

Physically-based simulation of rigid bodies

François Faure

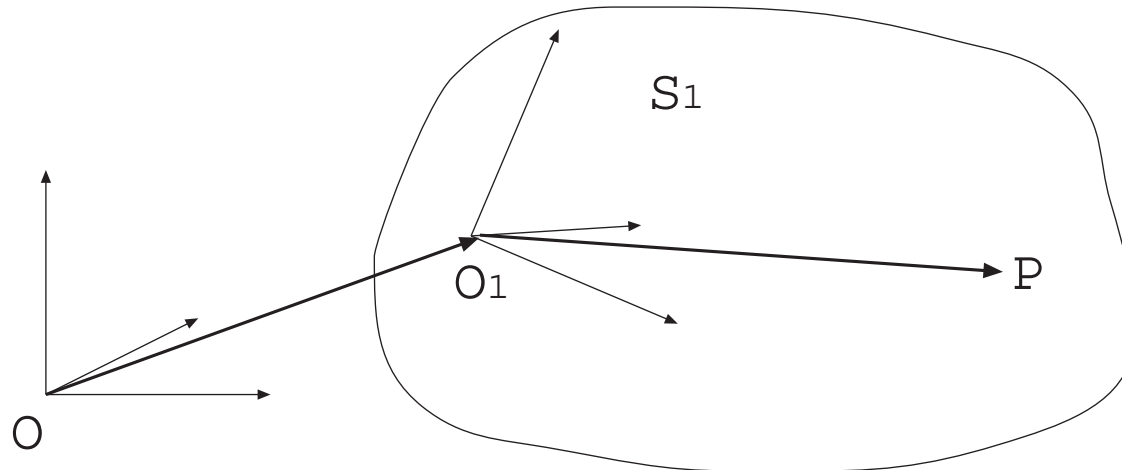
Institut für Computergraphik
Technische Universität, Wien
francois@cg.tuwien.ac.at

Conventions

- Vectors are denoted using bold lower-case letters. Ex: \mathbf{v}
- Matrices are denoted using bold upper-case letters. Ex: \mathbf{J}
- Scalar values are denoted using standard letters. Ex: m
- Velocities and accelerations are defined with respect to a fixed reference frame.

Solids

- Set of material points fixed with respect to a local frame



- Projections

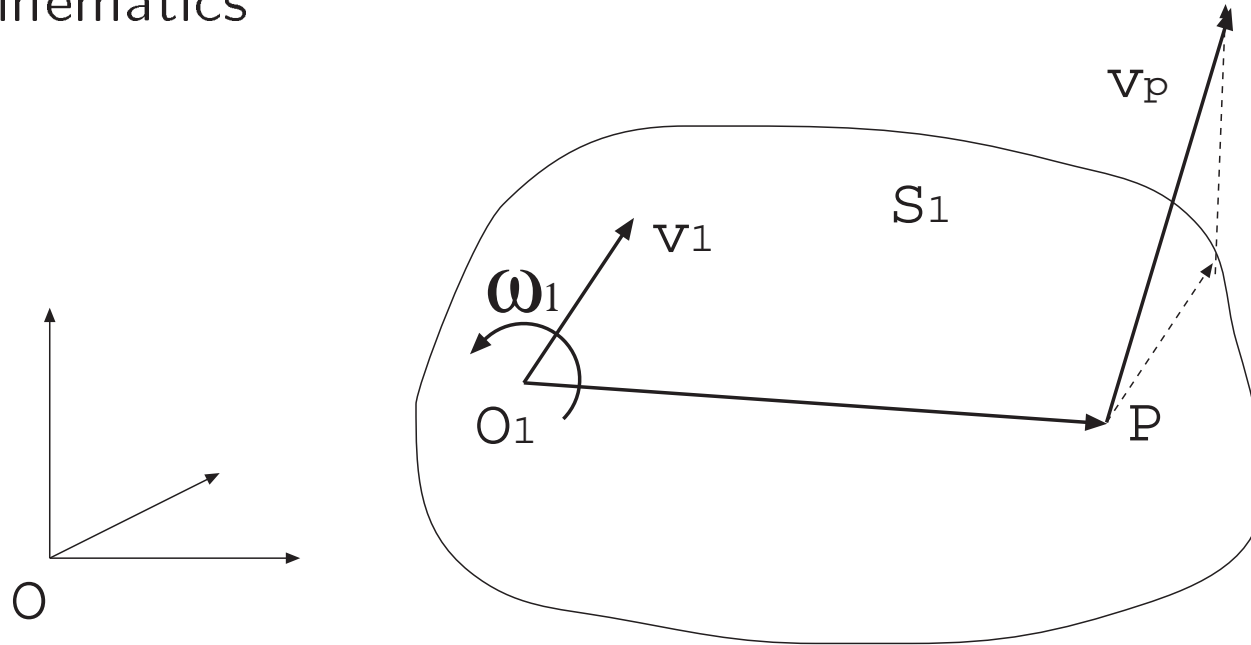
$$o p = o o_1 + o_1 p$$

$$o p^0 = o o_1^0 + \mathbf{R}^{0S_1} o_1 p^S$$

\mathbf{R} is a rotation matrix. Rotations can also be modeled using Euler angles or quaternions.

Solids

- Kinematics



$$\dot{\mathbf{p}} = \dot{\mathbf{o}}_1 + \boldsymbol{\omega}_1 \times \mathbf{o}_1 p$$

$$\ddot{\mathbf{p}} = \ddot{\mathbf{o}}_1 + \dot{\boldsymbol{\omega}}_1 \times \mathbf{o}_1 p + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \mathbf{o}_1 p)$$

Mass

- General expression:

$$m = \int_{\mathcal{P} \in S} \rho(\mathbf{p}) dx dy dz = \int dm$$

- Mass center \mathbf{c} :

$$\int_{\mathcal{P} \in S} \mathbf{c} \rho dm = \mathbf{0}$$

Inertia matrix

- At a given point q :

$$\mathbf{I}_q = \begin{pmatrix} \int_S (y - y_q)^2 + (z - z_q)^2 dm & -\int_S (x - x_q)(y - y_q) dm & -\int_S (x - x_q)(z - z_q) dm \\ -\int_S (x - x_q)(y - y_q) dm & \int_S (x - x_q)^2 + (z - z_q)^2 dm & -\int_S (y - y_q)(z - z_q) dm \\ -\int_S (x - x_q)(z - z_q) dm & -\int_S (y - y_q)(z - z_q) dm & \int_S (x - x_q)^2 + (y - y_q)^2 dm \end{pmatrix}$$

- A simplified inertia matrix is obtained by placing the origin of the local frame at the mass center, and aligning the axes with the eigenvectors of the inertia matrix:

$$\mathbf{I}_o = \begin{pmatrix} \int_S (y^2 + z^2) dm & 0 & 0 \\ 0 & \int_S (x^2 + z^2) dm & 0 \\ 0 & 0 & \int_S (x^2 + y^2) dm \end{pmatrix}$$

Dynamic actions

- A dynamic action (\mathbf{f}, t_p) applied at a given point p is composed of:
 - a force \mathbf{f} ($kg.m.s^{-2}$)
 - a torque t_p ($kg.m^2.s^{-2}$)
- This action is equivalent with (\mathbf{f}, t_o) applied at point o , with:

$$t_o = t_p + \mathbf{op} \times \mathbf{f}$$



- Dynamic actions can be summed if they are expressed at the same point:

$$(\mathbf{f}, t_o) + (\mathbf{f}', t'_o) = (\mathbf{f} + \mathbf{f}', t_o + t'_o)$$

Solid dynamics

- Newton's law on acceleration

$$f_1 = m\ddot{o}_1$$

- Euler's law on angular acceleration

- In the fixed frame (I_{o_1} varies over time):

$$t_{o_1} = I_{o_1}\dot{\omega}_1$$

- In the local frame (I_{o_1} is constant):

$$t_{o_1} = I_{o_1}\dot{\omega}_1 + \omega_1 \times I_{o_1}\omega_1$$

Solid dynamics

- Six-dimensional notation

$$\begin{pmatrix} f_1 \\ t_{o1} \end{pmatrix} = \begin{pmatrix} m & & & & & \\ & m & & & & \\ & & m & & & \\ & & & I_{xx} & I_{xy} & I_{xz} \\ & & & I_{xy} & I_{yy} & I_{yz} \\ & & & I_{xz} & I_{yz} & I_{zz} \end{pmatrix} \begin{pmatrix} \ddot{o}_1 \\ \dot{\omega}_1 \end{pmatrix}$$

