Introduction to Data-Driven Animation: Programming with Motion Capture

Jehee Lee
Seoul National University
(Organizer & Presenter)

Course Abstract

Data-driven animation using motion capture data has become a standard practice in character animation. A number of techniques have been developed to add flexibility on captured human motion data by editing joint trajectories, warping motion paths, blending a family of parameterized motions, splicing motion segments, and adapting motion to new characters and environments. Even with the abundance of motion capture data and the popularity of data-driven animation techniques, programming with motion capture data is still not easy. A single clip of motion data encompasses a lot of heterogeneous information including joint angles, the position and orientation of the skeletal root, their temporal trajectories, and a number of coordinate systems. Due to this complexity, even simple operations on motion data, such as linear interpolation, are rarely described as succinct mathematical equations in articles. This course provides not only a solid mathematical background but also a practical guide to programming with motion capture data. The course will begin with the brief review of affine geometry and coordinate-invariant (conventionally called coordinate-free) geometric programming, which will generalize incrementally to deal with three-dimensional rotations/orientations, the poses of an articulated figure, and full-body motion data. It will lead to identifying a collection of coordinate-invariant operations on full-body motion data and their object-oriented implementation. Finally, we will discuss the practical use of our programming framework in a variety of contexts ranging from data-driven manipulation/interpolation to state-of-the-art biped locomotion control.

About the Presenter

Jehee Lee
Associate Professor
School of Computer Science
Seoul National University
599 Gwanak-ro, Gwanak-gu
Seoul 151-742, Korea
(02) 880-1845
(02) 871-4912 Fax
jehee@cse.snu.ac.kr
http://mrl.snu.ac.kr/~jehee

Jehee Lee is an associate professor of Computer Science at Seoul National University. He received his B.S, M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology in 1993, 1995, and 2000, respectively. He is leading the SNU Movement Research Laboratory. His research interests are in the areas of computer graphics and animation. More specifically, he is interested in developing new ways of understanding, representing, and animating human movements. This involves full-body motion analysis and synthesis, biped control and simulation, motion capture, motion planning, data-driven and physically based techniques, interactive avatar control, crowd simulation, and facial animation.

Course Schedule and Syllabus

Introduction and Overview (5 minutes)

1. Data-driven animation using motion capture data
2. Why is it difficult to do programming with motion capture data?
3. Course overview

Coordinate-Invariant Programming with Points and Vectors (20 minutes)

1. What is coordinate-invariant geometric programming?
2. Affine geometry
3. Coordinate-invariant operations between points and vectors

Programming with Orientations and Rotations (35 minutes)

1. Representing orientations and rotations
2. Analogy between points/vectors and orientations/rotations
3. Coordinate-invariant operations with orientations and rotations

Programming with Motion Capture Data (10 minutes)

1. Representing motion data and motion displacements
2. Coordinate-invariant operations for motion data

Practical examples (30 minutes)

1. Motion exaggeration and style transfer
2. Hierarchical displacement mapping
3. Interpolation and transitioning

References

Jehee Lee. 2008. Representing rotations and orientations in geometric computing. IEEE Computer Graphics and Applications 28, 2, 75–83.

Yoonsang Lee, Sungeun Kim, Jehee Lee, Data-Driven Biped Control, ACM Transactions on Graphics (SIGGRAPH 2010), Vol. 29, No. 4, Article 129, July 2010

Jehee Lee, Jinxiang Chai, Paul Reitsma, Jessica Hodgins and Nancy Pollard. 2002. Interactive Control of Avatars Animated with Human Motion Data, ACM Transactions on Graphics (SIGGRAPH 2002), volume 21, number 3, 491-500.

Hyun Joon Shin, Jehee Lee, Michael Gleicher, and Sung Yong Shin, Computer Puppetry: An Importance-based Approach, ACM Transactions on Graphics, volume 20, number 2, 67-94, April 2001.

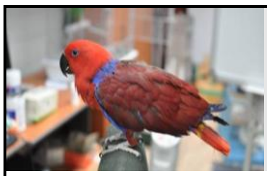
Jehee Lee and Sung Yong Shin. 2002. General Construction of Time-Domain Filters for Orientation Data, IEEE Transactions on Visualization and Computer Graphics, volume 8, number 2, 119-128.

Jehee Lee and Sung Yong Shin. 2001. A Coordinate-Invariant Approach to Multiresolution Motion Analysis, Graphical Models, volume 63, number 2, 87-105.

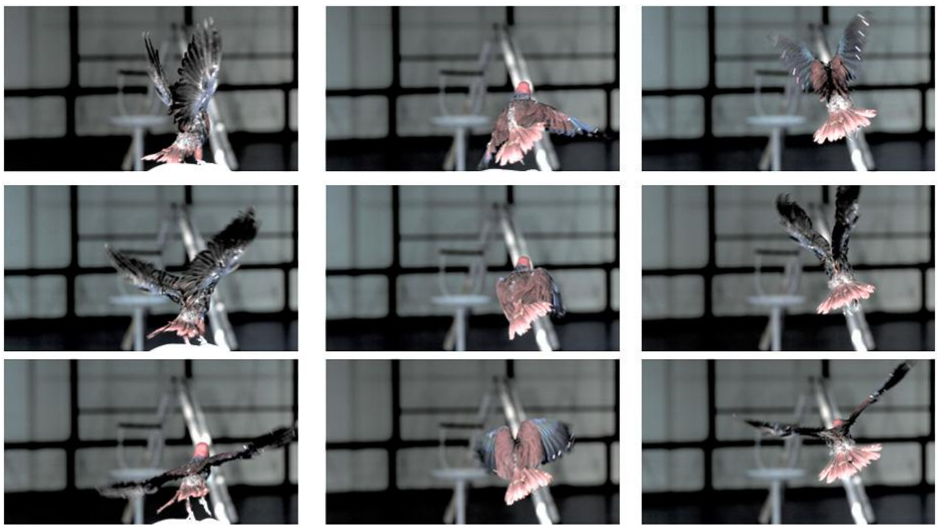
Jehee Lee and Sung Yong Shin. 1999. A hierarchical approach to interactive motion editing for human-like figures. In Proceedings of SIGGRAPH 99, 39–48.

Introduction to Data-Driven Animation: Programming with Motion Capture

Jehee Lee
Seoul National University

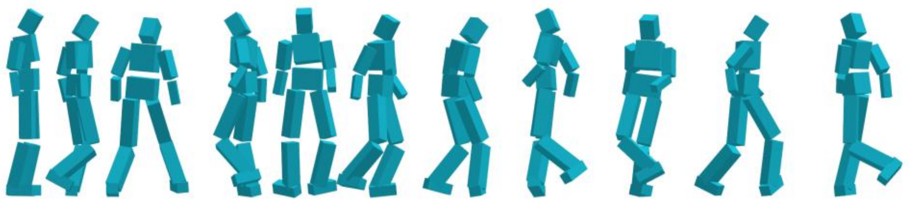


Data-Driven Animation with Motion Capture



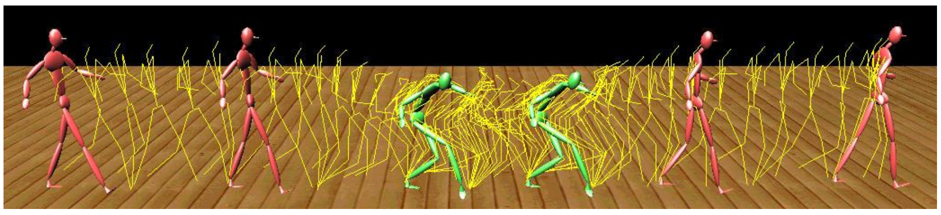
Programming with Motion Capture

- 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



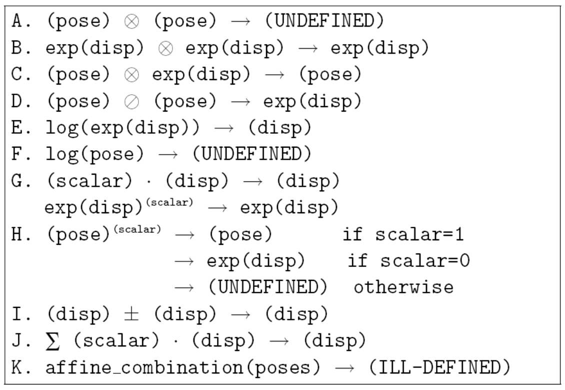
Mathematical Notation

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



Course Objectives

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



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

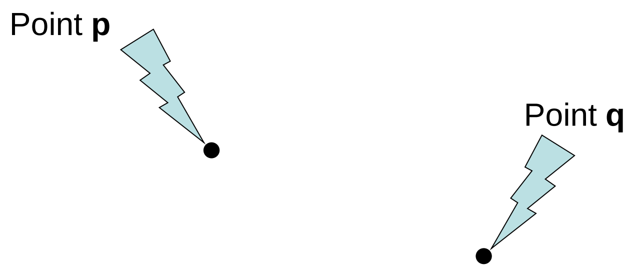
Course Overview

- **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

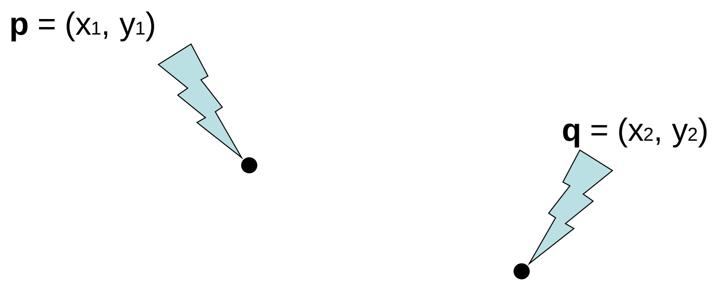
- A way of handling geometric entities such as vectors, points, and transforms.
- Write geometric programs relying on geometric reasoning rather than coordinate manipulation
- Pioneered by Goldman and DeRose
  - Geometric programming : A coordinate-free approach, SIGGRAPH 1988 Course #25 Notes
- Coodinate-Invariant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

Example of coordinate-dependence



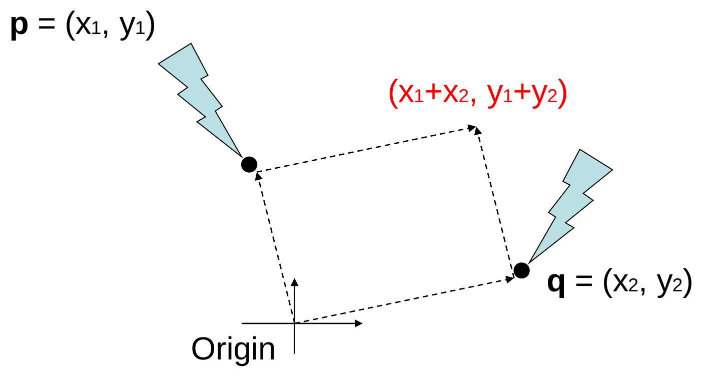
- What is the “sum” of these two positions ?

