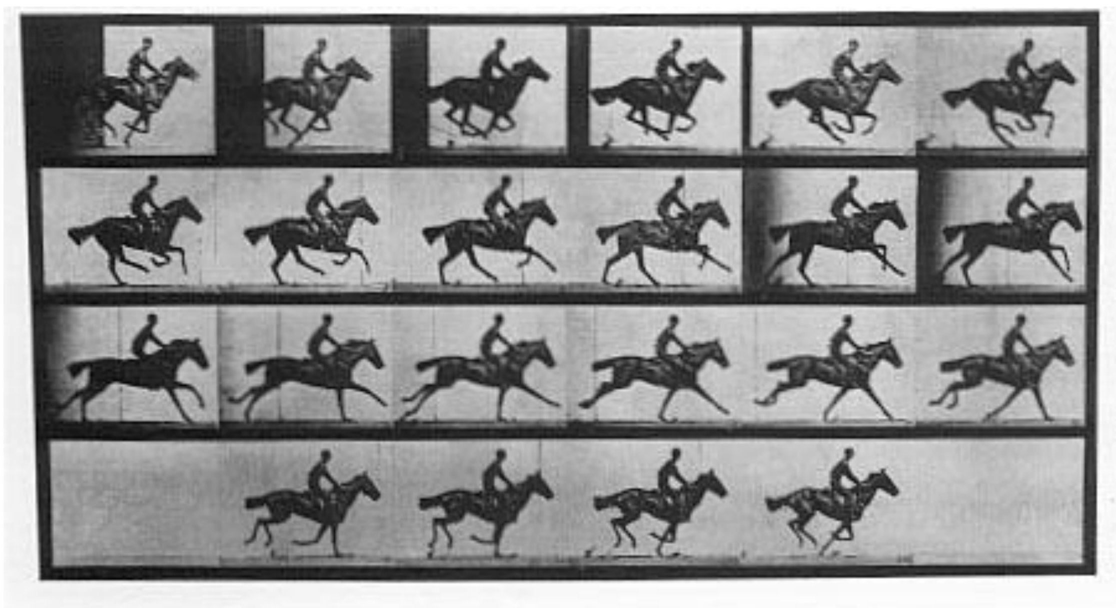
# Chapter 3 : Principles of Animation

# Animation

- “Animate” literally means “bring to life”
- To do so, the animator needs to define how objects move through space and time



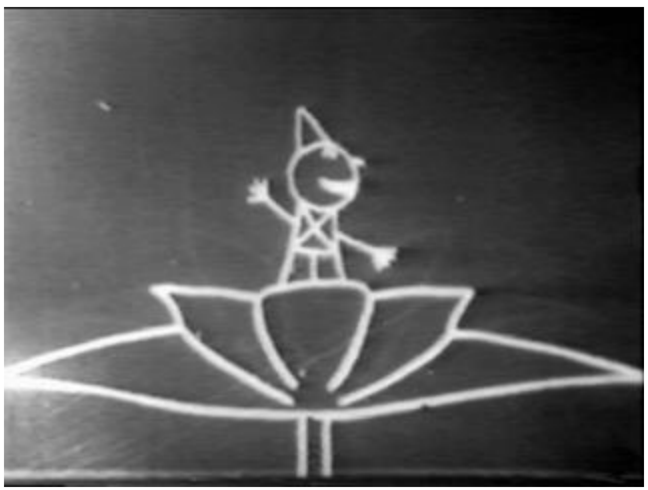
Muybridge 1887

# Frame

- A series of related still-images is interpreted as motion by human brain
- One minute of animation: 720-1800 frames

# History

- First animation films (Disney)
  - 30 drawings / second
  - animator in chief : “**key frames**”
  - others : secondary drawings



Frame from Fantasmagorie, E. Cohl (1908)

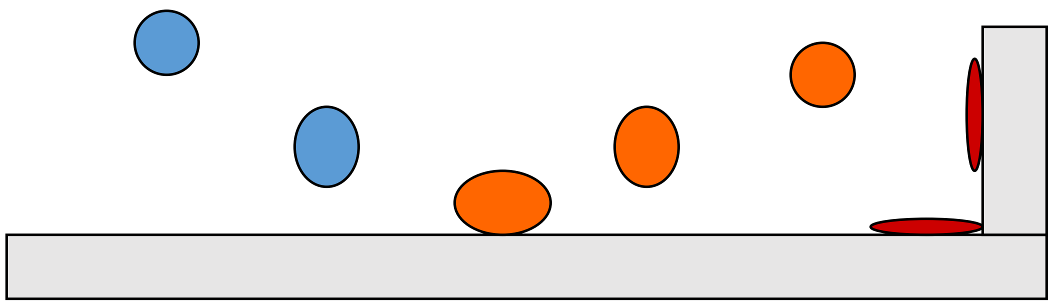

***« Descriptive animation »***

The animator fully controls motion

# Keyframe-based animation

- Option: use the computer to interpolate
  - positions (e.g. center of ball)
  - orientations (e.g. of ellipse)
  - shapes (e.g. aspect ratio of ellipse)