- Compact notation:

$$f_1 = M_1 \dot{v}_1$$

Dynamics of a set of solids

- Matrix notation

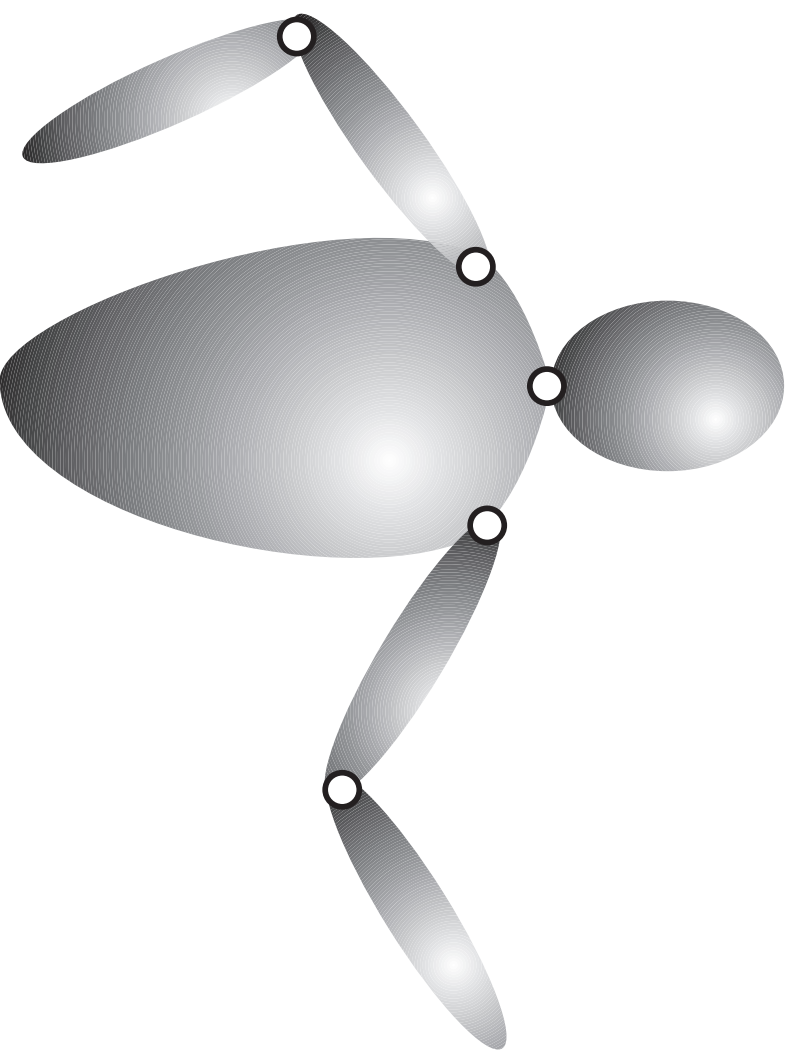
$$\begin{pmatrix} f_1 \\ f_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} M_1 & & \\ & M_2 & \\ & & \dots \end{pmatrix} \begin{pmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \vdots \end{pmatrix}$$

- Compact matrix notation

$$f = M\dot{v}$$

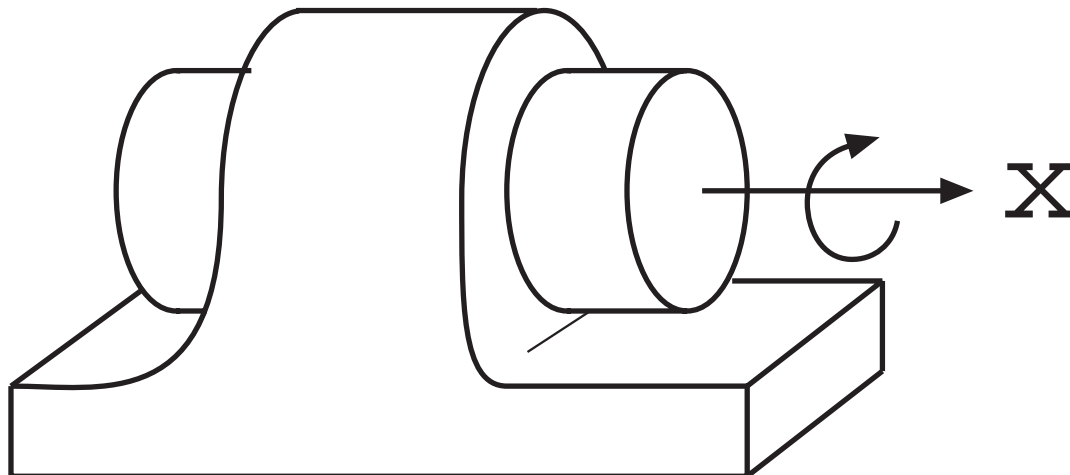
Joints

- Joints include geometric constraints which restrict the relative motion of two solids



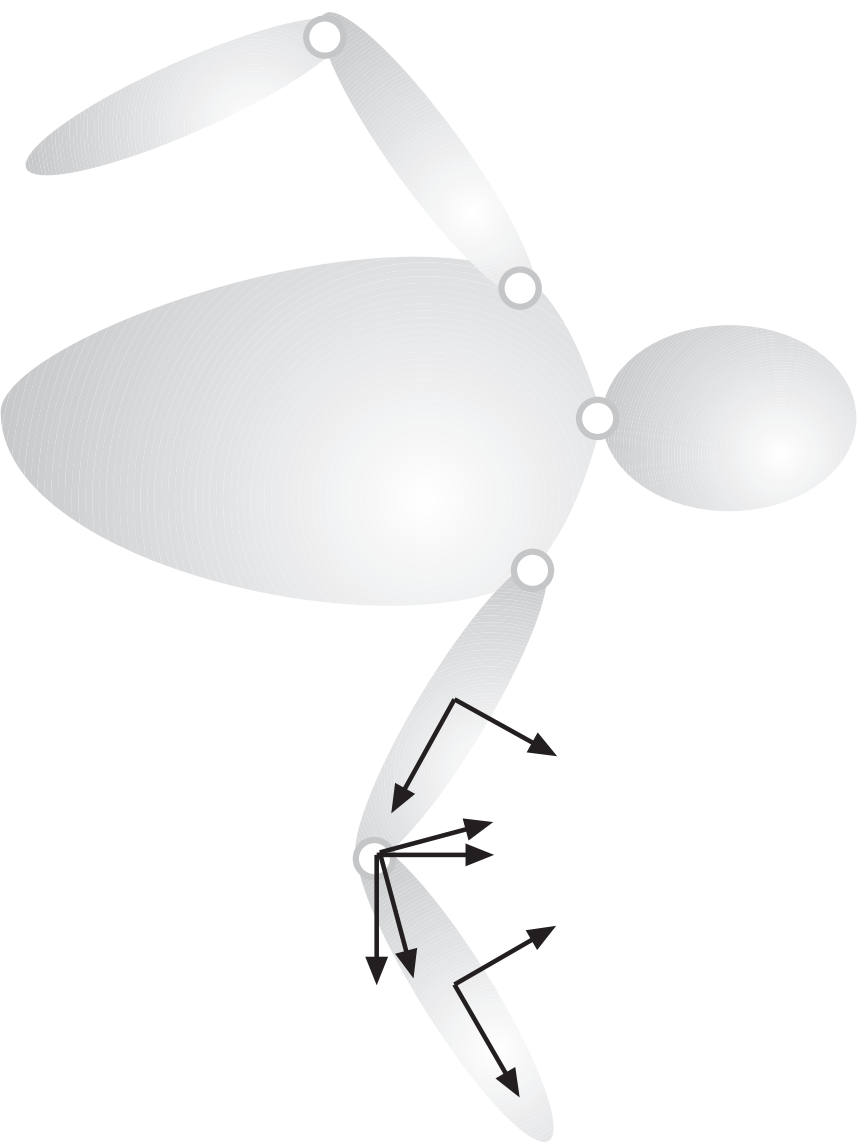
Constraints vs. Degrees of freedom (dof)

- $constraints + dof = 6$
- Example: cylindrical joint. 2 dof, 4 constraints
 - 2 dof: trans x, rot x
 - 4 constraints: trans y,z =0 rot y,z =0



Joint model

- We model joints using two local frames. Joint types are defined by their constraints.



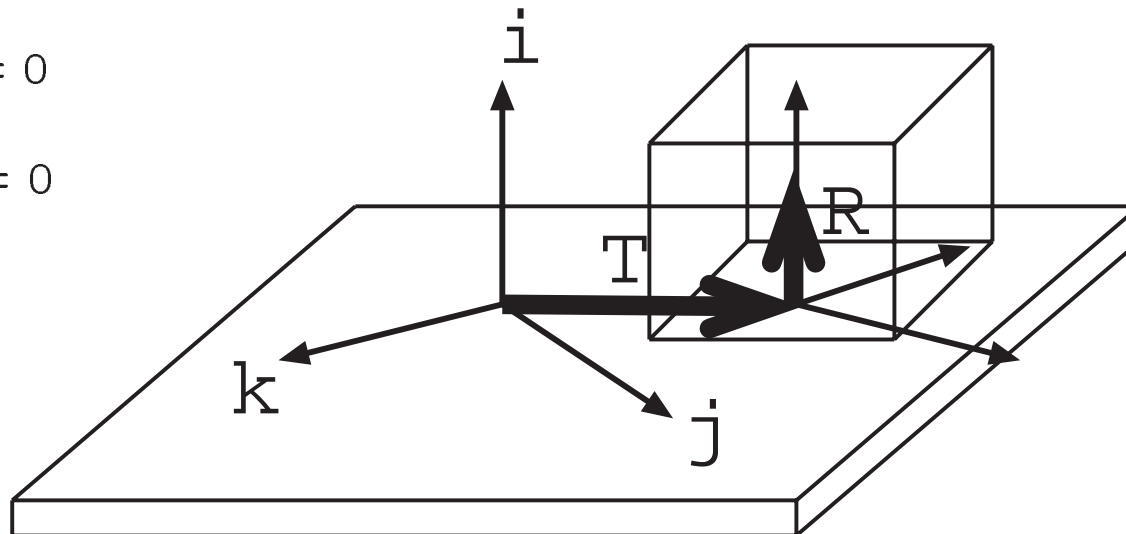
Modeling geometric constraints

- Projections of the relative translation and rotation of the joint frames allow us to model geometric constraints.
- Example: geometric constraints in a plane joint:

- $T \cdot i = 0$

- $R \cdot j = 0$

- $R \cdot k = 0$



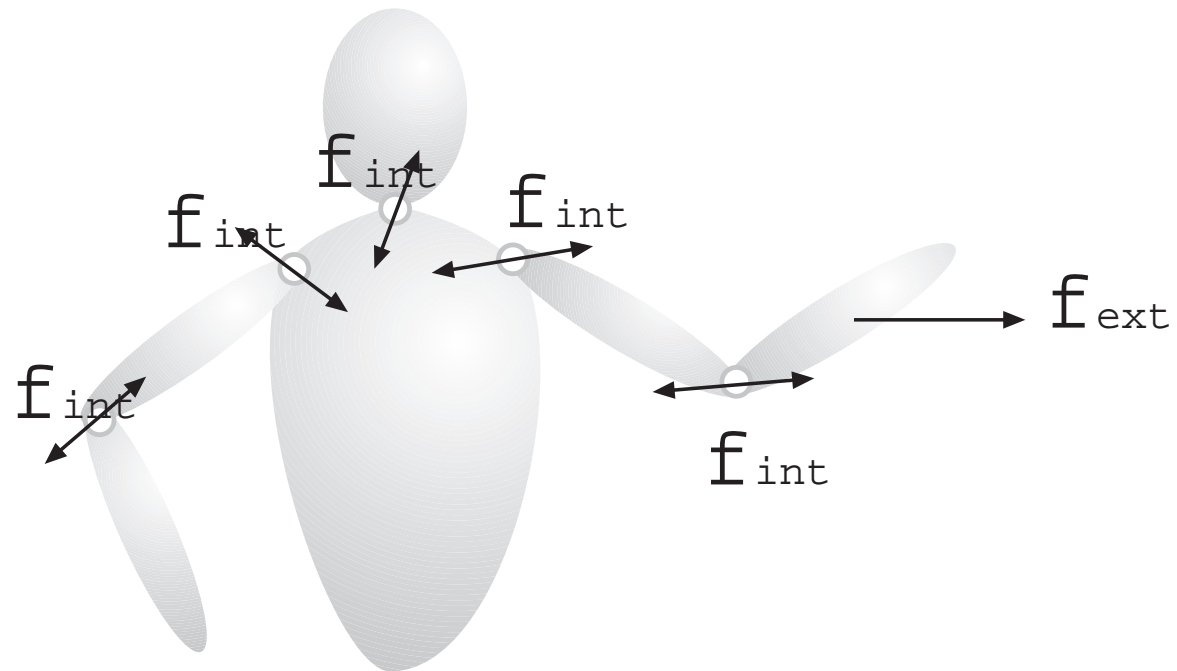
Simulation of constrained solids

- Differential equation with constraints (differential algebraic equation)

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{D}\mathbf{v} && \text{integration of velocities} \\ \mathbf{M}\dot{\mathbf{v}} &= \mathbf{f} && \text{Newton – Euler's law} \\ \mathbf{g}(\mathbf{q}, \mathbf{v}, t) &= \mathbf{0} && \text{constraints} \end{aligned}$$

Constraint forces

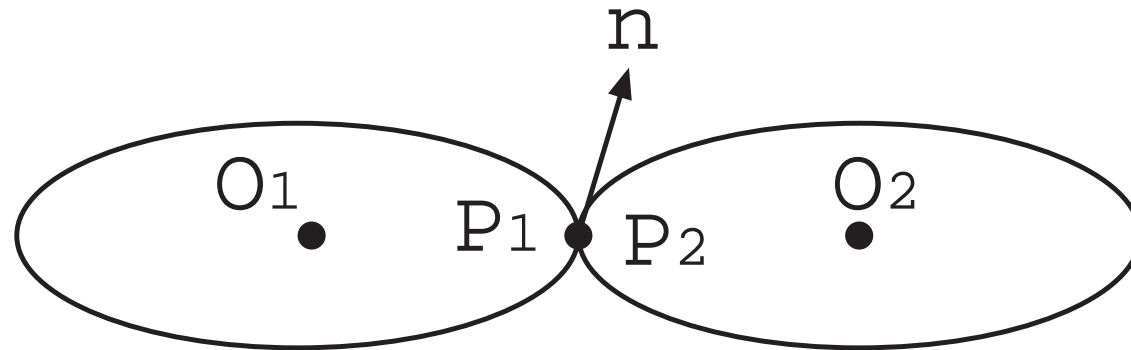
- Constraint forces are necessary to maintain the constraints met.



- Writing and solving an equation system is necessary for the computation of the constraint forces.

Constraint derivative

- Relative velocity along a constraint direction



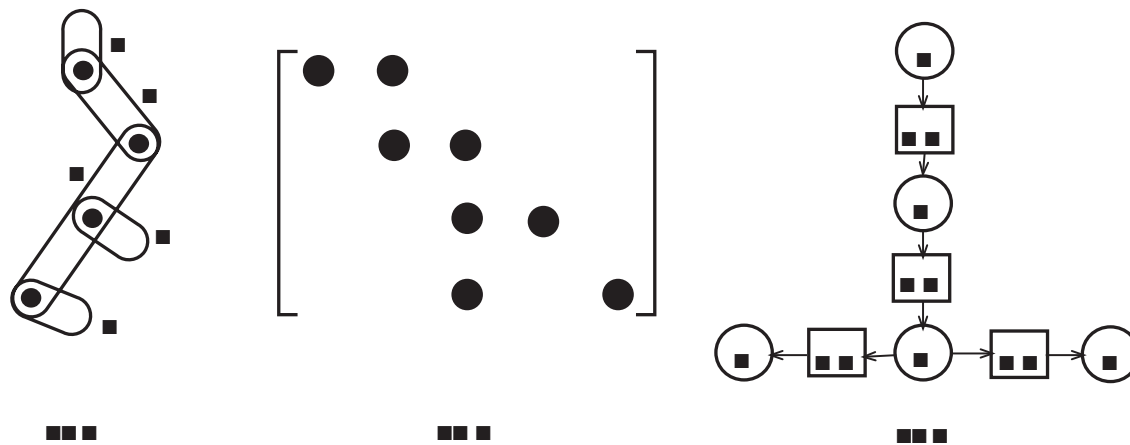
$$\begin{aligned}\dot{\mathbf{p}}_1 &= \dot{\mathbf{o}}_1 + \boldsymbol{\omega}_1 \times \mathbf{o}_1\mathbf{p}_1 \\ \mathbf{n} \cdot \dot{\mathbf{p}}_1 &= \mathbf{n} \cdot \dot{\mathbf{o}}_1 + (\mathbf{o}_1\mathbf{p}_1 \times \mathbf{n}) \cdot \boldsymbol{\omega}_1 \\ \text{let } \mathbf{j}_1 &\equiv (\mathbf{n}^T, (\mathbf{o}_1\mathbf{p}_1 \times \mathbf{n})^T) \\ \text{let } \mathbf{j}_2 &\equiv (\mathbf{n}^T, (\mathbf{o}_2\mathbf{p}_2 \times \mathbf{n})^T) \\ \mathbf{n} \cdot (\dot{\mathbf{p}}_1 - \dot{\mathbf{p}}_2) &= \mathbf{j}_1 \dot{\mathbf{v}}_1 - \mathbf{j}_2 \dot{\mathbf{v}}_2\end{aligned}$$

Constraint derivative

- Matrix expression of the constraint derivative:

$$\dot{g} = Jv$$

J is the **Jacobian** of the constraints. It is sparse.



Second derivative of the constraints

- Relative acceleration along a constraint direction

$$\dot{p}_1 \equiv \dot{o}_1 + \dot{\omega}_1 \times o_1 p_1 + \omega_1 \times (\omega_1 \times o_1 p_1)$$

$$n \cdot \dot{p}_1 = n \cdot \dot{o}_1 + (o_1 p_1 \times n) \cdot \dot{\omega}_1 + n \cdot (\omega_1 \times (\omega_1 \times o_1 p_1))$$

$$\text{let } c_1 \equiv n \cdot (\omega_1 \times (\omega_1 \times o_1 p_1))$$

$$\text{let } c_2 \equiv n \cdot (\omega_2 \times (\omega_2 \times o_2 p_2))$$

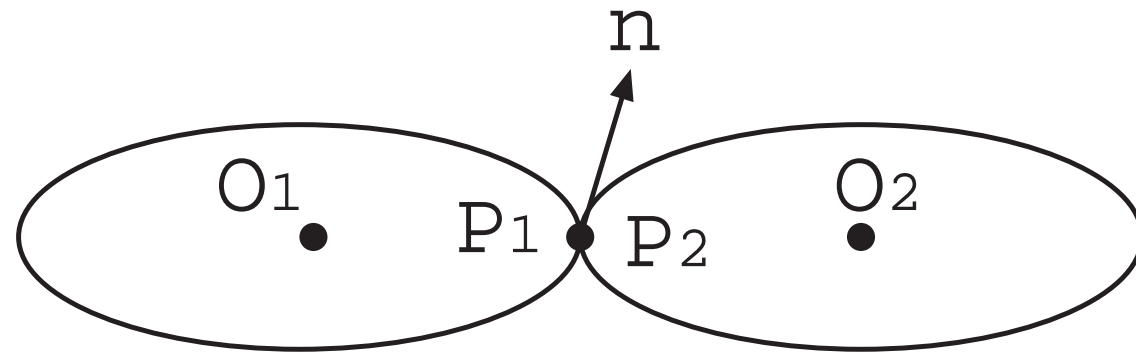
$$n \cdot (\ddot{p}_1 - \ddot{p}_2) = j_1 \dot{v}_1 - j_2 \dot{v}_2 + c_1 - c_2$$