If you assume coordinates, …



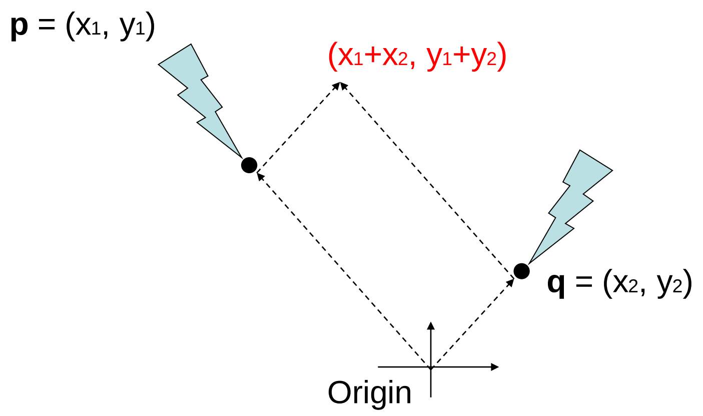
- The sum is (x_1+x_2, y_1+y_2)
  - Is it correct ?
  - Is it geometrically meaningful ?

If you assume coordinates, …



- Vector sum
  - (x_1, y_1) and (x_2, y_2) are considered as vectors from the origin to **p** and **q**, respectively.

If you select a different origin, …

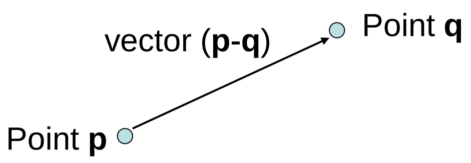


- If you choose a different coordinate frame, you will get a different result

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

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 spaces

- A ***vector space*** consists of
  - Set of vectors, together with
  - Two operations: addition of vectors and multiplication of vectors by scalar numbers
- A ***linear combination*** of vectors is also a vector

$$\mathbf{u},\mathbf{v},\mathbf{w}\in V \quad\Rightarrow\quad \alpha\mathbf{u}+\beta\mathbf{v}+\gamma\mathbf{w}\in V$$

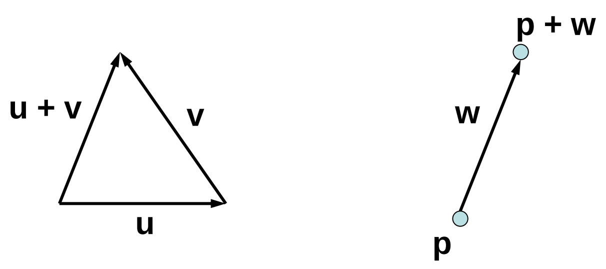
Affine Spaces

- An ***affine space*** consists of
  - Set of points, an associated vector space, and
  - Two operations: the difference between two points and the addition of a vector to a point

Coordinate-Free Geometric Operations

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

Addition

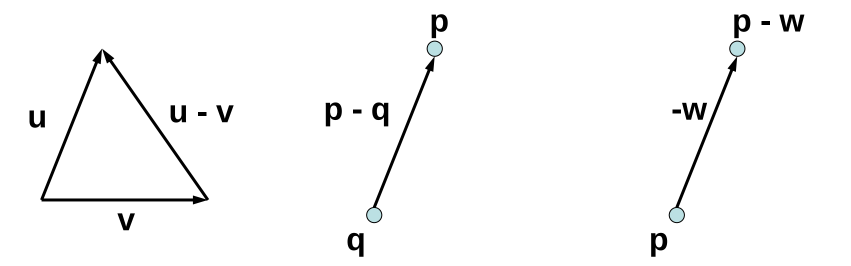


u + **v** is a vector

p + **w** is a point

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 • point = (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

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

$$= (\sum_{i=0}^{N} c_i)\mathbf{p}_0 + \sum_{i=1}^{N} c_i (\mathbf{p}_i - \mathbf{p}_0)$$

$\Sigma c_k \boldsymbol{P}_k = \boldsymbol{P}_0 + \Sigma c_k (\boldsymbol{P}_k - \boldsymbol{P}_0)$	$\Sigma c_k = 1$ (point)
$= \Sigma c_k (\boldsymbol{P}_k - \boldsymbol{P}_0)$	$\Sigma c_k = 0$ (vector)
= undefined	$\Sigma c_k \neq 0, 1$

Examples

- (**p** + **q**) / 2 : midpoint of line **pq**
- (**p** + **q**) / 3 : not valid
- (**p** + **q** + **r**) / 3 : center of gravity of Δ**pqr**
- (**p**/2 + **q**/2 – **r**) : a vector from **r** to the midpoint of **q** and **p**

Summary

1. point + point = undefined
2. point − point = vector
3. point ± vector = point
4. vector ± vector = vector
5. scalar • vector = vector
6. Σ scalar • vector = vector
7. scalar • point = point iff scalar = 1
   = vector iff scalar = 0
   = undefined otherwise
8. Σ scalar • point = point iff Σ scalar = 1
   = vector iff Σ 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

$$\underbrace{(x_1, y_1, z_1, 1)}_{\textit{point}} + \underbrace{(x_2, y_2, z_2, 1)}_{\textit{point}} = \underbrace{(x_1+x_2, y_1+y_2, z_1+z_2, 2)}_{\textit{undefined}}$$

$$\underbrace{(x_1, y_1, z_1, 1)}_{\textit{point}} - \underbrace{(x_2, y_2, z_2, 1)}_{\textit{point}} = \underbrace{(x_1-x_2, y_1-y_2, z_1-z_2, 0)}_{\textit{vector}}$$

$$\underbrace{(x_1, y_1, z_1, 1)}_{\textit{point}} + \underbrace{(x_2, y_2, z_2, 0)}_{\textit{vector}} = \underbrace{(x_1+x_2, y_1+y_2, z_1+z_2, 1)}_{\textit{point}}$$

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 !!

$$\begin{array}{ccccccccc}
(1,0,0,1) & + & (1,1,0,1) & = & (2,1,0,2) & = & (1, \frac{1}{2}, 0, 1) \\
\| & & \| & & & & \nparallel \\
(1,0,0,1) & + & (2,2,0,2) & = & (3,2,0,3) & = & (1, \frac{2}{3}, 0, 1)
\end{array}$$

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

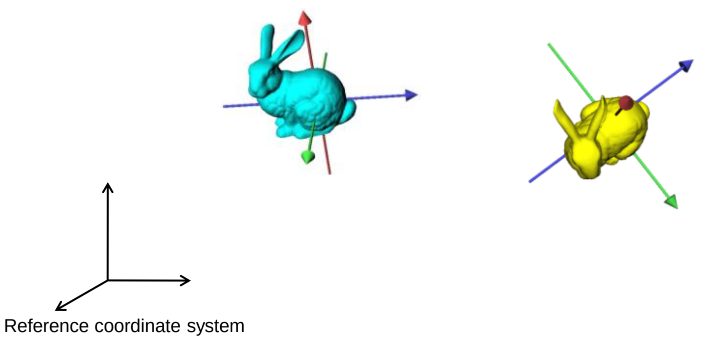
Orientation and Rotation

- ***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)

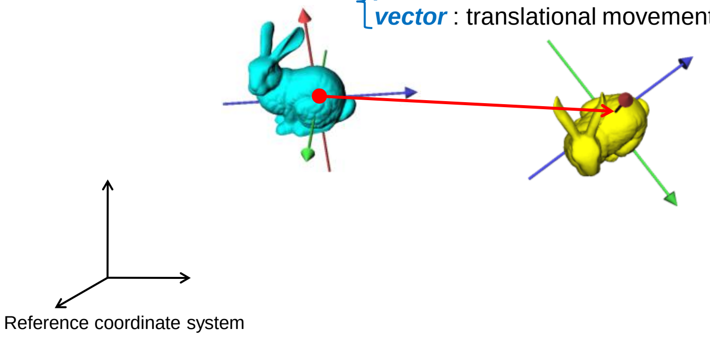


Analogy

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

Both represent a sort of (state : movement)

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

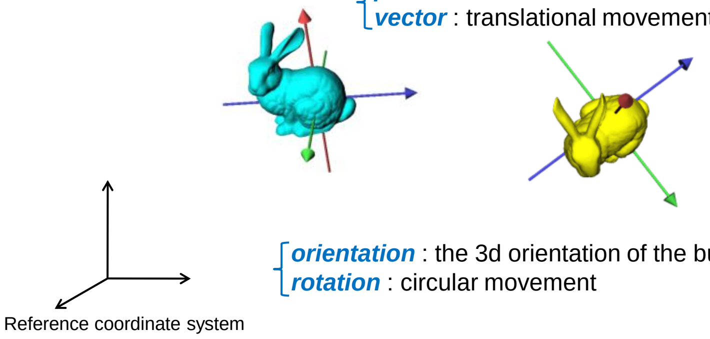


Analogy

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

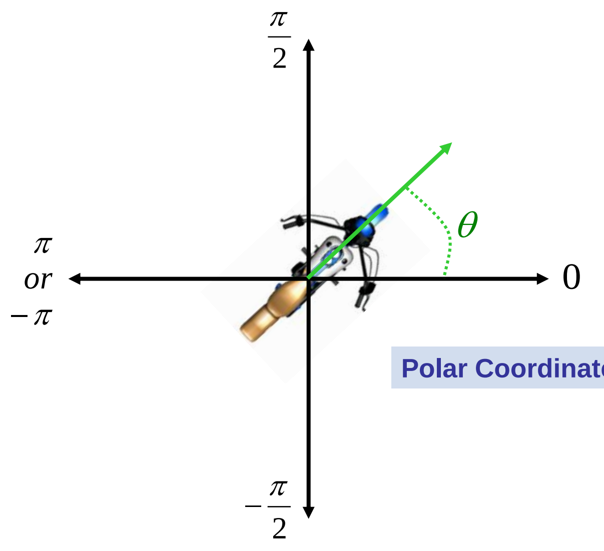
Both represent a sort of (state : movement)

