is it difficult?
 - Encompass a lot of heterogeneous information
 - Joint angles
 - Position/orientation of a skeletal root
 - Their temporal trajectories
 - A number of local/global coordinate systems



Mathematical Notation

- Can we describe operations in simple equations?
 - Linear interpolation between two motion clips
 - Splice two motion clips sequentially



Course Objectives

- Mathematical framework and notation
 - Geometric reasoning and intuition
- A practical guide to programming with motion capture

```
A. (pose) ⊗ (pose) → (UNDEFINED)
B. exp(displacement) ⊗ exp(displacement) → exp(displacement)
C. (pose) ⊗ exp(displacement) → (pose)
D. (pose) ⊙ (pose) → exp(displacement)
E. log(exp(displacement)) → (displacement)
F. log(pose) → (UNDEFINED)
G. (scalar) · (displacement) → (displacement)
   exp(displacement)(scalar) → exp(displacement)
H. (pose)(scalar) → (pose)      if scalar=1
   → exp(displacement)      if scalar=0
   → (UNDEFINED)           otherwise
I. (displacement) ± (displacement) → (displacement)
J. ∑ (scalar) · (displacement) → (displacement)
K. affine_combination(poses) → (ILL-DEFINED)
```

References

- Jehee Lee, [Representing Rotations and Orientations in Geometric Computing](#), *IEEE Computer Graphics and Applications*, 2008.
- Yoonsang Lee, Sungeun Kim, Jehee Lee, [Data-Driven Biped Control](#), SIGGRAPH 2010.
- Jehee Lee, Jinxiang Chai, Paul Reitsma, Jessica Hodgins, and Nancy Pollard, [Interactive Control of Avatars Animated with Human Motion Data](#), SIGGRAPH 2002.
- Jehee Lee and Sung Yong Shin, [A Coordinate-Invariant Approach to Multiresolution Motion Analysis](#), *Graphical Models*, 2001.
- Hyun Joon Shin, Jehee Lee, Michael Gleicher, and Sung Yong Shin, [Computer Puppetry: An Importance-based Approach](#), *ACM Transactions on Graphics*, 2001.
- Jehee Lee and Sung Yong Shin, [A Hierarchical Approach to Interactive Motion Editing for Human-like Characters](#), SIGGRAPH 99.

Course Overview

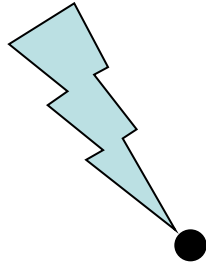
- **Introduction and Overview** (5 min)
- **Coordinate-Invariant Geometric Programming** (20 min)
 - What is coordinate-invariant?
 - Affine geometry
 - Coordinate-invariant operations between points and vectors
- **Programming with Orientations and Rotations** (35 min)
- **Programming with Motion Capture Data** (10 min)
- **Practical examples** (30 min)

Geometric Programming

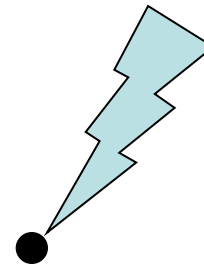
- A way of handling geometric entities such as vectors, points, and transforms.
- Write geometric programs relying on geometric reasoning rather than coordinate manipulation
- Pioneered by Goldman and DeRose
 - Geometric programming : A coordinate-free approach, SIGGRAPH 1988 Course #25 Notes
- Coordinate-Invariant vs Coordinate-Free

Example of coordinate-dependence

Point **p**



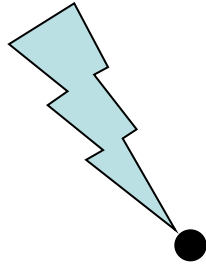
Point **q**



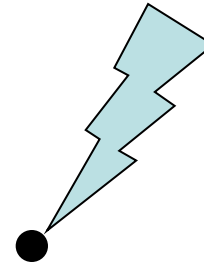
- What is the “sum” of these two positions ?

If you assume coordinates, ...

$$\mathbf{p} = (x_1, y_1)$$



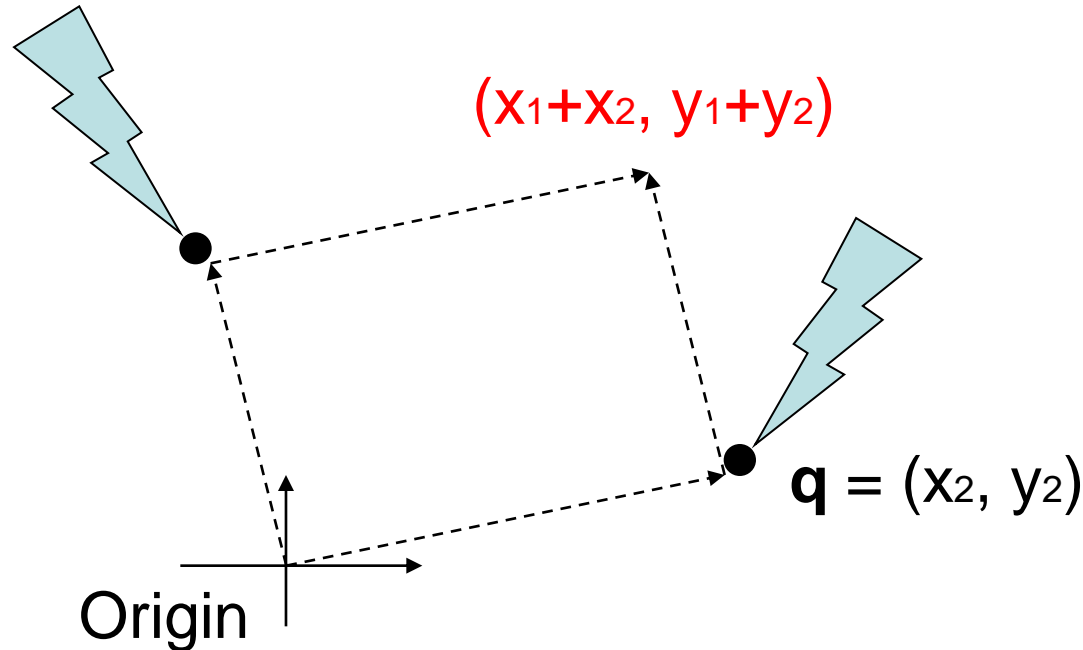
$$\mathbf{q} = (x_2, y_2)$$



- The sum is (x_1+x_2, y_1+y_2)
 - Is it correct ?
 - Is it geometrically meaningful ?

If you assume coordinates, ...

$$\mathbf{p} = (x_1, y_1)$$

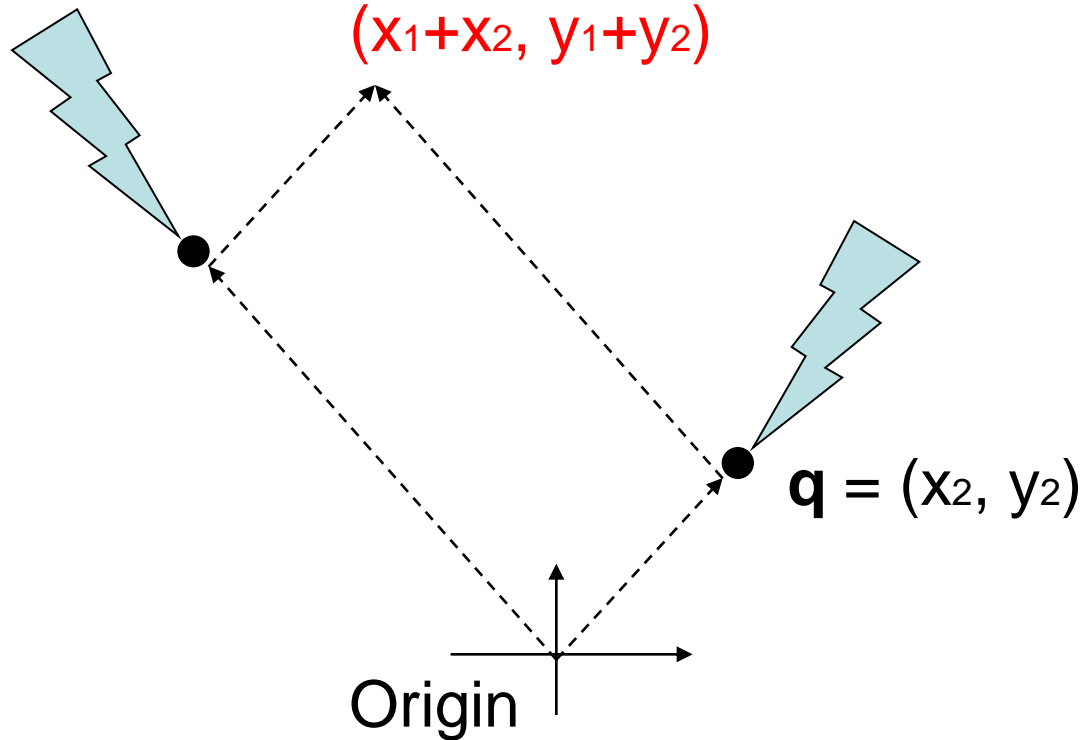


- **Vector sum**

- (x_1, y_1) and (x_2, y_2) are considered as vectors from the origin to \mathbf{p} and \mathbf{q} , respectively.

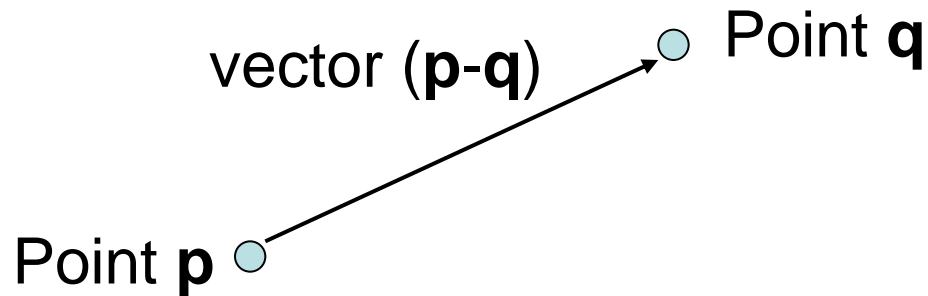
If you select a different origin, ...

$$\mathbf{p} = (x_1, y_1)$$



- If you choose a different coordinate frame, you will get a different result

Points and Vectors



- A **point** is a position specified with coordinate values.
- A **vector** is specified as the difference between two points.
- If an **origin** is specified, then a point can be represented by a vector from the origin.
- But, a point is still not a vector in **coordinate-free** concepts.

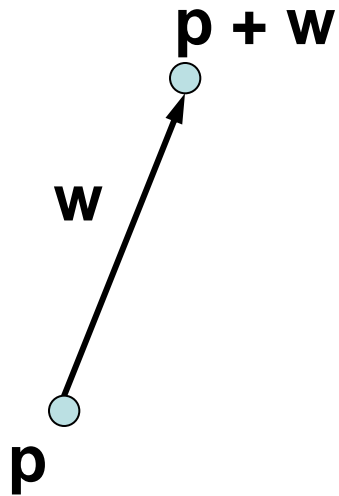
Vector and Affine Spaces

- ***Vector space***
 - Includes vectors and related operations
 - No points
- ***Affine space***
 - Superset of vector space
 - Includes vectors, points, and related operations

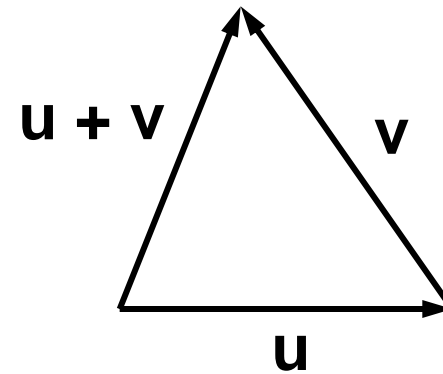
Coordinate-Invariant Geometric Operations

- Addition
- Subtraction
- Scalar multiplication
- Linear combination
- Affine combination

Addition



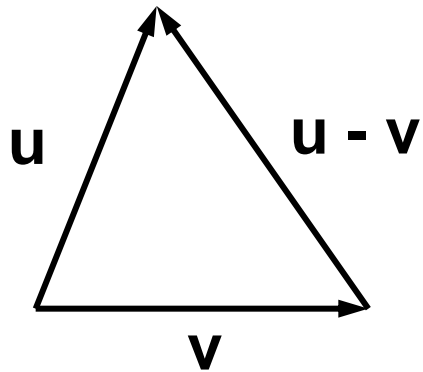
$p + w$ is a point



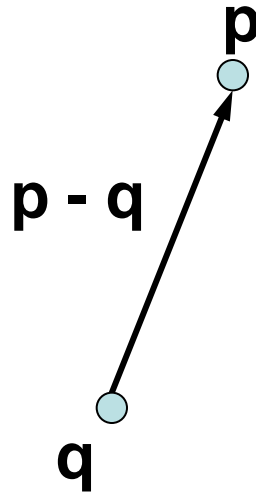
$u + v$ is a vector

u, v, w : vectors
 p, q : points

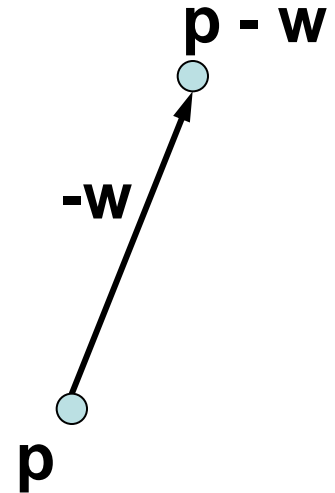
Subtraction



$u - v$ is a vector



$p - q$ is a vector



$p - w$ is a point

u, v, w : vectors
 p, q : points

Scalar Multiplication

scalar • vector = vector

1 • point = point

0 • point = vector

$c \cdot \text{point} = (\text{undefined})$ if $(c \neq 0, 1)$

Linear Combination

- A linear space is ***spanned*** by a set of bases
 - Any point in the space can be represented as a linear combination of bases

$$\sum_{i=0}^N c_i \mathbf{v}_i = c_0 \mathbf{v}_0 + c_1 \mathbf{v}_1 + \cdots + c_N \mathbf{v}_N = \mathbf{v}$$

Affine Combination

$$\begin{aligned}\sum_{i=0}^N c_i \mathbf{p}_i &= c_0 \mathbf{p}_0 + c_1 \mathbf{p}_1 + \cdots + c_N \mathbf{p}_N \\ &= \left(\sum_{i=0}^N c_i \right) \mathbf{p}_0 + \sum_{i=1}^N c_i (\mathbf{p}_i - \mathbf{p}_0)\end{aligned}$$

$$\begin{aligned}\sum c_k \mathbf{P}_k &= \mathbf{P}_0 + \sum c_k (\mathbf{P}_k - \mathbf{P}_0) & \sum c_k &= 1 \text{ (point)} \\ &= \sum c_k (\mathbf{P}_k - \mathbf{P}_0) & \sum c_k &= 0 \text{ (vector)} \\ &= \text{undefined} & \sum c_k &\neq 0, 1\end{aligned}$$

Examples

- $(\mathbf{p} + \mathbf{q}) / 2$: midpoint of line \mathbf{pq}
- $(\mathbf{p} + \mathbf{q}) / 3$: not valid
- $(\mathbf{p} + \mathbf{q} + \mathbf{r}) / 3$: center of gravity of $\Delta\mathbf{pqr}$
- $(\mathbf{p}/2 + \mathbf{q}/2 - \mathbf{r})$: a vector from \mathbf{r} to the midpoint of \mathbf{q} and \mathbf{p}

Summary

1. point + point = undefined
2. point - point = vector
3. point \pm vector = point
4. vector \pm vector = vector
5. scalar \cdot vector = vector
6. \sum scalar \cdot vector = vector
7. scalar \cdot point = point
iff scalar = 1
= vector
iff scalar = 0
= undefined
otherwise
8. \sum scalar \cdot point = point
iff \sum scalar = 1
= vector
iff \sum scalar = 0
= undefined
otherwise

Matrix Representation

- Use an “extra” coordinate
 - In 3-dimensional spaces
 - Point : $(x, y, z, 1)$
 - Vector : $(x, y, z, 0)$

- For example

$$\begin{array}{ccc} (x_1, y_1, z_1, 1) & + & (x_2, y_2, z_2, 1) = (x_1+x_2, y_1+y_2, z_1+z_2, 2) \\ \textit{point} & & \textit{point} \qquad \qquad \textit{undefined} \end{array}$$

$$\begin{array}{ccc} (x_1, y_1, z_1, 1) & - & (x_2, y_2, z_2, 1) = (x_1-x_2, y_1-y_2, z_1-z_2, 0) \\ \textit{point} & & \textit{point} \qquad \qquad \textit{vector} \end{array}$$

$$\begin{array}{ccc} (x_1, y_1, z_1, 1) & + & (x_2, y_2, z_2, 0) = (x_1+x_2, y_1+y_2, z_1+z_2, 1) \\ \textit{point} & & \textit{vector} \qquad \qquad \textit{point} \end{array}$$

Projective Spaces

- Homogeneous coordinates
 - $(x, y, z, w) = (x/w, y/w, z/w, 1)$
 - Useful for handling perspective projection
- But, it is algebraically inconsistent !!

$$(1,0,0,1) + (1,1,0,1) = (2,1,0,2) = (1, \frac{1}{2}, 0, 1)$$

|| || ✘

$$(1,0,0,1) + (2,2,0,2) = (3,2,0,3) = (1, \frac{2}{3}, 0, 1)$$

Course Overview

- **Introduction and Overview** (5 min)
- **Coordinate-Invariant Geometric Programming** (20 min)
- **Programming with Orientations and Rotations** (35 min)
 - Representing orientations and rotations
 - Analogy between points/vectors and orientations/rotations
 - Coordinate-invariant operations between orientations and rotations
- **Programming with Motion Capture Data** (10 min)
- **Practical examples** (30 min)

Orientation and Rotation

- **Not intuitive**
 - Formal definitions are also confusing
- **Many different ways to describe**
 - Rotation (direction cosine) matrix
 - Euler angles
 - Axis-angle
 - Rotation vector
 - Helical angles
 - Unit quaternions

Orientation and Rotation

- ***Rotation***

- Circular movement

- ***Orientation***

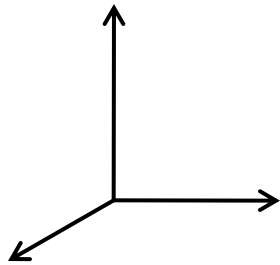
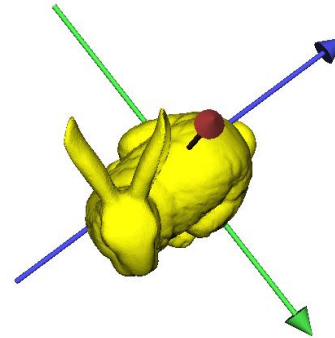
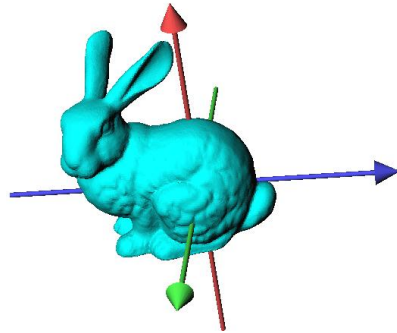
- The state of being oriented

- 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)



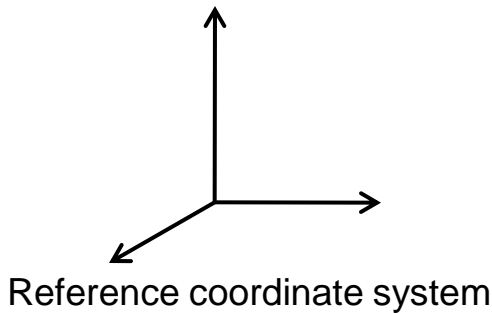
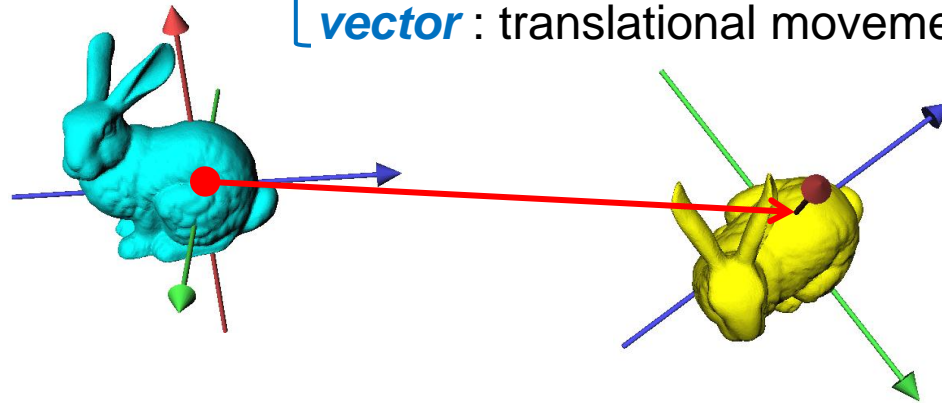
Reference coordinate system

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

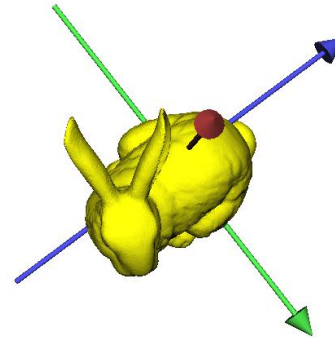
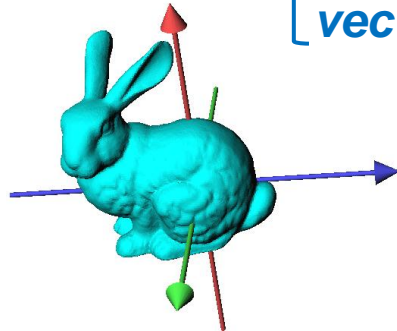


Analogy

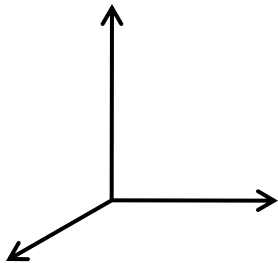
(**point** : **vector**) is similar to (**orientation** : **rotation**)

