Politecnico di Milano

7/11/2013

Numerical Methods for Large Scale Non-Smooth Multibody Problems

Ing. Alessandro Tasora

Dipartimento di Ingegneria Industriale
Università di Parma, Italy

tasora@ied.unipr.it
http://ied.unipr.it/tasora

Multibody simulation today
Multibody simulation tomorrow

THE COMPLEXITY CHALLENGE

Background

Joint work in the multibody field with

- M.Anitescu (ARGONNE National Labs, Chicago University)
- D.Negrut & al. (University of Wisconsin – Madison)
- J.Kleinert & al. (Fraunhofer ITWM, Germany)
- F.Pulvirenti & al. (Ferrari Auto, Italy)
- NVidia Corporation (USA)
- A.Jain (NASA – JPL)
- S.Negrini & al. (Politecnico di Milano, Italy)
- ENSAM Labs (France)
- Realsoft OY (Finland)

Structure of this lecture

Sections

- Concepts and applications
- Coordinate transformations
- Dynamics: a theoretical background
- A typical direct solver for classical MB problems
- Iterative method for nonsmooth dynamics
- Software implementation
- C++ implementation of the HyperOCTANT solver in Chrono::Engine
- Examples
- Future challenges

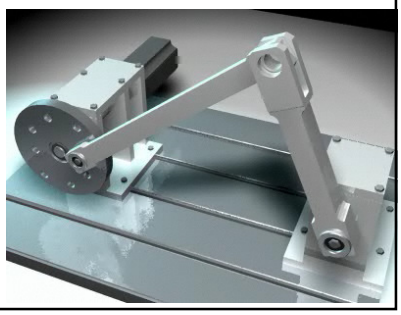
Section

Multibody simulation: concepts and applications

Introduction

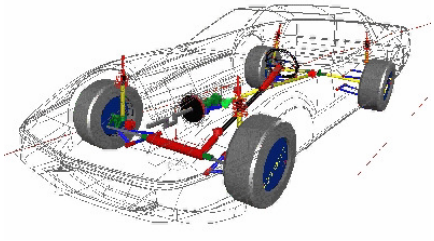
Multibody methods:

- Usually *general-purpose*: they can model many types of problems
- Solve motion equations *automatically*
- Should support an *arbitrary number* of parts, forces, geometries, constraints…
- Most often use *numerical methods* to compute simulations
- Often integrated in CAD tools, with GUI (*graphical user interfaces)*



Main types of multibody analyses

- **Statics**
- **Kinematics**
  - direct
  - inverse
- **Dynamics**
  - Large motions
  - Linearized motion

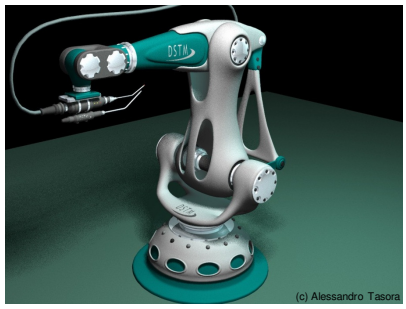


- **Modal analysis, with eigenvectors/eigenvalues**
- **Sensitivity analysis**
- **Local optimization**
- **Global optimization and topological synthesis**

Applications of multibody methods

Robotics
- Direct kinematics
- Inverse kinematics
- Dynamics
- Optimization of robot design

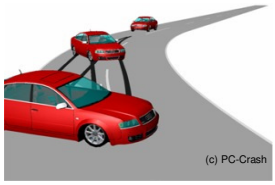
Automotive
- Optimization of car suspensions
- Train dynamics
- Handling simulation
- Comfort
- Accident reconstruction



Applications of multibody methods

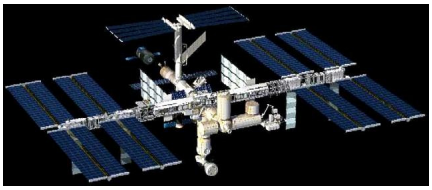
Crash tests

- Accident reconstruction
- Optimization of safety devices
- ….



Aerospace engineering

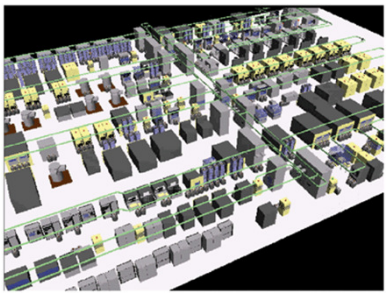
- Setellite deployment
- Balistics
- Flight simulation
- Landing probes
- Simulation of complex mechanisms (helicopter rotors, landing gears, etc.)
- …

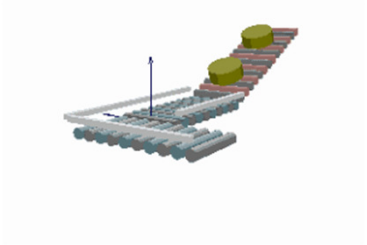


Applications of multibody methods

Automation

- Automated plant simulation
- Optimal selection of servo motors
- Mixed simulations (pneumatics+mechanics, etc.) in mechatronics
- Part feeders
- Size segregation machines
- Conveyor belts
- …





Applications of multibody methods

Generic applied mechanics

- Power trains, gears,
- Intermitting devices
- Cams, followers
- Clock mechanisms
- ...

(c) Alessandro Tasora

Applications of multibody methods

Articulated mechanism: synthesis

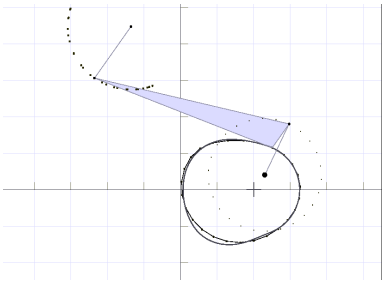
- Analytic synthesis
- Genetic synthesis
- Optimizations
- Topologic synthesis

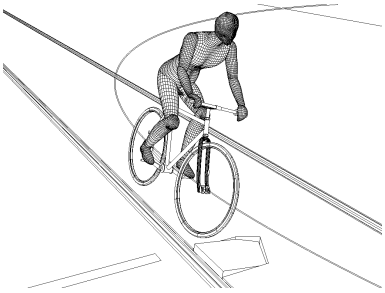
Virtual reality

- Environment simulation
- Training
- Vehicle simulation

Biomechanics

- Simulation of new prosthetic devices
- Sport biomechanics
- Motion capture & gait analysis

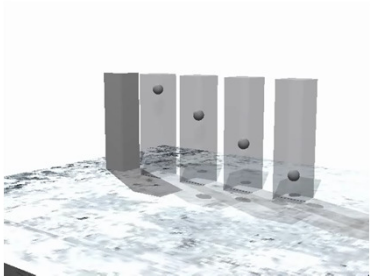


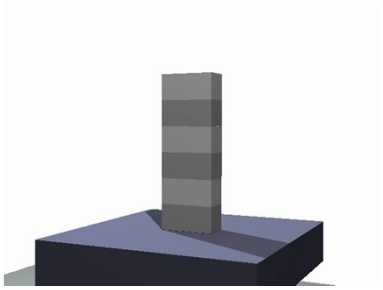


Applications of multibody methods

Civil engineering

- Rocking block dynamics
- Seismic simulations
- Masonry stability





Applications of multibody methods

CAD-CAM-CAE tools

- Simulations, optimizations
- Interactive manipulation of parts in 3D views
- Physical-oriented design

Video games

- Real-time dymamical simulation

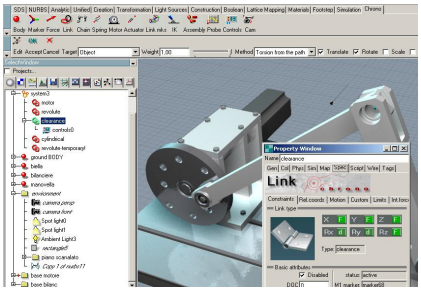
NOTE: 48'000 million of dollars of revenues in videogames, A relevant market for physical simulation software.

Special FX. movies

- Dynamical simulations will soon replace most special effects in films
- Skeletal animation, physical-based animation
- Fake ragdolls, herds, masses

Nuclear engineering

- Simulation of PBR reactors
- Simulation of tele-operating manipulators
- …

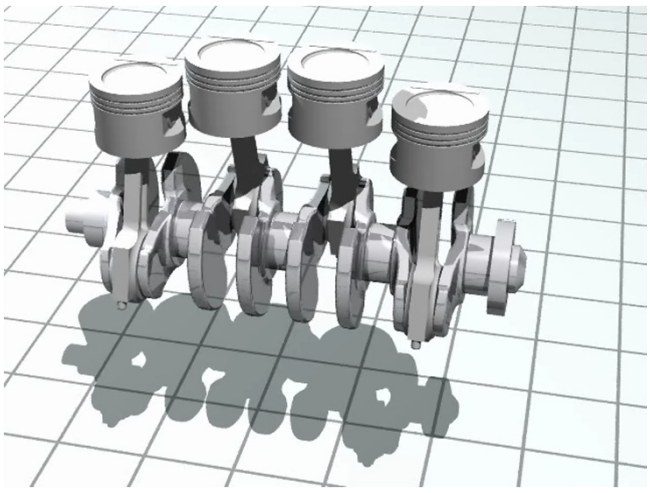


Applications of multibody methods

Tech demo of multibody simulation within a videogame engine (CryTek CryEngine)

Applications of multibody methods

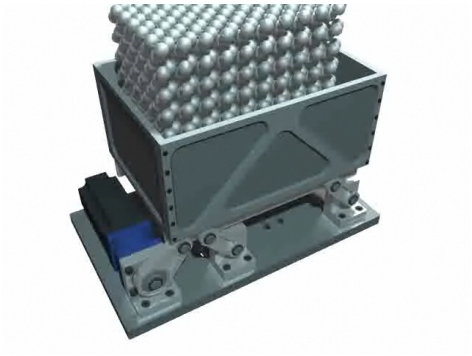
Example: dynamical simulation of an engine



Open problem: complexity

The simulation of massive scenarios with thousands / millions of bodies in contact is still an OPEN PROBLEM

- ***Granular flows***
- ***Rock / soil dynamics***
- ***Packaging***
- ***Size segregation***
- ***Powder mechanics***
- ***Off-road ground/tyre interaction***
- ***Etc.***

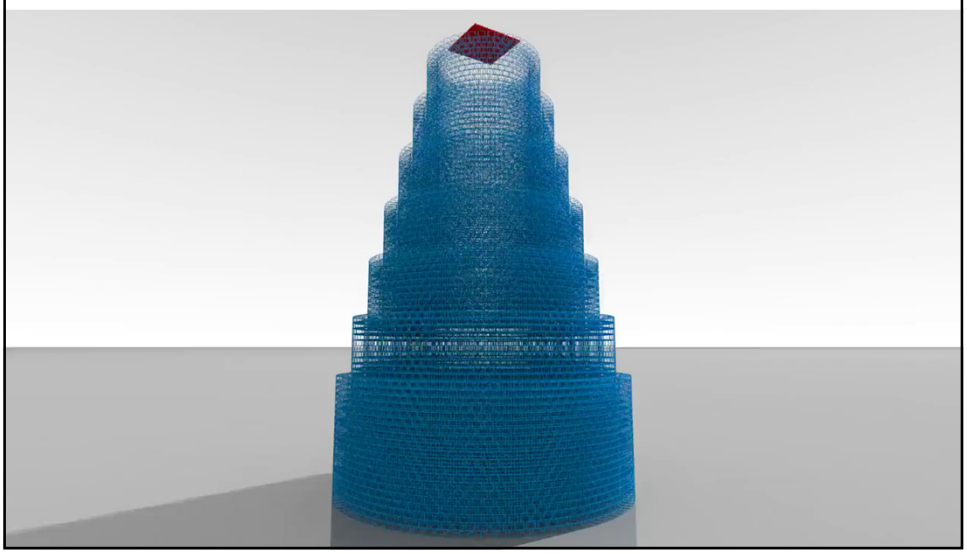


Example: size segregation device: about 2000 interacting objects simulated with our Chrono::Engine software

The non-smooth dynamics of massive scenarios (millions of DOFs) with frictional contact is still an OPEN PROBLEM

- Granular flows
- Soil / sand dynamics
- Powder mechanics
- ...

Example of benchmark of our Chrono::Engine software (H.Mazhar 2012)



Open problem: complexity

Example: bidisperse granular flow in the PBR nuclear reactor

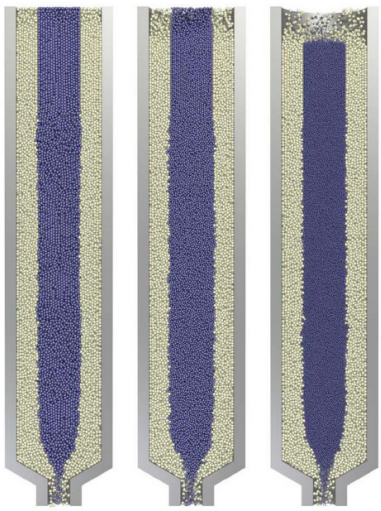
Today simulation use regularization
→ stiff problem, not rigid objects ..

GOAL: FIND A NUMERICAL METHOD WHICH CAN SIMULATE MILLIONS OF RIGID BODIES WITH CONTACTS AND FRICTION

→ Collaboration with Argonne National Laboratories

→ a new method (A.Tasora,M.Anitescu)

→ Collaboration with Dan Negrut (University of Wisconsin-Madison)



Reactor picture: Bazant et al. (MIT and Sandia laboratories).

Section

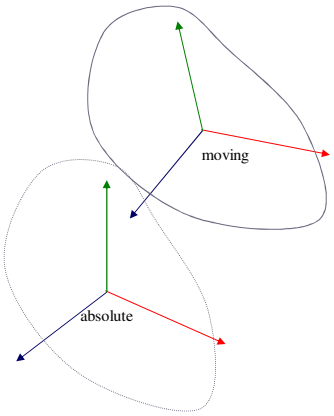
Coordinate transformations

Rigid body motion

We assume bodies to be rigid

Each body has a set of three axis that form a *moving* reference

Motion: 3D translation + 3D rotation



Rigid body motion

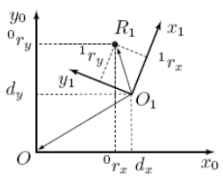
How are body's points transformed?

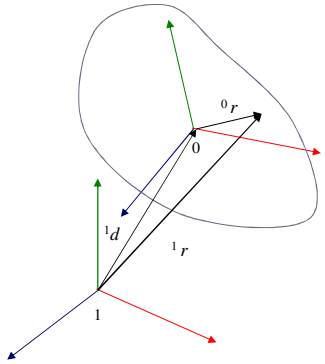
$$\{{}^0r\} = \{\ {}^0r_x \quad {}^0r_y \quad {}^0r_z\ \}^T$$

$$\{{}^1r\} = \{\ {}^1r_x \quad {}^1r_y \quad {}^1r_z\ \}^T$$

Affine linear transformation:

$$\{{}^1r\} = [{}^1_0A]\,\{{}^0r\} + \{{}^1d\}$$

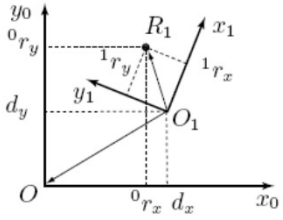




Rigid body motion

The [A] matrix is the rotation matrix (3x3 in 3D, 2x2 in 2D)

Example (in 2D):



$$\left\{ \begin{array}{c} {}^1r_x \\ {}^1r_y \end{array} \right\} = \left[\begin{array}{cc} \cos\theta & +\sin\theta \\ -\sin\theta & \cos\theta \end{array} \right] \left\{ \begin{array}{c} {}^0r_x \\ {}^0r_y \end{array} \right\} + \left\{ \begin{array}{c} {}^1d_{Ox} \\ {}^1d_{Oy} \end{array} \right\}$$

- [A] is built with X,Y versors columns : [A]=[X|Y]
- [A] is hemisymmetric
- [A] does not change distance between points

- Not as easy for 3D, though…

Rigid body motion

The [A] rotation matrix in 3D

$$[A] = \left[\begin{array}{ccc} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{array} \right]$$

Simple rotation, no translation:

$$\left\{ \begin{array}{c} {}^1r_x \\ {}^1r_y \\ {}^1r_z \end{array} \right\} = \left[\begin{array}{ccc} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{array} \right] \left\{ \begin{array}{c} {}^0r_x \\ {}^0r_y \\ {}^0r_z \end{array} \right\}$$

The [A] matrix is orthogonal: $[A]^{-1} = [A]^T$ (does not change distance between points)

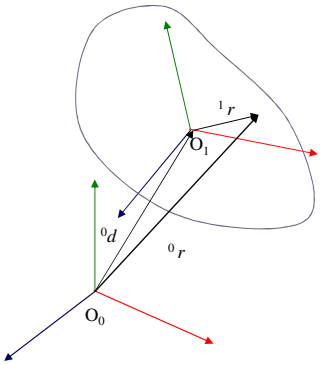
$$[A]\,[A]^T = [I]$$

$$\left\{ \begin{array}{c} {}^0r_x \\ {}^0r_y \\ {}^0r_z \end{array} \right\} = \left[\begin{array}{ccc} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{array} \right] \left\{ \begin{array}{c} {}^1r_x \\ {}^1r_y \\ {}^1r_z \end{array} \right\}$$

Rigid body motion

"Direct" transformation:

$$\{^{\mathbf{0}}r\} = [^{0}_{1}\mathrm{A}]\ \{^{\mathbf{1}}r\} + \{^{\mathbf{0}}d\}$$



"Inverse" transformation:

$$\begin{aligned}\{^{\mathbf{1}}r\} &= [^{0}_{1}\mathrm{A}]^{-1}\ (\{^{\mathbf{0}}r\} - \{^{\mathbf{0}}d\}) \\ &= [^{0}_{1}\mathrm{A}]^{\mathrm{T}}\ (\{^{\mathbf{0}}r\} - \{^{\mathbf{0}}d\}) \\ &= [^{1}_{0}\mathrm{A}]\ \ (\{^{\mathbf{0}}r\} - \{^{\mathbf{0}}d\})\end{aligned}$$

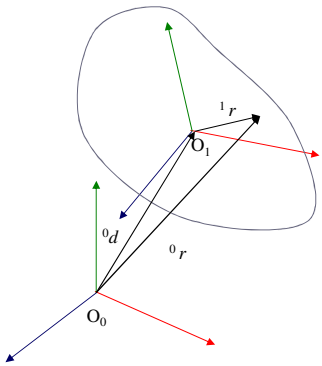
Rigid body motion

Each body should have 3 (translation d) + 3x3=9 (rotation $[^{0}_{1}\mathrm{A}]$) coordinates, that is 12 scalars.

Some would be redundant…

Is it possible to make $[^{0}_{1}\mathrm{A}]$ dependant on only three coordinates?

$$[^{0}_{1}\mathrm{A}(a,b,c)] = f(a,b,c)$$



Rigid body motion

Make $[{}^0_1\mathrm{A}]$ dependant on three angles?...

Different options, depending on the sequence of 3 rotations!

Ex:

$$\{{}^1r\} = [{}^1_0A]\,\{{}^0r\} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 \\ 0 & -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \{{}^0r\}$$

$$\{{}^2r\} = [{}^2_1A]\,\{{}^1r\} = \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \{{}^1r\}$$

$$\{{}^3r\} = [{}^3_2A]\,\{{}^2r\} = \begin{bmatrix} \cos\theta_3 & \sin\theta_3 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \{{}^2r\}$$

$$\{{}^3r\} = [{}^3_2A]\,[{}^2_1A]\,[{}^1_0A]\,\{{}^0r\} = [{}^3_0A]\,\{{}^0r\}$$

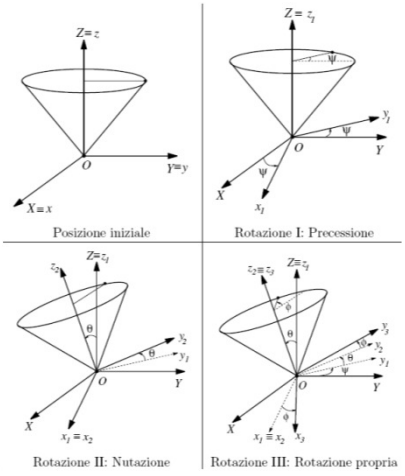
Rigid body motion

Ex: make $[{}^0_1\mathrm{A}]$ dependant on three 'Eulero' angles:

But also:

- 'Cardano' angles
- 'HPB' angles
- 'XYZ' angles,
- etc..

See also 'Rodriguez parameters'



$$\{{}^3r\} = [{}^3_2A]\,[{}^2_1A]\,[{}^1_0A]\,\{{}^0r\} = [{}^3_0A]\,\{{}^0r\}$$

Rigid body motion

Example: sequence X-Y-Z

$$\left[{}^{i}_{0}A\right] = [T]_{\zeta}\,[T]_{\eta}\,[T]_{\xi} = \begin{bmatrix} c_\eta c_\zeta & c_\xi s_\zeta + s_\xi s_\eta c_\zeta & s_\xi s_\zeta - c_\xi s_\eta c_\zeta \\ -c_\eta s_\zeta & c_\xi c_\zeta - s_\xi s_\eta s_\zeta & s_\xi c_\zeta + c_\xi s_\eta s_\zeta \\ s_\eta & -s_\xi c_\eta & c_\xi c_\eta \end{bmatrix}$$

Example: sequence Y-Z-X

$$\left[{}^{i}_{0}A\right] = [T]_{\xi}\,[T]_{\zeta}\,[T]_{\eta} = \begin{bmatrix} c_\zeta c_\zeta & s_\zeta & -s_\eta c_\zeta \\ -c_\xi c_\eta s_\zeta + s_\xi s_\eta & c_\xi c_\zeta & c_\xi s_\eta s_\zeta + s_\xi c_\eta \\ s_\xi c_\eta s_\zeta + c_\xi s_\eta & -s_\xi c_\zeta & -s_\xi s_\eta s_\zeta + c_\xi c_\eta \end{bmatrix}$$

NOTE: viceversa, how to compute ζ, ξ, η from $[A]$?

$\zeta = \text{asin}\,(A_{1,2})$
$\eta = \text{asin}\,(-A_{1,3} / \cos(\zeta))$
$\xi = \text{acos}\,(A_{2,2} / \cos(\zeta))$ → singularity for $\zeta = \pi/2 + n\,\pi$!!! *(Same for all sets of 3 angles!)*

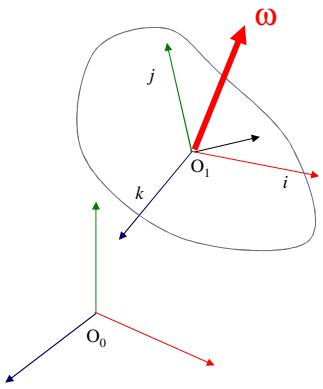
Rigid body motion

Angular velocity

$$\begin{Bmatrix} \frac{d\vec{i}_m}{dt} \\ \frac{d\vec{j}_m}{dt} \\ \frac{d\vec{k}_m}{dt} \end{Bmatrix} = \begin{bmatrix} \tilde{\omega} & 0 & 0 \\ 0 & \tilde{\omega} & 0 \\ 0 & 0 & \tilde{\omega} \end{bmatrix} \begin{Bmatrix} \vec{i}_m \\ \vec{j}_m \\ \vec{k}_m \end{Bmatrix}$$

$$[\tilde{\omega}] = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

$$\left[{}^{0}\tilde{\omega}\right] = \left[{}^{0}_{i}A\right]\left[{}^{i}\tilde{\omega}\right]\left[{}^{0}_{i}A\right]^{T}$$



Rigid body motion

Angular velocity, velocity

$$\vec{v} = \vec{\omega} \times \vec{r}$$

$$\{{}^0 v\} = [{}^0\tilde{\omega}] \{{}^0 r\}$$

$$\{{}^i v\} = [{}^i\tilde{\omega}] \{{}^i r\}$$

$$\{{}^0 v\} = [{}^0_i A] \{{}^i v\}$$
$$\{{}^0 r\} = [{}^0_i A] \{{}^i r\}$$

$$\{{}^i v\} = [{}^0_i A]^T [{}^0\tilde{\omega}] [{}^0_i A] \{{}^i r\}$$

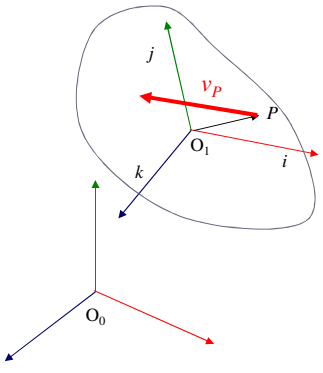
$$\{{}^0 \dot{s}\} = \left[{}^0_i \dot{A}\right] \{{}^i s\} + [{}^0_i A] \{{}^i \dot{s}\}$$

$$\left[{}^0_i \dot{A}\right] = [{}^0_i A] [{}^i\tilde{\omega}]$$
$$\left[{}^0_i \dot{A}\right] = [{}^0\tilde{\omega}] [{}^0_i A]$$

Rigid body motion

Velocity of a point on a moving frame

$$\begin{aligned}\{{}^0 v_P\} &= \{{}^0 v_{O_i}\} + [{}^0_i A] [{}^i\tilde{\omega}] [{}^0_i A]^T \{{}^0 s_P\} \\ &= \{{}^0 v_{O_i}\} + [{}^0_i A] [{}^i\tilde{\omega}] [{}^i_0 A] \{{}^0 s_P\} \\ &= \{{}^0 v_{O_i}\} + [{}^0_i A] [{}^i\tilde{\omega}] \{{}^i s_P\}\end{aligned}$$



Rigid body motion

Acceleration of a point on a moving frame

$$\begin{aligned}\left[{}^0_i\ddot{A}\right] &= \frac{d}{dt}\left(\left[{}^0\tilde{\omega}\right]\right)\left[{}^0_iA\right] + \left[{}^0\tilde{\omega}\right]\left[{}^0_i\dot{A}\right] \\ &= \left[{}^0\tilde{\alpha}\right]\left[{}^0_iA\right] + \left[{}^0\tilde{\omega}\right]\left[{}^0\tilde{\omega}\right]\left[{}^0_iA\right]\end{aligned}$$

$$\begin{aligned}\left\{{}^0a_P\right\} &= \left\{{}^0a_{O_i}\right\} + \left[{}^0\tilde{\alpha}\right]\left\{{}^0s_P\right\} + \left[{}^0\tilde{\omega}\right]\left[{}^0\tilde{\omega}\right]\left\{{}^0s_P\right\} \\ &= \left\{{}^0a_{O_i}\right\} + \left[{}^0\tilde{\alpha}\right]\left\{{}^0s_P\right\} + \left[{}^0_iA\right]\left[{}^i\tilde{\omega}\right]\left[{}^0_iA\right]^T\left[{}^0_iA\right]\left[{}^i\tilde{\omega}\right]\left[{}^0_iA\right]^T\left\{{}^0s_P\right\} \\ &= \left\{{}^0a_{O_i}\right\} + \left[{}^0\tilde{\alpha}\right]\left\{{}^0s_P\right\} + \left[{}^0_iA\right]\left[{}^i\tilde{\omega}\right]\left[{}^i\tilde{\omega}\right]\left[{}^0_iA^T\right]\left\{{}^0s_P\right\} \\ &= \left\{{}^0\ddot{r}_{o_i}\right\} + \left[{}^0\tilde{\alpha}\right]\left\{{}^0s_P\right\} + \left[{}^0_iA\right]\left[{}^i\tilde{\omega}\right]\left[{}^i\tilde{\omega}\right]\left\{{}^is_P\right\} \\ &= \left\{{}^0a_{O_i}\right\} - \left[{}^0\tilde{s}_P\right]\left\{{}^0\alpha\right\} - \left[{}^0_iA\right]\left[{}^i\tilde{\omega}\right]\left[{}^i\tilde{s}_P\right]\left\{{}^i\omega\right\}.\end{aligned}$$

Rigid body motion

Problem: recovering angles from matrix is not always possible (a singularity might happen…)

A solution is to use *quaternions* (4 coordinates for rotation)

Quaternion algebra makes kinematics easier.