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



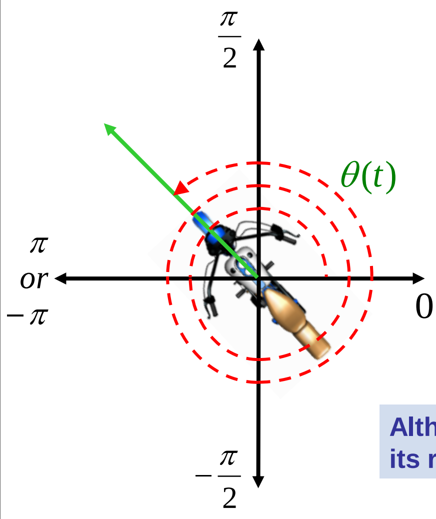
orientation : the 3d orientation of the bunny
rotation : circular movement

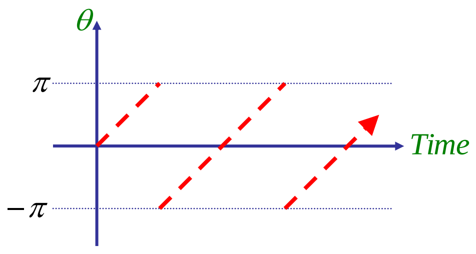
2D Orientation



Polar Coordinates

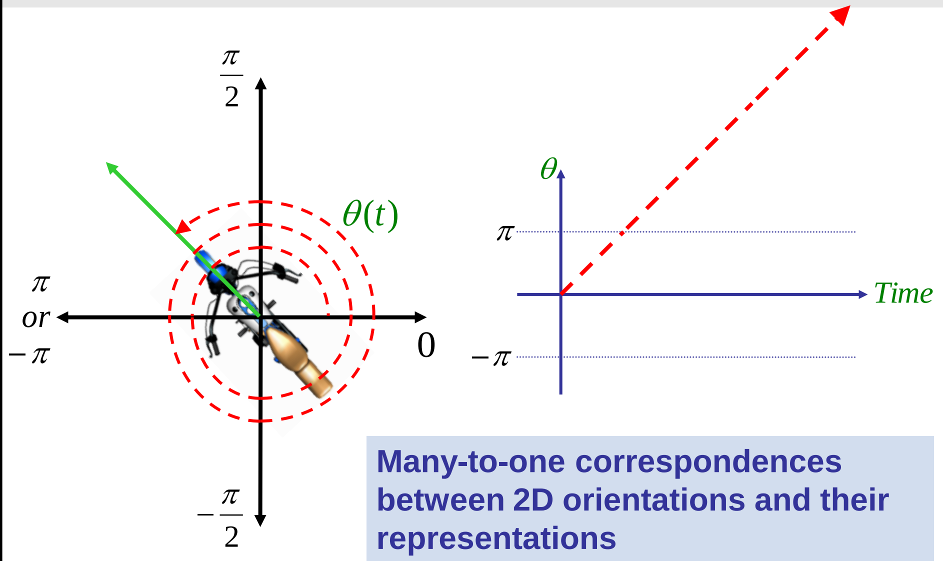
2D Orientation





Although the motion is continuous,
its representation could be discontinuous

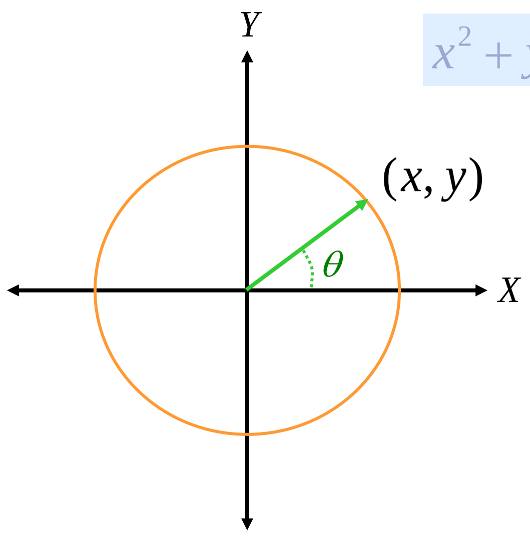
2D Orientation



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

Extra Parameter

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

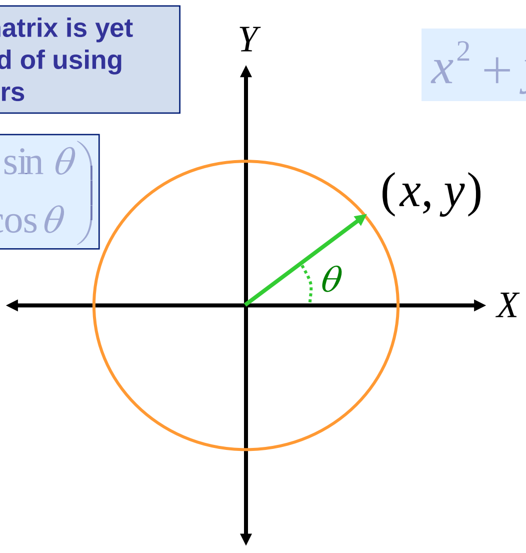


Extra Parameter

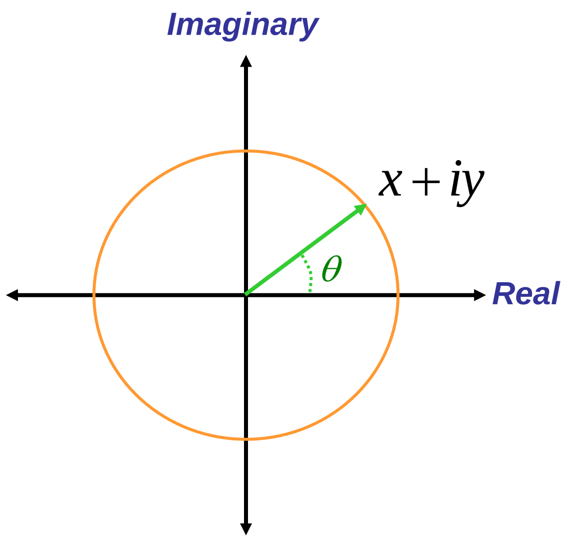
2x2 Rotation matrix is yet another method of using extra parameters

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

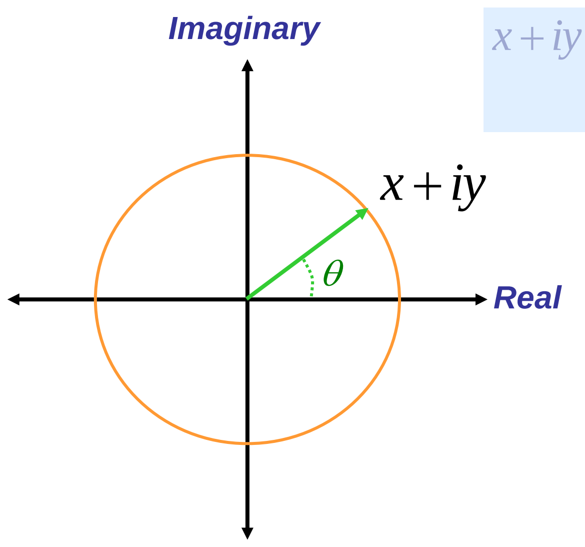
$$x^2 + y^2 = 1$$



Complex Number of Unit Length



Complex Exponentiation

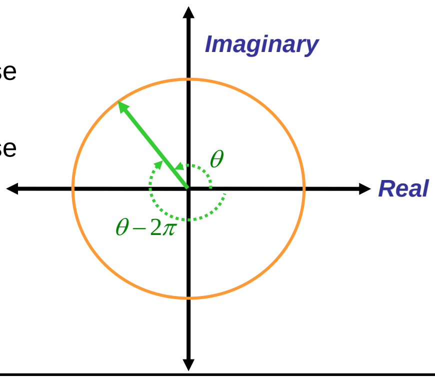


$$x + iy = \cos\theta + i\sin\theta$$
$$= e^{i\theta}$$

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

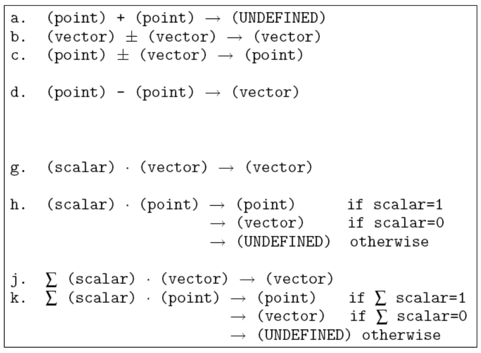


2D Rotation and Orientation

- ***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

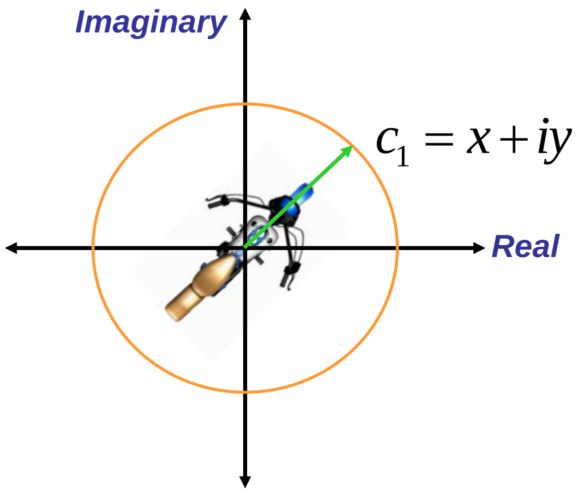
Operations in 2D

- (orientation) : complex number
- (rotation) : scalar value
- exp(rotation) : complex number

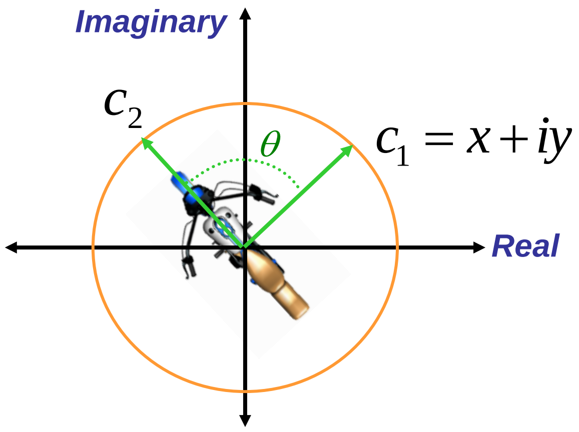


```
a.  (point) + (point) → (UNDEFINED)
b.  (vector) ± (vector) → (vector)
c.  (point) ± (vector) → (point)

d.  (point) - (point) → (vector)



g.  (scalar) · (vector) → (vector)

h.  (scalar) · (point) → (point)      if scalar=1
                       → (vector)     if scalar=0
                       → (UNDEFINED)  otherwise

j.  ∑ (scalar) · (vector) → (vector)
k.  ∑ (scalar) · (point) → (point)     if ∑ scalar=1
                         → (vector)    if ∑ scalar=0
                         → (UNDEFINED) otherwise
```

