

Animation

Computer Graphics
COMP 770 (236)
Spring 2007

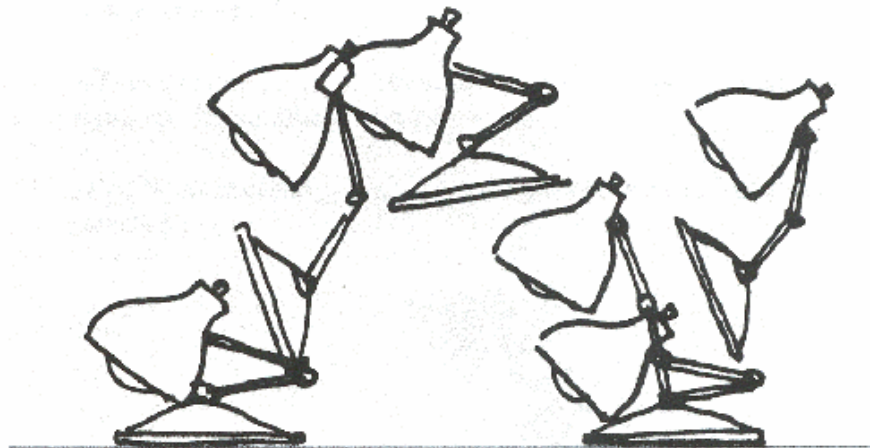
Instructor: Brandon Lloyd

Today's Topics

- Interpolation
- Forward and inverse kinematics
- Rigid body simulation
- Fluids
- Particle systems
- Behavioral modeling
- Motion capture

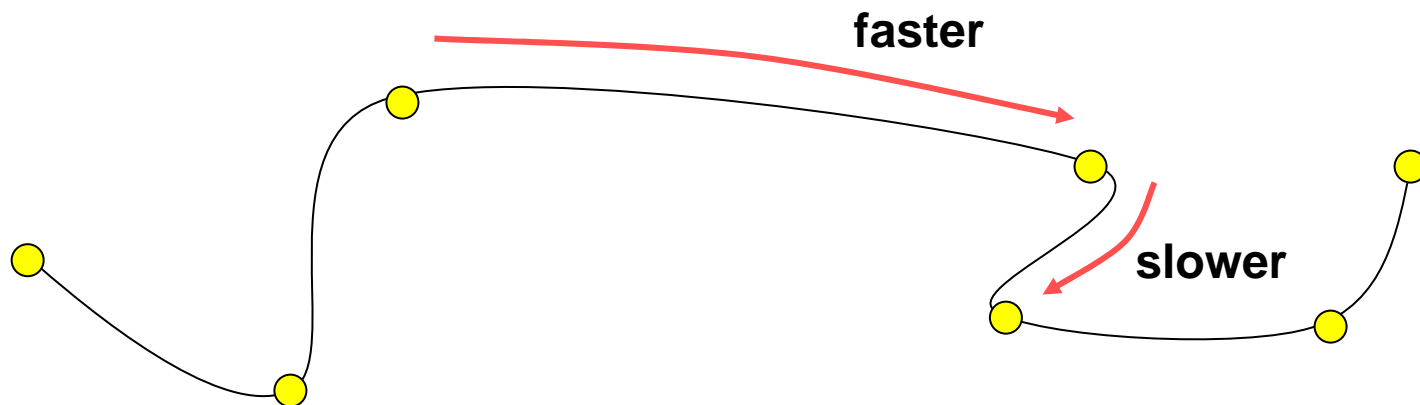
Interpolation

- Interpolation is at the heart of animation
 - gives control and continuity
- Keyframing
 - animator specifies parameters at a few time instances
 - interpolate for other times



Interpolating positions

- Can interpolate positions using splines
- Velocity on the curve also matters
 - uniform changes in parameter don't necessarily translate to uniform changes in position



Arc length reparameterization

- Define $s(u)$ as distance along a curve $\mathbf{P}(u)$:

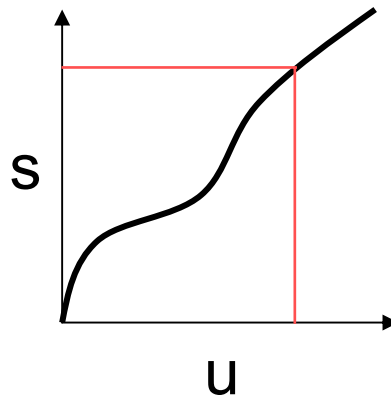
$$s(u) = \int_0^u \left\| \frac{d\mathbf{P}}{d\mu} \right\| d\mu$$

$$\frac{d\mathbf{P}}{du} = \begin{bmatrix} \frac{dx(u)}{du} & \frac{dy(u)}{du} & \frac{dz(u)}{du} \end{bmatrix}$$

- How to compute $s(u)$
 - Analytically
 - not possible with some curves
 - Numerically

Arc length reparameterization

- Compute inverse to get parameter as a function of distance $u(s)$

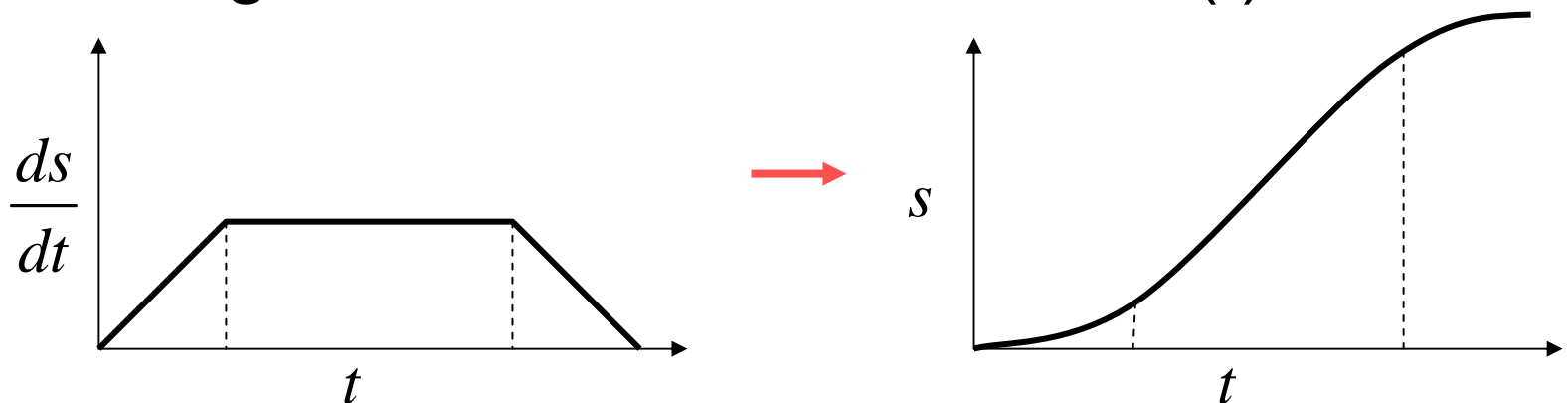


- Arc length reparameterization:

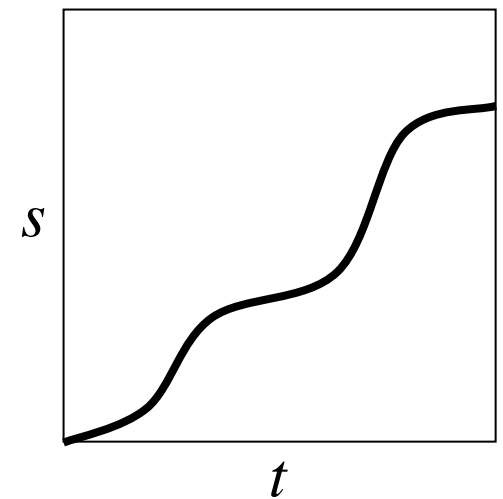
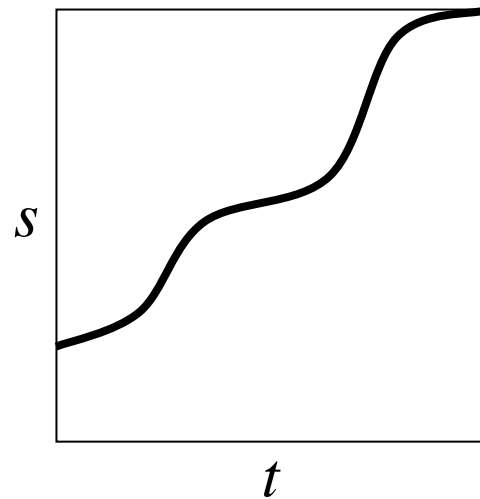
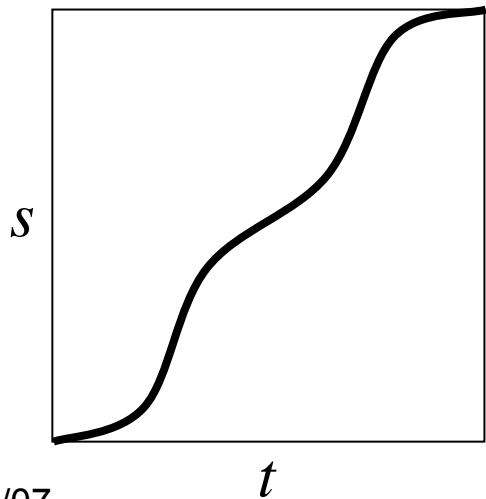
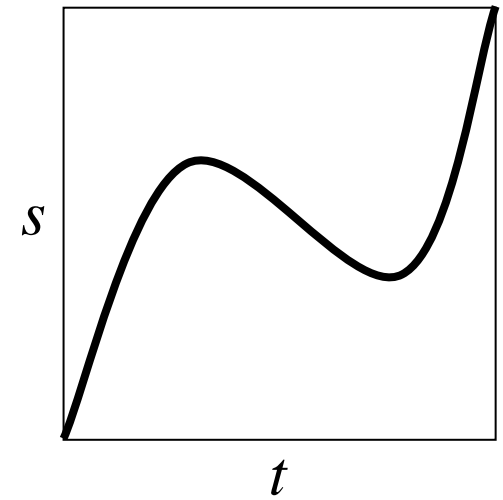
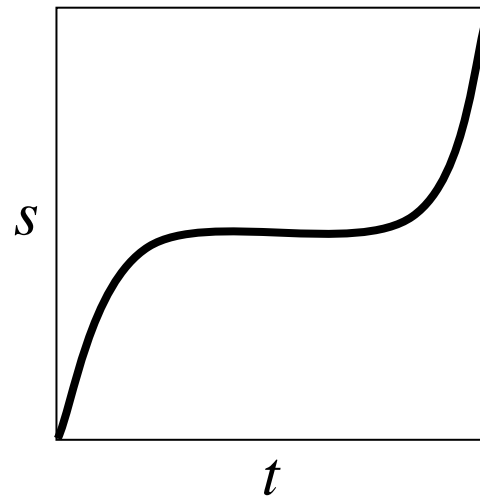
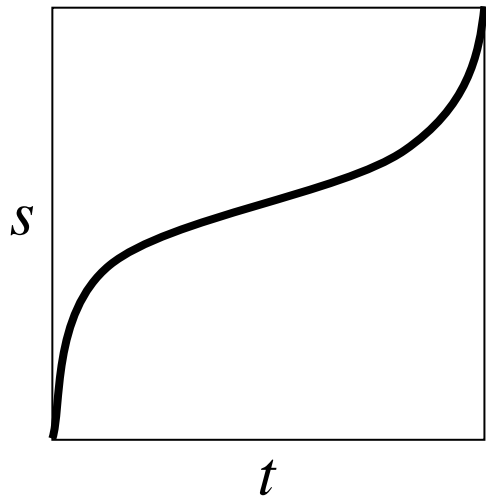
$$P(u(s)) \quad s \in [0, s(u_{end})]$$