Quaternions

- Hypercomplex 4-dimensional numbers
- Associative divisional algebra

$$\mathbf{q} = \mathrm{e}_0 + \mathbf{i}\cdot\mathrm{e}_1 + \mathbf{j}\cdot\mathrm{e}_2 + \mathbf{k}\cdot\mathrm{e}_3$$

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

$$\mathbf{q} = (\mathrm{s}, \mathbf{v}) \qquad \mathbf{q}^* = (\mathrm{s}, -\mathbf{v})$$

$$\|\mathbf{q}\| = \mathbf{q}^* \circ \mathbf{q} = (\mathrm{e}_0^2 + \mathrm{e}_1^2 + \mathrm{e}_2^2 + \mathrm{e}_3^2)$$

Why quaternions for the rotations?

- **No singularities**
- **Compact formalisms**
- ***sin() cos()* never used**
- **Easier analytic constraint jacobians** $[C_q]$

Quaternions

Sum:

$$\begin{aligned}\bar{c} &= \bar{a} \pm \bar{b} = \\ &= (a_0 + a_1\cdot i + a_2\cdot j + a_3\cdot k) \pm (b_0 + b_1\cdot i + b_2\cdot j + b_3\cdot k) = \\ &= (a_0 \pm b_0) \pm (a_1 \pm b_1)\cdot i \pm (a_2 \pm b_2)\cdot j \pm (a_3 \pm b_3)\cdot k\end{aligned}$$

Product:

$$\begin{aligned}\bar{c} &= \bar{a}\cdot\bar{b} := (s_a s_b - \vec{v}_a\cdot\vec{v}_b,\, s_a\vec{v}_b + s_b\vec{v}_a + \vec{v}_a\times\vec{v}_b) \\ &= (a_0 + a_1\cdot i + a_2\cdot j + a_3\cdot k)\cdot(b_0 + b_1\cdot i + b_2\cdot j + b_3\cdot k) = \\ &= (a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3) + \\ &+ (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)\cdot i + \\ &+ (a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)\cdot j + \\ &+ (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)\cdot k\end{aligned}$$

$$\bar{a}\,(\bar{b}\,\bar{c}) = (\bar{a}\,\bar{b})\,\bar{c}$$
$$\bar{a}\,\bar{b} \neq \bar{b}\,\bar{a}$$

Quaternions

Conjugate:

$$\bar{q} = (q_0 + q_1 i + q_2 j + q_3 k)$$
$$\bar{q}^* = (q_0 - q_1 i - q_2 j - q_3 k)$$

$$(\bar{a}^*)^* = \bar{a}$$
$$(\bar{a}\,\bar{b})^* = \bar{b}^*\,\bar{a}^*$$
$$(\bar{a} + \bar{b})^* = \bar{a}^* + \bar{b}^*$$

$$\bar{q}\,\bar{q}^* = (q_0^2 + q_1^2 + q_2^2 + q_3^2)$$
$$\bar{q}\,\bar{q}^* = \bar{q}^*\,\bar{q} = s \in \mathbb{R}$$

$$|\bar{q}| = \sqrt{\bar{q}\,\bar{q}^*}$$
$$|\bar{q}| = \sqrt{(q_0^2 + q_1^2 + q_2^2 + q_3^2)}$$

Inverse:

$$\bar{q}^{-1}\bar{q} = \mathbb{1}$$
$$\bar{q}^{-1} = \bar{q}^* \frac{1}{|\bar{q}|^2} \qquad |\bar{q}| = 1 \;\Rightarrow\; \bar{q}^{-1} = \bar{q}^*$$

Quaternions

Matricial expression for product:

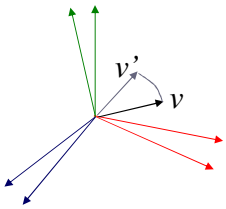
$$\bar{a}\bar{b} = \bar{c}$$

$$\begin{bmatrix} +a_0 & -a_1 & -a_2 & -a_3 \\ +a_1 & +a_0 & -a_3 & +a_2 \\ +a_2 & +a_3 & +a_0 & -a_1 \\ +a_3 & -a_2 & +a_1 & +a_0 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{Bmatrix}$$

Quaternions

Unimodular quaternions can be used to express 3D rotations: $|\bar{q}| = 1$

$$\bar{p}' = \bar{q}\,\bar{p}\,\bar{q}^*$$
$$(0, \vec{v}') = \bar{q}\,(0, \vec{v})\,\bar{q}^*$$



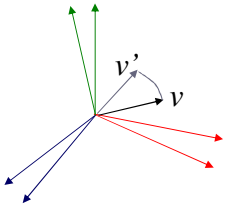
Inverse rotation:

$$\bar{p} = \bar{q}^* \bar{p}'\,\bar{q}$$

Quaternions

That is like rotation with matrix $[\mathrm{A}]$ **:**

$$\bar{p}' = \bar{q}\,\bar{p}\,\bar{q}^*$$
$$(0, \vec{v}') = \bar{q}\,(0, \vec{v})\,\bar{q}^*$$
$$\vec{v}' = [A(q)]\,\vec{v}$$



Matrix $[\mathrm{A}]$ **as a function of a quaternion :**

$$[A(q)] = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_3q_0) & 2(q_1q_3+q_2q_0) \\ 2(q_1q_2+q_3q_0) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(-q_1q_0+q_2q_3) \\ 2(q_1q_3-q_2q_0) & 2(q_1q_0+q_2q_3) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}$$

$$[A(q)] = \begin{bmatrix} +q_1 & +q_0 & -q_3 & +q_2 \\ +q_2 & +q_3 & +q_0 & -q_1 \\ +q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \begin{bmatrix} +q_1 & +q_2 & +q_3 \\ +q_0 & -q_3 & +q_2 \\ +q_3 & +q_0 & -q_1 \\ -q_2 & +q_1 & +q_0 \end{bmatrix} \vec{v}$$

$$[A(q)] = [F(q)_\oplus]\,[F(q)_\ominus]^T\,\vec{v}$$

Quaternions

Viceversa:

(no singularity!)

Algoritmo 1: Calcola quaternione q da matrice [A]

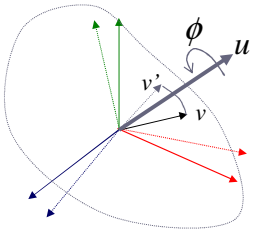
Input: matrice $[A]$
Output: quaternione $\bar{q}$

(1) $t_r = A_{0,0} + A_{1,1} + A_{2,2}$
(2) **if** tr ≥ 0
(3) $\quad s = \sqrt{t_r + 1}$
(4) $\quad q_0 = 0.5s$
(5) $\quad s = 0.5/s$
(6) $\quad q_1 = (A_{2,1} - A_{1,2}) * s$
(7) $\quad q_2 = (A_{0,2} - A_{2,0}) * s$
(8) $\quad q_3 = (A_{1,0} - A_{0,1}) * s$
(9) **else**
(10) $\quad i = 0$
(11) $\quad$ **if** $A_{1,1} > A_{0,0}$
(12) $\quad\quad i = 1$
(13) $\quad\quad$ **if** $A_{2,2} > A_{1,1}$ **then** $i = 2$
(14) $\quad\quad$ **else** $i = 1$
(15) $\quad$ **else**
(16) $\quad\quad$ **if** $A_{2,2} > A_{0,0}$ **then** $i = 2$
(17) $\quad$ **if** $i == 0$
(18) $\quad\quad s = \sqrt{A_{0,0} - A_{1,1} - A_{2,2} + 1}$
(19) $\quad\quad q_1 = 0.5s$
(20) $\quad\quad s = 0.5/s$
(21) $\quad\quad q_2 = (A_{0,1} + A_{1,0})s$
(22) $\quad\quad q_3 = (A_{2,0} + A_{0,2})s;$
(23) $\quad\quad q_0 = (A_{2,1} - A_{1,2})s;$
(24) $\quad$ **if** $i == 1$
(25) $\quad\quad s = \sqrt{A_{1,1} - A_{2,2} - A_{0,0} + 1}$
(26) $\quad\quad q_2 = 0.5s$
(27) $\quad\quad s = 0.5/s$
(28) $\quad\quad q_3 = (A_{1,2} + A_{2,1})s$
(29) $\quad\quad q_1 = (A_{0,1} + A_{1,0})s$
(30) $\quad\quad q_0 = (A_{0,2} - A_{2,0})s$
(31) $\quad$ **if** $i == 2$
(32) $\quad\quad s = \sqrt{A_{2,2} - A_{0,0} - A_{1,1} + 1}$
(33) $\quad\quad q_3 = 0.5s$
(34) $\quad\quad s = 0.5/s$
(35) $\quad\quad q_1 = (A_{2,0} + A_{0,2})s$
(36) $\quad\quad q_2 = (A_{1,2} + A_{2,1})s$
(37) $\quad\quad q_0 = (A_{1,0} - A_{0,1})s$
(38) $\quad$ **return** $\bar{q}$

Quaternions

Quaternion function of angle and axis

$$q_0 = \cos\left(\frac{\phi}{2}\right)$$
$$q_1 = u_x \sin\left(\frac{\phi}{2}\right)$$
$$q_2 = u_y \sin\left(\frac{\phi}{2}\right)$$
$$q_3 = u_z \sin\left(\frac{\phi}{2}\right)$$



Quaternions

Useful conversions

	Algebra dei quaternioni	Algebra matriciale
Trasformazione di coordinate (solo rotazione)	$\bar{p}' = \bar{q}\,\bar{p}\,\bar{q}^* \quad , \quad \bar{p} = (0, \vec{v})$	$\vec{v}\,' = [A]\,\vec{v}$
	$\dot{\bar{p}}' = \dot{\bar{q}}\,\bar{p}\,\bar{q}^* + \bar{q}\,\bar{p}\,\dot{\bar{q}}^* + \bar{q}\,\dot{\bar{p}}\,\bar{q}^*$	$\dot{\vec{v}}\,' = [\dot{A}(q)]\vec{v} + [A(q)]\dot{\vec{v}} \quad [\dot{A}(q)] = [A(q)][\tilde{\omega}_l]$
	$\ddot{\bar{p}}' = \ddot{\bar{q}}\,\bar{p}\,\bar{q}^* + \bar{q}\,\ddot{\bar{p}}\,\bar{q}^* + \bar{q}\,\bar{p}\,\ddot{\bar{q}}^* + 2\,\dot{\bar{q}}\,\dot{\bar{p}}\,\bar{q}^* + 2\,\dot{\bar{q}}\,\bar{p}\,\dot{\bar{q}}^* + 2\,\bar{q}\,\dot{\bar{p}}\,\dot{\bar{q}}^*$	$\ddot{\vec{v}}\,' = [\ddot{A}(q)]\vec{v} + 2[\dot{A}(q)]\dot{\vec{v}} + [A(q)]\ddot{\vec{v}}$ $[\ddot{A}(q)] = [A(q)][\tilde{\omega}_l][\tilde{\omega}_l] + [A(q)][\tilde{\alpha}_l]$
Da $\vec{\omega}$ a $\dot{\bar{q}}$	$\dot{\bar{q}} = \frac{1}{2}\,(0, \vec{\omega}_o)\,\bar{q}$	$\dot{\bar{q}} = \frac{1}{2}[F(q^*)_{\ominus}]^T \vec{\omega}_o$
	$\dot{\bar{q}} = \frac{1}{2}\bar{q}\,(0, \vec{\omega}_l)$	$\dot{\bar{q}} = \frac{1}{2}[F(q^*)_{\oplus}]^T \vec{\omega}_l$
Da $\dot{\bar{q}}$ a $\vec{\omega}$	$(0, \vec{\omega}_o) = 2\,\dot{\bar{q}}\,\bar{q}^*$	$\vec{\omega}_o = 2\,[F(q^*)_{\ominus}]\dot{\bar{q}}$
	$(0, \vec{\omega}_l) = 2\,\bar{q}^*\,\dot{\bar{q}}$	$\vec{\omega}_l = 2\,[F(q^*)_{\oplus}]\dot{\bar{q}}$
Da $\vec{\alpha}$ a $\ddot{\bar{q}}$	$\ddot{\bar{q}} = \frac{1}{2}\,(0, \vec{\alpha}_o)\,\bar{q} + \frac{1}{2}\,(0, \vec{\omega}_o)\,\dot{\bar{q}}$	$\ddot{\bar{q}} = \frac{1}{2}[F(\dot{q}^*)_{\ominus}]^T \vec{\omega}_o + \frac{1}{2}[F(q^*)_{\ominus}]^T \vec{\alpha}_o$
	$\ddot{\bar{q}} = \frac{1}{2}\dot{\bar{q}}\,(0, \vec{\omega}_l) + \frac{1}{2}\bar{q}\,(0, \vec{\alpha}_l)$	$\ddot{\bar{q}} = \frac{1}{2}[F(\dot{q}^*)_{\oplus}]^T \vec{\omega}_l + \frac{1}{2}[F(q^*)_{\oplus}]^T \vec{\alpha}_l$
Da $\ddot{\bar{q}}$ a $\vec{\alpha}$	$(0, \vec{\alpha}_o) = 2\,\ddot{\bar{q}}\,\bar{q}^* + 2\,\dot{\bar{q}}\,\dot{\bar{q}}^*$	$\vec{\alpha}_o = 2\,[F(q^*)_{\ominus}]\ddot{\bar{q}}$
	$(0, \vec{\alpha}_l) = 2\,\dot{\bar{q}}^*\,\dot{\bar{q}} + 2\,\bar{q}^*\,\ddot{\bar{q}}$	$\vec{\alpha}_l = 2\,[F(q^*)_{\oplus}]\ddot{\bar{q}}$

Section

Dynamics: formulations

Background

This section describes a typical direct solver

- **Can be used for classical 'smooth' MB problems…**
- **.. but it is unfit to 'large non-smooth' problems *(to this purpose, we will introduce our new iterative solver in the next section)***
- **Anyway: useful for didactical purposes, to introduce some basic concepts (quaternions, states, etc.)**

Approaches

Simple taxonomy of approaches to multibody dynamics:

- **Depending on set of state coordinates:**
  - Few reduced coordinates / recursive coordinates
  - Many 'natural' coordinates (plus Lagrangian multipliers)
- **Depending on methods to compute unknowns at each dt**
  - Direct methods
  - Iterative methods
- **Depending on handling of non-smooth dynamics**
  - Regularization (cast to smooth dynamics)
  - DVI Differential Variational Inequalities / MDI Measure Differential Inclusions
- **Depending on integration schemes**
  - Implicit, or explicit
  - Differential-Algebraic (DAE), or Ordinary Differential (ODE) with projection/stabilization
- **Etc.**

Approaches

A) "Reduced coordinates" method vs. "Lagrangian multipliers"

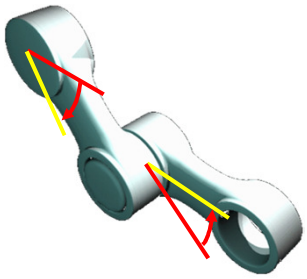
Few coordinates ('joint coordinates or 'recursive' coordinates)

Very fast simulation +

***O(n)* complexity order** +

Requires topological analysis -

Big troubles with closed chains!!! -



Approaches

B) "Natural coordinates" method vs. "reduced coordinates"

Many variables ($6 \times n_{body}$ + *constraint multipliers*)

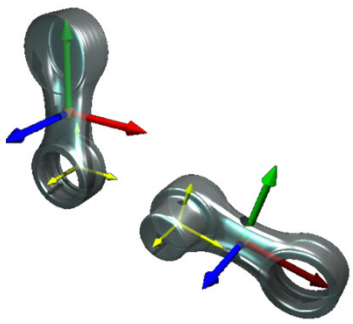
Closed chains: no problem +

Topology may change in run time +

DAE integration, or constr.stabilization -

***Trivial method: O(n³)* complexity order** -

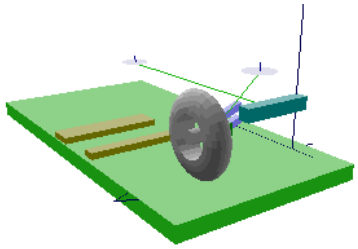
Slow simulation speed -



Approaches

Note: closed kinematic chains happens very frequently (a big trouble for reduced coordinate methods!)

Example:



Approaches

N.B: we will introduce a modern "natural coordinates" method with:

Complexity order $O(n^k)$, with $k \to 1$ +

High simulation speed +

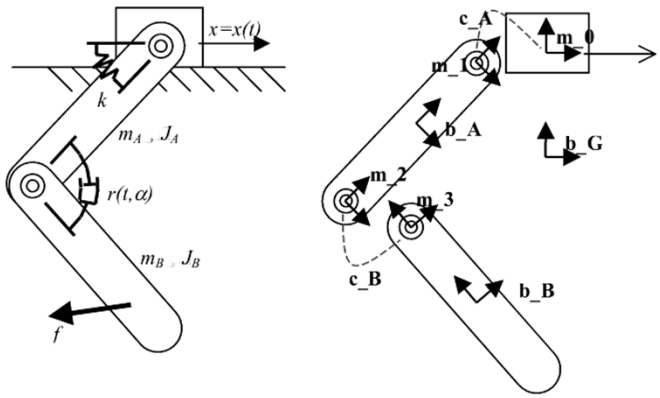
Avoid DAE drifting +

Handles redundant constraint +

Supports contacts, collisions, etc. +

Model

Example of model – using lagrangian 'natural coordinates' approach

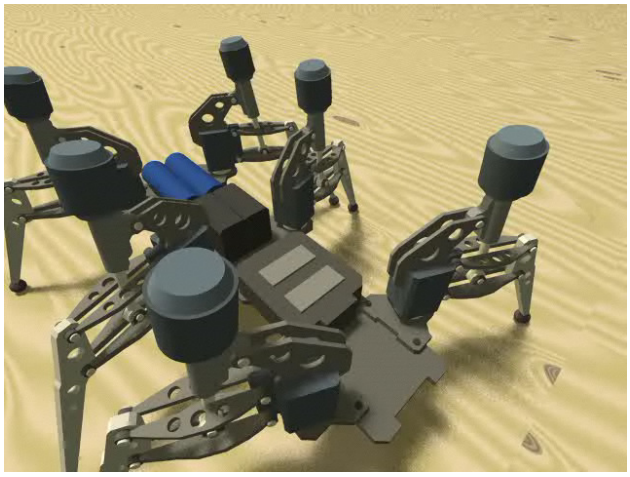


Model

Some constraint types in our Chrono::Engine software

Model

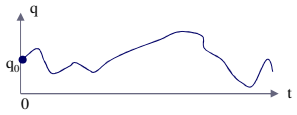
Example of model with many types of constraint (rheonomic scleronomic, etc.)



Simulation of a 6-legs robot with 42 joints, 38 rigid bodies, 12 motors, 6 contacts (A.Tasora 2007)

Formulations

We are interested in the integral of motion $\mathbf{q}(t)$ starting from boundary conditions $\mathbf{q}_0(0)$



Most often, the integrals must be approximated by *numerical integration*

How to develop the equations of motion?

Example: use

- **Lagrange formulation, or**

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_j}\right) - \frac{\partial \mathcal{L}}{\partial q_j} = \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = 0. \qquad \mathcal{L} = T - V.$$

- **Hamilton formulation**

$$\dot{p} = -\frac{\partial \mathcal{H}}{\partial q}$$
$$\dot{q} = \frac{\partial \mathcal{H}}{\partial p}.$$

$$\mathcal{H} = T + V$$
$$\mathcal{H} = \sum_i p_i \dot{q}_i - \mathcal{L}$$

Formulations

Introduce vector of independent *generalized coordinates* q for translation / rotation / etc.

- **Lagrange formulation:**

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_j}\right) - \frac{\partial \mathcal{L}}{\partial q_j} = \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = Q \qquad \mathcal{L} = T - V.$$

- **Hamilton formulation:**

$$\dot{p} = -\frac{\partial \mathcal{H}}{\partial q}$$
$$\dot{q} = \frac{\partial \mathcal{H}}{\partial p}.$$

$$\mathcal{H} = T + V$$
$$\mathcal{H} = \sum_i p_i \dot{q}_i - \mathcal{L}$$

- **Other various variational principles:**
  - Gauss least constraint principle
  - Jourdain principle
  - D'Alambert principle
  - Euler-Lagrange equations
  - etc.

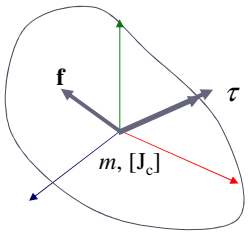
Formulations

For the special case of rigid bodies only, a classical mechanics formulation is:

- **Newton-Euler equations:**

$$\begin{pmatrix} m\mathbf{I} & 0 \\ 0 & \mathbf{J}_c \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix} + \begin{pmatrix} 0 \\ \boldsymbol{\omega} \times \mathbf{J}_c \boldsymbol{\omega} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{pmatrix}$$

Masses and inertia tensors — *Accel.* — *Gyroscopic term* — *Applied forces* — *Applied torques*



- These can be obtained by developing, for instance, the Lagrange equations.
- Note that the unknowns are the linear accelerations and the angular accelerations: $\begin{pmatrix} \ddot{\mathbf{q}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix}$
- The gyroscopic term is null if ω is parallel to one of the three principal axes of [J] tensor (ie. ω aligned to one of the eigenvectors of [J])
- External forces applied to center of mass, to get this simple formulation

Formulation with constraints

Lagrangian formulation, with constraints

$$\begin{cases} \left\{ \frac{d}{dt}\left[\frac{\partial E_c}{\partial \dot{\mathbf{x}}}\right] \right\}^T - \left[\frac{\partial E_c}{\partial \mathbf{x}}\right]^T + [C_x]^T \boldsymbol{\lambda} = \hat{\mathbf{Q}} \\ \mathbf{C}(\mathbf{x}, t) = \mathbf{0} \end{cases}$$

This is a *Differential-Algebraic-Equation* problem (DAE)
Without constraint equations, it would be an *Ordinary-Differential-Problem* (ODE)

$\mathbf{C}(\mathbf{x},t)$ is a vector of (nonlinear) equations, satisfied =0 if constraint is 'closed'
$\boldsymbol{\lambda}$ is the vector of constraint reaction (reaction forces/torques)

Lagrangian dynamics

Dinamics of a constrained system (only bilateral constraints, no unilaterals!):

$$\begin{cases} \left\{ \frac{d}{dt}\left[\frac{\partial E_c}{\partial \dot{\mathbf{x}}}\right] \right\}^T - \left[\frac{\partial E_c}{\partial \mathbf{x}}\right]^T + [C_x]^T \boldsymbol{\lambda} = \hat{\mathbf{Q}} \\ \mathbf{C}(\mathbf{x}, t) = \mathbf{0} \end{cases}$$

Trick: from a DAE....
(Differential Algebraic Equations)

$$\mathbf{C} = \mathbf{C}(\mathbf{x}, t) = \mathbf{0}$$
$$\dot{\mathbf{C}} = [C_x]\dot{\mathbf{x}} + \mathbf{C}_t = \mathbf{0}$$
$$\ddot{\mathbf{C}} = [C_x]\ddot{\mathbf{x}} - \mathbf{Q}_c = \mathbf{0}$$

$$\begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \cdot \begin{Bmatrix} \ddot{\mathbf{x}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_m \\ \mathbf{Q}_c \end{Bmatrix}$$

...to a simplier ODE
(Ordinary Differential Equations)

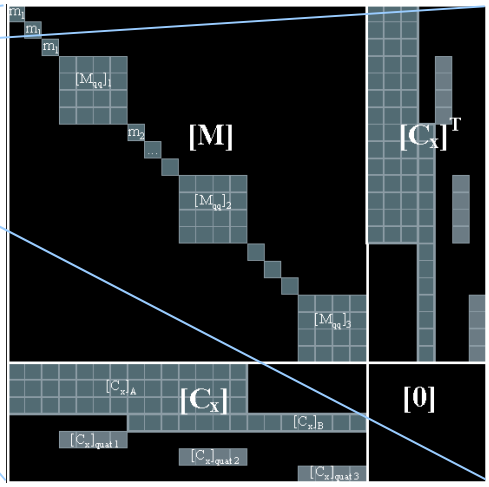
Solving for unknowns

Structure of the linear systems:

$$\begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \cdot \begin{Bmatrix} \ddot{\mathbf{x}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_m \\ \mathbf{Q}_c \end{Bmatrix}$$

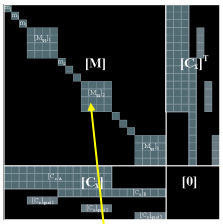
Highly sparse (good!)

Symmetric structure (good!)



Solving for unknowns

How to factorize and solve efficiently this system?



Gauss method?
- No decomposition
- Pivoting destroys symmetry!!!

LU decomposition?
- Pivoting destroys symmetry!!!

LL^T Cholesky decomposition?
- Exploits symmetry BUT…
- the matrix is not positive definite!!!

LDL^T decomposition?
$[A]=[L][D][L]^T$
- Exploits symmetry
- works also with nonpositive definite metrices,
- … troubles with [M] submatrices 4x4 rank-3 !!!

Solving for unknowns

Fix: project in α - ω space to get rid of rank-deficient [M] submatrices:

Temporary change of coordinates:

$$\begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \cdot \begin{Bmatrix} \ddot{\mathbf{x}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_m \\ \mathbf{Q}_c \end{Bmatrix}$$

Note...remember:

$$\frac{1}{4}[G_1(\mathbf{q})]^T [G_1(\mathbf{q})] = [I]$$

$$\boldsymbol{\alpha}_1 = [G_1(\mathbf{q}_{1,w})]\, \ddot{\mathbf{q}}$$

$$\ddot{\mathbf{x}}_\alpha = \{\ddot{\mathbf{p}}_{(1)}, \boldsymbol{\alpha}_{(1)}, \ldots, \ddot{\mathbf{p}}_{(n)}, \boldsymbol{\alpha}_{(n)}\}$$

$$\begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \begin{bmatrix} [T_q] & \\ & [I] \end{bmatrix} \cdot \begin{Bmatrix} \ddot{\mathbf{x}}_\alpha \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_m \\ \mathbf{Q}_c \end{Bmatrix}$$

$$[T_q] = \begin{bmatrix} [I] & & \\ & \frac{1}{4}[G_1]_{(1)}^T & \\ & & \ddots \\ & & & [I] \\ & & & & \frac{1}{4}[G_1]_{(n)}^T \end{bmatrix}$$

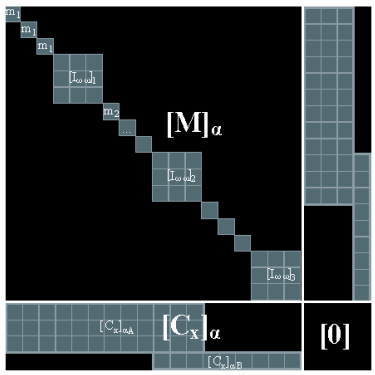
Solving for unknowns

To keep symmetry, pre-multiply everything by $[T_q]^T$:

$$\begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \begin{bmatrix} [T_q] & \\ & [I] \end{bmatrix} \cdot \begin{Bmatrix} \ddot{\mathbf{x}}_\alpha \\ \boldsymbol{\lambda} \end{Bmatrix} =$$

$$\begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_m \\ \mathbf{Q}_c \end{Bmatrix}$$

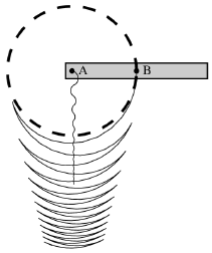
More compact
Still symmetric
Still sparse
Well conditioned diag.pivoting
Inertia tensor as in Newton-Euler
Quaternions (not angles!) for $[C_x]$
..efficient LDL^T decomposition !!!



