NUMERICAL METHODS FOR LARGE SCALE NON-SMOOTH MULTIBODY PROBLEMS

Alessandro Tasora

UNIVERSITÀ DI PARMA

Research network

Radu Serban
Dan Negrut et al.

THE UNIVERSITY of WISCONSIN MADISON

SBEL

Mihai Anitescu

Argonne NATIONAL LABORATORY

THE UNIVERSITY OF CHICAGO

Alessandro Tasora,
Dario Mangoni
Simone Benatti et al.

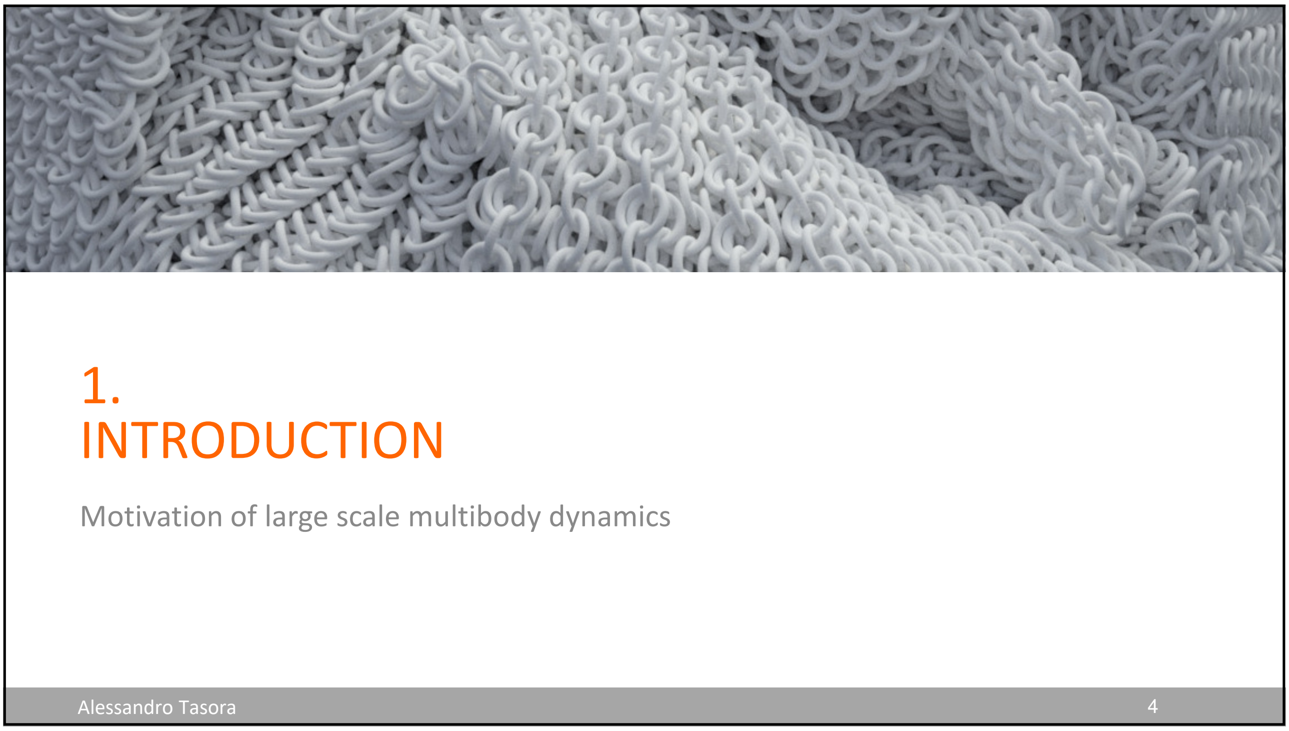
UNIVERSITÀ DI PARMA

POLITECNICO MILANO 1863

Some users / sponsors of PROJECT CHRONO

Thanks to everybody (sorry if I forgot someone)

THE UNIVERSITY WISCONSIN MADISON
SBEL
TARDEC
Argonne NATIONAL LABORATORY
Georgia Tech
National Research Council of Italy
OSHKOSH
NVIDIA
NASA
SimlabSoft
Simertis
cpu gpu DualSPHysics
Altair
Ferrari
Danfoss
Fraunhofer ITWM
FRyDoM
D-ICE ENGINEERING
intuitive MACHINES
Technische Universität Berlin
POLITECNICO MILANO 1863
UNIVERSITÀ DI PARMA
THE UNIVERSITY OF IOWA
AARHUS UNIVERSITY
AUDEMARS PIGUET Le Brassus
esa



1. INTRODUCTION

Motivation of large scale multibody dynamics

Alessandro Tasora 4

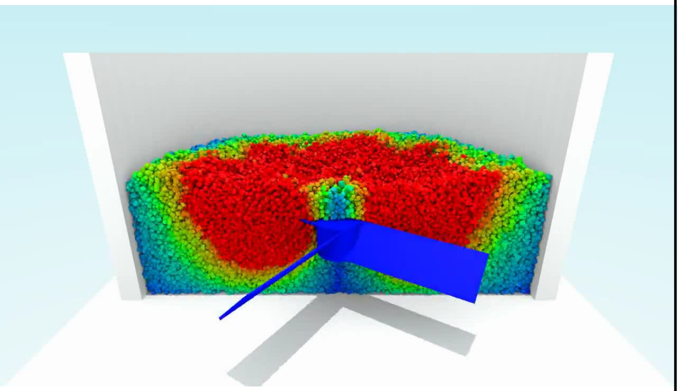
Motivations for large-scale multibody dynamics

- Robotics
- Granular flows
- Machine-ground interaction
- Architecture
- AI training, AGVs, etc

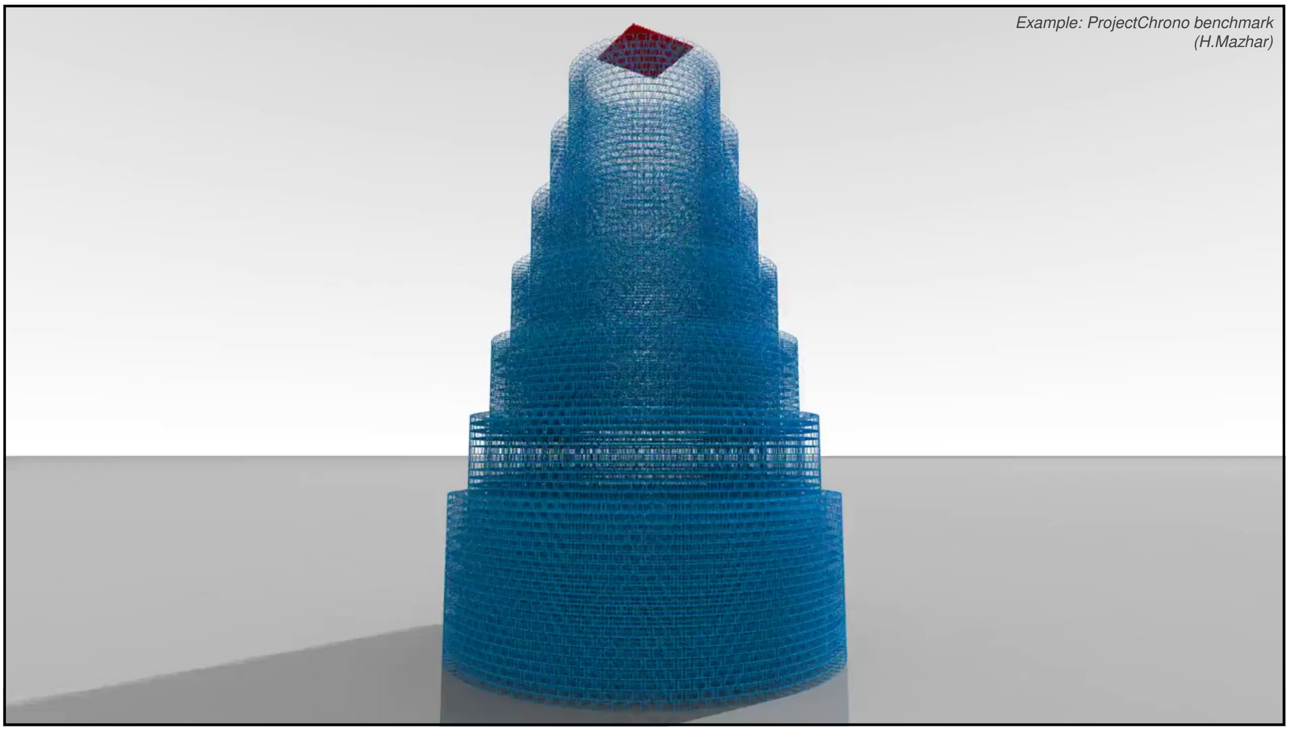
Our efforts are implemented and tested in the open source software PROJECT CHRONO

Goals

- Simulate >1M bodies
  - Need for linear memory scaling
  - Parallelizable algorithms
- Simulate >1M contacts
  - Non-smooth methods
- Simulate >1M constraints
- Stable, robust implicit integration
  - Differential-variational formulation
  - Might be used in RT/HRT, HIL, MIL
- Add finite elements
- Add fluids



ProjectChrono simulation by H.Mazhar



Example: ProjectChrono benchmark (H.Mazhar)

Structure of this lecture

Sections

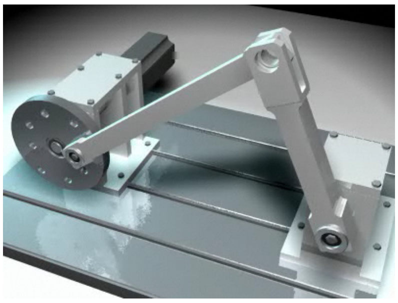
- Multibody Simulation: Concepts and applications
- Coordinate transformations
- Dynamics: Basic concepts on ODEs and DAEs
- Non-smooth Multibody Dynamics
- Collision detection
- Available software
- ProjectChrono
- Examples and applications
- Future challenges

2. MULTIBODY SIMULATION: CONCEPTS AND APPLICATIONS

Overview of multibody simulation

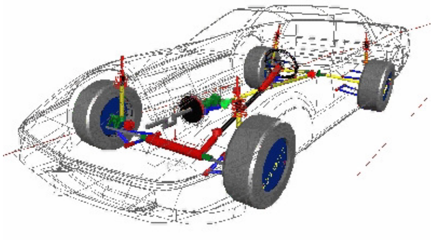
Introduction

- 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)*



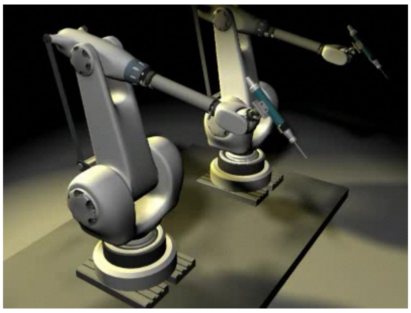
Main types of multibody analyses

- Statics
- Kinematics
  - direct
  - inverse
- Dynamics
  - Large motions
  - Linearized motion
- Modal analysis
- Sensitivity analysis
- Optimization
- …



Applications of multibody methods

- Robotics
  - Direct kinematics
  - Inverse kinematics
  - Dynamics
  - Artificial Intelligence
- Automotive
  - Powertrain dynamics
  - Handling
  - Real-time Man-In-The loop
  - Noise-Vibration-Harshness (NVH)
  - …

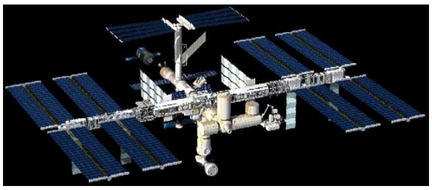


(c) Alessandro Tasora

(c) Alessandro Tasora

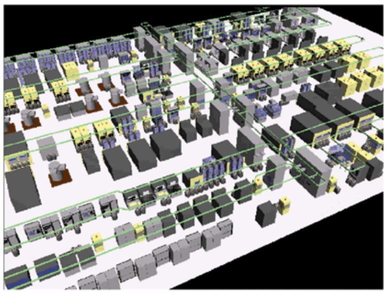
Applications of multibody methods

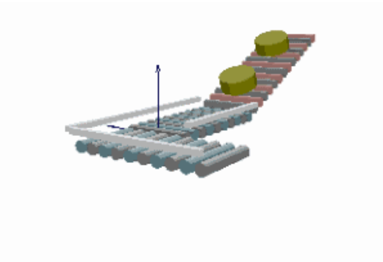
- Aerospace engineering
  - Orbital mechanics
  - Flight simulators
  - Rovers and probes
  - Simulation of complex subsystems (helicopter rotors, landing gears, etc.)
  - …