- Matrix expression of the constraint derivative:

$$\ddot{g} = J \dot{v} + c$$

Dynamic actions applied by the constraint forces

- The constraint forces act along the constraint directions



- Action applied by a unit constraint force along n :

$$\begin{aligned} f_1 &= j_1^T \\ f_2 &= -j_2^T \end{aligned}$$

- Actions applied by the constraint forces λ :

$$f = J^T \lambda$$

Articulated solid dynamic equation

- Enforcing $\ddot{\mathbf{g}} = 0$ allows us to write a linear equation system:

$$\ddot{\mathbf{g}} = 0 = \mathbf{J}\dot{\mathbf{i}} + \mathbf{c}$$

$$\mathbf{M}\dot{\mathbf{i}} = \mathbf{J}^T\boldsymbol{\lambda} + \mathbf{f}_{ext}$$

$$0 = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} + \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_{ext} + \mathbf{c}$$

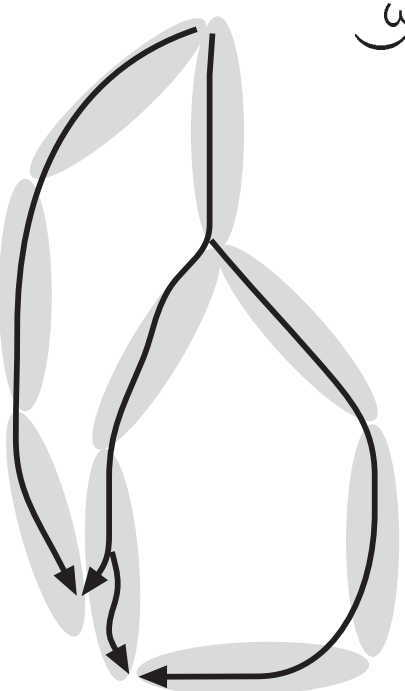
$$0 = \mathbf{A}\boldsymbol{\lambda} + \mathbf{b}$$

- The solution $\boldsymbol{\lambda}$ of the equation system allows us to compute the acceleration of the solids:

$$\dot{\mathbf{i}} = \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} + \mathbf{M}^{-1}\mathbf{f}_{ext}$$

Equation solution

- $(JM^{-1}J^T)\lambda + b = 0$
- Straightforward solution: $O(n^3)$
- Structured solutions: $O(a + al + l^3)$
Featherstone 87, Baraff 96
 - a acyclic constraints
 - l loop constraints
- Conjugate gradient solution: $O(n^2)$
Gleicher 94

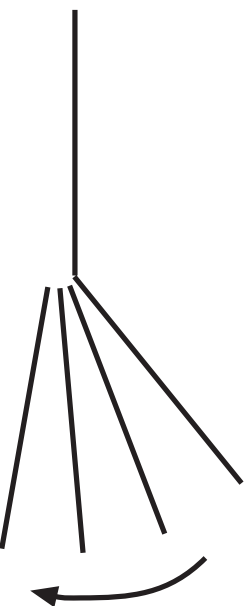


- **Bottleneck:** cubic complexity, or poor efficiency for acyclic structures.

Constraint stabilization

- Accelerations can be integrated using different schemes (Euler, Runge-Kutta...)

- Numerical integration generates constraint drift



- Baumgarte stabilizers are used to limit the drift

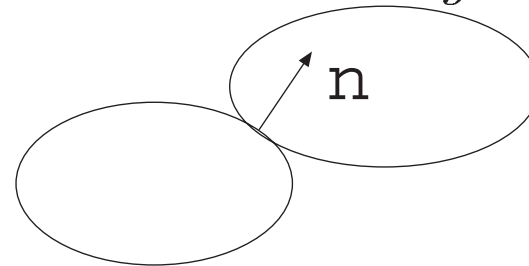
$$\ddot{\mathbf{q}} = -\alpha \mathbf{g} - \beta \dot{\mathbf{q}}$$

- **Bottleneck:** Baumgarte stabilization induces stiffness. Small time steps are necessary.

Contact forces

- Bodies in contact can not interpenetrate but are free to move away from each other

$$\begin{aligned}\lambda_n &\geq 0 && \text{contact force along the normal} \\ a_n &\geq 0 && \text{relative acceleration along the normal} \\ a_n f_n &= 0\end{aligned}$$



- Formulation as a linear complementarity problem

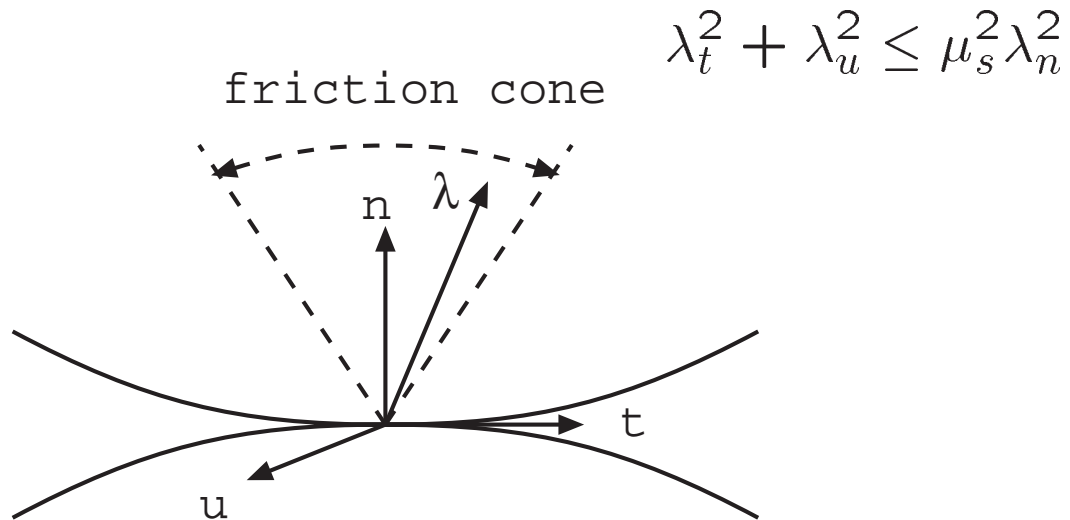
$$A\lambda + b \geq 0, \quad \lambda \geq 0, \quad \lambda^T (A\lambda + b) = 0$$

- Formulation as a quadratic programming problem

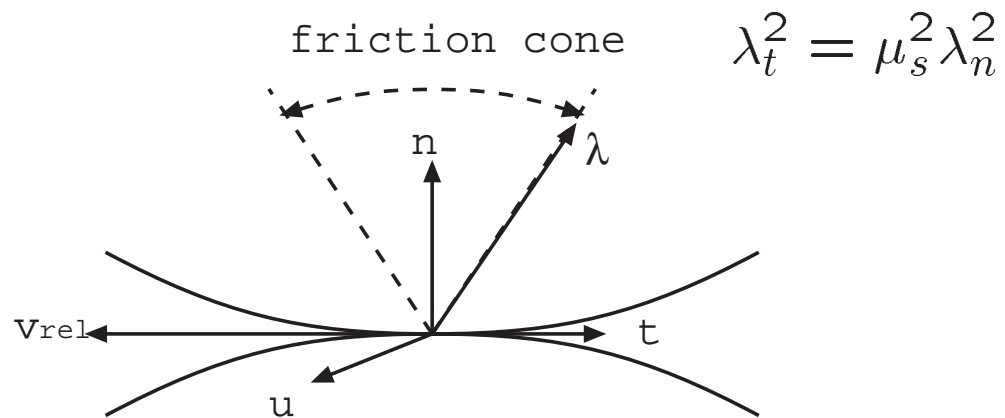
$$\min_{\lambda} \lambda^T (A\lambda + b) \quad \text{subject to} \quad \begin{cases} A\lambda + b \geq 0 \\ \lambda \geq 0 \end{cases}$$

Friction

- Static friction (null tangential motion)



- Dynamic friction (opposed to tangential motion)

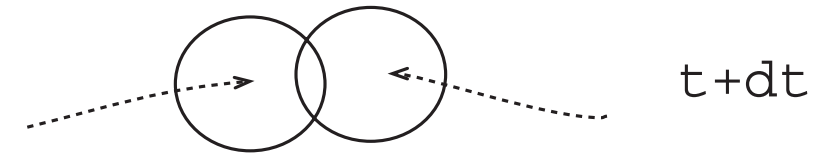


Contact force computation

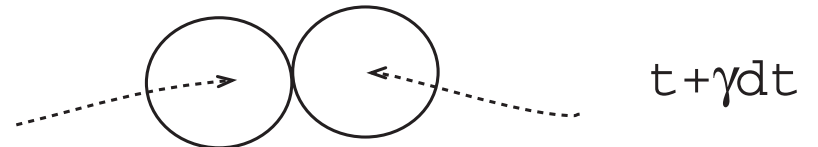
- Pairwise contact force computation
Hahn 88, Moore&Wilhelms 88, Mirtich 96
Bottleneck: propagation
- Global computation
 - Lötstedt 82: quadratic programming. **Bottleneck:** poor convergence
 - Baraff 94: linear complementarity. **Bottleneck:** at least $O(n^3)$

Collision

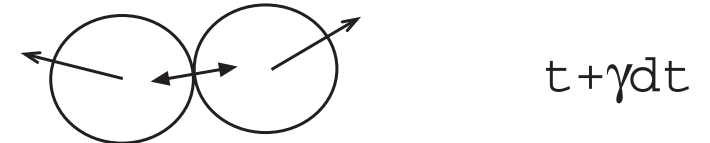
- Collision detection



- Backtracking



- Impulsion



– dynamic solution with λ ($kg.m.s^{-1}$) and \mathbf{b} ($m.s^{-1}$)