Both represent a sort of (**state** : **movement**)

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

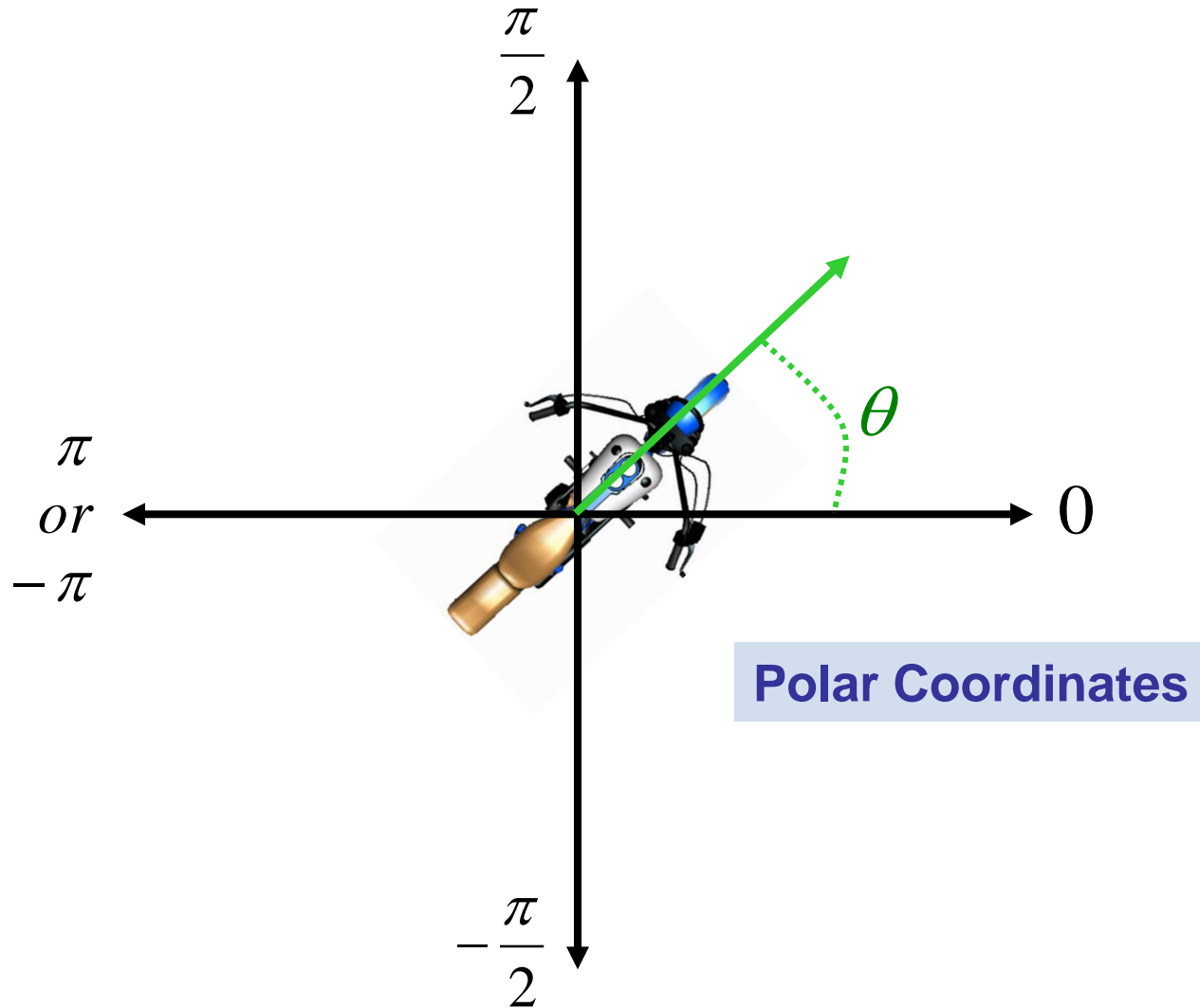


[**orientation** : the 3d orientation of the bunny
[**rotation** : circular movement

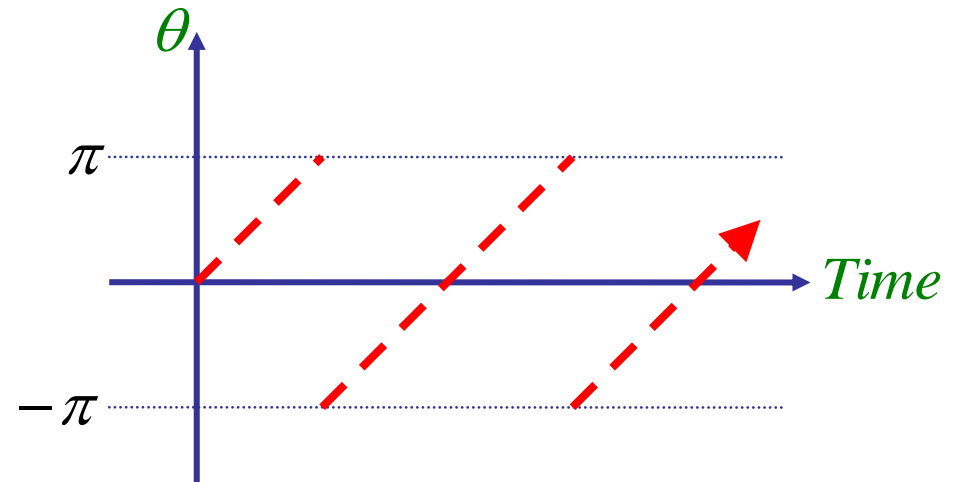
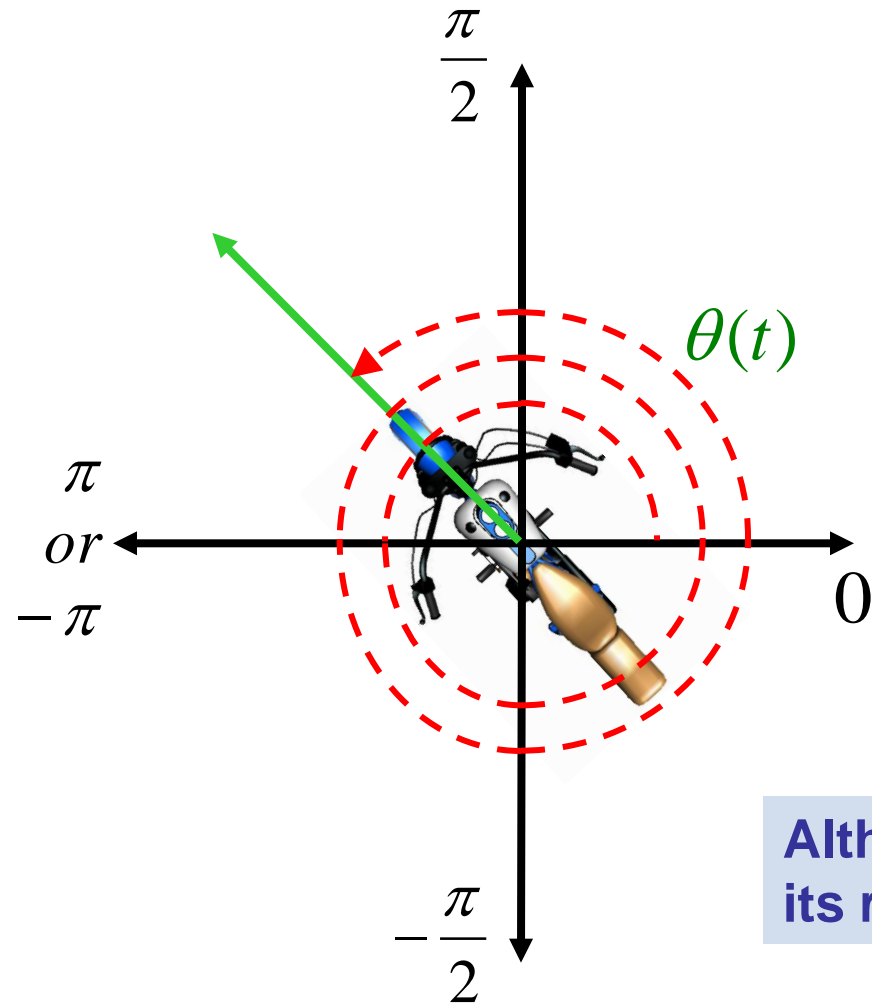


Reference coordinate system

2D Orientation

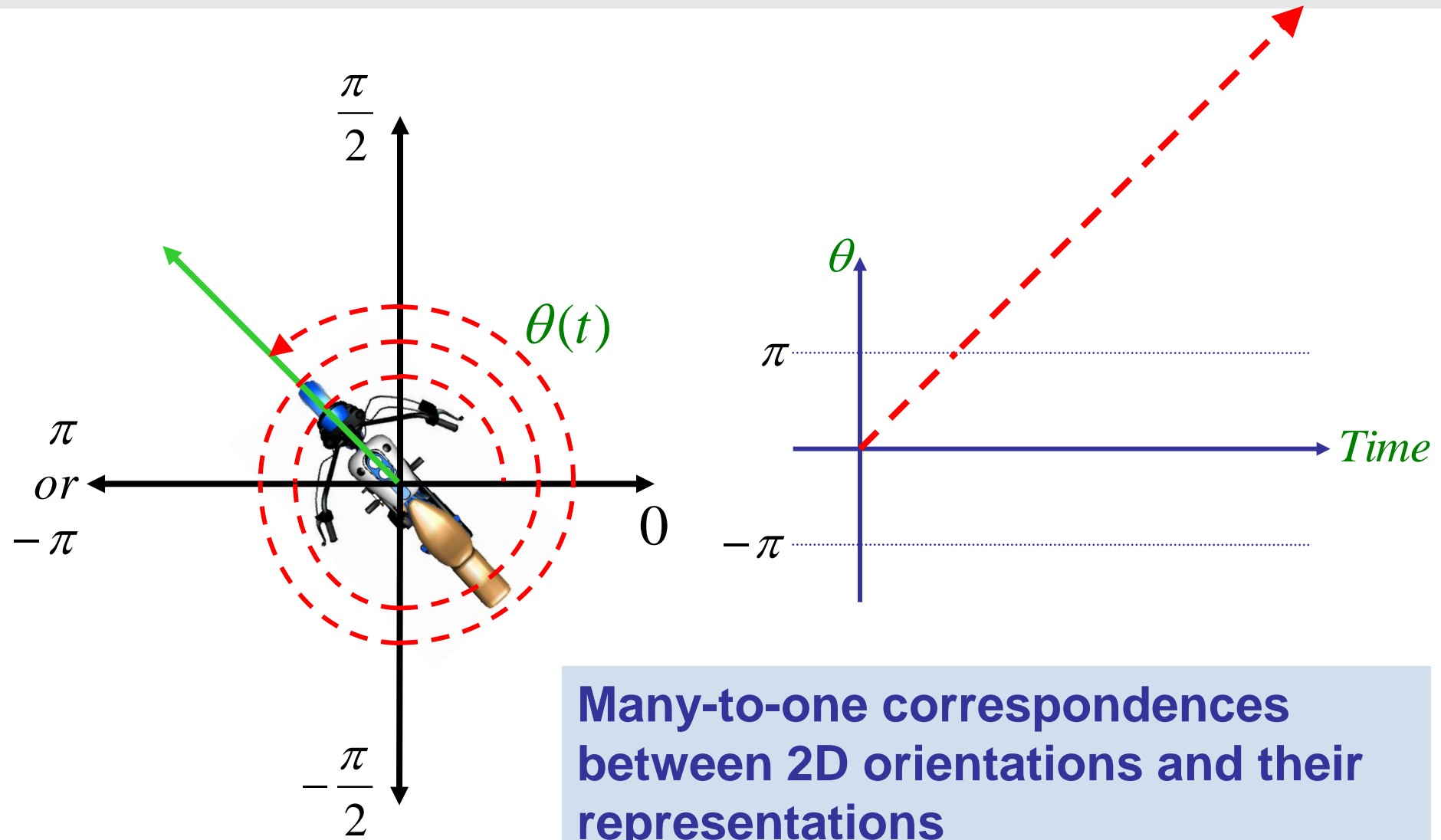


2D Orientation



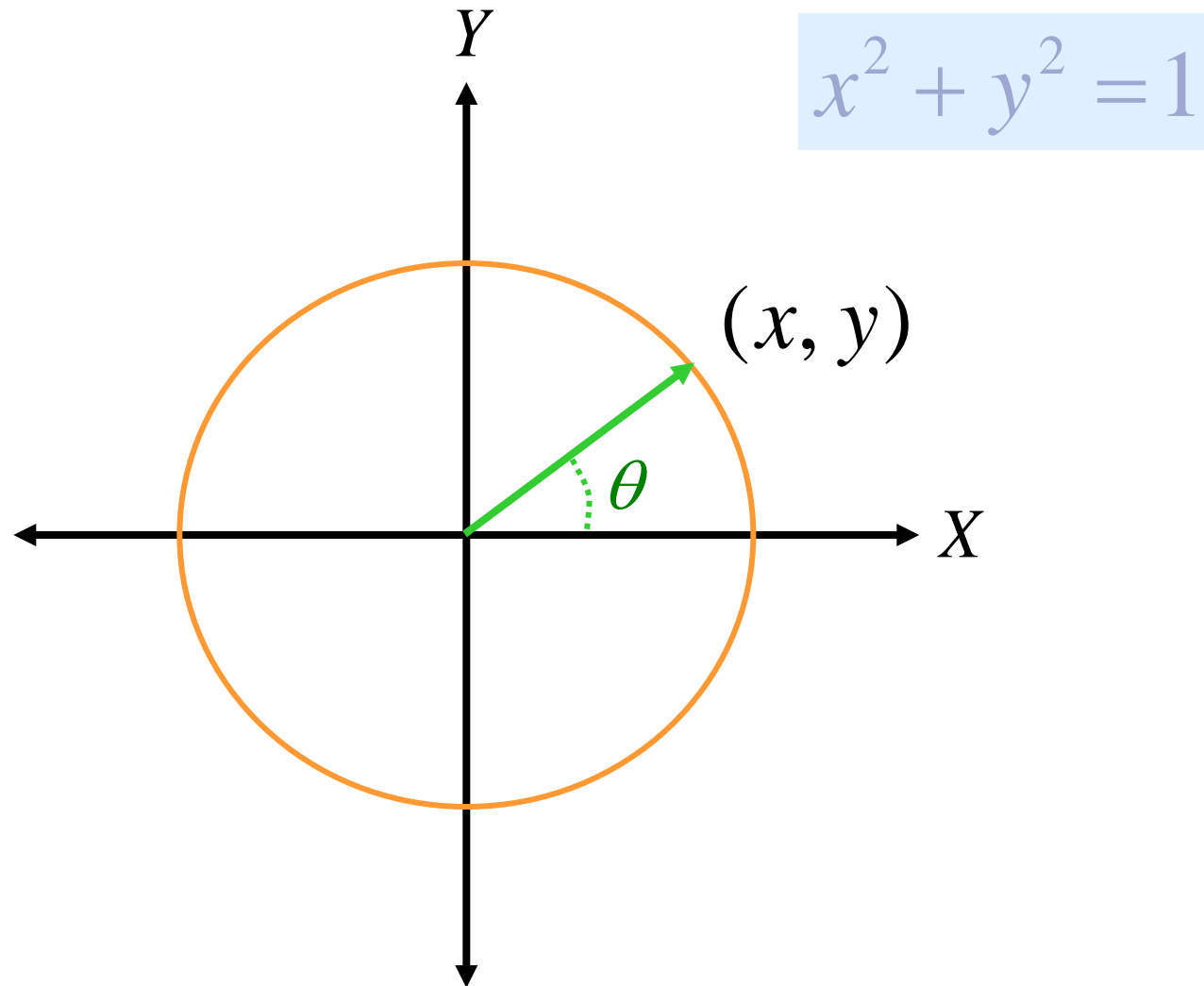
Although the motion is continuous, its representation could be discontinuous

2D Orientation



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

Extra Parameter

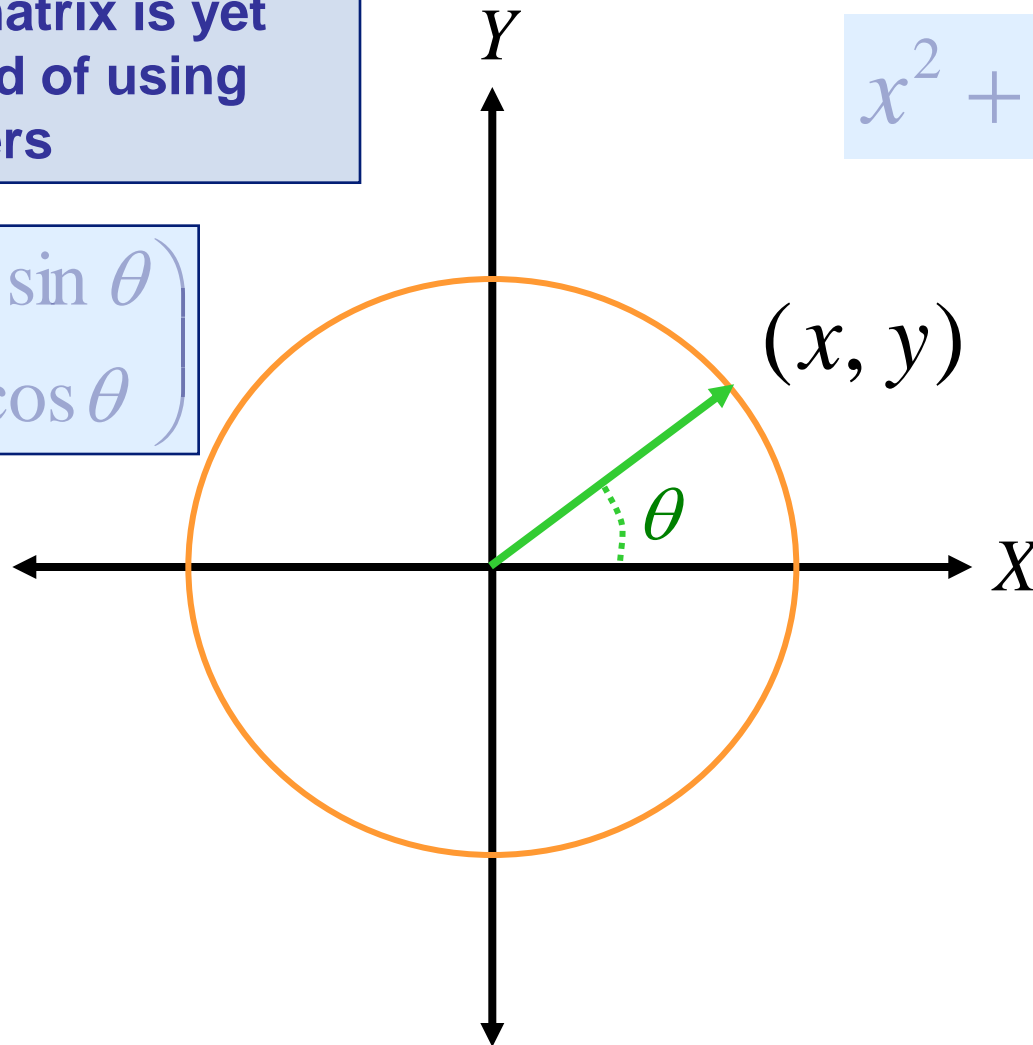


Extra Parameter

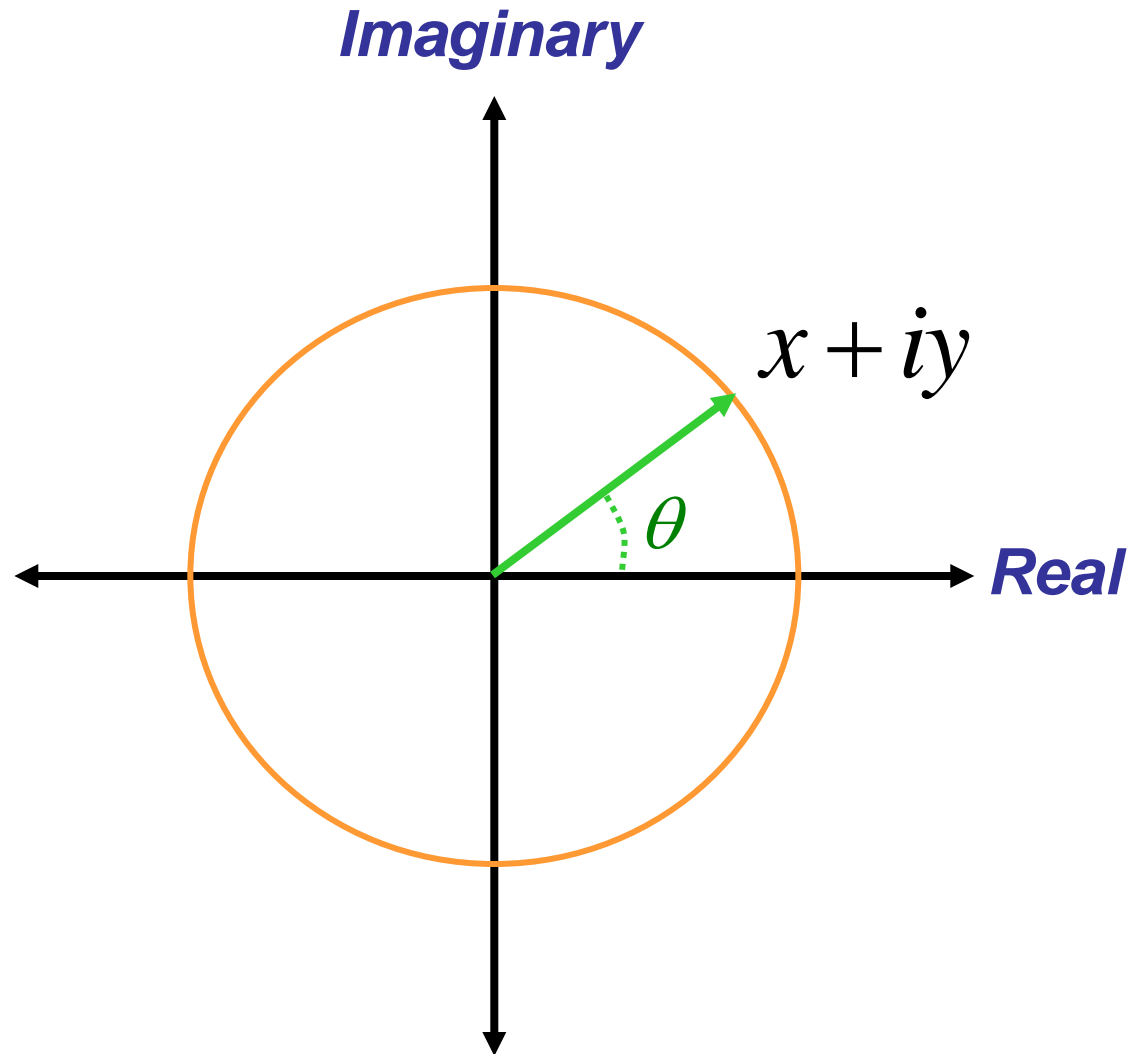
2x2 Rotation matrix is yet another method of using extra parameters

