CS 428: Fall 2010

Introduction to Computer Graphics

Computer animation

Animation

A brief history

- 1800s – Zoetrope
- 1890s – Start of film animation (“cells”)
- 1915 – Rotoscoping
  - Drawing on cells by tracing over live action
- 1920s – Disney
  - Storyboarding (for story review)
  - Camera stand animation (parallax etc.)

Animation

A brief history

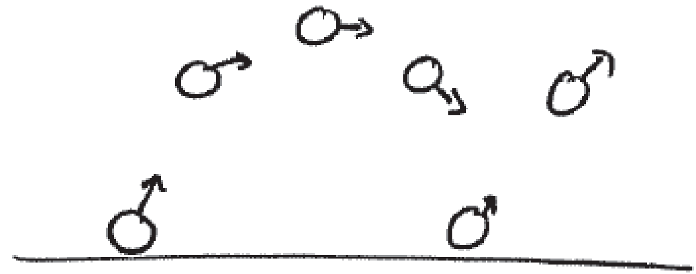
- 1960s – Early computer animation
- 1986 – Luxo Jr.
- 1987 – John Lasseter's SIGGRAPH article
  - Applying traditional animation to CG animation (squash, stretch, ease in-out, anticipation, etc.)
- Before this
  - Tron (1982), Star wars (1977), etc.
- After this: artists needs became important!
  - Artists need a way of defining motion

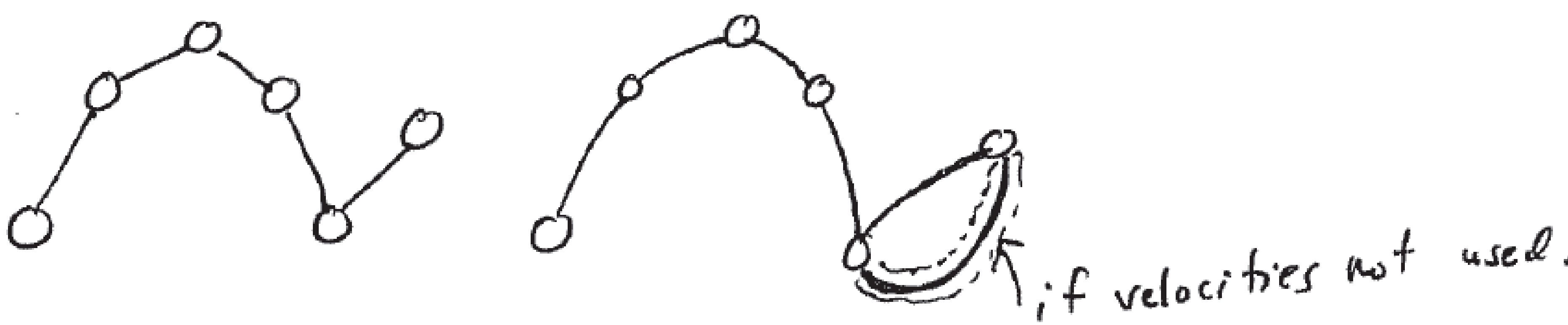
Interpolation

- Interpolation of
  - Object/world geometry (positions)
  - Object/world parameters (angles, colors)
  - Object/world properties (lights, time of day)
- But what to interpolate between?
- Basic idea: **keyframe interpolation**
  - Sparse specification of *key* moments of an animation sequence

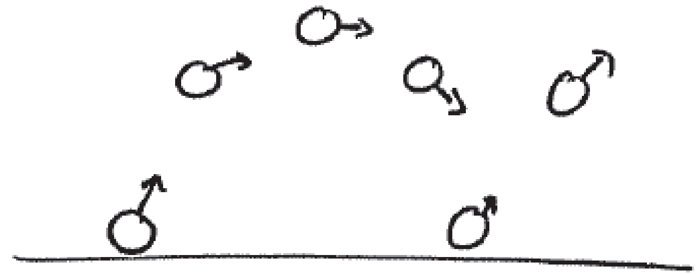
Keyframe interpolation

- Position / configuration
- Time of event
- Optional: velocity, acceleration, etc.
- Generate “in betweens” automatically
  - Interpolated motion paths are not unique

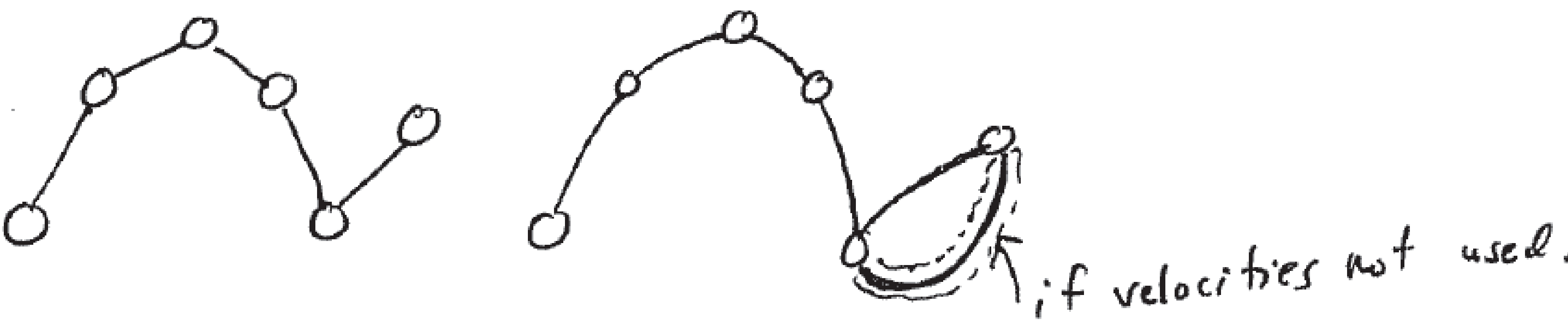




Keyframe interpolation



- Position / configuration
- Time of event
- Optional: velocity, acceleration, etc.
- Generate "in betweens" automatically
  - Linear and/or splines (keyframes at the knots)