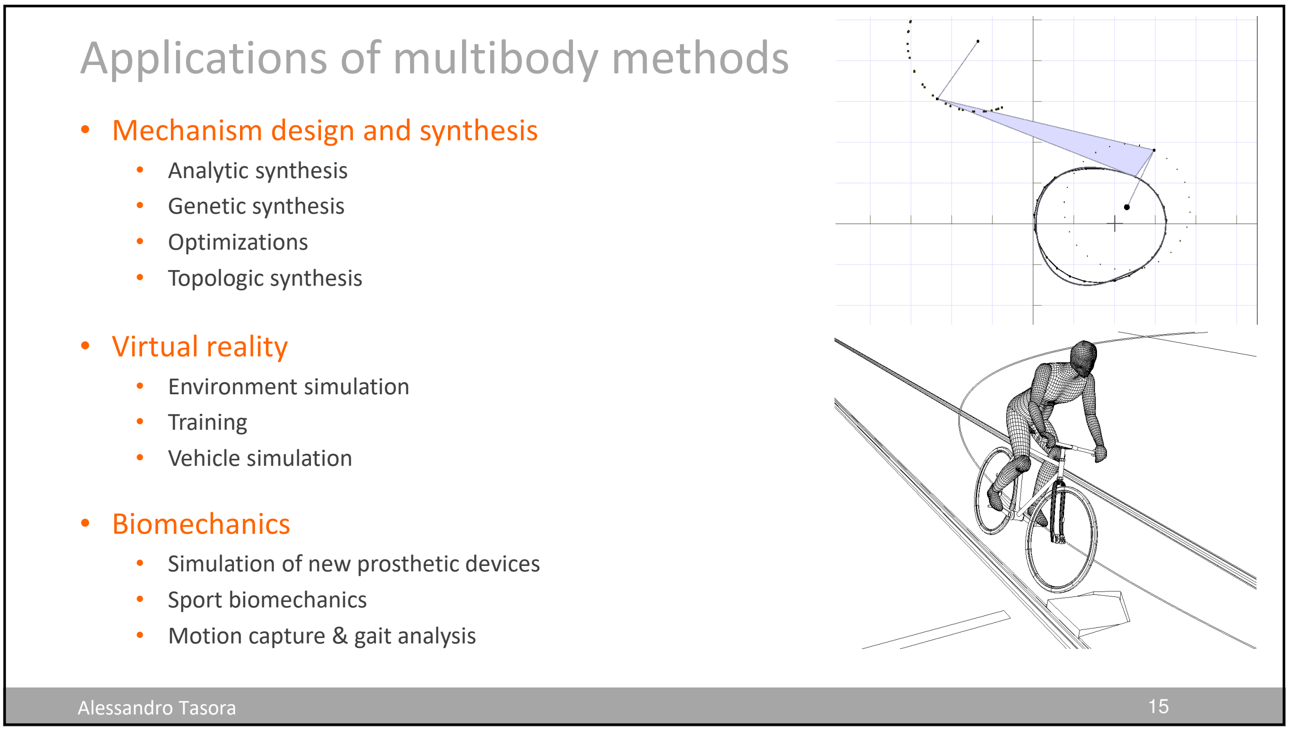


Applications of multibody methods

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

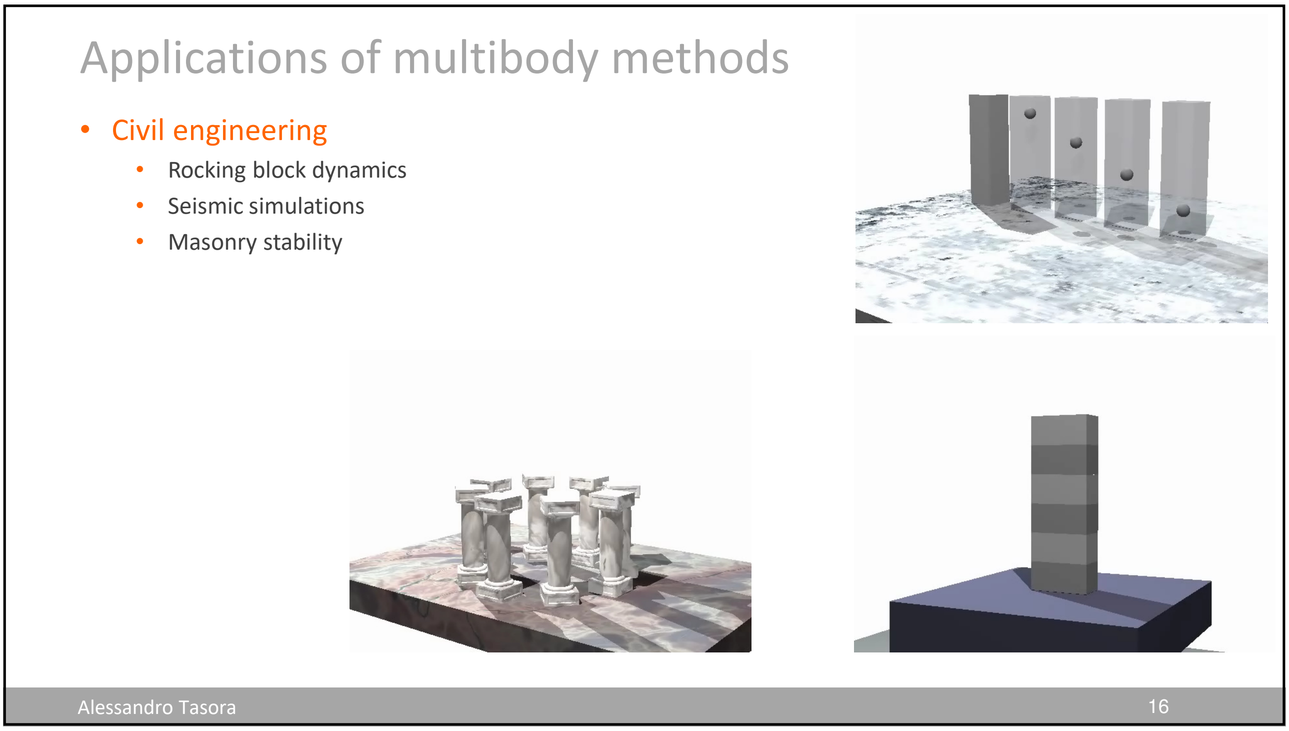






Applications of multibody methods

- Mechanism design and 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

- Special FX in movies
  - Dynamical simulations will soon replace most special effects in films
  - Skeletal animation, physical-based animation
  - Fake ragdolls, herds, masses
- Video games
  - Real-time dymamical simulation
  - *NOTE: 48'000 million of dollars of revenues in videogames,*
    A relevant market for physical simulation software.

L. Whitehouse - Houdini & Nuke,

(c) Havok

Applications of multibody methods

- Other
  - Power trains, gears,
  - Indexing devices
  - Cams & followers
  - Clock mechanisms
  - Amusement parks
  - Windmills
  - Trains
  - Toys
  - …

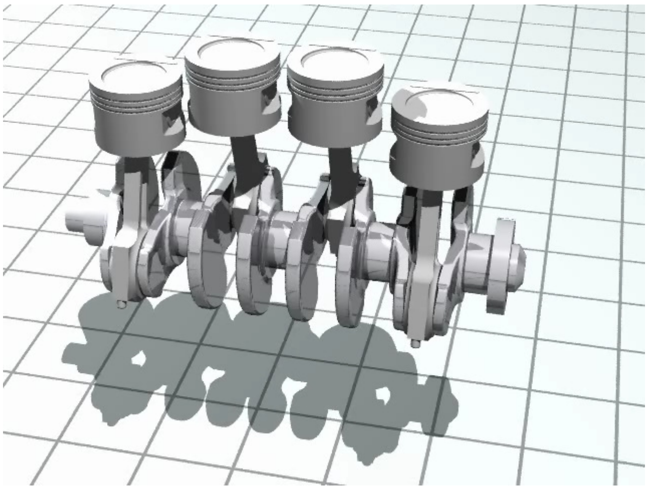
(c) Alessandro Tasora

Applications of multibody methods

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

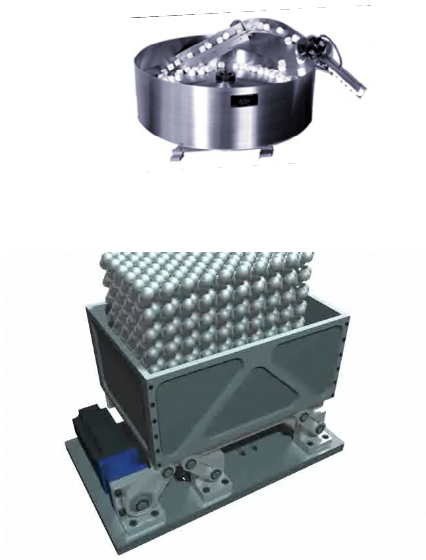
Applications of multibody methods

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.



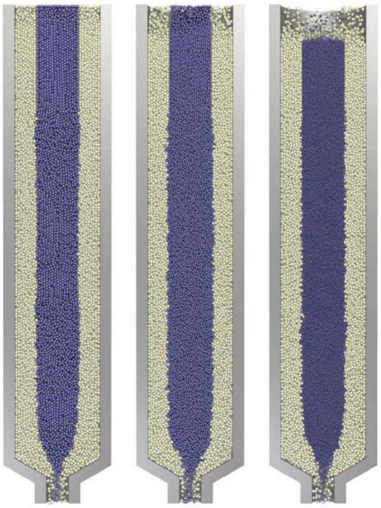
Example: size segregation device: about 2000 interacting objects simulated with our ProjectChrono software

Open problem: complexity

Example: bidisperse granular flow in the PBR nuclear reactor

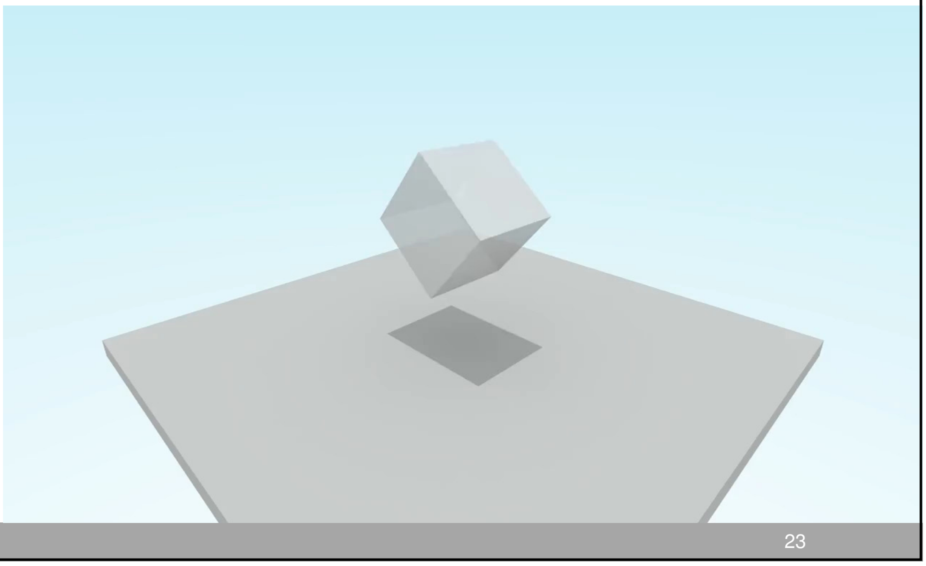
- Goal: find a numerical method which can simulate millions of rigid bodies with contacts and friction
- Collaboration with Argonne National Laboratories

  → a new method (A.Tasora,M.Anitescu)



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

Open problem: complexity



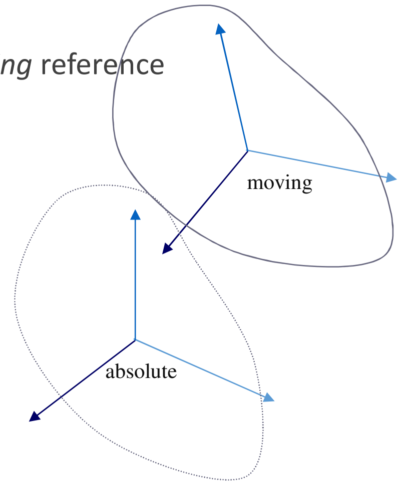
Project Chrono Benchmark (H. Mazhar)

3. COORDINATE TRANSFORMATIONS

A primer in rigid body kinematics

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

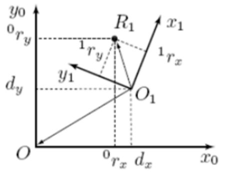
- How are body's points transformed?

