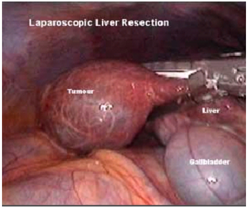
Introduction to Physically Based Animation

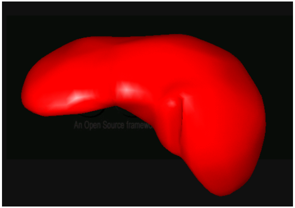
François Faure

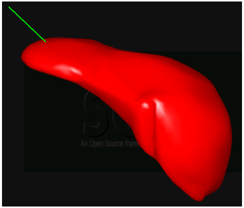
EVASION-LJK

Motivation

- Realistic motion
- Interaction

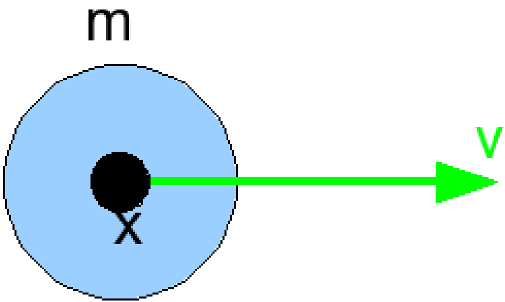






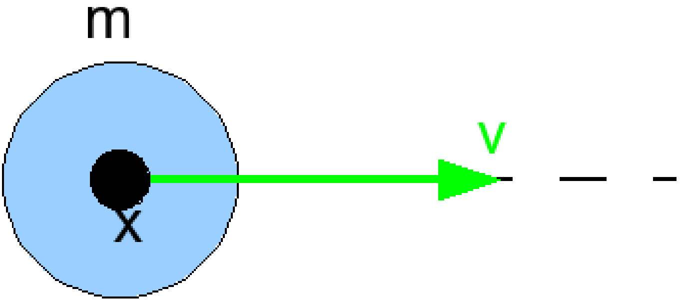
A physical particle

- Position x in m
- Velocity $v = \frac{dx}{dt} = \dot{x}$ in m/s
- Mass m in kg



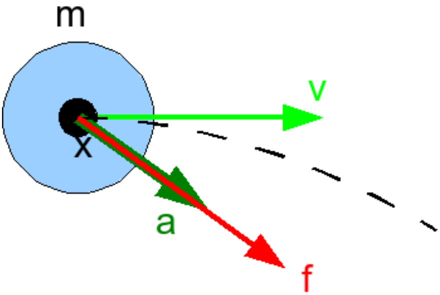
Newton's first law

- An isolated system has a constant velocity



Newton's second law

$$\boxed{f = ma}$$



- Acceleration $a = \frac{dv}{dt} = \frac{d^2x}{dt^2} = \ddot{x}$
- Force in $kg.m/s^2$
- A force is "something" able to modify the trajectory or the shape of an object

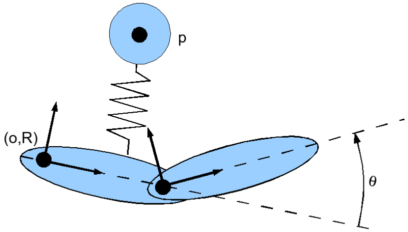
Newton's third law

$$f_{1\to 2} = -f_{2\to 1}$$

- The net force applied to an isolated system is null, even if internal forces are applied
- Its center of mass has a linear trajectory

Generalization: Lagrangian dynamics

$$\frac{d}{dt}\left(\frac{\partial(T-P)}{\partial\dot{q}}\right) - \frac{\partial(T-P)}{\partial q} = Q(q, \dot{q}, t)$$



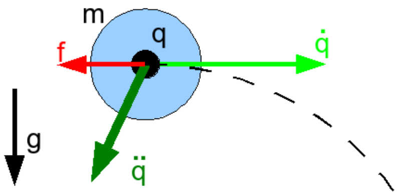
- q denote the mechanically independent parameters (here o, R, p, θ)
- P is the potential energy
- T is the kinetic energy
- Q is the non-conservative forces

Example of Lagrangian dynamics

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right) + \frac{\partial P}{\partial q} = Q(q, \dot{q}, t)$$