Speed curves

- Arc-length parameterization gives uniform velocity on the curve
 - provides a base for arbitrary velocity
- Ease-in / ease-out
 - smooth start and stop motion at endpoints
- Can directly specify speed and integrate over time to get a distance-time function $s(t)$:



Distance-time functions



Interpolating orientation

■ Fixed-axis angles

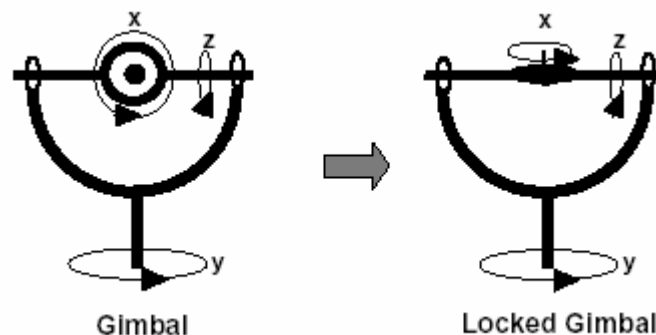
- Rotate about fixed global axes in some order
e.g. x - y - z

■ Euler angles

- Rotate about local axis in some order

■ Problems

- does not always interpolate along a “direct” path
- Gimbal lock – loss of a degree of freedom



Interpolating orientation

■ Axis-angle

- interpolate axis and angle separately
- difficult to concatenate two axis-angle rotations

■ Quaternions

- essentially the same information as axis-angle
- easy to concatenate rotations

Quaternions

■ Represented by 4-tuple $q = [s \ x \ y \ z] = [s \ \vec{v}]$

■ A vector in space $[0 \ \vec{v}]$

■ Addition $q_1 + q_2 = [s_1 + s_2 \ \vec{v}_1 + \vec{v}_2]$

■ Multiplication

- associative but not commutative

$$q_1 q_2 = [s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2 \quad s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2]$$

■ Inverse

$$q^{-1} = (1/\|q\|)^2 [s \ -\vec{v}] \quad \|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}$$

$$qq^{-1} = [1 \ 0 \ 0 \ 0]$$

Rotations with Quaternions

- Representing axis-angle rotations

$$q = [\cos(\theta/2) \quad \sin(\theta/2)\vec{a}]$$

- Concatenate rotations by multiplication

$$q = q_1 q_2$$

- Rotating a vector

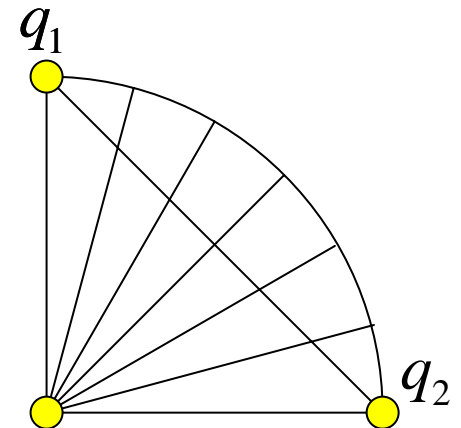
$$\vec{v}' = q\vec{v}q^{-1} \quad \longleftarrow \text{inverse rotates in opposite direction}$$

- magnitude cancels out. Thus $-q$ and q represent the same orientation

Interpolating Quaternions

- Since $q = -q$ we can interpolate from q_1 to q_2 or q_1 to $-q_2$
 - desirable rotation is usually the shorter one
 - check with 4D dot product $q_1 \cdot q_2 = \cos \theta$ which gives the cosine of the angle between the two orientations
- Linear interpolation doesn't give correct results
- Spherical linear interpolation:

$$slerp(q_1, q_2, t) = \frac{\sin((1-u)\theta)}{\sin(\theta)} q_1 + \frac{\sin(u\theta)}{\sin(\theta)} q_2$$