Interpolation examples

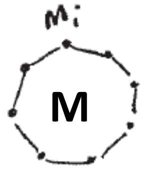
- **Tweening** – interpolate from one mesh to another with some mesh connectivity

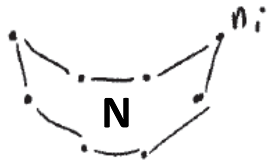
  - Interpolate vertices

$$A = M \cdot (1-t) + N \cdot (t)$$

$$a_i = m_i\,(1-t) + n_i\,(t)$$

↑ not time





Interpolation examples

- **Time warping** – adjust time to influence anim

M at T_1

N at T_2

$$f(t) = T_1 + (T_2 - T_1)\,t$$

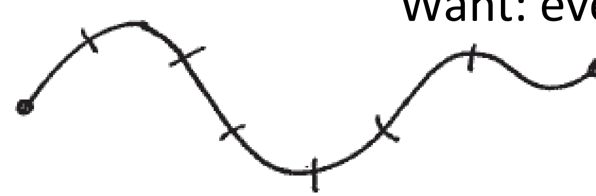
$$f(0) = T_1 \qquad f^{-1}(T_1) = 0$$

$$f(1) = T_2 \qquad f^{-1}(T_2) = 1$$

$$A(T) = M \cdot \left(1 - f^{-1}(T)\right) + N \cdot f^{-1}(T)$$

- Perhaps use a spline to represent f
  - Gives animator more control
  - Move knots for an arc-length parameterization

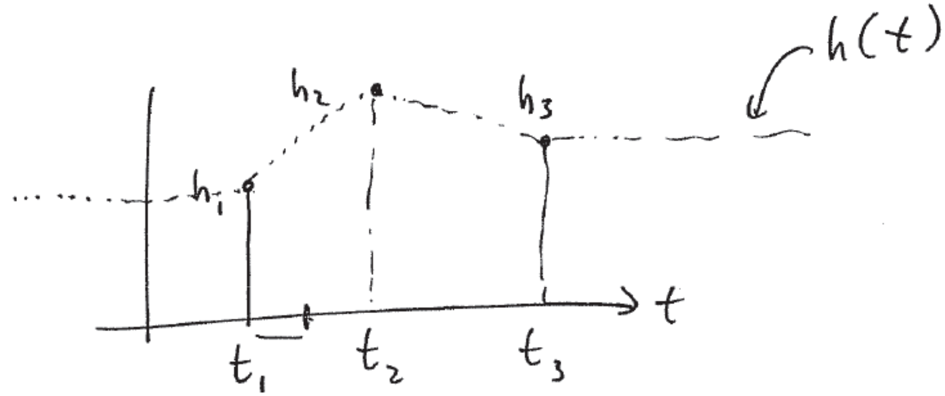
Have: evenly spaced t



Want: evenly spaced T

Interpolation examples

- Simple linear interpolation in (pseudo) code




```
double h(double t)
    if (t < t1)
        return h1
    if (t < t2)
        return h1 + (t - t1)/(t2 - t1) (h2 - h1)
    if (t < t3)
        return h2 + (t - t2)/(t3 - t2) (h3 - h2)
    return h3
```

Interpolating parameters

$s = 0$ $s = 1$ $s = -1$

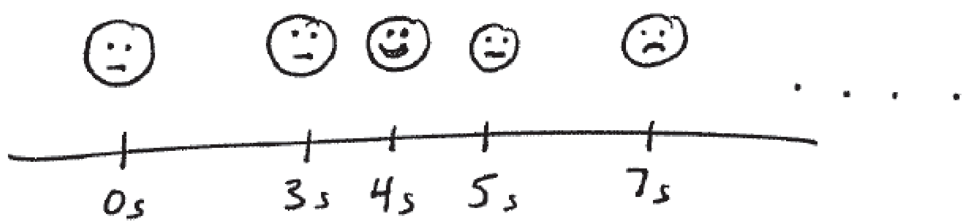
- Interpolate s as before
- Interpolating rotation angles can be tricky
  - Euler angles $R_x(\theta_x) R_y(\theta_y) R_z(\theta_z)$
  - Counterintuitive + erratic for distant keyframes
  - Use quaternions instead

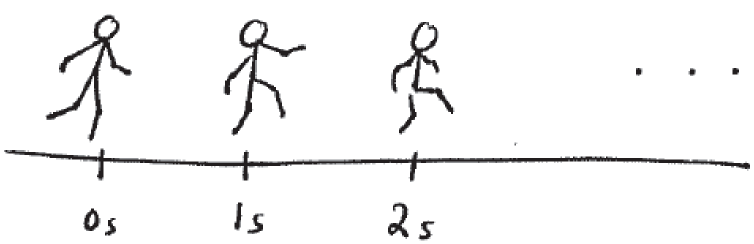
$[s, a_x, a_y, a_z]$ where $s^2 + a_x^2 + a_y^2 + a_z^2 = 1$