- $q = (x, y, z)$
- $P = -mgq$ with gravity vector g
- $T = \frac{1}{2}m\dot{q}^2$
- $Q =$ viscous force $-\nu\dot{\mathbf{q}}$

$$m\ddot{q} = mg - \nu\dot{\mathbf{q}}$$



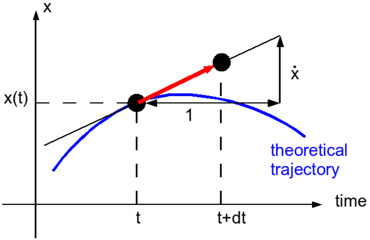
Basic time integration

Explicit Euler integration over a time set dt:

- compute acceleration $\ddot{\mathbf{q}}$
- update time, positions and velocities:

$$\begin{aligned} t &\mathrel{+}= dt \\ \mathbf{q} &\mathrel{+}= \dot{\mathbf{q}} * dt \\ \dot{\mathbf{q}} &\mathrel{+}= \ddot{\mathbf{q}} * dt \end{aligned}$$

- precision depends on dt because update follows the tangent



Structure of a physically based animation program

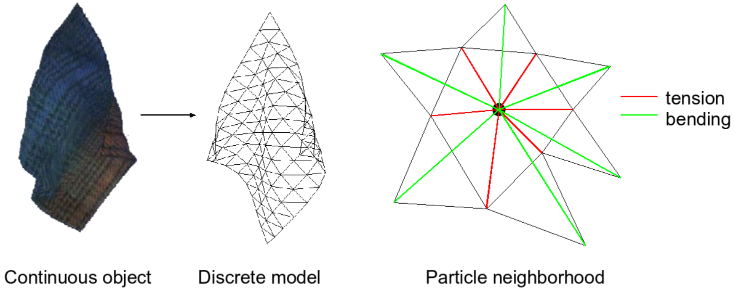
A classical structure:

- init
- display
- repeat:
  - input (data, user action)
  - compute forces
  - update state
  - repeat:
    - apply constraints
  - display

There are many variants !

Mass-spring systems

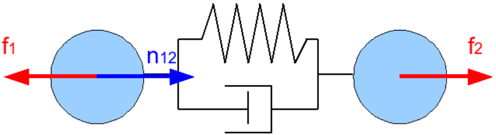
- 1D, 2D or 3D mesh



- vertices = particles, edges = springs
- simple, but parameters are difficult to tune

The spring model

- Viscoelastic force



- In one dimension: $f_1 = k\frac{x_2 - x_1 - l_0}{l_0} + \nu(\dot{x}_2 - \dot{x}_1)$
- In 2D or 3D:

$$f_1 = \left(k\frac{\|\mathbf{q}_2 - \mathbf{q}_1\| - l_0}{l_0} + \nu(\dot{\mathbf{q}}_2 - \dot{\mathbf{q}}_1).\mathbf{n}_{12}\right)\mathbf{n}_{12}$$

$$\text{with } \mathbf{n}_{12} = \frac{\mathbf{q}_2 - \mathbf{q}_1}{\|\mathbf{q}_2 - \mathbf{q}_1\|}$$

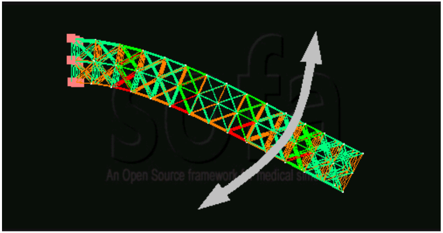
Acceleration of mass-spring particles

for each particle i:
$\mathbf{F}_i = \mathbf{f}_i(\mathbf{q}_i, \dot{\mathbf{q}}_i, t)$ // unary forces
for each spring i,j:
$\mathbf{F} = \mathbf{f}_{ij}(\mathbf{q}_i, \dot{\mathbf{q}}_i, \mathbf{q}_j, \dot{\mathbf{q}}_j, t)$ // interaction forces
$\mathbf{F}_i$ += $\mathbf{F}$
$\mathbf{F}_j$ -= $\mathbf{F}$
for each particle i:
$\mathbf{A}_i = \mathbf{F}_i / m_i$ // accelerations
for each *fixed* particle i:
$\mathbf{A}_i = \mathbf{0}$ // fixed points do not accelerate