2D Rotation and Displacement



2D Rotation and Displacement



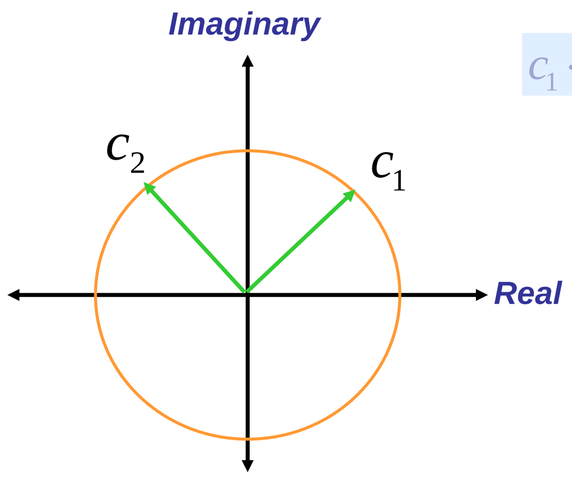
$$c_2 = c_1 \cdot e^{i\theta}$$

or

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

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

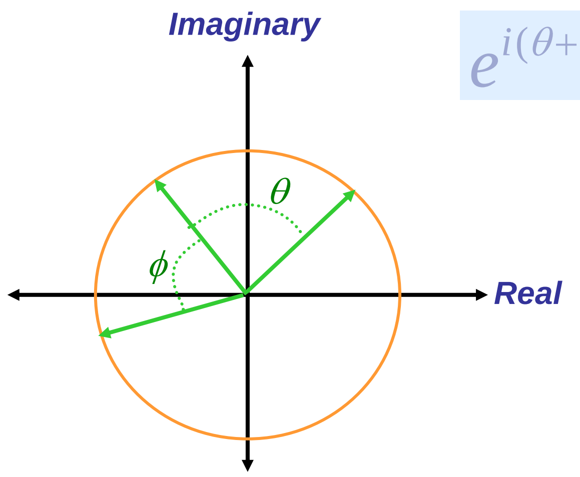
2D Orientation Composition



$c_1 \cdot c_2 = (undefined)$

`(orientation) · (orientation) → (UNDEFINED)`

2D Rotation Composition



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

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

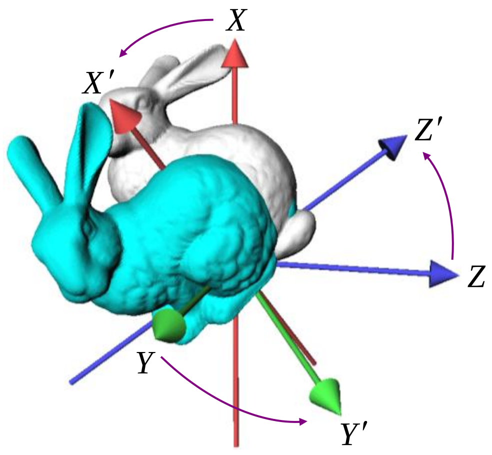
Analogy

```
(point) + (point) → (UNDEFINED)
(vector) ± (vector) → (vector)
(point) ± (vector) → (point)
(point) - (point) → (vector)
```

```
(orientation) · (orientation) → (UNDEFINED)
exp(rotation) · exp(rotation) → exp(rotation)
(orientation) · exp(rotation) → (orientation)
(orientation)^-1 · (orientation) → exp(rotation)
```

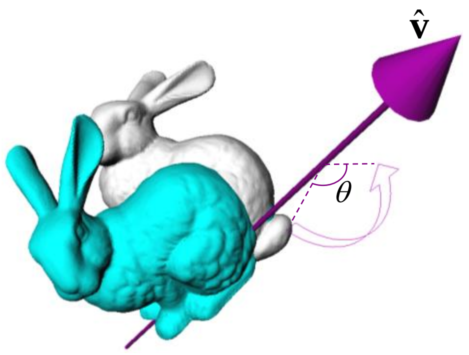
3D Rotation

- Given two arbitrary orientations of a rigid object,

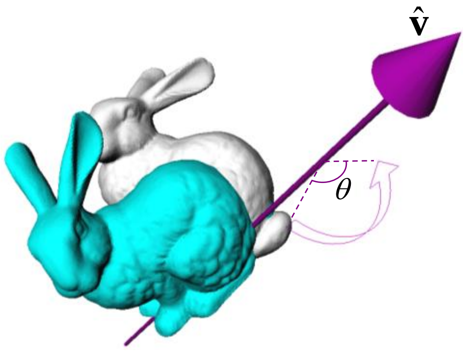


3D Rotation

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



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

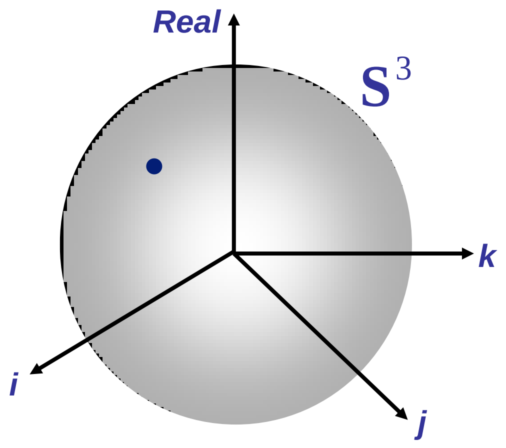
Orientations and rotations are different in coordinate-invariant geometric programming

Use unit quaternions for orientation representation
- 3x3 orthogonal matrix is theoretically identical

Use 3-vectors for rotation representation

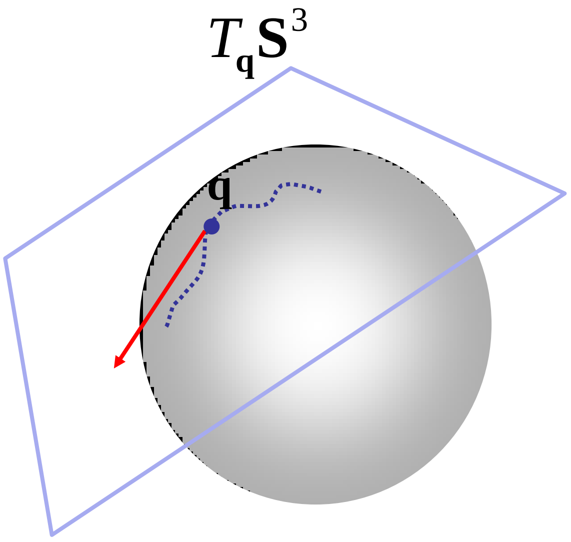
Unit Quaternions

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

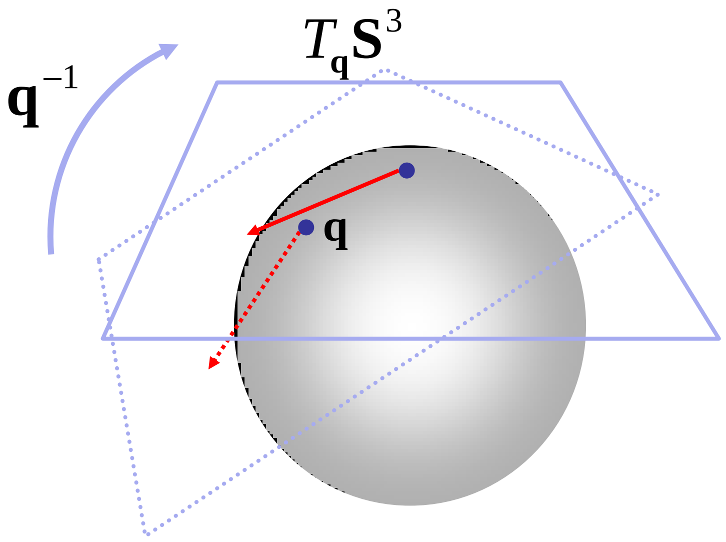


$$w^2 + x^2 + y^2 + z^2 = 1$$

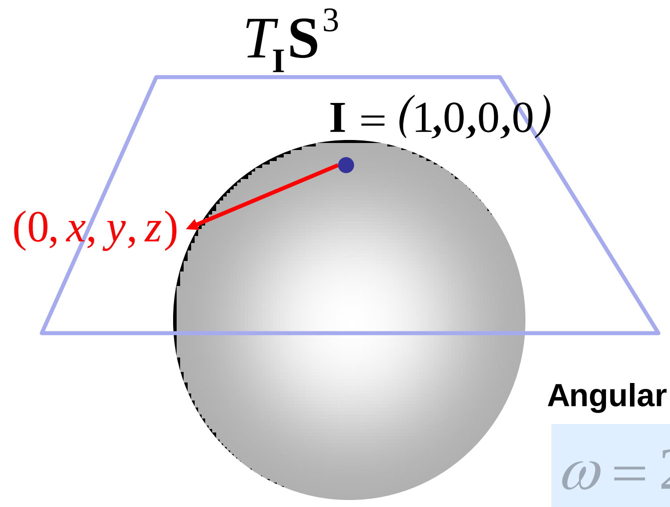
Tangent Vector
(Infinitesimal Rotation)



Tangent Vector
(Infinitesimal Rotation)



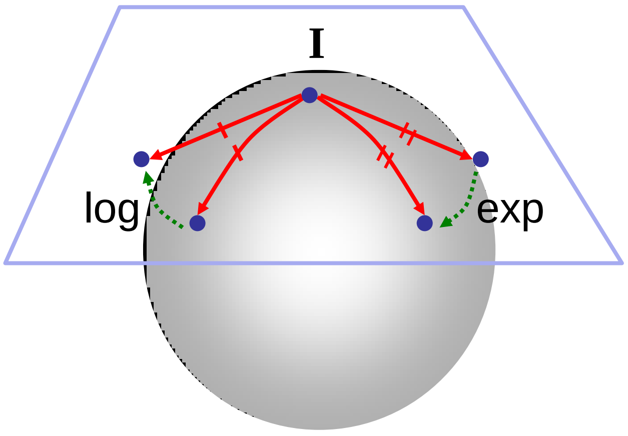
Tangent Vector
(Infinitesimal Rotation)