User interfaces for keyframes

- time lines

not necessarily pictures, but rather text descriptions or function plots



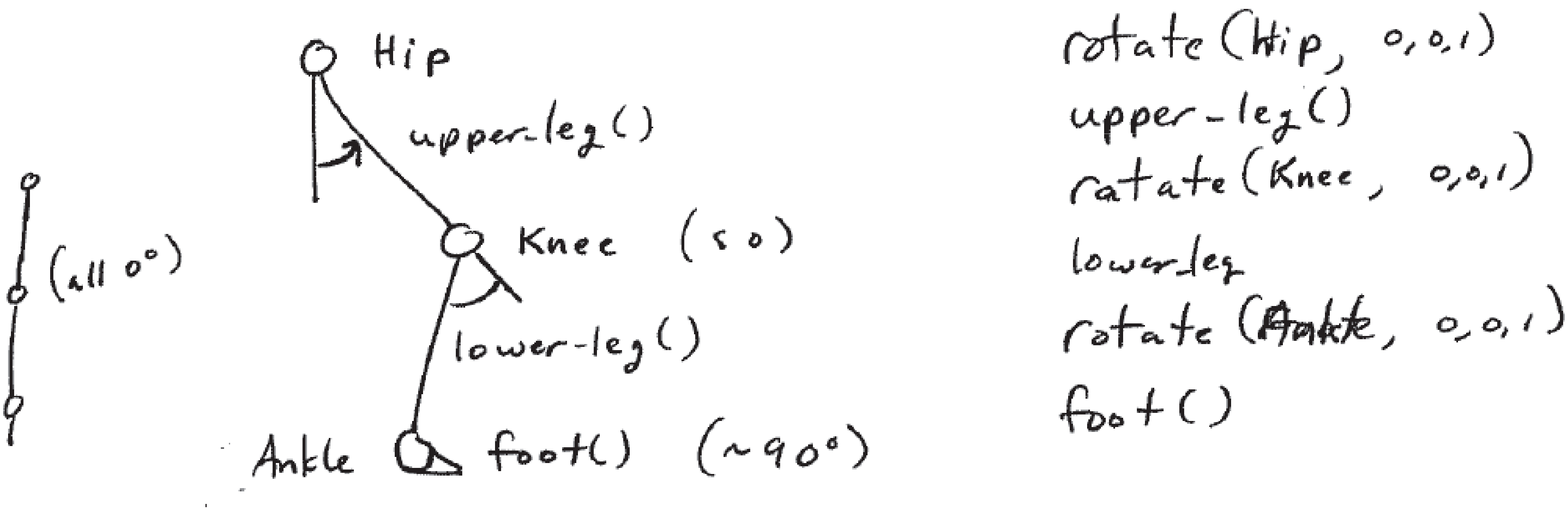


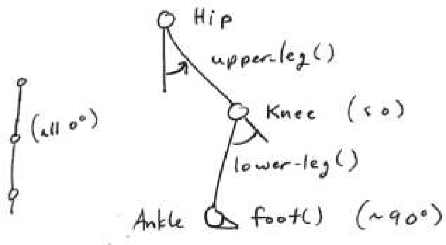
Typically 1 in 5 frames (in 30 frames/sec) are keyframes