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

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

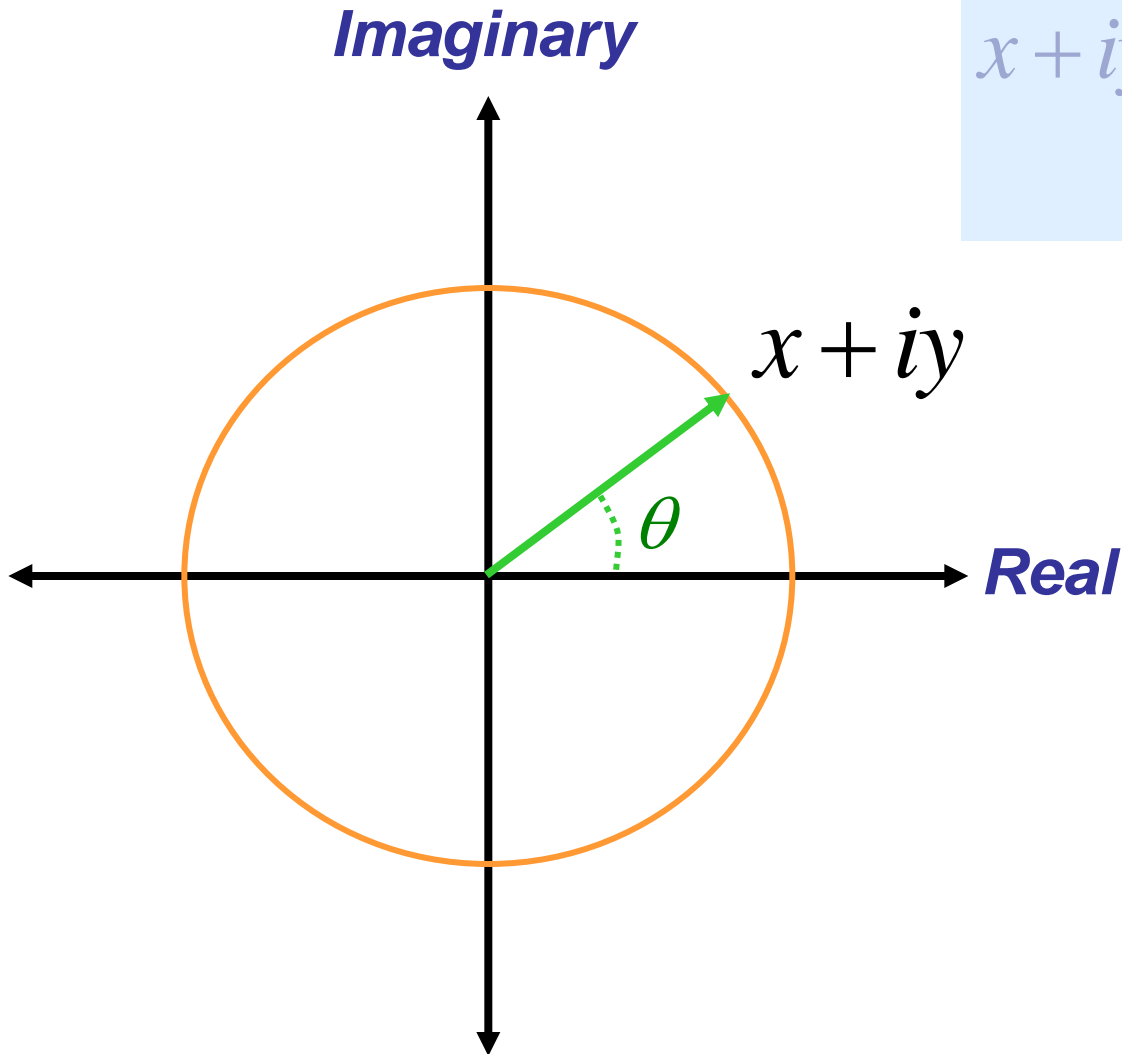


Complex Number of Unit Length



Complex Exponentiation

$$\begin{aligned}x + iy &= \cos \theta + i \sin \theta \\ &= e^{i\theta}\end{aligned}$$



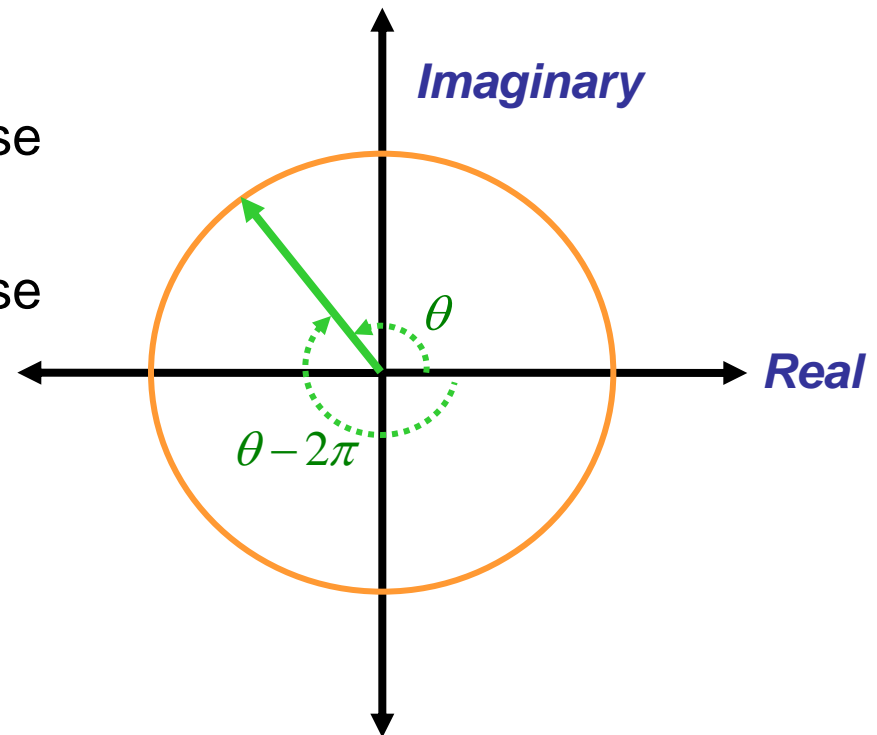
2D Rotation

- Complex numbers of unit length are
 - good for representing **2D orientations**,
 - but inadequate for **2D rotations**
- A complex number cannot distinguish different rotational movements that result in the same final orientation

Turn 120 degree counter-clockwise

Turn -240 degree clockwise

Turn 480 degree counter-clockwise



2D Rotation and Orientation

- **2D Rotation**

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

- **2D Orientation**

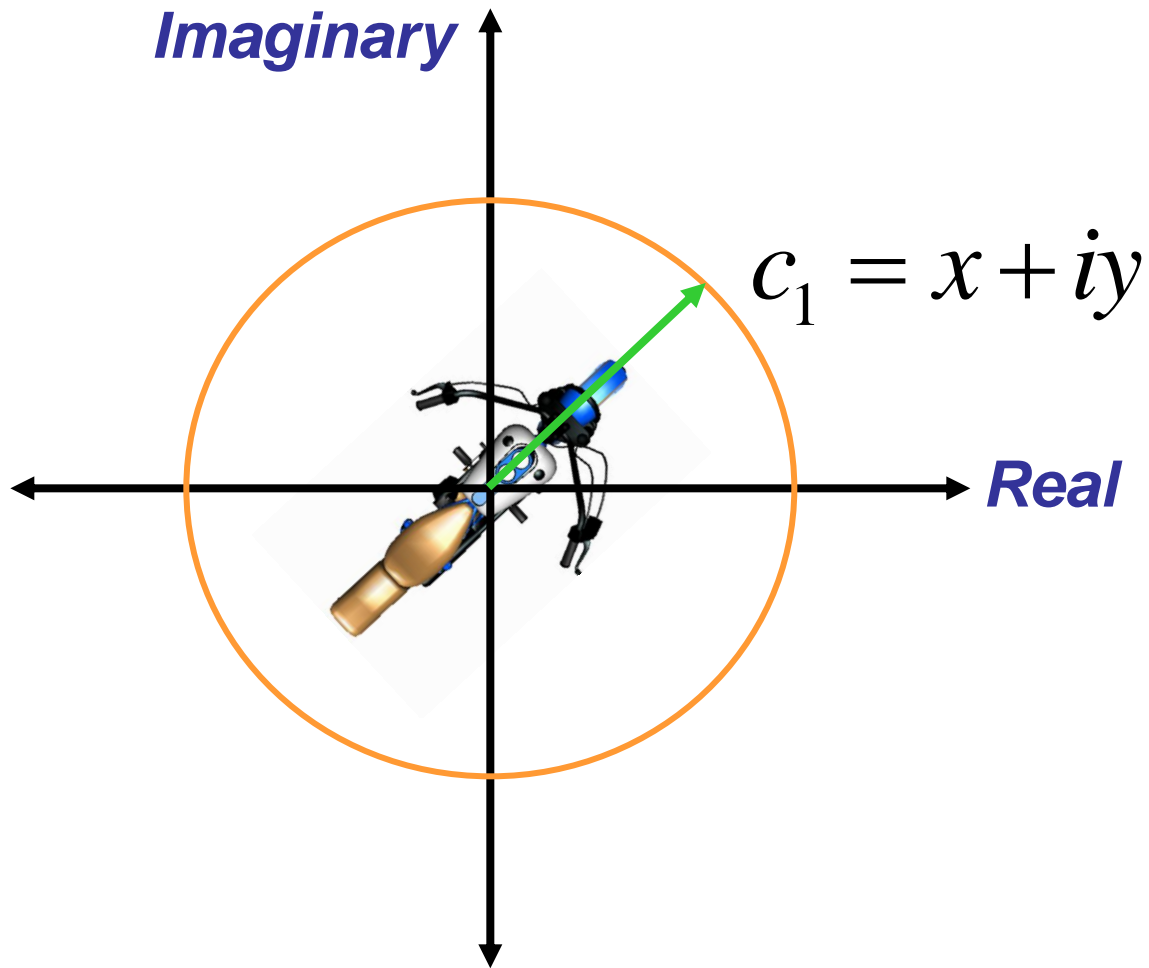
- The non-singular parameterization of **2D orientations** requires extra parameters
- Eg) Complex numbers, 2x2 rotation matrices

Operations in 2D

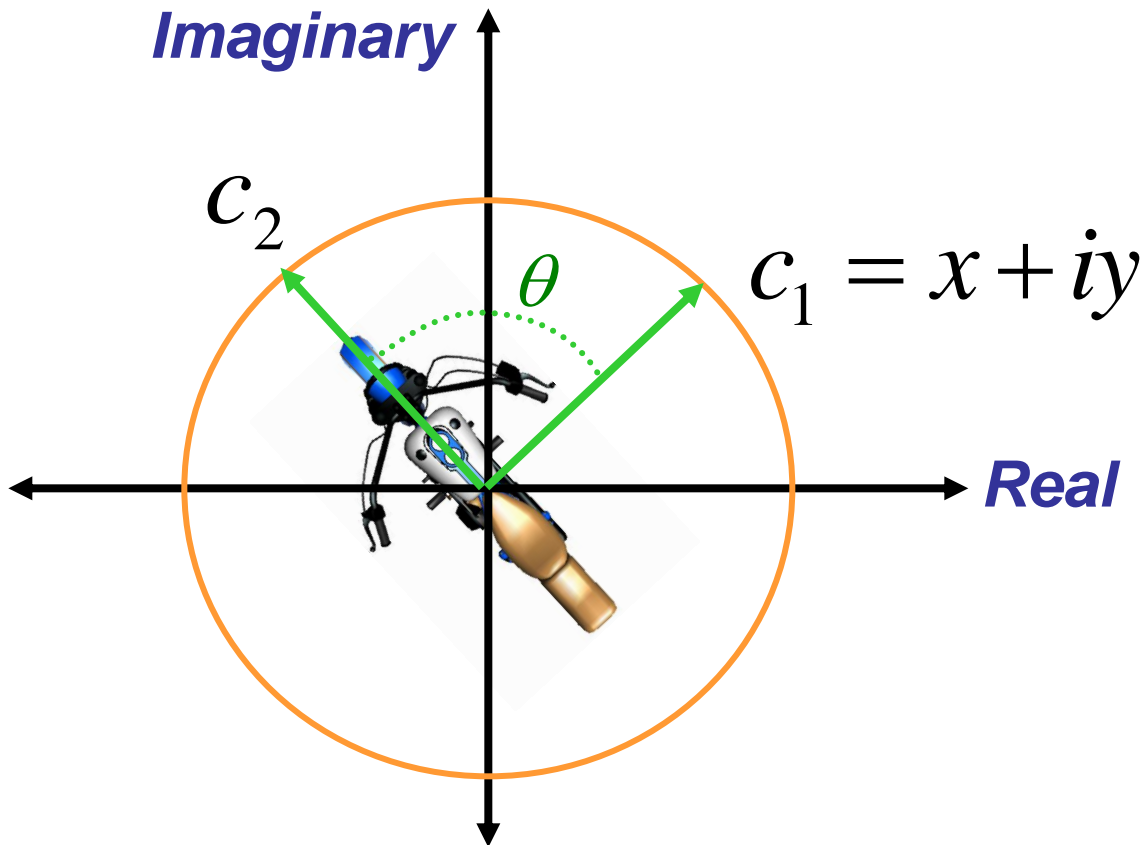
- (orientation) : complex number
- (rotation) : scalar value
- $\exp(\text{rotation})$: complex number

a.	(point) + (point)	→	(UNDEFINED)
b.	(vector) ± (vector)	→	(vector)
c.	(point) ± (vector)	→	(point)
d.	(point) - (point)	→	(vector)
g.	(scalar) · (vector)	→	(vector)
h.	(scalar) · (point)	→	(point) if scalar=1 (vector) if scalar=0 (UNDEFINED) otherwise
j.	\sum (scalar) · (vector)	→	(vector)
k.	\sum (scalar) · (point)	→	(point) if \sum scalar=1 (vector) if \sum scalar=0 (UNDEFINED) otherwise

2D Rotation and Displacement



2D Rotation and Displacement



$$c_2 = c_1 \cdot e^{i\theta}$$

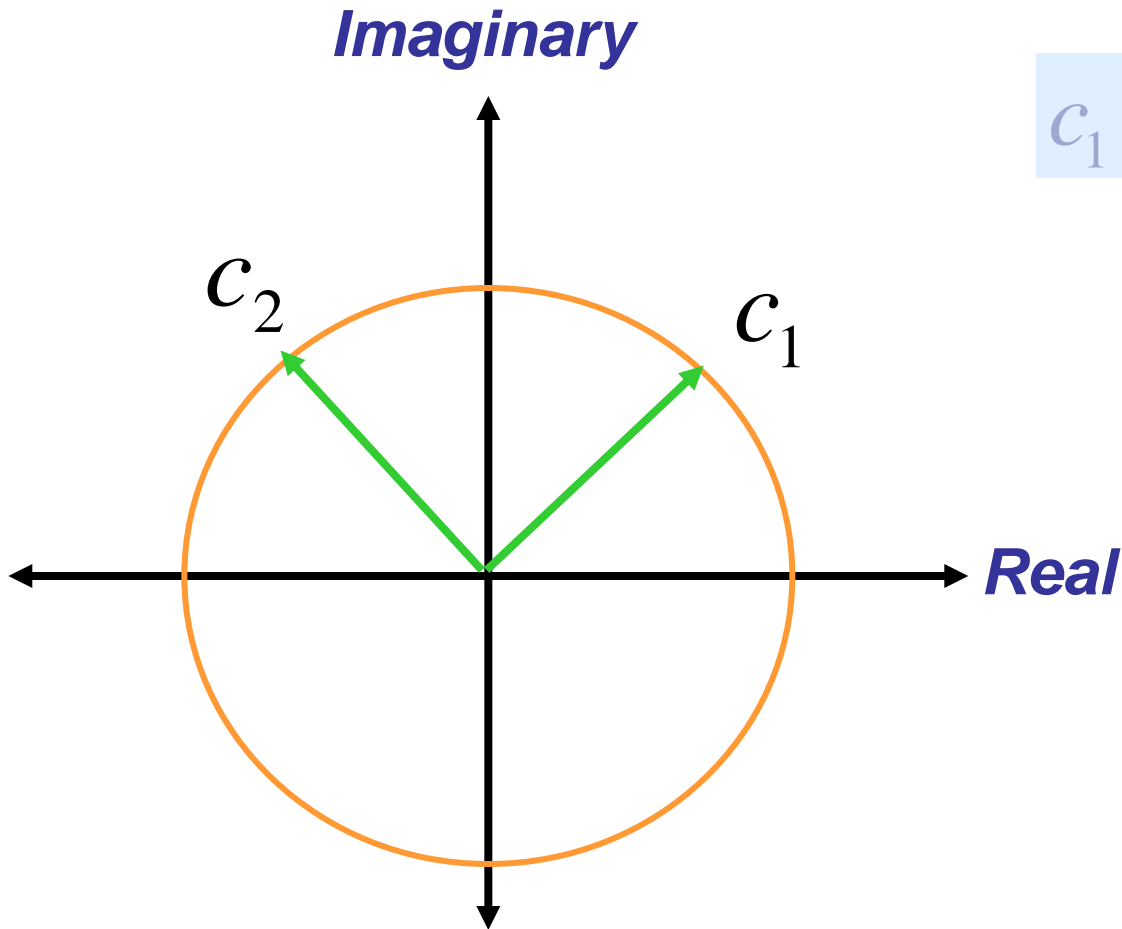
or

$$c_1^{-1} c_2 = e^{i\theta}$$

(orientation) \cdot exp(rotation) \rightarrow (orientation)
(orientation)⁻¹ \cdot (orientation) \rightarrow exp(rotation)

2D Orientation Composition

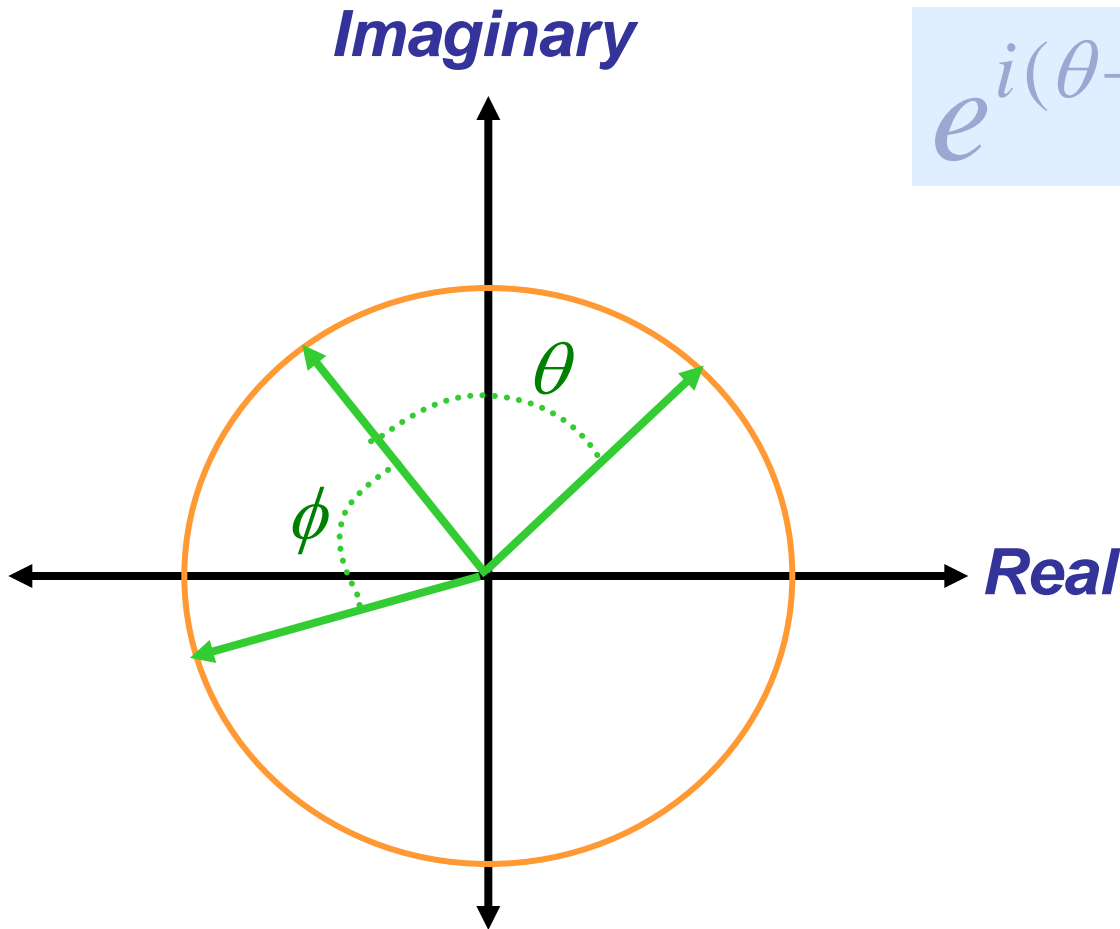
$$c_1 \cdot c_2 = (\textit{undefined})$$



(orientation) · (orientation) → (UNDEFINED)