Stabilization schemes

From step to step, errors might accumulate in *positions* or *speeds* of constraints
(we transformed the DAE in ODE, so we satisfy constraints only in accelerations)

Example of constraint that accumulate violation in position:



Stabilization schemes

Different approaches to solve the "constraint drifting":

- **Solve DAE directly with a special method** (ex. DASSL integrator)
  - Numerically intensive
  - May suffer ill-conditioning, exp. for small timesteps
  - Requires *precise* initial consistent state!
  - Other: RADAU, GEAR, etc.
- **Use stabilization methods**
  - Example: Baumgarte stabilization
  - Example: regularization & penalty functions
  - Fast, but difficult to adjust, not very precise, may cause divergence.
- **Use projection methods**
  - Example, see W.Blajer method
  - Projections are like repeated 'corrections' of positions and speeds
  - Project onto speed manifold each timestep – linear problem
  - Project onto position manifold each timestep – nonlinear problem (iterate 1-3 times)
  - Note that the position projection is like an 'assembly' operation.

Complete solution scheme, with custom constraint stabilization

Finds accelerations, to integrate speeds and positions

(re-map in quaternions q'')

$$\begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{bmatrix}[M] & [C_x]^T \\ [C_x] & [0]\end{bmatrix}\begin{bmatrix}[T_q] & \\ & [I]\end{bmatrix}\cdot\begin{Bmatrix}\ddot{\mathbf{x}}_\alpha \\ \boldsymbol{\lambda}\end{Bmatrix}=$$

$$\begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{Bmatrix}\hat{\mathbf{Q}}+\mathbf{Q}_m \\ \mathbf{Q}_c\end{Bmatrix}$$

$$\ddot{\mathbf{q}}_{1,w\,(i)} = \tfrac{1}{4}\left[G_1(\mathbf{q}_{1,w\,(i)})\right]^T$$

These matrices are the same!
(if C_x does not change a lot, a single symbolic factorization can suffice...)

Correct constraint position-violation

(re-map in quaternions q)

$$\begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{bmatrix}[M] & [C_x]^T \\ [C_x] & [0]\end{bmatrix}$$

$$\begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{Bmatrix}0 \\ -\mathbf{C}^{(j)}\end{Bmatrix}$$

$$\Delta\mathbf{q}_{1,w\,(i)} = \tfrac{1}{4}\left[G_1(\mathbf{q}_{1,w\,(i)})\right]^T\ \Delta\boldsymbol{\beta}_{1\,(i)}$$

Correct constraint speed-violation

(re-map in quaternions q')

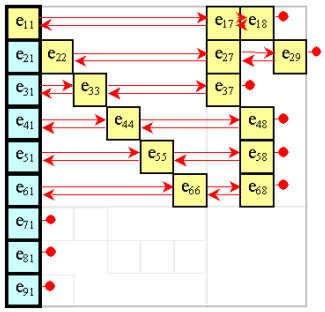
$$\begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{bmatrix}[M] & [C_x]^T \\ [C_x] & [0]\end{bmatrix}\begin{bmatrix}[T_q] & \\ & [I]\end{bmatrix}\cdot\begin{Bmatrix}\Delta\dot{\mathbf{x}}_\omega \\ \boldsymbol{\mu}\end{Bmatrix}=$$

$$\begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{Bmatrix}0 \\ -\dot{\mathbf{C}}_{old}\end{Bmatrix}$$

$$\Delta\dot{\mathbf{q}}_{1,w\,(i)} = \tfrac{1}{4}\left[G_1(\mathbf{q}_{1,w\,(i)})\right]^T\ \Delta\boldsymbol{\omega}_{1\,(i)}$$

Sparse LDLt factorization

Linked-list half-storage:

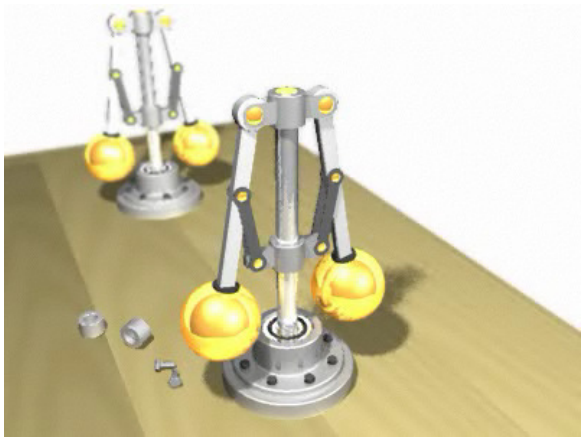


Custom list-based LDL^T algorithm
Can withstand redundant constraints
Linear-time decomposition for acyclic systems!!!

hence, O(n) complexity for mechanisms without closed chains!

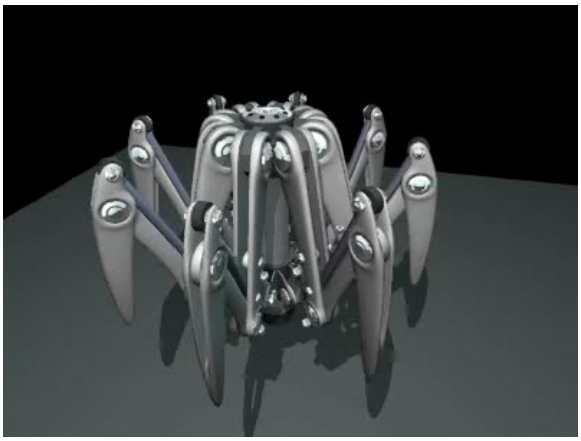
Examples

Test: simulation of a Watt mechanism, *with ray-traced rendering in Realsoft3D*



Examples

Benchmark to test the efficiency of our sparse solver

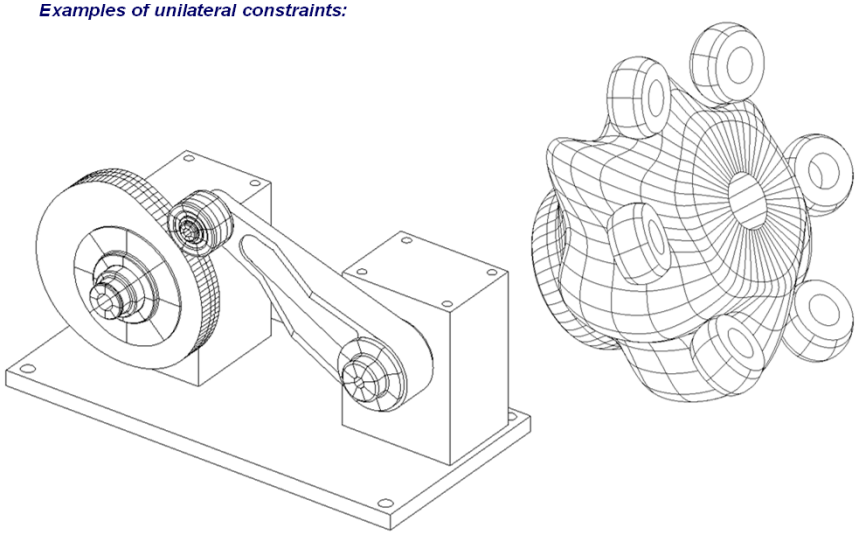


Section

Iterative method for nonsmooth-dynamics

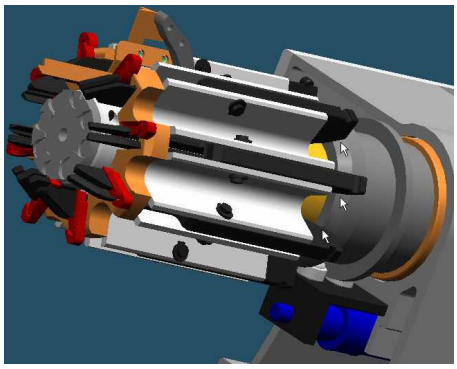
Why non-smooth dynamics?

Examples of unilateral constraints:



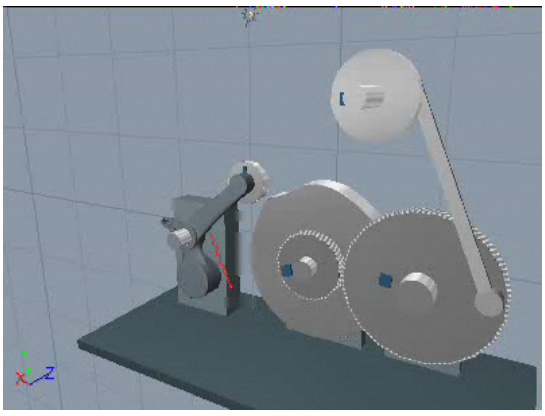
Introduction to non-smooth dynamics

- **Unilateral constraints and friction: happen in many mechanisms**
- **Inequalities → lead to a DVI (*Differential Variational Inclusion problem*)**



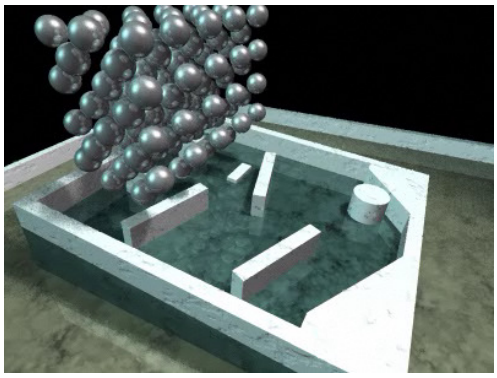
Introduction to non-smooth dynamics

- **Current solvers: slow, low efficiency, not robust**
- **Need of solvers for systems with thousands or million of constraints**

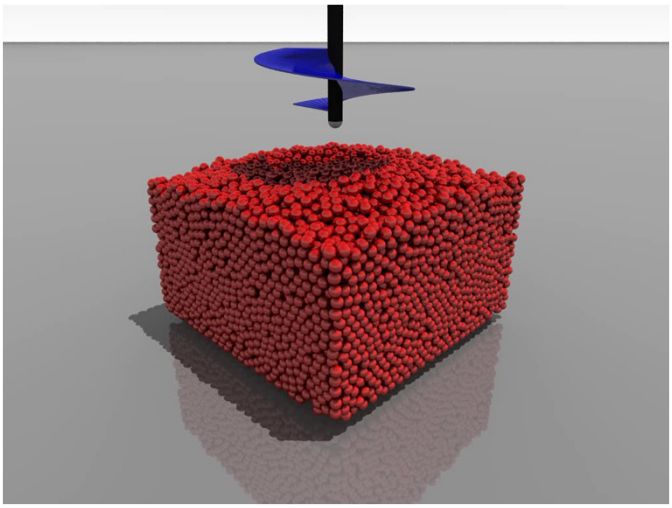


Introduction to non-smooth dynamics

- → Develop new ITERATIVE methods, aimed at multibody problems with an high number of constraints and parts
- .. if possible, in real time..
- .. of P class, O(*n*) ..
- .. and parallel !



Introduction to non-smooth dynamics



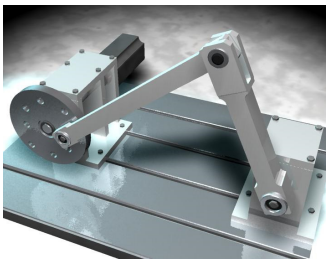
Introduction to non-smooth dynamics

Multi rigid body systems, n g.d.l

Only bilateral constraints:

DAE / ODE:
Solve for unknown accelerations at each time step using **linear systems**

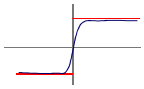
- **$O(n^3)$ computational complexity** [Gauss]
- **$O(n)$ computational complexity** with recent methods [A.Tasora, D.Baraff]



Introduction to non-smooth dynamics

.. Adding also non-smooth constraints (es.friction):

ODE or DAE + regularization methods
(trick: approximate with stiff force fields) — NO

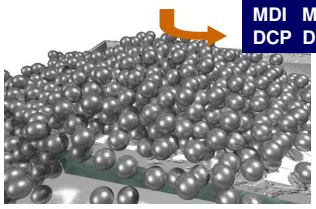


- ***Stiffness: too small time steps!***

ODE or DAE + "stop-and-restart" methods — NO

- ***Impracticable for complex systems!***

DVI Differential Variational Inequalitied
MDI Measure Differential Inclusions
DCP Differential-Complementarity Problems — OK



- ***Handle large steps, multiple discontinuities***
- ***Embed complementarity problem, each step***
- **How to solve DVI problems?**

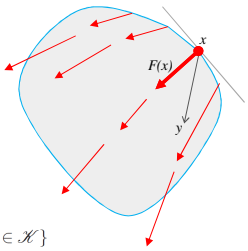
Variational Inequalities

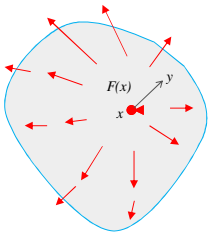
Definition of Variational Inequality (VI):

$$\boldsymbol{x} \in \mathbb{K} \quad : \quad \langle F(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \mathbb{K}$$

- for continuous $F(\boldsymbol{x}) : \mathbb{K} \to \mathbb{R}^n$
- with closed and convex $\mathbb{K}$

(see Kinderleher and Stampacchia ,1980)





Alternative formulation:

$$\mathbf{x} \in \mathscr{K}, \ \mathbf{g}(\mathbf{x}) \in -\mathsf{N}_{\mathscr{K}}(\mathbf{x})$$

$$\mathsf{N}_{\mathscr{K}}(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : \langle \mathbf{y}, \mathbf{x} - \mathbf{z} \rangle \geq 0, \forall \mathbf{z} \in \mathscr{K}\}$$

Differential Variational Inequality

Differential Variational Inequality (DVI)

$$\frac{d\boldsymbol{x}}{dt} = f(t, \boldsymbol{x}, \boldsymbol{u})$$
$$\boldsymbol{u} \in \mathsf{SOL}\left(\mathbb{K}, F(t, \boldsymbol{x}(t), \cdot)\right)$$
$$\Xi(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0$$

where $\mathsf{SOL}\left(\mathbb{K}, F(t, \boldsymbol{x}(t), \cdot)\right)$ is the set of solutions to the VI $\left(\mathbb{K}, F(t, \boldsymbol{x}(t), \cdot)\right)$

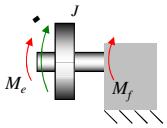
Differential Inclusions: motivation

Most differential problems can be posed as equalities like:

$dx/dt = f(x,t)$ → **ODE, DAE , ok**

But some problems require inequalities or inclusions like

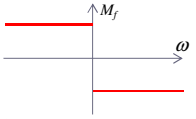
$dx/dt \in f(x,t)$ → **Differential Inclusion! (DI)**



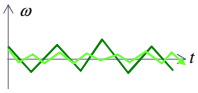
***Example**: a flywheel with brake torque and applied torque (looks simple?!)*

$J\, d\omega/dt = M_f(\omega) + M_e(t)$ where $M_f = -M_{f\,max}$ if $\omega > 0$

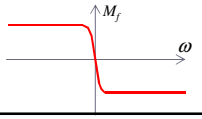
and $M_f = M_{f\,max}$ if $\omega < 0$



- All ODE integrator would never stop in $\omega = 0$!
  It would just ripple about $\omega = 0$ ..



- Reducing Δt in ODE integrator may reduce the ripple,
  But what if low J ? Divergence!
- Regularization methods? A) Numerical stiffness!
  B) Approximation! C) The brake would never stick! …



- Also, if ever $\omega = 0$, which M_f ? Not computable!

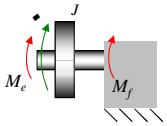
Differential Inclusions: motivation

Most differential problems can be posed as equalities like:

$dx/dt = f(x,t)$ → **ODE, DAE , ok**

But some problems require inequalities or inclusions like

$dx/dt \in f(x,t)$ → **Differential inclusion! (DI)**



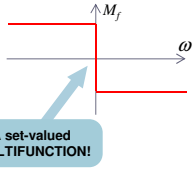
***Example**: a flywheel with brake torque and applied torque (simple?!)*

Improved model!

$J\, d\omega/dt = M_f(\omega) + M_e(t)$ where $M_f = -M_{f\,max}$ for $\omega > 0$

and $M_f = M_{f\,max}$ for $\omega < 0$

and $-M_{f\,max} < M_f < M_{f\,max}$ for $\omega = 0$



- This could handle also $\omega = 0$ case, ex. brake sticking
- But now we have a differential inclusion $d\omega/dt \in f(\omega,t)$. **How to solve it**?

Differential Inclusions

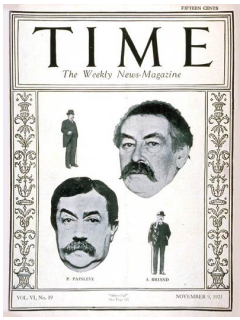
Differential inclusions: not an ODE anymore…
We need some new concepts from convex algebra …

- **Filippov Differential Inclusions (DI) continuous $x(t)$ solution of ODE discontinuous RHS $f(\cdot)$ if**

$$\frac{dx}{dt} \in \mathcal{F}f(t,x), \qquad \mathcal{F}f(t,x) = \bigcap_{\eta>0} \bigcap_{N:\lambda_0(N)=0} \overline{\text{co}}\, f(t,(x+\eta B_1)\backslash N)$$

  - $\mathcal{F}f(t,x)$ is a vector-valued multifunction
  - $\boldsymbol{\lambda}_0(N)$ is the Lebesgue measure on set N
  - $f(\cdot)$ upper semi-continuous
  - $f(\cdot)$ measurable in t
  - $f(\cdot)$ closed, bounded, convex

  } *mild conditions…*



Difficult set-valued problem

Can solve Coulomb friction, but fails the Painlevé paradox (1895)..

Measure Differential Inclusions

- **What if the velocity must have discontinuities?** $dv/dt \in f(v,t)$
  - ..because of impulses,
  - ..because of impacts,
  - ..because of friction effects such as in Painlevé paradox

- **The RHS has 'peaks' (impulses),**
  The velocity has 'jumps' →*distribution, generalized function*
  - Which is the meaning of the derivative of discontinuous velocity?
  - Integrals in the Lebesgue-Stieltjes sense – not Riemann
  - RHS not Lipschitz continuous: Picard-Lindelöf theorem does not apply.. What about existence and uniqueness of solution?

- **Measure Differential Inclusion (MDI): *strong* solution** [Moreau]

$$\mathrm{d}\mu/\mathrm{d}v(t) \in K(t)$$

For singular decomposition of Borel measure $\mu = m\,v + \mu_s$

$$m(t) \in K(t) \qquad \frac{\mathrm{d}\mu_s}{\mathrm{d}|\mu_s|}(t) \in K(t)_\infty$$

MDI as DVI

- **For MB dynamics: *weak* solution to MDI / DVI** [D.E.Stewart]

$$\mathrm{d}\mu/\mathrm{d}v(t) \in K(t) \quad \text{if} \quad \frac{\int \phi(t)\,\mu(\mathrm{d}t)}{\int \phi(t)\,v(\mathrm{d}t)} \in \overline{\mathrm{co}} \bigcup_{\tau:\phi(\tau)\neq 0} K(\tau)$$

with Borel measure μ (=Dv) and continuous Φ with compact support

- $K(\cdot)$ must be closed, convex, pointed $K(t)_\infty \cap (-K(t)_\infty) = \{0\}$ for all t
- discontinuous μ has bounded variation $\bigvee_0^T \mu$

Encompasses Dirac functions and vector distributions → impulsive forces OK.

Solves Painlevé paradoxes

Solution in terms of speed variations & impulses – not accelerations & forces

The DVI model

Mechanical system with

- Set $\mathcal{G}_\mathcal{B}$ of bilateral joints
- Set $\mathcal{G}_\mathcal{A}$ of point contacts
- External forces

$$\dot{\boldsymbol{q}} = \Gamma(\boldsymbol{q})\boldsymbol{v}$$

$$M(\boldsymbol{q})\frac{d\boldsymbol{v}}{dt} = \sum_{i\in\mathcal{G}_\mathcal{B}} \hat{\gamma}_\mathcal{B}^i \nabla\Psi^i + \sum_{i\in\mathcal{G}_\mathcal{A}} \hat{\gamma}_\mathcal{A}^i D^i + f_t(t,q,v)$$

$$\Psi^i(q,t) \subset \emptyset, \quad i \in \mathcal{G}_\mathcal{B}$$

$$\hat{\gamma}_\mathcal{A}^i \in \mathrm{SOL}\left(\Upsilon^i, F(t,q(t),v(t),\cdot)\right), \quad i \in \mathcal{G}_\mathcal{A}.$$

Bilateral constraint equations

Contact forces VIs

The DVI model

We get a differential problem with equilibrium constraints (DPEC), → a Differential Complementarity Problem (DCP), also a Differential Variational Inequality (DVI)

$$
\begin{aligned}
M(q^l)\frac{d\boldsymbol{v}}{dt} &= \sum_{i\in\mathcal{A}(\boldsymbol{q},\epsilon)} \left(\widehat{\gamma}_n^i \boldsymbol{D}_n^i + \widehat{\gamma}_u^i \boldsymbol{D}_u^i + \widehat{\gamma}_v^i \boldsymbol{D}_v^i\right) + \\
&\quad + \sum_{i\in\mathcal{G}_\mathcal{B}} \widehat{\gamma}_\mathcal{B}^i \nabla\Psi^i + \boldsymbol{f}_t(t,\boldsymbol{q},\boldsymbol{v}) \\
\dot{\boldsymbol{q}} &= \Gamma(\boldsymbol{q},\boldsymbol{v}) \\
\Psi^i(\boldsymbol{q},t) &= 0 \quad i\in\mathcal{G}_\mathcal{B} \\
\widehat{\gamma}_n^i &\geq 0 \quad \perp \quad \Phi^i(\boldsymbol{q},\Omega) \geq 0, \quad i\in\mathcal{G}_\mathcal{P} \\
\left(\widehat{\gamma}_u^i, \widehat{\gamma}_v^i\right) &= \operatorname{argmin}_{\mu^i\widehat{\gamma}_n^i \geq \sqrt{(\widehat{\gamma}_u^i)^2+(\widehat{\gamma}_v^i)^2}} \quad i\in\mathcal{G}_\mathcal{P} \\
&\quad \boldsymbol{v}^T\left(\widehat{\gamma}_u \boldsymbol{D}_u^i + \widehat{\gamma}_v \boldsymbol{D}_v^i\right).
\end{aligned}
$$

Bilateral constraint equations

Contact constraint equations

The tricky part: the Coulomb friction

The DVI model

After time-step discretization, as a measure differential inclusion, it translates into a Nonlinear Complementarity Problem (NCP), also a Variational Inequality (VI):

$$
\begin{aligned}
M(\boldsymbol{v}^{(l+1)} - \boldsymbol{v}^l) &= \sum_{i\in\mathcal{A}(q^{(l)},\epsilon)} \left(\gamma_n^i \boldsymbol{D}_n^i + \gamma_u^i \boldsymbol{D}_u^i + \gamma_v^i \boldsymbol{D}_v^i\right) + \\
&\quad + \sum_{i\in\mathcal{G}_\mathcal{B}} \left(\gamma_h^i \nabla\Psi^i\right) + h\boldsymbol{f}_t(t^{(l)},\boldsymbol{q}^{(l)},\boldsymbol{v}^{(l)}) \\
0 &= \frac{1}{h}\Psi^i(\boldsymbol{q}^{(l)}) + \nabla\Psi^{i^T}\boldsymbol{v}^{(l+1)} + \frac{\partial\Psi^i}{\partial t}, \quad i\in\mathcal{G}_\mathcal{B} \\
0 &\leq \frac{1}{h}\Phi^i(\boldsymbol{q}^{(l)}) + \nabla\Phi^{i^T}\boldsymbol{v}^{(l+1)} \\
&\perp \quad \gamma_n^i \geq 0, \; i\in\mathcal{A}(q^{(l)},\epsilon) \\
\left(\gamma_u^i, \gamma_v^i\right) &= \operatorname{argmin}_{\mu^i\gamma_n^i \geq \sqrt{(\gamma_u^i)^2+(\gamma_v^i)^2}} \quad i\in\mathcal{A}(q^{(l)},\epsilon) \\
&\quad \left[\boldsymbol{v}^T(\gamma_u \boldsymbol{D}_u^i + \gamma_v \boldsymbol{D}_v^i)\right] \\
\boldsymbol{q}^{(l+1)} &= \boldsymbol{q}^{(l)} + h\boldsymbol{v}^{(l+1)},
\end{aligned}
$$

Speeds

Reaction impulses

Forces

Stabilization terms

Bilateral constraint equations

Contact constraint equations

COMPLEMENTARITY!

Coulomb 3D friction model

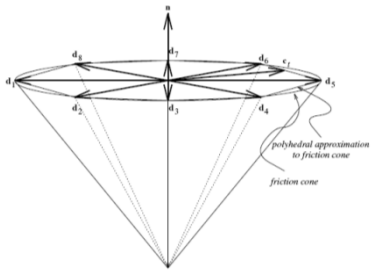
VI as complementarity

The NCP is a generalization of Linear Complementarity Problems (LCP)

Why not using off-the-shelf LCP solvers?

- **LCP require finitely-generated approximations of NL convex sets, see the friction cones:**

  → We want NO polyhedral approximations



Example (D.Stewart): Polyhedral approximation of friction cones, to feed i

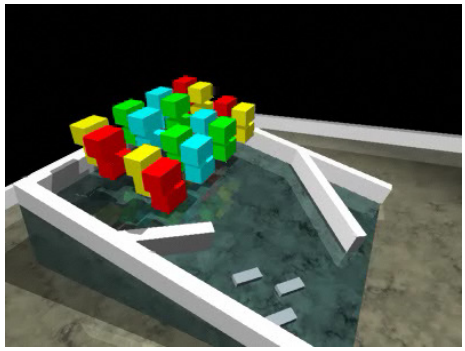
Need for a custom iterative **non-linear complementarity solver**

VI as complementarity

Why not using off-the-shelf LCP solvers (for Linear Complementarity)?

- **Most LCP solvers (Lemke, Dantzig) are based on exact simplex methods, with NP-hard complexity class. Risk of combinatorial explosion!**
- **Simplex methods do not support threshold 'premature' termination**

→ We prefer an approximate O(n) iterative method



Example: what happens when stopping too early the simplex method

Cone complementarity

A modification (relaxation, to get a convex problem):

$$
\begin{aligned}
M(\boldsymbol{v}^{(l+1)} - \boldsymbol{v}^{l}) = & \sum_{i \in \mathcal{A}(q^{(l)}, \epsilon)} \left(\gamma_n^i \boldsymbol{D}_n^i + \gamma_u^i \boldsymbol{D}_u^i + \gamma_v^i \boldsymbol{D}_v^i\right) + \\
& + \sum_{i \in \mathcal{G}_\mathcal{B}} \left(\gamma_b^i \nabla \Psi^i\right) + h \boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)}) \\
0 = & \frac{1}{h}\Psi^i(\boldsymbol{q}^{(l)}) + \nabla \Psi^{i^T} \boldsymbol{v}^{(l+1)} + \frac{\partial \Psi^i}{\partial t}, \quad i \in \mathcal{G}_\mathcal{B} \\
0 \leq & \frac{1}{h}\Phi^i(\boldsymbol{q}^{(l)}) + \nabla \Phi^{i^T} \boldsymbol{v}^{(l+1)} \quad \boxed{-\mu^i \sqrt{(\boldsymbol{D}_u^{i,T}\boldsymbol{v})^2 + (\boldsymbol{D}_v^{i,T}\boldsymbol{v})^2}} \\
& \perp \quad \gamma_n^i \geq 0, \; i \in \mathcal{A}(q^{(l)}, \epsilon) \\
(\gamma_u^i, \gamma_v^i) = & \operatorname{argmin}_{\mu^i \gamma_n^i \geq \sqrt{(\gamma_u^i)^2 + (\gamma_v^i)^2}} \quad i \in \mathcal{A}(q^{(l)}, \epsilon) \\
& \left[\boldsymbol{v}^T (\gamma_u \boldsymbol{D}_u^i + \gamma_v \boldsymbol{D}_v^i)\right] \\
\boldsymbol{q}^{(l+1)} = & \boldsymbol{q}^{(l)} + h \boldsymbol{v}^{(l+1)},
\end{aligned}
$$

For small h and/or small speeds, almost no differences from the Coulomb theory.

Also, convergence proved as in the original scheme.