Kinematics and Dynamics

■ Kinematics

- Considers only motion
- Deals mostly with geometric constraints

■ Dynamics

- Considers underlying forces
- Compute configurations from initial conditions and law of physics

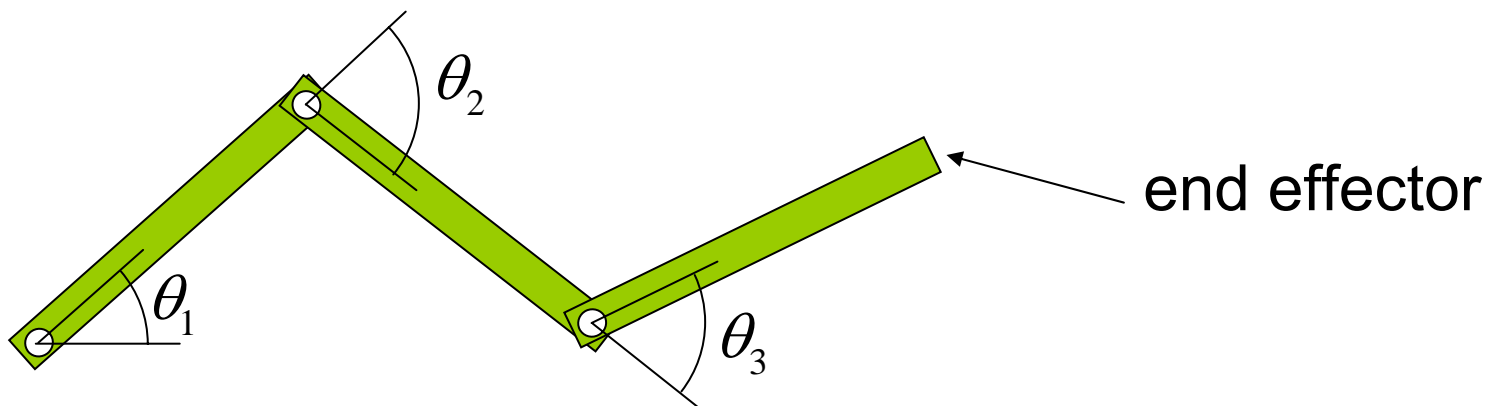
Kinematics

■ Forward Kinematics

- Specify conditions (joint angles)
- Compute positions of end effectors

■ Inverse Kinematics

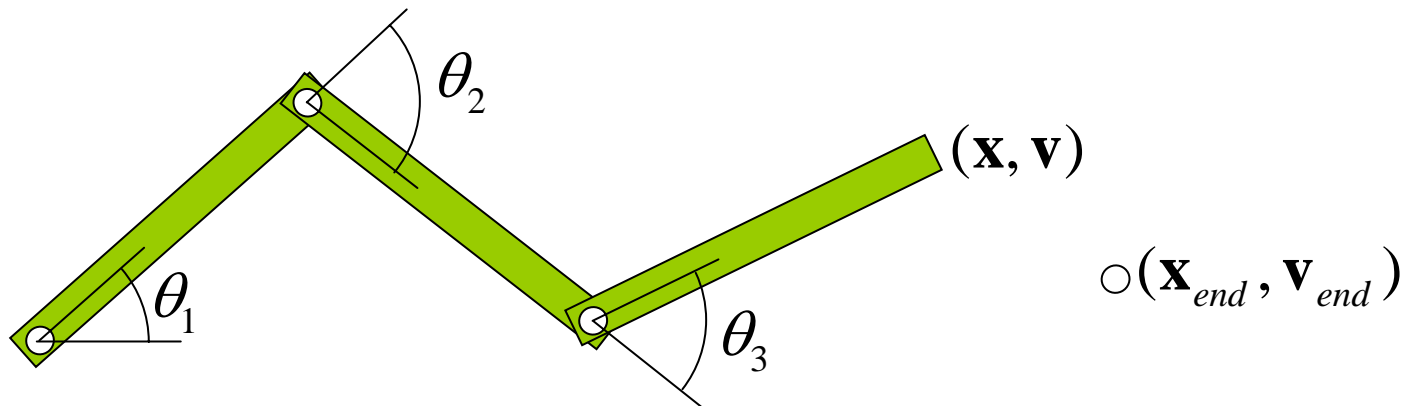
- Specify goal positions of end effectors
- Compute conditions required to achieve goals