2D Rotation Composition

$$e^{i(\theta+\phi)} = e^{i\theta} \cdot e^{i\phi}$$



`exp(rotation) · exp(rotation) → exp(rotation)`

Analogy

$(\text{point}) + (\text{point}) \rightarrow (\text{UNDEFINED})$

$(\text{vector}) \pm (\text{vector}) \rightarrow (\text{vector})$

$(\text{point}) \pm (\text{vector}) \rightarrow (\text{point})$

$(\text{point}) - (\text{point}) \rightarrow (\text{vector})$

$(\text{orientation}) \cdot (\text{orientation}) \rightarrow (\text{UNDEFINED})$

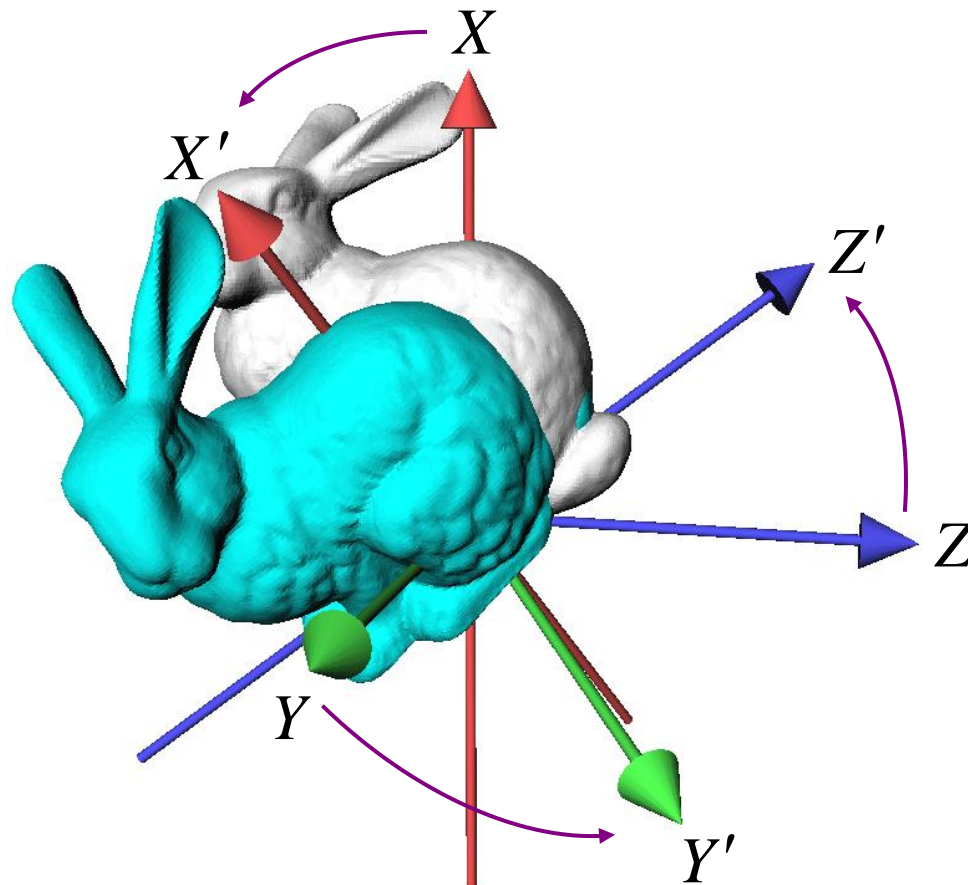
$\exp(\text{rotation}) \cdot \exp(\text{rotation}) \rightarrow \exp(\text{rotation})$

$(\text{orientation}) \cdot \exp(\text{rotation}) \rightarrow (\text{orientation})$

$(\text{orientation})^{-1} \cdot (\text{orientation}) \rightarrow \exp(\text{rotation})$

3D Orientation and Rotation

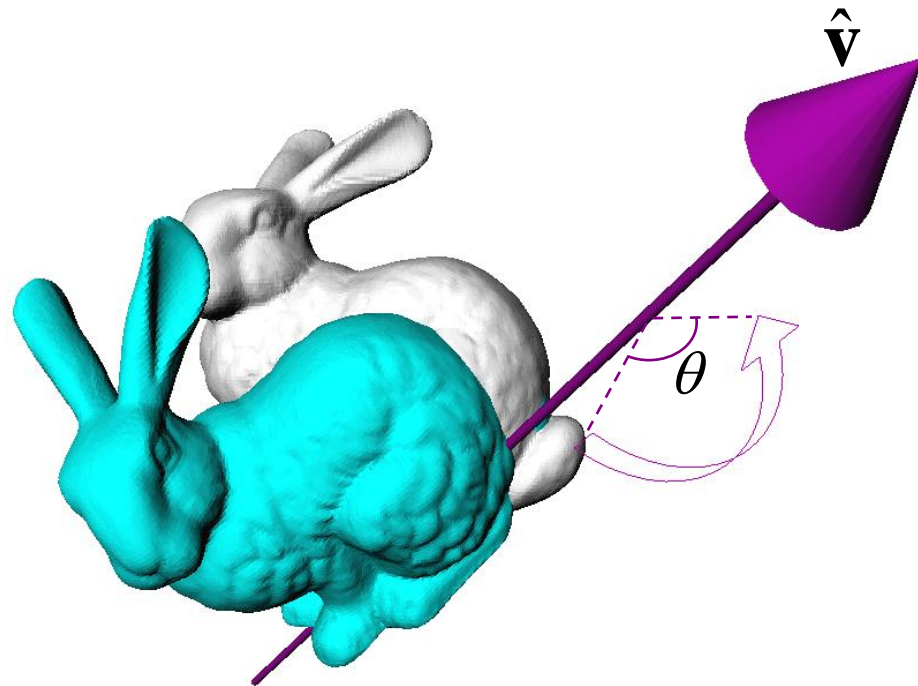
- Given two arbitrary orientations of a rigid object,



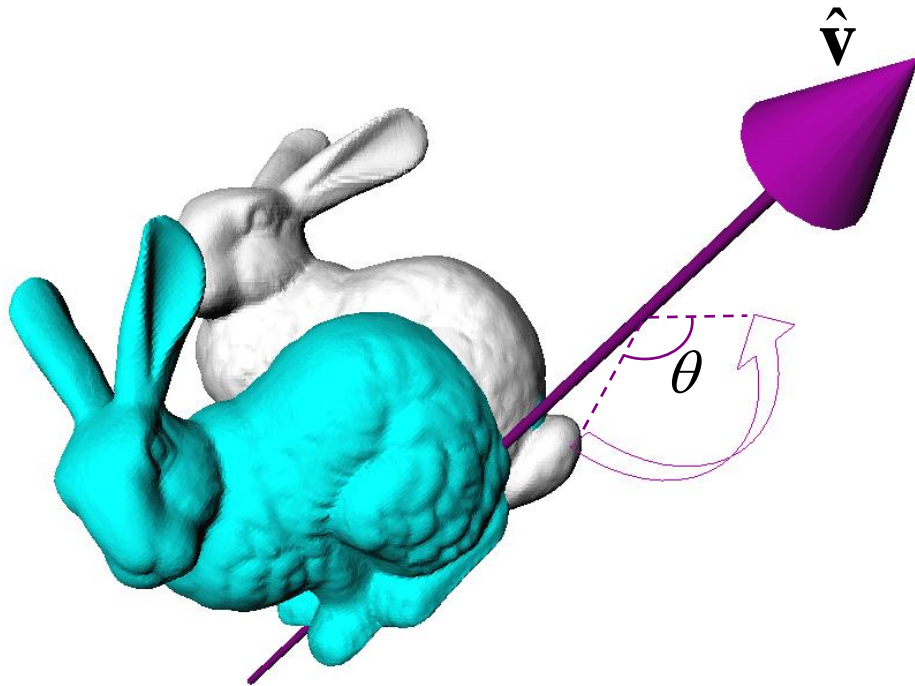
3D Orientation and Rotation

- **Euler's fundamental theorem**

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



Rotation Vector



$\hat{\mathbf{v}}$: unit vector

θ : scalar angle

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

3D Orientations and Rotations

Orientations and rotations are different in coordinate-invariant geometric programming

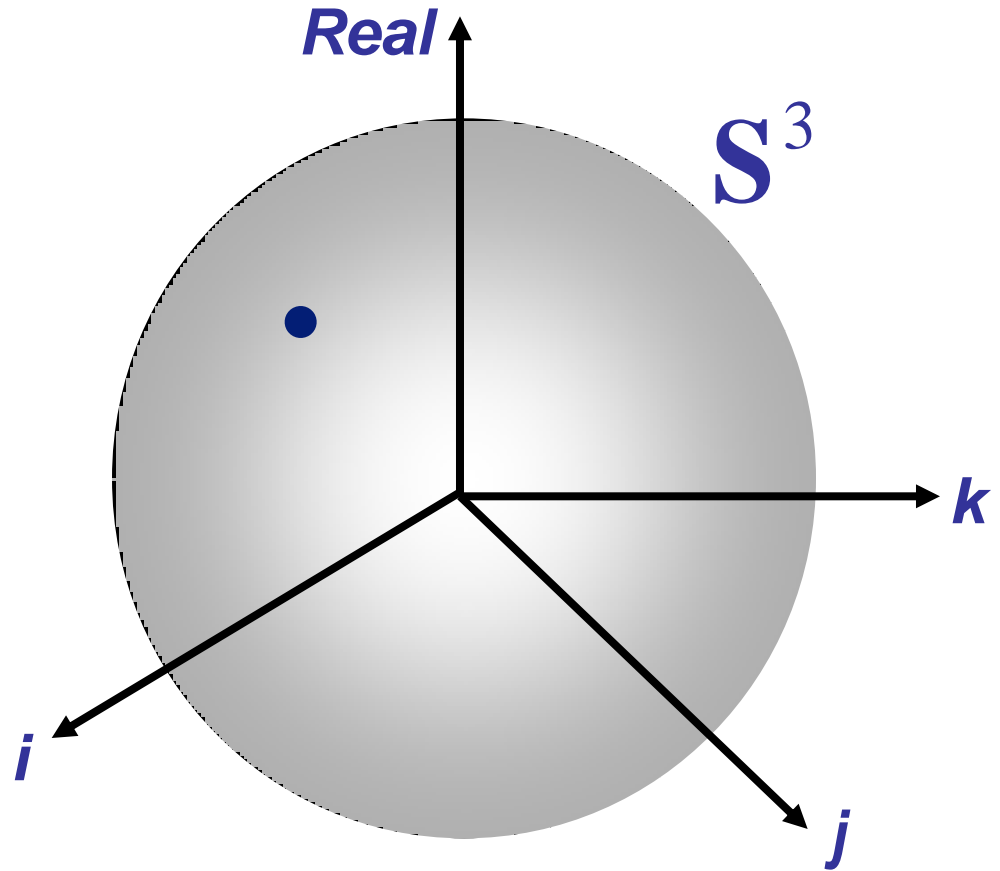
Use **unit quaternions** for **orientation** representation

– Alternatively, 3x3 orthogonal matrices

Use **3-vectors** for **rotation** representation

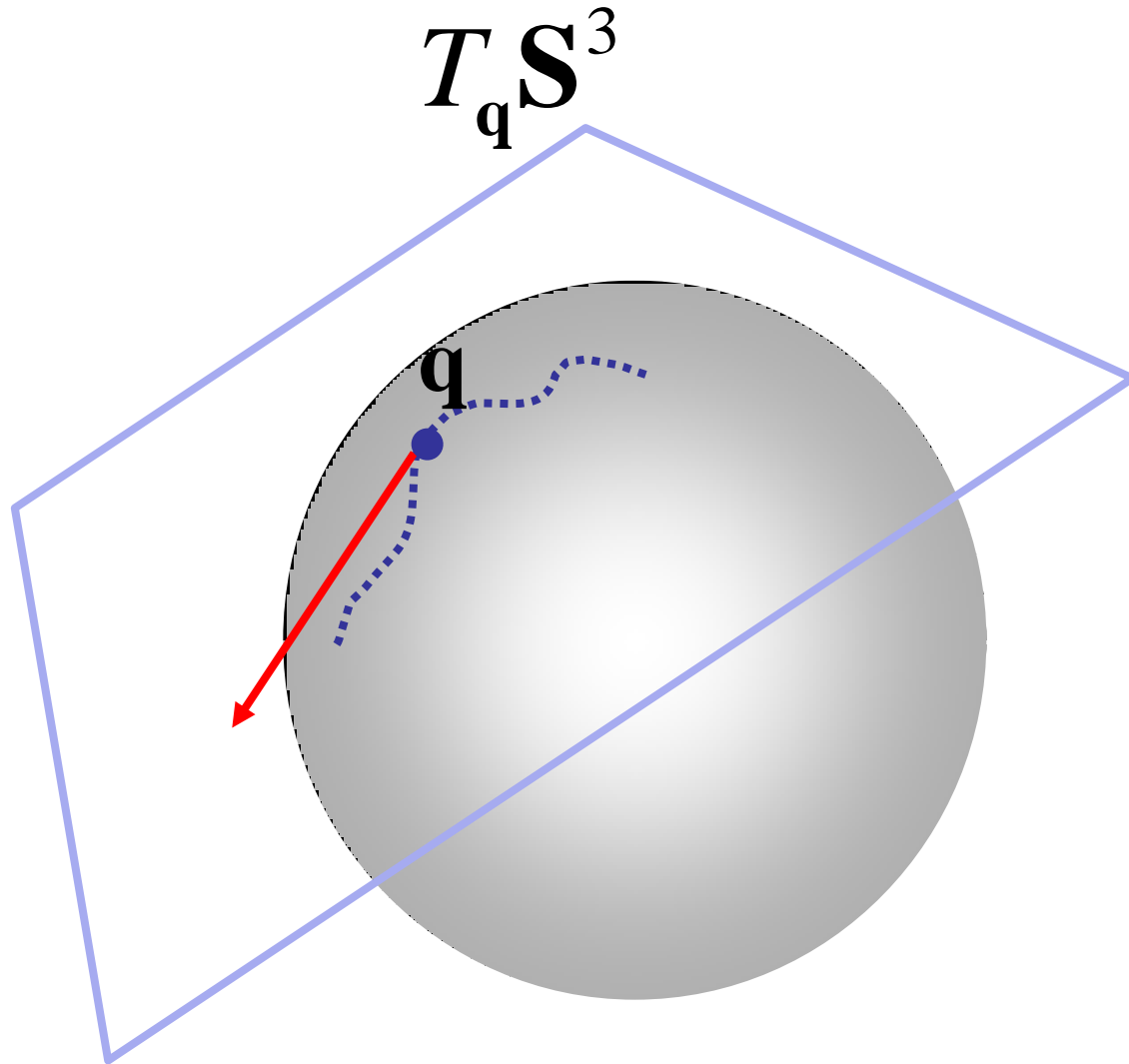
Unit Quaternions

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

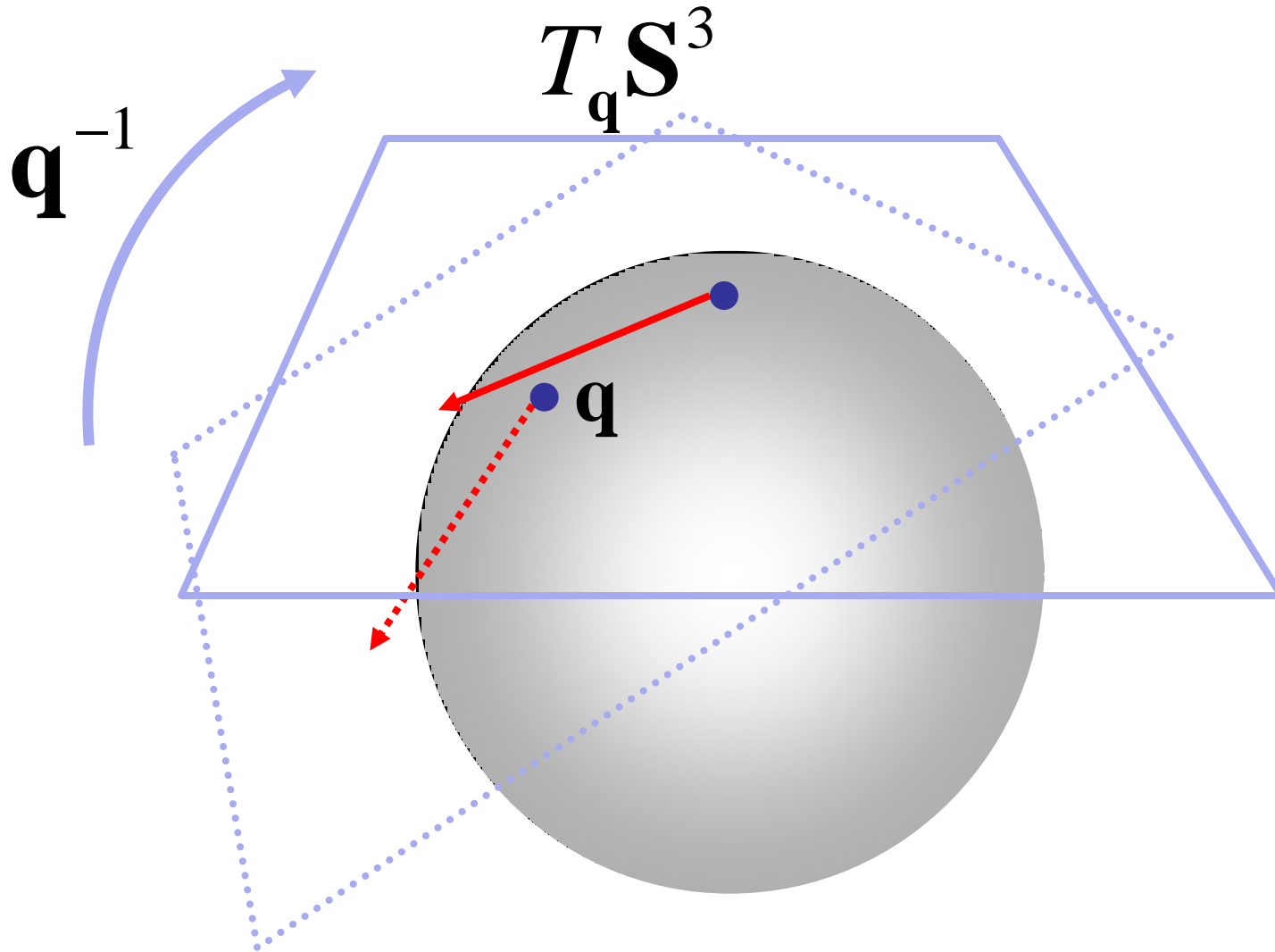


$$w^2 + x^2 + y^2 + z^2 = 1$$

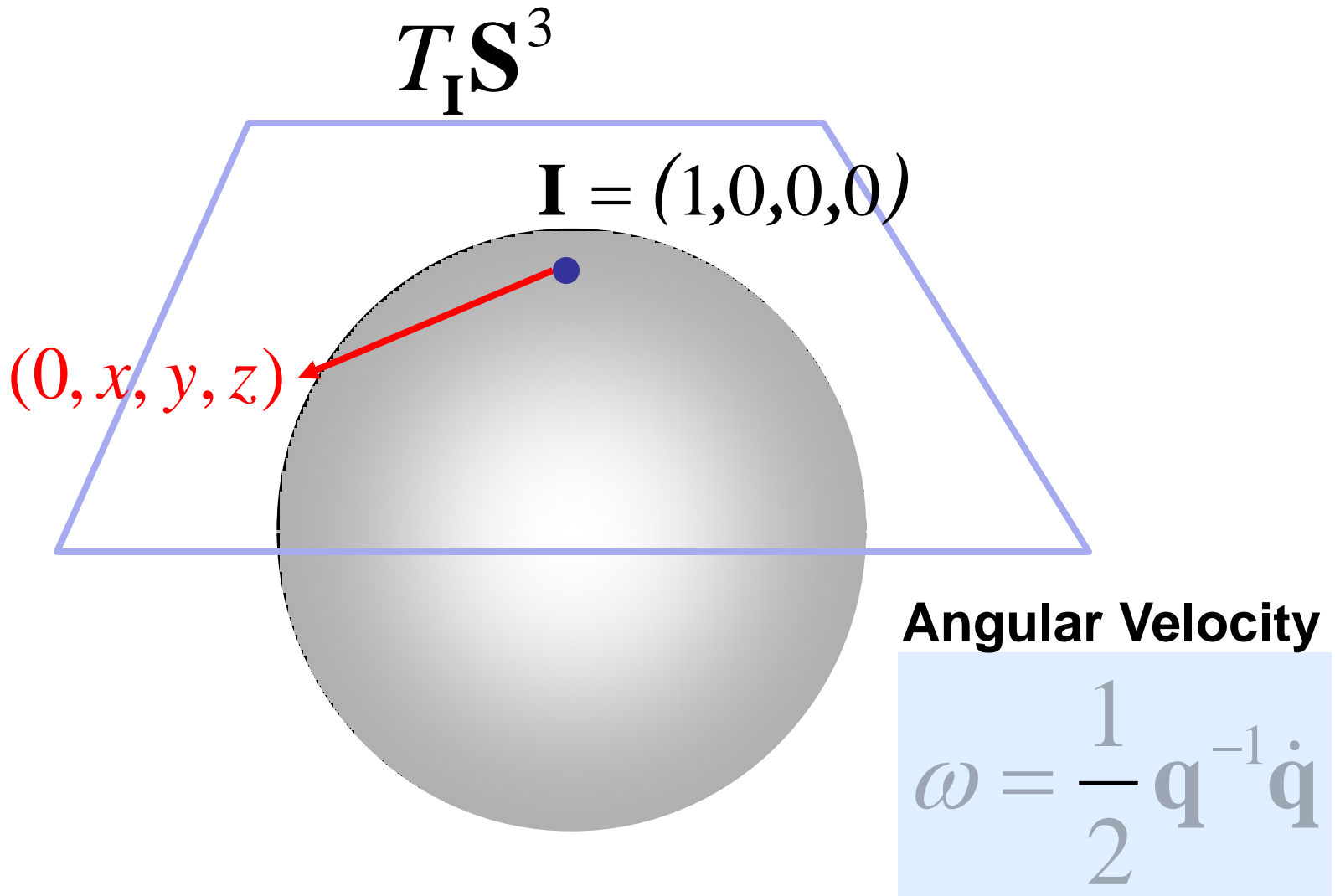
Tangent Vector (Infinitesimal Rotation)