Angular Velocity

$$\omega = 2\mathbf{q}^{-1}\dot{\mathbf{q}}$$

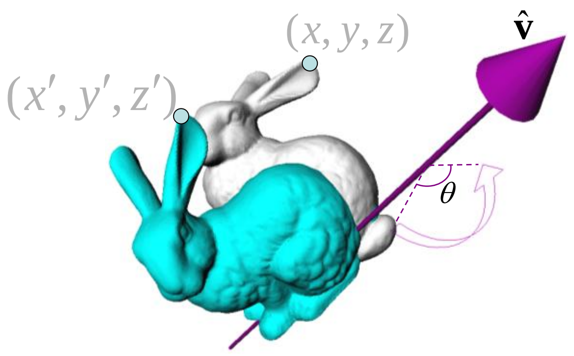
Exp and Log



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

Rotation about an Arbitrary Axis

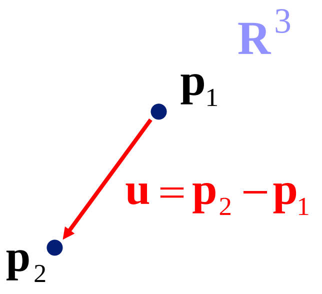
- Rotation about axis $\hat{\mathbf{v}}$ by angle θ

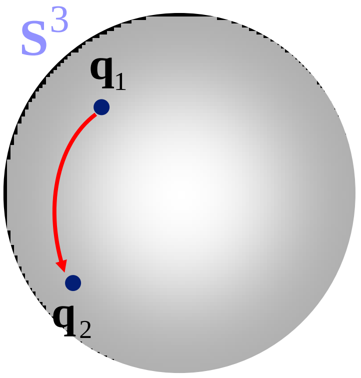


$\mathbf{p}' = \exp(v/2) \cdot \mathbf{p} \cdot \exp(-v/2)$ **where** $\mathbf{p} = (0, x, y, z)$

Purely Imaginary Quaternion

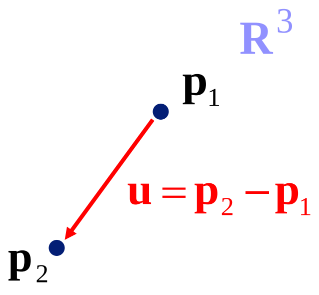
3D Rotation and Displacement

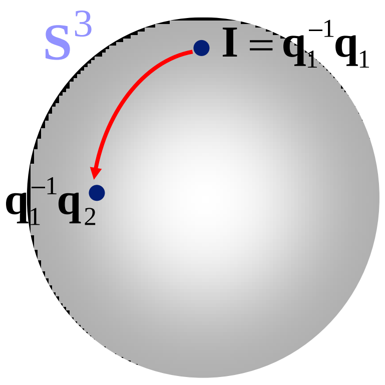




$$\begin{aligned}\mathbf{p}_2 &= \mathbf{p}_1 + \mathbf{u} \\ &= \mathbf{p}_1 + (\mathbf{p}_2 - \mathbf{p}_1)\end{aligned}$$

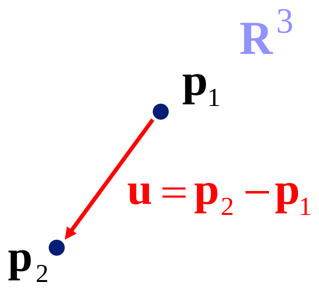
3D Rotation and Displacement

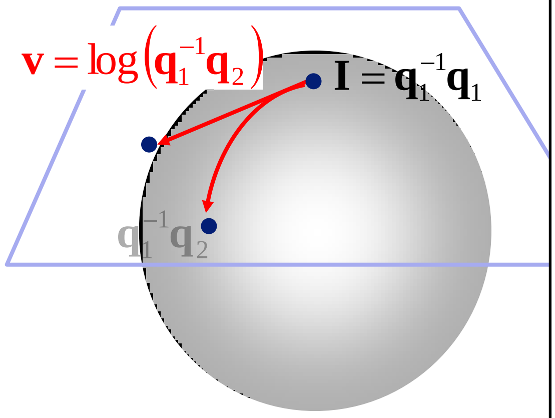



$\mathbf{S}^3$
$\mathbf{I} = \mathbf{q}_1^{-1}\mathbf{q}_1$
$\mathbf{q}_1^{-1}\mathbf{q}_2$

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

3D Rotation and Displacement



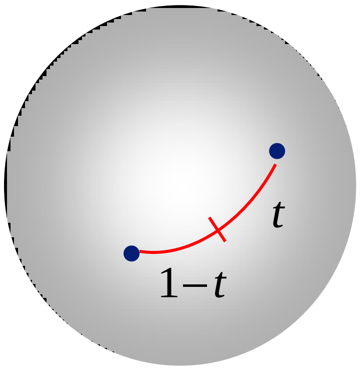

$\mathbf{v} = \log\left(\mathbf{q}_1^{-1}\mathbf{q}_2\right)$
$\mathbf{I} = \mathbf{q}_1^{-1}\mathbf{q}_1$
$\mathbf{q}_1^{-1}\mathbf{q}_2$

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

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

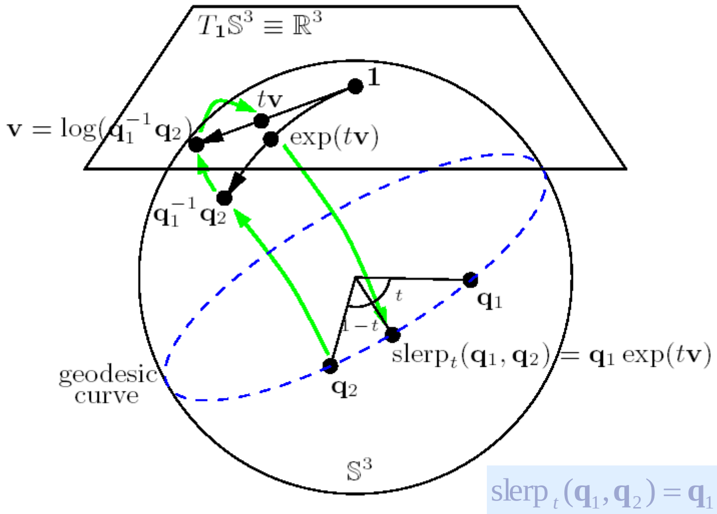
Spherical Linear Interpolation

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



$$\mathrm{slerp}_t(\mathbf{q}_1,\mathbf{q}_2)=\mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t$$
$$=\mathbf{q}_1\exp(t\cdot\log(\mathbf{q}_1^{-1}\mathbf{q}_2))$$

Spherical Linear Interpolation



$T_1\mathbb{S}^3 \equiv \mathbb{R}^3$
$\mathbf{1}$
$t\mathbf{v}$
$\mathbf{v}=\log(\mathbf{q}_1^{-1}\mathbf{q}_2)$
$\exp(t\mathbf{v})$
$\mathbf{q}_1^{-1}\mathbf{q}_2$
t
$\mathbf{q}_1$
$1-t$
$\mathrm{slerp}_t(\mathbf{q}_1,\mathbf{q}_2)=\mathbf{q}_1\exp(t\mathbf{v})$
geodesic curve
$\mathbf{q}_2$
$\mathbb{S}^3$

$$\mathrm{slerp}_t(\mathbf{q}_1,\mathbf{q}_2)=\mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t$$
$$=\mathbf{q}_1\exp(t\cdot\log(\mathbf{q}_1^{-1}\mathbf{q}_2))$$

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

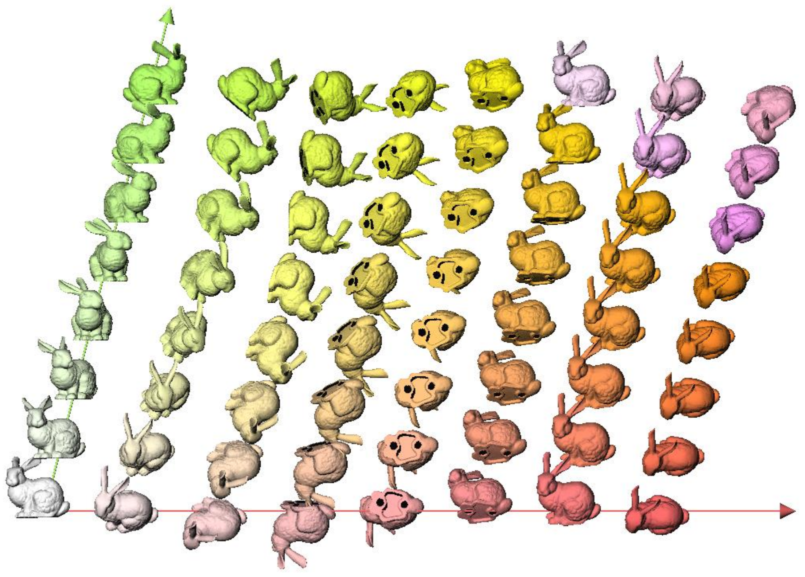
j. ∑ (scalar) · (vector) → (vector)
k. ∑ (scalar) · (point) → (point) if ∑ scalar=1
→ (vector) if ∑ scalar=0
→ (UNDEFINED) otherwise

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)$^{(\text{scalar})}$ → exp(rotation)
H. (orientation)$^{(\text{scalar})}$ → (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)