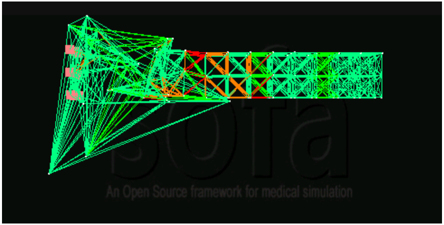
The problem of stiffness

For a given time step *dt*

- With low stiffness, smooth oscillations are obtained



- With high stiffness, instabilities make the simulation "explode"

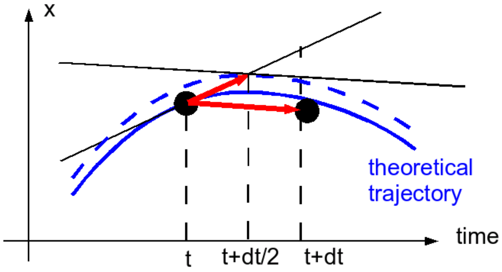


- reducing the time step is more expensive

Higher-order explicit integration

Midpoint method (second-order Runge-Kutta)

- perform a fictitious dt/2 Euler step
- compute the derivative there
- use this derivative for a full Euler step



- error is proportional to dt^2 instead of dt
- even more sophisticated methods exist
- better, but instability remains

Symplectic methods

Symplectic Euler:

- compute acceleration $\ddot{\mathbf{q}}$
- Use updated velocity to update position:

$$
\begin{aligned}
t &\mathrel{+}= dt \\
\dot{\mathbf{q}} &\mathrel{+}= \ddot{\mathbf{q}} * dt \\
\mathbf{q} &\mathrel{+}= \dot{\mathbf{q}} * dt
\end{aligned}
$$

- much better energy conservation
- but instability still occurs
- variants: leap-frog, Stoermer-Verlet

Implicit time integration

- Use $\ddot{\mathbf{q}}$(t+dt) to update velocity
- Implicit Euler:

$$\begin{aligned} \dot{\mathbf{q}} &\mathrel{+}= \ddot{\mathbf{q}}(t+dt) * dt \\ \mathbf{q} &\mathrel{+}= \dot{\mathbf{q}} * dt \end{aligned}$$

- inconditionally stable
- but an equation system must be solved

Linearized implicit Euler

- Solve

$$\begin{aligned}\left(\mathbf{M} - \mathbf{D}dt - \mathbf{K}dt^2\right)\Delta\dot{\mathbf{q}} &= \left(\mathbf{f} + \mathbf{K}\dot{\mathbf{q}}dt\right)dt \\ \text{with } \mathbf{M} &= \text{diagonal mass matrix} \\ \mathbf{K} &= \frac{\partial \mathbf{f}}{\partial \mathbf{q}} \text{ stiffness matrix} \\ \mathbf{D} &= \frac{\partial \mathbf{f}}{\partial \dot{\mathbf{q}}} \text{ damping matrix}\end{aligned}$$

- then

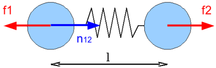
$$\begin{aligned}\dot{\mathbf{q}} &\mathrel{+}= \Delta\dot{\mathbf{q}} \\ \mathbf{q} &\mathrel{+}= \dot{\mathbf{q}} * dt\end{aligned}$$

- popular assumption: Rayleigh damping $\mathbf{D} = \alpha\mathbf{M} + \beta\mathbf{K}$

Implicit Euler in practice

- We have to solve a linear equation system $\mathbf{Ax} = \mathbf{b}$
- $\mathbf{A}$ is PSD, we use the conjugate gradient solution:
  - only implement the product of $\mathbf{A}$ with a vector
  - iterative solution
- To apply simple constraints:
  - Solve $\mathbf{CAx} = \mathbf{Cb}$
  - $\mathbf{C}$ is a diagonal matrix with null diagonal values for constrained directions (a trivial filter)
- Spring stiffness:

$$\begin{aligned}
\mathbf{K}_{12} &= \mathbf{K}_{21} = \frac{\partial \mathbf{f}_1}{\partial \mathbf{q}_2} \\
&= (k - \frac{f}{l}) \left[\mathbf{n}_{12}\mathbf{n}_{12}^T\right] + \frac{f}{l}\mathbf{I_3} \\
\mathbf{K}_{11} &= \mathbf{K}_{22} = -\mathbf{K}_{12}
\end{aligned}$$