→ more than cell animation, since no human there with good judgement

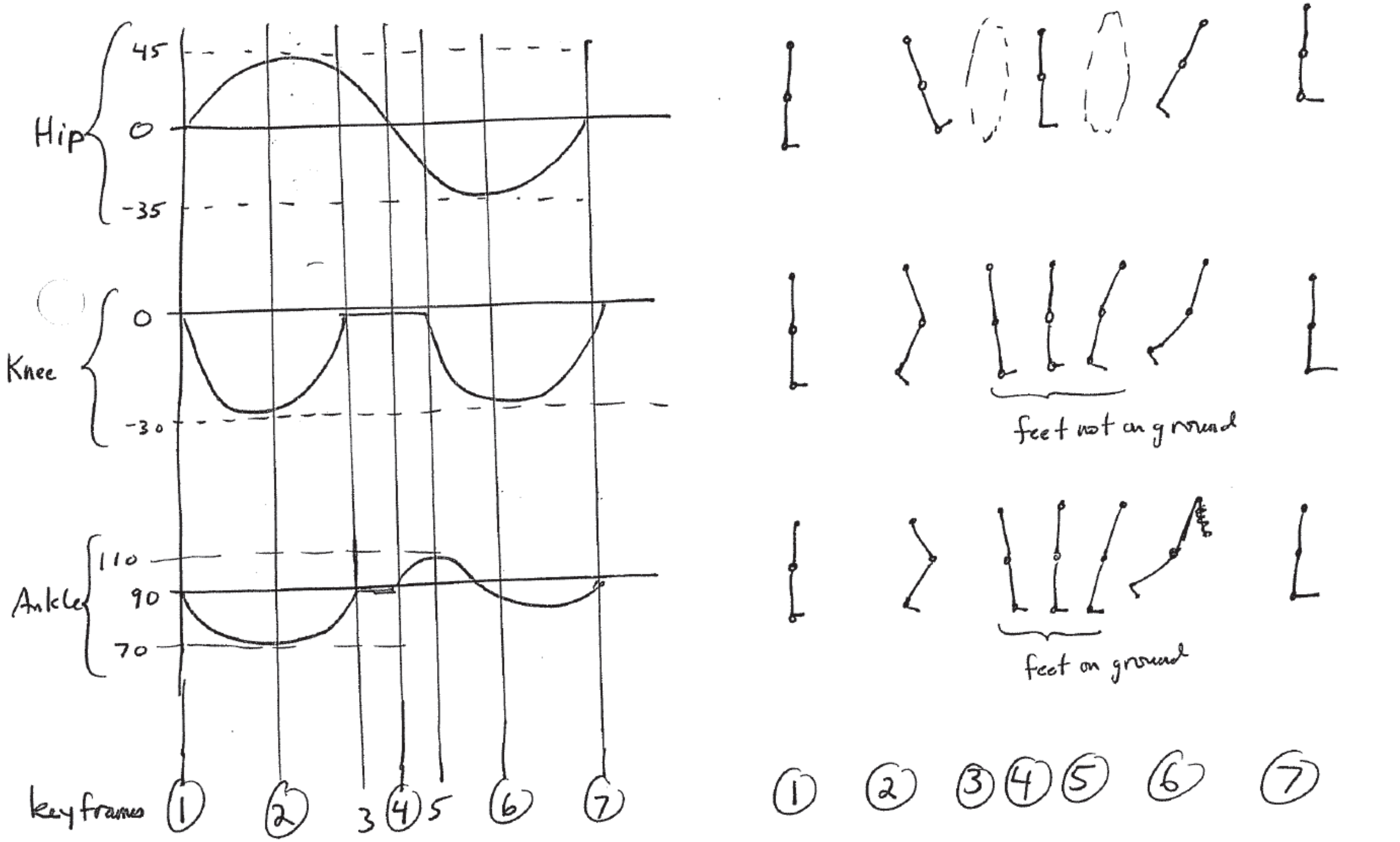
Timelines /w function plots

- Leg example



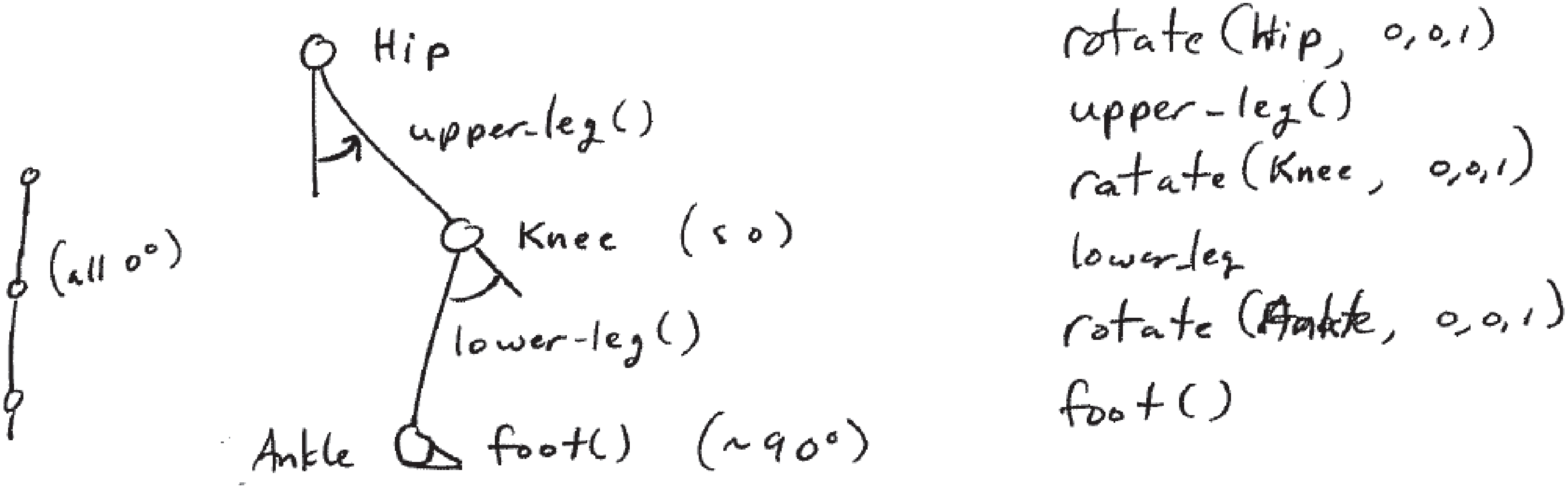


Timelines /w function plots



Timelines /w function plots

- A lot of work!
- Even worse: ankle depends on hip + knee!
- Kinematics: animation/w motion parameters (pos, vel, accel). No reference to forces



Physically based animation

- Each moving object is a point in a force field
  - Position and velocity
  - **Acceleration**: computed from the environment and integrated over time to determine pos + vel

$$\frac{d\begin{pmatrix} v \\ x \end{pmatrix}}{dt} = \begin{pmatrix} a \\ v \end{pmatrix}$$