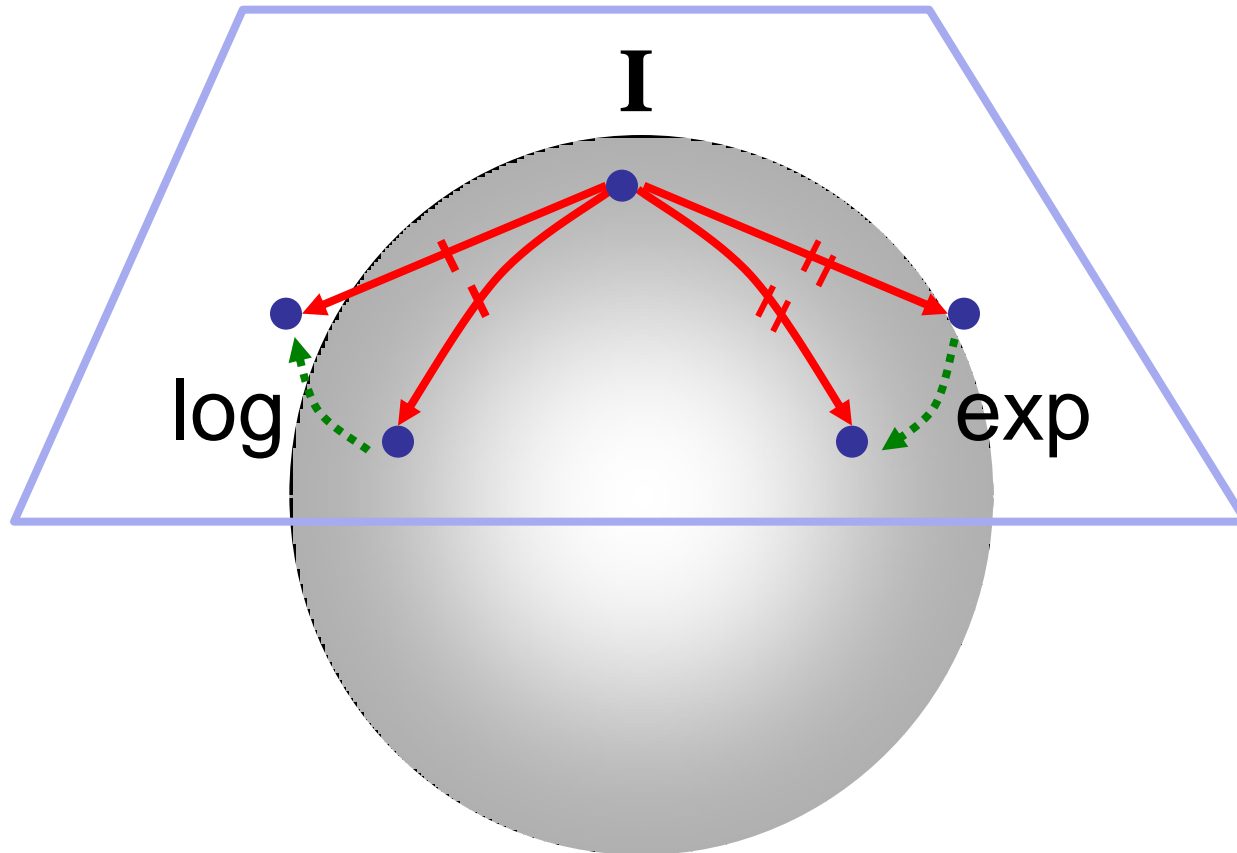
Tangent Vector (Infinitesimal Rotation)



Tangent Vector (Infinitesimal Rotation)

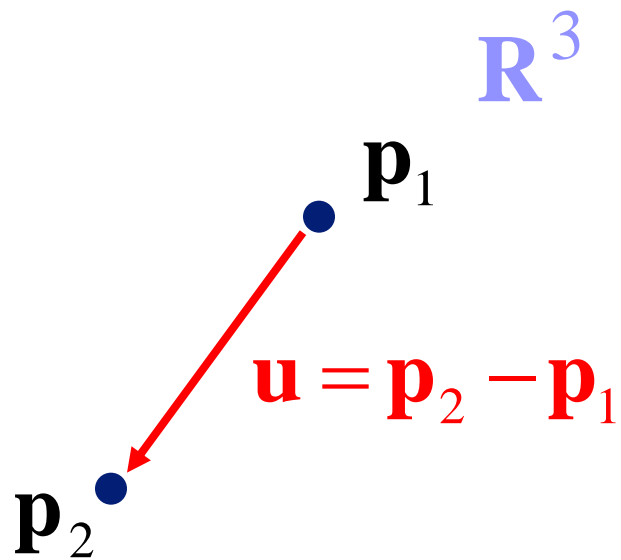


Exp and Log

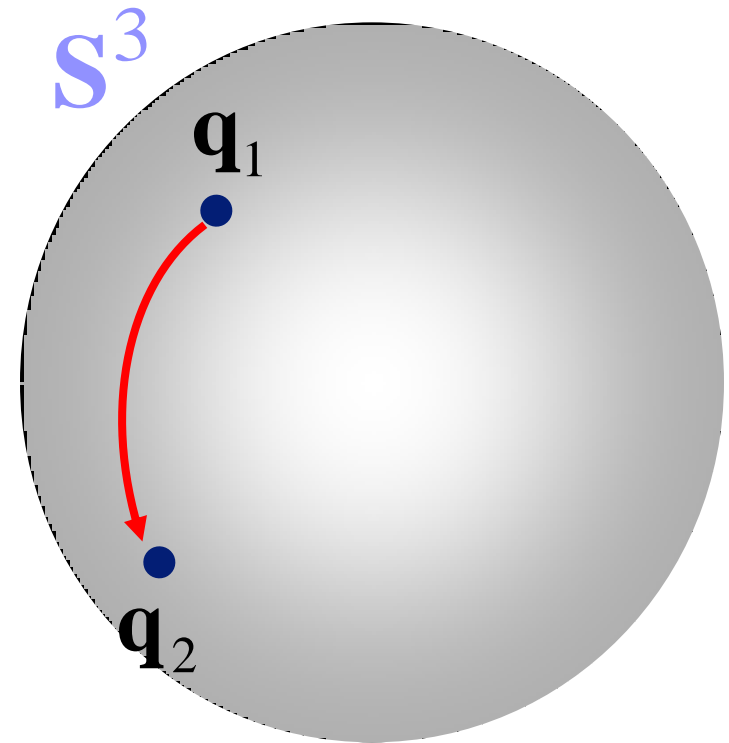


$$\exp(\mathbf{v}) = \exp(\theta \hat{\mathbf{v}}) = (\cos \theta, \hat{\mathbf{v}} \sin \theta)$$

3D Rotation and Displacement

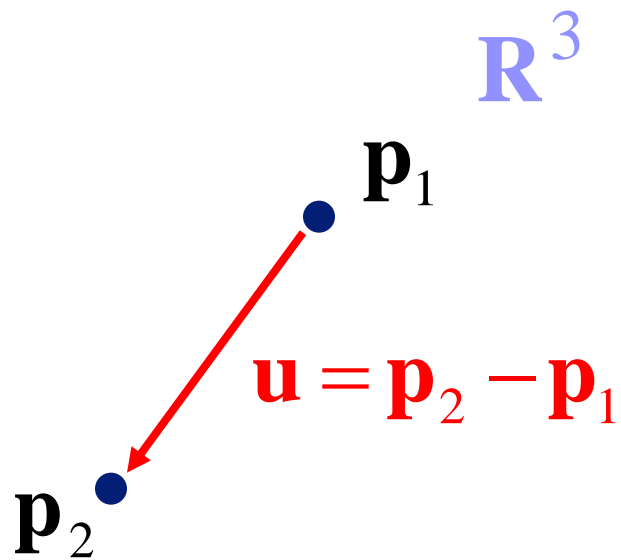


$$\begin{aligned}\mathbf{p}_2 &= \mathbf{p}_1 + \mathbf{u} \\ &= \mathbf{p}_1 + (\mathbf{p}_2 - \mathbf{p}_1)\end{aligned}$$

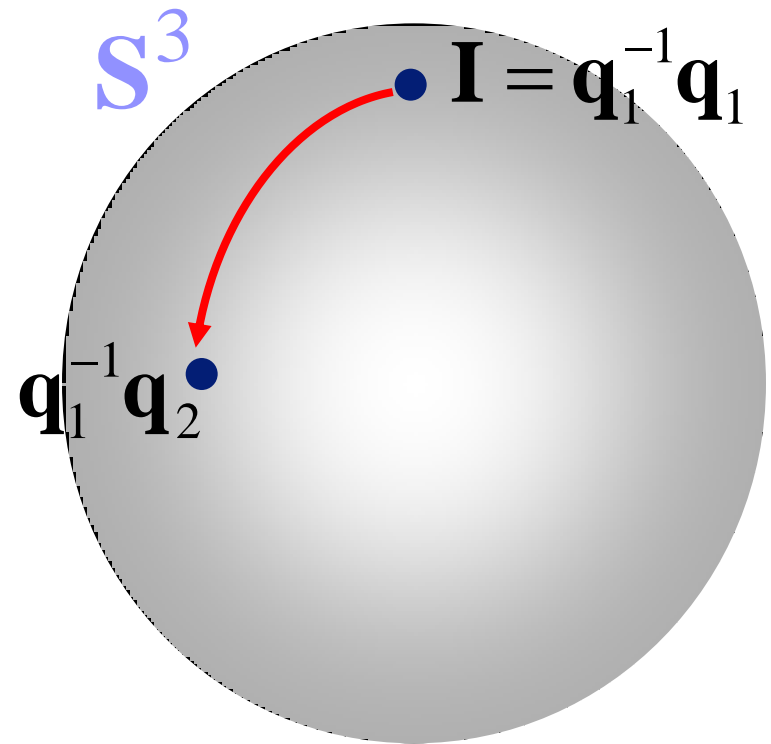


$$\begin{aligned}\mathbf{q}_2 &= \mathbf{q}_1 \exp(\mathbf{v}) \\ &= \mathbf{q}_1 \exp(\log(\mathbf{q}_1^{-1} \mathbf{q}_2))\end{aligned}$$

3D Rotation and Displacement

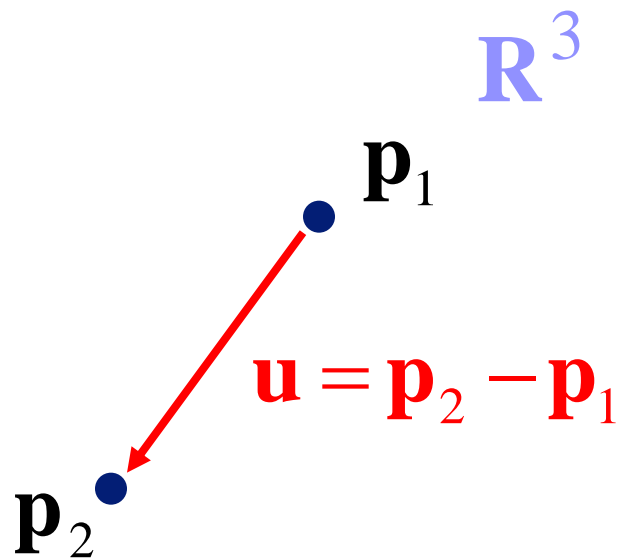


$$\begin{aligned}\mathbf{p}_2 &= \mathbf{p}_1 + \mathbf{u} \\ &= \mathbf{p}_1 + (\mathbf{p}_2 - \mathbf{p}_1)\end{aligned}$$

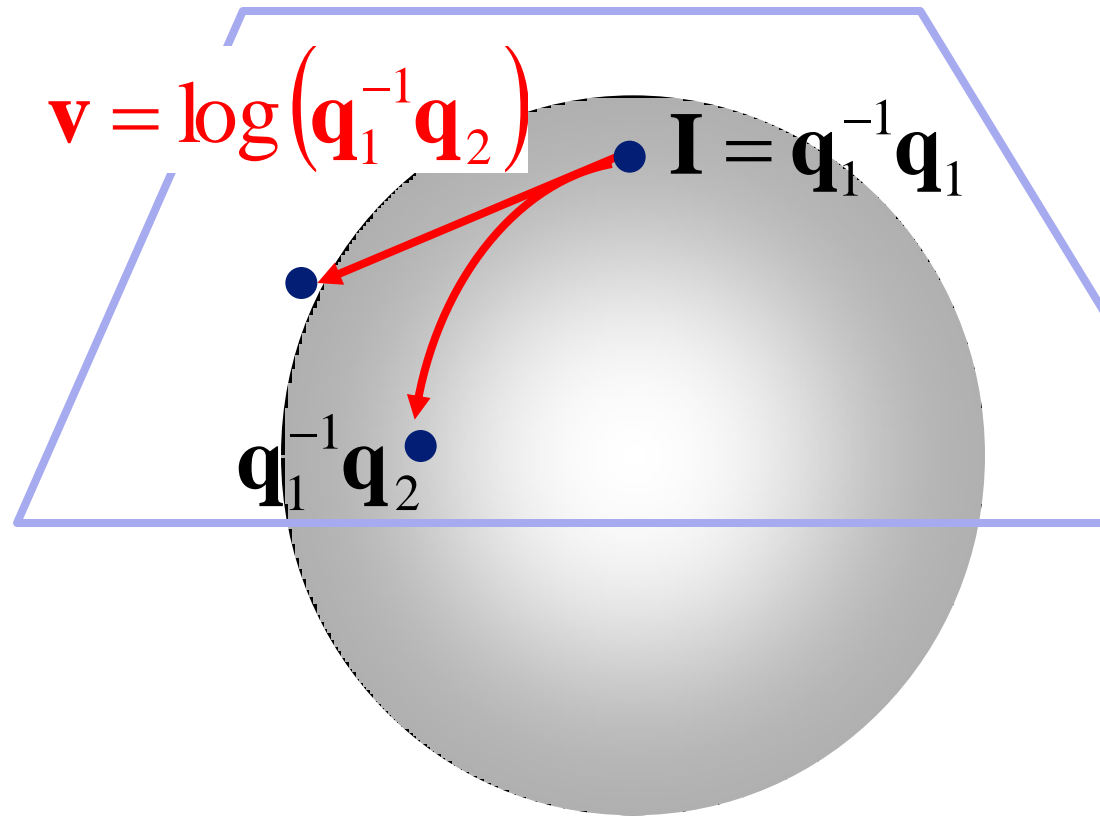


$$\begin{aligned}\mathbf{q}_2 &= \mathbf{q}_1 \exp(\mathbf{v}) \\ &= \mathbf{q}_1 \exp(\log(\mathbf{q}_1^{-1} \mathbf{q}_2))\end{aligned}$$

3D Rotation and Displacement



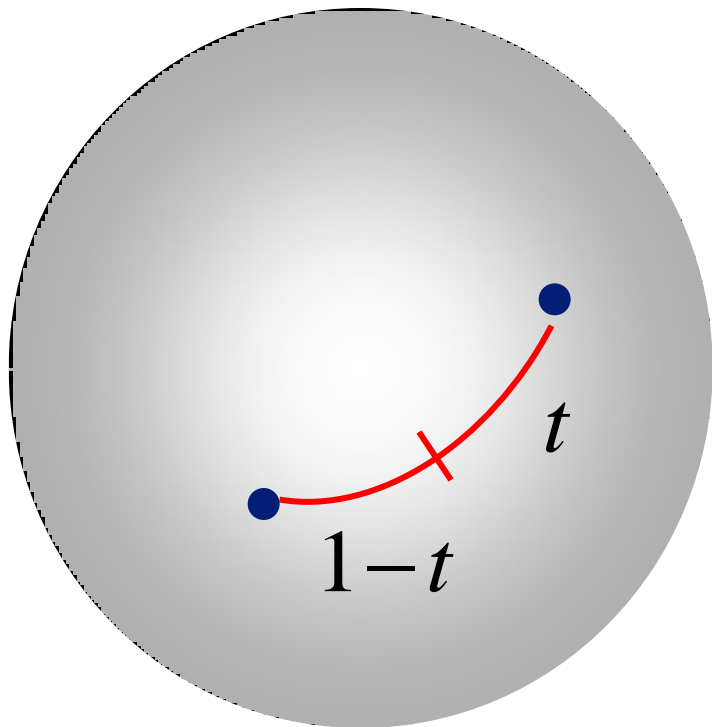
$$\begin{aligned}\mathbf{p}_2 &= \mathbf{p}_1 + \mathbf{u} \\ &= \mathbf{p}_1 + (\mathbf{p}_2 - \mathbf{p}_1)\end{aligned}$$



$$\begin{aligned}\mathbf{q}_2 &= \mathbf{q}_1 \exp(\mathbf{v}) \\ &= \mathbf{q}_1 \exp(\log(\mathbf{q}_1^{-1} \mathbf{q}_2))\end{aligned}$$

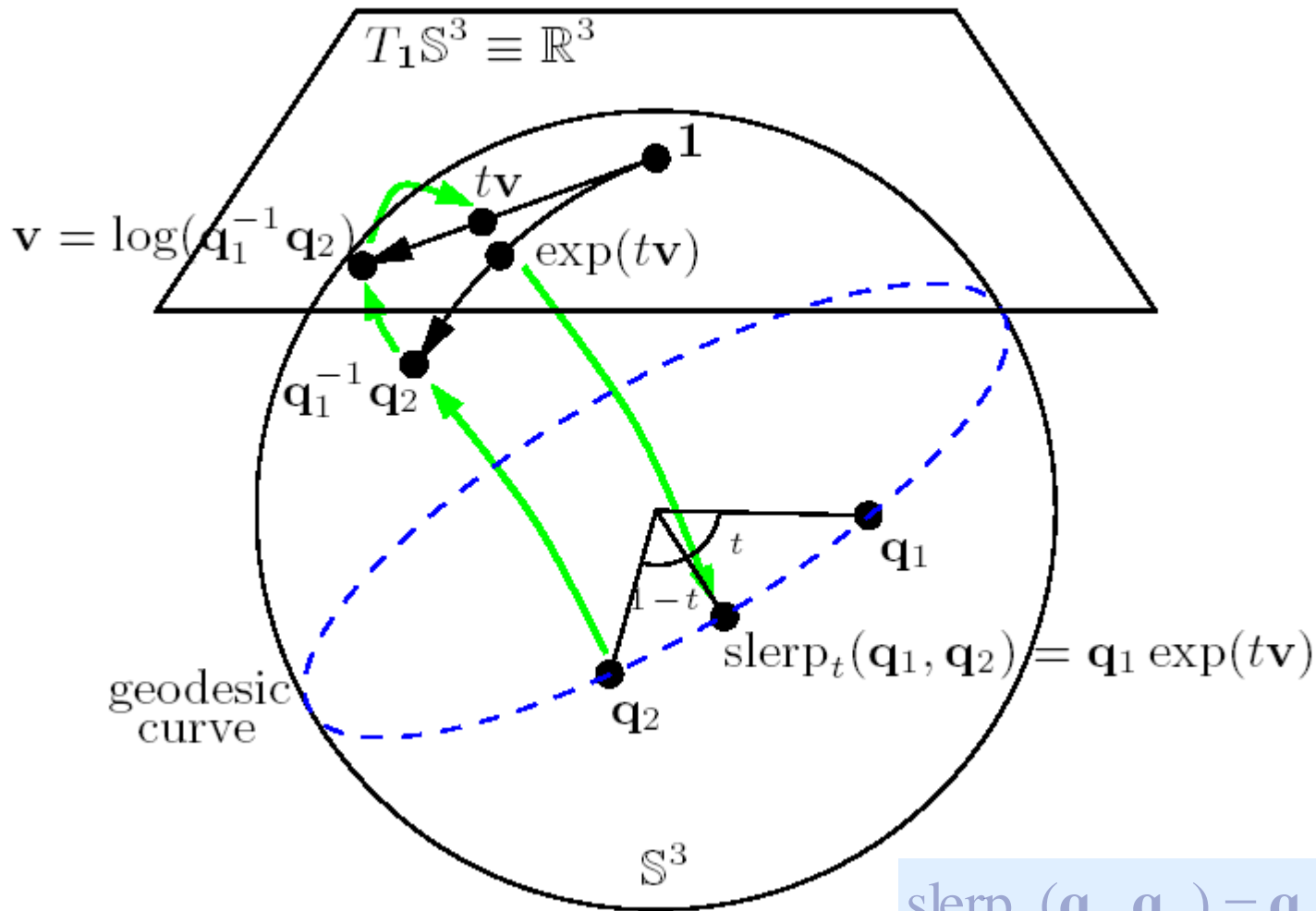
Spherical Linear Interpolation

- SLERP [Shoemake 1985]
 - Linear interpolation between two orientations



$$\begin{aligned}\text{slerp}_t(\mathbf{q}_1, \mathbf{q}_2) &= \mathbf{q}_1 (\mathbf{q}_1^{-1} \mathbf{q}_2)^t \\ &= \mathbf{q}_1 \exp(t \cdot \log(\mathbf{q}_1^{-1} \mathbf{q}_2))\end{aligned}$$

Spherical Linear Interpolation



$$\begin{aligned} \text{slerp}_t(q_1, q_2) &= q_1 (q_1^{-1} q_2)^t \\ &= q_1 \exp(t \cdot \log(q_1^{-1} q_2)) \end{aligned}$$

Analogy

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

a. (point) + (point) → (UNDEFINED)
b. (vector) ± (vector) → (vector)
c. (point) ± (vector) → (point)
d. (point) - (point) → (vector)