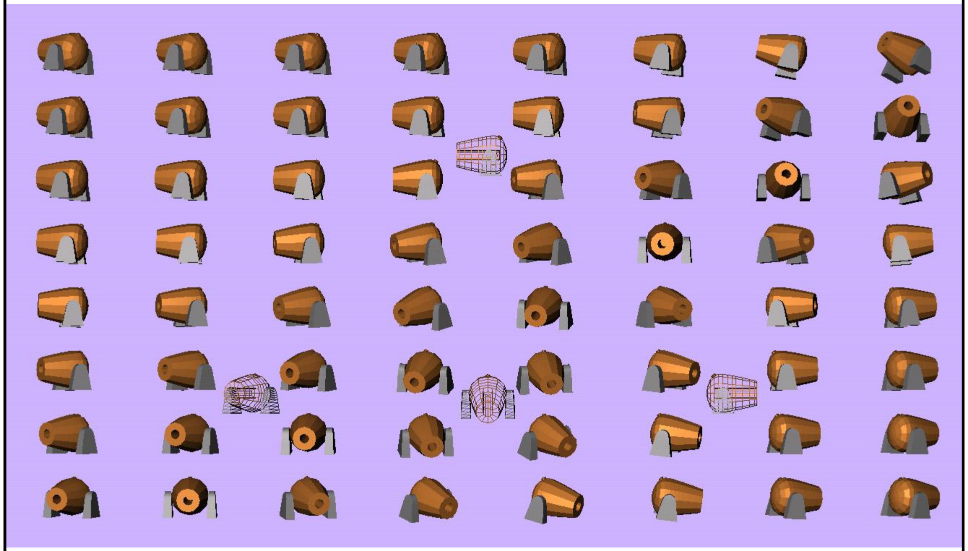
Coordinate-Invariant Operations

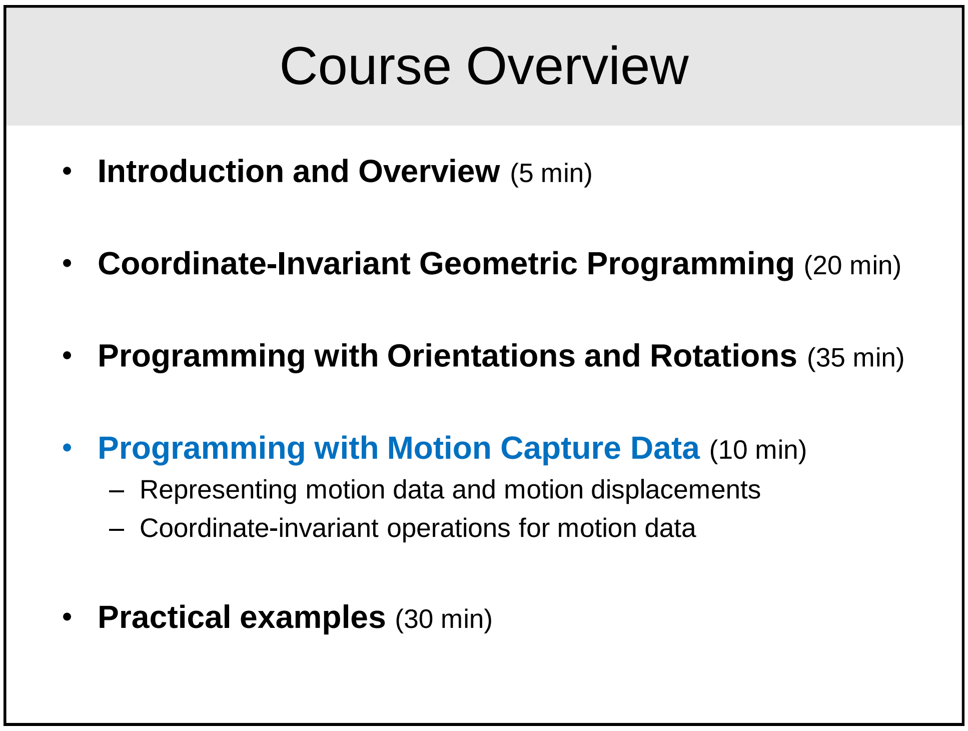
A. (orientation) · (orientation) → (UNDEFINED)
B. exp(rotation) · exp(rotation) → exp(rotation)
C. (orientation) · exp(rotation) → (orientation)
exp(rotation) · (orientation) → (orientation)
D. (orientation)$^{-1}$ · (orientation) → exp(rotation)
(orientation) · (orientation)$^{-1}$ → exp(rotation)
E. log(exp(rotation)) → (rotation)
F. log(orientation) → (UNDEFINED)
G. (scalar) · (rotation) → (rotation)
exp(rotation)$^{(\text{scalar})}$ → exp(rotation)
H. (orientation)$^{(\text{scalar})}$ → (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



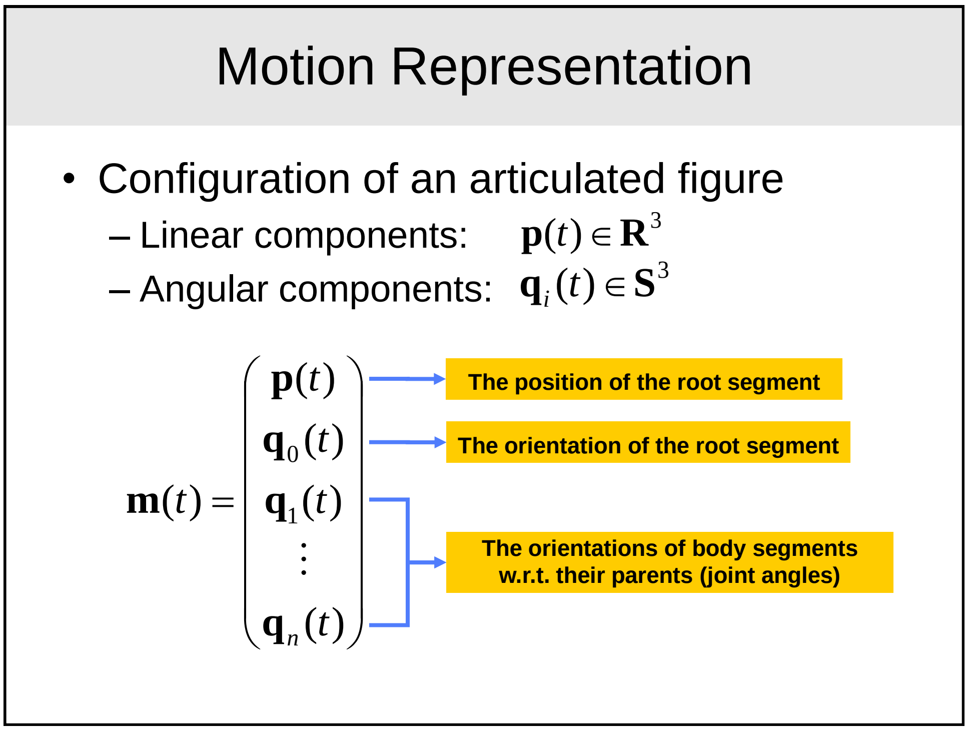
Affine Combination of Orientations





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: $\mathbf{q}_i(t) \in \mathbf{S}^3$

$$\mathbf{m}(t) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_0(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix}$$

The position of the root segment

The orientation of the root segment

The orientations of body segments w.r.t. their parents (joint angles)

Motion Displacement

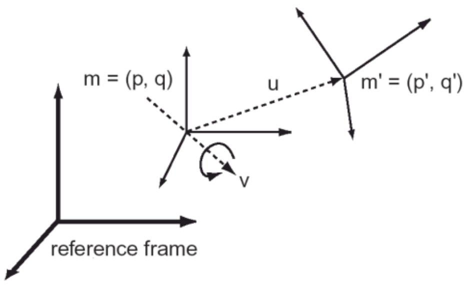
- Independent translation and rotation

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

Motion Displacement

- Rigid Transformation at the root segment

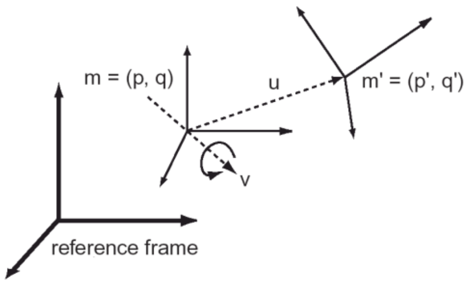
$$\mathbf{m}'(t) = \mathbf{m}(t) \oplus \mathbf{d}(t) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}_1(t) \\ \vdots \\ \mathbf{q}_n(t) \end{pmatrix} \oplus \begin{pmatrix} \mathbf{v}_0(t) \\ \exp(\mathbf{v}_1(t)) \\ \vdots \\ \exp(\mathbf{v}_n(t)) \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1\mathbf{v}_0\mathbf{q}_1^{-1}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



Motion Displacement

- Rigid Transformation at the root segment

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



Element-wise Operations

- Translation

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

- Rotation

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

Element-wise Operations

- Exercise joints

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

Operations

- Valid operations

$$\mathbf{m}_1(t) \oplus \widetilde{\exp}(\mathbf{d}(t)) = \mathbf{m}_2(t)$$
$$\mathbf{m}_2(t) \ominus \mathbf{m}_1(t) = \widetilde{\exp}(\mathbf{d}(t))$$
$$\alpha \mathbf{d}_1(t) = \mathbf{d}_2(t)$$
$$\mathbf{d}_1(t) + \mathbf{d}_2(t) = \mathbf{d}_3(t) \quad \text{(Be careful !)}$$