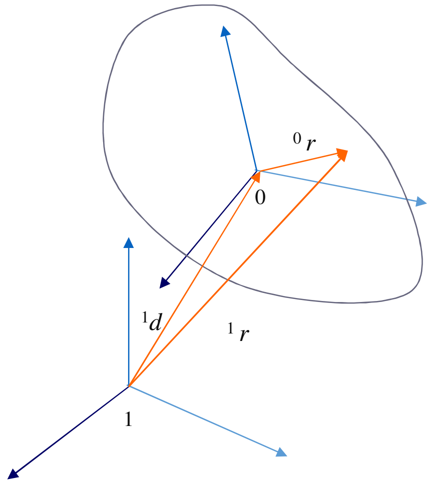
$$\{{}^0r\} = \{\ {}^0r_x \quad {}^0r_y \quad {}^0r_z\ \}^T$$

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

- *Affine linear transformation:*

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

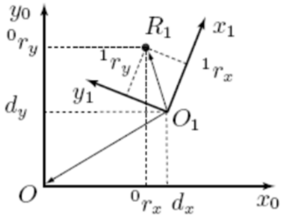




Rigid body motion

- 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)

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

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

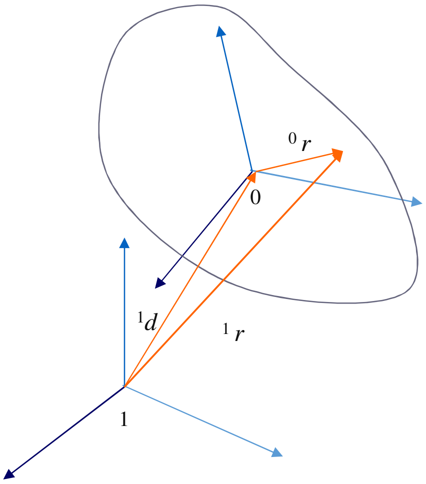
Rigid body motion

- “Direct” transformation:

$$\{^0r\} = [^0{}_1\mathrm{A}]\,\{^1r\} \; + \{^0d\}$$

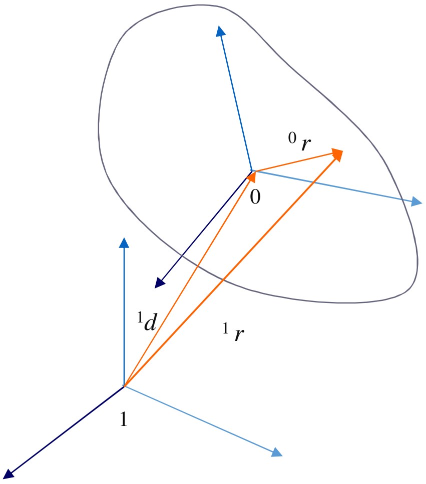
- “Inverse” transformation:

$$\begin{aligned}\{^1r\} &= [^0{}_1\mathrm{A}]^{-1}\,(\{^0r\} \; - \{^0d\}) \\ &= [^0{}_1\mathrm{A}]^{\mathrm{T}}\,(\{^0r\} \; - \{^0d\}) \\ &= [^1{}_0\mathrm{A}] \;\; (\{^0r\} \; - \{^0d\})\end{aligned}$$



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:

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

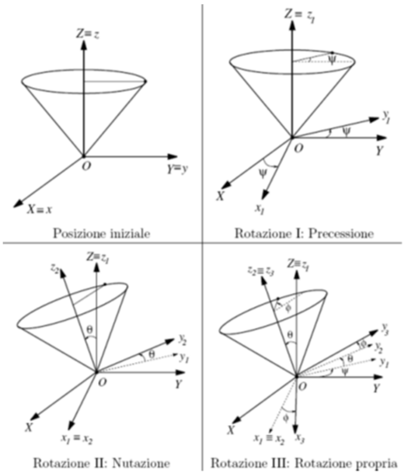
$$\{{}^{2}r\} = [{}^{2}_{1}A]\,\{{}^{1}r\} = \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \{{}^{1}r\}$$

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

$$\{{}^{3}r\} = [{}^{3}_{2}A]\,[{}^{2}_{1}A]\,[{}^{1}_{0}A]\,\{{}^{0}r\} = [{}^{3}_{0}A]\,\{{}^{0}r\}$$

Rigid body motion

- Ex: make $[{}^{0}{}_{1}\mathrm{A}]$ dependant on three 'Eulero' angles:



Posizione iniziale

Rotazione I: Precessione

Rotazione II: Nutazione

Rotazione III: Rotazione propria

- But also:
- 'Cardano' angles
- 'HPB' angles
- 'XYZ' angles,
- etc..

- See also 'Rodriguez parameters'

$$\{{}^{3}r\} = [{}^{3}_{2}A]\,[{}^{2}_{1}A]\,[{}^{1}_{0}A]\,\{{}^{0}r\} = [{}^{3}_{0}A]\,\{{}^{0}r\}$$

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 $[A]$?

η = asin (- $A_{1,3}$ / cos(ζ))

ξ = acos ($A_{2,2}$ / cos(ζ)) → singularity for $\zeta = \pi/2 + n\,\pi$!!! *(Same for all sets of 3 angles!)*

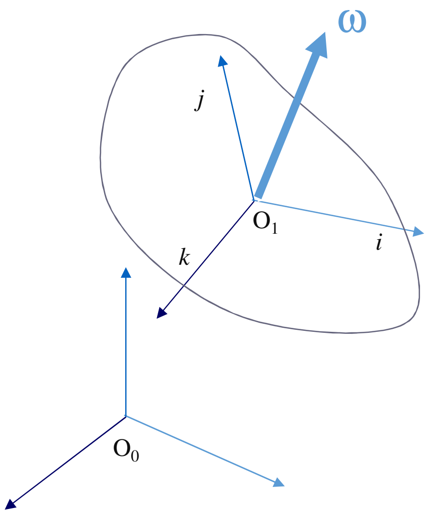
Rigid body motion

- Angular velocity

$$\left\{ \begin{array}{c} \frac{d\vec{i}_m}{dt} \\ \hdashline \frac{d\vec{j}_m}{dt} \\ \hdashline \frac{d\vec{k}_m}{dt} \end{array} \right\} = \left[\begin{array}{c:c:c} \widetilde{\omega} & 0 & 0 \\ \hdashline 0 & \widetilde{\omega} & 0 \\ \hdashline 0 & 0 & \widetilde{\omega} \end{array} \right] \left\{ \begin{array}{c} \vec{i}_m \\ \hdashline \vec{j}_m \\ \hdashline \vec{k}_m \end{array} \right\}$$

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

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



Rigid body motion

- Angular velocity, velocity

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

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

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

$$\left\{{}^{0}v\right\} = \left[{}^{0}_{i}A\right]\left\{{}^{i}v\right\}$$

$$\left\{{}^{0}r\right\} = \left[{}^{0}_{i}A\right]\left\{{}^{i}r\right\}$$

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

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

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

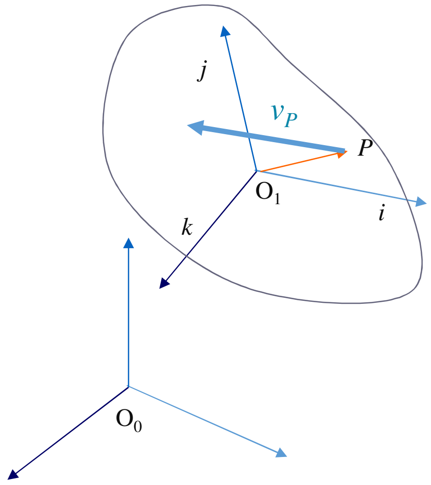
$$\left[{}^{0}_{i}\dot{A}\right] = \left[{}^{0}\tilde{\omega}\right]\left[{}^{0}_{i}A\right]$$

Alessandro Tasora

Rigid body motion

- Velocity of a point on a moving frame

$$\begin{aligned}\left\{{}^{0}v_{P}\right\} &= \left\{{}^{0}v_{O_i}\right\} + \left[{}^{0}_{i}A\right]\left[{}^{i}\tilde{\omega}\right]\left[{}^{0}_{i}A\right]^{T}\left\{{}^{0}s_{P}\right\} \\ &= \left\{{}^{0}v_{O_i}\right\} + \left[{}^{0}_{i}A\right]\left[{}^{i}\tilde{\omega}\right]\left[{}^{i}_{0}A\right]\left\{{}^{0}s_{P}\right\} \\ &= \left\{{}^{0}v_{O_i}\right\} + \left[{}^{0}_{i}A\right]\left[{}^{i}\tilde{\omega}\right]\left\{{}^{i}s_{P}\right\}\end{aligned}$$



Alessandro Tasora

Rigid body motion

- Acceleration of a point on a moving frame

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

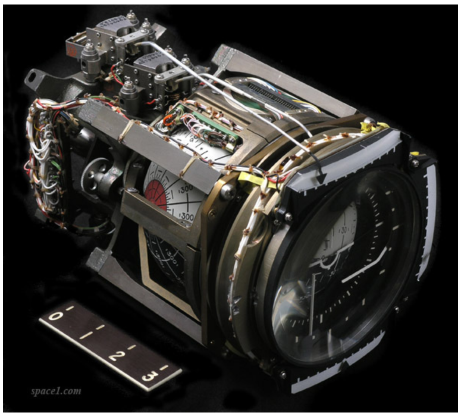
$$\begin{aligned}\left\{{}^0a_P\right\} &= \left\{{}^0a_{O_i}\right\} + \left[{}^0\widetilde{\alpha}\right]\left\{{}^0s_P\right\} + \left[{}^0\widetilde{\omega}\right]\left[{}^0\widetilde{\omega}\right]\left\{{}^0s_P\right\}\\ &= \left\{{}^0a_{O_i}\right\} + \left[{}^0\widetilde{\alpha}\right]\left\{{}^0s_P\right\}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

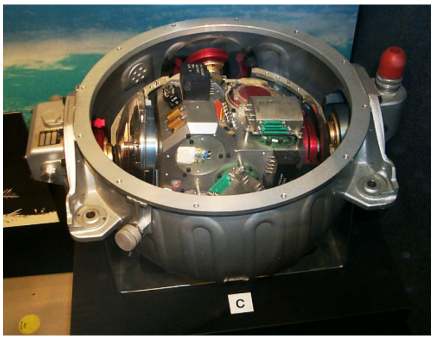
Rigid body motion

- Rotation in 3D nt as easy as in 2D…
- Problem: recovering 3 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.

