Computer Graphics & Scientific Visualization

MPRI 2-39

Descriptive Animation

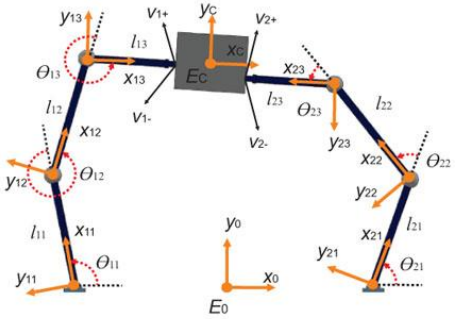
Animation in Computer Graphics

2 ways of description:

1- **Kinematics**

Shape deformation/motion without knowledge of the physical cause of the motion

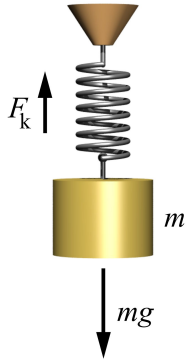
(Today)



2- **Dynamics**

Motion through the forces acting on the shapes

(Physically based simulation)



Animation in Computer Graphics

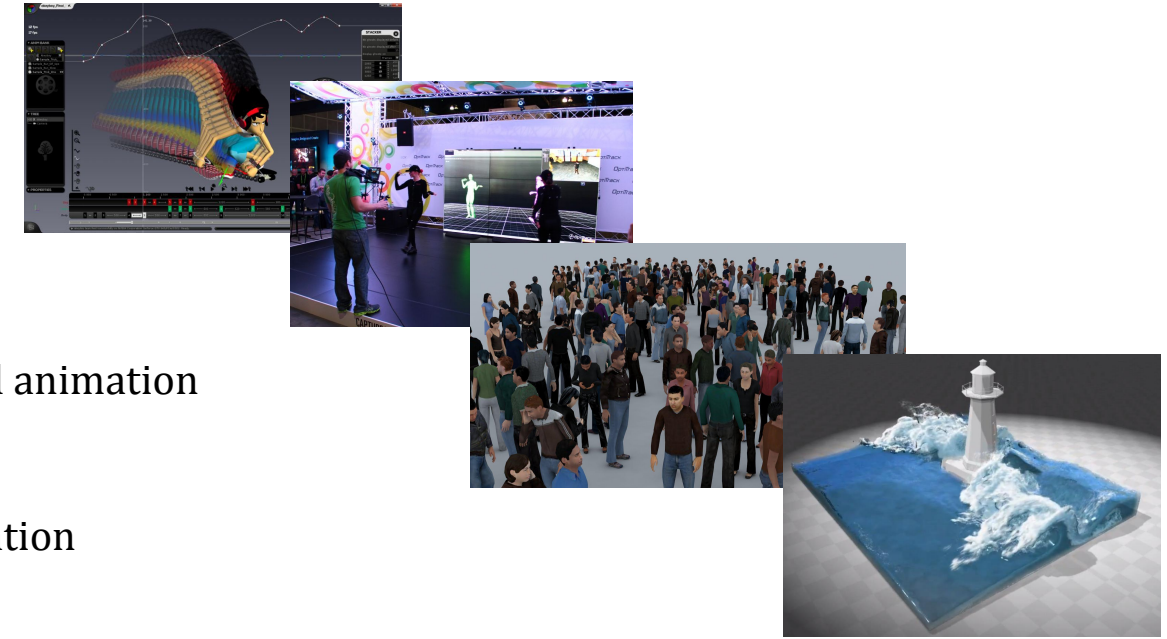
4 ways of computing/handling animation:

1- Descriptive animation

2- Acquired motion

3- Procedurally generated animation

4- Physically-based animation



CG Animation in this course

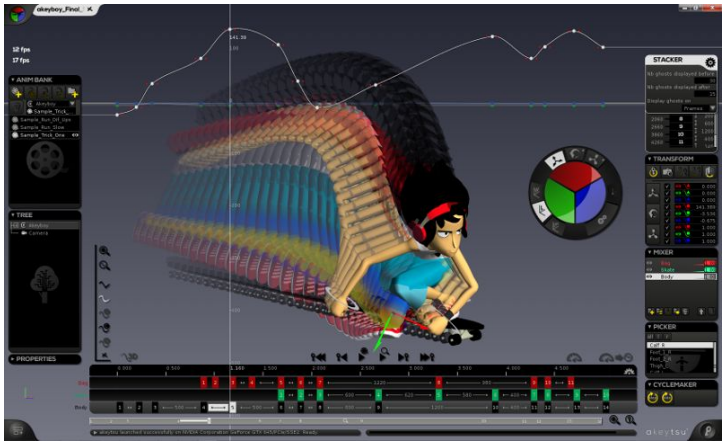
- **Today**: General descriptive kinematics animation.
- 06/12 Principles of physically-based simulation.
- 10/01 Character animation (skeleton).
- 17/01 Character animation (skin, clothes, hair).