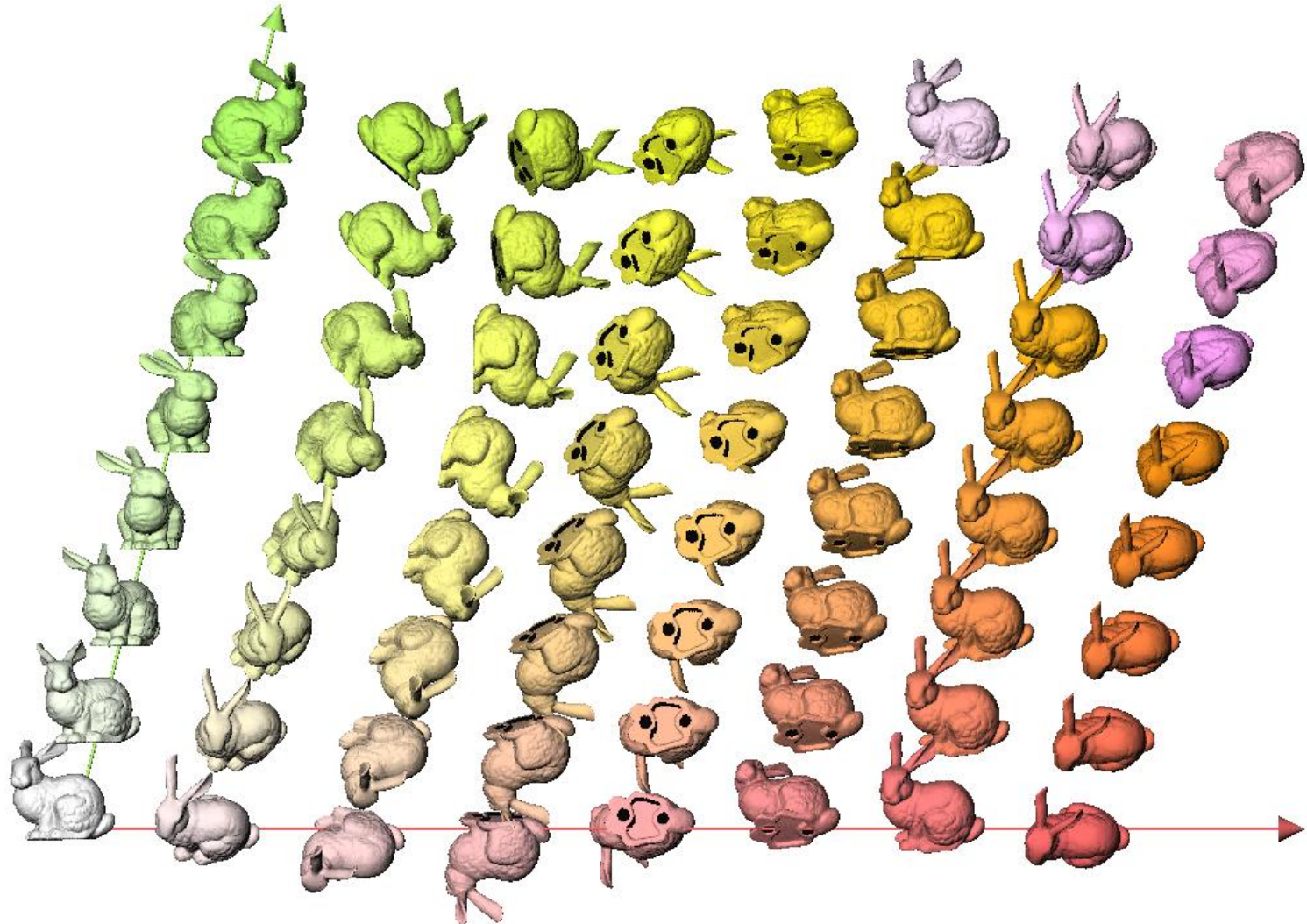
g. (scalar) · (vector) → (vector)
h. (scalar) · (point) → (point) if scalar=1
→ (vector) if scalar=0
→ (UNDEFINED) otherwise
j. \sum (scalar) · (vector) → (vector)
k. \sum (scalar) · (point) → (point) if \sum scalar=1
→ (vector) if \sum scalar=0
→ (UNDEFINED) otherwise

A. (orientation) · (orientation) → (UNDEFINED)
B. exp(rotation) · exp(rotation) → exp(rotation)
C. (orientation) · exp(rotation) → (orientation)
exp(rotation) · (orientation) → (orientation)
D. (orientation)⁻¹ · (orientation) → exp(rotation)
(orientation) · (orientation)⁻¹ → exp(rotation)
E. log(exp(rotation)) → (rotation)
F. log(orientation) → (UNDEFINED)
G. (scalar) · (rotation) → (rotation)
exp(rotation)^(scalar) → exp(rotation)
H. (orientation)^(scalar) → (orientation) if scalar=1
→ exp(rotation) if scalar=0
→ (UNDEFINED) otherwise
I. (rotation) ± (rotation) → (rotation)
J. \sum (scalar) · (rotation) → (rotation)
K. affine_combi(orientations) → (ILL-DEFINED)

Coordinate-Invariant Operations

- A. (orientation) · (orientation) → (UNDEFINED)
- B. exp(rotation) · exp(rotation) → exp(rotation)
- C. (orientation) · exp(rotation) → (orientation)
exp(rotation) · (orientation) → (orientation)
- D. (orientation)⁻¹ · (orientation) → exp(rotation)
(orientation) · (orientation)⁻¹ → exp(rotation)
- E. log(exp(rotation)) → (rotation)
- F. log(orientation) → (UNDEFINED)
- G. (scalar) · (rotation) → (rotation)
exp(rotation)^(scalar) → exp(rotation)
- H. (orientation)^(scalar) → (orientation) if scalar=1
→ exp(rotation) if scalar=0
→ (UNDEFINED) otherwise
- I. (rotation) ± (rotation) → (rotation)
- J. ∑ (scalar) · (rotation) → (rotation)
- K. affine_combi(orientations) → (ILL-DEFINED)

Linear Combination of Rotations



Marc Alexa, Linear combination of transformations, Siggraph 2002.

Coordinate-Invariant Operations

- A. $(\text{orientation}) \cdot (\text{orientation}) \rightarrow (\text{UNDEFINED})$
- B. $\text{exp}(\text{rotation}) \cdot \text{exp}(\text{rotation}) \rightarrow \text{exp}(\text{rotation})$
- C. $(\text{orientation}) \cdot \text{exp}(\text{rotation}) \rightarrow (\text{orientation})$
 $\text{exp}(\text{rotation}) \cdot (\text{orientation}) \rightarrow (\text{orientation})$
- D. $(\text{orientation})^{-1} \cdot (\text{orientation}) \rightarrow \text{exp}(\text{rotation})$
 $(\text{orientation}) \cdot (\text{orientation})^{-1} \rightarrow \text{exp}(\text{rotation})$
- E. $\log(\text{exp}(\text{rotation})) \rightarrow (\text{rotation})$
- F. $\log(\text{orientation}) \rightarrow (\text{UNDEFINED})$
- G. $(\text{scalar}) \cdot (\text{rotation}) \rightarrow (\text{rotation})$
 $\text{exp}(\text{rotation})^{(\text{scalar})} \rightarrow \text{exp}(\text{rotation})$
- H. $(\text{orientation})^{(\text{scalar})} \rightarrow (\text{orientation})$ if $\text{scalar}=1$
 $\rightarrow \text{exp}(\text{rotation})$ if $\text{scalar}=0$
 $\rightarrow (\text{UNDEFINED})$ otherwise
- I. $(\text{rotation}) \pm (\text{rotation}) \rightarrow (\text{rotation})$
- J. $\sum (\text{scalar}) \cdot (\text{rotation}) \rightarrow (\text{rotation})$
- K. $\text{affine_combi}(\text{orientations}) \rightarrow (\text{ILL-DEFINED})$

Affine Combination of Orientations

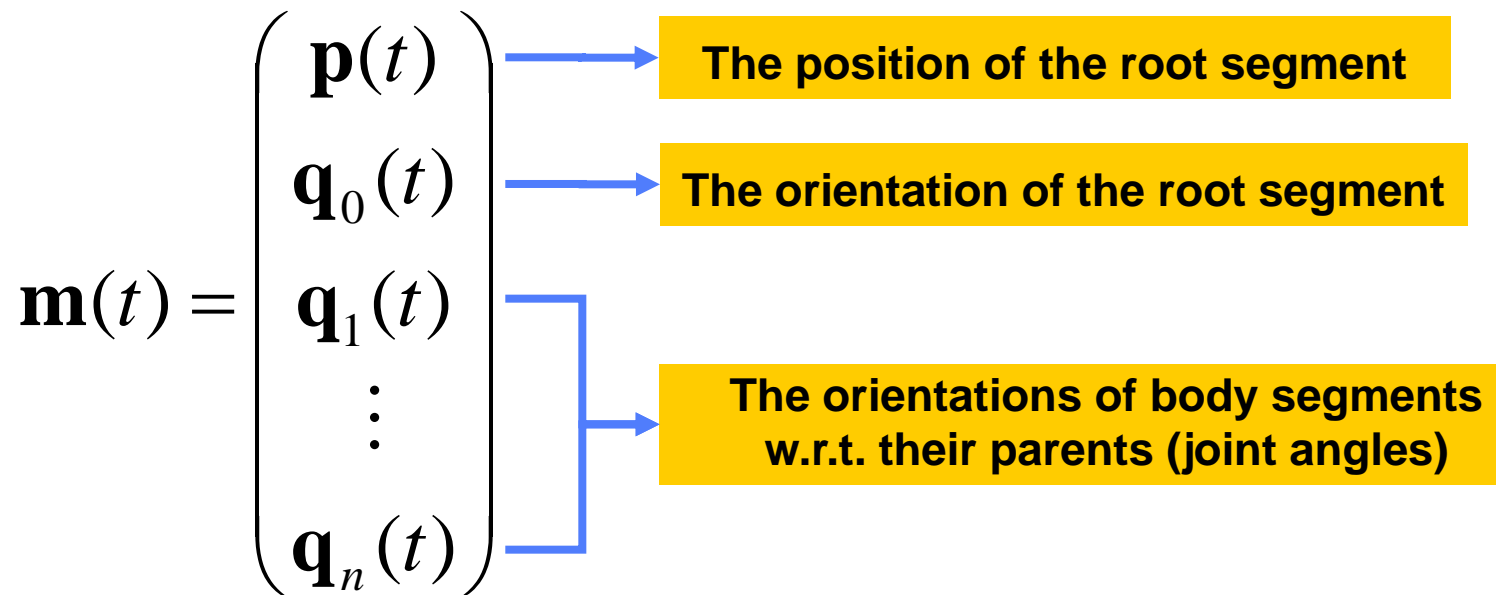


Course Overview

- **Introduction and Overview** (5 min)
- **Coordinate-Invariant Geometric Programming** (20 min)
- **Programming with Orientations and Rotations** (35 min)
- **Programming with Motion Capture Data** (10 min)
 - Representing motion data and motion displacements
 - Coordinate-invariant operations for motion data
- **Practical examples** (30 min)

Motion Representation

- Configuration of an articulated figure
 - Linear components: $\mathbf{p}(t) \in \mathbf{R}^3$
 - Angular components: $\mathbf{q}_i(t) \in \mathbf{S}^3$



Element-wise Operations

- Translation

$$\mathbf{m}'(t) = \mathbf{m}(t) \oplus \text{exp}(\mathbf{d}^t) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \mathbf{q}_2(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \begin{pmatrix} \mathbf{v} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{p}(t) + \mathbf{v} \\ \mathbf{q}_1(t) \\ \mathbf{q}_2(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix}$$

- Rotation

$$\mathbf{m}'(t) = \mathbf{m}(t) \oplus \text{exp}(\mathbf{d}^r) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \mathbf{q}_2(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \begin{pmatrix} \mathbf{0} \\ \text{exp}(\mathbf{v}) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \text{exp}(\mathbf{v}) \\ \mathbf{q}_2(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix}$$

Element-wise Operations

- Exercise joints

$$\mathbf{m}'(t) = \mathbf{m}(t) \oplus \text{exp}(\mathbf{d}^j) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_i(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \text{exp}(\mathbf{v}) \\ \vdots \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_i(t) \text{exp}(\mathbf{v}) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix}$$

Motion Displacement

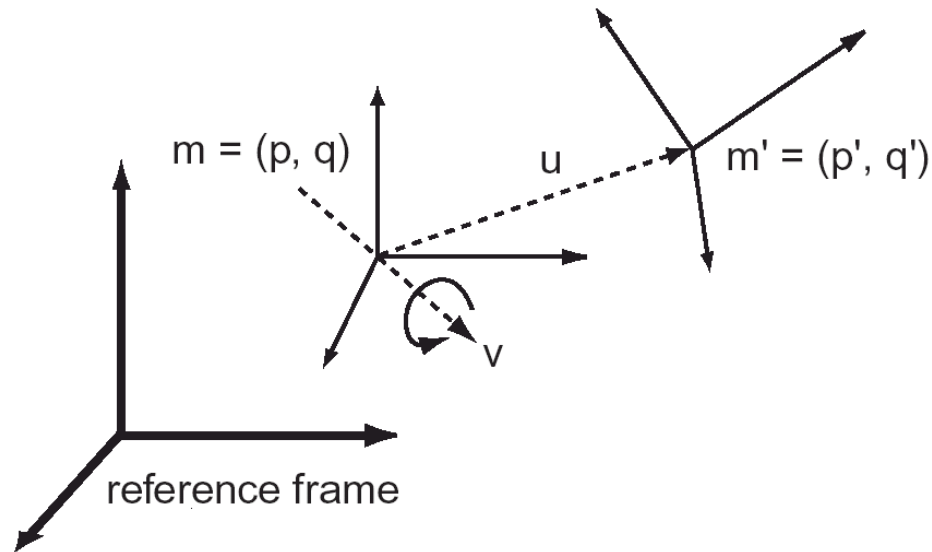
- Independent translation and rotation for root

$$\begin{aligned}\mathbf{m}'(t) &= \mathbf{m}(t) \oplus \tilde{\text{exp}}(\mathbf{d}(t)) \\ &= \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \tilde{\text{exp}} \begin{pmatrix} \mathbf{v}_0(t) \\ \mathbf{v}_1(t) \\ \vdots \\ \mathbf{v}_n(t) \end{pmatrix} = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \begin{pmatrix} \mathbf{v}_0(t) \\ \text{exp}(\mathbf{v}_1(t)) \\ \vdots \\ \text{exp}(\mathbf{v}_n(t)) \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{p}(t) + \mathbf{v}_0(t) \\ \mathbf{q}_1(t) \text{exp}(\mathbf{v}_1(t)) \\ \vdots \\ \mathbf{q}_n(t) \text{exp}(\mathbf{v}_n(t)) \end{pmatrix}\end{aligned}$$

Motion Displacement

- Rigid transformation at root

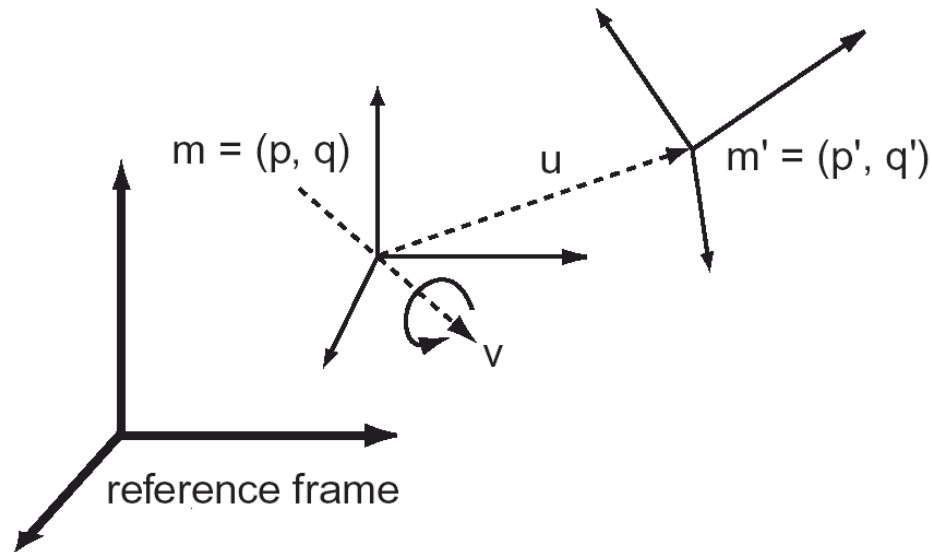
$$\mathbf{m}'(t) = \mathbf{m}(t) \oplus \tilde{\text{exp}}(\mathbf{d}(t)) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \begin{pmatrix} \mathbf{v}_0(t) \\ \exp(\mathbf{v}_1(t)) \\ \vdots \\ \exp(\mathbf{v}_n(t)) \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1 \mathbf{v}_0 \mathbf{q}_1^{-1} + \mathbf{p} \\ \mathbf{q}_1 \exp(\mathbf{v}_1) \\ \vdots \\ \mathbf{q}_n \exp(\mathbf{v}_n) \end{pmatrix}$$



Motion Displacement

- Rigid Transformation at root

$$\exp(\mathbf{d}(t)) = \mathbf{m}'(t) \ominus \mathbf{m}(t) = \begin{pmatrix} \mathbf{p}'(t) \\ \mathbf{q}'_1(t) \\ \vdots \\ \mathbf{q}'_n(t) \end{pmatrix} \ominus \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1^{-1}(\mathbf{p}'(t) - \mathbf{p}(t))\mathbf{q}_1 \\ \mathbf{q}_1^{-1}\mathbf{q}'_1 \\ \vdots \\ \mathbf{q}_n^{-1}\mathbf{q}'_n \end{pmatrix}$$



Operations

- Valid operations

$$\mathbf{m}_1(t) \oplus \text{e}\tilde{\text{x}}\text{p}(\mathbf{d}(t)) = \mathbf{m}_2(t)$$

$$\mathbf{m}_2(t) \ominus \mathbf{m}_1(t) = \text{e}\tilde{\text{x}}\text{p}(\mathbf{d}(t))$$

$$\alpha \mathbf{d}_1(t) = \mathbf{d}_2(t)$$

$$\mathbf{d}_1(t) + \mathbf{d}_2(t) = \mathbf{d}_3(t) \quad (\text{Be careful !})$$

- Invalid operations

$$\mathbf{m}_1(t) \oplus \mathbf{m}_2(t) = ?$$

$$\alpha \mathbf{m}(t) = ?$$

Operations

- Time warping

$$\mathbf{m}'(t) = \mathbf{m}(s(t))$$

- Properties

$$\mathbf{m}(t) \oplus \mathbf{d}(t) \neq \mathbf{d}(t) \oplus \mathbf{m}(t)$$

$$(\mathbf{m}(t) \oplus \mathbf{d}_1(t)) \oplus \mathbf{d}_2(t) \neq \mathbf{m}(t) \oplus (\mathbf{d}_1(t) + \mathbf{d}_2(t))$$

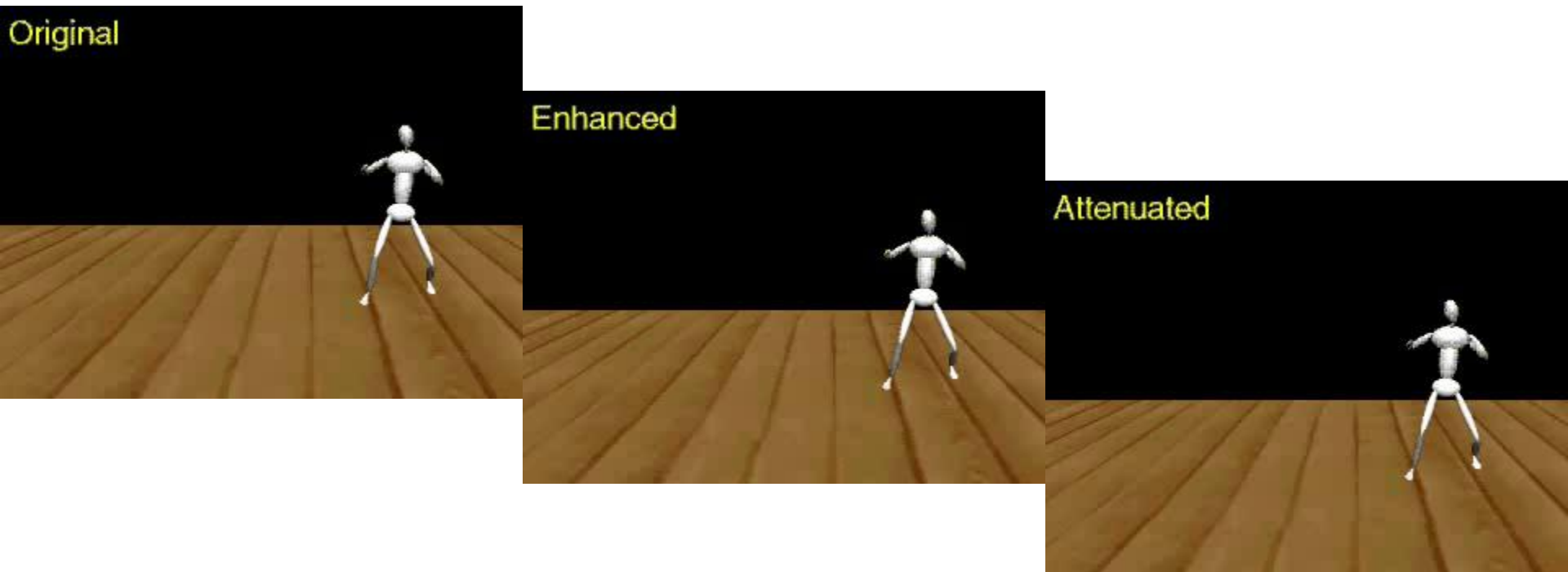
Coordinate-Invariant Operations

- A. $(\text{pose}) \otimes (\text{pose}) \rightarrow (\text{UNDEFINED})$
- B. $\exp(\text{disp}) \otimes \exp(\text{disp}) \rightarrow \exp(\text{disp})$
- C. $(\text{pose}) \otimes \exp(\text{disp}) \rightarrow (\text{pose})$
- D. $(\text{pose}) \oslash (\text{pose}) \rightarrow \exp(\text{disp})$
- E. $\log(\exp(\text{disp})) \rightarrow (\text{disp})$
- F. $\log(\text{pose}) \rightarrow (\text{UNDEFINED})$
- G. $(\text{scalar}) \cdot (\text{disp}) \rightarrow (\text{disp})$
 $\exp(\text{disp})^{(\text{scalar})} \rightarrow \exp(\text{disp})$
- H. $(\text{pose})^{(\text{scalar})} \rightarrow (\text{pose})$ if scalar=1
 $\rightarrow \exp(\text{disp})$ if scalar=0
 $\rightarrow (\text{UNDEFINED})$ otherwise
- I. $(\text{disp}) \pm (\text{disp}) \rightarrow (\text{disp})$
- J. $\sum (\text{scalar}) \cdot (\text{disp}) \rightarrow (\text{disp})$
- K. $\text{affine_combination}(\text{poses}) \rightarrow (\text{ILL-DEFINED})$

Course Overview

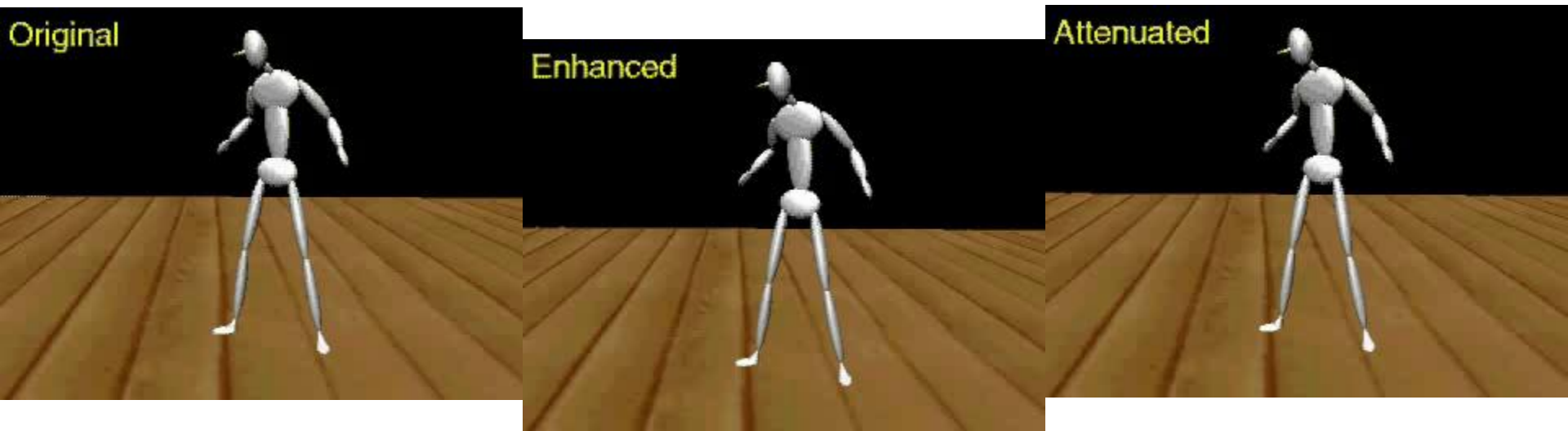
- **Introduction and Overview** (5 min)
- **Coordinate-Invariant Geometric Programming** (20 min)
- **Programming with Orientations and Rotations** (35 min)
- **Programming with Motion Capture Data** (10 min)
- **Practical examples** (30 min)
 - Motion exaggeration and style transfer
 - Aligning and warping
 - Interpolation and transitioning

Motion Exaggeration



Jehee Lee and Sung Yong Shin, A Coordinate-Invariant Approach to Multiresolution Motion Analysis and Synthesis, Graphical Models, 2001.