Inverse kinematics

- Problem is typically underconstrained
 - multiple solutions
 - can choose “minimum energy” solution
- At each step:
 - Compute difference between goal and current state
 - Compute Jacobian of state vector
 - Solve for change in state that moves closer to goal
 - Integrate
- Easier specification for many animation tasks, but more expensive to compute

Inverse Kinematics

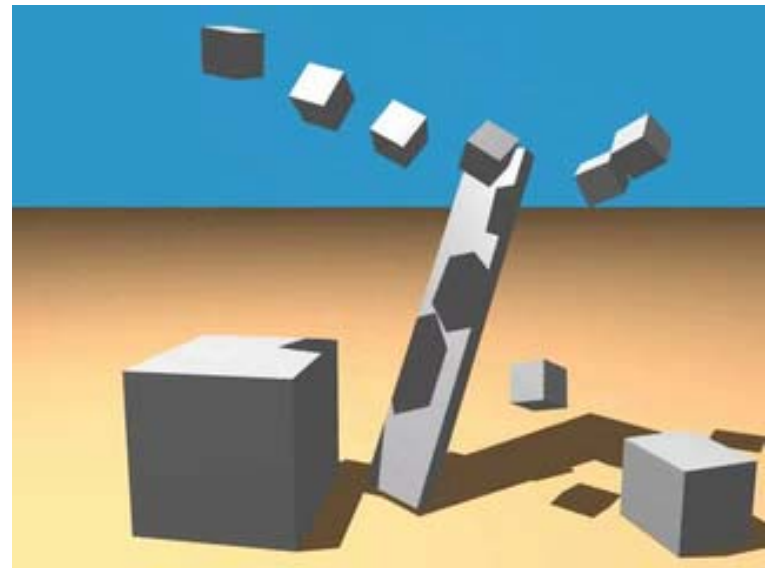


$$\begin{bmatrix} \frac{d\mathbf{x}}{d\theta_1} & \frac{d\mathbf{x}}{d\theta_2} & \frac{d\mathbf{x}}{d\theta_3} \\ \frac{d\mathbf{v}}{d\theta_1} & \frac{d\mathbf{v}}{d\theta_2} & \frac{d\mathbf{v}}{d\theta_3} \end{bmatrix} \begin{bmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \\ \frac{d\theta_3}{dt} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{end} \\ \mathbf{v}_{end} \end{bmatrix} - \begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix}$$

$$\begin{bmatrix} \theta_1^{i+1} \\ \theta_2^{i+1} \\ \theta_3^{i+1} \end{bmatrix} = \begin{bmatrix} \theta_1^i \\ \theta_2^i \\ \theta_3^i \end{bmatrix} + \Delta t \begin{bmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \\ \frac{d\theta_3}{dt} \end{bmatrix}$$

Rigid body dynamics

- Uses laws of physics to provide motion
- Basic steps
 - Compute forces
 - Compute accelerations from forces
 - Integrate to find positions and velocities