Kinematics animation

Descriptive animation:

the artist/an algorithm fully describes the motion/deformation (only Kinematics)

+ Full control on the result
- May introduce un-physical artifacts



History of CG Animation

First production animation films (Disney)

- 30 drawings/s

Principle:

- Animator in chief: create **Key Frames**
- Others: secondary drawings
  (/in between)

(c) Walt Disney Company, from "The Illusion of Life"

=> Principle of "Key Framing"

Key Framing in CG

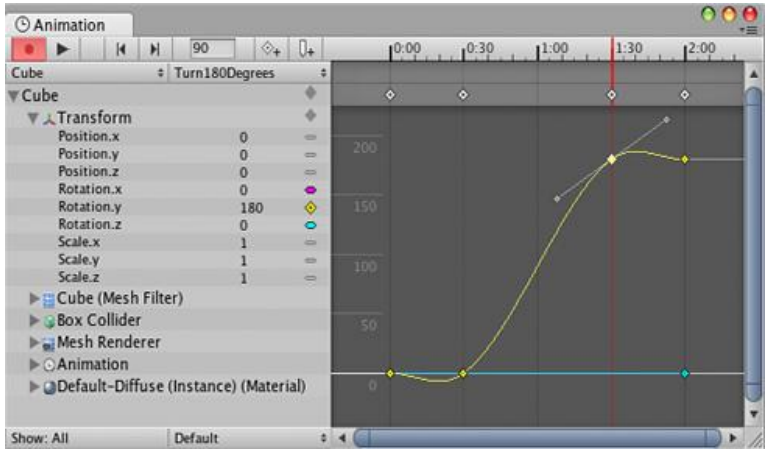
- Create “manually” a set of **KeyFrames** at specific time
- Compute automatically intermediate frames (30 fps) using **interpolation**

Key Framing in CG

1) **Posing** the “key frames”
   = Geometrical deformation of the character
   *(may require **Rigging**)*

2) **Animating** the in-betweens
   - *Spacing* (placement of the key frames in time)
   - *Timing* (speed of interpolation)

Use of **Animation Curves**



Key Framing in CG

1) **Posing** the “key frames”
   = Geometrical deformation of the character
   *(may require **Rigging**)*

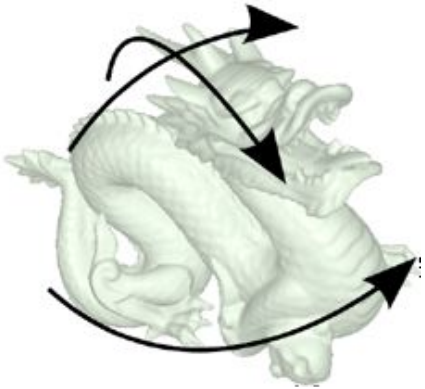
More details on Character Animation courses
(10/01, 17/01)

In-between interpolation

Interpolation for animation

What do we need to interpolate:

- Individual **positions**
- Surface element: **translation**, **scaling**, **rotation**

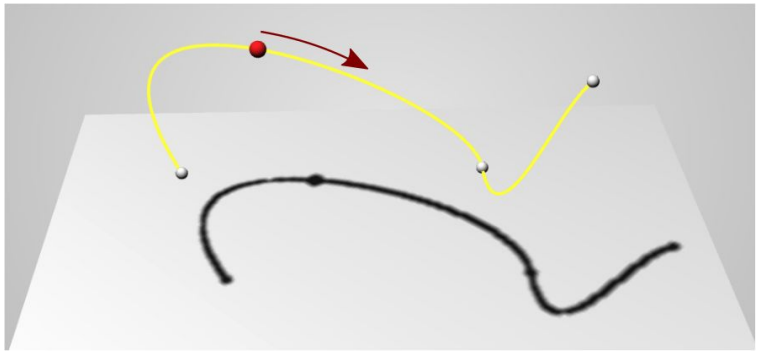


Interpolating position (/scaling)

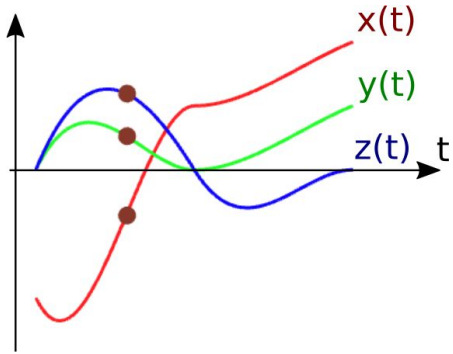
- Interpolating positions(/scaling): Use of Hermite interpolation