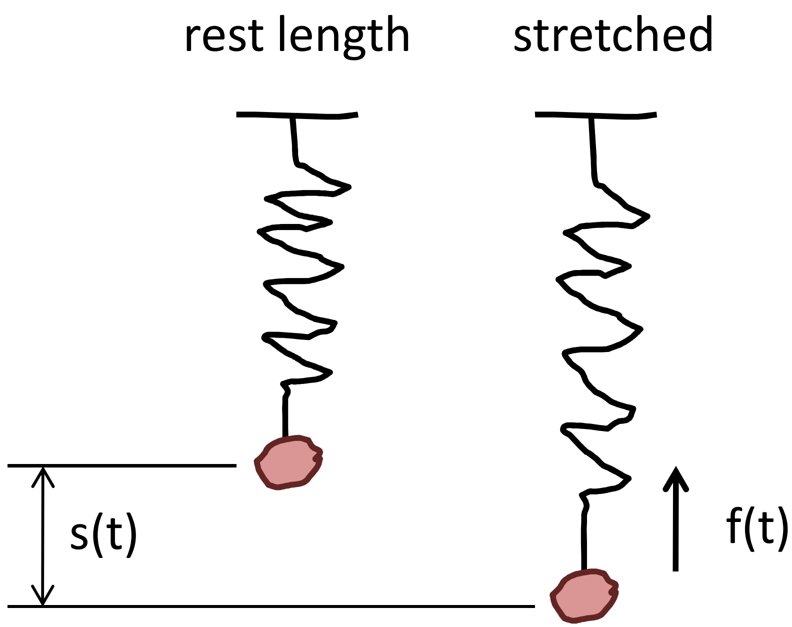
Euler integ (given $a(t)$)

$$v(t+\Delta t) = v(t) + a(t)\,\Delta t$$
$$x(t+\Delta t) = x(t) + v(t)\,\Delta t$$

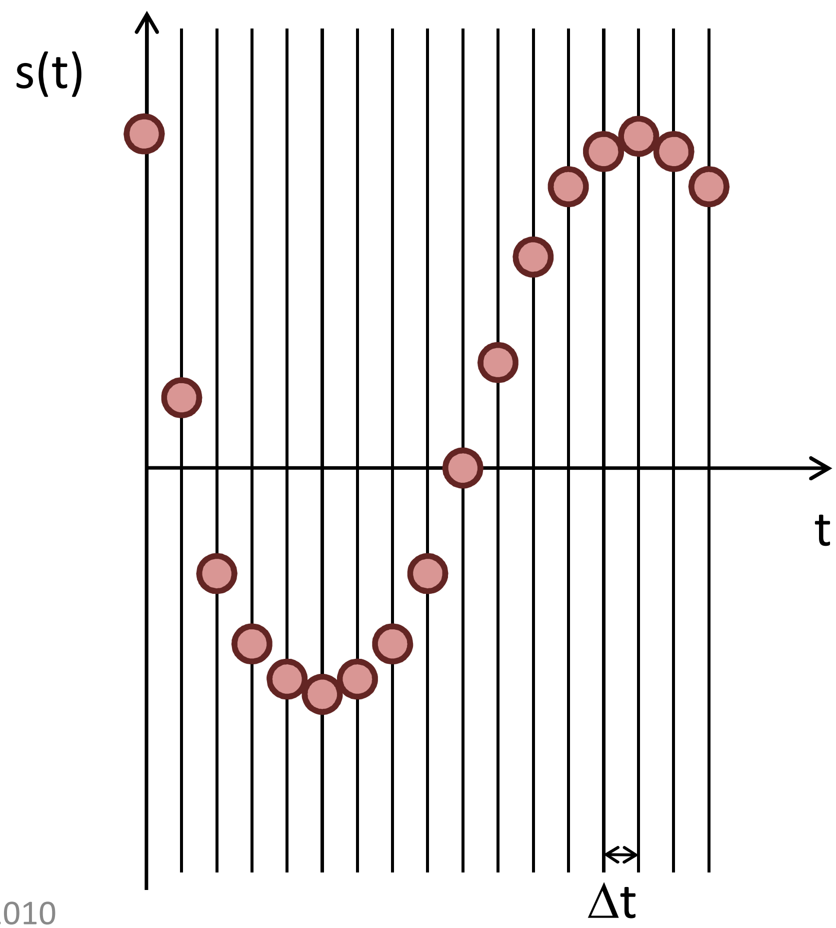
  - **f = m·a** (or **a = f/m**) → Newton's 2nd law
  - Careful about choice of Δt !

Physically based animation

- Time step in Euler integration
  - Depending on stiffness of ODE, **smaller time step**