Motion Exaggeration

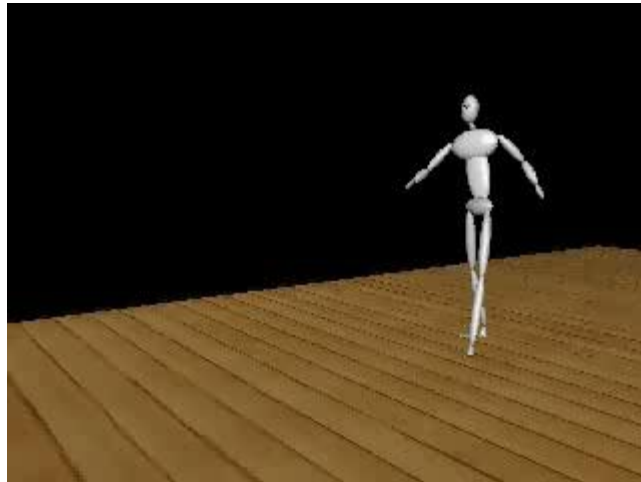
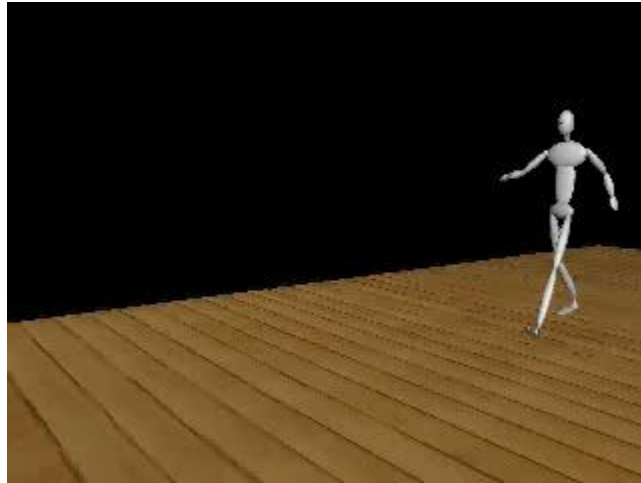


$$\mathbf{M}'(t) = \mathbf{L}(t) \oplus (\mathbf{M}(t) \ominus \mathbf{L}(t))^{\alpha}$$

$\mathbf{M}(t)$: input data

$\mathbf{L}(t)$: simplified and lowpass filtered

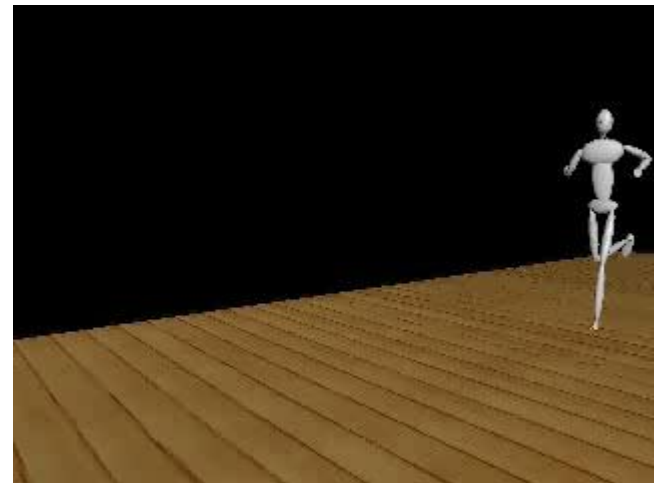
Walk



Strut

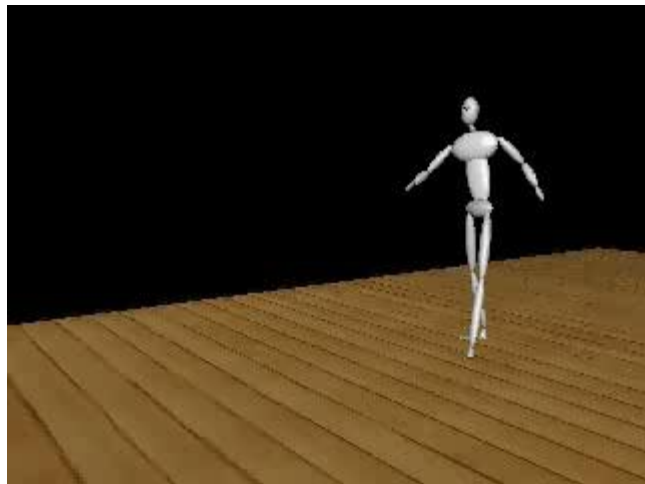
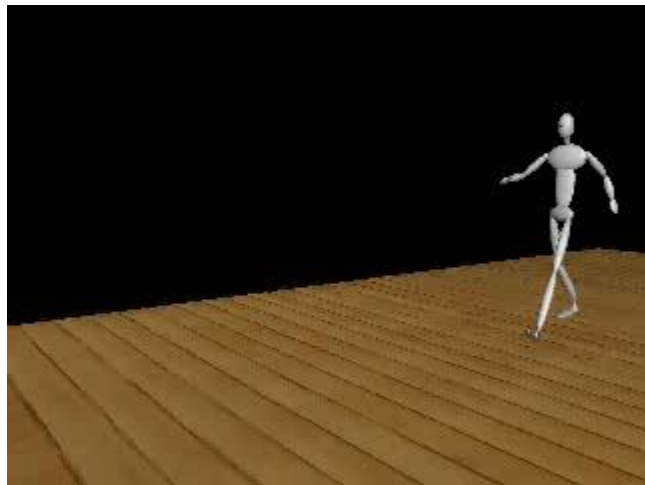


Run



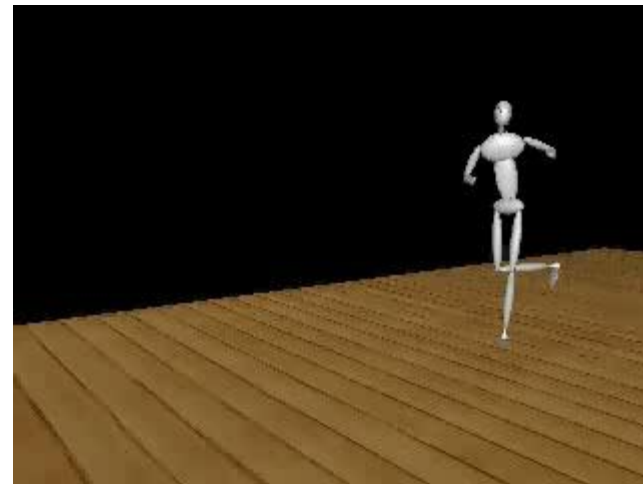
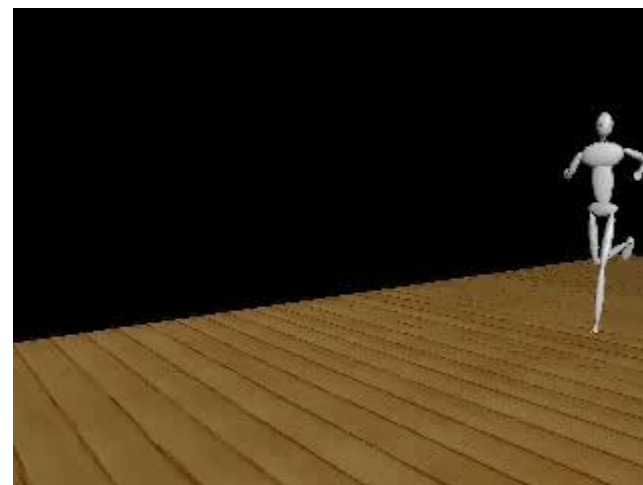
Run in a pompous manner

Walk

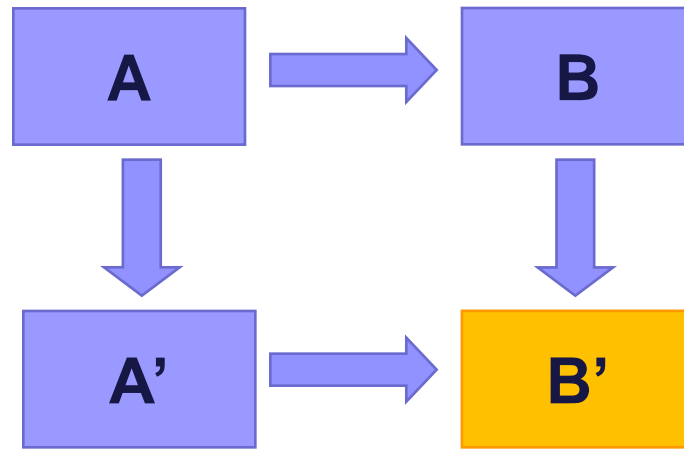


Strut

Run



Style Transfer



$$\mathbf{B}'(t) = \mathbf{B}(t) \oplus (\mathbf{A}(t) \ominus \mathbf{A}'(t))$$

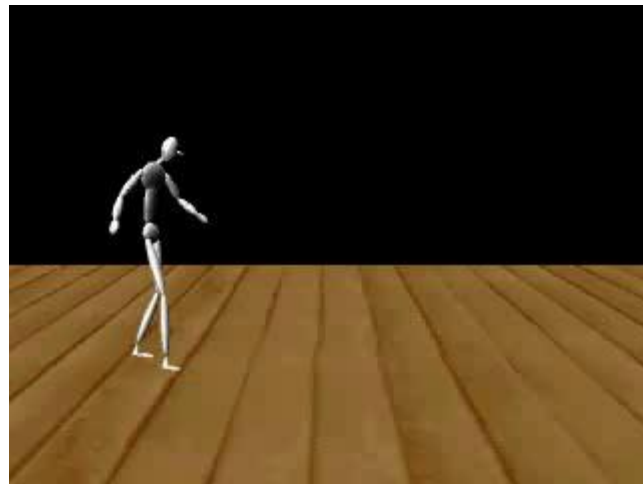
Walk



Turn



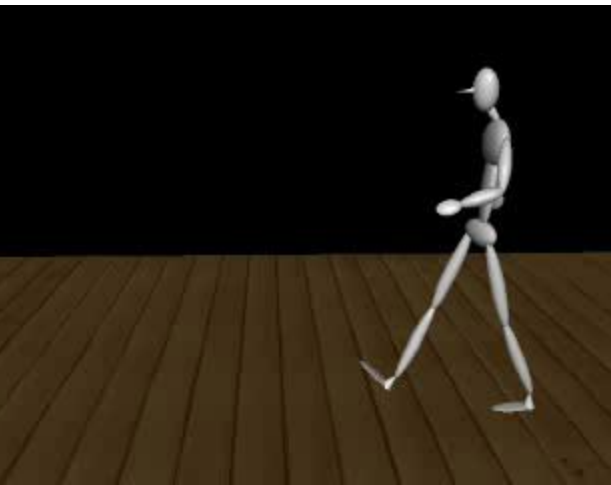
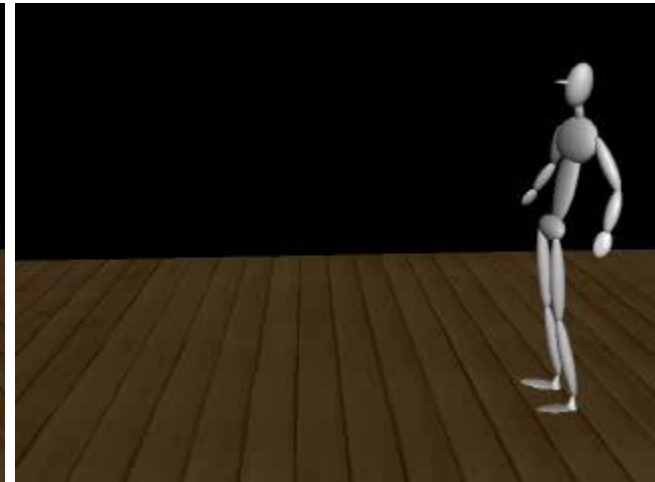
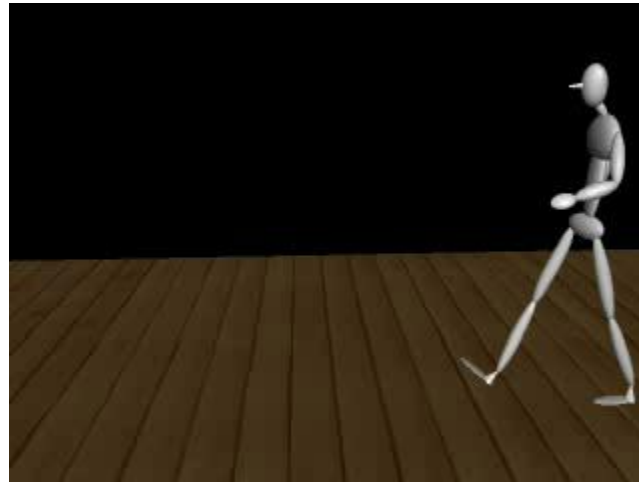
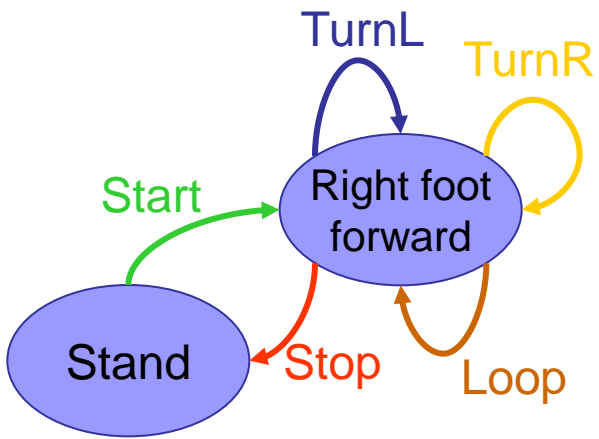
Limp



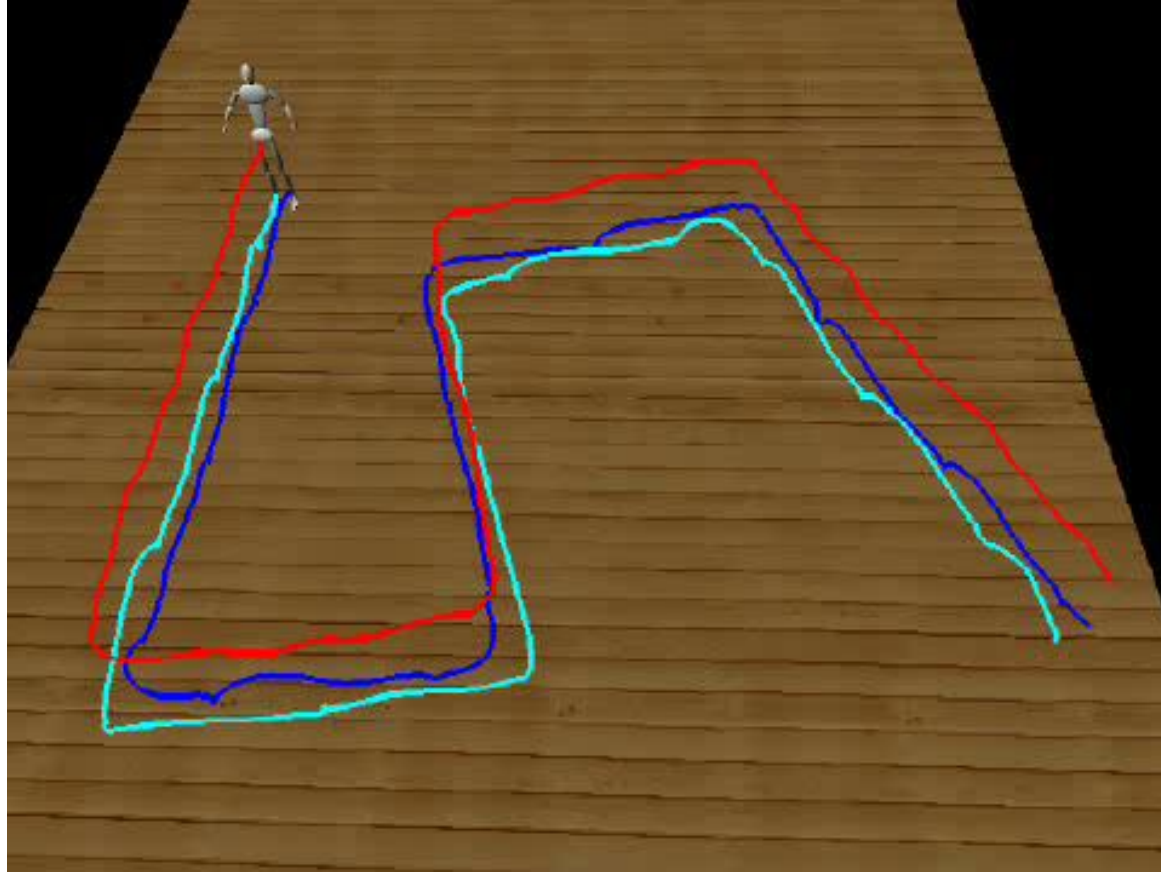
Turn with a limp



Transition Graph

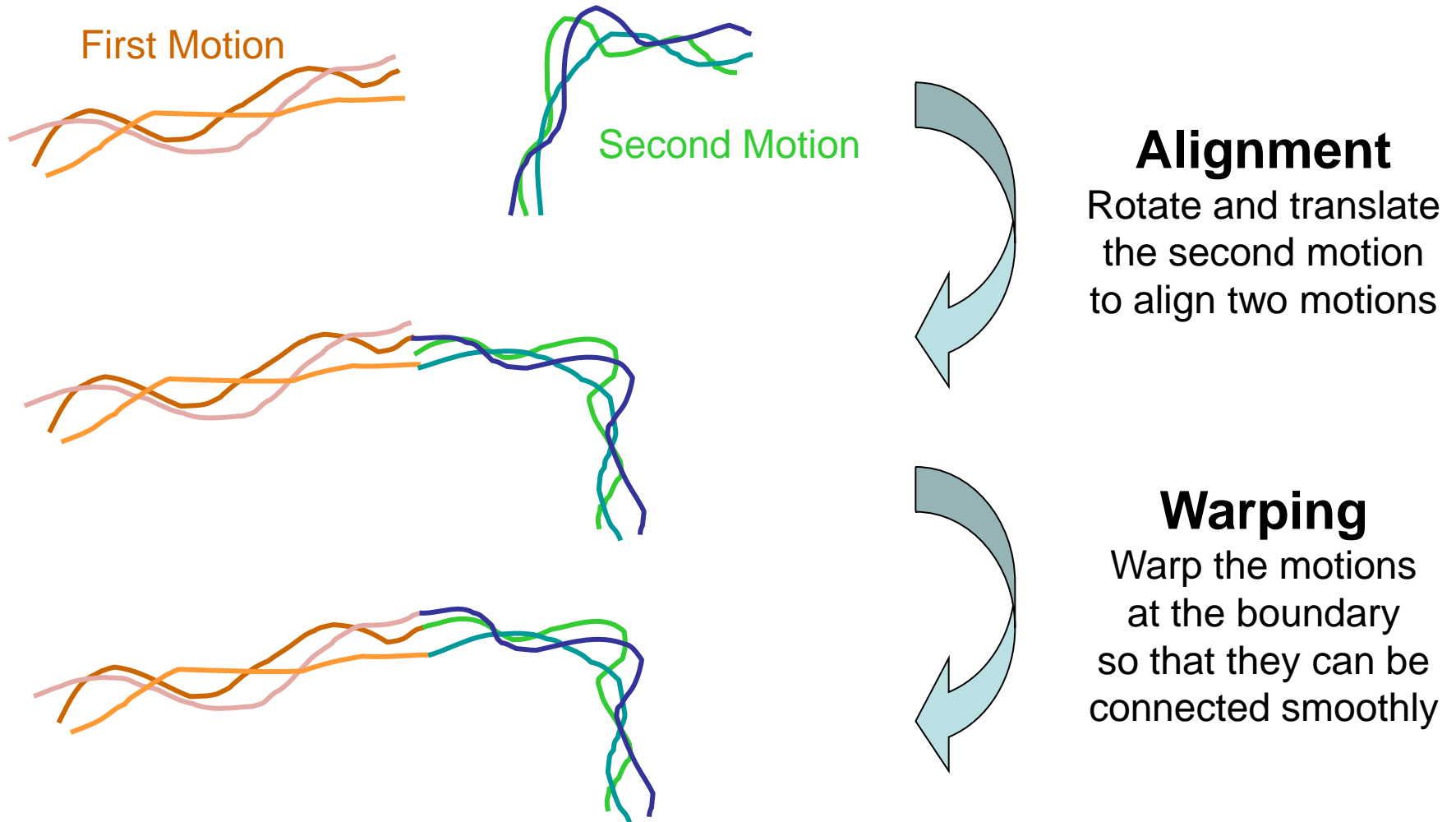


Transition Graph



Jehee Lee, Jinxiang Chai, Paul Reitsma, Jessica Hodgins, Nancy Pollard,
Interactive Control of Avatars Animated with Human Motion Data,
SIGGRAPH 2002