Rigid body motion

- Ex. The gimbal lock problem in Apollo 11 IMUs: only 3 gimbals were not sufficient





Quaternions

- Hypercomplex 4-dimensional numbers
- Associative divisional algebra

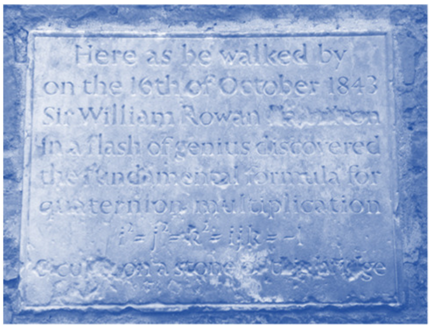
$$\mathbf{q} = e_0 + \mathbf{i} \cdot e_1 + \mathbf{j} \cdot e_2 + \mathbf{k} \cdot e_3$$

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

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

$$\|\mathbf{q}\| = \mathbf{q}^* \circ \mathbf{q} = (e_0^2 + e_1^2 + e_2^2 + e_3^2)$$

- Why quaternions for the rotations?
  - No singularities
  - Compact formalisms
  - *sin() cos()* never used
  - Easier analytic constraint jacobians $[C_q]$


Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
i² = j² = k² = ijk = -1
& cut it on a stone of this bridge

> *"Quaternions have been an unmixed evil to those who have touched them in any way"*
>
> Lord Kelvin, 1892

Quaternions

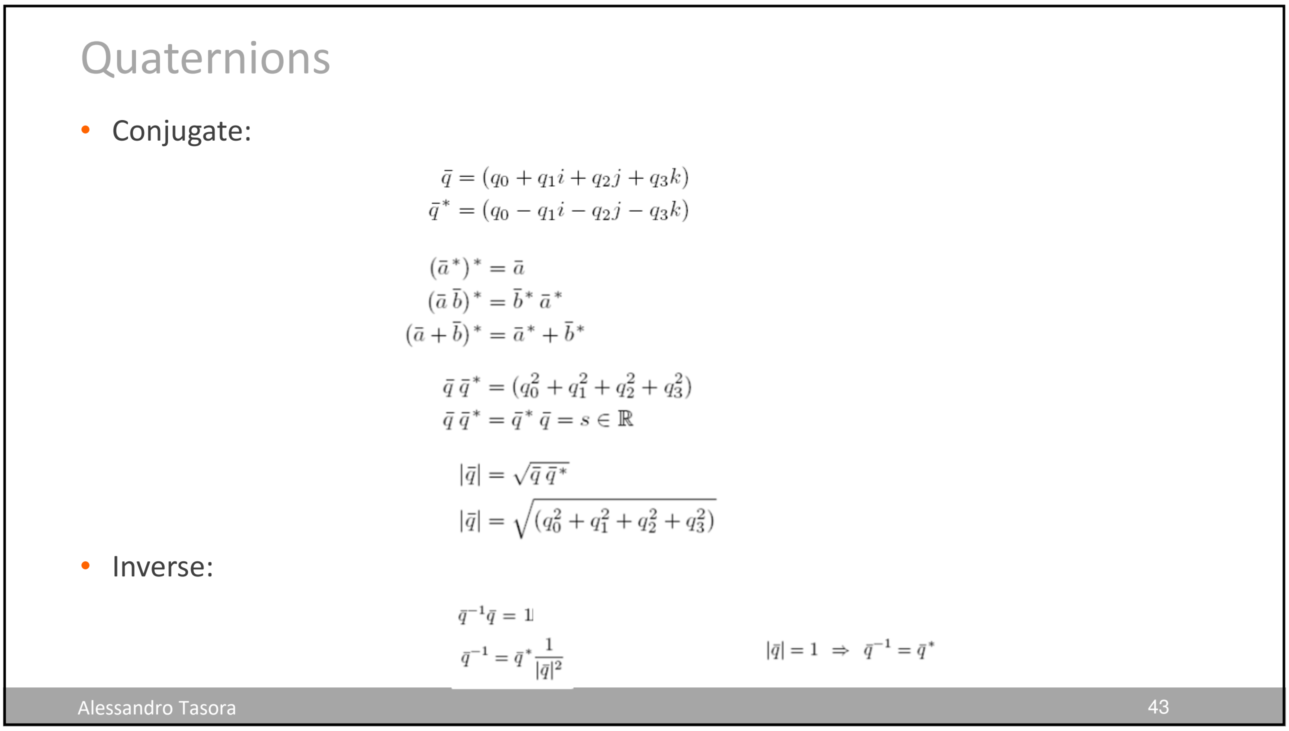
- Sum:

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

- Product:

$$\begin{aligned}\bar{c} = \bar{a} \cdot \bar{b} &:= (s_a s_b - \vec{v}_a \cdot \vec{v}_b, s_a \vec{v}_b + s_b \vec{v}_a + \vec{v}_a \times \vec{v}_b) \\ &= (a_0 + a_1 \cdot i + a_2 \cdot j + a_3 \cdot k) \cdot (b_0 + b_1 \cdot i + b_2 \cdot j + b_3 \cdot k) = \\ &= (a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3) + \\ &+ (a_0 b_1 + a_1 b_0 + a_2 b_3 - a_3 b_2) \cdot i + \\ &+ (a_0 b_2 - a_1 b_3 + a_2 b_0 + a_3 b_1) \cdot j + \\ &+ (a_0 b_3 + a_1 b_2 - a_2 b_1 + a_3 b_0) \cdot k\end{aligned}$$

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



Quaternions

- Conjugate:

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

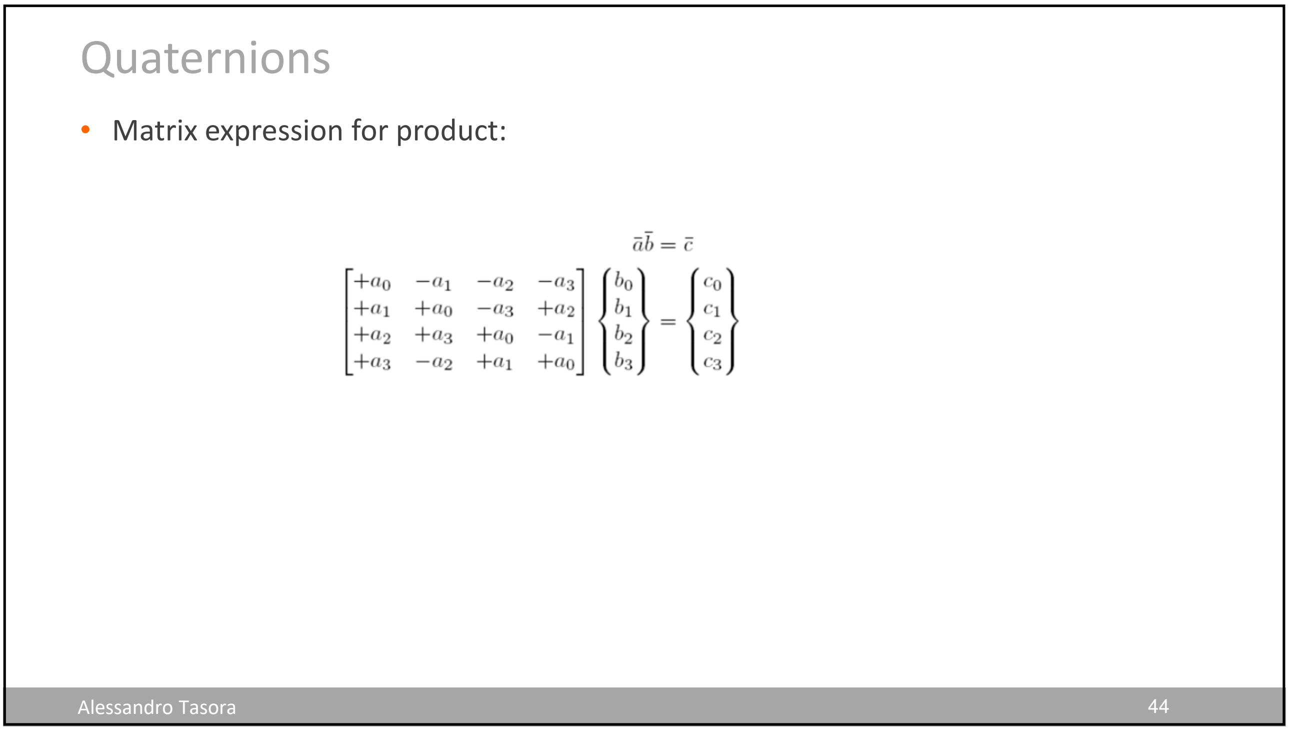
$$(\bar{a}^*)^* = \bar{a}$$
$$(\bar{a}\,\bar{b})^* = \bar{b}^*\,\bar{a}^*$$
$$(\bar{a} + \bar{b})^* = \bar{a}^* + \bar{b}^*$$

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

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

- Inverse:

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

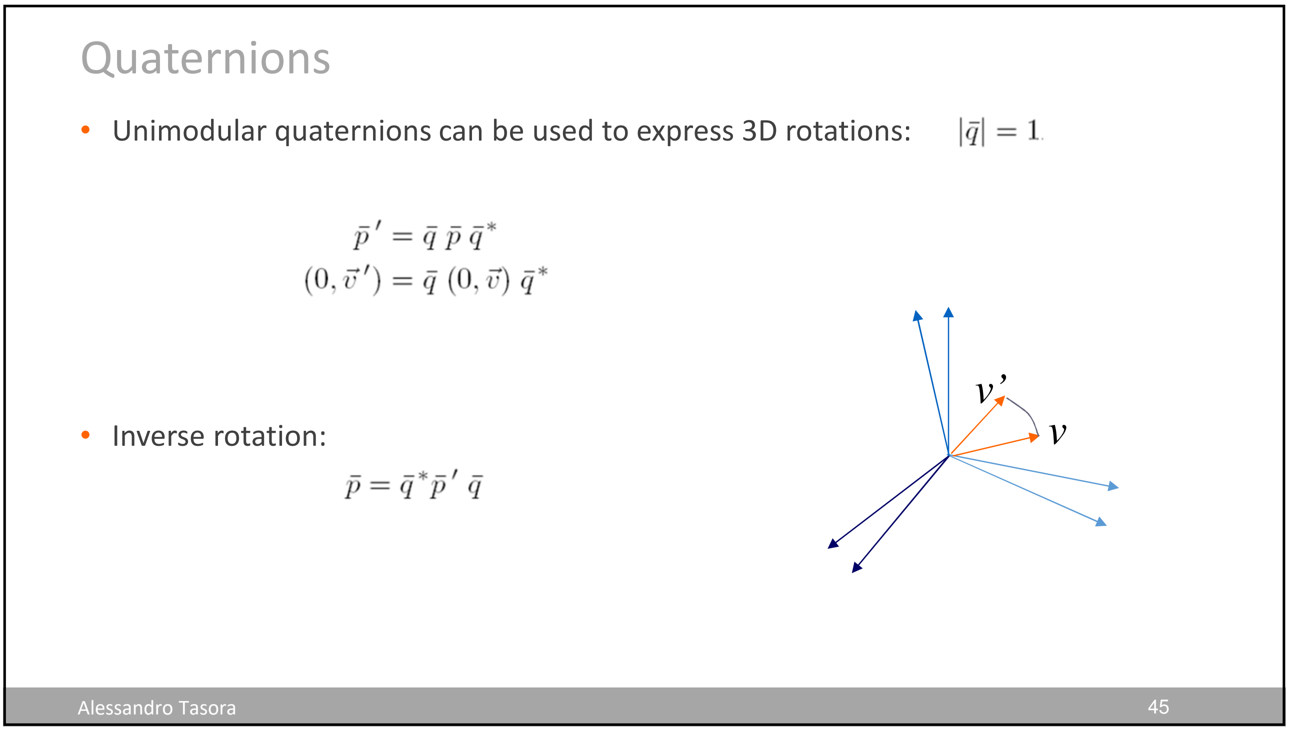


Quaternions

- Matrix expression for product:

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

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



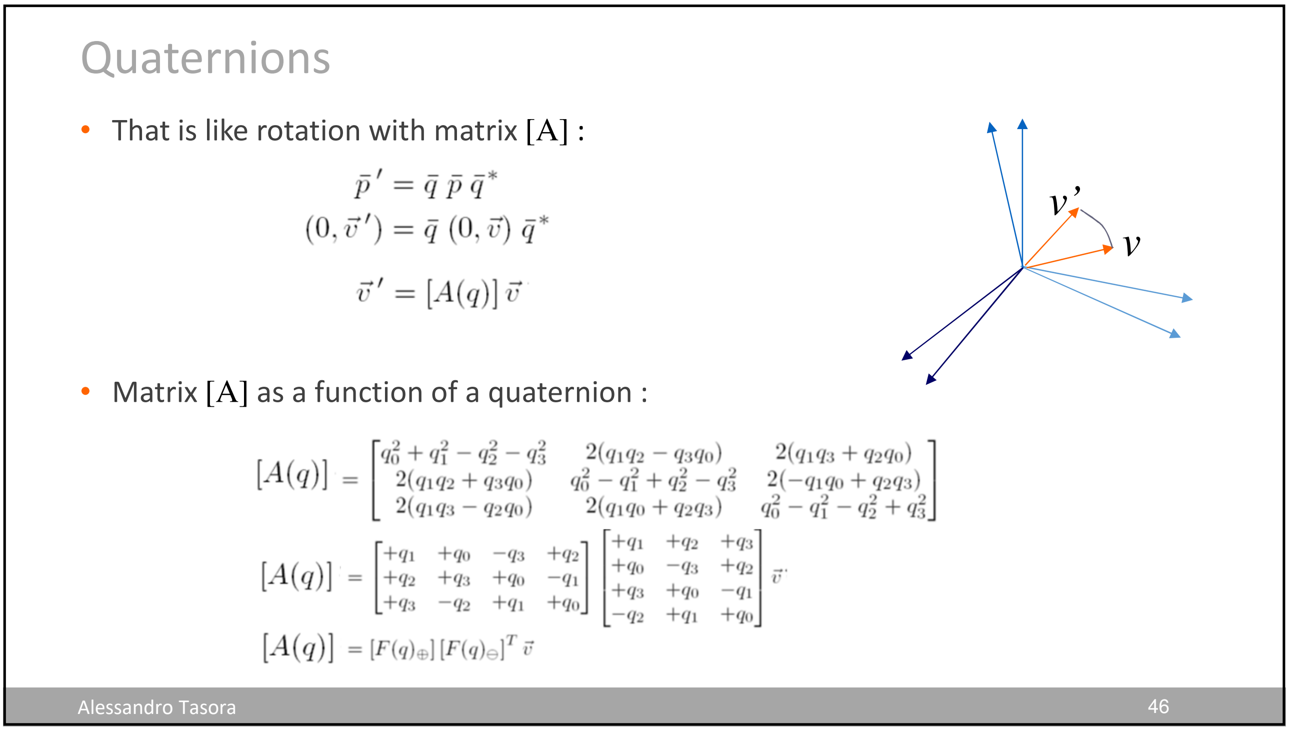
Quaternions

- 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 [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 [A] as a function of a quaternion :

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

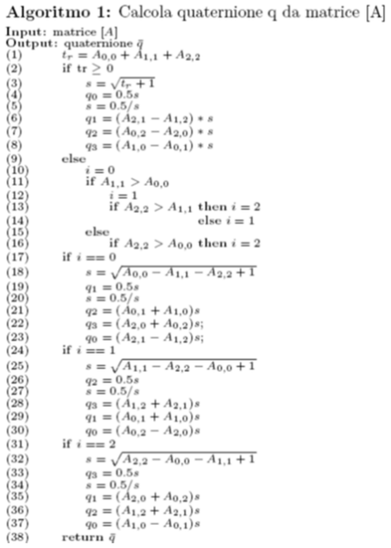
$$[A(q)] = \begin{bmatrix} +q_1 & +q_0 & -q_3 & +q_2 \\ +q_2 & +q_3 & +q_0 & -q_1 \\ +q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \begin{bmatrix} +q_1 & +q_2 & +q_3 \\ +q_0 & -q_3 & +q_2 \\ +q_3 & +q_0 & -q_1 \\ -q_2 & +q_1 & +q_0 \end{bmatrix} \vec{v}$$

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

Quaternions

- Viceversa:

(note: no singularity!)



Algoritmo 1: Calcola quaternione q da matrice [A]

```
Input: matrice $[A]$
Output: quaternione $\bar{q}$
    $t_r = A_{0,0} + A_{1,1} + A_{2,2}$
    if tr $\geq 0$
        $s = \sqrt{t_r + 1}$
        $q_0 = 0.5s$
        $s = 0.5/s$
        $q_1 = (A_{2,1} - A_{1,2}) * s$
        $q_2 = (A_{0,2} - A_{2,0}) * s$
        $q_3 = (A_{1,0} - A_{0,1}) * s$
    else
        $i = 0$
        if $A_{1,1} > A_{0,0}$
            $i = 1$
            if $A_{2,2} > A_{1,1}$ then $i = 2$
                                   else $i = 1$
        else
            if $A_{2,2} > A_{0,0}$ then $i = 2$
    if $i == 0$
        $s = \sqrt{A_{0,0} - A_{1,1} - A_{2,2} + 1}$
        $q_1 = 0.5s$
        $s = 0.5/s$
        $q_2 = (A_{0,1} + A_{1,0})s$
        $q_3 = (A_{2,0} + A_{0,2})s;$
        $q_0 = (A_{2,1} - A_{1,2})s;$
    if $i == 1$
        $s = \sqrt{A_{1,1} - A_{2,2} - A_{0,0} + 1}$
        $q_2 = 0.5s$
        $s = 0.5/s$
        $q_3 = (A_{1,2} + A_{2,1})s$
        $q_1 = (A_{0,1} + A_{1,0})s$
        $q_0 = (A_{0,2} - A_{2,0})s$
    if $i == 2$
        $s = \sqrt{A_{2,2} - A_{0,0} - A_{1,1} + 1}$
        $q_3 = 0.5s$
        $s = 0.5/s$
        $q_1 = (A_{2,0} + A_{0,2})s$
        $q_2 = (A_{1,2} + A_{2,1})s$
        $q_0 = (A_{1,0} - A_{0,1})s$
    return $\bar{q}$
```

Quaternions

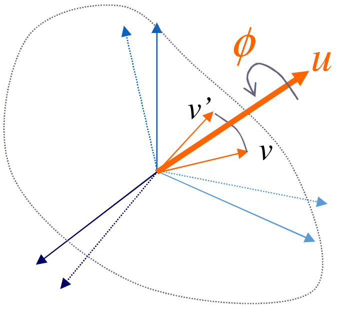
- Quaternion function of angle and axis

$$q_0 = \cos\left(\frac{\phi}{2}\right)$$

$$q_1 = u_x \sin\left(\frac{\phi}{2}\right)$$

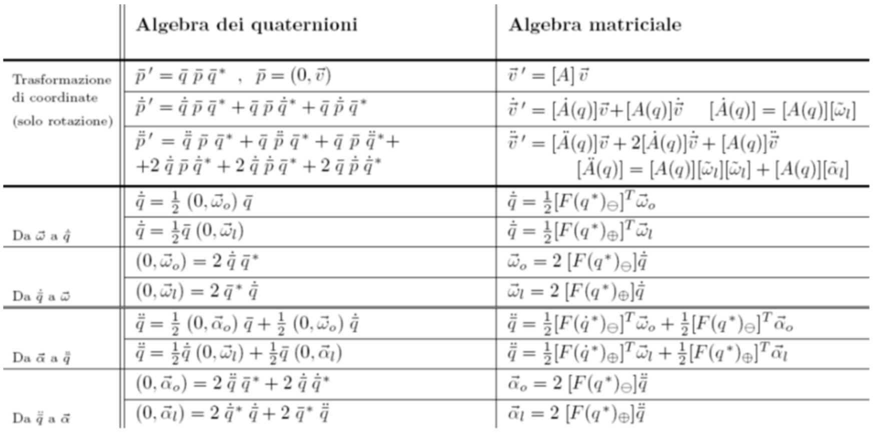
$$q_2 = u_y \sin\left(\frac{\phi}{2}\right)$$

$$q_3 = u_z \sin\left(\frac{\phi}{2}\right)$$



Quaternions

- Useful conversions



	Algebra dei quaternioni	Algebra matriciale
Trasformazione di coordinate (solo rotazione)	$\bar{p}\,' = \bar{q}\,\bar{p}\,\bar{q}^* \;\;,\;\; \bar{p} = (0, \vec{v})$	$\vec{v}\,' = [A]\,\vec{v}$
	$\dot{\bar{p}}\,' = \dot{\bar{q}}\,\bar{p}\,\bar{q}^* + \bar{q}\,\bar{p}\,\dot{\bar{q}}^* + \bar{q}\,\dot{\bar{p}}\,\bar{q}^*$	$\dot{\vec{v}}\,' = [\dot{A}(q)]\vec{v} + [A(q)]\dot{\vec{v}} \quad [\dot{A}(q)] = [A(q)][\tilde{\omega}_l]$
	$\ddot{\bar{p}}\,' = \ddot{\bar{q}}\,\bar{p}\,\bar{q}^* + \bar{q}\,\ddot{\bar{p}}\,\bar{q}^* + \bar{q}\,\bar{p}\,\ddot{\bar{q}}^* +$ $+2\,\dot{\bar{q}}\,\bar{p}\,\dot{\bar{q}}^* + 2\,\dot{\bar{q}}\,\dot{\bar{p}}\,\bar{q}^* + 2\,\bar{q}\,\dot{\bar{p}}\,\dot{\bar{q}}^*$	$\ddot{\vec{v}}\,' = [\ddot{A}(q)]\vec{v} + 2[\dot{A}(q)]\dot{\vec{v}} + [A(q)]\ddot{\vec{v}}$ $[\ddot{A}(q)] = [A(q)][\tilde{\omega}_l][\tilde{\omega}_l] + [A(q)][\tilde{\alpha}_l]$
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 $\vec{\alpha}$	$(0, \vec{\alpha}_o) = 2\,\ddot{\bar{q}}\,\bar{q}^* + 2\,\dot{\bar{q}}\,\dot{\bar{q}}^*$	$\vec{\alpha}_o = 2\,[F(q^*)_\ominus]\ddot{\bar{q}}$
	$(0, \vec{\alpha}_l) = 2\,\dot{\bar{q}}^*\,\dot{\bar{q}} + 2\,\bar{q}^*\,\ddot{\bar{q}}$	$\vec{\alpha}_l = 2\,[F(q^*)_\oplus]\ddot{\bar{q}}$

4. DYNAMICS

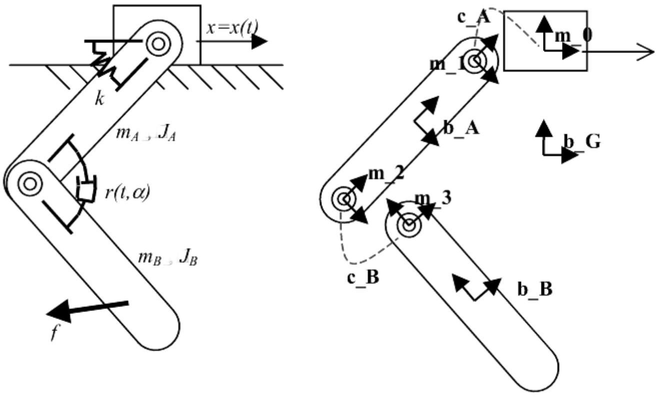
Basic concepts on ODEs and DAEs

Background

- This section describes a basic multibody 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.)