- Or give the trajectory (of position) explicitly

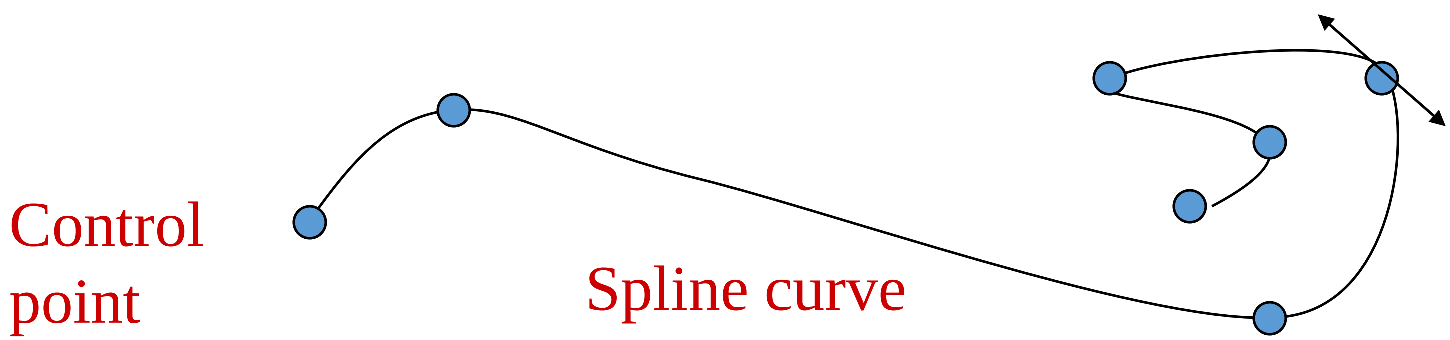


# Interpolating Positions

## Possible using splines curves

Interpolation: Hermite curves or Cardinal splines

- Local control
  - Made of polynomial curve segments
  - degree 3, class $C^1$

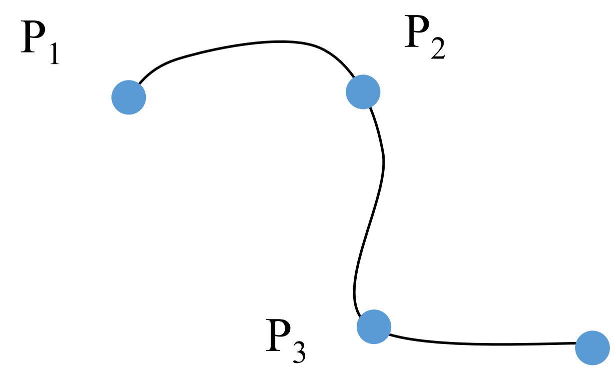


# Hermite Curves of Order 1

- Piecewise segments of degree 3 of the form

$$x(t) = a_3t^3 + a_2t^2 + a_1t + a_0$$
$$y(t) = b_3t^3 + b_2t^2 + b_1t + b_0$$
$$z(t) = c_3t^3 + c_2t^2 + c_1t + c_0$$

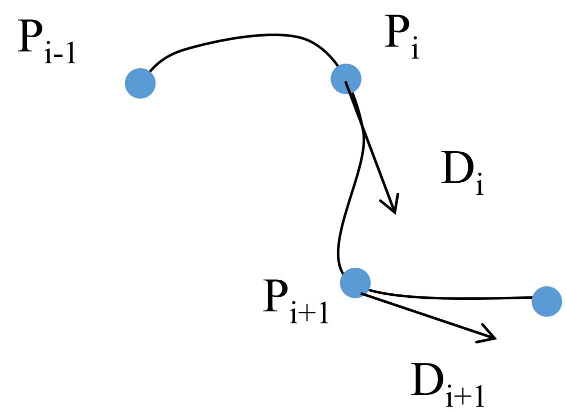
- 12 degrees of freedom per segment
- Order 1 ($C^1$) transition between segments



# Hermite Curves of Order 1

- Each curve segment defined by:
  $C_i(0) = P_i$
  $C_i(1) = P_{i+1}$
  $C'_i(0) = D_i$
  $C'_i(1) = D_{i+1}$

- Advantage: local control

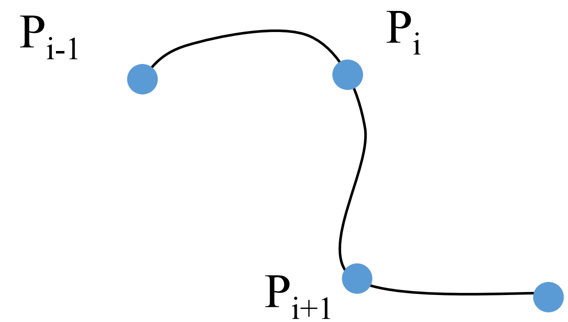



**Exercise:** how to ease the definition for general users?
Propose an automatic way to compute tangents.

# Hermite Curves of Order 2

- Degree 3, Order 2 ($C^2$).

$$C_{i-1}(1) = P_i$$
$$C_i(0) = C_{i-1}(1)$$
$$C'_i(0) = C'_{i-1}(1)$$
$$C''_i(0) = C''_{i-1}(1)$$




- Can be solved by adding tangent constraints at the extremities
- Problem: Global definition only! (costly & no local control)

# Cardinal Splines

Degree 3, Order 1 ($C^1$).
Each curve segment defined by

$$C_i(0) = P_i$$
$$C_i(1) = P_{i+1}$$
$$C'_i(0) = k\,(P_{i+1} - P_{i-1})$$
$$C'_i(1) = k\,(P_{i+2} - P_i)$$

*Exercise*
Order of locality?
What is the effect of k?
How can we model a closed curve?