[see M.Anitescu, "Optimization Based Simulation of Nonsmooth Rigid Body Dynamics"]

Cone complementarity

Aiming at a more compact formulation:

$$
\begin{aligned}
\boldsymbol{b}_\mathcal{A} &= \left\{\frac{1}{h}\Phi^{i_1}, 0, 0, \frac{1}{h}\Phi^{i_2}, 0, 0, \ldots, \frac{1}{h}\Phi^{i_{n_\mathcal{A}}}, 0, 0\right\} \\
\boldsymbol{\gamma}_\mathcal{A} &= \left\{\gamma_n^{i_1}, \gamma_u^{i_1}, \gamma_v^{i_1}, \gamma_n^{i_2}, \gamma_u^{i_2}, \gamma_v^{i_2}, \ldots, \gamma_n^{i_{n_\mathcal{A}}}, \gamma_u^{i_{n_\mathcal{A}}}, \gamma_v^{i_{n_\mathcal{A}}}\right\} \\
\boldsymbol{b}_\mathcal{B} &= \left\{\frac{1}{h}\Psi^1 + \frac{\partial \Psi^1}{\partial t}, \frac{1}{h}\Psi^2 + \frac{\partial \Psi^2}{\partial t}, \ldots, \frac{1}{h}\Psi^{n_\mathcal{B}} + \frac{\partial \Psi^{n_\mathcal{B}}}{\partial t}\right\} \\
\boldsymbol{\gamma}_\mathcal{B} &= \left\{\gamma_b^1, \gamma_b^2, \ldots, \gamma_b^{n_\mathcal{B}}\right\} \\
D_\mathcal{A} &= \left[D^{i_1} | D^{i_2} | \ldots | D^{i_{n_\mathcal{A}}}\right], \quad i \in \mathcal{A}(\boldsymbol{q}^l, \epsilon) \qquad D^i = \left[\boldsymbol{D}_n^i | \boldsymbol{D}_u^i | \boldsymbol{D}_v^i\right] \\
D_\mathcal{B} &= \left[\nabla \Psi^{i_1} | \nabla \Psi^{i_2} | \ldots | \nabla \Psi^{i_{n_\mathcal{B}}}\right], \quad i \in \mathcal{G}_\mathcal{B}
\end{aligned}
$$

$$
\boxed{
\begin{aligned}
\boldsymbol{b}_\mathcal{E} &\in \mathbb{R}^{n_\mathcal{E}} = \{\boldsymbol{b}_\mathcal{A}, \boldsymbol{b}_\mathcal{B}\} \\
\boldsymbol{\gamma}_\mathcal{E} &\in \mathbb{R}^{n_\mathcal{E}} = \{\boldsymbol{\gamma}_\mathcal{A}, \boldsymbol{\gamma}_\mathcal{B}\} \\
D_\mathcal{E} &= [D_\mathcal{A} | D_\mathcal{B}]
\end{aligned}
}
$$

Cone complementarity

To get the convex Cone Complementarity Problem (CCP), also define:

$$\tilde{\boldsymbol{k}}^{(l)} = M\boldsymbol{v}^{(l)} + h\boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)})$$
$$N = D_{\mathcal{E}}^T M^{-1} D_{\mathcal{E}}$$
$$\boldsymbol{r} = D_{\mathcal{E}}^T M^{-1} \tilde{\boldsymbol{k}} + \boldsymbol{b}_{\mathcal{E}}$$

Then:

$$M(\boldsymbol{v}^{(l+1)} - \boldsymbol{v}^l) = \sum_{i \in \mathcal{A}(q^{(l)}, \epsilon)} (\gamma_n^i \boldsymbol{D}_n^i + \gamma_u^i \boldsymbol{D}_u^i + \gamma_v^i \boldsymbol{D}_v^i) + \sum_{i \in \mathcal{G}_{\mathcal{B}}} (\gamma_k^i \nabla \Psi^i) + h\boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)})$$
$$0 = \frac{1}{h}\Psi^i(\boldsymbol{q}^{(l)}) + \nabla\Psi^{iT}\boldsymbol{v}^{(l+1)} + \frac{\partial \Psi^i}{\partial t}, \quad i \in \mathcal{G}_{\mathcal{B}}$$
$$0 \le \frac{1}{h}\Phi^i(\boldsymbol{q}^{(l)}) + \nabla\Phi^{iT}\boldsymbol{v}^{(l+1)} \perp \gamma_n^i \ge 0, \; i \in \mathcal{A}(q^{(l)}, \epsilon)$$
$$(\gamma_u^i, \gamma_v^i) = \operatorname{argmin}_{\mu\gamma_n^i \ge \sqrt{(\gamma_u^i)^2 + (\gamma_v^i)^2}} \; i \in \mathcal{A}(q^{(l)}, \epsilon)$$
$$[\boldsymbol{v}^T(\gamma_u \boldsymbol{D}_u^i + \gamma_v \boldsymbol{D}_v^i)]$$

becomes..

$$(N\boldsymbol{\gamma}_{\mathcal{E}} + \boldsymbol{r}) \in -\Upsilon^{\circ} \quad \perp \quad \boldsymbol{\gamma}_{\mathcal{E}} \in \Upsilon$$

This is a CCP, **CONE COMPLEMENTARITY PROBLEM**

Cone complementarity

Here we introduced the convex cone

$$\Upsilon = \left(\bigoplus_{i \in \mathcal{A}(\boldsymbol{q}^l, \epsilon)} \mathcal{FC}^i \right) \oplus \left(\bigoplus_{i \in \mathcal{G}_{\mathcal{B}}} \mathcal{BC}^i \right)$$

$\mathcal{FC}^i$ is i-th friction cone

$\mathcal{BC}^i$ is R

..and its polar cone:

$$\Upsilon^{\circ} = \left(\bigoplus_{i \in \mathcal{A}(\boldsymbol{q}^l, \epsilon)} \mathcal{FC}^{i\circ} \right) \oplus \left(\bigoplus_{i \in \mathcal{G}_{\mathcal{B}}} \mathcal{BC}^{i\circ} \right)$$

CCP: $(N\boldsymbol{\gamma}_{\mathcal{E}} + \boldsymbol{r}) \in -\Upsilon^{\circ} \quad \perp \quad \boldsymbol{\gamma}_{\mathcal{E}} \in \Upsilon$

The iterative method

How to practically solve the Cone Complementarity Problem ?

$$(N\boldsymbol{\gamma}_{\mathcal{E}} + \boldsymbol{r}) \in -\Upsilon^{\circ} \quad \perp \quad \boldsymbol{\gamma}_{\mathcal{E}} \in \Upsilon$$

Our method: use a fixed-point iteration

$$\boldsymbol{\gamma}^{r+1} = \lambda \Pi_{\Upsilon} \left(\boldsymbol{\gamma}^{r} - \omega B^{r} \left(N\boldsymbol{\gamma}^{r} + \boldsymbol{r} + K^{r} \left(\boldsymbol{\gamma}^{r+1} - \boldsymbol{\gamma}^{r} \right) \right) \right) + (1-\lambda)\boldsymbol{\gamma}^{r}$$

with:

- **matrices:**

$$B^{r} = \begin{bmatrix} \eta_1 I_{n_1} & 0 & \cdots & 0 \\ 0 & \eta_2 I_{n_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \eta_{n_k} I_{n_{n_k}} \end{bmatrix} \qquad K^{T} = \begin{bmatrix} 0 & K_{12} & K_{13} & \cdots & K_{1n_k} \\ 0 & 0 & K_{23} & \cdots & K_{2n_k} \\ 0 & 0 & 0 & \cdots & K_{3n_k} \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & & 0 \end{bmatrix}$$

- **..and a non-extensive orthogonal projection operator onto feasible set**

$$\Pi_{\Upsilon} : \mathbb{R}^{n_{\mathcal{E}}} \rightarrow \mathbb{R}^{n_{\mathcal{E}}}$$

The iterative method

ASSUMPTIONS

A1 The matrix N of the problem (CCP) is symmetric and positive semi-definite.

A2 There exists a positive number, $\alpha > 0$ such that, at any iteration r, $r = 0, 1, 2, \ldots$, we have that $B^{r} \succ \alpha I$

A3 There exists a positive number, $\beta > 0$ such that, at any iteration r, $r = 0, 1, 2, \ldots$, we have that $(x^{r+1} - x^{r})^{T} \left((\lambda\omega B^{r})^{-1} + K^{r} - \frac{N}{2} \right) (x^{r+1} - x^{r}) \geq \beta \left\| x^{r+1} - x^{r} \right\|^{2}$.

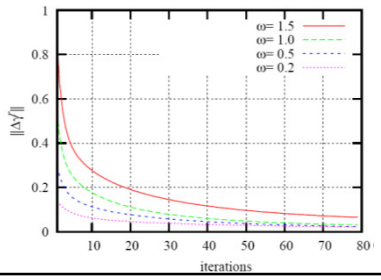
Always satisfied in multibody systems

Free choiche of the B^{r} matrix

Use ω factor and the B^{r} matrix to adjust this

Under the above assumptions, we can prove THEOREMS about convergence.

The method produces a bounded sequence with an unique accumulation point.

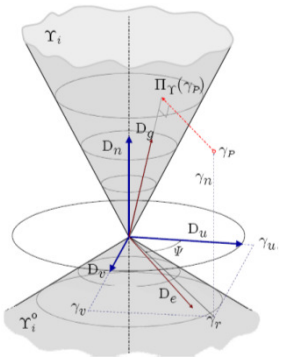


The projection operator

The projection operator must be non-extensive, i.e. lipschitzian with $\|f(a)-f(b)\| \leq \|a-b\|$

- ***For each frictional contact constraint:***

$$\Pi_\Upsilon = \left\{ \Pi_{\Upsilon_1}(\gamma_1)^T, \ldots \Pi_{\Upsilon_{n_\mathcal{A}}}(\gamma^{n_\mathcal{A}})^T, \Pi_b^1(\gamma_b^1), \ldots, \Pi_b^{n_B}(\gamma_b^{n_B}) \right\}^T$$



- ***For each bilateral constraint, simply do nothing.***

The complete operator:

$$\forall i \in \mathcal{A}(\boldsymbol{q}^{(l)}, \epsilon)$$

$$\begin{array}{ll}
\gamma_r < \mu_i \gamma_n & \Pi_i = \boldsymbol{\gamma}_i \\
\gamma_r < -\frac{1}{\mu_i}\gamma_n & \Pi_i = \{0,0,0\} \\
\gamma_r > \mu_i \gamma_n \wedge \gamma_r > -\frac{1}{\mu_i}\gamma_n & \Pi_{i,n} = \frac{\gamma_r \mu_i + \gamma_n}{\mu_i^2 + 1} \\
 & \Pi_{i,u} = \gamma_u \frac{\mu_i \Pi_{i,n}}{\gamma_r} \\
 & \Pi_{i,v} = \gamma_v \frac{\mu_i \Pi_{i,n}}{\gamma_r}
\end{array}$$

The algorithm

Development of an efficient algorithm for fixed point iteration:

$$\boldsymbol{\gamma}^{r+1} = \lambda \Pi_\Upsilon \left(\boldsymbol{\gamma}^r - \omega B^r \left(N\boldsymbol{\gamma}^r + \boldsymbol{r} + K^r \left(\boldsymbol{\gamma}^{r+1} - \boldsymbol{\gamma}^r \right) \right) \right) + (1-\lambda)\boldsymbol{\gamma}^r$$

With $K=$ 

$$N = D^T M^{-1} D$$

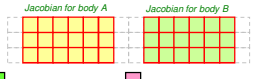
At each r-th iteration:

$$\delta^{i,r+1} = \gamma^{i,r} - \omega\eta_i \left(\boldsymbol{D}^{i,T} M^{-1} \left(\sum_{z=1}^{i-1} D^z \gamma^{z,r+1} + \sum_{z=i}^{n_\mathcal{A}} D^z \gamma^{z,r} + \tilde{\boldsymbol{k}}^i \right) + \boldsymbol{b}^i \right)$$

$$\gamma^{i,r+1} = \lambda \Pi_{\Upsilon^i} \left(\delta^{i,r+1} \right) + (1-\lambda)\boldsymbol{\gamma}^{i,r}$$

Loop on all i-th constraints

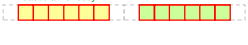
If i-th is a contact constraint:

$D^{i,T} =$ 

$\boldsymbol{\gamma}_a^i =$  $\boldsymbol{b}_a^i =$ 

$\eta_a^i = \frac{3}{\mathrm{Trace}(g_a^i)}$  $g_a^i = D^{i,T} M^{-1} D^i$

If i-th is a scalar bilateral constraint

$D^{i,T} = \nabla \Psi^{i,T} =$ 

$\gamma_b^i =$ $b_b^i =$ 

$\eta_b^i = \frac{1}{g_b^i}$  $g_b^i = D^{i,T} M^{-1} D^i$

The algorithm

Even better, in incremental form:

$$\delta^{i,r+1} = \gamma^{i,r} - \omega\eta_i \left(\boldsymbol{D}^{i,T} M^{-1} \left(\sum_{z=1}^{i-1} D^z \gamma^{z,r+1} + \sum_{z=i}^{n_A} D^z \gamma^{z,r} + \tilde{\boldsymbol{k}}^i \right) + b^i \right)$$

$$\gamma^{i,r+1} = \lambda \Pi_{\Upsilon^i} \left(\delta^{i,r+1} \right) + (1-\lambda)\gamma^{i,r}$$

Loop on all i-th constraints

Avoid these loops, otherwise each iteration would be O(n²)

Only one of these multiplier changes at each iteration...

We know that: $\boldsymbol{v} = M^{-1} D \boldsymbol{\gamma} + M^{-1}\tilde{\boldsymbol{k}}$ *..so we rewrite:*

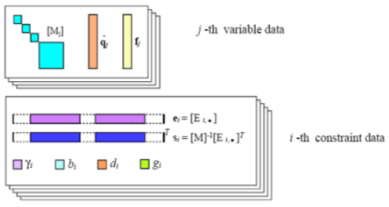
$$\begin{aligned}
\boldsymbol{\delta}^{i,r+1} &= \left(\boldsymbol{\gamma}^{i,r} - \omega\eta^i \left(D^{i,T} \boldsymbol{v}^r + \boldsymbol{b}^i \right) \right); \\
\boldsymbol{\gamma}^{i,r+1} &= \lambda \Pi_{\Upsilon} \left(\boldsymbol{\delta}^{i,r+1} \right) + (1-\lambda)\boldsymbol{\gamma}^{i,r} \; ; \\
\Delta\boldsymbol{\gamma}^{i,r+1} &= \boldsymbol{\gamma}^{i,r+1} - \boldsymbol{\gamma}^{i,r} \; ; \\
\boldsymbol{v} &:= \boldsymbol{v} + M^{-1} D^i \Delta\boldsymbol{\gamma}^{i,r+1}
\end{aligned}$$

Loop on all i-th constraints

This 'incremental' form has O(n) complexity!!!

The algorithm

Development of an efficient algorithm for fixed point iteration:



- ***avoid temporary data, exploit sparsity. Never compute explicitly the N matrix!***
- ***implemented in incremental form. Compute only deltas of multipliers.***
- ***O(n) space requirements***
- ***supports premature termination***
- ***for real-time purposes: O(n) time***

The algorithm

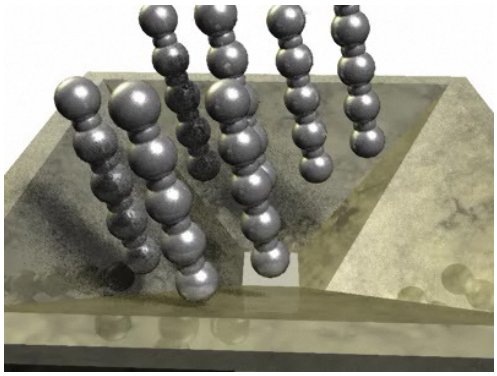
Pseudocode:

(1) *// Pre-compute some data for friction constraints*
(2) **for** $i := 1$ **to** $n_{\mathcal{A}}$
(3) $\quad \boldsymbol{s}_a^i = M^{-1} D^i$
(4) $\quad g_a^i = D^{i,T} \boldsymbol{s}_a^i$
(5) $\quad \eta_a^i = \frac{3}{\mathrm{Trace}(g_a^i)}$
(6) *// Pre-compute some data for bilateral constraints*
(7) **for** $i := 1$ **to** $n_{\mathcal{B}}$
(8) $\quad s_b^i = M^{-1} \nabla \Psi^i$
(9) $\quad g_b^i = \nabla \Psi^{i,T} s_b^i$
(10) $\quad \eta_b^i = \frac{1}{g_b^i}$
(11)
(12) *// Initialize impulses*
(13) **if** warm start with initial guess $\boldsymbol{\gamma}_{\mathcal{E}}^*$
(14) $\quad \boldsymbol{\gamma}_{\mathcal{E}}^0 = \boldsymbol{\gamma}_{\mathcal{E}}^*$
(15) **else**
(16) $\quad \boldsymbol{\gamma}_{\mathcal{E}}^0 = 0$
(17)
(18) *// Initialize speeds*
(19) $\boldsymbol{v} = \sum_{i=1}^{n_{\mathcal{A}}} \boldsymbol{s}_a^i \boldsymbol{\gamma}_a^{i,0} + \sum_{i=1}^{n_{\mathcal{B}}} s_b^i \gamma_b^{i,0} + M^{-1} \tilde{\boldsymbol{k}}$
(21) *// Main iteration loop*
(22) **for** $r := 0$ **to** r_{max}
(23) $\quad$ *// Loop on frictional constraints*
(24) $\quad$ **for** $i := 1$ **to** $n_{\mathcal{A}}$
(25) $\quad\quad \boldsymbol{\delta}_a^{i,r} = \left(\boldsymbol{\gamma}_a^{i,r} - \omega \eta_a^i \left(D^{i,T} \boldsymbol{v}^r + \boldsymbol{b}_a^i \right) \right);$
(26) $\quad\quad \boldsymbol{\gamma}_a^{i,r+1} = \lambda \Pi_{\Upsilon} \left(\boldsymbol{\delta}_a^{i,r} \right) + (1-\lambda) \boldsymbol{\gamma}_a^{i,r}$;
(27) $\quad\quad \Delta \boldsymbol{\gamma}_a^{i,r+1} = \boldsymbol{\gamma}_a^{i,r+1} - \boldsymbol{\gamma}_a^{i,r}$;
(28) $\quad\quad \boldsymbol{v} := \boldsymbol{v} + \boldsymbol{s}_a^{iT} \Delta \boldsymbol{\gamma}_a^{i,r+1}$.
(29) $\quad$ *// Loop on bilateral constraints*
(30) $\quad$ **for** $i := 1$ **to** $n_{\mathcal{B}}$
(31) $\quad\quad \delta_b^{i,r} = \left(\gamma_b^{i,r} - \omega \eta_b^i \left(\nabla \Psi^{i,T} \boldsymbol{v}^r + b_b^i \right) \right);$
(32) $\quad\quad \gamma_b^{i,r+1} = \lambda \Pi_{\Upsilon} \left(\delta_b^{i,r} \right) + (1-\lambda) \gamma_b^{i,r}$;
(33) $\quad\quad \Delta \gamma_b^{i,r+1} = \gamma_b^{i,r+1} - \gamma_b^{i,r}$;
(34) $\quad\quad \boldsymbol{v} := \boldsymbol{v} + s_b^{iT} \Delta \gamma_b^{i,r+1}$.
(35)
(36) **return** $\boldsymbol{\gamma}_{\mathcal{E}}, \boldsymbol{v}$

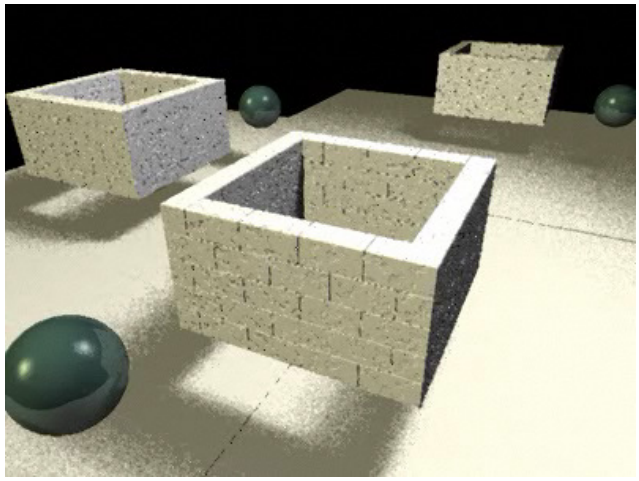
Examples

Mixing bilateral and unilateral constraints

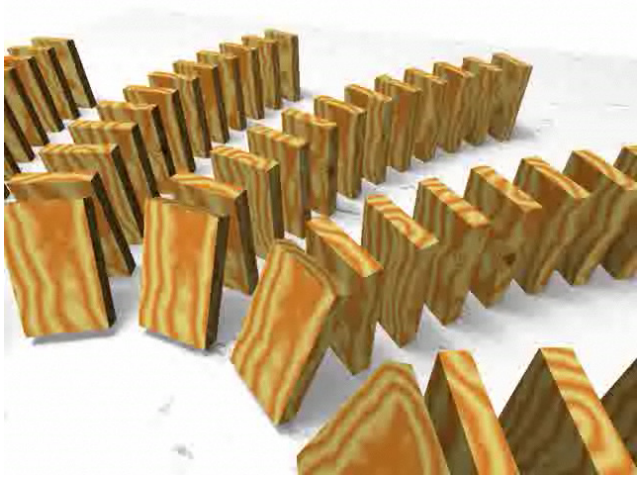
- **Bilaterals: spherical joints between the balls**
- **Unilaterals: collisions + min/max rotation limits for balls**
- **No fritcion**



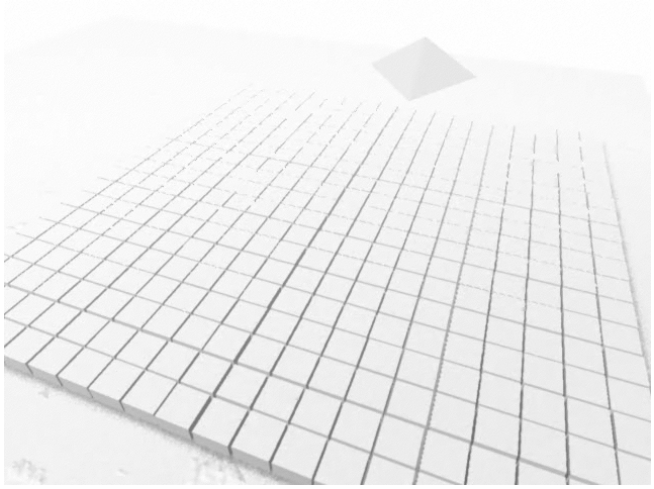
Examples



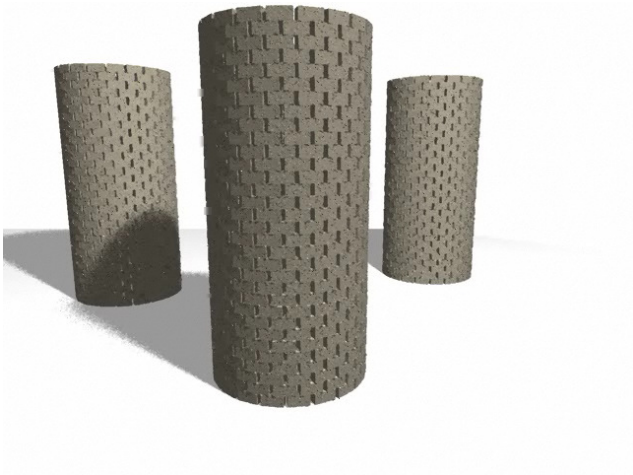
Examples



Examples



Examples

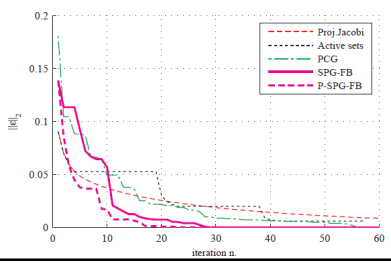


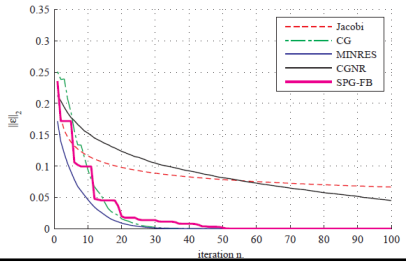
Better algorithms?

The projected fixed point method has slow convergence!

New methods under development

- **SPG modified Spectral Projected Gradient P-SPG-FB**
- **APGD Accelerated Projected Gradient Descend**
- **Interior point?**
- **FAS Multigrid?**





Better algorithms?

- **Currently most solvers for the VI / CCP problem are based on *fixed point* iterations:**
  - Projected Gauss-Jacobi,
  - Projected Gauss-Seidell / SOR, ← *presented in the previous slides*
  - Mirtich 'microimpulses' method,

  These are robust, but their convergence is slow!

- **On the other side, Krylov stationary methods have fast convergence, but are limited to linear problems (no contacts!)**
  - Conjugate Gradient
  - MINRES
  - GMRES
  - Etc.

WE NEED THE BENEFITS OF BOTH, without their shortcomings!

Better algorithm: our P-SPG-FB

In case of convexified problem (i.e. 'associative flows' as our CCP) one can express the VI as a constrained quadratic program:

$$\begin{aligned} \min \quad & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{x}^T\mathbf{b} \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{K} \end{aligned}$$

One can use the Spectral Projected Gradient method for solving it!