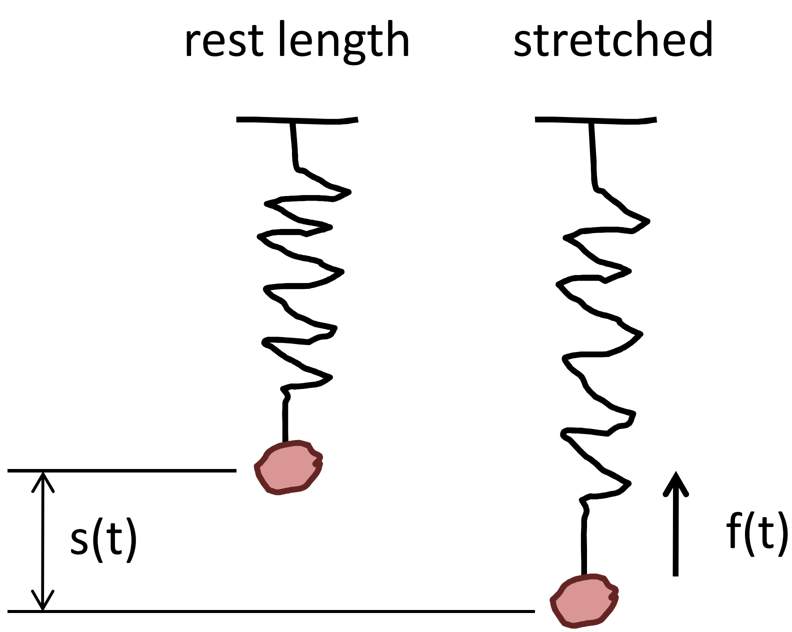


$f(t) = k \cdot s(t)$

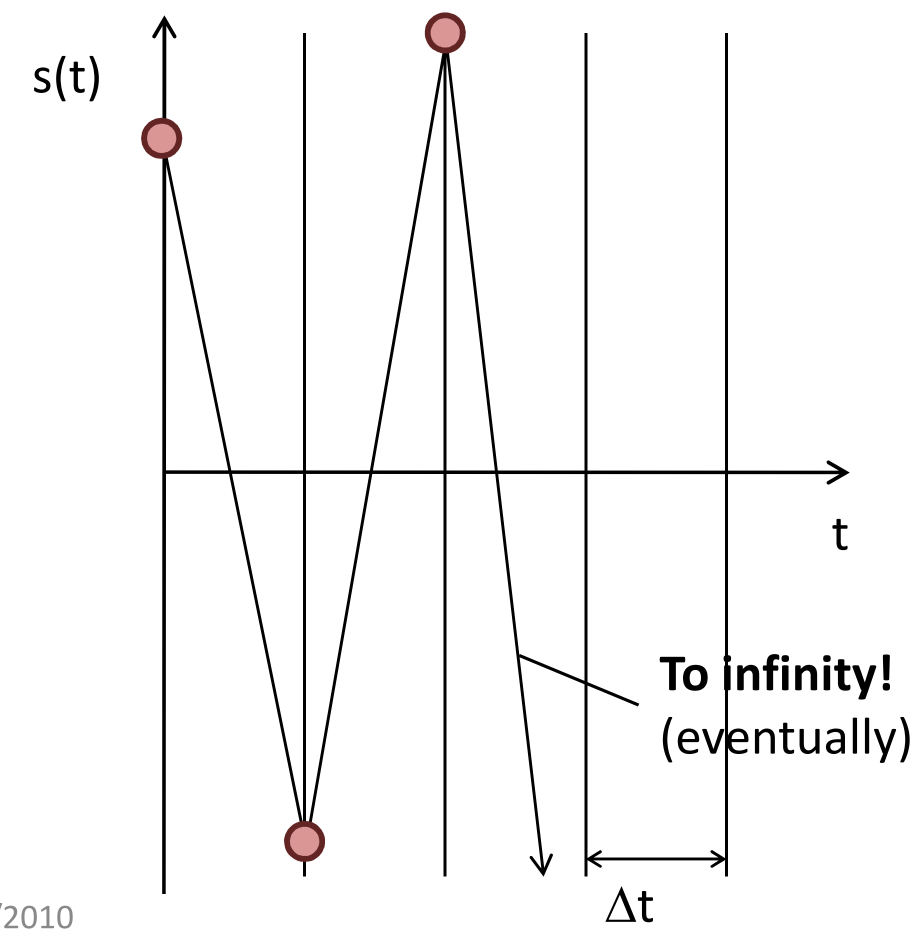


Physically based animation

- Time step in Euler integration
  - Depending on stiffness of ODE, **smaller time step**



f(t) = k·s(t)



Accelerations

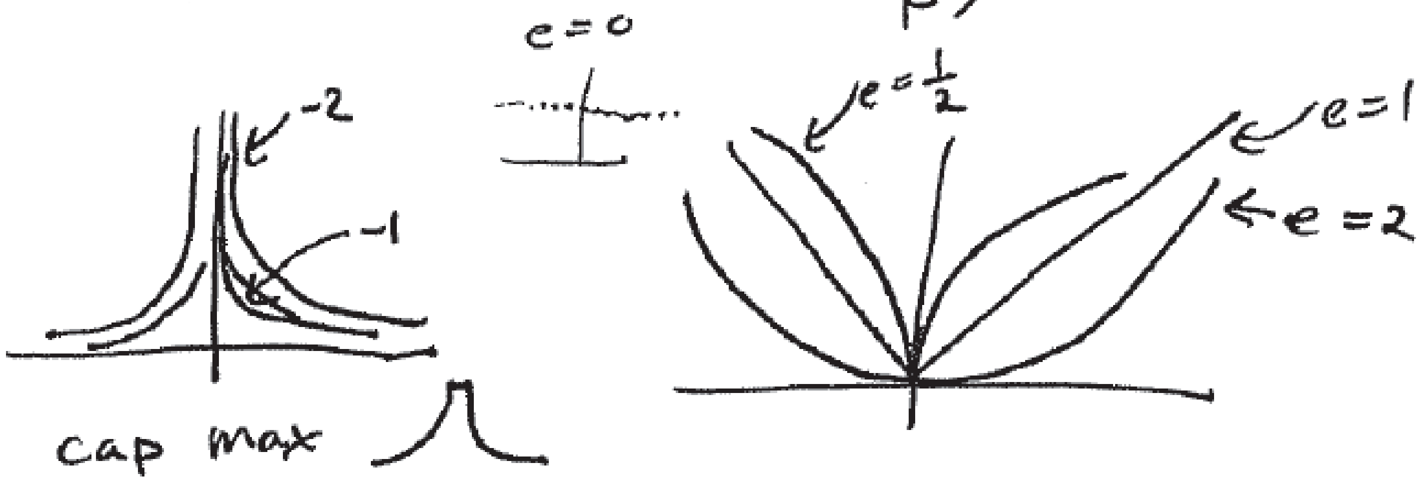
- Distance based Atrraction + repulsion forces

$$a = k\,\hat{d}\,\|d\|^{e}$$

power (e); scaling constant (k)