– bouncing: use $(1 + \alpha)\mathbf{b}$ or apply $(1 + \alpha)\lambda$

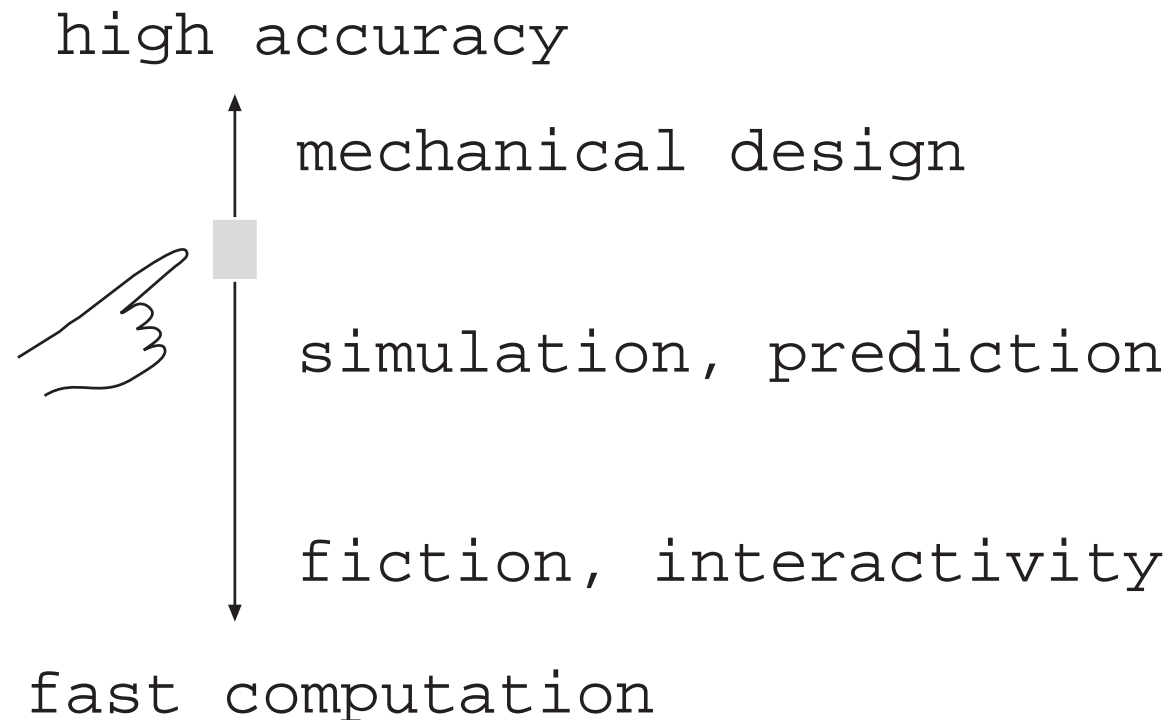
- **Bottlenecks**

– collision detection

– small time steps due to multiple collisions

Trading-off accuracy for computation time

- The need for accuracy depends on the type of application



Contributions

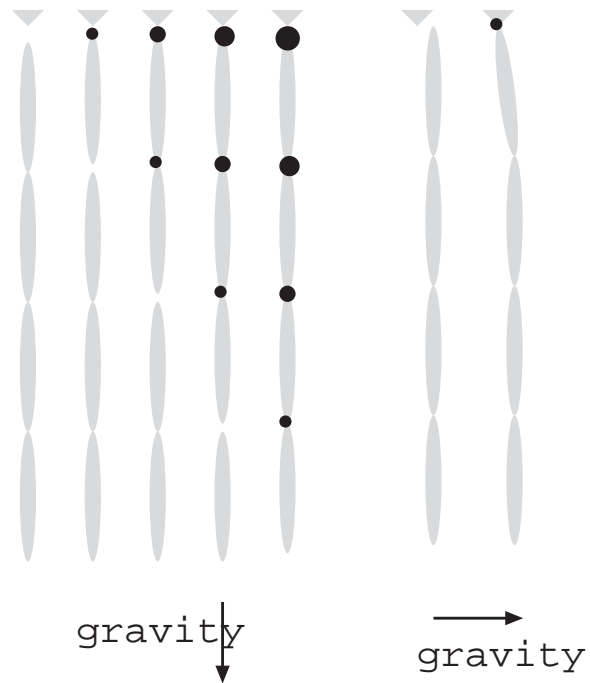
- Fast, refinable equation solution of articulated solid dynamics
- Efficient stabilization of the constraints
- Fast, refinable contact force computation for frictionless contacts

Conjugate gradient algorithm

- Solves $Ax = b$ iteratively
with A $n \times n$, symmetric positive definite
- Addresses A only through its product with a vector
- Computes the solution in less than n steps
 $\Rightarrow \begin{cases} \text{time} & O(n^2) \\ \text{space} & O(n) \end{cases}$
- The biconjugate gradient algorithm handles indefinite matrices

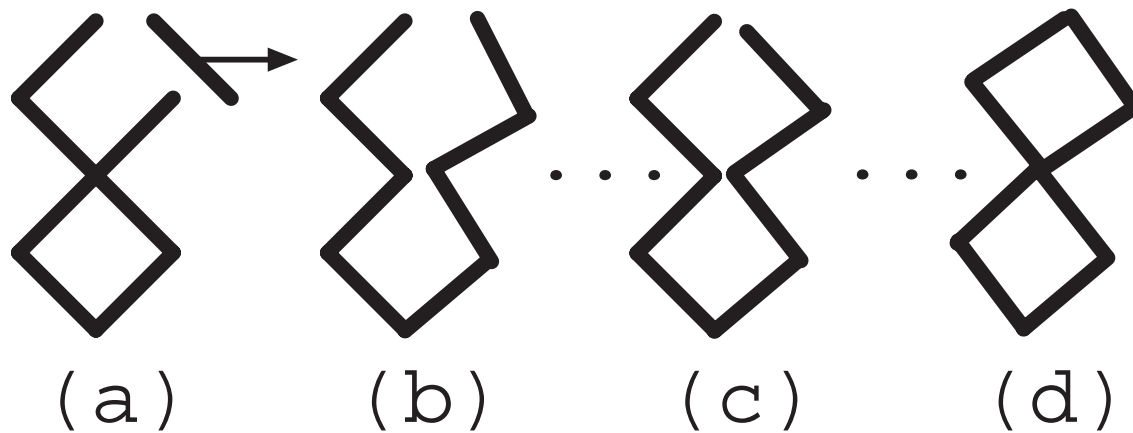
Constraint propagation within the conjugate gradient solution

- Two examples



Iterative structured solid dynamics

- Apply linear-time solutions to the acyclic subgraph
- Solve the closed loops iteratively



Structured solution

- We separate acyclic constraints from loop constraints

$$\begin{pmatrix} \boxed{J_a} \\ \boxed{J_l} \end{pmatrix} M^{-1} \begin{pmatrix} \boxed{J_a^T} & \boxed{J_l^T} \end{pmatrix} \begin{pmatrix} \lambda_a \\ \lambda_l \end{pmatrix} = \begin{pmatrix} b_a \\ b_l \end{pmatrix}$$

- We can compute $(J_a M^{-1} J^T)^{-1} x$ in linear time
(Baraff 96)

Structured solution

- Rewrite the equation system

$$\text{let } \mathbf{A}_{xy} \equiv \mathbf{J}_x \mathbf{M}^{-1} \mathbf{J}_y^T$$

$$\begin{pmatrix} A_{aa} & A_{al} \\ A_{la} & A_{ll} \end{pmatrix} \begin{pmatrix} \lambda_a \\ \lambda_l \end{pmatrix} = \begin{pmatrix} b_a \\ b_l \end{pmatrix}$$

- Perform the substitution

$$\begin{aligned} \lambda_a &= \mathbf{A}_{aa}^{-1} (\mathbf{b}_a - \mathbf{A}_{al} \lambda_l) \\ (\mathbf{A}_{ll} - \mathbf{A}_{la} \mathbf{A}_{aa}^{-1} \mathbf{A}_{al}) \lambda_l &= \mathbf{b}_l - \mathbf{A}_{la} \mathbf{A}_{aa}^{-1} \mathbf{b}_a \end{aligned}$$

- Compute the closed loop forces

$$\hat{\mathbf{A}} \lambda_l = \hat{\mathbf{b}}$$

Solution of the closed loop equation system

- Use the biconjugate gradient algorithm to solve $\hat{A}\lambda_l = \hat{b}$
 - addresses only the matrix through its product $\hat{A}x$ or $\hat{A}^T x$
 - performs an iterative minimization of $(\hat{A}x - \hat{b})^2$