Connecting Motion Segments



Alignment

- The end of one motion A should be aligned to the beginning of the next motion B

- The root location of the end of A :

$$(\mathbf{p}_n^A, \mathbf{q}_n^A)$$

- The root location of the beginning of B :

$$(\mathbf{p}_0^B, \mathbf{q}_0^B)$$

- Apply $\tilde{\text{exp}}\begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{p}_n^A \\ \mathbf{q}_n^A \end{pmatrix} \ominus \begin{pmatrix} \mathbf{p}_0^B \\ \mathbf{q}_0^B \end{pmatrix}$ to motion B , that is

$$\mathbf{m}^B(t) \oplus \tilde{\text{exp}}(\mathbf{d}) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \mathbf{q}_2(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \begin{pmatrix} \mathbf{u} \\ \text{exp}(\mathbf{v}) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1(t)\mathbf{u}\mathbf{q}_1^{-1}(t) + \mathbf{p}(t) \\ \mathbf{q}_1(t)\text{exp}(\mathbf{v}) \\ \mathbf{q}_2(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix}$$

Alignment Using In-Plane Transformation

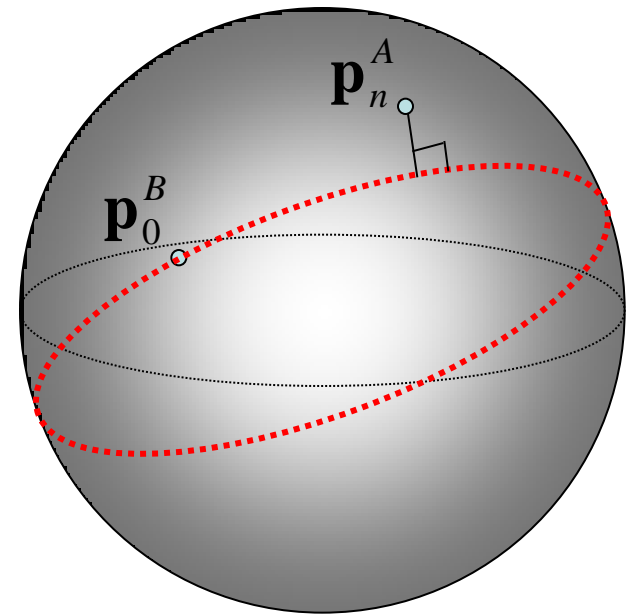
- Rigid transformation restricted within a plane
 - Rotation about the vertical axis, followed by
 - Translation along two horizontal axes
- Finding in-plane rotation close to given rotation
 - Using Euler angles
 - Discard rotation about x- and z-axes
 - Not optimal

$$R = R_x R_y R_z$$

Optimal In-Plane Transformation

- The **geodesic curve** represents a set of orientation that can be reached by rotating about a fixed axis from any orientation on the curve

$$\begin{aligned} G(\hat{y}, \mathbf{p}_0^B) &= \exp(\theta \hat{y}) \cdot \mathbf{p}_0^B \\ &= (\cos \theta, \hat{y} \sin \theta) \cdot \mathbf{p}_0^B \end{aligned}$$



Optimal In-Plane Transformation

Given a quaternion point $q_s = (w_s, v_s)$ and

a geodesic curve $G(\theta) = (\cos \theta, u \sin \theta)(w_0, v_0)$,

the closest point $\bar{q} \in G$ from q_s is:

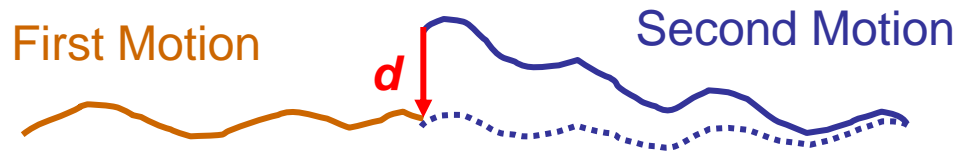
$$\bar{q} = \begin{cases} G(-\alpha + \frac{\pi}{2}) & \text{if } q_s \cdot G(-\alpha + \frac{\pi}{2}) > q_s \cdot G(-\alpha - \frac{\pi}{2}), \\ G(-\alpha - \frac{\pi}{2}) & \text{if } q_s \cdot G(-\alpha + \frac{\pi}{2}) < q_s \cdot G(-\alpha - \frac{\pi}{2}), \end{cases}$$

where $a = w_s w_0 + v_s \cdot v_0$, $b = w_s(u \cdot v_0) + w_0(v_s \cdot u) + v_s \cdot (u \times v_0)$,

and $\tan \alpha = \frac{a}{b}$.

Warping

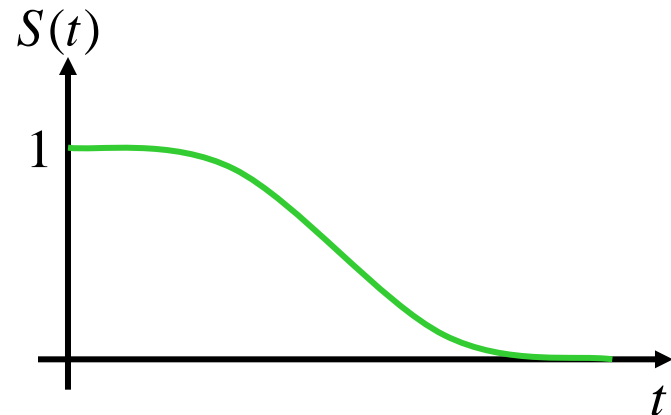
- Deform a motion smoothly so that it seamlessly connects to its previous motion



$$\mathbf{d} = \mathbf{m}_n^A \ominus \mathbf{m}_0^B$$

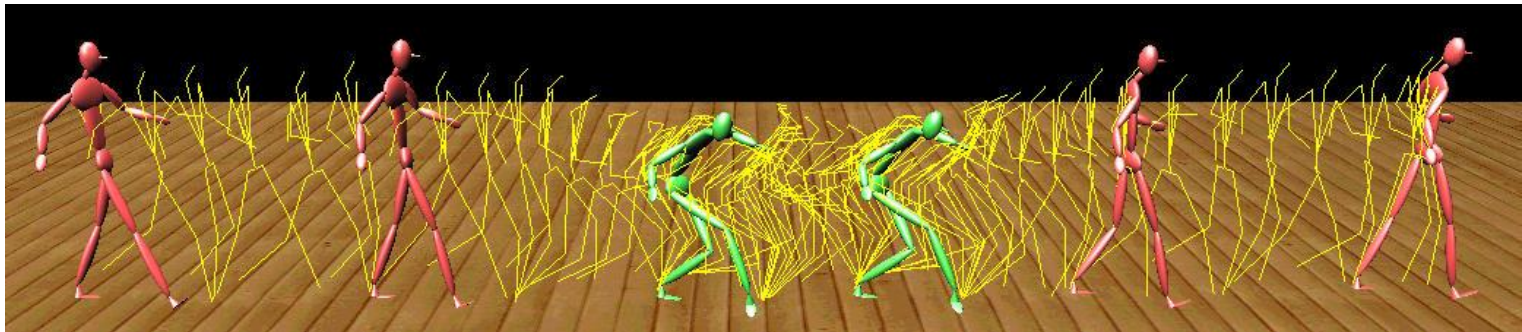
- A scalar transition function $s(t)$

$$\mathbf{m}^B(t) \oplus \{s(t) \cdot \mathbf{d}\}$$



Blending for Smooth Transition

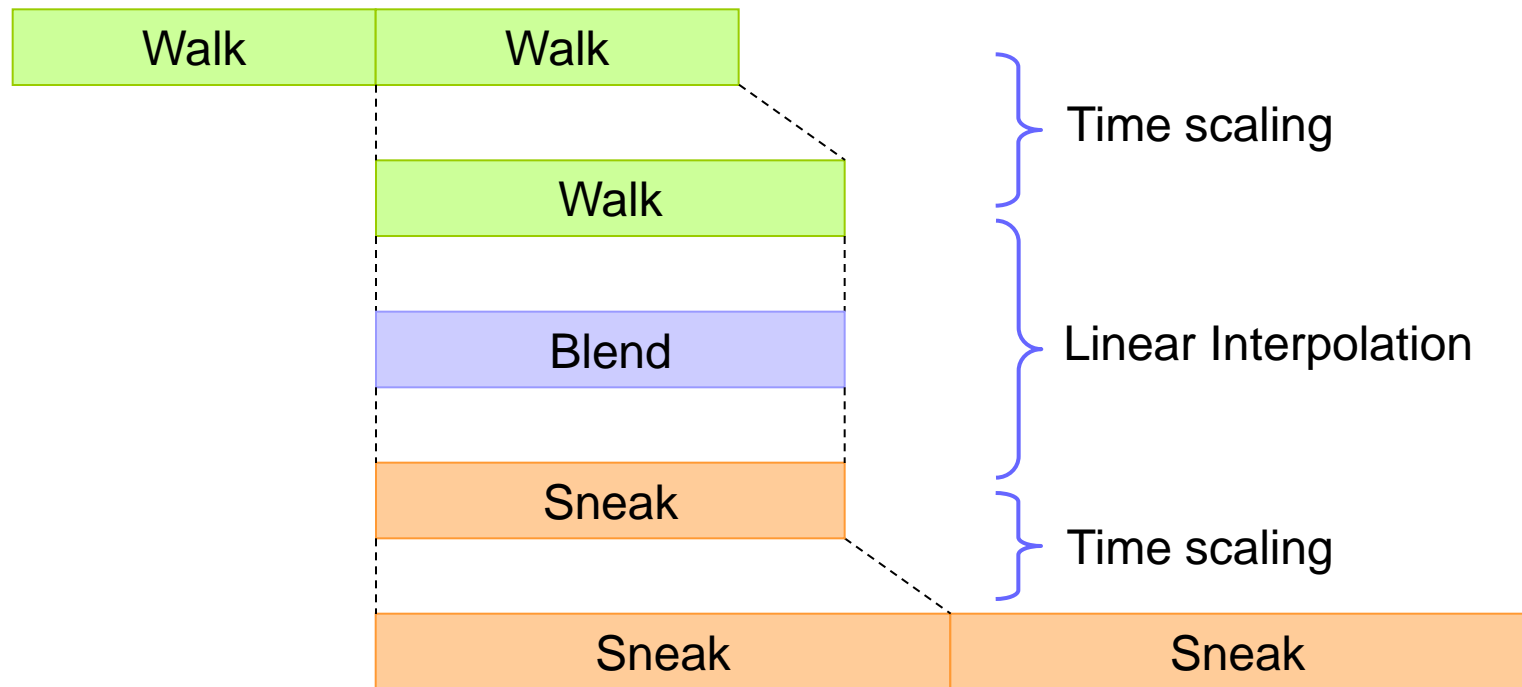
- Case study: walk-to-sneak
 - Transit smoothly over one cycle of locomotion
 - “Walk” is faster than “sneak”
 - One cycle of “sneak” is longer than one cycle of “walk”



Jehee Lee and Sung Yong Shin, A Hierarchical Approach to Interactive Motion Editing for Human-like Characters, SIGGRAPH 99

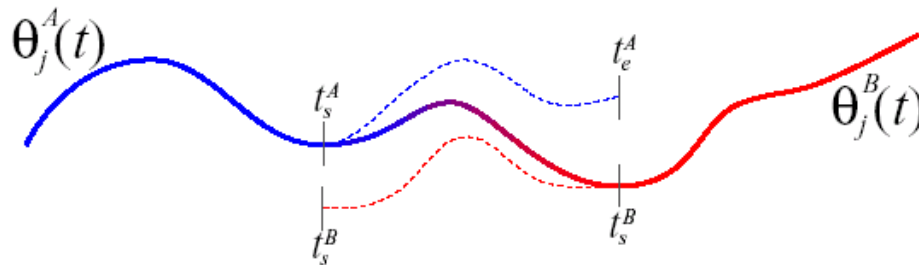
Blending for Smooth Transition

- Blend over the overlapping time interval

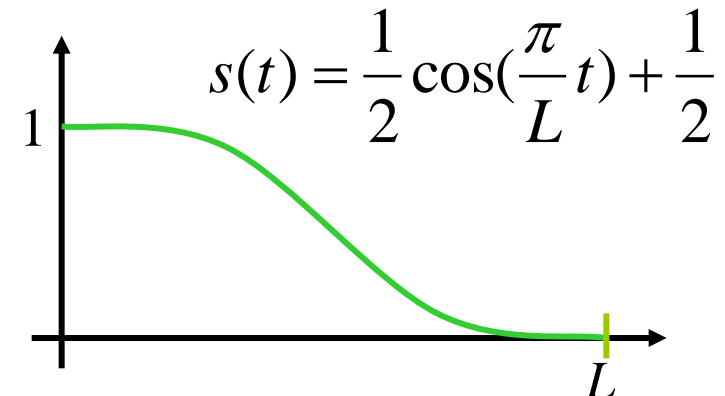


Blending for Smooth Transition

- Linear interpolation between motions
 - Slerp for orientation components
 - A scalar transition function $\mathbf{s}(t)$

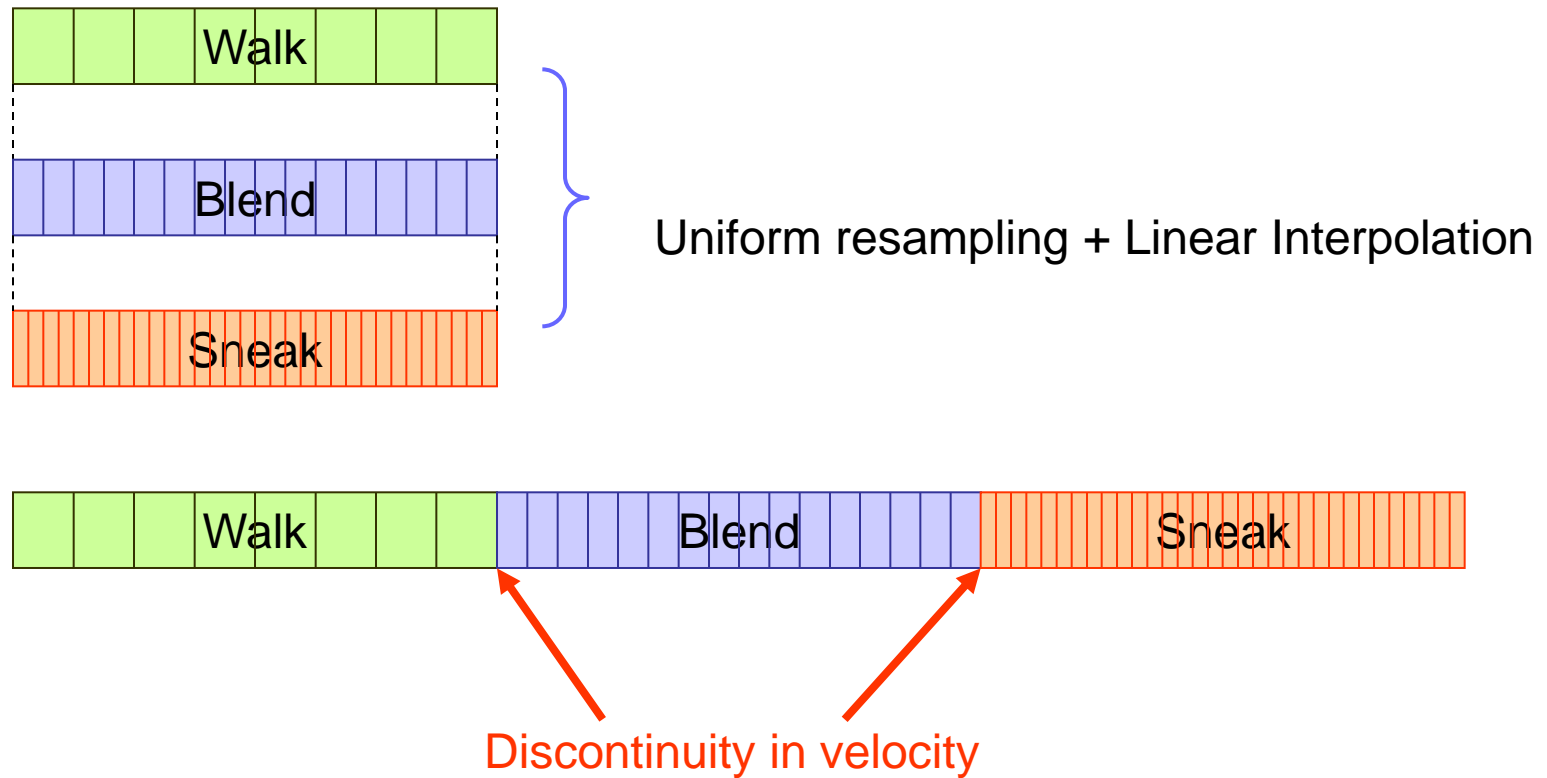


$$s(t) \cdot \mathbf{m}^A(t) \oplus (1 - s(t)) \cdot \mathbf{m}^B(t)$$



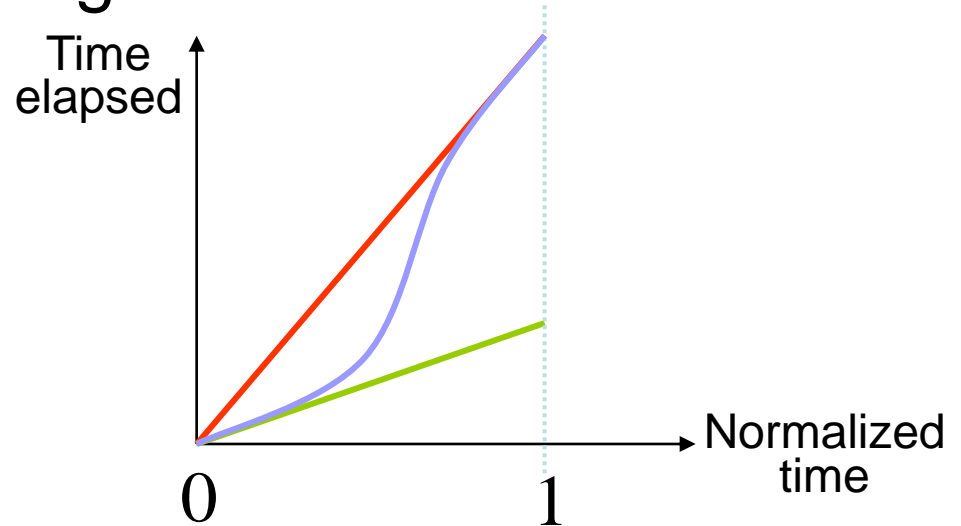
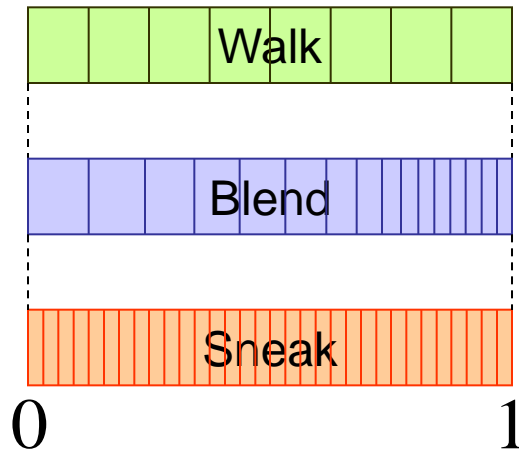
Time Scaling

- Discontinuity in velocity



Time Scaling

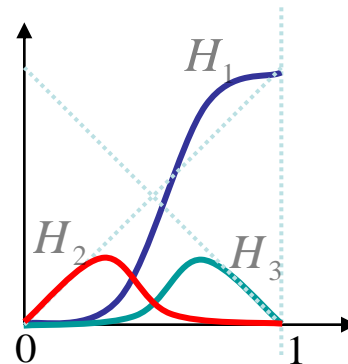
- Non-uniform resampling



$$T(t) = H_1(t) + H_2(t)f_1 + H_3(t)f_2$$

$$T(0) = 0, \quad T(1) = 1,$$

$$T'(0) = f_1, \quad T'(1) = f_2$$



$$H_1(t) = -2t^3 + 3t^2$$

$$H_2(t) = t^3 - 2t^2 + t$$

$$H_3(t) = t^3 - t^2$$

Wrap Up

- **Coordinate-Invariance** does matter
- Distinguish **orientations** from **rotations** as we do **points** from **vectors**
- The algebraic structure of motion data is similar to the structure of orientation/rotation data