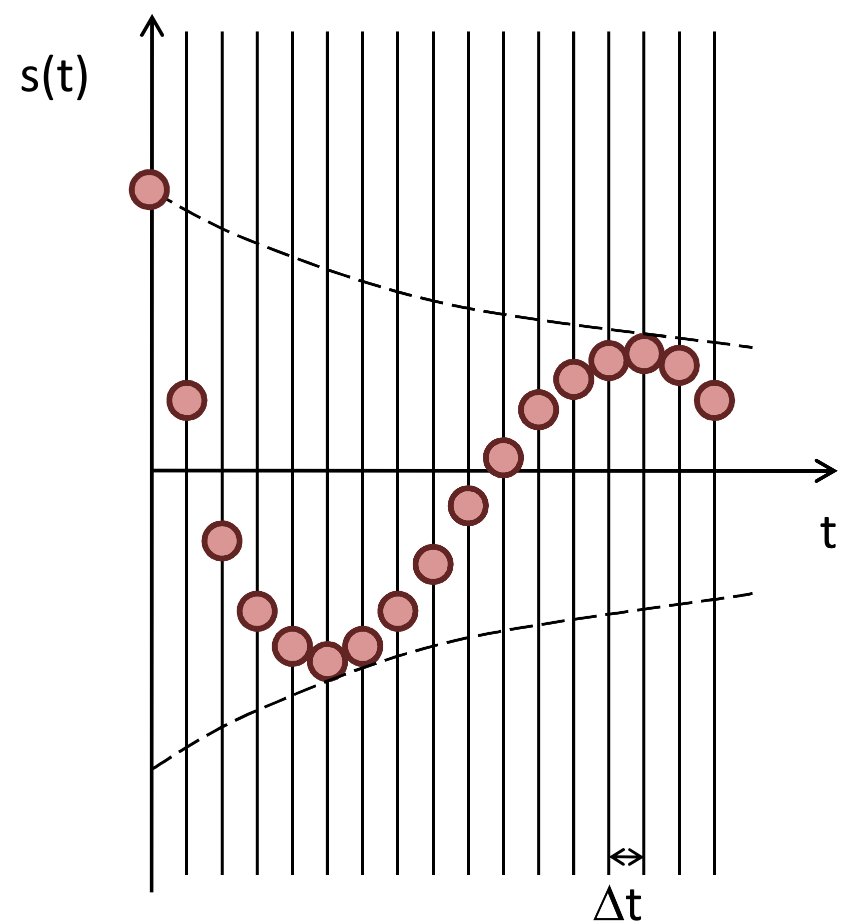
$k > 0 \Rightarrow$ attraction ; $k < 0 \Rightarrow$ repulsion

(towards p)



Accelerations

- Viscous drag $a = -k\,v$
- Numerical stability
- Linearly depends on velocity
  - Air drag
  - Drag inside a liquid
  - k depends on medium in which object is immersed