- products can be computed in linear time

$$\hat{A}x = (J_l(1 - M^{-1}J_a^T A_{aa}^{-1}J_a)M^{-1}J_l^T)x$$

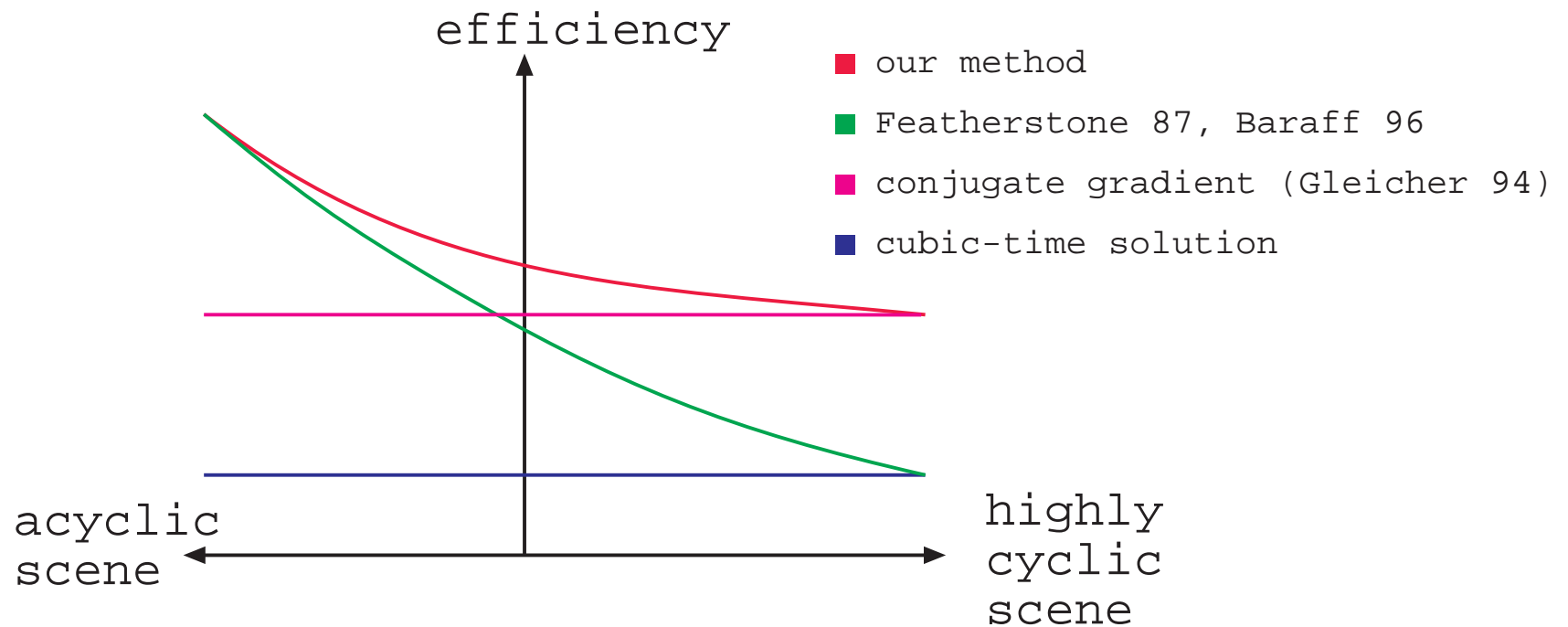
Efficiency

- Complexity (a acyclic constraints, l loop constraints)

	time $O()$	space $O()$
full dense	$(a + l)^3$	$a^2 + al + l^2$
full iterative	$a^2 + al + l^2$	$a + l$
structured dense	$a + al + l^3$	$a + l + l^2$
our method	$a + al + l^2$	$a + l$

Efficiency

- Compared efficiencies

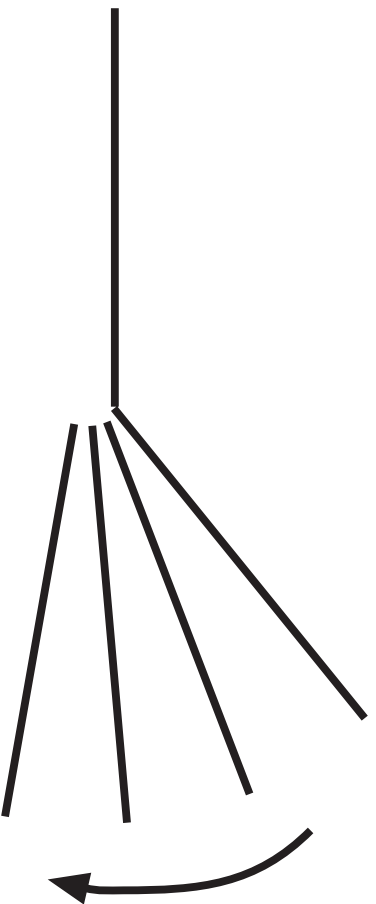


Trading-off efficiency for computation time

- We can terminate the iterative solution:
 - either when a given accuracy is reached
 - or when a given number of iterations have been performed
- We can perform approximate solutions much faster than accurate solutions.

Error accumulation

- Wrong accelerations lead to wrong velocities
- Wrong velocities lead to wrong positions



Constraint stabilization

- Avoid Baumgarte stabilizers ($\ddot{\mathbf{g}} = -\alpha\mathbf{g} - \beta\dot{\mathbf{g}}$)
 - difficult to chose
 - induce stiffness
- Use projection methods
 - project the velocities to a subspace compatible with the constraints
 - project the positions to a subspace compatible with the constraints

A projection method

- Integrate time using your favorite integration scheme

$$(\mathbf{q}, \mathbf{v}, \dot{\mathbf{v}})(t) \longrightarrow (\tilde{\mathbf{q}}, \tilde{\mathbf{v}})(t + dt)$$

- Correct positions and velocities

$$\begin{aligned}\mathbf{q} &= \tilde{\mathbf{q}} - M^{-1}(\mathbf{J}M^{-1}\mathbf{J}^T)^{-1}\mathbf{g} \\ \mathbf{v} &= \tilde{\mathbf{v}} - M^{-1}(\mathbf{J}M^{-1}\mathbf{J}^T)^{-1}\dot{\mathbf{g}}\end{aligned}$$

- Use the iterative structured solution to compute the corrections !

Modeling the errors

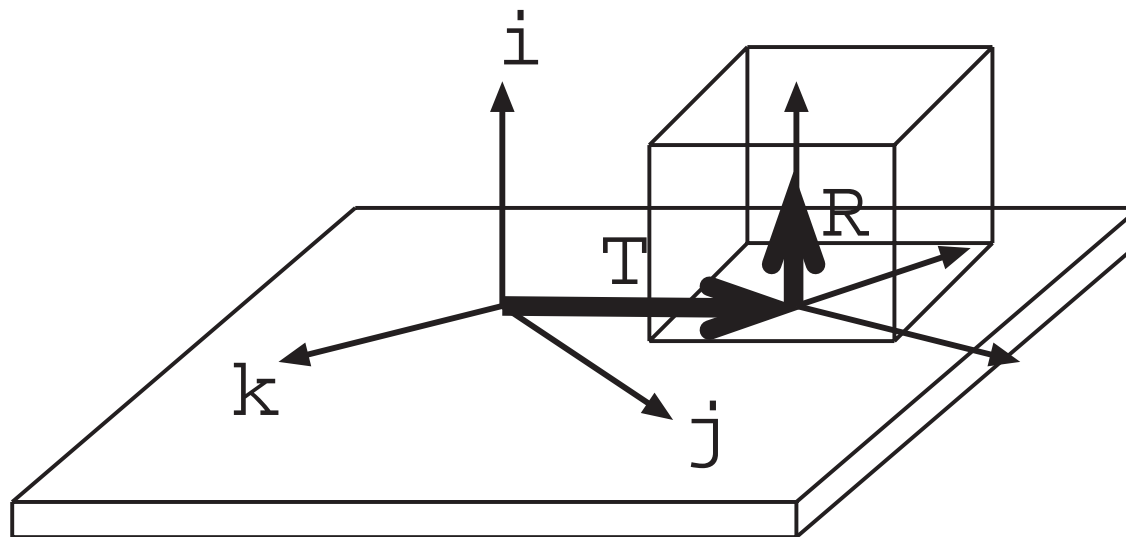
- Velocity errors

$$\dot{g} = Jv$$

- Position errors: project the relative positions to the axes of the joints.