Catmull-Rom
Cardinal with tension k = 0.5

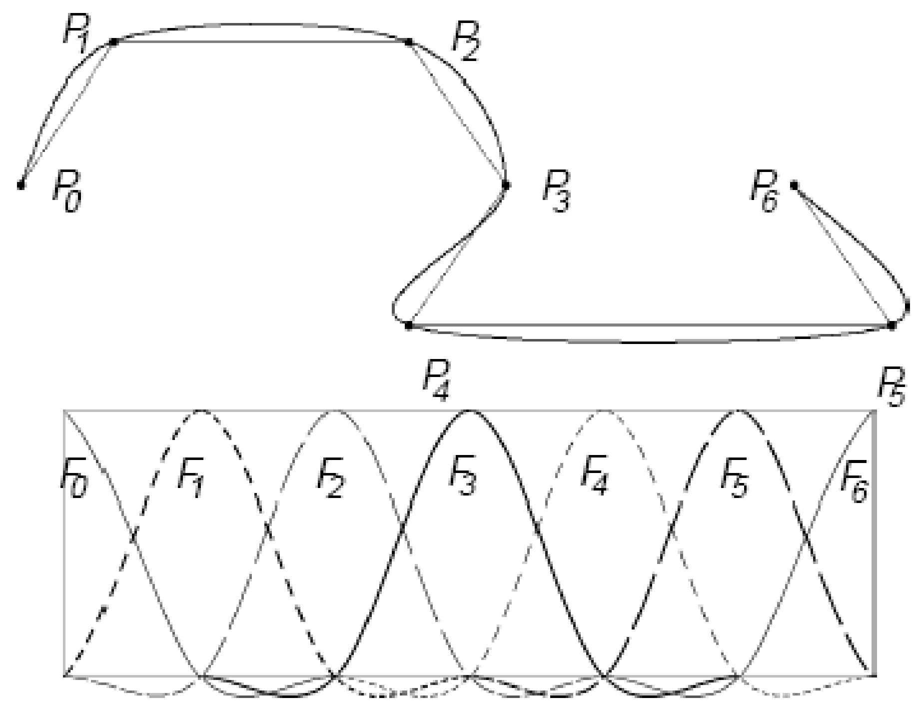
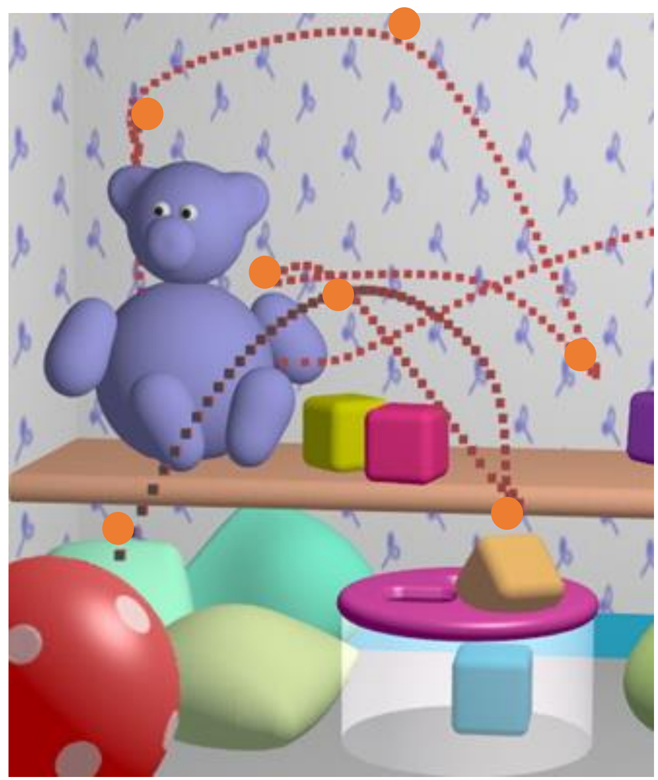



**Figure 2:** *Catmull-Rom spline curve*

# Interpolating Positions

***Exercise***

- Goal: animate a bouncing ball
  - Describe a method for computing the trajectory from the control points.
  - How would you animate  the changes of speed?
  - What is missing in this kinematic animation in terms of realism?



# Interpolating Positions

- Interpolating key positions
  - Interpolation curves

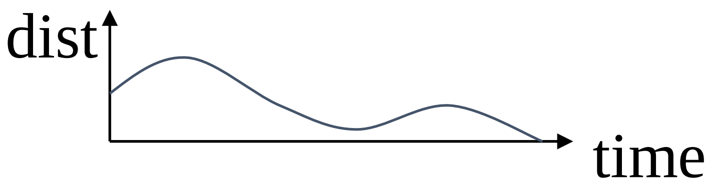
  Enable inflection points!

  (where $C^0$ only)

  - Speed control:

  Reparamterize the trajectory

  « velocity curve »

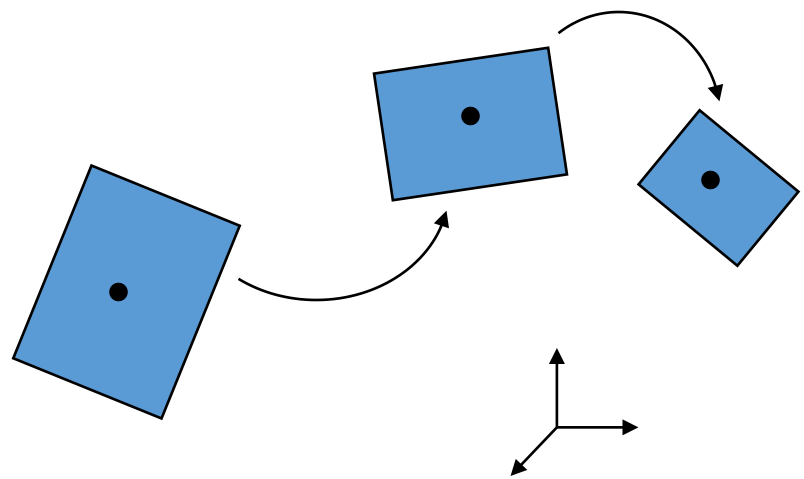



# Interpolating Orientations

- Interpolation of orientations

  Choose the right representation !

  - Rotation matrix ?
  - Euler angle ?
  - Quaternion ?