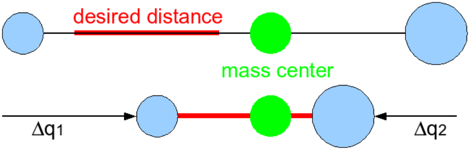
$\mathbf{I_3}$ being the 3×3 identity matrix,
$f = \|\mathbf{f}_1\|$ and $l = \|\mathbf{q}_2 - \mathbf{q}_1\|$

The Provot approach

- Apply simple time integration, then prevent springs to extend or compress too much
- Algorithm:

  apply symplectic Euler
  repeat:
      for each spring i, j:
          if extension or compression $> 10\%$
              move the particles to 10% of extension or compression
  until no spring is too much extended or compressed

Distance correction



- compute desired relative displacement

$$\begin{aligned}\mathbf{\Delta q} &= \mathbf{\Delta q}_2 - \mathbf{\Delta q}_1 \\ &= -(\text{desired length-current length})\mathbf{n}_{12}\end{aligned}$$

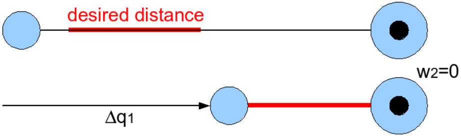
- move the particles without moving their mass center

$$\begin{aligned}\mathbf{\Delta q}_1 &= \frac{m_2}{m_1 + m_2}\mathbf{\Delta q} \\ \mathbf{\Delta q}_2 &= -\frac{m_1}{m_1 + m_2}\mathbf{\Delta q}\end{aligned}$$

- update the velocities

$$\begin{aligned}\dot{\mathbf{q}}_1 &\mathrel{+}= \mathbf{\Delta q}_1/dt \\ \dot{\mathbf{q}}_2 &\mathrel{+}= \mathbf{\Delta q}_2/dt\end{aligned}$$

A more general formulation



- Fixed points are considered as points with infinite masses
- Use inverse mass $w = 1/m$
- $w = 0$ for a fixed point
- move the particles

$$\begin{aligned}\mathbf{\Delta q}_1 &= \frac{w_1}{w_1 + w_2}\mathbf{\Delta q} \\ \mathbf{\Delta q}_2 &= -\frac{w_2}{w_1 + w_2}\mathbf{\Delta q}\end{aligned}$$

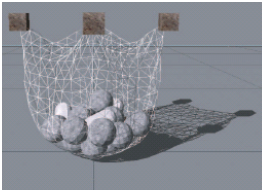
Generalization: position-based dynamics

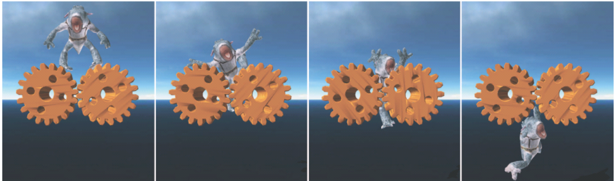
Complex constraints: aligned points, are or volume conservation, etc.

- model a constraint as a value to cancel $c = C(\mathbf{q}, \dot{\mathbf{q}})$
- Solve:

$$\frac{\partial c}{\partial \mathbf{q}} \Delta \mathbf{q} \quad = \quad -c$$

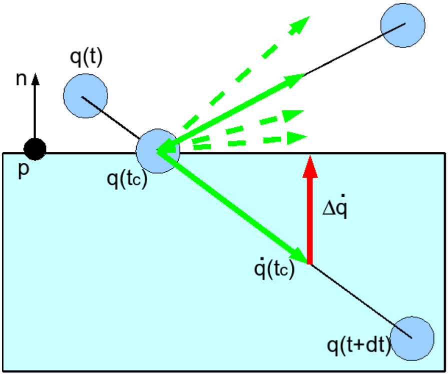
without moving the mass center





Collision of a particle with a surface

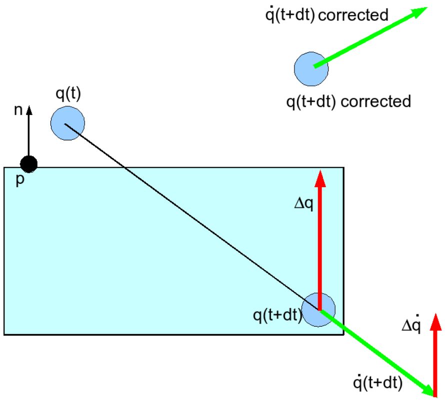
- criterion: $\mathbf{pq}.\mathbf{n} < 0$
- backtrack to collision time t_c
- compute velocity increment for an inelastic collision $\Delta\dot{\mathbf{q}} = -(\dot{\mathbf{q}}.\mathbf{n})\mathbf{n}$
- apply a bouncing coefficient ϵ: $\dot{\mathbf{q}} \mathrel{+}= (1+\epsilon)\Delta\dot{\mathbf{q}}$
- continue simulation
- problem: with several particles, several backtracks and restarts may be necessary