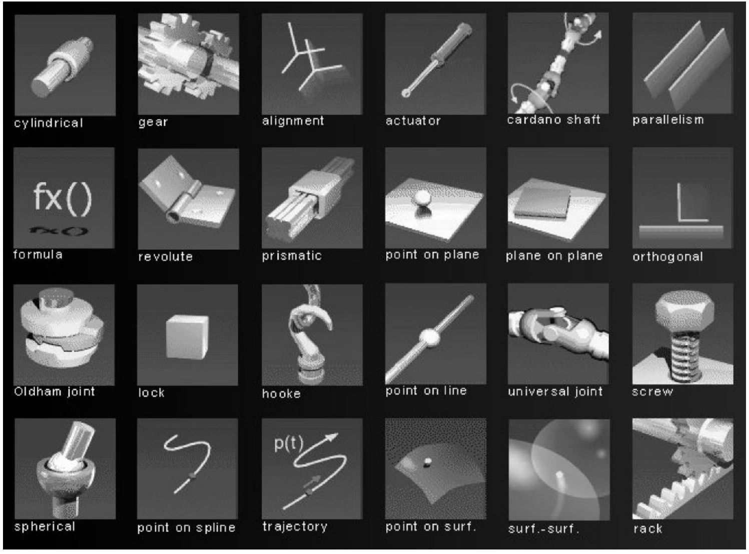
Model

- Example of model – using lagrangian ‘natural coordinates’ approach



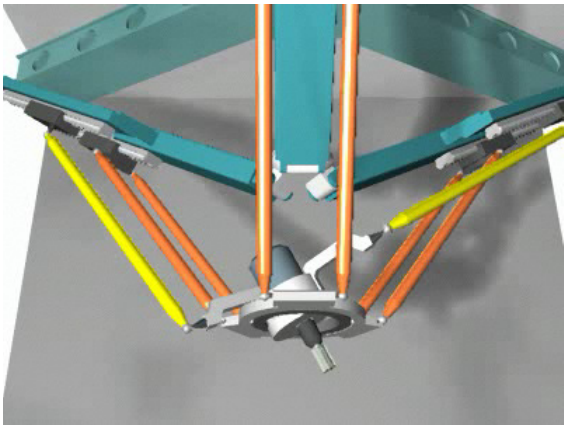
Model

Some constraint types in our Chrono::Engine software



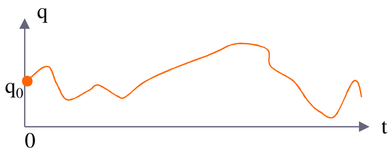
Examples

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



Equations of motion

- We are interested in the integral of motion $q(t)$ starting from boundary conditions $q_0(0)$

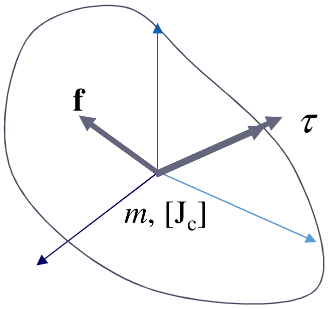


- Most often, the integrals must be approximated by *numerical integration*

Equations of motion

- Example: Newton-Euler equations, single body:

$$\underbrace{\begin{pmatrix} m\mathbf{I} & 0 \\ 0 & \mathbf{J}_c \end{pmatrix}}_{\text{Masses and inertia tensors}} \underbrace{\begin{pmatrix} \ddot{\mathrm{q}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix}}_{\text{Accel.}} + \underbrace{\begin{pmatrix} 0 \\ \boldsymbol{\omega} \times \mathbf{J}_c \boldsymbol{\omega} \end{pmatrix}}_{\text{Gyroscopic term}} = \begin{pmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{pmatrix} \begin{matrix} \text{Applied forces} \\ \text{Applied torques} \end{matrix}$$



- These can be obtained by developing, for instance, the Lagrange equations.
- Note that the unknowns are the linear accelerations and the angular accelerations: $\begin{pmatrix} \ddot{\mathrm{q}} \\ \dot{\omega} \end{pmatrix}$
- 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

Equations of motion

- More general: vector of independent generalized coordinates q for translation / rotation / etc.
- Lagrange formulation:

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

- Hamilton formulation:

$$\dot{p} = -\frac{\partial \mathcal{H}}{\partial q} \qquad \mathcal{H} = T + V$$

$$\dot{q} = \frac{\partial \mathcal{H}}{\partial p}. \qquad \mathcal{H} = \sum_i p_i \dot{q}_i - \mathcal{L}$$

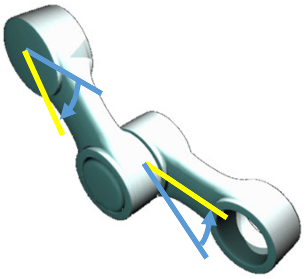
- Other variational principles:
  - Gauss least constraint principle
  - Jourdain principle
  - D'Alambert principle
  - Euler-Lagrange equations
  - etc.

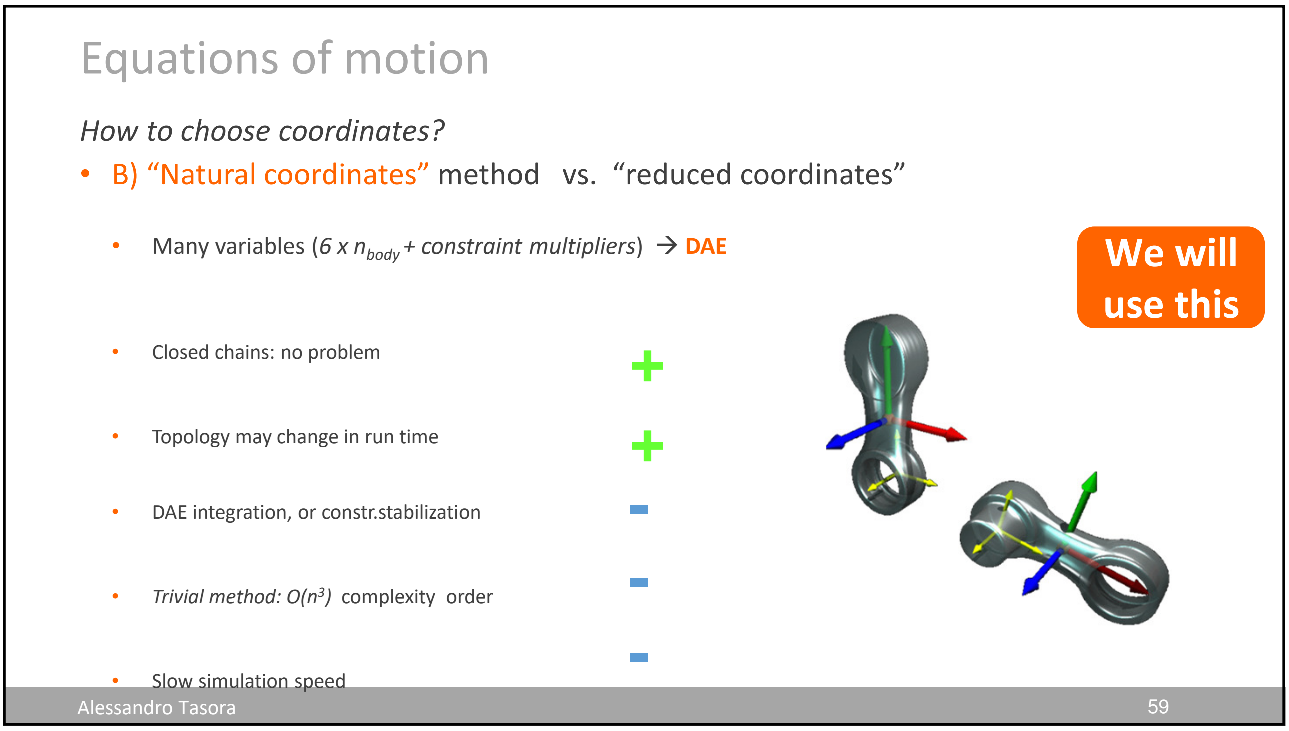
Equations of motion

How to choose coordinates?

- A) "Reduced coordinates" method vs. "Lagrangian multipliers"
  - Few coordinates ('joint coordinates or 'recursive' coordinates) → **ODE**

  - Very fast simulation +
  - *O(n)* complexity order +
  - Requires topological analysis −
  - Troubles with closed chains!!! −



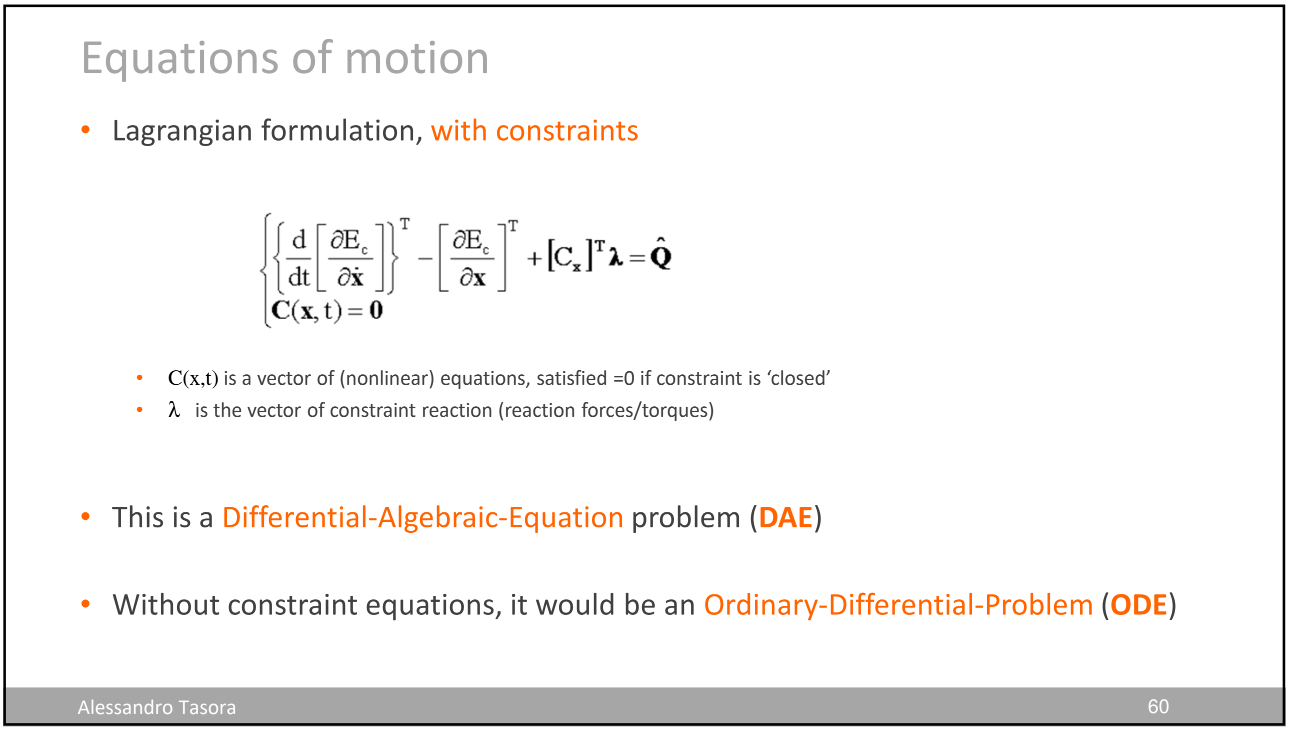


Equations of motion

How to choose coordinates?

- B) "Natural coordinates" method vs. "reduced coordinates"
  - Many variables (*6 x n_{body} + constraint multipliers*) → DAE
  - Closed chains: no problem +
  - Topology may change in run time +
  - DAE integration, or constr.stabilization -
  - *Trivial method: O(n^3)* complexity order -
  - Slow simulation speed -

We will use this



Equations of motion

- Lagrangian formulation, with constraints

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

  - $C(x,t)$ is a vector of (nonlinear) equations, satisfied =0 if constraint is 'closed'
  - λ is the vector of constraint reaction (reaction forces/torques)

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

How to solve a DAE?

- Integration of a DAEs is way more complex than a ODE
- One of the simpliest methods: index reduction:

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

Trick: from a DAE….