# Rotation matrix

- Representation : orthogonal matrix
  - each orientation = 9 coefficients
- Interpolation :
  - Interpolate coefficients one by one
  - Re-orthogonalize and re-normalize

Costly and inappropriate :

$M = k\, M_1 + (1\text{-}k)\, M_2$ can be degenerate

Impossible to approximate it by an orthogonal matrix in this case

Exemple:

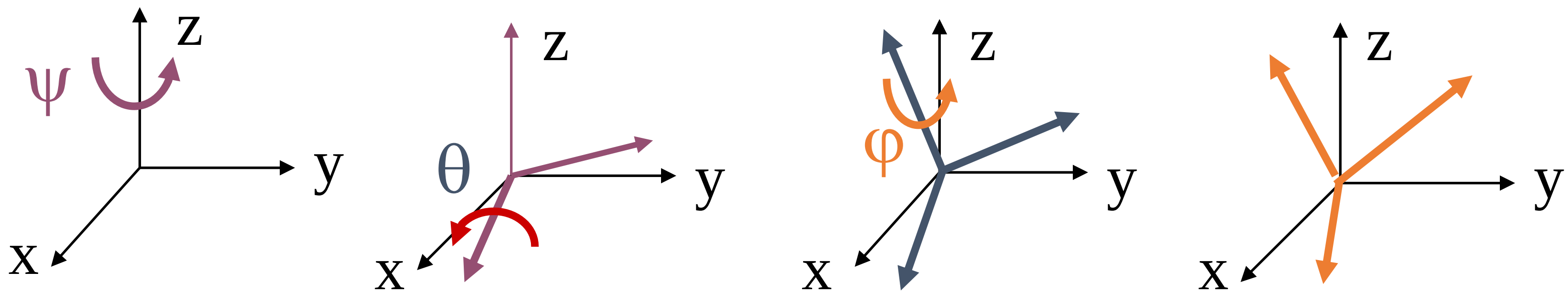
Axis x, angle α

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

***Exercise:***

$M_1$ = Id
$M_2$ = rotation
x axis, α = π
M for k=0.5?

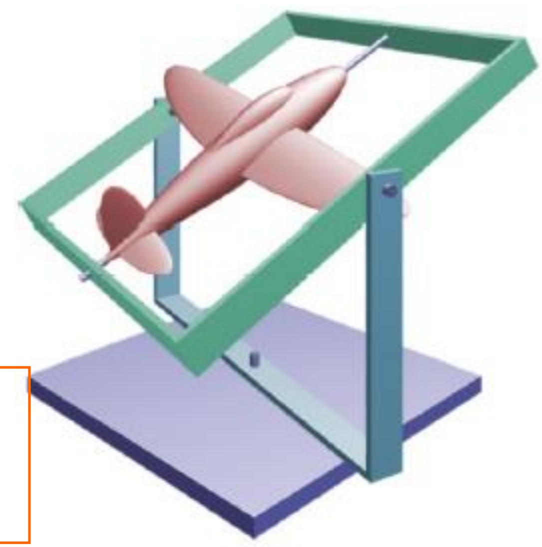
# Euler Angles




**Representation :**

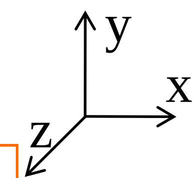
- Three angles (ψ, θ, φ)
- Intuitive : $R(V) = R_{z,\varphi}(R_{x,\theta}(R_{z,\psi}(V)))$

« Roll, pitch, yaw »
in flight simulators

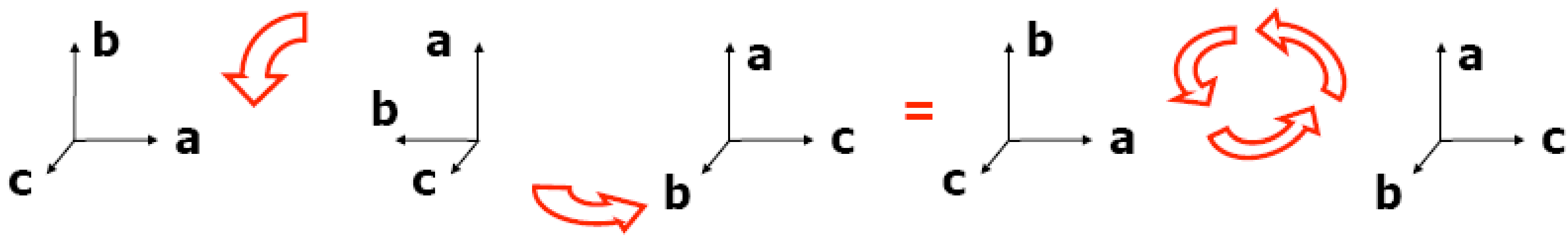


# Interpolating Euler Angles

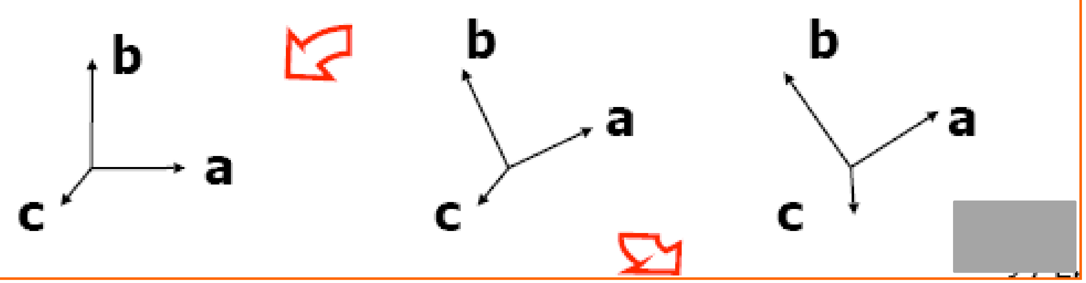
- \+ more efficient : 3 values for 3 Degrees of Freedom (DoF)
- \- non-invariant by rotation, and un-natural result




rotation of 90° around Z, then 90° around Y = 120° around (1, 1, 1)