Can control position and tangents

$$\mathbf{p}(\alpha) = (2\alpha^3 - 3\alpha^2 + 1)\,\mathbf{p}_1 + (-2\alpha^3 + 3\alpha^2)\,\mathbf{p}_2 + (\alpha^3 - 2\alpha^2 + \alpha)\,\mathbf{d}_1 + (\alpha^3 - \alpha^2)\,\mathbf{d}_2$$



Spatial trajectory



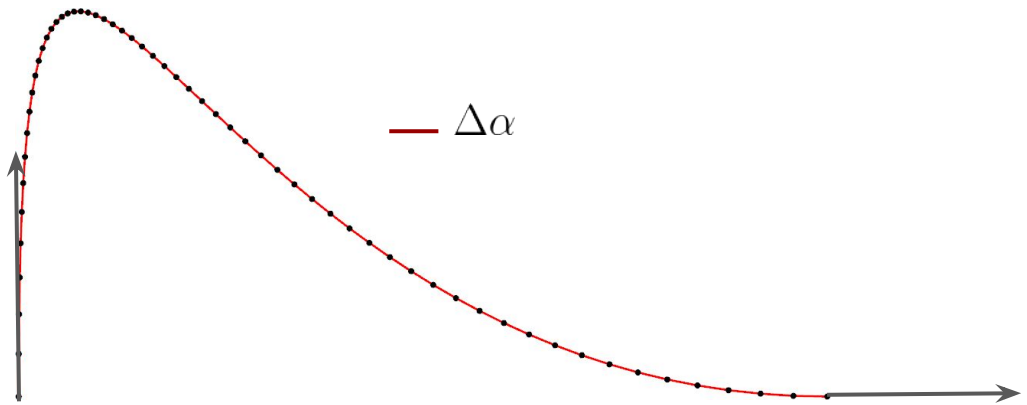
Temporal animation curves

Animation curve: re-sampling

Warning: Time parameter $t \neq \alpha$

α : Hermite curve parameter (curvature dependant)

t : Actual time parameter



Animation curve: re-sampling

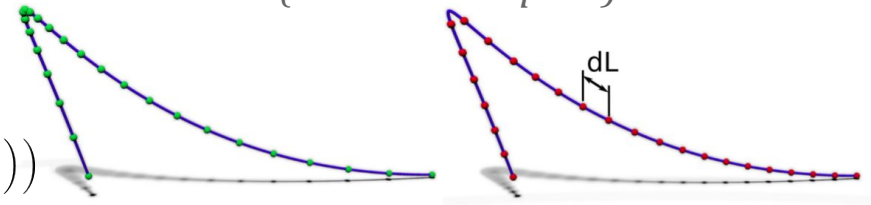
1- **Resample** uniformly the curve w/r arc length

Arc length: $s = \int_0^{\alpha} \|\mathbf{p}'(u)\| \, \mathrm{d}u$

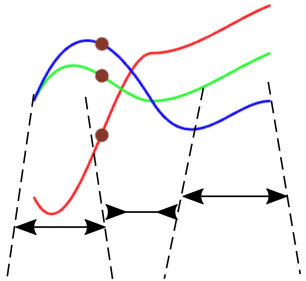
New parameterized curve: $\tilde{\mathbf{p}}(\alpha) = \mathbf{p}(s^{-1}(\alpha))$

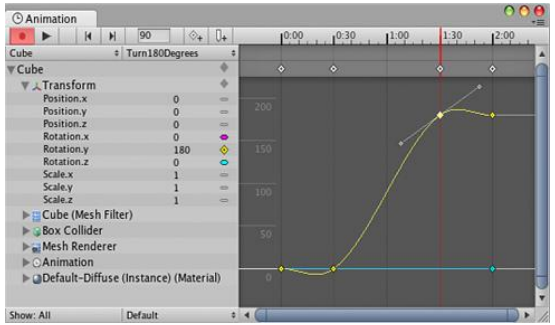
In practice: evaluated numerically

Use Hermite curve again (in time and space)