(Differential Algebraic Equations)

$$\mathbf{C} = \mathbf{C}(\mathbf{x}, t) = \mathbf{0}$$

$$\dot{\mathbf{C}} = [C_{\mathbf{x}}]\dot{\mathbf{x}} + \mathbf{C}_t = \mathbf{0}$$

$$\ddot{\mathbf{C}} = [C_{\mathbf{x}}]\ddot{\mathbf{x}} - \mathbf{Q}_c = \mathbf{0}$$

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

…to a simplier ODE

(Ordinary Differential Equations)

Solving for unknowns

- Transform from quaternion accelerations into angular accelerations (temporary change of coordinates)

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

Note...remember:

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

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

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

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

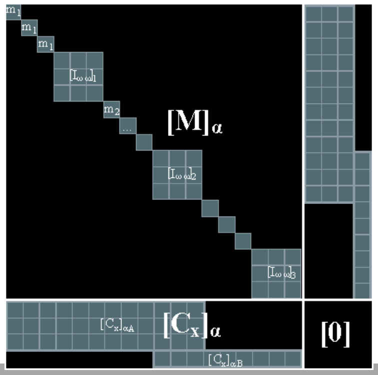
$$[T_q] = \begin{bmatrix} [I] & & & & \\ & \tfrac{1}{4}[G_1]^T_{(1)} & & & \\ & & \ldots & & \\ & & & [I] & \\ & & & & \tfrac{1}{4}[G_1]^T_{(n)} \end{bmatrix}$$

Solving for unknowns

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

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

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



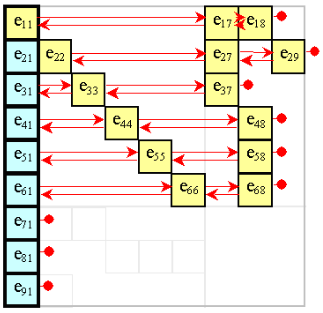
- More compact
- Still symmetric
- Still sparse
- Well conditioned diag.pivoting
- Inertia tensor as in Newton-Euler
- Quaternions (not angles!) for $[C_x]$
- ..efficient LDL^T decomposition !!!

Solving for unknowns

- Sparse matrix storage

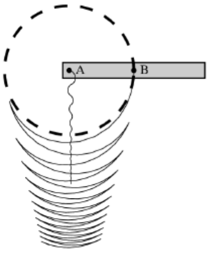
- Direct solvers?
  - LU decomposition – but does not exploit symmetry
  - LDL^T decomposition, symmetric
    - Can withstand redundant constraints
    - Linear-time decomposition for acyclic systems

- Iterative solvers?
  - Krylov methods (better with preconditioning)
    - GMRES
    - MINRES
    - …
  - Multigrid
  - …



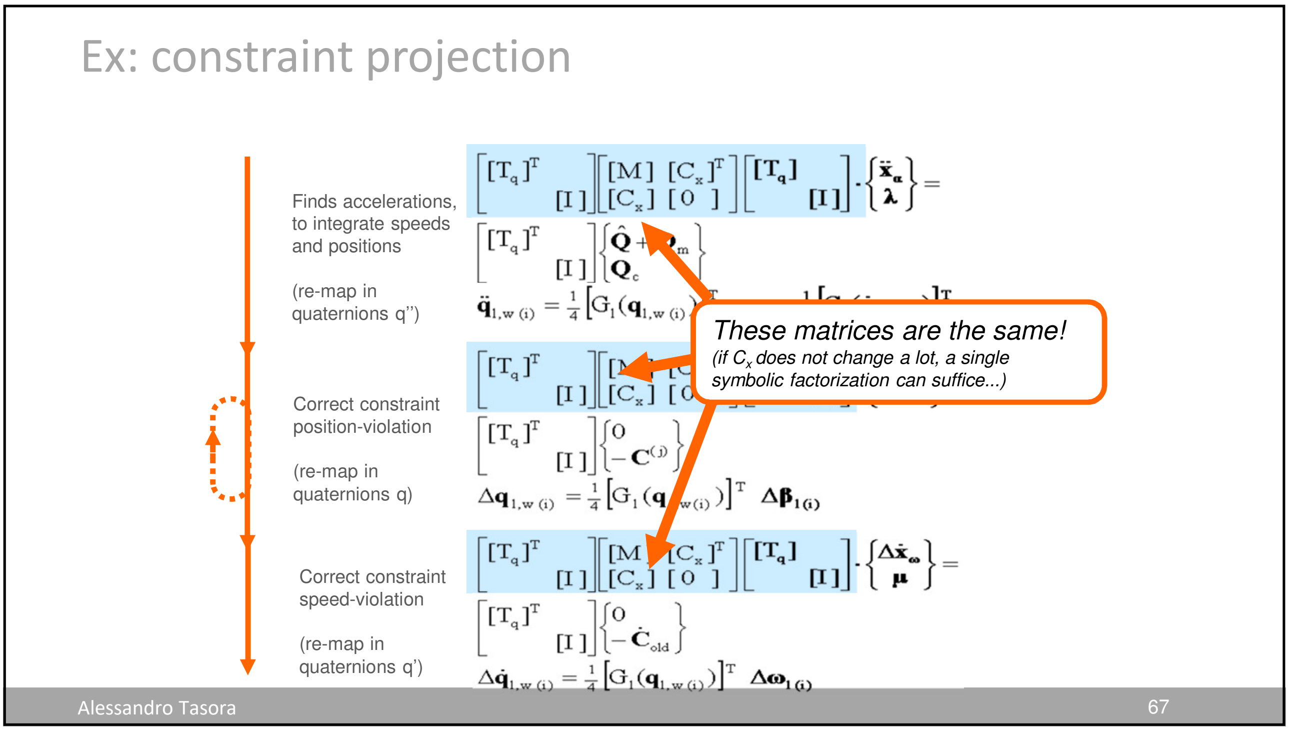
Stabilization schemes

- 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 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.



Ex: constraint projection

Finds accelerations, to integrate speeds and positions

(re-map in quaternions q'')

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

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

Correct constraint position-violation

(re-map in quaternions q)

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

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

Correct constraint speed-violation

(re-map in quaternions q')

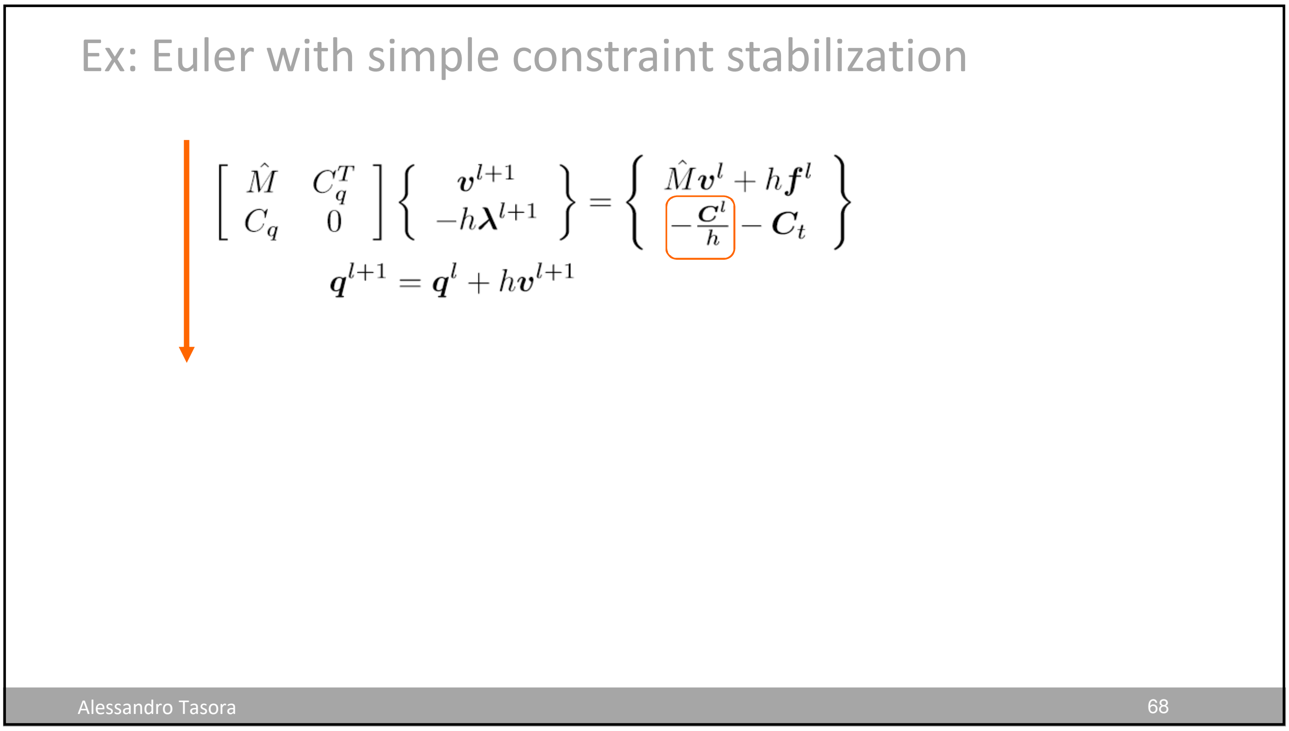
$$\begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{bmatrix}[M] & [C_x]^T \\ [C_x] & [0]\end{bmatrix}\begin{bmatrix}[\mathbf{T_q}] & \\ & [\mathbf{I}]\end{bmatrix}\cdot\begin{Bmatrix}\Delta\dot{\mathbf{x}}_\omega \\ \boldsymbol{\mu}\end{Bmatrix} = \begin{bmatrix}[T_q]^T & \\ & [I]\end{bmatrix}\begin{Bmatrix}0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

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

These matrices are the same!

(if C_x does not change a lot, a single symbolic factorization can suffice...)

Alessandro Tasora 67



Ex: Euler with simple constraint stabilization

$$\begin{bmatrix}\hat{M} & C_q^T \\ C_q & 0\end{bmatrix}\begin{Bmatrix}\boldsymbol{v}^{l+1} \\ -h\boldsymbol{\lambda}^{l+1}\end{Bmatrix} = \begin{Bmatrix}\hat{M}\boldsymbol{v}^l + h\boldsymbol{f}^l \\ \boxed{-\frac{\boldsymbol{C}^l}{h}} - \boldsymbol{C}_t\end{Bmatrix}$$

$$\boldsymbol{q}^{l+1} = \boldsymbol{q}^l + h\boldsymbol{v}^{l+1}$$

Alessandro Tasora 68

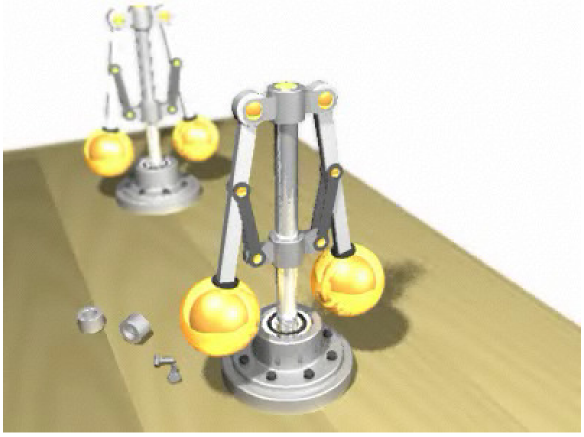
Ex: implicit DAE solver: Euler implicit with constraints

$$\begin{bmatrix} \left[\hat{M} - h^2 \nabla_q \boldsymbol{f}^{l+1} - h \nabla_v \boldsymbol{f}^{l+1}\right] & C_q^T \\ C_q & 0 \end{bmatrix} \begin{Bmatrix} \Delta \boldsymbol{v}^{l+1} \\ -h\Delta\boldsymbol{\lambda}^{l+1} \end{Bmatrix} = \begin{Bmatrix} \left(\boldsymbol{v}^l - \boldsymbol{v}^{l+1}\right)\hat{M} + h\boldsymbol{f}^{l+1} + hC_q^T\boldsymbol{\lambda}^{l+1} \\ -\frac{\boldsymbol{C}^{l+1}}{h} \end{Bmatrix}$$

$$\begin{aligned} \boldsymbol{v}_{n+1}^{l+1} &= \boldsymbol{v}_n^{l+1} + \Delta\boldsymbol{v}^{l+1} \\ \boldsymbol{\lambda}_{n+1}^{l+1} &= \boldsymbol{\lambda}_n^{l+1} + \Delta\boldsymbol{\lambda}^{l+1} \\ \boldsymbol{q}^{l+1} &= \boldsymbol{q}^l + h\boldsymbol{v}_{n+1}^{l+1} \end{aligned}$$