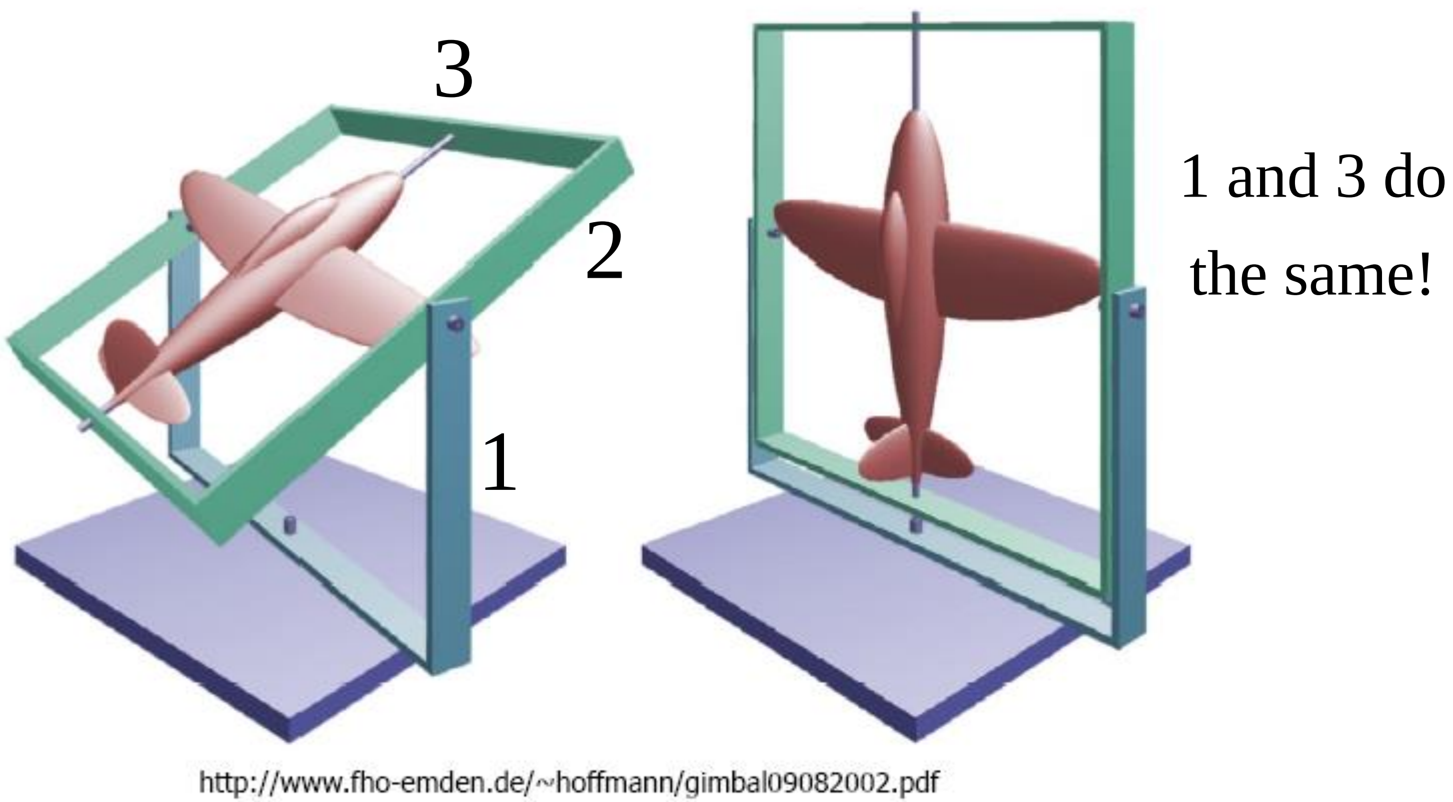



But rotation of 30° around Z then 30° around Y ≠ 40° around (1, 1, 1)



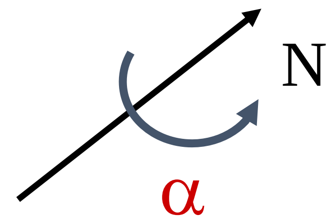
# Problem with Euler Angles: gimbal lock

- Two or more axes aligned = loss of rotation DoF



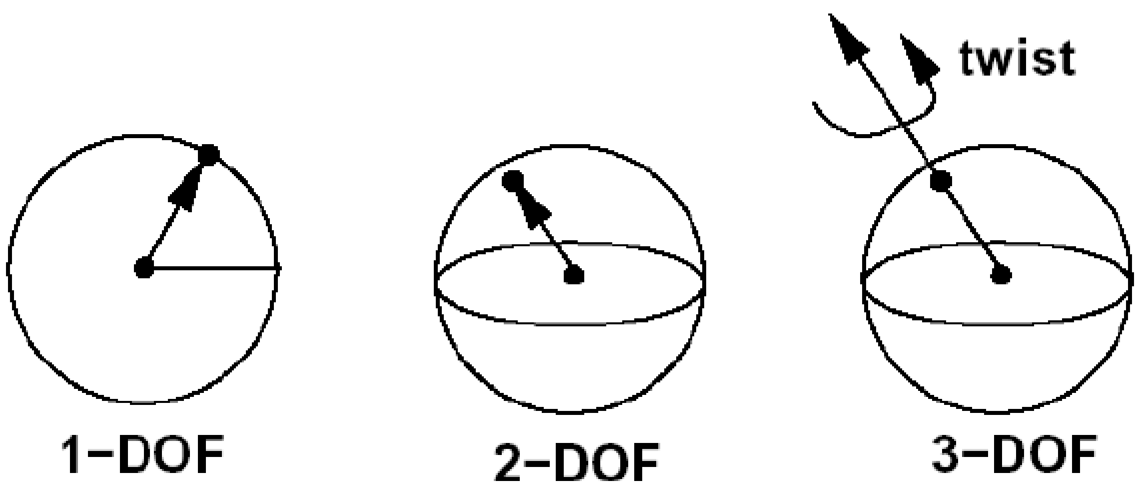
# Quaternions

Representation : $q = (\cos(\alpha/2), \sin(\alpha/2)N) \in S^4$




By analogy:

1, 2, 3-DoF rotations as points on 2D, 3D, 4D spheres



# Quaternions

*Algebra of quaternions*

- Generated by (1,i,j,k)

  neutral element: (1,0,0,0)
  $i^2 = j^2 = k^2 = ijk = -1, \quad 1^2 = 1$
  $ij = -ji = k$
  $jk = -kj = i$
  $ki = -ik = j$

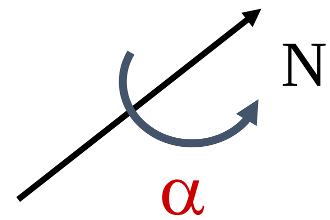
- Notation $q = (q_{r,}\ q_p)$ where $q_p \in R^3$

  $p \,.\, q = (p_r\, q_r - p_p\, q_p\ ,\ p_r\, q_p + q_r\, p_p + p_p \wedge q_p)$

  $q^{-1} = (q_{r\,,}\ -q_p) \,/\, (q_r^{\,2} + q_p.q_p)$

# Quaternions

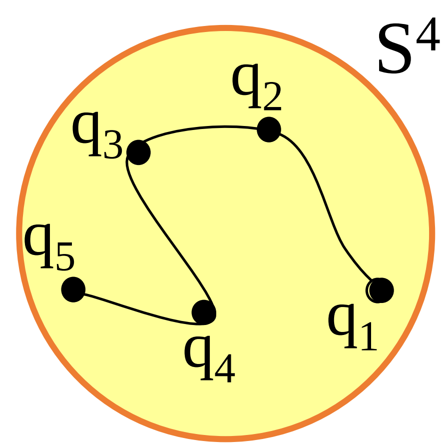
*Used to represent rotations*




Rotation ($\alpha$, N): $q = (\cos(\alpha/2), \sin(\alpha/2)\, N)$

- Unit quaternion $\in S^4$
- Apply a rotation

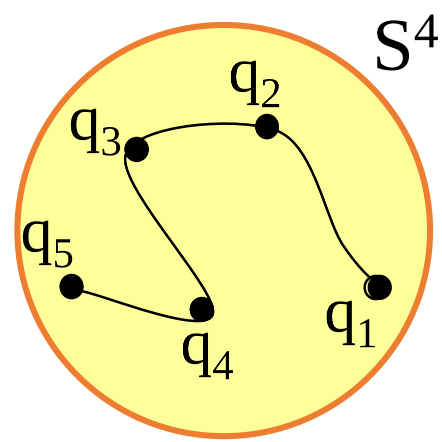
  $R(V) = q \,.\, (0,V) \,.\, q^{-1}$
- Compose two rotations : $p \,.\, q$

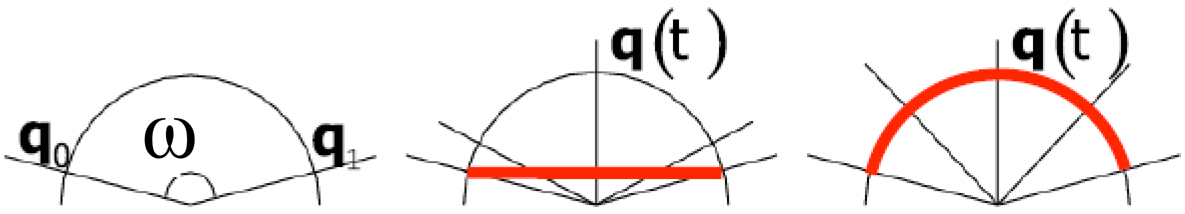


# Quaternions

- Interpolate quaternions? : splines on $S^4$

- Interpolation method?




non-uniform speed!

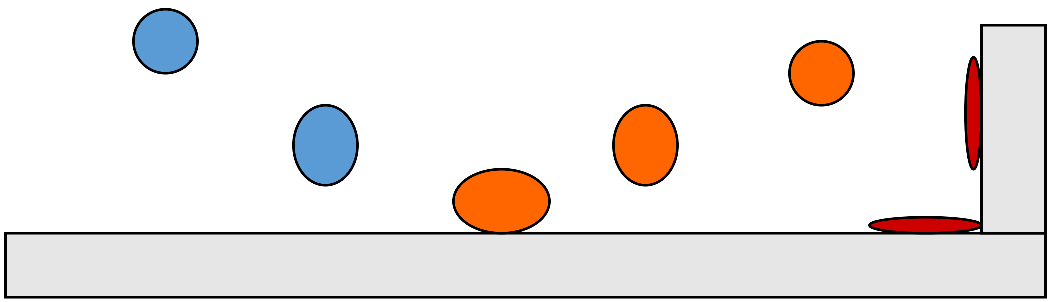
  - Use spherical!