Synchronized collisions

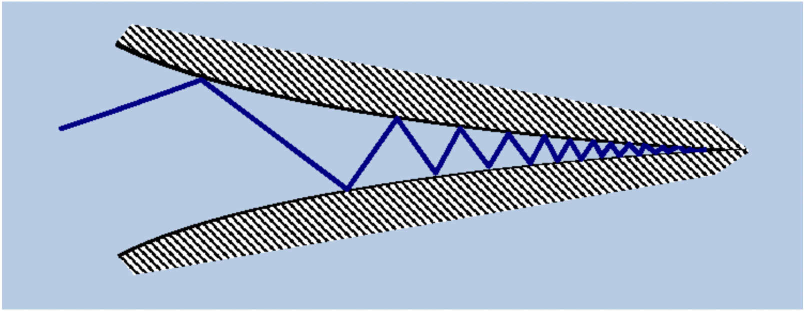
Similar, but:

- do not backtrack to collision time
- compute position increment to project the particle to the surface $\Delta \mathbf{q} = -(\mathbf{pq}.\mathbf{n})\mathbf{n}$
- apply a bouncing to position also: $\mathbf{q} \mathrel{+}= (1+\epsilon)\Delta \mathbf{q}$
- advantage: all collisions are handled at the same time



A bad case

Rattling



Collision of two spheres

- criterion: $\|\mathbf{q_1 q_2}\| < r_1 + r_2$
- compute position increments for an inelastic collision

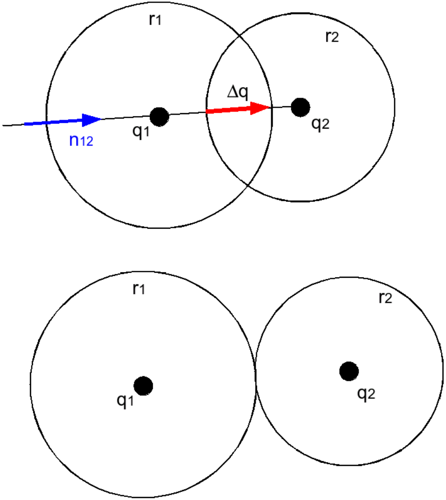
$$\Delta\mathbf{q} \quad = \quad (r_1 + r_2 - \|\mathbf{q_1 q_2}\|)\mathbf{n}_{12}$$

- use the inverse masses to maintain the center of mass

$$\begin{aligned} \mathbf{\Delta q}_1 &= \frac{w_1}{w_1 + w_2}\mathbf{\Delta q} \\ \mathbf{\Delta q}_2 &= -\frac{w_2}{w_1 + w_2}\mathbf{\Delta q} \end{aligned}$$

- apply a bouncing coefficient ϵ:

$$\begin{aligned} \mathbf{q}_1 &\mathrel{+}= (1 + \epsilon)\mathbf{\Delta q}_1 \\ \mathbf{q}_2 &\mathrel{+}= (1 + \epsilon)\mathbf{\Delta q}_2 \end{aligned}$$



Collision of two spheres (continued)

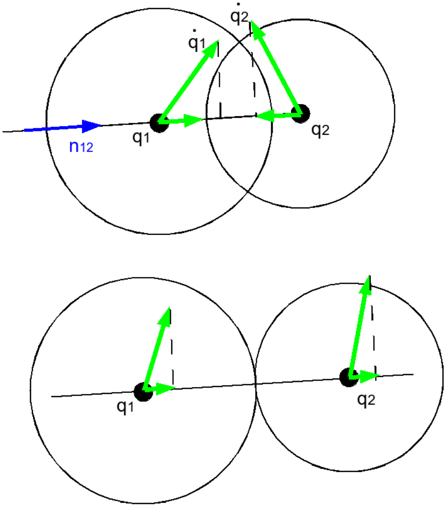
- compute velocity increments for an inelastic collision $\Delta\dot{\mathbf{q}} = ((\dot{\mathbf{q}}_2 - \dot{\mathbf{q}}_1).\mathbf{n}_{12})\mathbf{n}_{12}$
- use the inverse masses to maintain the center of mass

$$\begin{aligned}
\mathbf{\Delta\dot{q}}_1 &= \frac{w_1}{w_1 + w_2}\mathbf{\Delta\dot{q}} \\
\mathbf{\Delta\dot{q}}_2 &= -\frac{w_2}{w_1 + w_2}\mathbf{\Delta\dot{q}}
\end{aligned}$$