- Invalid operations

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

Operations

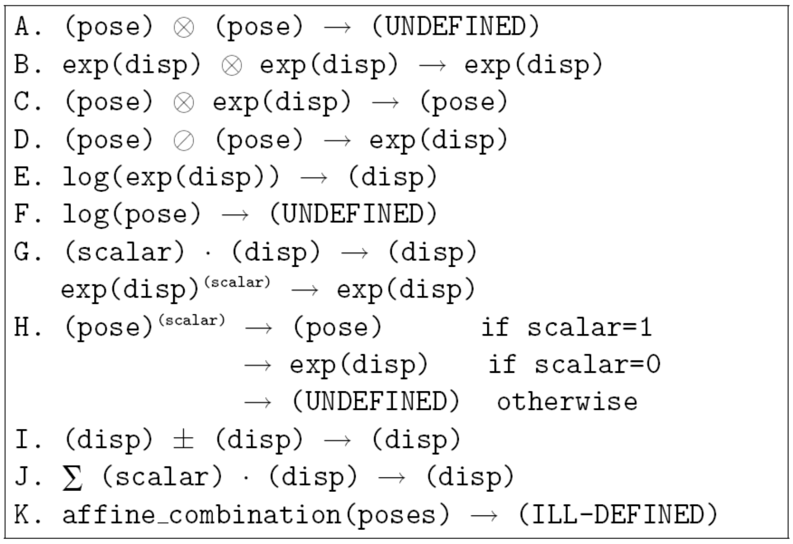
- Time warping

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

- Properties

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

Coordinate-Invariant Operations

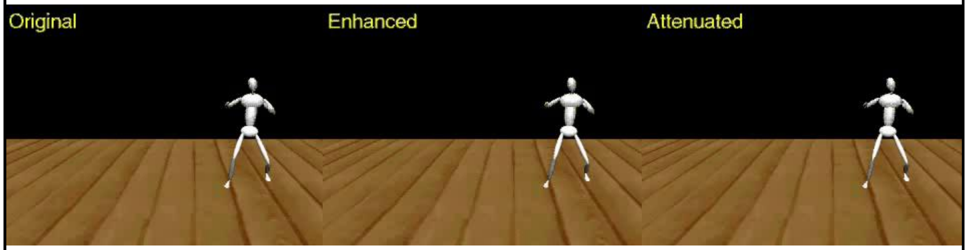


```
A. (pose) ⊗ (pose) → (UNDEFINED)
B. exp(disp) ⊗ exp(disp) → exp(disp)
C. (pose) ⊗ exp(disp) → (pose)
D. (pose) ⊘ (pose) → exp(disp)
E. log(exp(disp)) → (disp)
F. log(pose) → (UNDEFINED)
G. (scalar) · (disp) → (disp)
   exp(disp)^(scalar) → exp(disp)
H. (pose)^(scalar) → (pose)        if scalar=1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

Course Overview

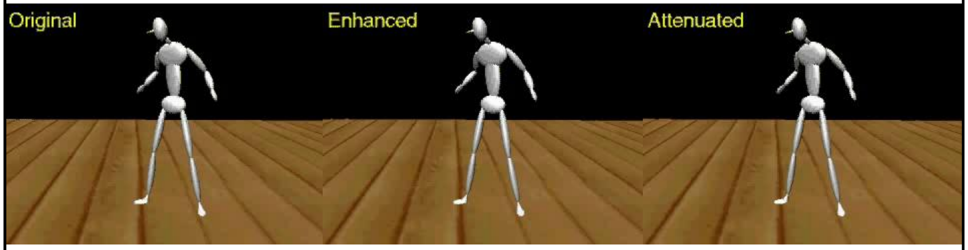
- **Introduction and Overview** (5 min)
- **Coordinate-Invariant Geometric Programming** (20 min)
- **Programming with Orientations and Rotations** (35 min)
- **Programming with Motion Capture Data** (10 min)
- **Practical examples** (30 min)
  - Motion exaggeration and style transfer
  - Aligning and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

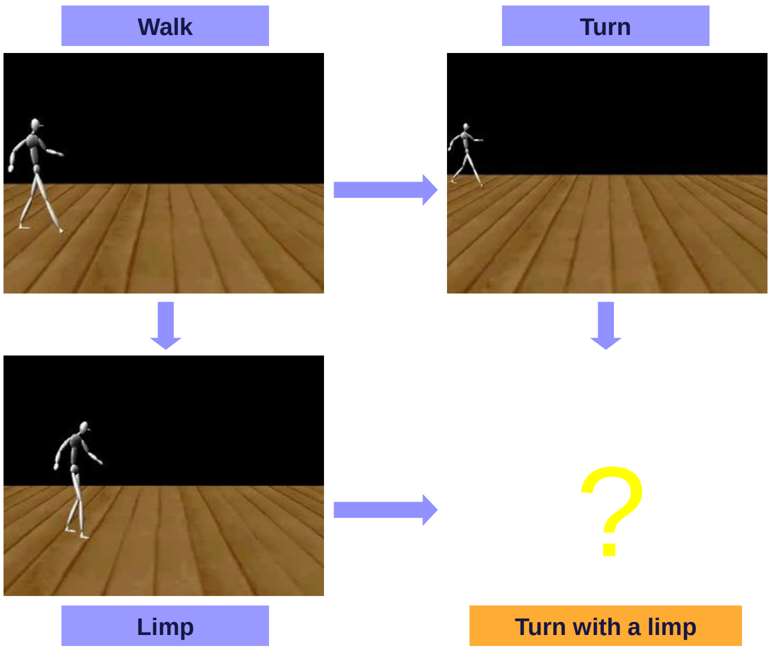
Motion Exaggeration

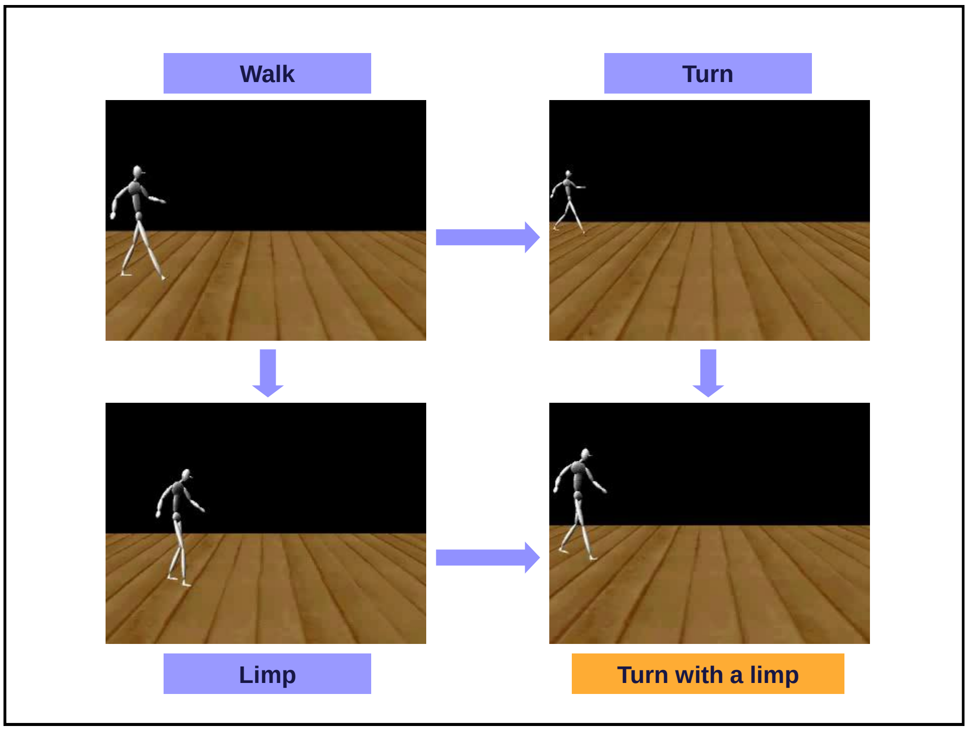


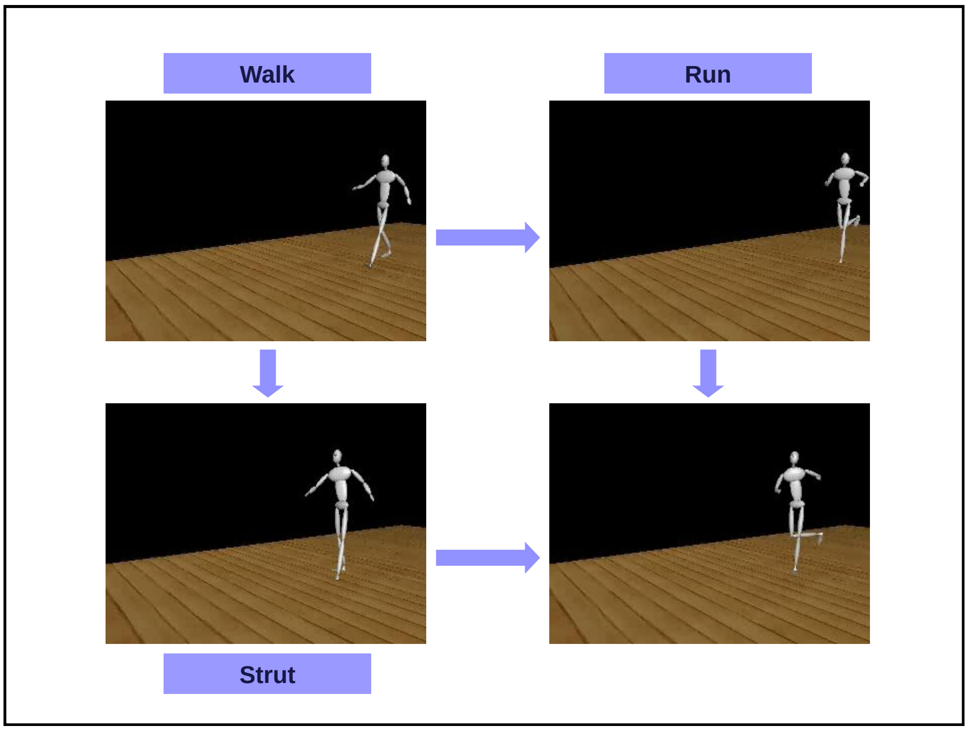
$$\mathbf{M}'(t) = \mathbf{L}(t) \oplus \left(\mathbf{M}(t) \ominus \mathbf{L}(t)\right)^{\alpha}$$

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

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




Walk
Turn
Limp
Turn with a limp

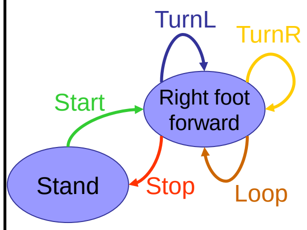

Walk
Run
Strut

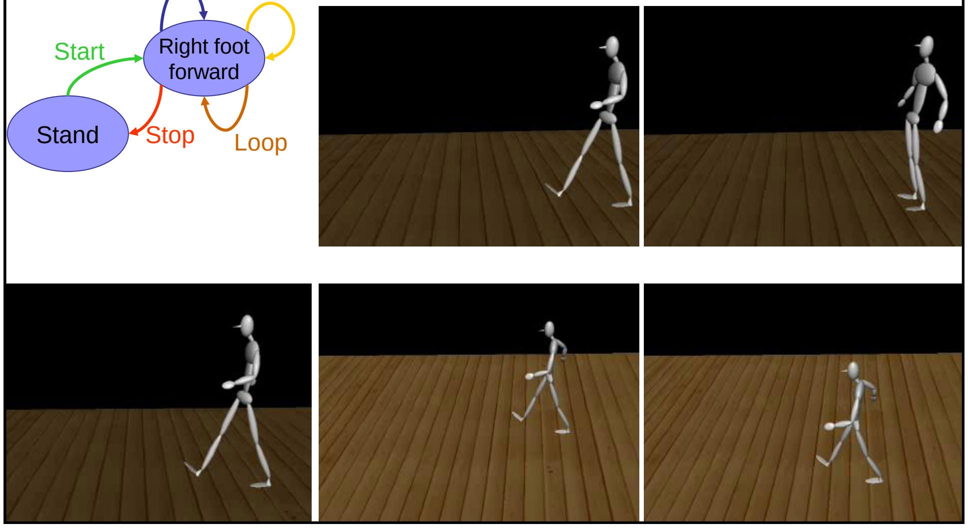
Style Transfer

$$\begin{tikzcd} A \arrow[r] \arrow[d] & A' \arrow[d] \\ B \arrow[r] & \boxed{B'} \end{tikzcd}$$

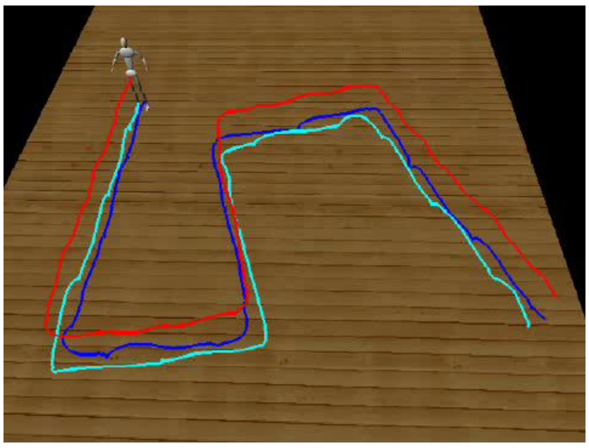
$$\mathbf{B}'(t) = \mathbf{B}(t) \oplus \left(\mathbf{A}(t) \ominus \mathbf{A}'(t) \right)$$

Transition Graph



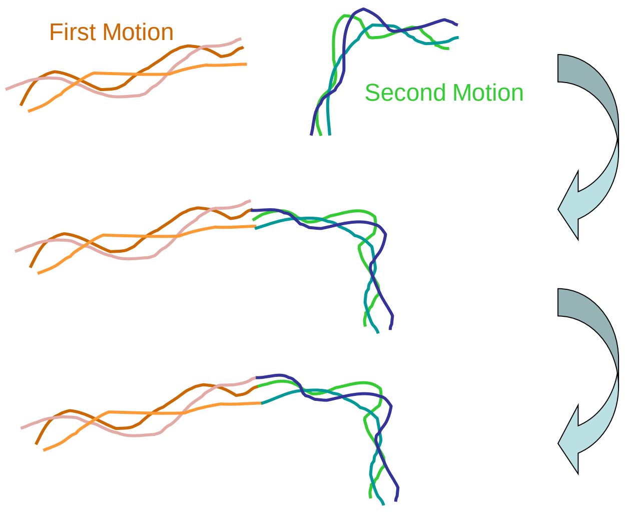


Transition Graph



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
Rotate and translate the second motion to align two motions

Warping
Warp the motions at the boundary so that they can be connected smoothly

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

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

Alignment Using In-Plane Transformation

- Rigid transformation restricted within a plane
  - Rotation about the vertical axis, followed by
  - Translation along two horizontal axes

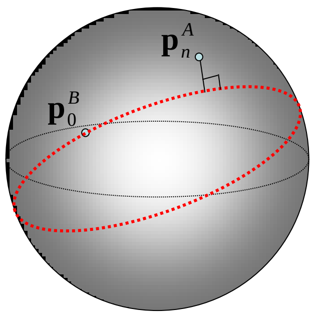
- Finding in-plane rotation close to given rotation
  - Using Euler angles
  - Discard rotation about x- and z-axes
  - Not optimal

$$R = R_x R_y R_z$$

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:*

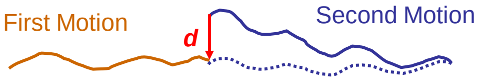
$$