An example

■ Cannon ball

$$\mathbf{f}_{gravity} = -mG\mathbf{y}$$

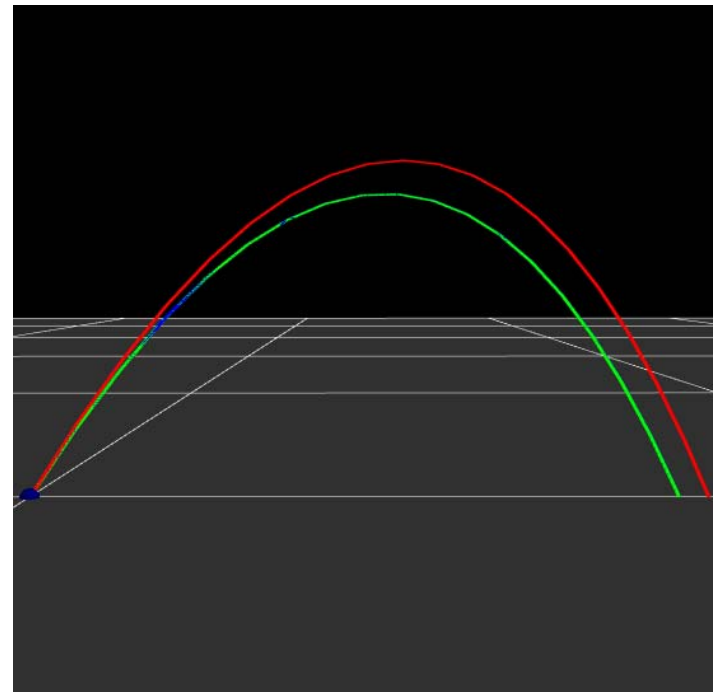
$$\mathbf{f}_{drag} = -m\mu\mathbf{v}$$

$$\mathbf{f} = \sum \mathbf{f}_j = m\mathbf{a}$$

$$\mathbf{p} = m\mathbf{v}$$

$$\mathbf{p}^{(i+1)} = \mathbf{p}^{(i)} + \mathbf{f}\Delta t$$

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \frac{\mathbf{p}^{(i)}}{m}$$



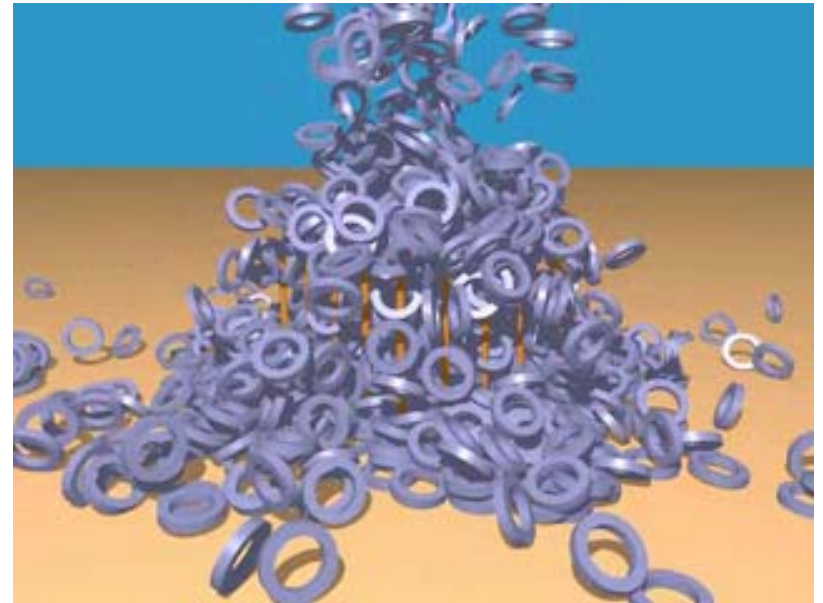
- Euler
- Midpoint
- RK4

Rigid body dynamics

- For 3D bodies add
 - orientation
 - angular momentum
- Constraints
 - Add hard constraints by modifying equations
 - Add soft-constraints with springs and dampers

Collisions

- Collision detection - expensive
 - Accelerate with hierarchies (e.g. bounding volumes)
 - Exploit coherence
- Collision response
 - kinematic
 - penalty methods
 - impulse methods