The P-SPG-FB method

Our modified P-SPG-FB algorithm

- **Supports premature termination with fall-back strategy (FB)**
- **Uses alternating step sizes**
- **Uses diagonal preconditioning (with isothropic cone scaling)** $P = \overline{\text{diag}}(A)$
- **Performs projection onto Lorentz cones**

ALGORITHM P-SPG-FB(A, $\mathbf{b}$, $\mathbf{x}_0$, $\mathcal{K}$, $P \mapsto \mathbf{x}$)
$\mathbf{x}_0 := \Pi_{\mathcal{K}}(\mathbf{x}_0)$, $\mathbf{x}_{FB} = \mathbf{x}_0$,
$\hat{\alpha}_0 \in [\alpha_{min}, \alpha_{max}]$
$\mathbf{g}_0 := A\mathbf{x}_0 + \mathbf{b}$, $f(\mathbf{x}_0) = \frac{1}{2}\mathbf{x}_0^T A\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}$, $w_0 = 10^{29}$
for $j := 0$ **to** N_{max}
  $\mathbf{p}_j = P^{-1}\mathbf{g}_j$
  $\mathbf{d}_j = \Pi_{\mathcal{K}}(\mathbf{x}_j - \hat{\alpha}_j\mathbf{p}_j) - \mathbf{x}_j$
  if $\langle \mathbf{d}_j, \mathbf{g}_j \rangle \geq 0$
    $\mathbf{d}_j = \Pi_{\mathcal{K}}(\mathbf{x}_j - \hat{\alpha}_j\mathbf{g}_j) - \mathbf{x}_j$
  $\lambda := 1$
  while line search
    $\mathbf{x}_{j+1} := \mathbf{x}_j + \lambda\mathbf{d}_j$
    $\mathbf{g}_{j+1} := A\mathbf{x}_{j+1} + \mathbf{b}$
    $f(\mathbf{x}_{j+1}) = \frac{1}{2}\mathbf{x}_{j+1}^T A\mathbf{x}_{j+1} + \mathbf{x}_{j+1}^T\mathbf{b}$
    if $f(\mathbf{x}_{j+1}) > \max_{i=0,\ldots,\min(j,N_{GLL})} f(\mathbf{x}_{j-i}) + \gamma\lambda\langle \mathbf{d}_j, \mathbf{g}_j \rangle$
      define $\lambda_{new} \in [\sigma_{min}\lambda, \sigma_{max}\lambda]$ and repeat line search
    else
      terminate line search
  $\mathbf{s}_j = \mathbf{x}_{j+1} - \mathbf{x}_j$
  $\mathbf{y}_j = \mathbf{g}_{j+1} - \mathbf{g}_j$
  if j is odd
    $\hat{\alpha}_{j+1} = \frac{\langle \mathbf{s}_j, P\mathbf{s}_j \rangle}{\langle \mathbf{s}_j, \mathbf{s}_j \rangle}$
  else
    $\hat{\alpha}_{j+1} = \frac{\langle \mathbf{s}_j, \mathbf{s}_j \rangle}{\langle \mathbf{s}_j, P^{-1}\mathbf{s}_j \rangle}$
  $\hat{\alpha}_{j+1} = \min(\alpha_{max}, \max(\alpha_{min}, \hat{\alpha}_{j+1}))$
  $w_{j+1} = \left|\left| [\mathbf{x}_{j+1} - \Pi_{\mathcal{K}}(\mathbf{x}_{j+1} - \tau_g\mathbf{g}_{j+1})] / \tau_g \right|\right|_2 = ||\varepsilon||_2$
  if $w_{j+1} \leq \min_{k=0,\ldots,j} w_k$
    $\mathbf{x}_{FB} = \mathbf{x}_{j+1}$
return $\mathbf{x}_{FB}$

The P-SPG-FB method

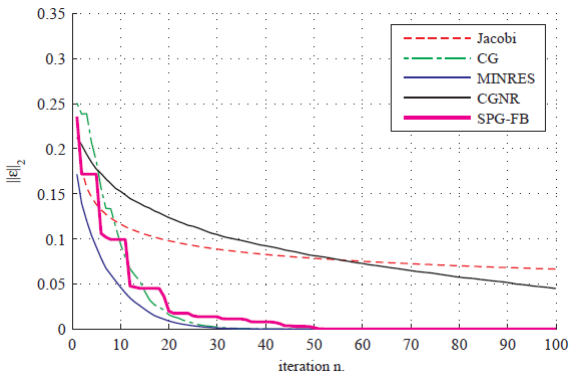
Our modified P-SPG-FB algorithm

- **Supports premature termination with fall-back strategy (FB)**
- **Uses alternating step sizes**
- **Uses diagonal preconditioning (with isothropic cone scaling)** $P = \overline{\text{diag}(A)}$
- **Performs projection onto Lorentz cones**

ALGORITHM P-SPG-FB(A, $\mathbf{b}$, $\mathbf{x}_0$, $\mathcal{K}$, $P \mapsto \mathbf{x}$)
  $\mathbf{x}_0 := \Pi_{\mathcal{K}}(\mathbf{x}_0)$, $\mathbf{x}_{FB} = \mathbf{x}_0$,
$\bar{\alpha}_0 \in [\sigma_{min}, \sigma_{max}]$
$\mathbf{g}_0 := A\mathbf{x}_0 + \mathbf{b}$, $f(\mathbf{x}_0) = \frac{1}{2}\mathbf{x}_0^T A\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}$, $w_0 = 10^{29}$
for $j := 0$ **to** N_{max}
  $\mathbf{p}_j = P^{-1}\mathbf{g}_j$
  $\mathbf{d}_j = \Pi_{\mathcal{K}}(\mathbf{x}_j - \bar{\alpha}_j\mathbf{p}_j) - \mathbf{x}_j$
  if $\langle \mathbf{d}_j, \mathbf{g}_j \rangle \geq 0$
    $\mathbf{d}_j = \Pi_{\mathcal{K}}(\mathbf{x}_j - \bar{\alpha}_j\mathbf{g}_j) - \mathbf{x}_j$
  $\lambda := 1$
  while line search
    $\mathbf{x}_{j+1} := \mathbf{x}_j + \lambda\mathbf{d}_j$
    $\mathbf{g}_{j+1} := A\mathbf{x}_{j+1} + \mathbf{b}$
    $f(\mathbf{x}_{j+1}) = \frac{1}{2}\mathbf{x}_{j+1}^T A\mathbf{x}_{j+1} + \mathbf{x}_{j+1}^T\mathbf{b}$
    if $f(\mathbf{x}_{j+1}) > \max_{i=0,\ldots,\min(j,N_{GLL})} f(\mathbf{x}_{j-i}) + \gamma\lambda\langle \mathbf{d}_j, \mathbf{g}_j \rangle$
      define $\lambda_{new} \in [\sigma_{min}\lambda, \sigma_{max}\lambda]$ and repeat line search
    else
      terminate line search
  $\mathbf{s}_j = \mathbf{x}_{j+1} - \mathbf{x}_j$
  $\mathbf{y}_j = \mathbf{g}_{j+1} - \mathbf{g}_j$
  if j is odd
    $\bar{\alpha}_{j+1} = \frac{\langle \mathbf{s}_j, P\mathbf{s}_j \rangle}{\langle \mathbf{s}_j, \mathbf{y}_j \rangle}$
  else
    $\bar{\alpha}_{j+1} = \frac{\langle \mathbf{s}_j, \mathbf{s}_j \rangle}{\langle \mathbf{s}_j, P^{-1}\mathbf{y}_j \rangle}$
  $\bar{\alpha}_{j+1} = \min(\sigma_{max}, \max(\sigma_{min}, \bar{\alpha}_{j+1}))$
  $w_{j+1} = \left|\left| [\mathbf{x}_{j+1} - \Pi_{\mathcal{K}}(\mathbf{x}_{j+1} - \tau_g\mathbf{g}_{j+1})] / \tau_g \right|\right|_2 = ||e||_2$
  if $w_{j+1} \leq \min_{k=0,\ldots,j} w_k$
    $\mathbf{x}_{FB} = \mathbf{x}_{j+1}$
return $\mathbf{x}_{FB}$

Results

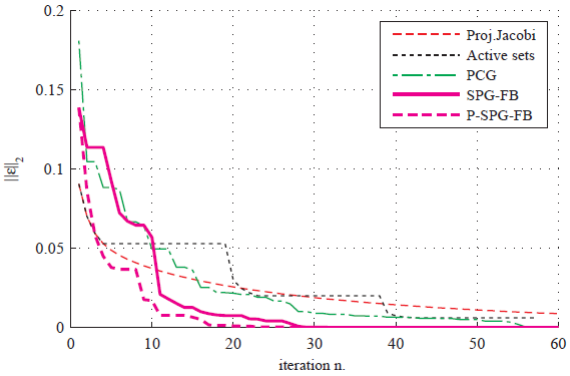
Comparison with other Krylov solvers for simple linear case (only bilateral constraints):

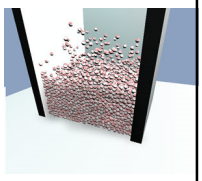


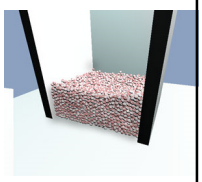
Results

Comparison with other solvers for complemetarity problems

(only unilateral contacts, no friction)



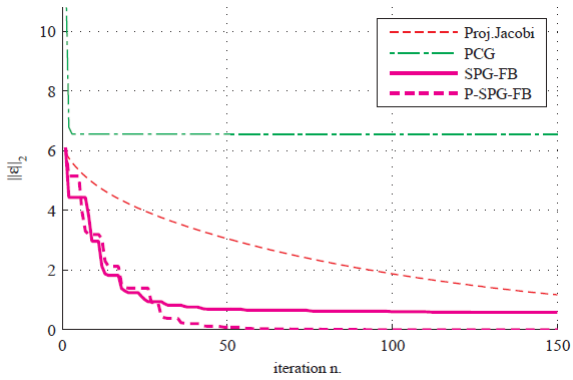


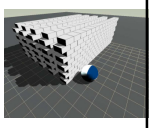


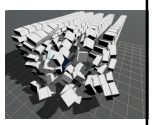
Results

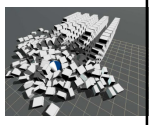
Comparison with other solvers for complemetarity problems

(unilateral contacts AND friction - few solvers can handle it)



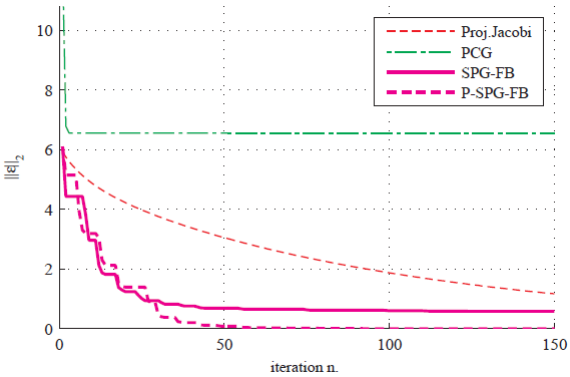


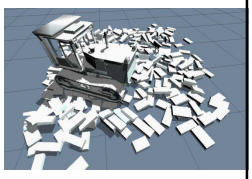




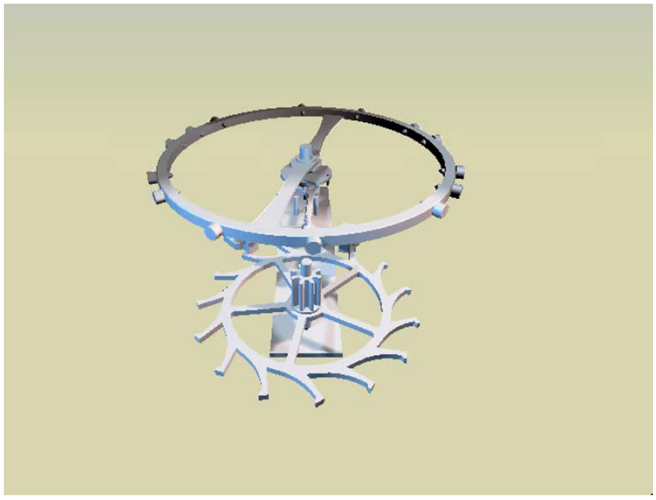
Results

Effect of preconditioning:



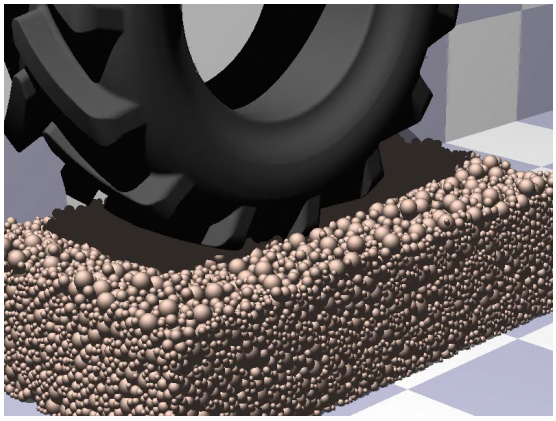


Examples – Swiss escapement



Example - Vehicles on granular soil

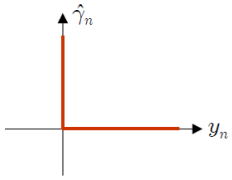
Tire on a granular soil, with CHRONO::ENGINE



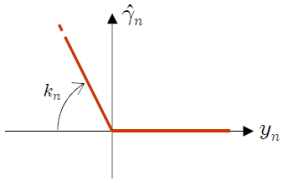
In collaboration with F.Braghin, Politecnico di Milano.

DVI advanced contact laws

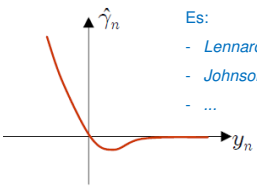
Rigid contact:



Compliant contact:

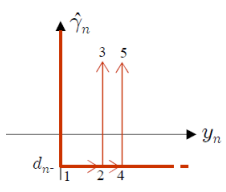


Nonlinear, with cohesion:



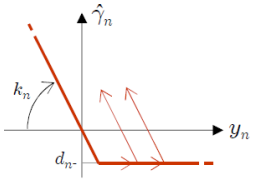
Es:
- *Lennard-Jones*
- *Johnson-Kendall-Roberts*
- ...

Rigid, with plastic cohesion

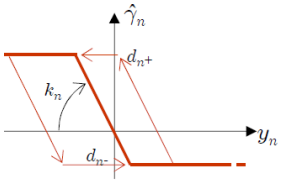


DVI advanced contact laws

Compliant, plastic cohesion

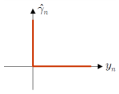


Compliant, plastic cohesion and compression



In general, DVI are useful for various reasons that are diffult to handle in DAE:

- ***very stiff or rigid contacts → set valued force laws → VI***
- ***plasticity in contacts → yield surfaces → VI***
- ***friction → set valued force laws → VI***



DVI Elasto-Plastic contact

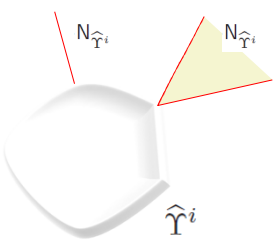
Contact forces $\hat{\gamma}^i_{\mathcal{A}} = \{\hat{\gamma}^i_n, \hat{\gamma}^i_u, \hat{\gamma}^i_w\}^T$

Inclusion in yield surface: $\hat{\gamma}^i_{\mathcal{A}} \in \hat{\Upsilon}^i$

Prandtl-Reuss-like assumption $\boldsymbol{y}^i = \boldsymbol{y}^i_E + \boldsymbol{y}^i_P$ *on displacements* y

Associated flow assumption: $\dot{\boldsymbol{y}}^i_P \in -\mathrm{N}_{\hat{\Upsilon}^i}(\hat{\gamma}^i_{\mathcal{A}})$

The increment to the plastic flow is orthogonal to the yield surface



DVI Elasto-Plastic contact

Elasto-plastic model:

$$\begin{aligned} \widehat{\gamma}^i_{\mathcal{A}} &= -K^i \left(\boldsymbol{y}^i - \boldsymbol{y}^i_P \right) \qquad\qquad K^i \in \mathbb{R}^{3\times 3} \\ \dot{\boldsymbol{y}}^i_P &\in -\mathsf{N}_{\widehat{\Upsilon}^i}(\widehat{\gamma}^i_{\mathcal{A}}) \quad ; \quad \widehat{\gamma}^i_{\mathcal{A}} \in \widehat{\Upsilon}^i \end{aligned}$$

$$\dot{\widehat{\gamma}}^i_{\mathcal{A}} = -K^i \left(\dot{\boldsymbol{y}}^i - \dot{\boldsymbol{y}}^i_P \right)$$

With time discretization: $h = t^{l+1} - t^l \qquad h\widehat{\gamma} = \gamma \qquad \Upsilon = h\widehat{\Upsilon}$

$$\frac{\widehat{\gamma}^{i,l+1}_{\mathcal{A}} - \widehat{\gamma}^{i,l}_{\mathcal{A}}}{h} = -K^i \left(D^{i^T} \boldsymbol{v}^{l+1} - \dot{\boldsymbol{y}}^i_P \right)$$

$$\dot{\boldsymbol{y}}^i_P = D^{i^T} \boldsymbol{v}^{l+1} + \left(h^2 K^i\right)^{-1} \gamma^{i,l+1}_{\mathcal{A}} - \left(h^2 K^i\right)^{-1} \gamma^{i,l}_{\mathcal{A}} \quad \in -\mathsf{N}_{\Upsilon^i}(\gamma^i_{\mathcal{A}})$$

DVI Elasto-Plastic contact

$$\dot{\boldsymbol{y}}^i_P = D^{i^T} \boldsymbol{v}^{l+1} + \left(h^2 K^i\right)^{-1} \gamma^{i,l+1}_{\mathcal{A}} - \frac{1}{h}\left(\boldsymbol{y}^{i,l} - \boldsymbol{y}^{i,l}_P \right) \quad \in -\mathsf{N}_{\Upsilon^i}(\gamma^i_{\mathcal{A}})$$

Define:

$$\begin{aligned} E^i &= -\left(h^2 K^i\right)^{-1} \\ \boldsymbol{c}^i &= -\frac{1}{h}\left(\boldsymbol{y}^{i,l} - \boldsymbol{y}^{i,l}_P \right) \end{aligned}$$

$$\dot{\boldsymbol{y}}^i_P = D^{i^T} \boldsymbol{v}^{l+1} - E^i \widehat{\gamma}^{i,l+1}_{\mathcal{A}} - \boldsymbol{c}^i \quad \in -\mathsf{N}_{\Upsilon^i}(\widehat{\gamma}^i_{\mathcal{A}})$$

$$M\boldsymbol{v}^{l+1} = M\boldsymbol{v}^l + \sum_{i\in\mathcal{G}_{\mathcal{A}}} D^i \gamma^{i,l+1}_{\mathcal{A}} + h\boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t)$$

$$\begin{aligned} \gamma_{\mathcal{E}} &= \left\{ \gamma^{1^T}_{\mathcal{A}}, \gamma^{2^T}_{\mathcal{A}}, \dots \right\}^T \\ D_{\mathcal{E}} &= \left[D^1 | D^2 | \dots \right] \\ \boldsymbol{c} &= \left\{ \boldsymbol{c}^{1^T}, \boldsymbol{c}^{2^T}, \dots \right\}^T \end{aligned}$$

$$\dot{\boldsymbol{y}}_P = \left[D_{\mathcal{E}}{}^T M D_{\mathcal{E}} - E_{\mathcal{E}} \right] \gamma^{l+1}_{\mathcal{E}} + D_{\mathcal{E}}{}^T \left(\boldsymbol{v}^l + hM^{-1}\boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t) \right) - \boldsymbol{c} \quad \in -\mathsf{N}_{\Upsilon}(\widehat{\gamma}_{\mathcal{E}})$$

DVI Elasto-Plastic contact

Posing:

$$N = \left[D_{\mathcal{E}}{}^T M D_{\mathcal{E}} - E_{\mathcal{E}}\right]$$
$$r = +D_{\mathcal{E}}{}^T \left(v^l + hM^{-1}\boldsymbol{f}(\boldsymbol{q}, v, t)\right) - c$$

One finally gets the VI:

$$N\gamma_{\mathcal{E}}^{l+1} + \boldsymbol{r} \in -\mathsf{N}_{\Upsilon}(\gamma_{\mathcal{E}}) \; ; \; \gamma_{\mathcal{E}}^{l+1} \in \Upsilon$$

That can be written also as the 'classical' VI:

$$\gamma_{\mathcal{E}}^{l+1} \in \Upsilon : \quad \left\langle N\gamma_{\mathcal{E}}^{l+1} + \boldsymbol{r}, \boldsymbol{z} - \gamma_{\mathcal{E}}^{l+1} \right\rangle \geq 0 \quad \forall \boldsymbol{z} \in \Upsilon$$

DVI Elasto-Plastic contact

Note: the VI, for associated plastic flow, is also a convex minimization problem

$$\gamma_{\mathcal{E}}^{l+1} \in \Upsilon : \quad \left\langle N\gamma_{\mathcal{E}}^{l+1} + \boldsymbol{r}, \boldsymbol{z} - \gamma_{\mathcal{E}}^{l+1} \right\rangle \geq 0 \quad \forall \boldsymbol{z} \in \Upsilon$$

$$\Updownarrow$$

$$\min_{\gamma_{\mathcal{E}} \in \Upsilon} f(\gamma_{\mathcal{E}}) \quad \Leftrightarrow \quad \gamma_{\mathcal{E}} \in \Upsilon, \; \mathrm{grad} f(\gamma_{\mathcal{E}}) \in -\mathsf{N}_{\Upsilon}(\gamma_{\mathcal{E}})$$

$$\Updownarrow$$

$$\min_{\gamma_{\mathcal{E}} \in \Upsilon} \frac{1}{2}\gamma_{\mathcal{E}}^T N \gamma_{\mathcal{E}} + \gamma_{\mathcal{E}}^T \boldsymbol{r}$$

Visco-Elasto-Plastic contact

By introducing also viscous damping, one gets the model

$$\begin{aligned}\widehat{\gamma}_{\mathcal{A}}^{i} &= -K^{i}\left(\boldsymbol{y}^{i}-\boldsymbol{y}_{P}^{i}\right)-R^{i}\left(\dot{\boldsymbol{y}}^{i}-\dot{\boldsymbol{y}}_{P}^{i}\right)\\ \dot{\boldsymbol{y}}_{P}^{i} &\in -\mathsf{N}_{\widehat{\Upsilon}^{i}}(\widehat{\gamma}_{\mathcal{A}}^{i}) \quad ; \quad \widehat{\gamma}_{\mathcal{A}}^{i}\in\widehat{\Upsilon}^{i}\end{aligned}$$

Again one obtains a VI, this time with:

$$\begin{aligned}E^{i} &= -\left(h^{2}K^{i}+hR^{i}\right)^{-1}\\ c^{i} &= -\left(h^{2}K^{i}+hR^{i}\right)^{-1}\left(\gamma_{\mathcal{A}}^{i,l}+hR^{i}(\dot{\boldsymbol{y}}^{l}-\dot{\boldsymbol{y}}_{P}^{l})\right)\\ N &= \left[D_{\mathcal{E}}{}^{T}MD_{\mathcal{E}}-E_{\mathcal{E}}\right]\\ \boldsymbol{r} &= +D_{\mathcal{E}}{}^{T}\left(\boldsymbol{v}^{l}+hM^{-1}\boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t)\right)-\boldsymbol{c}\end{aligned}$$

$$\gamma_{\mathcal{E}}^{l+1}\in\Upsilon: \quad \left\langle N\gamma_{\mathcal{E}}^{l+1}+\boldsymbol{r},\boldsymbol{z}-\gamma_{\mathcal{E}}^{l+1}\right\rangle\geq 0 \quad \forall\boldsymbol{z}\in\Upsilon$$

DVI Visco-Elasto-Plastic contact

With Raleygh damping → simplification

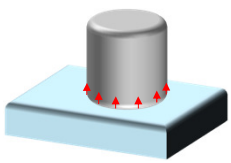
$$R^{i}=\alpha_{K}^{i}K^{i}$$

Obtaining:

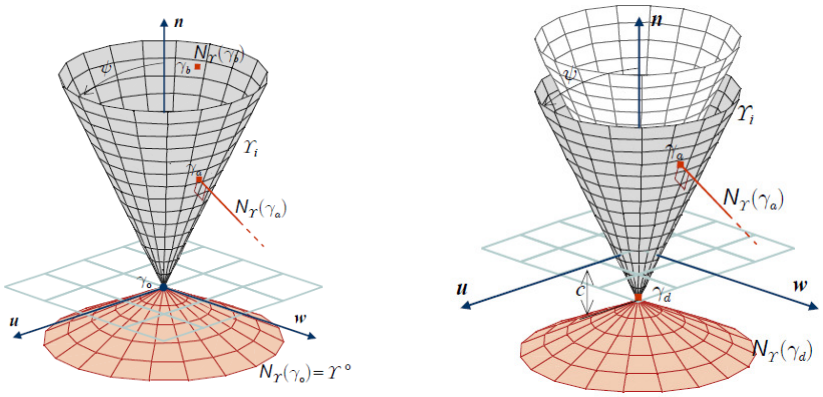
$$\begin{aligned}E^{i} &= -\frac{1}{h(h+\alpha_{K}^{i})}K^{i^{-1}}\\ c^{i} &= -\frac{1}{h+\alpha_{K}^{i}}\left(\dot{\boldsymbol{y}}^{l}-\dot{\boldsymbol{y}}_{P}^{l}\right)\end{aligned}$$

NOTE

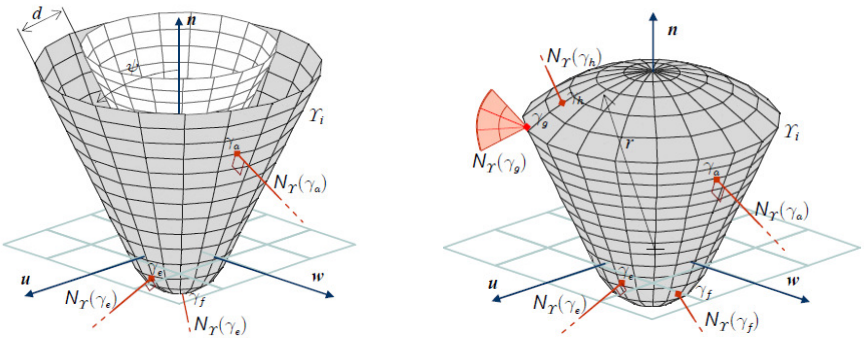
the E term works as a Tykhonov regularization of the Schur complement $N=\left[D_{\mathcal{E}}{}^{T}MD_{\mathcal{E}}-E_{\mathcal{E}}\right]$



Yield surfaces

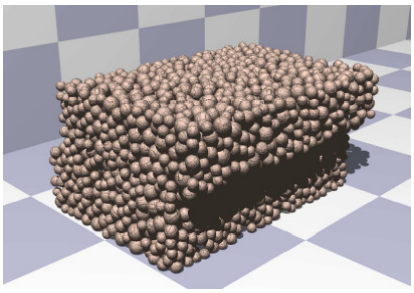


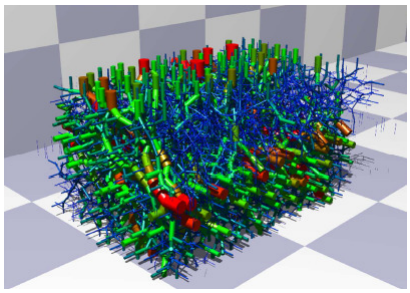
Yield surfaces



Examples

Granular flows (shear test)

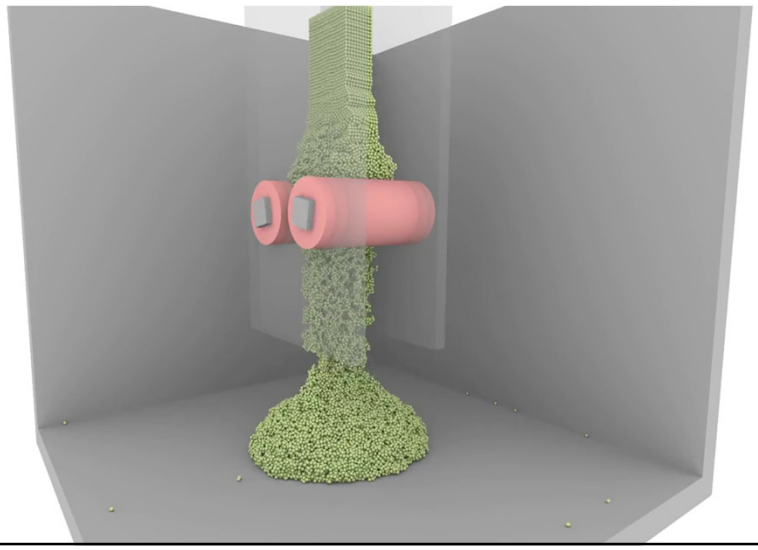




Examples

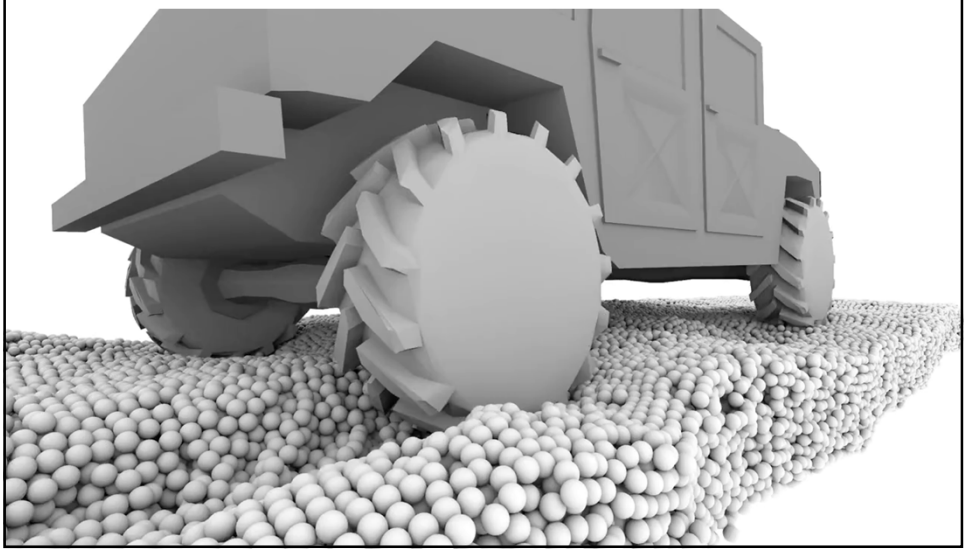
Benchmark of our software Chrono::Engine

(animation by H.Mazhar ,2013)



Examples

Benchmark of our software Chrono::Engine



Section

Software implementation

Multibody software

Classification of MB software

By license:
- Commercial
- Open source

By architecture:
- Stand-alone application with user interface (GUI)
- Only solver (batch processing)
- As a plug-in for 3rd party CAD
- As middleware (library)

By purpose:
- General purpose
- Vertical (application-oriented)
- Real-time

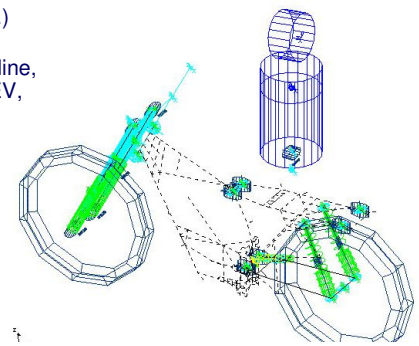
...