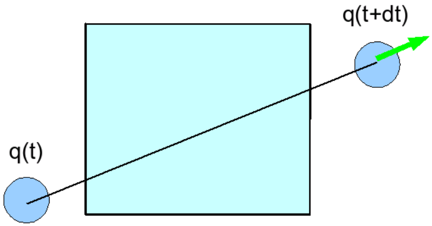
- apply a bouncing coefficient ϵ:

$$\begin{aligned}
\dot{\mathbf{q}}_1 &\mathrel{+}= (1+\epsilon)\Delta\dot{\mathbf{q}}_1 \\
\dot{\mathbf{q}}_2 &\mathrel{+}= (1+\epsilon)\Delta\dot{\mathbf{q}}_2
\end{aligned}$$

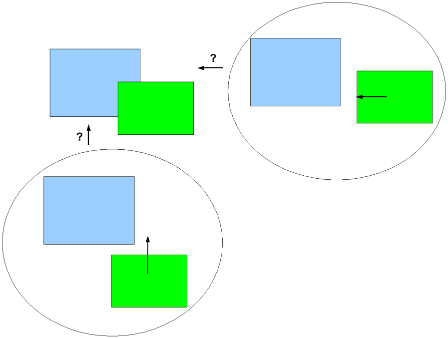
- or compute $\Delta\dot{\mathbf{q}}_i = \Delta\mathbf{q}_i/dt$



Limitations of discrete-time collision detection



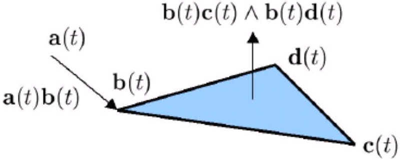
- Thin objects can be traversed
- The history is sometimes necessary



Continuous-time collision detection

- Search four coplanar point (solve cubic equation in time)
- Point-triangle intesection:

$$\mathbf{a}(t)\mathbf{b}(t) \cdot (\mathbf{b}(t)\mathbf{c}(t) \wedge \mathbf{b}(t)\mathbf{d}(t)) = 0$$

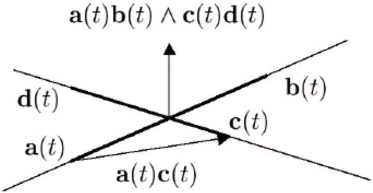


- Then for the smallest $0 < t < dt$ compute point positions

Continuous-time collision detection

- Search four coplanar point (solve cubic equation in time)
- Edge-edge intesection:

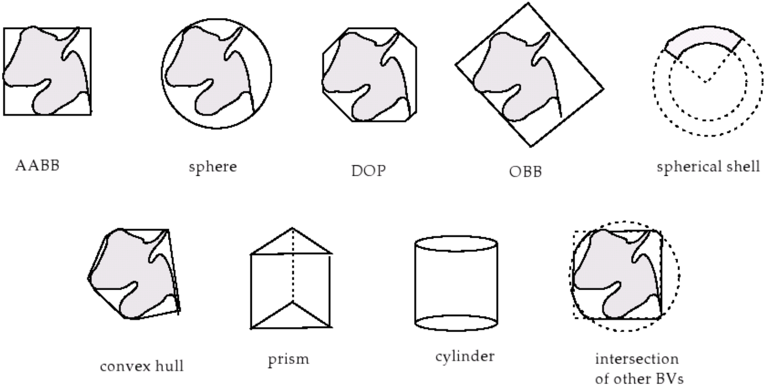
$$\mathbf{a}(t)\mathbf{c}(t) \cdot (\mathbf{a}(t)\mathbf{b}(t) \wedge \mathbf{c}(t)\mathbf{d}(t)) = 0$$