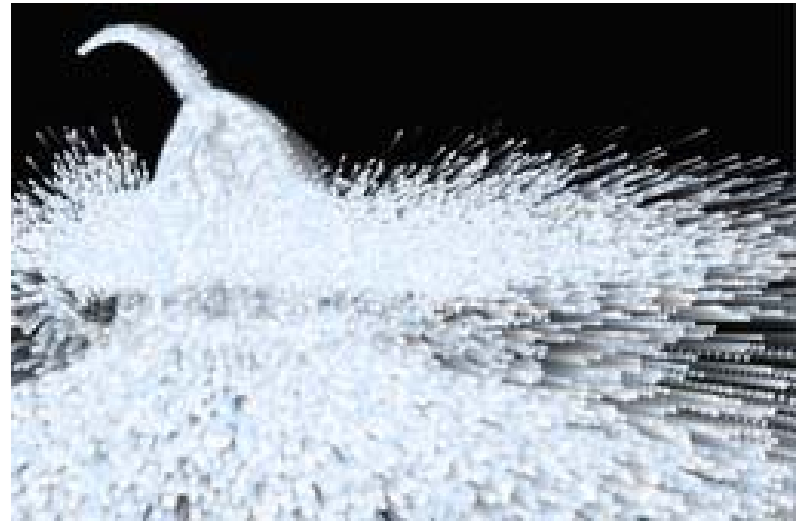
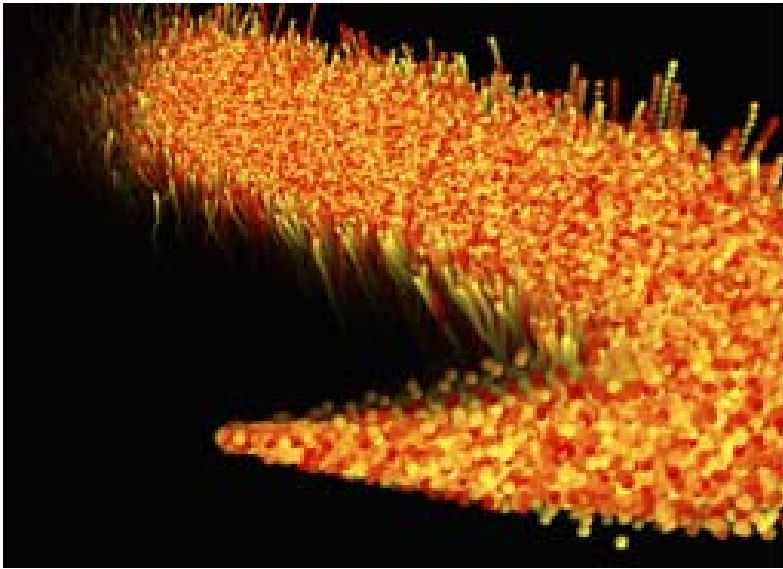
Fluids

- Very difficult to model by hand
- Use computational fluid dynamics
 - more emphasis on speed and visual plausibility than accuracy



Particle systems

- Good for simulating smoke, fire, explosions, water, rain, snow, etc.



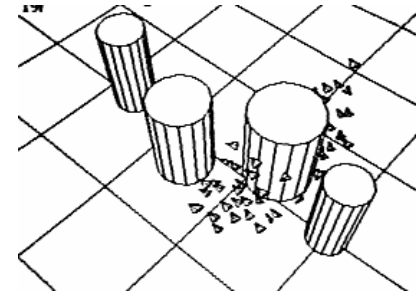
Particle systems

- Steps in computing a frame for a particle system
 - New particles are born
 - New particles assigned attributes
 - Particles removed that exceed life span
 - Particles are animated and shading parameters changed
 - Render particles

<http://video.google.com/videoplay?docid=6837287098782625175>

Behavioral modeling

- Flocking
- Autonomous agents
 - add AI to the animation process



snake learns how to move



crowds in train station

Motion capture

- Track actor's motion
 - optical
 - magnetic
 - exoskeleton
- Recover joint angles from captured data
- Retarget motion to CG character



Motion capture

■ Issues

- noise
- slippage
- occluded markers
- limited capture space
- people aren't really rigid bodies
- dynamics of character aren't the same due to different proportions and mass distribution