\bar{q} = \begin{cases} G(-\alpha + \frac{\pi}{2}) & \text{if } q_s \cdot G(-\alpha + \frac{\pi}{2}) > q_s \cdot G(-\alpha - \frac{\pi}{2}), \\ G(-\alpha - \frac{\pi}{2}) & \text{if } q_s \cdot G(-\alpha + \frac{\pi}{2}) < q_s \cdot G(-\alpha - \frac{\pi}{2}), \end{cases}
$$

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

Hyun Joon Shin, Jehee Lee, Michael Gleicher, Sung Yong Shin,
Computer Puppetry: An Importance-based Approach
ACM Transactions on Graphics, 2001

Warping

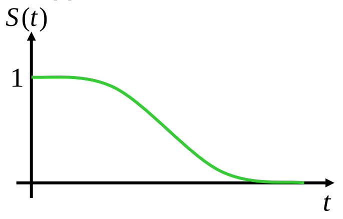
- Deform a motion smoothly so that it is seamlessly connected its previous motion



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

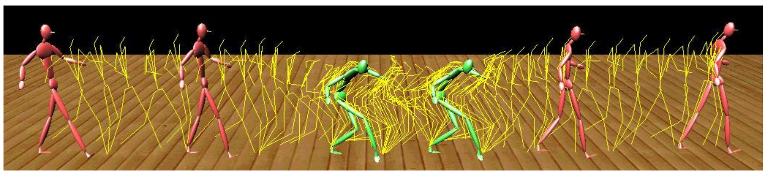
- A scalar transition function *s*(t)

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



Blending for Smooth Transition

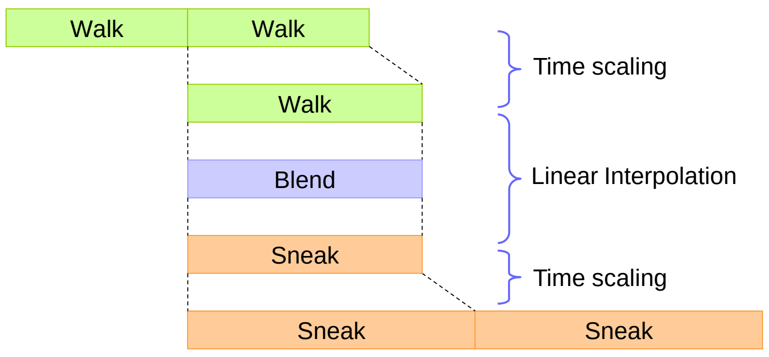
- Case study: walk-to-sneak
  - Transit smoothly over one cycle of locomotion
  - "Walk" is faster than "sneak"
  - One cycle of "sneak" is longer than one cycle of "walk"



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

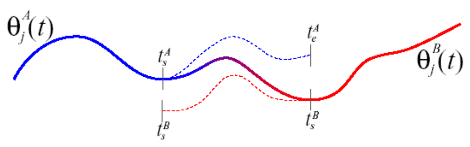
Blending for Smooth Transition

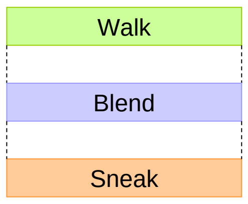
- Blend over the overlapping time interval



Blending for Smooth Transition

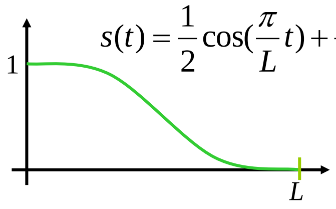
- Linear interpolation between motions
  - Slerp for orientation components
  - A scalar transition function **s(t)**





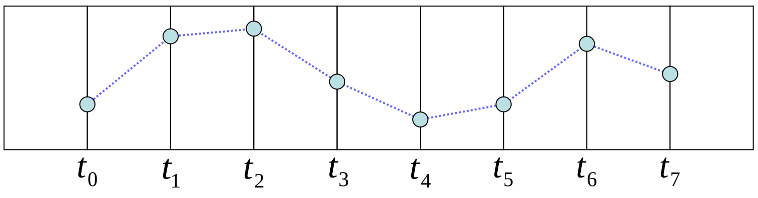
$$s(t)\cdot\mathbf{m}^A(t)\oplus(1-s(t))\cdot\mathbf{m}^B(t)$$

$$s(t)=\frac{1}{2}\cos\left(\frac{\pi}{L}t\right)+\frac{1}{2}$$



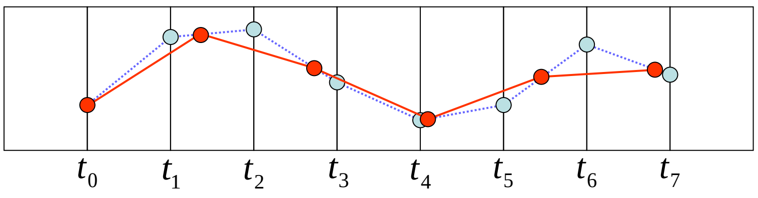
Time Scaling

- Uniform resampling
  - Motion data may be given as a sequence of discrete frames
  - It can be considered as a piecewise linear signal



Time Scaling

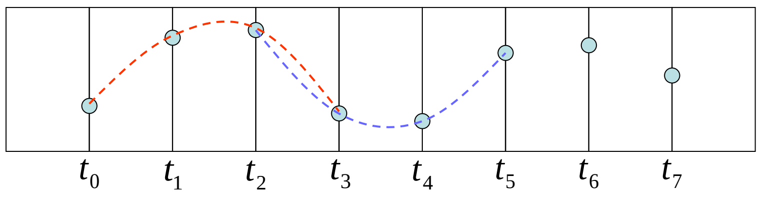
- Uniform resampling
  - Motion data may be given as a sequence of discrete frames
  - It can be considered as a piecewise linear signal



$$\mathbf{m}(t) = (1-\alpha)\mathbf{m}(t_i) \oplus \alpha\mathbf{m}(t_{i+1}), \quad \text{where} \quad \alpha = \frac{t - t_i}{t_{i+1} - t_i}$$

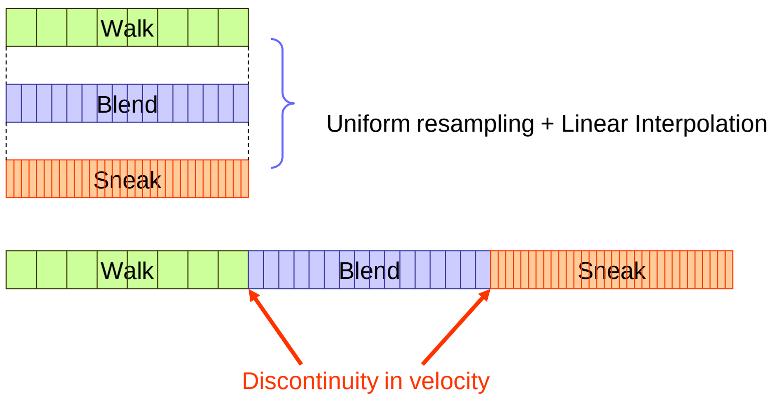
Time Scaling

- Cubic resampling
  - piecewise cubic interpolation



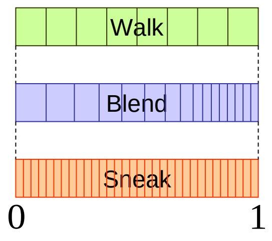
Time Scaling

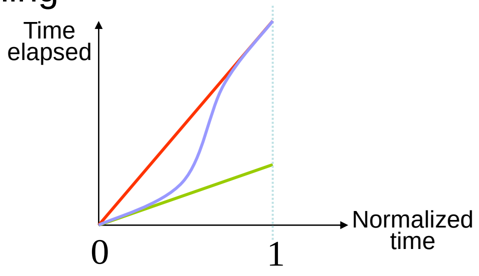
- Discontinuity in velocity


Walk
Blend
Sneak
Uniform resampling + Linear Interpolation
Walk
Blend
Sneak
Discontinuity in velocity

Time Scaling

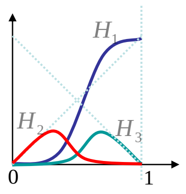
- Non-uniform resampling





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

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



$$H_1(t) = -2t^3 + 3t^2$$
$$H_2(t) = t^3 - 2t^2 + t$$
$$H_3(t) = t^3 - t^2$$

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