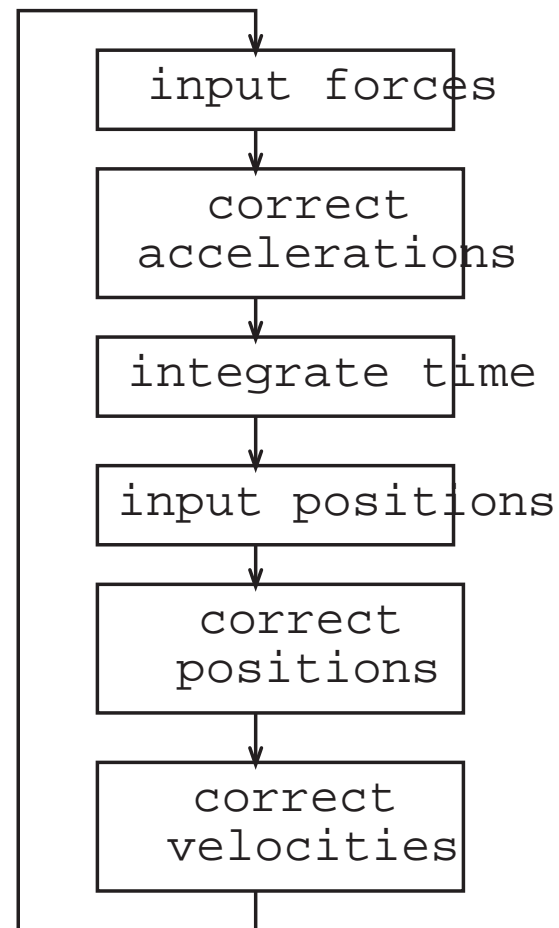
Example: errors within a plane joint

- $T.i$
- $R.j$
- $R.k$

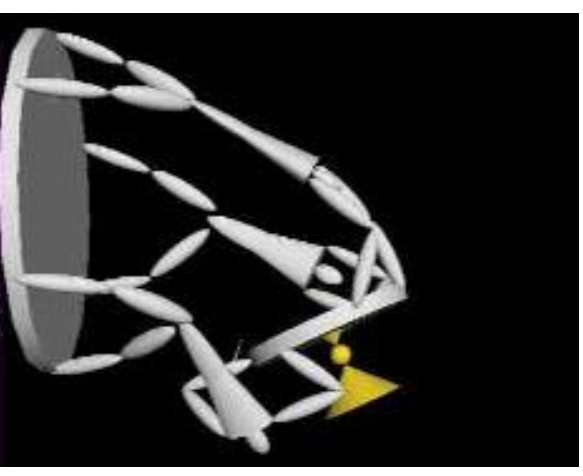
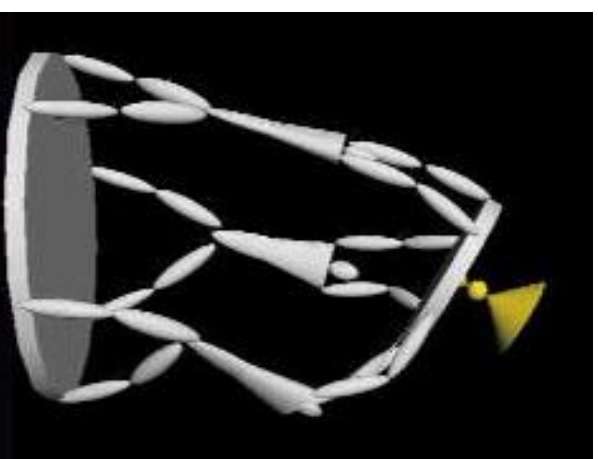
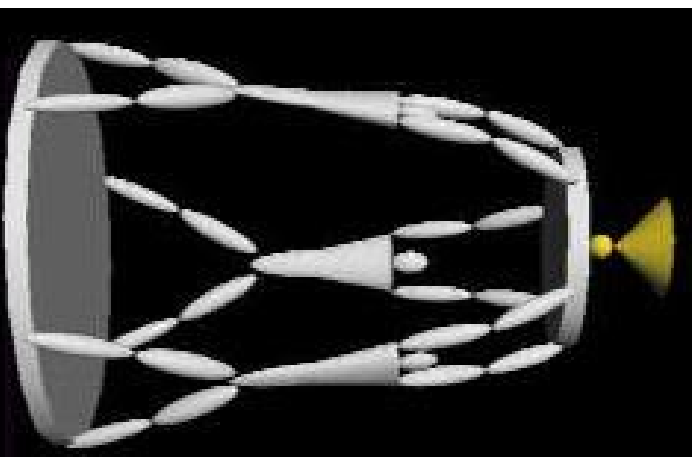
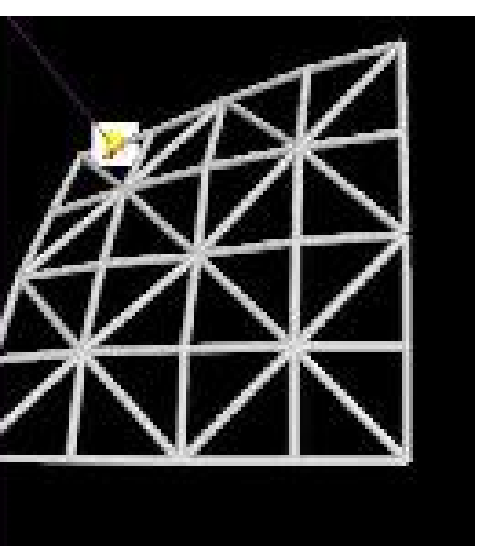
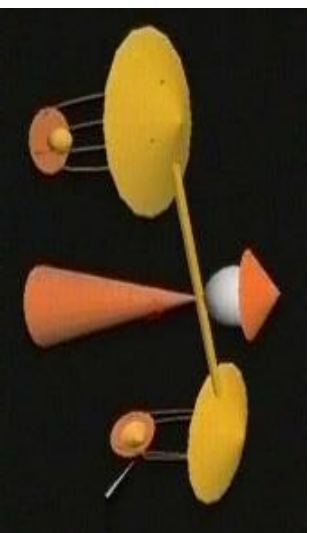
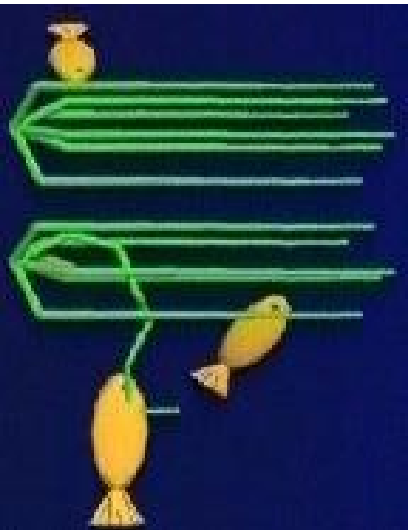


Animation loop with error control

- Accelerations, velocities as well as positions are refined

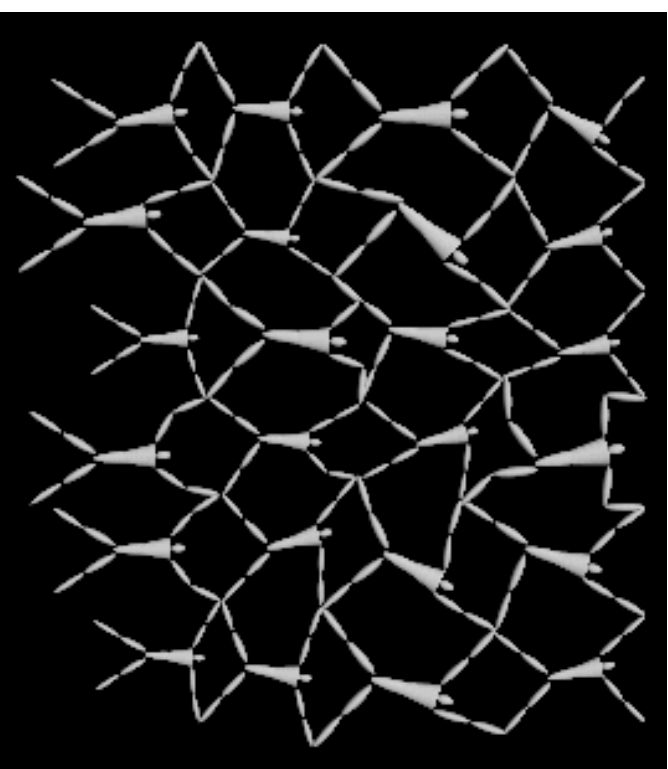
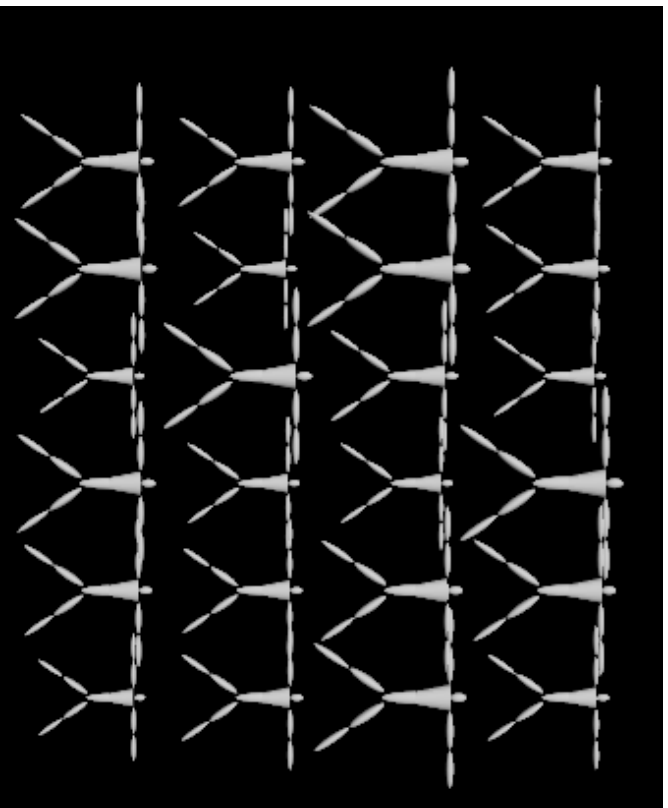


Application of error control to real-time animation



Application of error control to assembly

- Assembly of a complex structure



Fast contact force computation

- Apply iterative refinement to the contact equations

$$\min_f \mathbf{f}^T (\mathbf{A}\mathbf{f} + \mathbf{b}) \quad \text{subject to} \quad \begin{cases} \mathbf{A}\mathbf{f} + \mathbf{b} \geq 0 \\ \mathbf{f} \geq 0 \end{cases}$$

where \mathbf{f} are the contact forces.

- The biconjugate gradient minimizes $(\mathbf{A}\mathbf{f} + \mathbf{b})^2$

Reduced equation system

- Clamped and vanishing contacts
 - C is the set of clamped contacts: $a_i = 0, f_i \geq 0$
 - NC is the set of vanishing contacts: $f_i = 0, a_i \geq 0$
- Reduced system

$$A_C f_C + b_C = 0$$

- If C is well chosen then the inequalities are satisfied

Heuristic approach

- start from an arbitrary C
- solve $A_C f_C + b_C = 0$
- check the inequalities:
 - if $i \in NC$ and $a_i < 0$
move i to C
 - if $i \in C$ and $f_i < 0$
move i to NC
- repeat until all inequalities are satisfied

Iterative solution of the linear system

- Solve $A_G f_G + b_G = 0$ using the biconjugate gradient algorithm
- Filter the vectors to restrict the solution to the clamped contacts

```
procedure filter( $x$ )  
  for  $i = 1$  to  $n$   
    if  $s_i = \text{vanishing}$   
       $x_i = 0$   
return  $x$ 
```