- Then for the smallest $0 < t < dt$ compute point positions

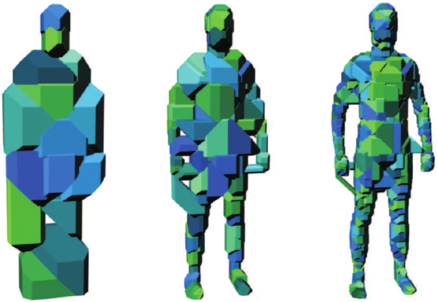
Acceleration of collision detection using bounding volumes

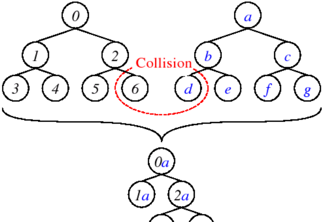
- If the BVs don not intersect then the objects do not intersect



Hierarchies of bounding volumes

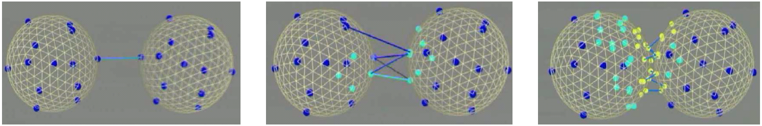
- Accelerate even more
- Hierarchy update is expensive for deformable objects





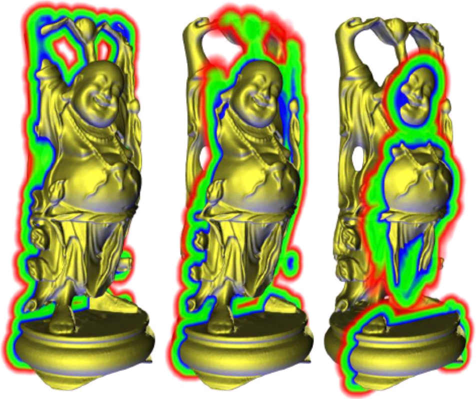
Stochastic methods

- Pick sample pairs
- Refine where proximities are found



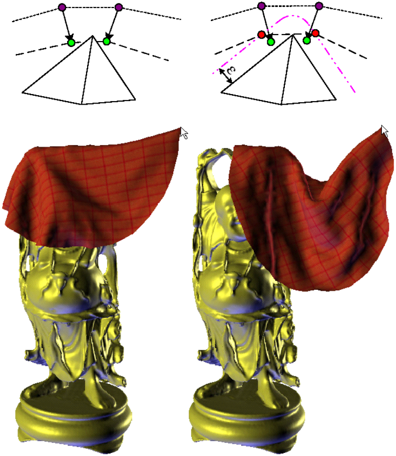
Distance fields

- function returning the closest surface point
- project particles to the surface



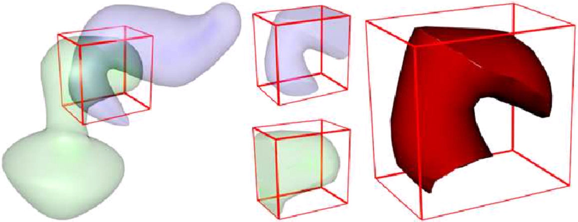
Distance fields (continued)

- Distance offsets are necessary to prevent edge collisions



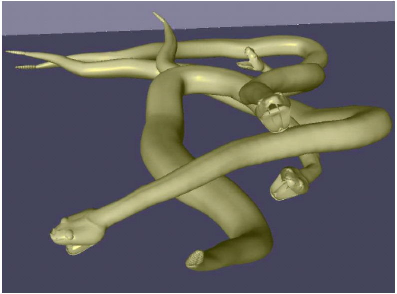
An image-space technique

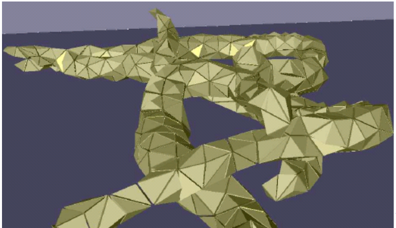
- Compute AABB intersection
- If intersection, compute Layerd Depth Images of both objects
- Test ecah vertex of one body agains the LDI of the other



Simplified geometries

- Embed a complex geometry in a coarser one
- Apply dynanmics and collisions to the coarse geometry
- render the fine geometry





Other topics

- rigid bodies
- fluids
- hair
- ...