2- **Time warping** to adjust for the timing

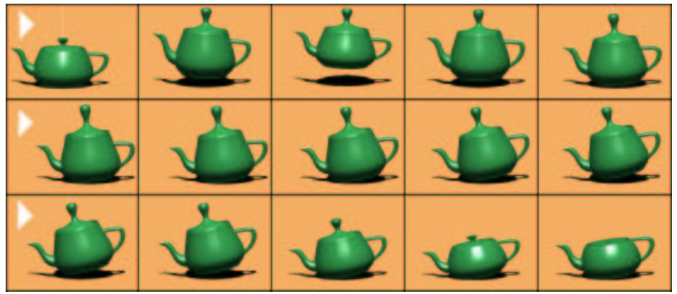




Key Frame on cages

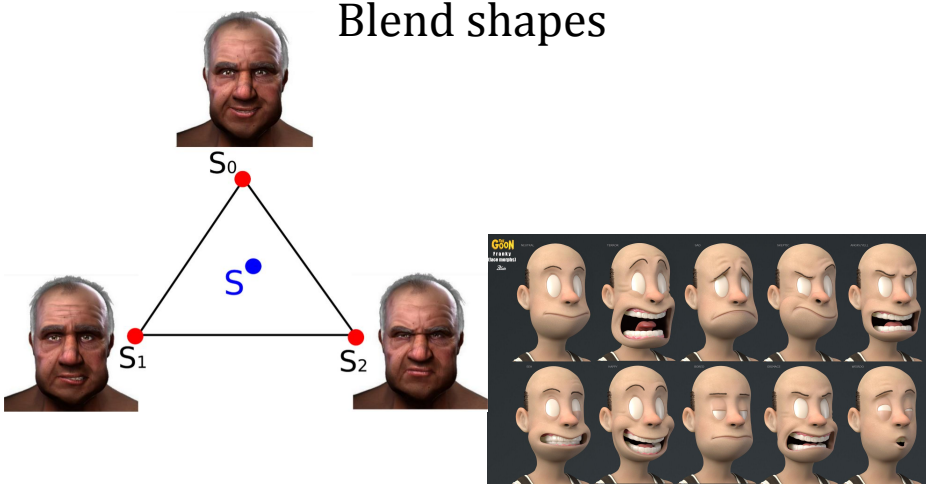
Note: The key framing can be set on a controller of the shape (see modeling courses)

Ex. Animated FFD, cages



[Faloutsos et al. Dynamic Animation Synthesis with Free-Form Deformations, TVCG 1997]

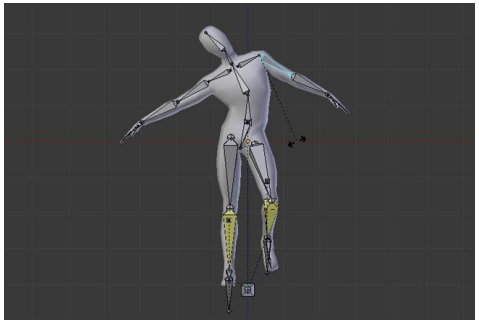
Blend shapes

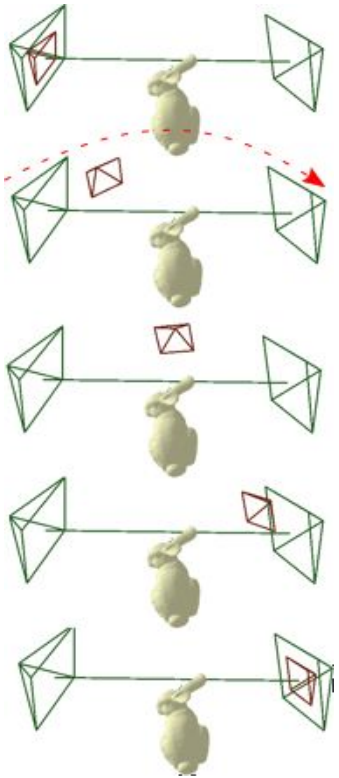


Interpolating **rotation**

Typical use of rotation interpolation

- Skeleton based character
- Camera interpolation





Problem: Rotation is **not** a vectorial space

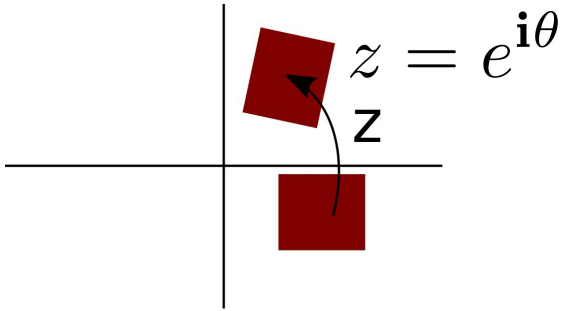
$$0.5\ (R_1 + R_2) \notin \text{Rotation}$$

Interpolating rotation in 2D

1- Model rotation by an angle.