Multibody software

Famous commercial software (with GUI):

- **ADAMS**
  - Pioneer of MB, tested and reliable
  - Poweful analysis functions
  - Targeted at 'serious' engineering stuff
  - Customizable
  - Many solvers (but unfit for contacts..)
  - Available modules for powertrains and vehicle dynamics (Adams/Driveline, Adams/Car, Adams/SmartDriver, FEV, etc.)
  - Pre-post processing GUI not always easy to use…

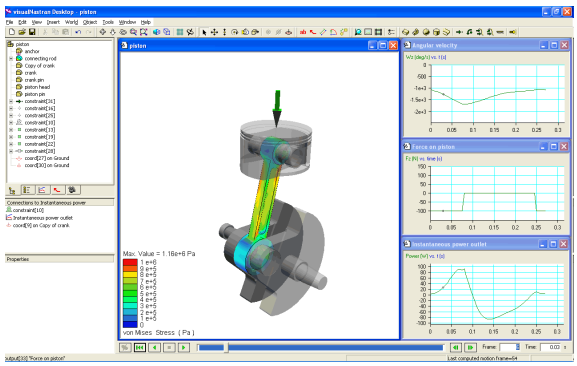
MSC Software
Simulate More



Multibody software

Famous commercial software (with GUI):

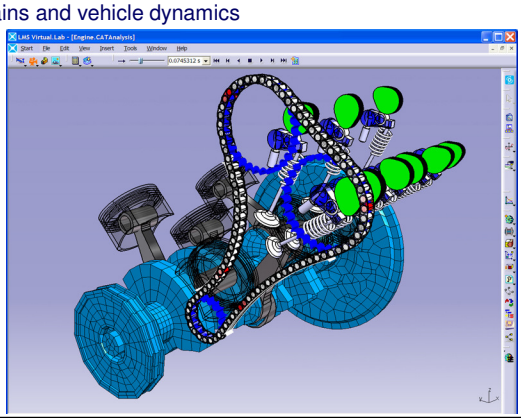
- **VisualNastran4D**
  - Simple to use
  - Cheap (but discontinued!)
  - Good GUI interface
  - Fast simulation



Multibody software

Famous commercial software (with GUI):

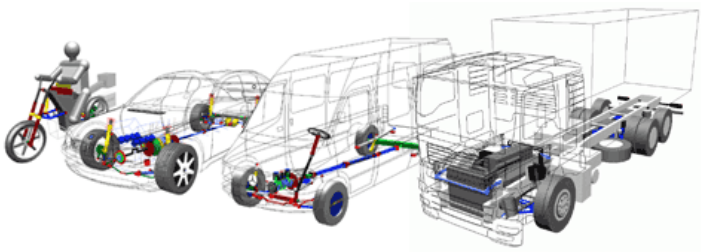
- **LMS Virtual.Lab Motion (DADS)**
  - For engineering tasks
  - It was a competitor of ADAMS (Prof. Haug)
  - Available modules for powertrains and vehicle dynamics
  - Sunspension templates, etc.
  - Interfaced with CATIA



Multibody software

Famous commercial software (with GUI):

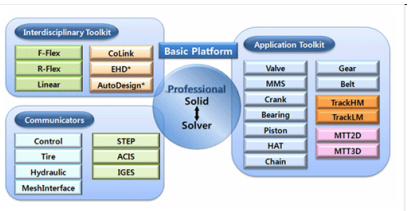
- **SIMPACK**
  - Poweful features
  - Based on fast recursive formulation
  - Quickly growing in automotive field
  - De-facto standard in train engineering
  - Available modules for powertrains and vehicle dynamics

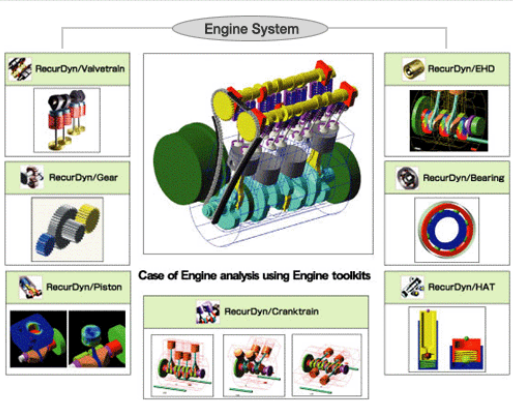


Multibody software

Famous commercial software (with GUI):

- **RECURDYN**
  - Based on fast recursive formulation
  - Developed in Korea,
  - Recent product
  - Lot of modules for automotive applications
  - In NX CAD as 'NX Motion'

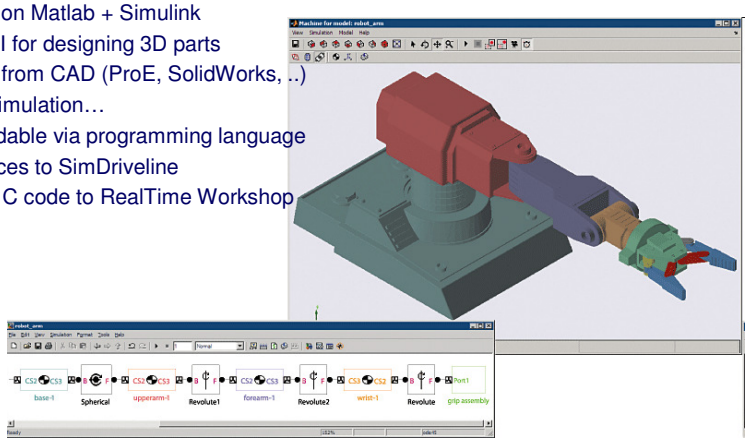




Multibody software

Famous commercial software (with GUI*):

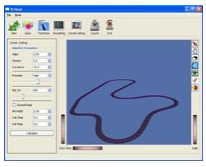
- **SimMechanics (Matlab)**
  - Based on Matlab + Simulink
  - No GUI for designing 3D parts
  - Import from CAD (ProE, SolidWorks, ..)
  - Slow simulation…
  - Expandable via programming language
  - Interfaces to SimDriveline
  - Export C code to RealTime Workshop

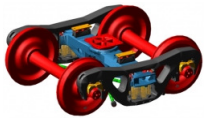


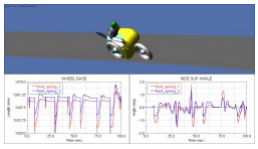
Multibody software

Special purpose commercial software – ex: vehicles

- **VI-GRADE suite (based on Adams)**
  - VI-Sportcar
  - VI-Train
  - VI-Motorcycle
  - etc…







Multibody software

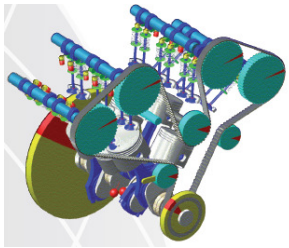
Special purpose commercial software – ex: vehicles

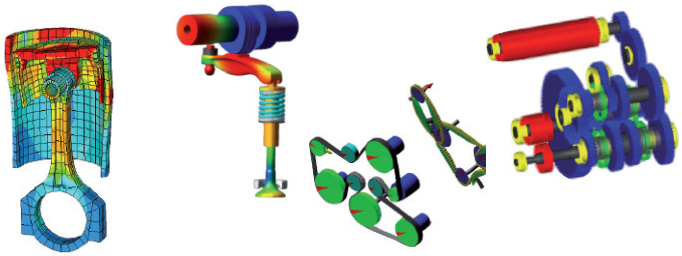
- **VI-GRADE suite (based on Adams)**
  - VI-CarRealTime

Multibody software

Special purpose commercial software – ex: vehicles

- **VI-GRADE FEV VIRTUAL ENGINE**
  - Crank train module
  - Timing Drive module
  - Valve train module
  - Gear drive module
  - Piston dynamics module

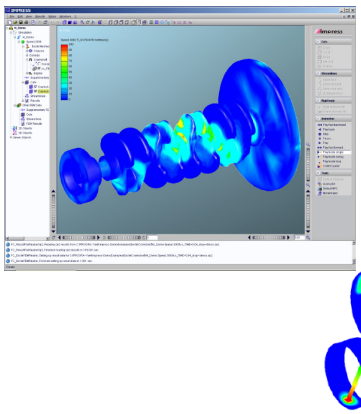


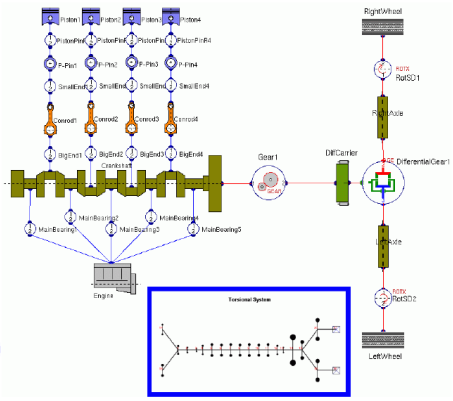


Multibody software

Special purpose commercial software – ex: vehicles

- **AVL EXCITE**





Multibody software

Commercial middleware & APIs:

- **HAVOK**
  - Widespread
  - For videogames mostly
  - Very fast & reliable
  - Implemented on GPU boards

- **PhysX (ex Ageia, ex Novodex, ex Meqon)**
  - Powerful SDK
  - Used also for engineering
  - Competing with HAVOK – bought by NVIDIA

- **PIXELUX**
  - Digital molecular matter (DMM)
  - Realtime FEM
  - Biased toward efficiency

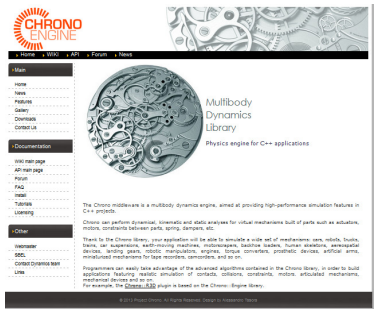
Multibody software

Open source / opaque source free middleware:

- **ODE**
  - OpenSource
  - Large user base
  - Not optimized, dirty API
- **NEWTON**
- **TOKAMAK**
  - All are very fast
  - Opaque source, all are free for non-commercial purposes
- **CHRONO::ENGINE**
  - Our project…
  - Work in progress..
- **BULLET**
  - Specialized in collision detection – biased toward efficiency
- **MBDYN**
  - Developed at Politecnico – biased toward precision

Multibody software

Our Chrono::Engine *middleware* project:



- **Middleware: can be used by third parties**
- **Efficient and fast, real-time if possible**
- **Expandable via C++ class inheritance**
- **Robust and reliable**
- **Embeddable in VR applications**
- **Cross-platform**
- **State-of-the-art collision-detection**

Multibody software

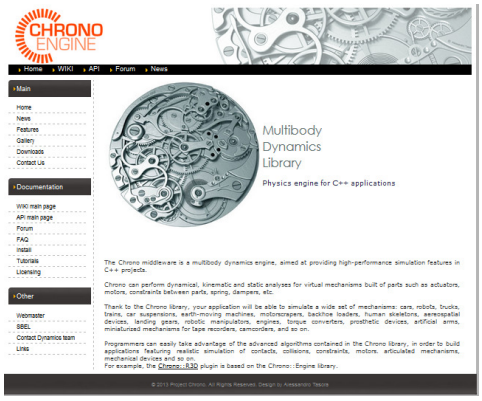
Part of ProjectChrono: ← *very recent initiative, more to come...*

CHRONO ENGINE

CHRONO RENDER

CHRONO FLEX

CHRONO



Features

Features of the Chrono::Engine middleware (1)

Core features

- Optimized custom classes for vectors, quaternions, matrices.
- Optimized custom classes for coordinate systems and coordinate transformations
- All operations on points/ speeds/ accelerations are based on quaternion algebra
- Custom sparse matrix class
- Linear algebra functions for LU decomposition, Choleski, Von Kauffmann, LDLt -SVD etc.
- Custom redirectable stream classes, featuring platform independent archiving.
- Special archive engine, with persistent/transient serialization, versioning and deep pointers storage.
- Nonintrusive memory debugger, to track memory leaks.
- Intrusive and non-intrusive smart pointers.
- High resolution timer, platform independent.
- …

Features

Features of the Chrono::Engine middleware (2)

Physical modeling

- Rigid bodies, markers, forces, torques
- Bodies can be activated/deactivated, and can selectively partecipate to collision detection.
- Exact Coloumb friction model, for precise stick-slip of bodies.
- Parts can collide and rebounce, depending on restitution coefficients.
- Springs and dampers, even with non-linear features
- Wide set of joints (spherical, revolute joint, prismatic, universal joint, glyph, etc.)
- Constraints to impose trajectories, or to force motion on splines, curves, surfaces, etc.
- Constraints can have limits (ex. elbow) and can be motorized
- Custom constraint for linear motors.
- Custom constraint for pneumatic cylinders.
- Custom constraint for motors, with reducers, learning mode, etc.
- Constraints can be activated and deactivated.
- Brakes and clutches, with precise stick-slip effect. , *etc.*
- Conveyors, 1-DOF elements, driveline simulation, etc.

Features

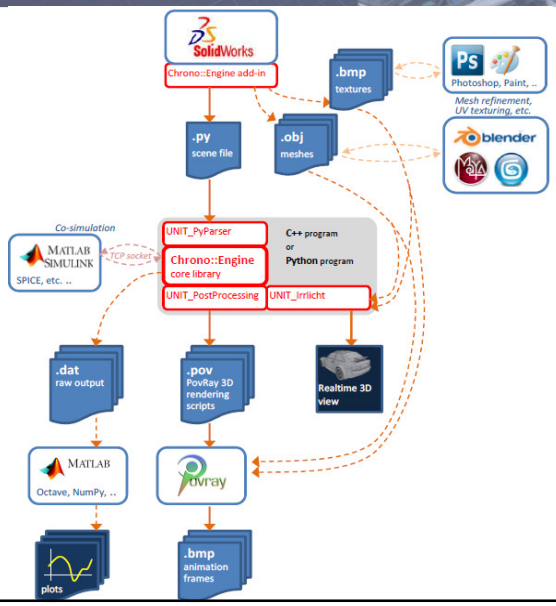
Features of the Chrono::Engine middleware (3)

Other features

- Fast collision detection between compound shapes
- Handling of redundant and ill-posed constraints.
- Integration with ‘measure differential inclusions’ approach.
- Inverse kinematics and interactive manipulation.
- Classes for genetic & local optimization.
- Classes for co-simulation
- Classes for interfacing foreign geometric data (NURBS, splines).
- The multibody engine can be scripted via Javascript and Python
- 'Probes' and 'controls' for man-in-the-loop simulations.
- Makefile system based on CMake, compatible with MS 'nmake' and GNU 'make'.
- Wide set of examples and demos,
- Powertrain 1D simulation
- Multithreading and GPU support, etc.

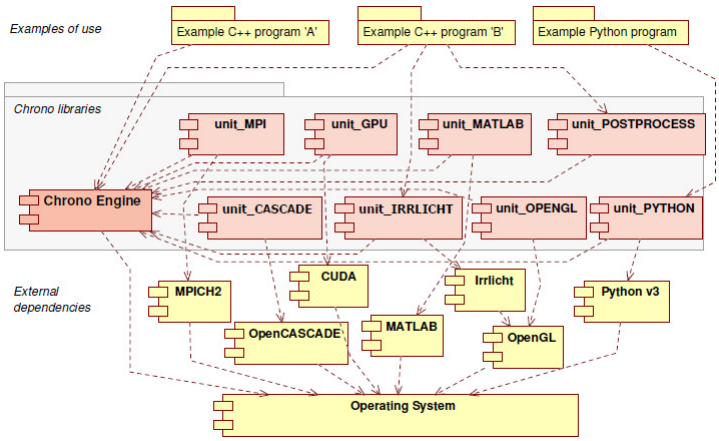
Architecture

Workflow:



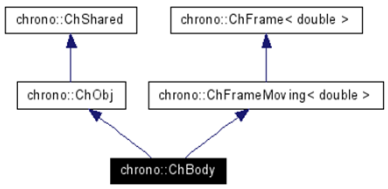
Architecture

Modules:

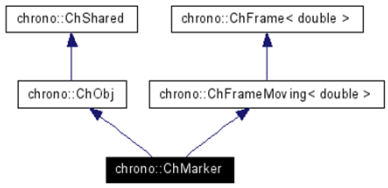


C++ class hierarchy -examples-

Rigid bodies

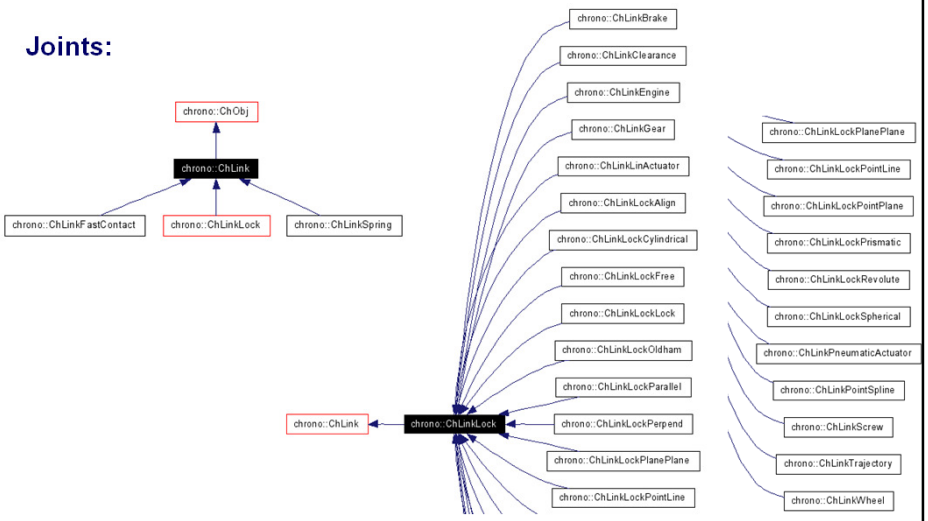


Markers and auxiliary coordinates



C++ class hierarchy -examples-

Joints:

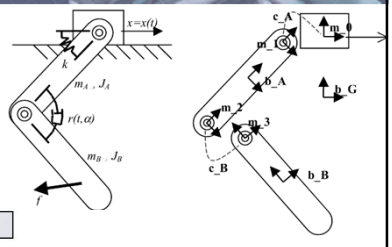


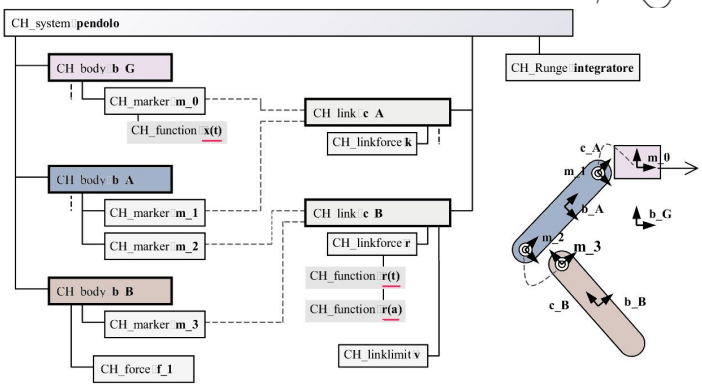
C++ class hierarchy -examples-

Some constraint types in our Chrono::Engine software

C++ transient database

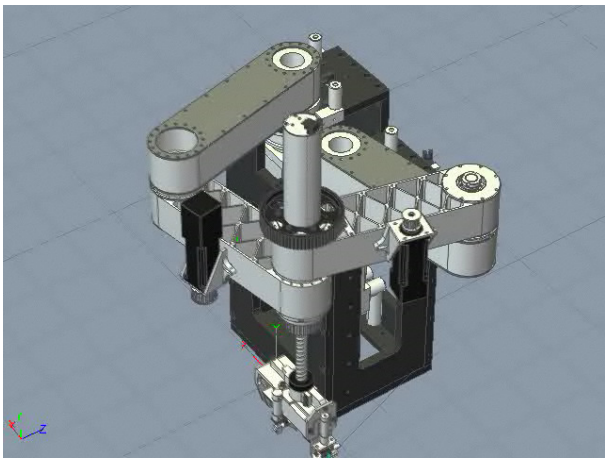
Complex object hierarchy:
smart shared pointers are used.





Multibody software

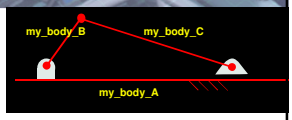
Example of simulation with our Chrono::Engine technology



The GRANIT parallel-kinematics robot (Tasora, Righettini, Chatterton, 2007)

Examples

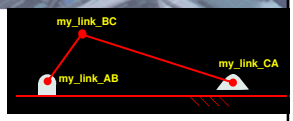
Example of Chrono::Engine C++ code (1..)




```
// 1- Create a ChronoENGINE physical system: all bodies and constraints
//    will be handled by this ChSystem object.
ChSystem my_system;

// 2- Create the rigid bodies of the slider-crank mechanical system
//   (a crank, a rod, a truss), maybe setting position/mass/inertias of
//   their center of mass (COG) etc.

// ..the truss
ChSharedBodyPtr  my_body_A(new ChBody);
my_system.AddBody(my_body_A);
my_body_A->SetBodyFixed(true);  // truss does not move!

// ..the crank
ChSharedBodyPtr  my_body_B(new ChBody);
my_system.AddBody(my_body_B);
my_body_B->SetPos(ChVector<>(1,0,0));       // position of COG of crank

// ..the rod
ChSharedBodyPtr  my_body_C(new ChBody);
my_system.AddBody(my_body_C);
my_body_C->SetPos(ChVector<>(4,0,0));       // position of COG of rod
```

Examples

Example of Chrono::Engine C++ code (..2..)




```
...
    // 3- Create constraints: the mechanical joints between the
    //    rigid bodies.

    // .. a revolute joint between crank and rod
    ChSharedPtr<ChLinkLockRevolute>  my_link_BC(new ChLinkLockRevolute);
    my_link_BC->Initialize(my_body_B, my_body_C, ChCoordsys<>(ChVector<>(2,0,0)));
    my_system.AddLink(my_link_BC);

    // .. a slider joint between rod and truss
    ChSharedPtr<ChLinkLockPointLine> my_link_CA(new ChLinkLockPointLine);
    my_link_CA->Initialize(my_body_C, my_body_A, ChCoordsys<>(ChVector<>(6,0,0)));
    my_system.AddLink(my_link_CA);

    // .. an engine between crank and truss
    ChSharedPtr<ChLinkEngine> my_link_AB(new ChLinkEngine);
    my_link_AB->Initialize(my_body_A, my_body_B, ChCoordsys<>(ChVector<>(0,0,0)));
    my_link_AB->Set_eng_mode(ChLinkEngine::ENG_MODE_SPEED);
    my_link_AB->Get_spe_funct()->Set_yconst(CH_C_PI); // speed w=3.145 rad/sec
    my_system.AddLink(my_link_AB);

...
```


Examples

Example of Chrono::Engine C++ code (..3)

```
...
    // 4- THE SOFT-REAL-TIME CYCLE, SHOWING THE SIMULATION

    // .. This will help choosing an integration step which matches the
    //    real-time step of the simulation..
    ChRealtimeStepTimer m_realtime_timer;

    while(device->run())           // cycle on simulation steps
    {
           // Redraw items (lines, circles, etc.) in
           // the 3D screen, for each simulation step
           [..]
              HERE DRAW THINGS ON THE SCREEN; FOR EXAMPLE:

           // .. draw the rod (from joint BC to joint CA)
           ChIrrTools::drawSegment(driver,
                       my_link_BC->GetMarker1()->GetAbsCoord().pos,
                       my_link_CA->GetMarker1()->GetAbsCoord().pos,
                       video::SColor(255,   0,255,0));
           [..]

           // HERE CHRONO INTEGRATION IS PERFORMED!!!:
           my_system.StepDynamics( m_realtime_timer.SuggestSimulationStep(0.02) );

    }
```

Demo_crank.exe

Demo_fourbar.exe

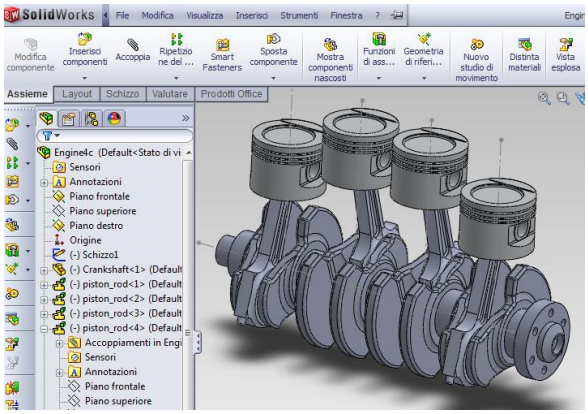
Demo_pendulum.exe

Demo_gears.exe

Chrono::SolidWorks

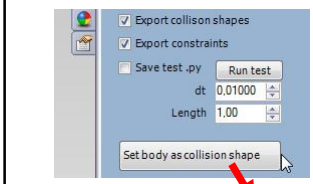
Our Chrono::SolidWorks *add-in* for CAD software:

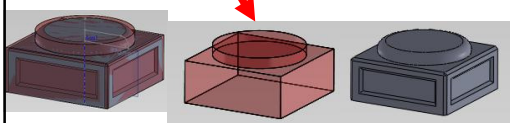
- **Expands SolidWorks with new buttons, tools**
- **Export a mechanism into a .PY file**
- **Load the system in a C++ simulator**

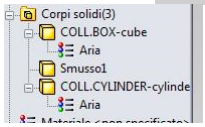


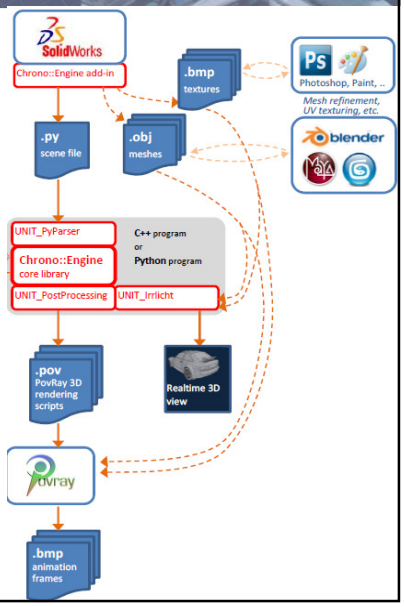
Chrono::SolidWorks

Our Chrono::SolidWorks *add-in:*



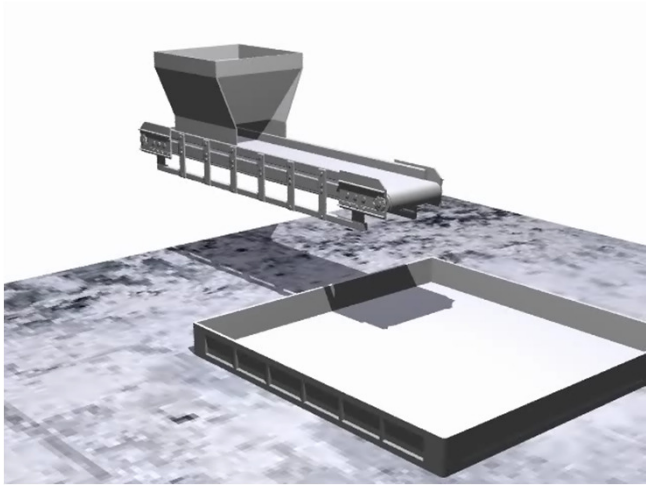






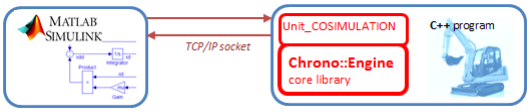
Chrono::SolidWorks

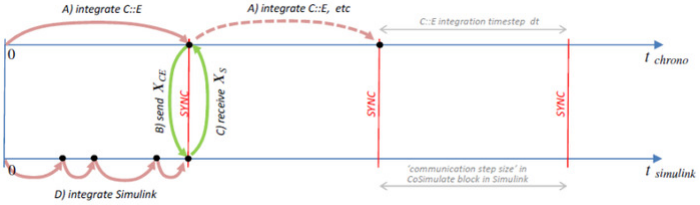
Our Chrono::SolidWorks *add-in:*



Cosimulation unit

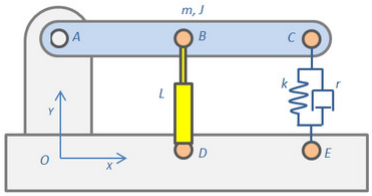
The unit_COSIMULATION:

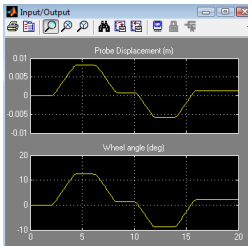


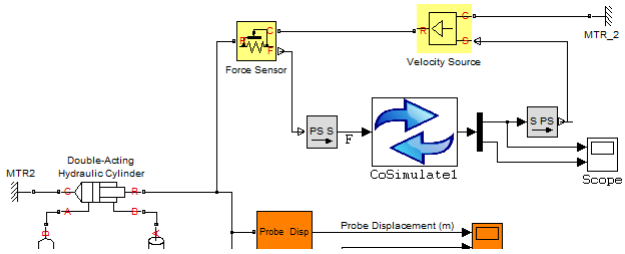


Cosimulation unit

The unit_COSIMULATION:





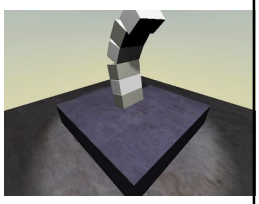


Python unit

The unit_PYTHON

- Generates a Python parser to use .py files in C++ programs
- Generates Python modules for using Chrono::Engine from Python:

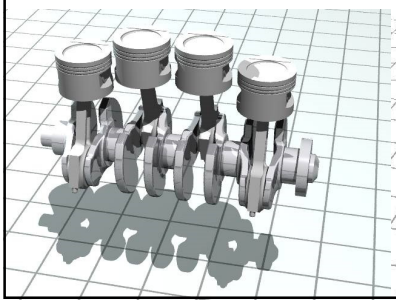
```
my_quat = chrono.ChQuaternionD(1,2,3,4)
my_qconjugate = ~my_quat
print ('quat. conjugate  =', my_qconjugate)
print ('quat. dot product=', my_qconjugate ^ my_quat)
print ('quat. product=',     my_qconjugate % my_quat)
ma = chrono.ChMatrixDynamicD(4,4)
ma.FillDiag(-2)
mb = chrono.ChMatrixDynamicD(4,4)
mb.FillElem(10)
mc = (ma-mb)*0.1;   # operator overloading of +,-,* is supported
print (mc);
mr = chrono.ChMatrix33D()
mr.FillDiag(20)
print  (mr*my_vect1);
...
```

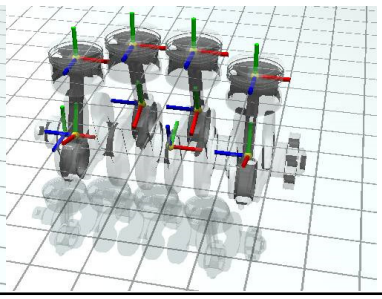


Postprocessing unit

The unit_POSTPROCESSING:

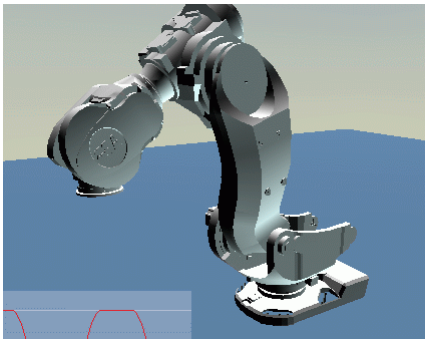
- Based on ChAsset classes (interface agnostic)
- For batch processing in:
  - POVray
  - *planned*: VTK
  - ...





Other units...

- **Unit_CASCADE**
- **Unit_POSTPROCESSING**
- **Unit_MATLAB**
- **Unit_FEM**
- **Unit_GPU**
- **Unit_MPI**
- **...**

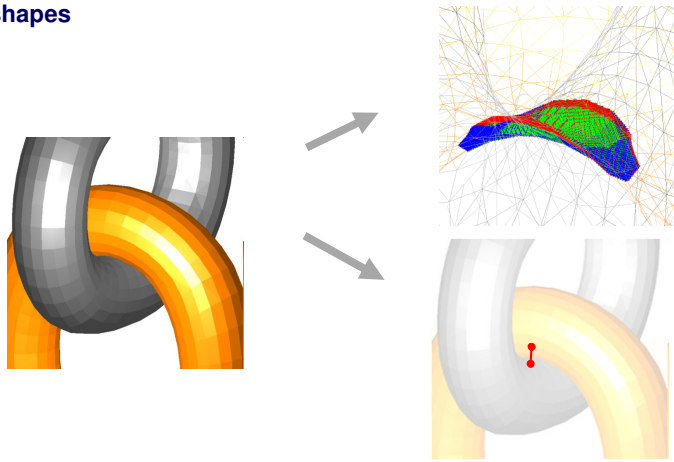


Section

Collision detection

Collision detection

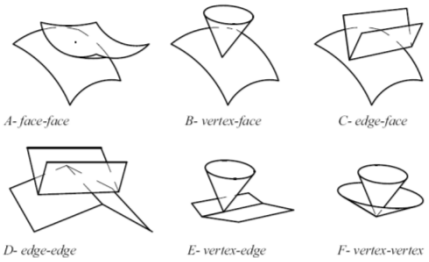
- **Still one of the hardest problems of computational geometry**
- **Approaches: find points or areas/volumes of contact between two shapes**



Collision detection

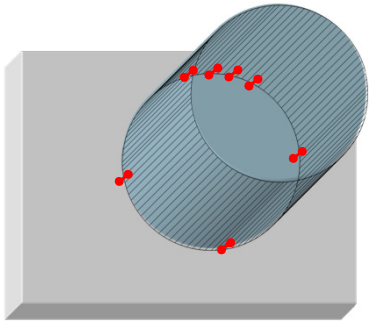
- **Approaches based on areas/volumes fit better in stiffness-based contact models, are more related to physics, but..**
- **approaches based on points are much faster!**

- **Different sub-problems depending on shape's topological entities:**



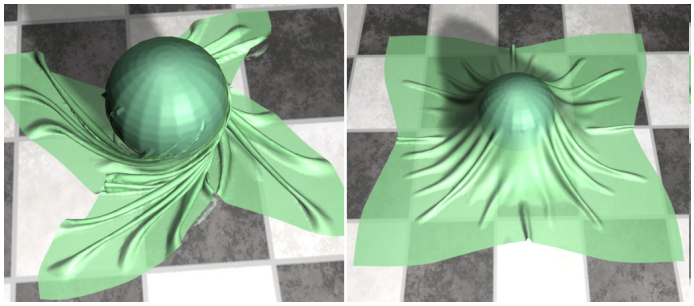
Collision detection

- **Note: point-based methods exhibit singularity problems in degenerate cases (flat surface vs. flat surface)**
  - *How many points are strictly necessary in the following case?*



Collision detection

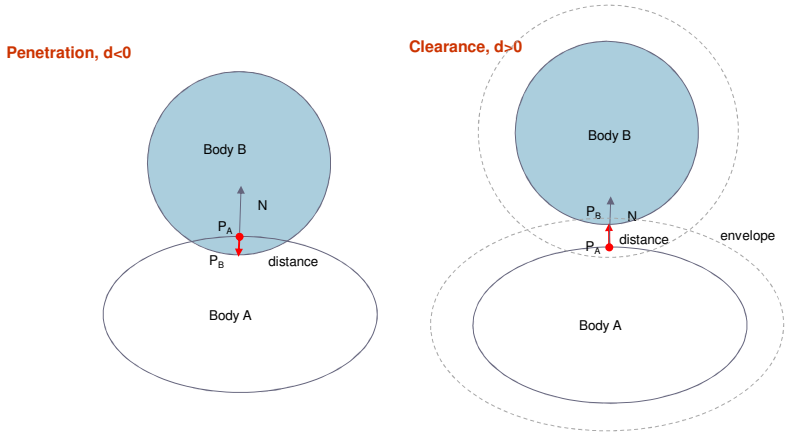
- **Both point-based methods and area/volume methods can be used for deformable models, better if with CCD (see)**



Duk-Su Kim, Jae-Pil Heo, Jaehyuk Huh, John Kim, ,and Sung-eui Yoon, 2010

Collision detection

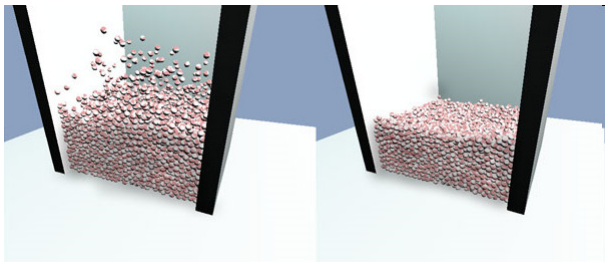
Contact distance and normal between convex shapes



Contacts are created when objects 'envelopes' start to intersect

Collision stages

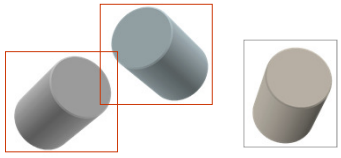
- For large N of bodies, it is not practical to check collisions between all $\frac{1}{2}N^2-N$ pairs → naïve implementation: $O(n^2)$ complexity, too much CPU time!



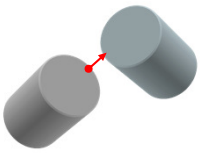
- Solution: check collision points between pairs of bodies that are 'near enough', using a preliminary trick to discard 'too far' pairs.

Collision stages

- **BROAD PHASE**
  A 'broad-phase' stage is used to roughly identify the pairs that are near enough, and to discard the pairs that are too far



- **NARROW PHASE**
  A 'narrow phase' stage is used to find exact collision points (or volumes/areas) between the pairs that comes from the broad-phase.



Collision stages

Various methods… Most famous:

BROAD PHASES

- **'SAP'**
- **Octree**
- **'DBVT' dynamic bounding boxes tree**
- **Lattice/grid domain decomposition**
- **Spatial hashing**
- **…**

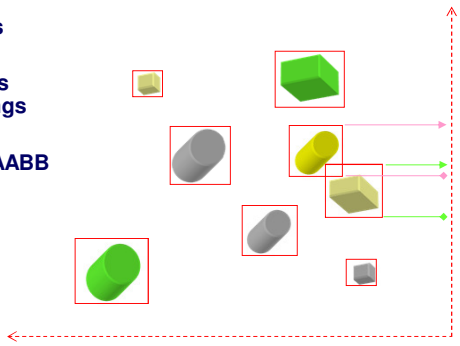
NARROW PHASES

- **Analitic solutions**
- **GJK**
- **…**

Broad phases

'SAP' broadphases

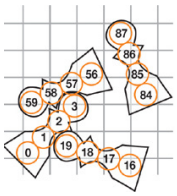
- **SAP = 'sweep-and-prune'**
- **Operates on AABB = Axis Aligned Bounding Boxes**
- **Basically, sorts X,Y,Z intervals of AABB and finds overlappings**
- **Optimization: use *quantized* AABB**
- **One of the most used and *fastest* broadphases!**
- **Not good for *deformable* objects!**

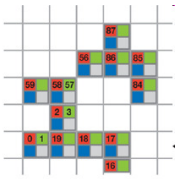


Broad phases

'Grid / lattice / bins' broadphases

- Less efficient than SAP
- More 'false positives'..
- But very simple to implement!
- Data structures are 3D arrays of pairs.
  If only not-empty cells are stored, few RAM is needed.
- Very good for very *large number* of particles
- Problem: what to do if object size is much larger or much smaller than the grid cell? → suboptimal!

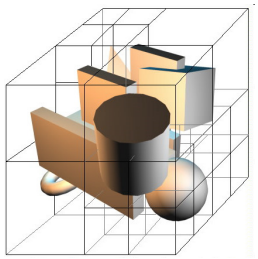


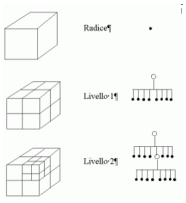


Broad phases

'Octree' broadphase
'Dynamic bounding boxes tree' broadphase

- Almost as efficient as SAP
- Fit better in case of deformable bodies
- Data structures are *trees* of pointers
- Variants: also as 'KD-trees', etc.

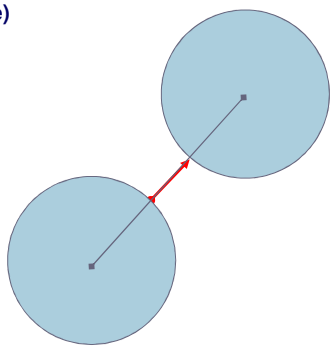




Narrow phases

Analytical solutions

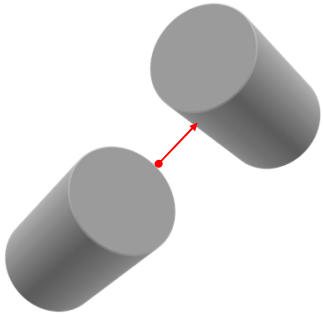
- For limited number of primitives (es: sphere vs. sphere, sphere vs. plane)
- Fastest approach, but not general
- Not always possible (es: analytical solution for ellipsoid vs. ellipsoid ?)



Narrow phases

GJK Gilbert Jordan Keerti algorithm

- For all convex shapes
- One of the most used
- Finds the minimum distance in few iterations
- Works for spheres, ellipsoids, boxes, polytopes, etc.
- Fast, robust
- Does not support penetration!

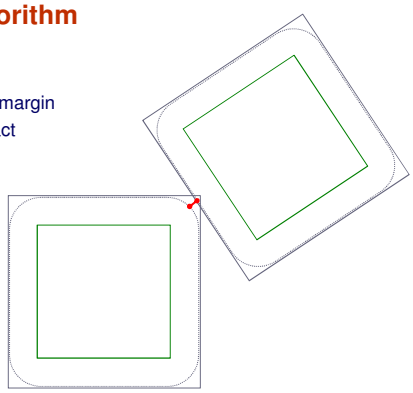


Narrow phases

GJK Gilbert Johnson Keerthi algorithm

- **Trick 1 for supporting penetration:**
  - Work on 'shrunk' objects, reduced by a margin
  - Add the margin when creating the contact

Drawback: objects are 'smoothed' a bit

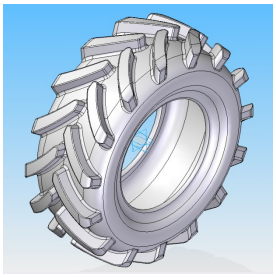


- **Trick 2 for supporting penetration:**
  - Use the EPA (Expanding Polytope Algorithm) for d<0

Drawback: slow method

Narrow phases

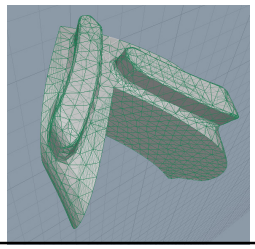
GJK Gilbert Johnson Keerthi algorithm

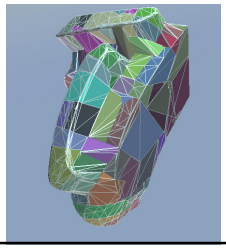


What happens in case of concave shapes?

Es. 'polygon soups', meshes..

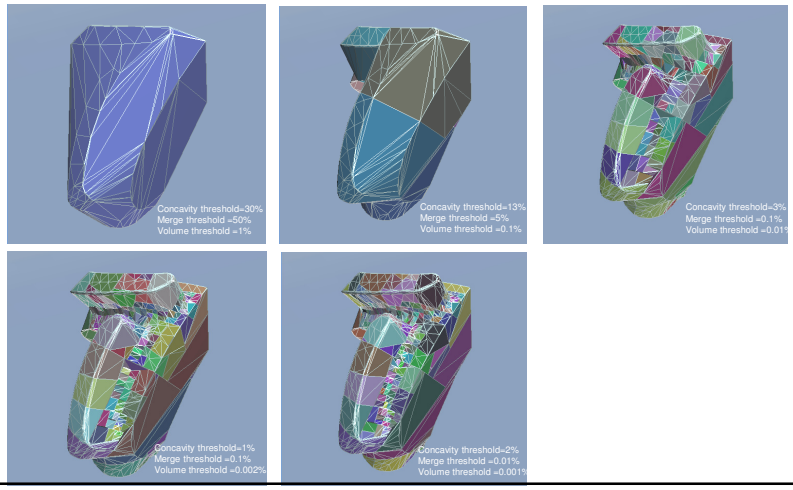
Possible algorithm: decompose concave shapes in many *convex* shapes, and process each one with GJK.





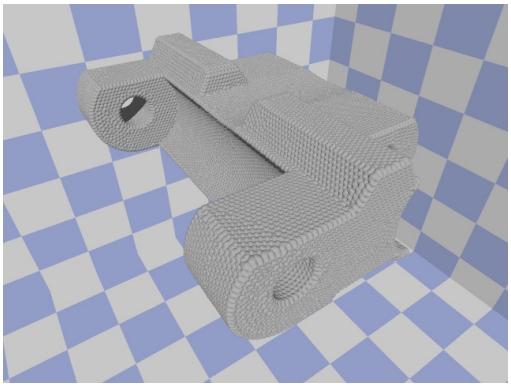
Narrow phases

Note: convex decomposition of concave shapes is not always easy…
Sometimes, results are precise but not efficient, or viceversa.



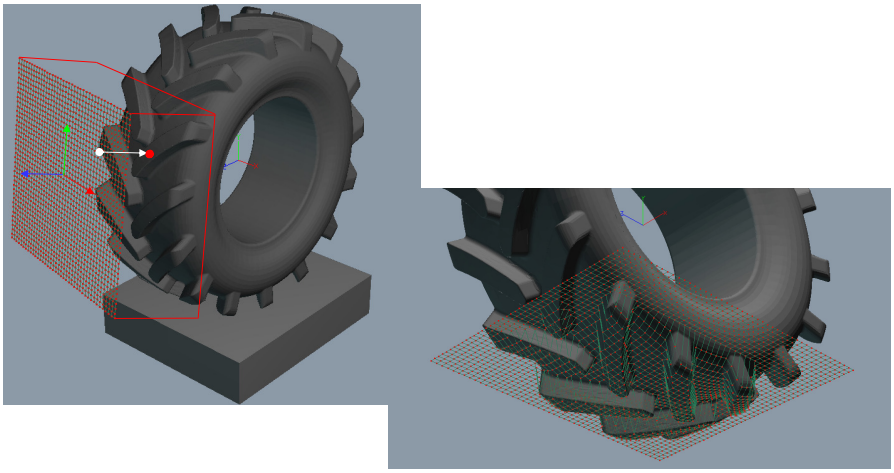
Narrow phases

Note: also spherical decomposition can be used in case of concave shapes, if RAM is not an issue…



Narrow phases

For *deformable* meshes, sometimes the concavity problem is less severe (raycasting methods, CCD methods)



Middle phase

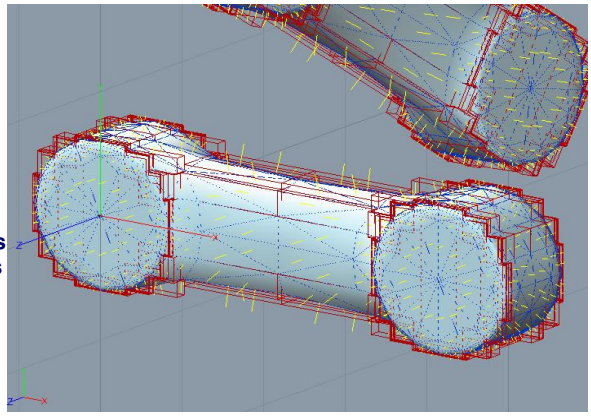
If a shape is decomposed in many sub-shapes, the narrow-phase can still hit the $O(n^2)$ problem…

Solution: use a …

Middle phase

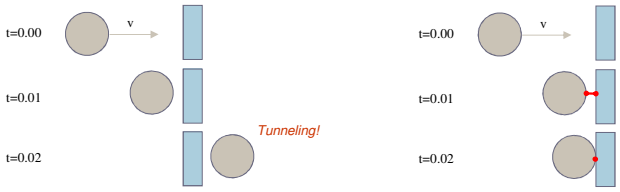
Example:

- **Uses BVh trees of AABB to manage objects with thousands of triangles or sub-convex shapes**



Continuous collision detection

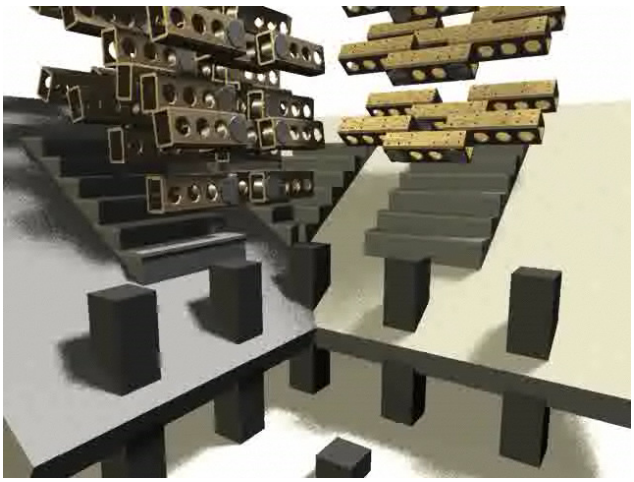
- The CCD *Continuous Collision Detection* is used for very fast objects to avoid the tunneling effect
- Few software has CCD.



- Also needed for very thin objects
- Often, it is a GJK algorithm on Minkowski sums of shapes

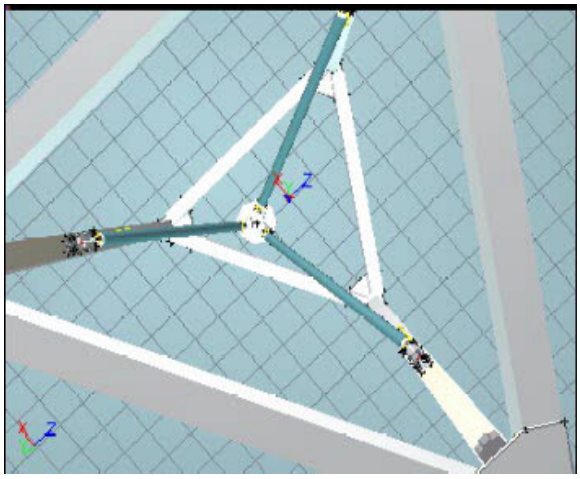
Examples

Contacts with friction



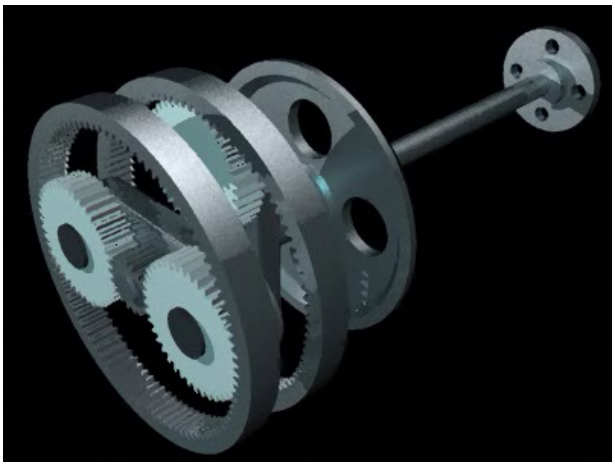
Examples

Simulation of the pneumatic-actuated TORX parallel robot



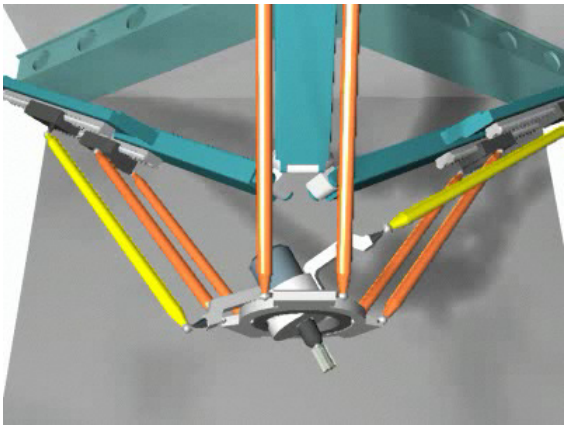
Examples

Simulation of a two stage epicycloidal reducer



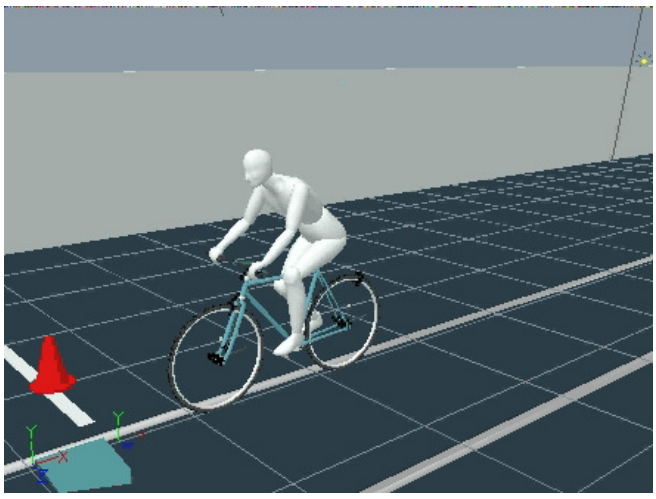
Examples

Simulation of a parallel robot for wood milling ('tenoning machine')



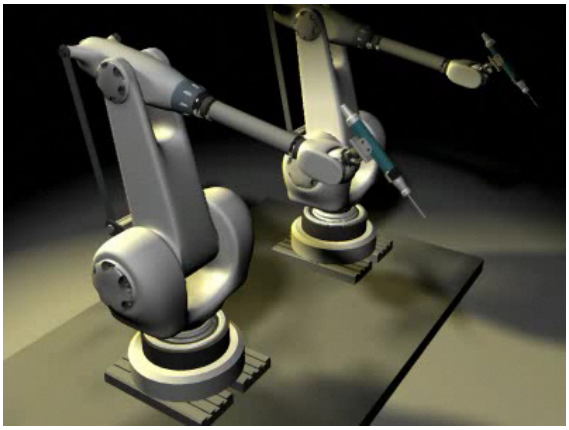
Examples

Multibody simulation of a bike on uneven terrain



Examples

Robotics: simulation of 6-DOF manipulators



MB software developement hints

Developing fast simulation middleware is difficult. Our guidelines:

- **Avoid large classes: divide and specialize as much as possible**
- **Operator overloading may cause unwanted temporary data on stack**
- **Avoid frequent allocation/deallocation on heap**
- **Do not waste RAM: exploit cache coherency**
- **Exploit SIMD processing of new processors**
- **Use virtual member methods only when needed**
- **Use templates and metaprogramming**