$$\mathrm{lerp}(\mathbf{q}_0,\mathbf{q}_1,t)=\mathbf{q}(t)=\mathbf{q}_0(1-t)+\mathbf{q}_1 t$$

$$\mathrm{slerp}(\mathbf{q}_0,\mathbf{q}_1,t)=\mathbf{q}(t)=\frac{\mathbf{q}_0\sin((1-t)\omega)+\mathbf{q}_1\sin(t\omega)}{\sin(\omega)}$$

# Interpolating Shape

- Difficult in general
- Simple cases: parameterize shape (e.g. using major axes for ellipses) and interpolate parameters



- Complex cases:
  - Simple method: sample shapes with keypoints and interpolate their positions
  - More complex methods: in advanced geometry lecture

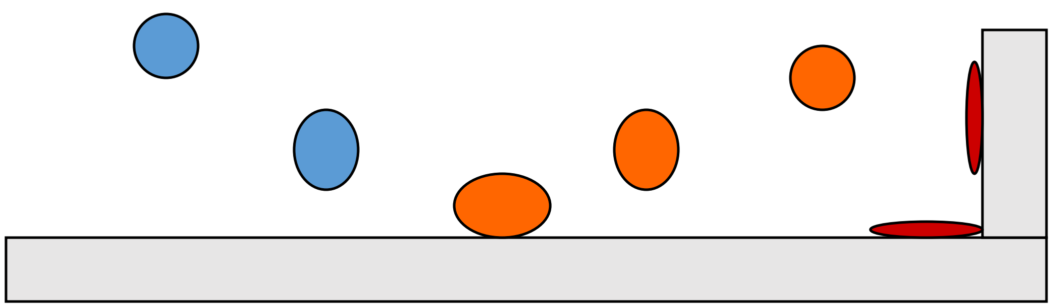
# Descriptive Models

## Animate Deformations

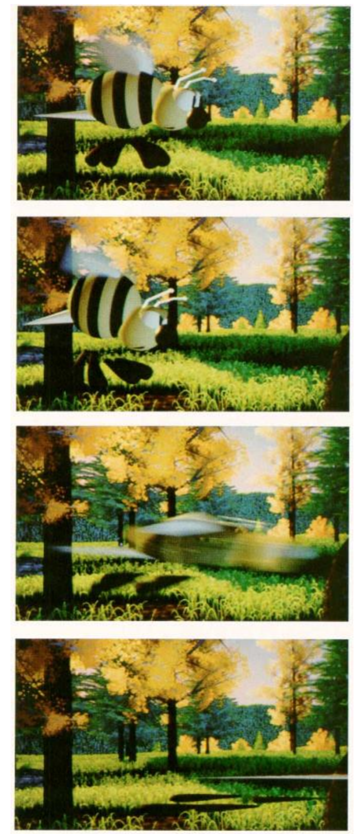
Interpolate « key shapes »

- Example : « Disney effects»
  - Change scaling, color…

$$k\,(u) = (u^3\, u^2\, u\ 1)\ M_{spline}\ [k_{i-1}\ k_i\ k_{i+1}\ k_{i+2}]^{\,t}$$



*[Lasseter 1987]*



# Descriptive Models
# Animate Deformations

Animate a geometric model

= animate its parameters

***Exo:*** Propose methods to design and animate this bee with “Disney effects” including:

- squash & stretch
- anticipation

# A Note on Timing

- Basic animation rendering loop could look like this

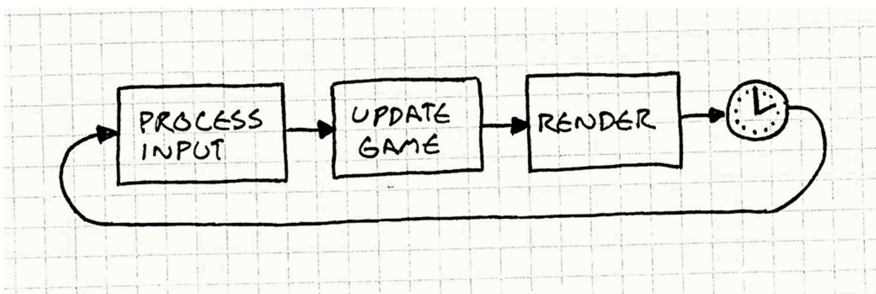
```
while(true)
{
        processInput() // if applicable take user input into account
        update()
        render()
}
```

- Problem: timing of animation depends on processing power of computer running it

# A Note on Timing

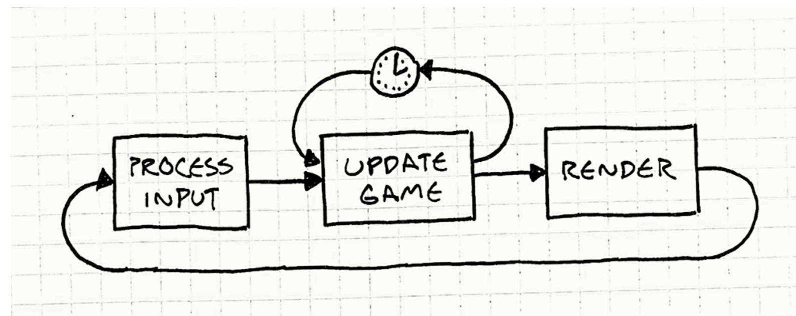
Option 1:

- Take a nap until it is time
- Works when machine is too fast (compared to animation time)




Option 2:

- Adapt what is rendered
- Works when machine is too fast or too slow



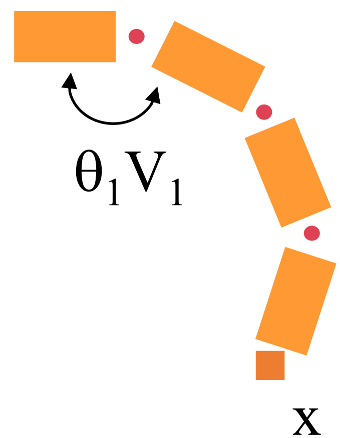

Figures from http://gameprogrammingpatterns.com.