2- Interpolate the angle

Note: Can be formalized using complex numbers representation



Given 2 rotation

$$z_0 = e^{\mathbf{i}\theta_0}$$

$$z_1 = e^{\mathbf{i}\theta_1}$$

the middle one is

$$z_{1/2} = e^{\mathbf{i}(\theta_0+\theta_1)/2}$$

Representing 3D rotation

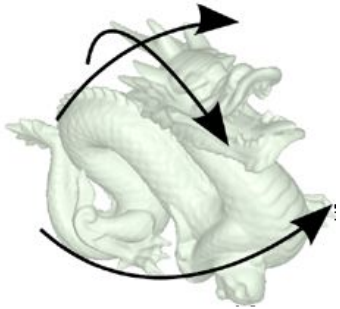
In 3D: Several ways to model a rotation

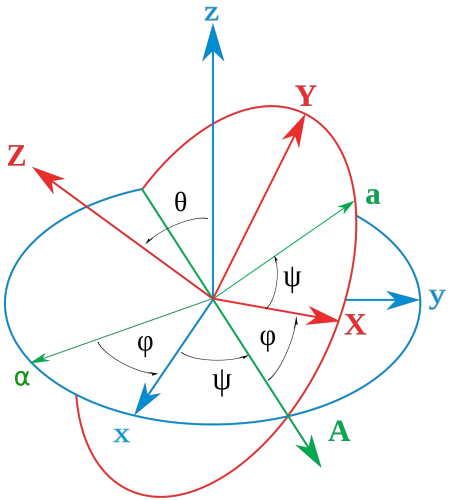
1- Matrices

2- Euler/Trait-bryan angles

3- Axis angle

4- Quaternion





Representing rotation: Matrix

$$R = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}$$

$$\det(R) = 1$$
$$R^T = R^{-1}$$

Advantage:

- Application on vector $R\,\mathbf{v}$
- Composition $R_1\,R_2$

Standard to store/manipulate

Limitations:

- 9 coefficients for 6 dof. Rotation parameters non-explicit.
- Unclear interpolation

$$M = (1-\alpha)\,R_1 + \alpha\,R_2$$

M: not a rotation, may be degenerated to 0

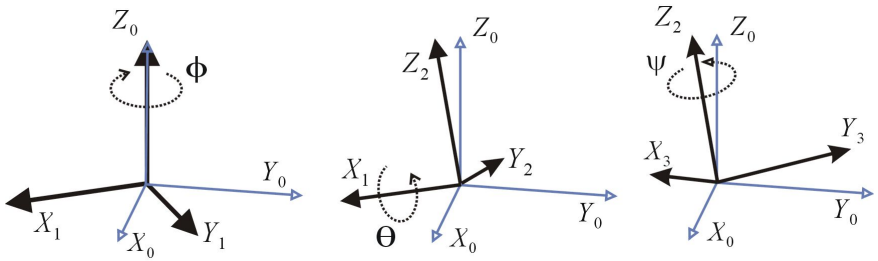
Representing rotation: Euler angle

Three consecutive rotations along (x,y,z) coordinates

Several different Euler angles

[Proper Euler] (z-x-z, x-y-x, y-z-y, … etc)

[Trait-Bryan] (x-y-z, y-z-x, z-x-y, … etc)

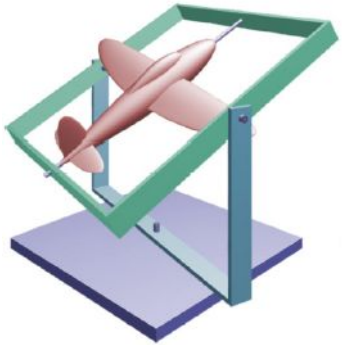


Advantage:

Combination of rotation around basis axes:

- Comprehensive parameters (3 dof)
- Easy conversion to matrix.

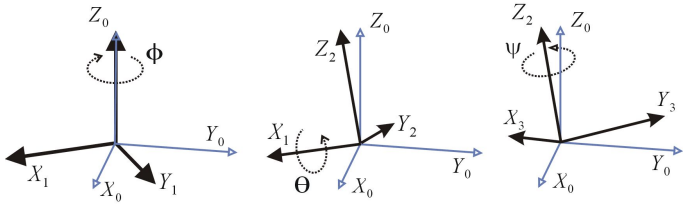
-> Widely used in software and robotics to define rotation.

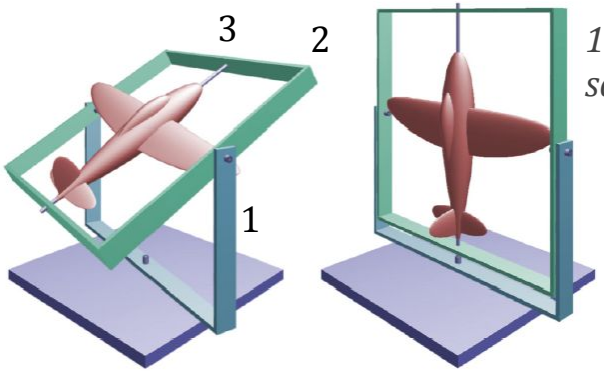


Representing rotation: Euler angle - limitations

Limitations:

- “Gimbal lock” when composing b/w rotations
- Non intuitive interpolated trajectory