Fast heuristic approach using the biconjugate gradient algorithm

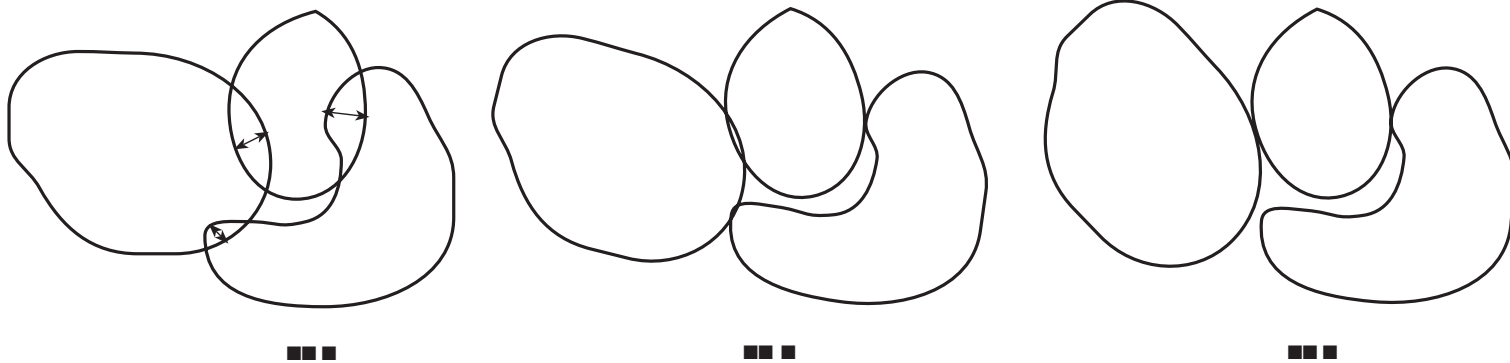
- insert inequality checking within the minimization loop
 1. compute a search direction $p^{(j)}$
 2. compute a step length α
 3. perform the step:
$$f^{(j+1)} = f^{(j)} + \alpha p^{(j)}$$
$$r^{(j+1)} = \text{filter}(A f^{(j+1)} + b)$$
 4. check the inequalities and recompute $r^{(j+1)}$
 5. check termination according to $r^{(j+1)}$

Convergence

- Early detection of inequality violations
- Minimization of $(A_G \mathbf{f}_G + \mathbf{b}_G)^2$
- Trading-off accuracy for computation time

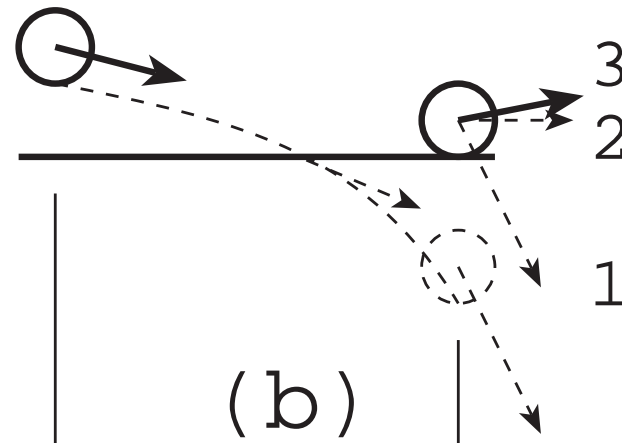
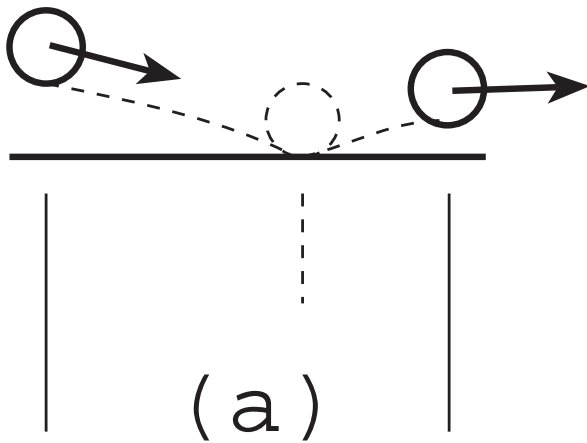
Collision processing

- Collision backtracking is unnecessary due to position correction

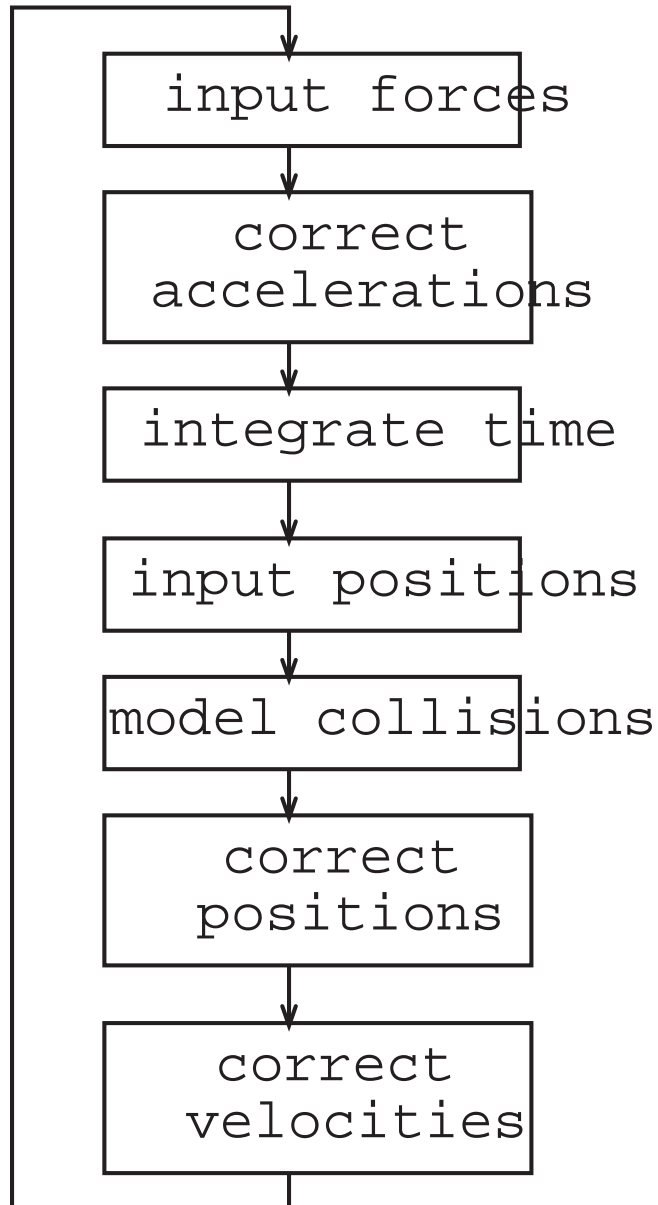


Collision processing

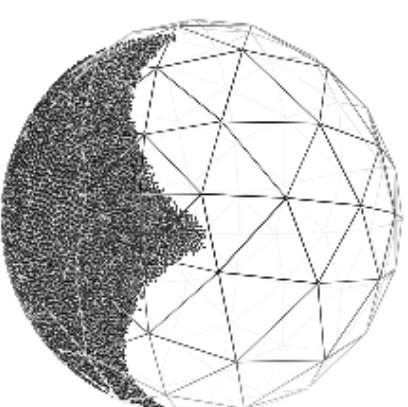
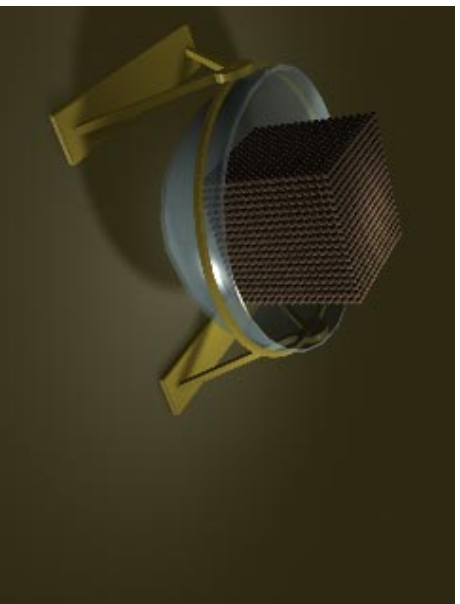
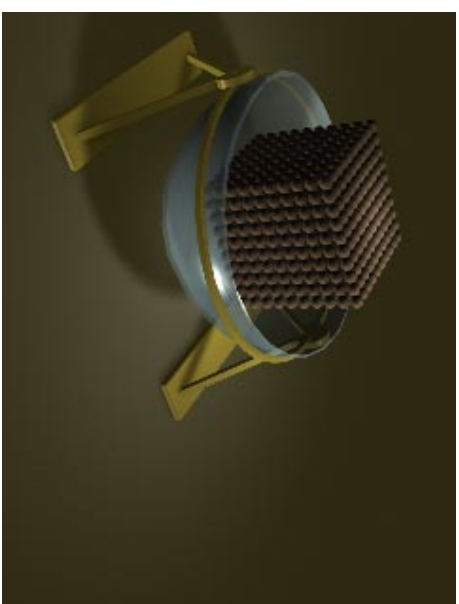
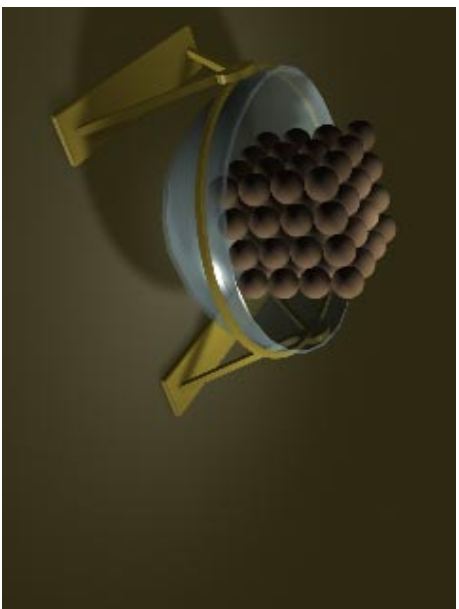
- Bouncing is applied at the end of the time step



Animation loop



Scalability



Applications