…

MB software developement hints

…

- Avoid unneeded computations, defer 'may-be-useful' data updating
- Avoid old 'C-style' tricks (#defines, varargs, static vars, etc.)
- If you often use 'switch{..}', most likely you are not a C++ expert…
- Do not reinvent the wheel: use STL for containers (but not 100% efficient!)
- Prefer constant-time or linear-time algorithms
- Avoid exponential or NP-hard algorithms!
- Use Hash tables for sublinear key-value searches

MB software developement hints

…

- Avoid platform dependent code (or segregate it)
- Use namespaces to avoid name pollution
- Refactor your code once in a while, for readability
- Automatic API documentation via Doxygen
- Comment your code
- Use profiler, use memory debugger
- Pass args by reference or pointer:

```
myfunct(obj& shaft) vs. myfunct(obj shaft)
```

Demo_suspension.exe

Demo_oscillator.exe

Demo_racing.exe

Section

Examples and applications

Example – Forklift truck

The forklift truck simulator benchmark

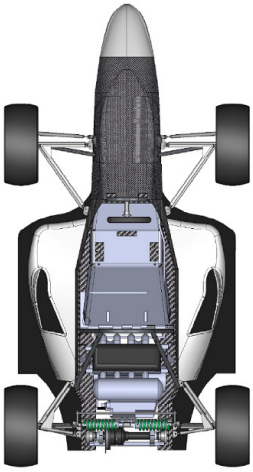
Up to **1600** forklift trucks simulated simultaneously

Demo_forklift.exe

Demo_forklift100.exe

Example – SAE Formula car real-time simulator

Multibody simulation of the PR43100 racing car (SAE Formula) for optimal design



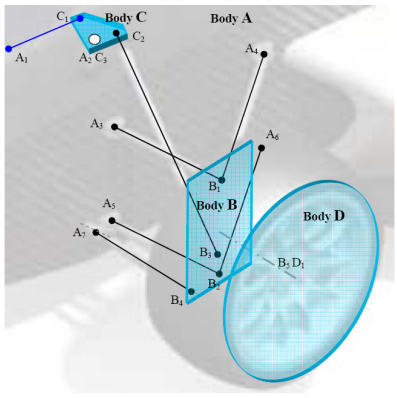
- Light alloy suspensions
- Suzuki Racing engine with EFI control
- Honeycomb carbon frame (a first in Italian SAE)
- Optimized push-rod / coilover geometry
- In collaboration with PR43100 team (M.Alfieri)

Example – SAE Formula car real-time simulator

- **Special model based on 13 rigid bodies and 43 constraints**
- **Car model with 14 DOFs (78 DOFs unconstrained)**

Bodies:

- car truss,
- front left wheel
- front left hub
- front left rocker
- front right wheel
- front right hub
- front right rocker
- rear left wheel
- rear left hub
- rear left rocker
- rear right wheel
- rear right hub
- rear right rocker



Example – SAE Formula car real-time simulator

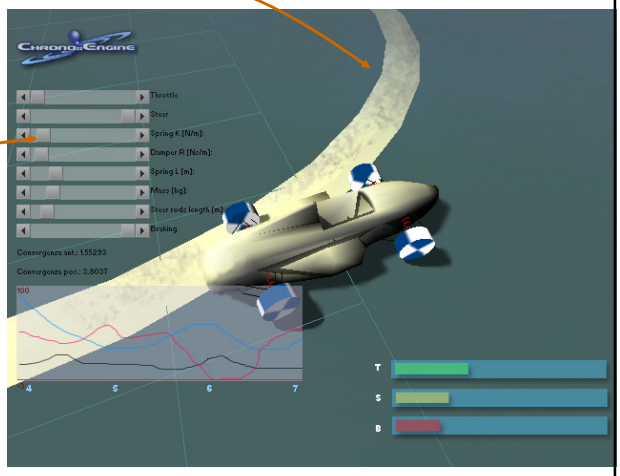
Simple 'benchmark'lane with few obstacles

User can drive with:

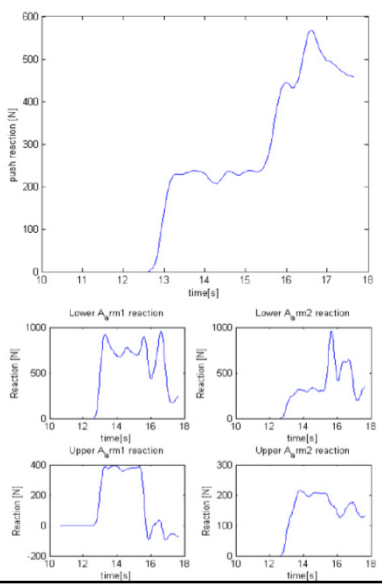
- **Throttle**
- **Steer**
- **Brake**
- **Gear**
- **Clutch**

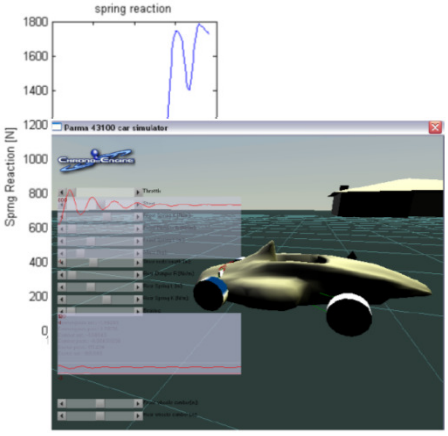
Settings:

- **Caster**
- **Camber**
- **Toe-in**
- **Weight**
- **Wheel friction**
- **Etc.**



Example – SAE Formula car real-time simulator





Example: push rod and spring forces during a simulated manouver (a curve over a small bump)

Example – SAE Formula car real-time simulator

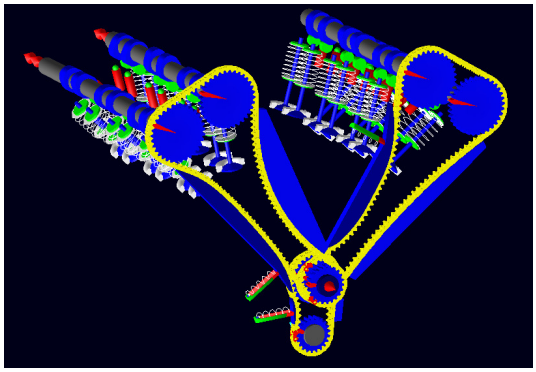
Fiorano, 2008: the PR43100 car after the competition

Example - Engines

Simulation of high performance engines

collaboration with
F.Pulvirenti et al., Ferrari Auto

Valve train & timing chain
with Adams + FEV



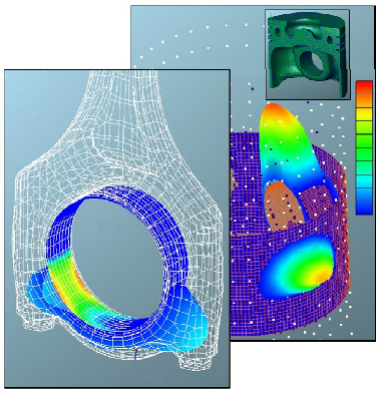
Example - Engines

Simulation of high performance engines

collaboration with
F.Pulvirenti et al., Ferrari Auto

Engine crank train, TEHD, etc.
with AVL Excite

- *wear prediction*
- *oil temperature*
- *etc.*

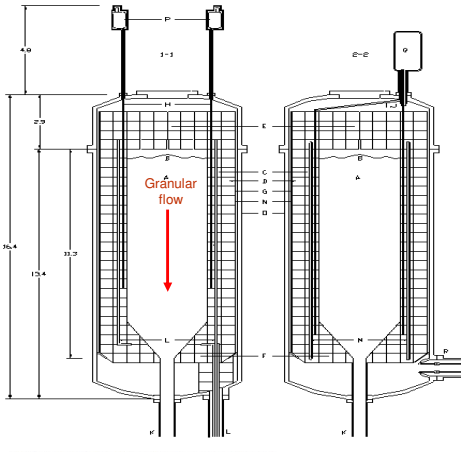


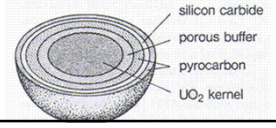
Example - Simulating the PBR nuclear reactor

The PBR nuclear reactor:

- **Fourth generation design**
- **Inherently safe, by Doppler broadening of fission cross section**
- **Helium cooled > 1000 °C**
- **Can crack water (mass production of hydrogen)**
- **Continuous cycling of 360'000 graphite spheres in a pebble bed**

Research in collaboration with M.Anitescu, Argonne National Laboratories, USA





Example - Simulating the PBR nuclear reactor

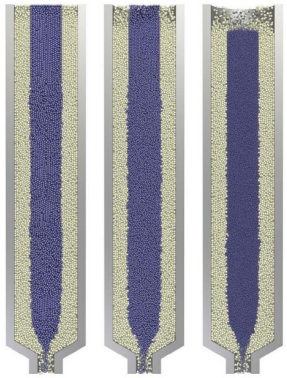
The 360'000 spheres have different radii, % of actinides, etc.

Most important: central spheres should have less Uranium/Thorium.

Problem of bidisperse granular flow with dense packing.

One of the most difficult problems in computational dynamics.

Previous attempts: DEM methods on supercomputers at Sandia Labs (but introducing compliance!)



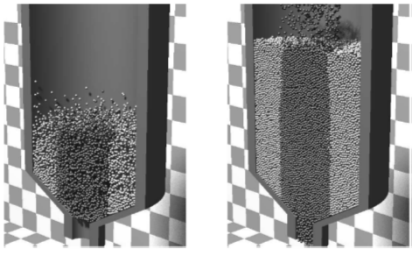
Simulations with DEM. Bazant et al. (MIT and Sandia laboratories).

Example - Simulating the PBR nuclear reactor

Our method (2007) can simulate systems with one million of frictional contacts:

- ***with rigid bodies (no fake springs-dashpots)***

- ***requires one day on a PC where a supercomputer required a week.***



Example - Simulating the PBR nuclear reactor

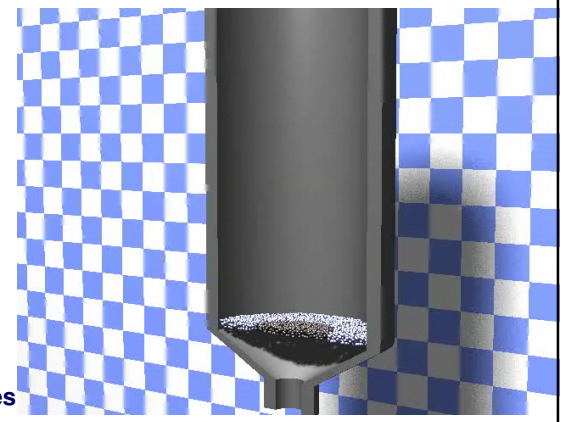
Recent test (2008) for reactor refueling cycle

180'000 Uranium-Graphite spheres

700'000 contacts on average

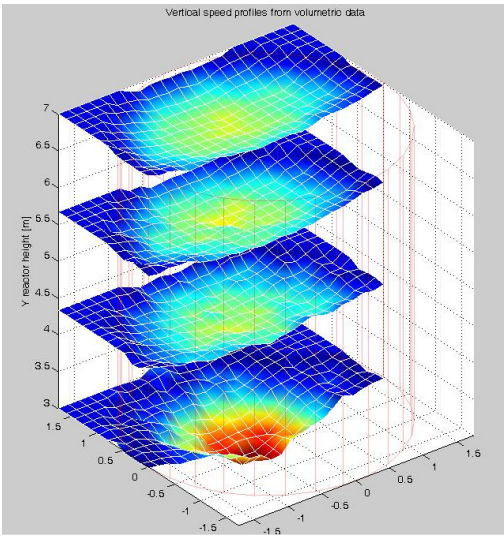
More than two millions of complementarity equations

Two millions of primal variables ten millions of dual variables



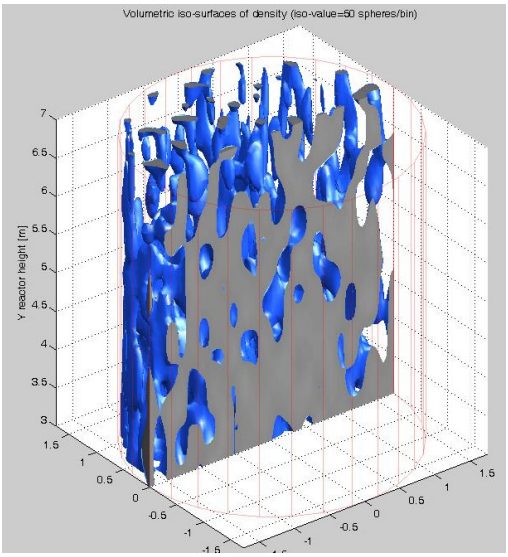
Example - Simulating the PBR nuclear reactor

Example of results



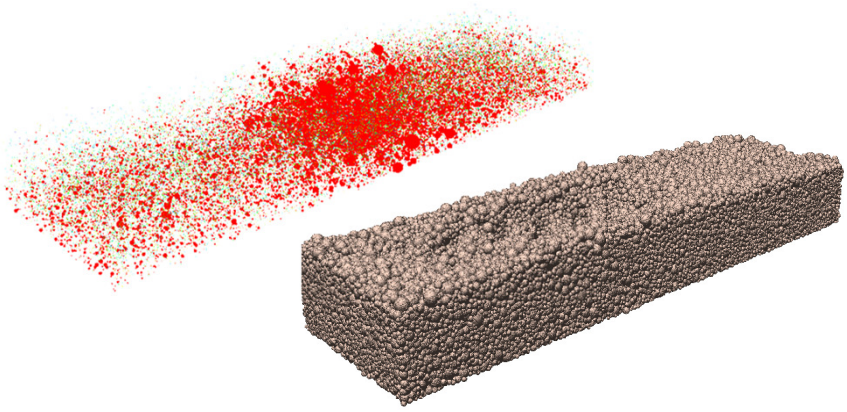
Example - Simulating the PBR nuclear reactor

Example of results



Vehicles on granular soil

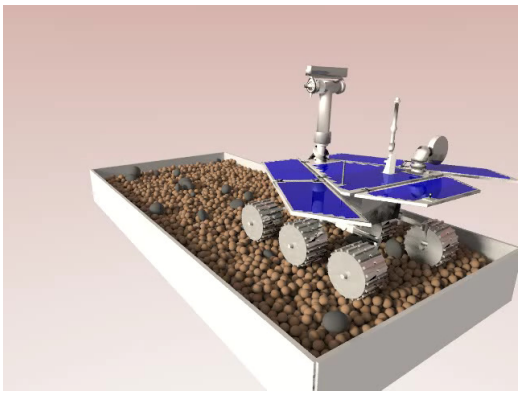
Tire on a granular soil, with CHRONO::ENGINE



In collaboration with F.Braghin, Politecnico di Milano.

Vehicles on granular soil

The Mars rover on a granular soil, with CHRONO::ENGINE

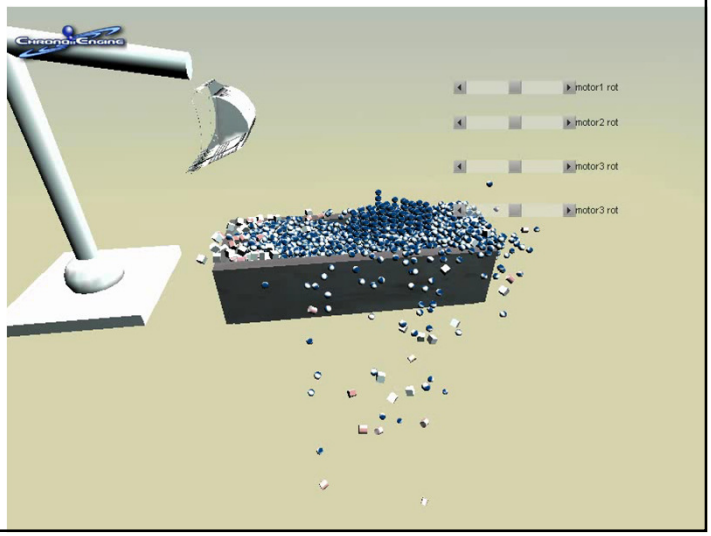


In collaboration with D.Negrut (USA) and SBEL labs [test]

Vehicles on granular soil

Soil-machine interaction

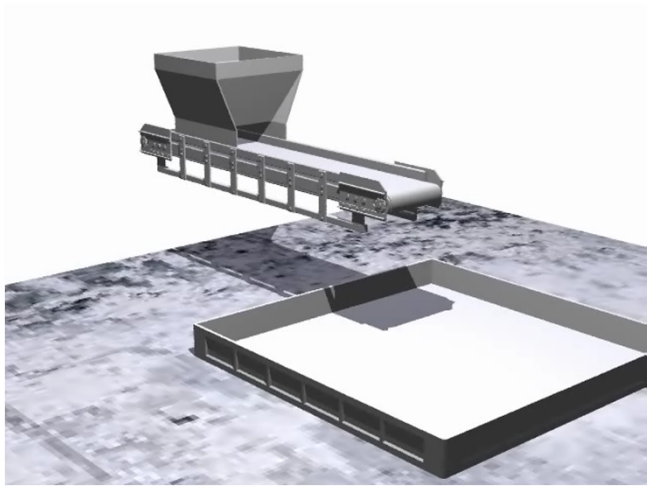
Aiming at a real-time simulation of scooping and bulldozing



(work in progress,
low quality video..)

Processing of waste material

Conveyor belts, hoppers, separating devices, ...

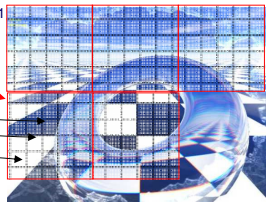


Section

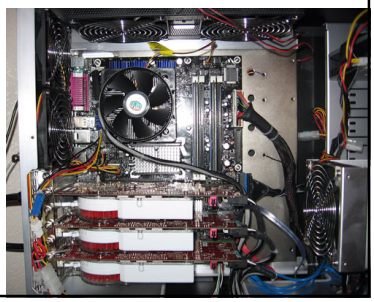
Future challenges

GPU stream supercomputing

- **GPU, Graphical Processor Units = "stream processors"[1] already used in hi-end gfx boards for pixel shading in realtime OpenGL 3D views.**
  [1] Once known as "fragment processors".
- **One GPU = cluster of *N* "stream processors"**
- **Recent GPU have *floating-point* stream processors.. Why not using them for physics?**



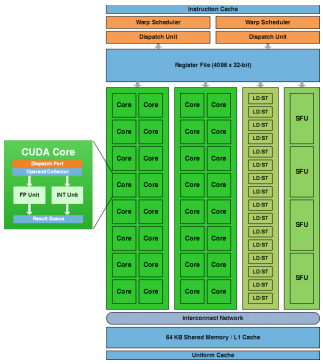
→ **Can be used for general purpose parallel computation! (GP-GPU = General Purpose GPU)**



Note: multiple GPU? Yes!
(ex: 4x256=1024 stream processors)

GPU parallel computing

Exploit GPU parallel processing

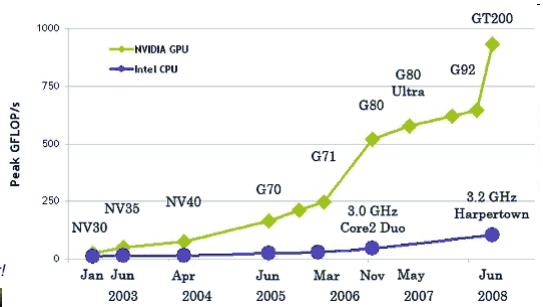


- Current NVIDIA GPU boards (Fermi, Tesla) feature hundreds of multiprocessors (cores), allowing more than **1 TFlop** on a desktop system.
- Anyway, RAM on the GPU is not unlimited: ex. **< 1'000'000** bodies on a 2GB GPU.

GPU parallel computing

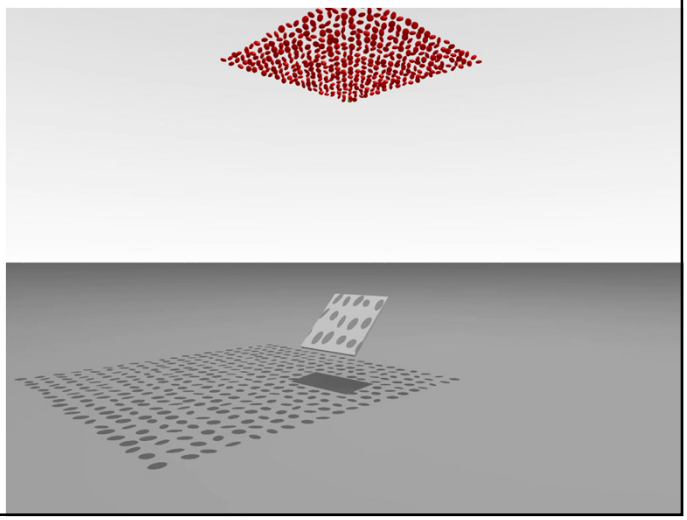
Performance: > 1 TFLOP with recent GPU processors !!!

A single TESLA™ GPU board is as powerful as the ASCI-RED 1996 supercomputer!



GPU parallel computing

Example: the M&M benchmark on a TESLA GPU



Rendered by H.Mazhar, 2011,
with Chrono::Engine 'GPU unit'

HPC high performance computing

HPC motivation: *many-body* dynamics

Examples, with *massive* number of particles:

- **Interaction between buldozzer blade and sand, debris and pebbles,**
- **Powder compaction and blending in pharmaceutical engineering,**
- **etc.**

> 1'000'000 *particles*

- *Not practical on a single CPU,*
- *better with a* ***cluster of computers***
- *Possibly, each computer fitted with* ***one or more GPU boards***

A Caterpillar D11 bulldozer

A Stable Micro System powder-flow tester

Heterogeneous parallelism

A solution for very large multibody problems:

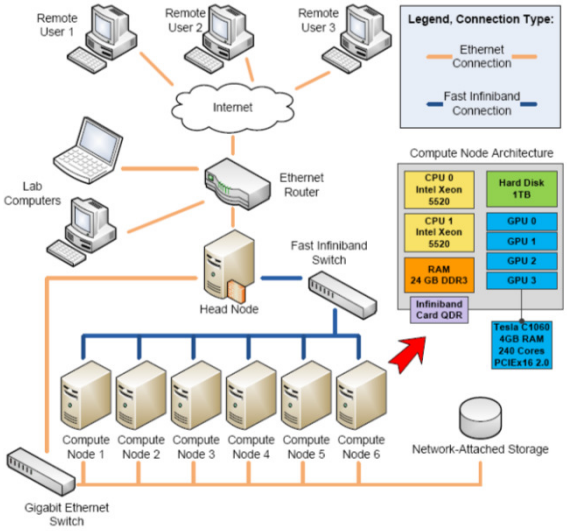
- **use a cluster of computing nodes connected with Infiniband..**
- **..each computer fitted with one or multiple GPU boards**

→ ***MPI*** *is used to handle the node-level parallelism*

→ ***CUDA*** *is used to handle the GPU-level parallelism*

Heterogeneous parallelism

Our heterogeneous cluster (at University of Wisconsin, Madison SBEL labs)



Heterogeneous parallelism

- Nodes: 6+1 quad core Intel Xeon 5520 CPUs
- 24 Tesla C1060 cards
- Each Tesla GPU has 240 cores
- Infiniband Switch: QLogic 12200-BS01
- Windows Server 2008 R2 with HPC
- MPICH for message-passing

Total **5,760** cores
Peak power: **21 TFlop**

Computing topology

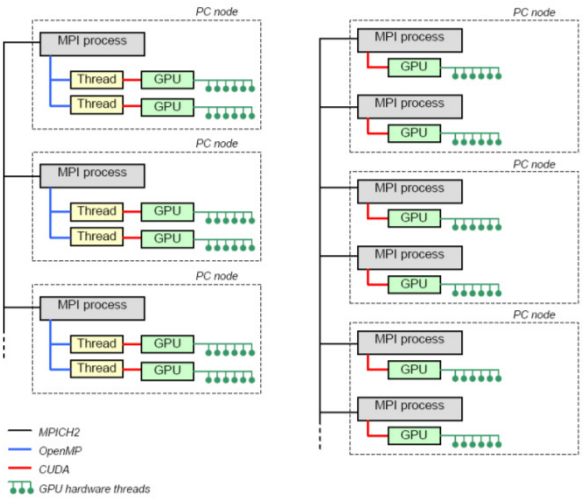
$T_c(V_c, E_c)$

Nodes= computing hardware
(CPU cores and/or GPU thread processors)

Edges= communication
(MPI messages, CUDA data flow, etc)

The computing topology must be implemented via software

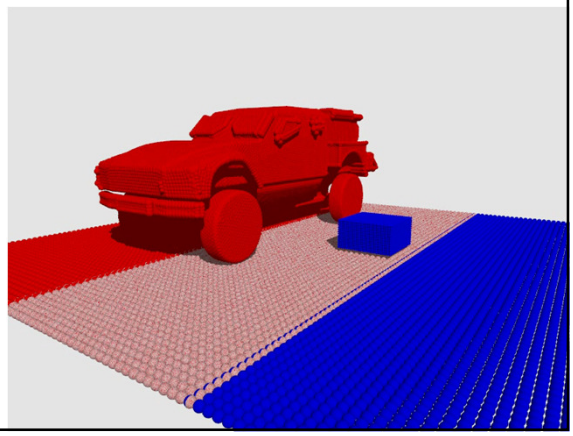
Two options shown here



MPI and domain decomposition for HPC

Example of benchmark computed on the EULER cluster

- MPICH-2 message passing interface (MPI) between the nodes
- Simple Cartesian domain decomposition



Future challenges

Future goals:

- **Comparison with DEM and validation of granular flows**
- **Faster method: multigrid (Ruge Stuben / Smooth Aggregation)**
- **Use Schwarz additive preconditioning / domain decomposition**
- **Implementation on multiple GPU**
- **Better HPC implementation on supercomputers, with MPI**
- **Beyond the 1 Million body barrier: powder/sand simulation**

Thank you for the attention!

For more informations:

tasora@ied.unipr.it
http://www.chronoengine.info
http://ied.unipr.it/tasora

Reference textbooks

- Cinematica e dinamica dei sistemi multibody, Eds. Pennestrì, Cheli, CEA, 2007 (Vol I)
- Dynamics of Multibody Systems, A.Shabana, Cambridge Press, 2005
- Dynamics of Multibody Systems, E.Robertson, R.Schwertassek, Springer, 1988
- Edward J. Haug: Computer Aided Kinematics and Dynamics of Mechanical Systems: Basic Methods (1989)
- Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, by U. Ascher and L. Petzold, SIAM, 1998
- Solving Ordinary Differential Equations I: Nonstiff Problems, by E. Hairer, S. Norsett, G. Wanner, 1993
- Solving Ordinary Differential Equations II: Stiff and differential-algebraic Problems (Second Revised Edition) by E. Hairer and G. Wanner, 2002
- The Finite Element Method, O. C. Zienkiewicz, R.L.Taylor, Butterworth-Heinemann; 6 edition (September 19, 2005) Vol I, II, III

- Dispense (A.Tasora)

Reference papers

- Anitescu, M. & Tasora, A.
  An iterative approach for cone complementarity problems for nonsmooth dynamics
  Computational Optimization and Applications, **2010**, *47(2)*, 207-235

- Tasora, A. & Anitescu, M.
  A convex complementarity approach for simulating large granular flows
  Journal of Computational and Nonlinear Dynamics, **2010**, *5*, 1-10

- Tasora, A. & Anitescu, M.
  A matrix-free cone complementarity approach for solving large-scale, nonsmooth, rigid body dynamics
  Computer Methods in Applied Mechanics and Engineering, **2010**, *doi:10.1016/j.cma.2010.06.030*

- Tasora, A.; Negrut, D. & Anitescu, M.
  Large-scale parallel multi-body dynamics with frictional contact on the graphical processing unit
  Journal of Multi-body Dynamics, **2008**, *222*, 315-326

- Heyn, T.; Mazhar, H.; Madsen, J.; Tasora, A. & Negrut, D.
  GPU-Based Parallel Collision Detection for Granular Flow Dynamics
  Proceedings of IDETC 09, San Diego, **2009**