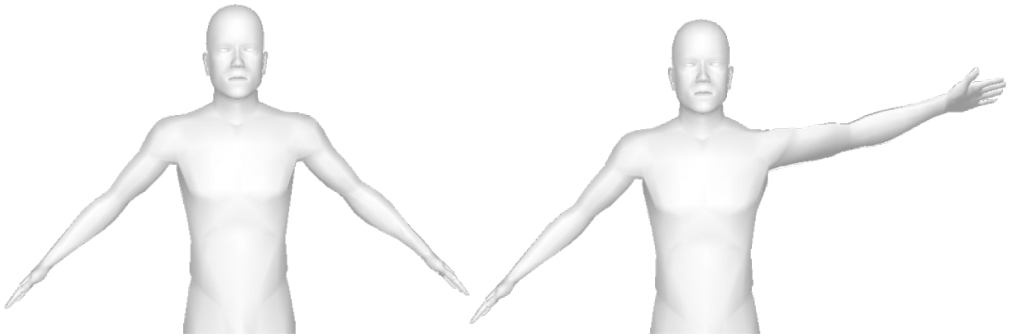
1 and 3 are the same

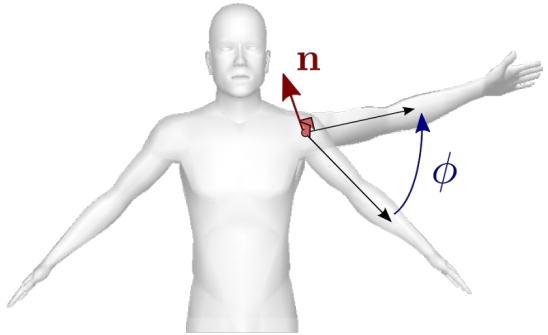
http://www.fho-emden.de/~hoffmann/gimbal09082002.pdf

Representing rotation: Axis angle

n: axis of rotation (unit norm)
ϕ: angle of rotation

3 degrees of freedom





Representing rotation: Axis angle

Applying rotation (n,ϕ) to a vector **v**

$$\mathbf{v}' = \cos(\phi)\,\mathbf{v} + \sin(\phi)\,\mathbf{n} \times \mathbf{v} + (1 - \cos(\phi))(\mathbf{n} \cdot \mathbf{v})\,\mathbf{n}$$

Rodrigues' rotation formula

Corresponding matrix

$$R(\mathbf{n}, \phi) = \begin{pmatrix} \cos(\phi) + n_x^2(1-\cos(\phi)) & n_x n_y(1-\cos(\phi)) - n_z \sin(\phi) & n_y \sin(\phi) + n_x n_z(1-\cos(\phi)) \\ n_z \sin(\phi) + n_x n_y(1-\cos(\phi)) & \cos(\phi) + n_y^2(1-\cos(\phi)) & -n_x \sin(\phi) + n_y n_z(1-\cos(\phi)) \\ -n_y \sin(\phi) + n_x n_z(1-\cos(\phi)) & n_x \sin(\phi) + n_y n_z(1-\cos(\phi)) & \cos(\phi) + n_z n_z(1-\cos(\phi)) \end{pmatrix}$$

But: Not easy to concatenate rotations

Representing rotation: Axis angle

Advantages:

- Concise and general representation
- Expressive parameters (axe-angle)
- Efficient rotation and correspondence with matrix

Limitation:

- No easy composition between rotations

Representing rotation: Quaternion

Generalization of complex numbers in 3D space

Difficulty: requires 4 parameters

$$q = x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k} + w$$

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$$
$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}$$
$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$
$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j}$$
$$\mathbf{ijk} = -1$$

Writing $q = (\mathbf{e}, w)$, where $\mathbf{e} = (x, y, z)$

$$q_1\, q_2 = (\mathbf{e}_1, w_1)\,(\mathbf{e}_2, w_2)$$
$$q_1\, q_2 = (w_1\, w_2 - \mathbf{e}_1 \cdot \mathbf{e}_2\,,\ w_1\mathbf{e}_2 + w_2\mathbf{e_1} + \mathbf{e_1} \times \mathbf{e_2})$$

Unit quaternion

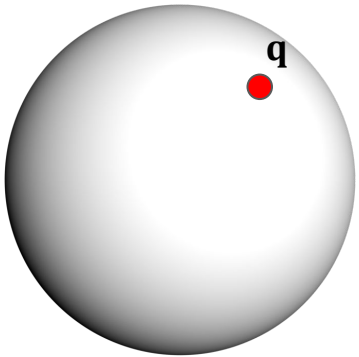
Rotation can be represented by q, with |q|=1

(similar to complex number in 2D)

point on the 4D unit sphere

$$q = \left(\mathbf{n} \sin\left(\frac{\phi}{2}\right), \cos\left(\frac{\phi}{2}\right) \right)$$

3 degrees of freedom



"Complex exponential" representation $q = e^{\frac{\phi}{2}\mathbf{n}}$

Applying q_1 q_2 is equivalent to compute R_1 R_2

Note: Generally $q_1 q_2 != q_2 q_1$

Unit quaternion

Convert q=(q_0,q_1,q_2,q_3) => Matrix R

$$\mathrm{R} = \begin{pmatrix} 1-2(q_1^2+q_2^2) & 2(q_0q_1-q_3q_2) & 2(q_0q_2+q_3q_1) \\ 2(q_0q_1+q_3q_2) & 1-2(q_0^2+q_2^2) & 2(q_1q_2-q_3q_0) \\ 2(q_0q_2-q_3q_1) & 2(q_1q_2+q_3q_0) & 1-2(q_0^2+q_1^2) \end{pmatrix}$$