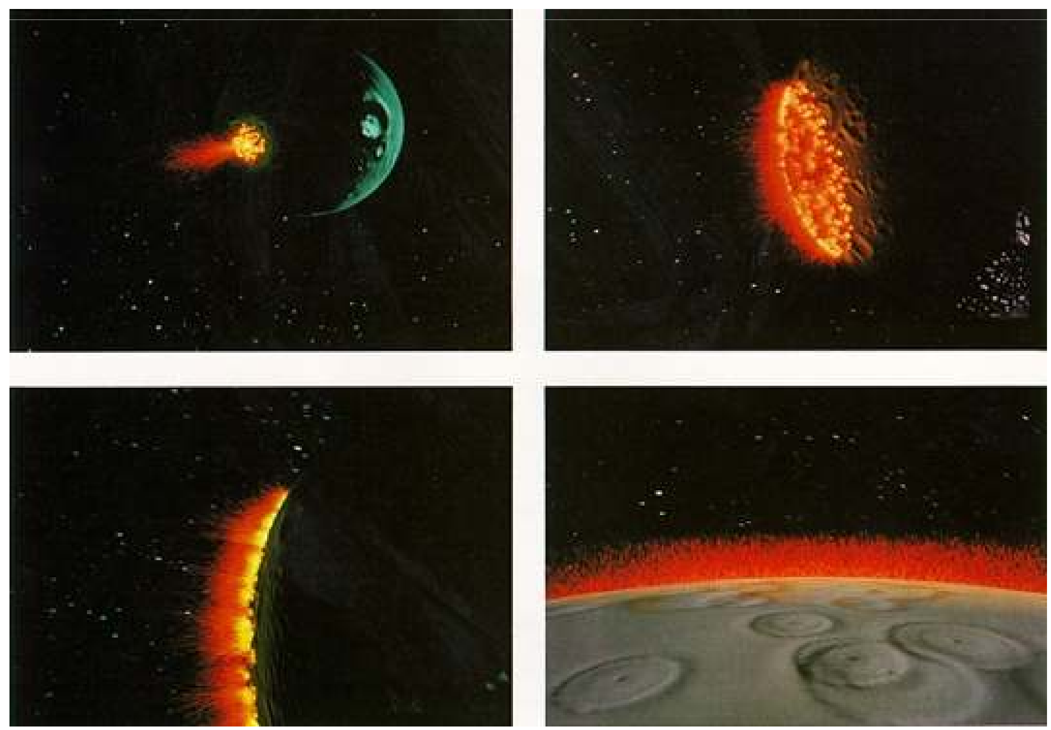
Simulation loop

- Sum up all accelerations per point at time step t
  - Springs
  - Gravity
  - Attraction + repulsion
- Perform one step of Euler integration
  - Obtain updated velocity and position at time step t+1
- Repeat

$$a_{total} = \sum a$$

Particle systems

- For modeling moving, amorphous phenomena
  - Fire, gas, water, explosions
- Collection of particles, where each has
  - Initial position and velocity
  - Initial size, shape, transparency
  - Shape
  - Lifetime
  - Etc.

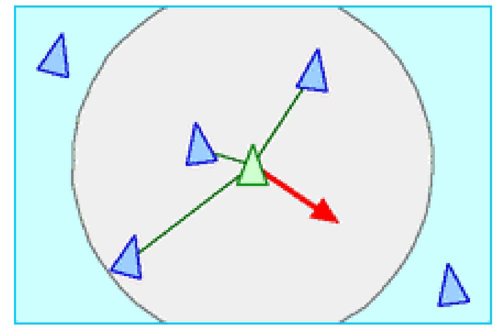


$$f_{avoid} = \frac{(p - obj)}{\|p - obj\|} \cdot k \|p - obj\|^{p}$$

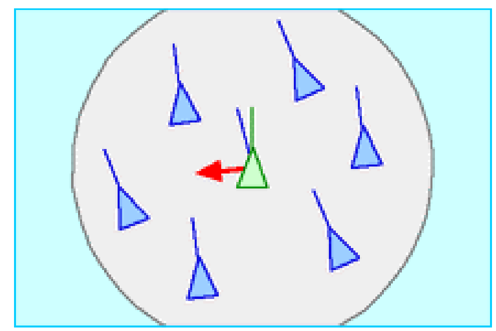
Behaviors

For higher level control

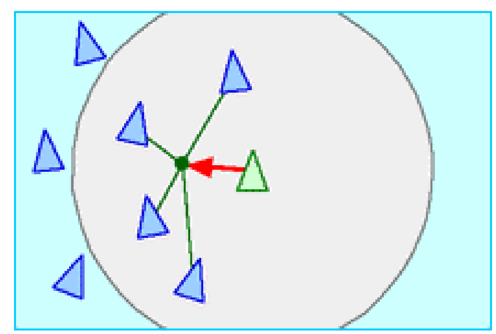
- **Flocking**: three layered behaviors
  - Separation / collision avoidance
    - Steer to avoid crowding flockmates



  - Alignment / velocity matching
    - steer towards the average heading of local flockmates



  - Cohesion / flock centering
    - steer to move toward the average position of local flockmates



Simulation

- Dynamics

$$f = ma = m \frac{d^2 x}{dt^2}$$

$a(t)$
– accel is a function of time

$$v(t+\Delta t) = v(t) + \frac{f}{m} \Delta t$$

$$x(t+\Delta t) = x(t) + v(t) \Delta t$$

a, v, are vectors
x is a position } $\mathbb{R}^3$

- Forces: gravity, viscous drag, attraction, etc.
- Collision detection + response?
- Animator control?

Alternatives

- Closed form solutions

bouncing ball:

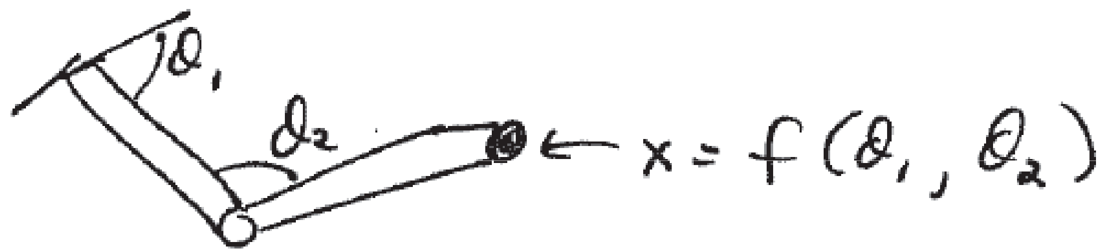
amplitude, angular freq, phase, damping

$$y(x) = A\,|\sin(\omega x + \theta_0)|\,e^{-kx}$$

- Not always available
  - Leg motion or walking is too complicated

Alternatives

- Don't use keyframes, but instead **constraints**
  - "Keep foot flat on floor from frame 3-5"
  - "Elbow/hand is at position x"



- Given x, solve for θ_1 / θ_2: **inverse kinematics**
  - Use of nonlinear equations solvers
  - Problems: non-uniqueness
    - Gets worse with more degrees of freedom (DOFs)
    - Use objective functions $E(\theta_1,\theta_2)$ and nearby solutions