# Hierarchical Animation

# Hierarchical structures

They are essential for animation!

- Eyes move with head
- Hands move with arms
- Feet move with legs…




- Frame hierarchy
  - Root expressed in the world frame (translation + r
  - Relative rotation with respect to the parent

# Hierarchical structures

Generalized coordinates

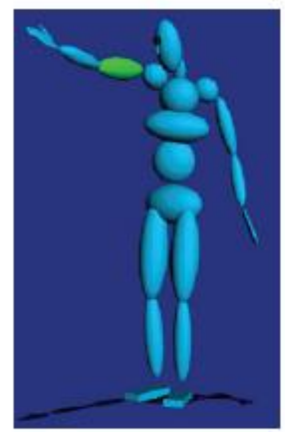
- Vector of degrees of freedom (DoF) at each joint

Example

**1 DOF: knee**

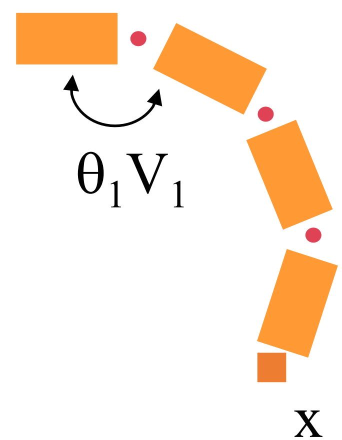
**2 DOF: wrist**

**3 DOF: arm**



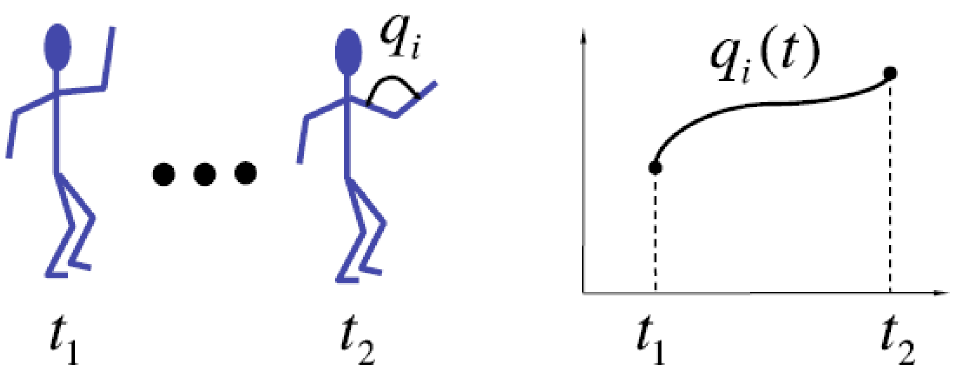
# Hierarchical structures

- To compute composite transformation
  - Put matrices in order of hierarchy on a stack
  - Multiply matrices to obtain composite transformation
- Quaternions
  - Typically transformed into matrix first
  - Not strictly necessary



# Direct Animation with Forward Kinematics

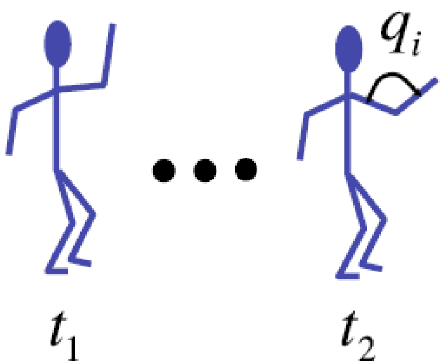
Method: Interpolate key rotations

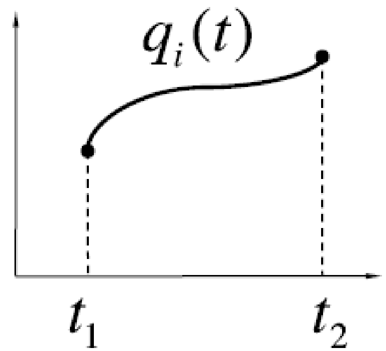



***Exercise :*** Controlling a cycling motion

- Define key-rotations over time
- What is the main difficulty?
- What would be the extra problem for a walking motion?

# Forward kinematics




Conclusion:

- Difficult to control extremities!

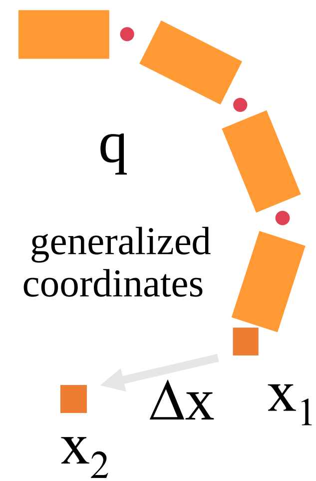
    (example : foot position while cycling)

- In practice: Top-down set-up method
    - Try to compensate un-desired motions!

# Advanced method: Inverse kinematics

- Control of the end of a chain
  - Automatically compute the other orientations ?

    $x_1 = f(q)$ $x_2 = f(????)$




## Method from robotics

- Local inversions of a non-linear system

  $\Delta x = J\,\Delta q$, with $J_{ij} = \dfrac{\partial x_i}{\partial q_j}$ Jacobian matrix
- Under-constrained system, pseudo-inverse : $J^+ = J^t (J\,J^t)^{-1}$

**Exo:** Show that ($\Delta q = J^+ \Delta x$) and ($\Delta q = J^+ \Delta x + (I-J^+J)\,\Delta z$) are solutions.

What can $\Delta z$ (called "secondary task") be used for ?