Applying unit quaternion (/rotation) to a 3D vector v = v_x **i** + v_y **j** + v_z **k** "=quaternion (0,v)"

$$q' = q\, v\, \overline{q} \quad \text{with} \quad \overline{q} = (-x, -y, -z, w)$$

Interpolating rotation

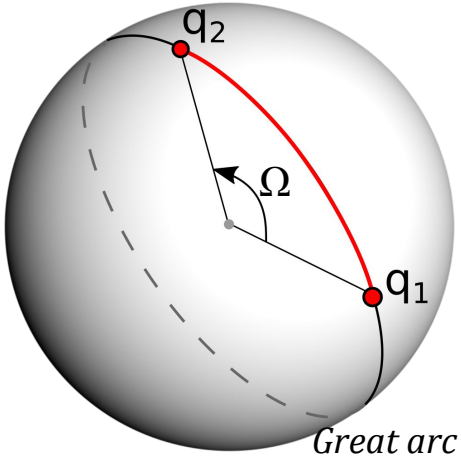
Spherical Linear intERPolation (**SLERP**)

$$q(\alpha) = q_1 \left(q_1^{-1} q_2\right)^{\alpha} \qquad \alpha \in [0,1]$$

$$q(\alpha) = \frac{\sin\left((1-\alpha)\Omega\right)}{\sin(\Omega)} q_1 + \frac{\sin(\alpha\Omega)}{\sin(\Omega)} q_2$$

$$\cos(\Omega) = q_1 \cdot q_2$$

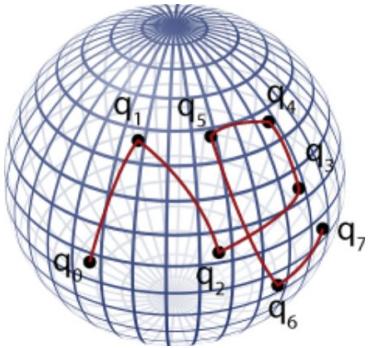
Constant angular velocity

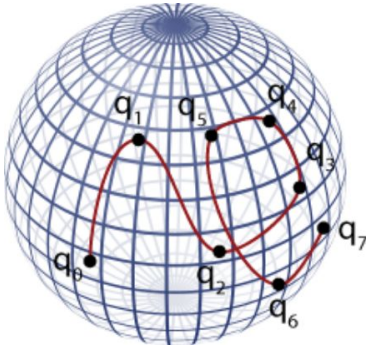


Interpolating rotation

Generalized to curves on the 4D sphere

[Ken Shoemake, SIGGRAPH 85. Animating Rotation with Quaternion Curves]





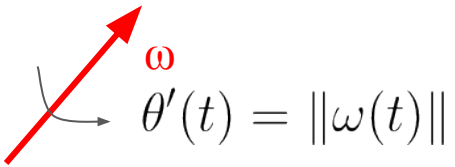
Bezier/Spline generalization to quaternions
Use of approximate normalized linear interpolation (NLERP)
(lost of constant angular speed)

$$q = \frac{\sum_i \alpha_i \; q_i}{\| \sum_i \alpha_i \; q_i \|}$$

Quaternion and angular velocity

Side note: Quaternion derivative naturally express the angular velocity

$$q'(t) = \frac{1}{2}\,\omega(t)\,q(t)$$

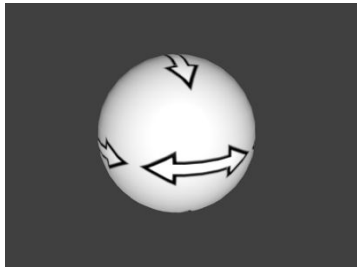


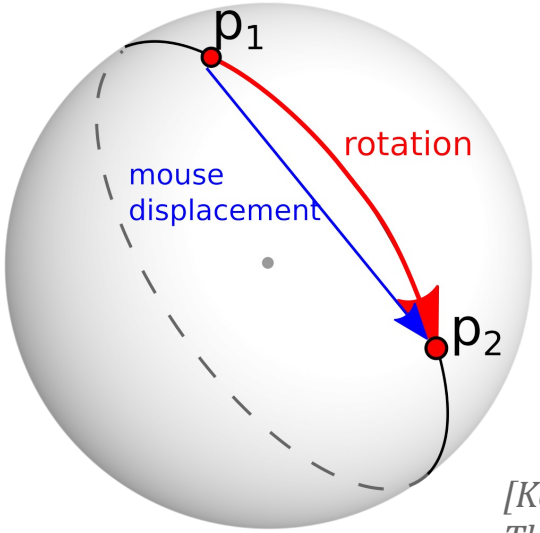
Useful for rigid body dynamics, express the angular momentum.
ex. Kinematics energy of a rotating body around a fixed axis given by $E_c = \frac{1}{2}\,\mathrm{I}\,\|\omega\|^2$

Application to Arcball camera motion

Spherical coordinates -> locking at the poles

Idea: build rotation between two points as if we were clicking on points of a sphere





- project point click (x_1,y_1) (x_2,y_2) onto a 3D sphere p_1, p_2

$$p = (x, y, \sqrt{1 - x^2 - y^2})$$

- compute rotation between p1 and p2

[Ken Shoemake. ARCBALL: A User Interface for Specifying Three-Dimensional Orientation Using a Mouse. 92]

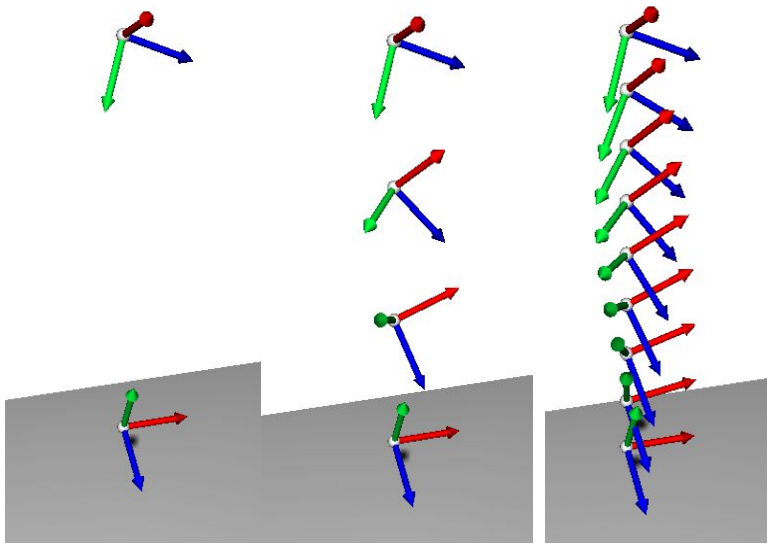
Interpolation of rigid transformation

Rigid transformation:

Two objects describes as (position,orientation)

- Linear interpolation of the position
- SLERP on the rotation

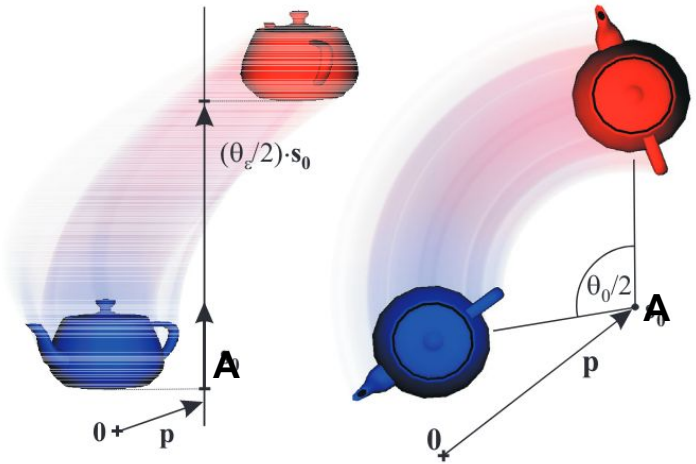
De-couple translatation & orientation



Rigid transformation: Screw motion

Specific motion with

- minimal angle rotation around a constant axis A (not passing by O)
- translation along the same axis A



[L. Kavan et al.,
Dual Quaternions for Rigid Transformation, 2006]

- *represent a twisting moving object around an axis*
- *also used to represent the force and torque (wrench) applied on a rigid mechanical object*

Dual quaternion

Extension of quaternion to handle rigid transformation

Naturally represents screw motion

$\hat{q} = q_0 + \epsilon q_\epsilon$ where ϵ is a dual number : $\epsilon^2 = 0$

q_ϵ is the dual part

ϵ similar to modeling an infinitesimal quantity
(also used for automatic differentiation)

Dual quaternion

Similar properties that quaternions

$$\hat{q} = \cos(\hat{\phi}/2) + \hat{n}\,\sin(\hat{\phi}/2)$$

$$\hat{\phi} = \phi_0 + \epsilon\phi_\epsilon$$

amount of translation

angle of rotation

$$\hat{n} = n_0 + \epsilon n_\epsilon$$

moment of axis (/position)

axis of rotation/translation

Dual quaternion

Converting (quaternion q_0,translation t) to dual quaternion

$$\hat{q} = q_0 + \frac{\epsilon}{2} t\, q_0 \quad \text{with } t = i\, t_x + j\, t_y + k\, t_z$$

Interpolating b/w dual quaternions: ScLERP (Screw Linear Interpolation)