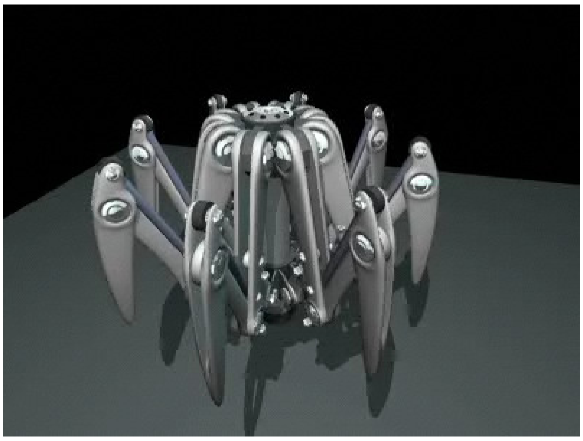
Examples

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



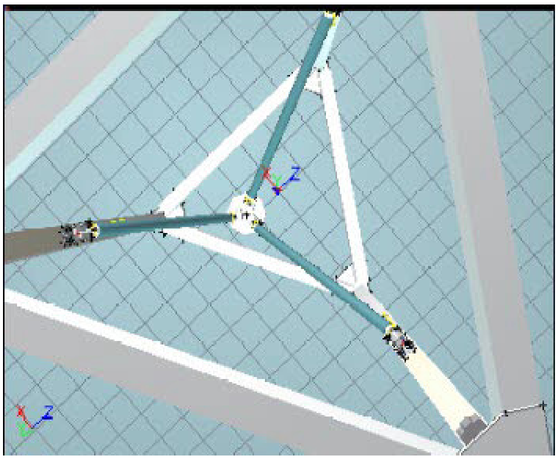
Examples

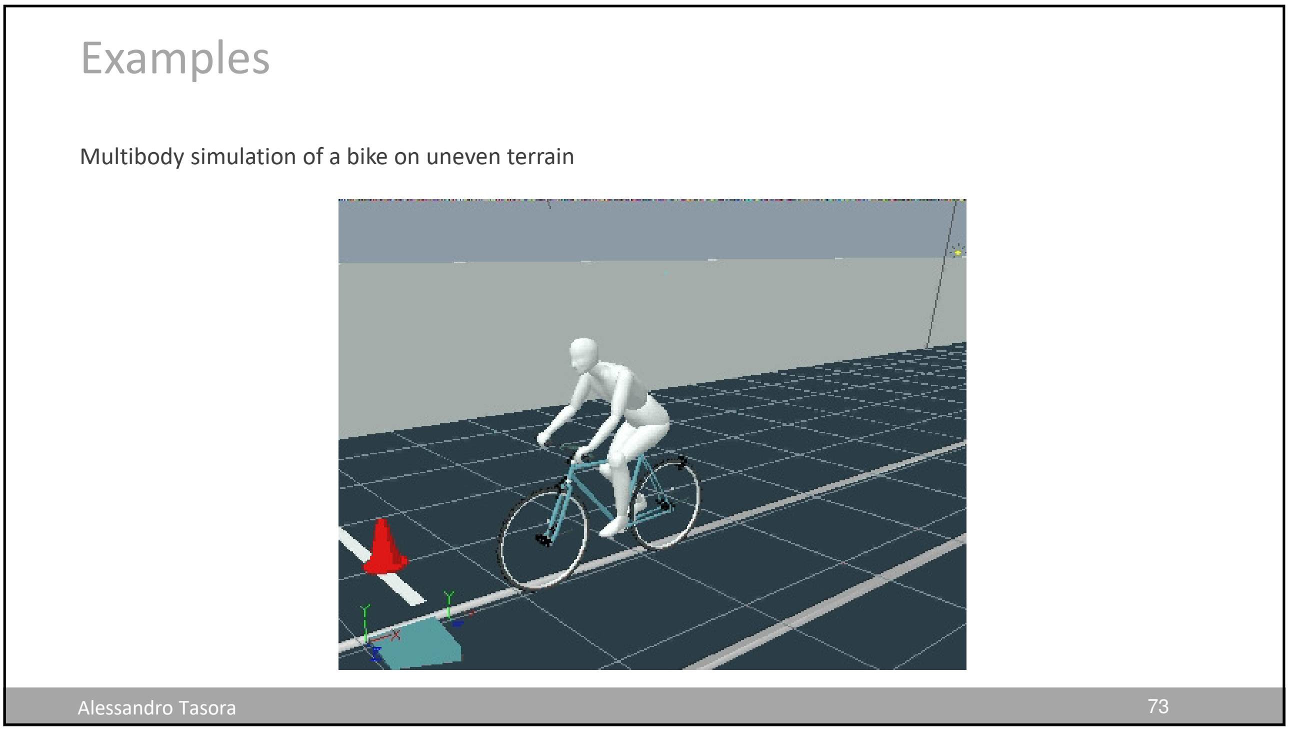
Benchmark to test the efficiency of our sparse solver



Examples

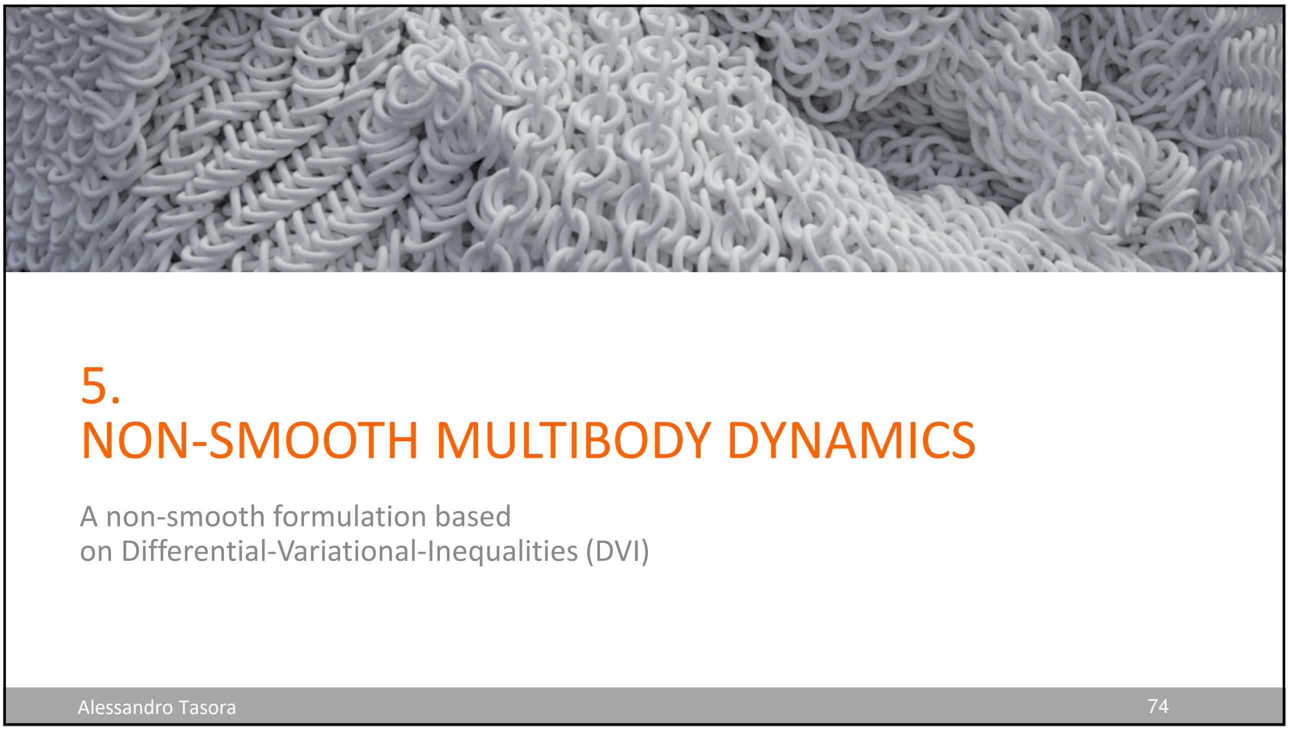
Simulation of the pneumatic-actuated TORX parallel robot





Examples

Multibody simulation of a bike on uneven terrain



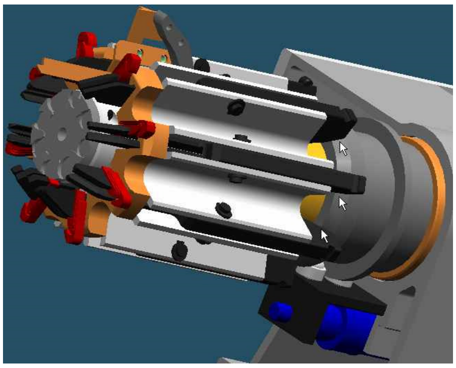
5. NON-SMOOTH MULTIBODY DYNAMICS

A non-smooth formulation based
on Differential-Variational-Inequalities (DVI)

Introduction to non-smooth dynamics

- Unilateral constraints and friction: happen in many mechanisms
- Set-valued force laws lead to a DVI (*Differential Variational Inclusion problem*)

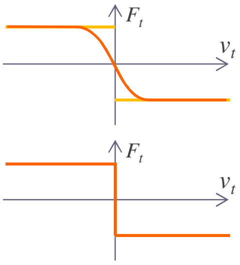
Example: packaging device



Example: radial multi-gripper

Why non-smooth dynamics?

- "hard" frictional contacts happen in many mechanisms
  - Packaging devices
  - Keylocks
  - Toys
  - Masonry, etc.
- Two main approaches to simulate contacts:
  - **Smooth dynamics** with regularization of non-smooth contact forces → DAEs, ODEs
  - **Non-smooth dynamics** with set-valued contact forces → DVIs, MDIs, etc.



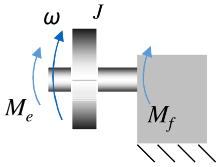
Why non-smooth dynamics?

- 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

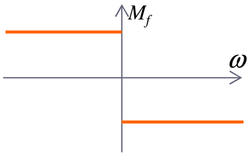
  $dx/dt \in \ f\,(x,t)$ → Differential Inclusion! (DI)

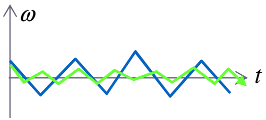


- *Example: a flywheel with brake torque and applied torque (looks simple?!)*

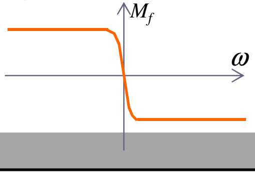
$$J\ d\omega/dt \ = \ M_f(\omega) + M_e(t) \quad \text{where} \quad M_f = -M_{f\,max} \ \text{if} \ \ \omega > 0$$

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





- 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!



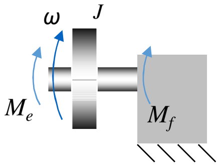
Why non-smooth dynamics?

- 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)

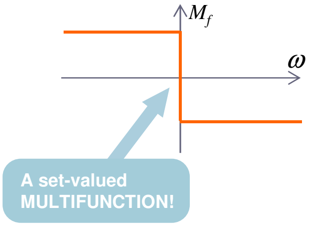


- *Example: a flywheel with brake torque and applied torque (simple?!)*
- *Improved model!*

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

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

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

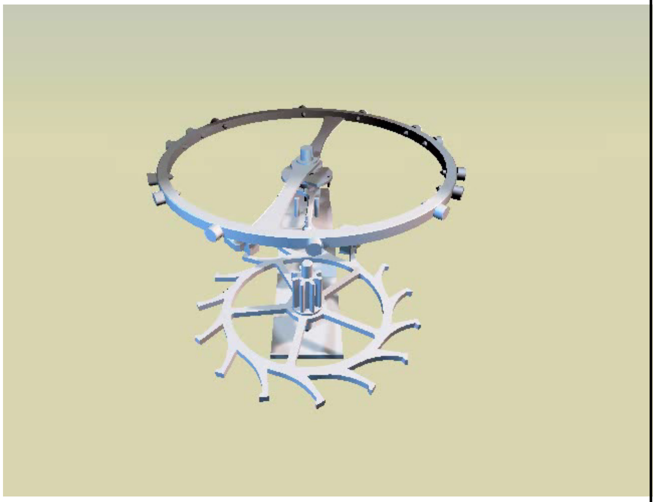


- This could handle also $\omega=0$ case, ex. brake sticking
- But now we have a differential inclusion $d\omega/dt \in \ f\,(\omega,t)$. WE NEED A METHOD TO SOLVE IT

Example

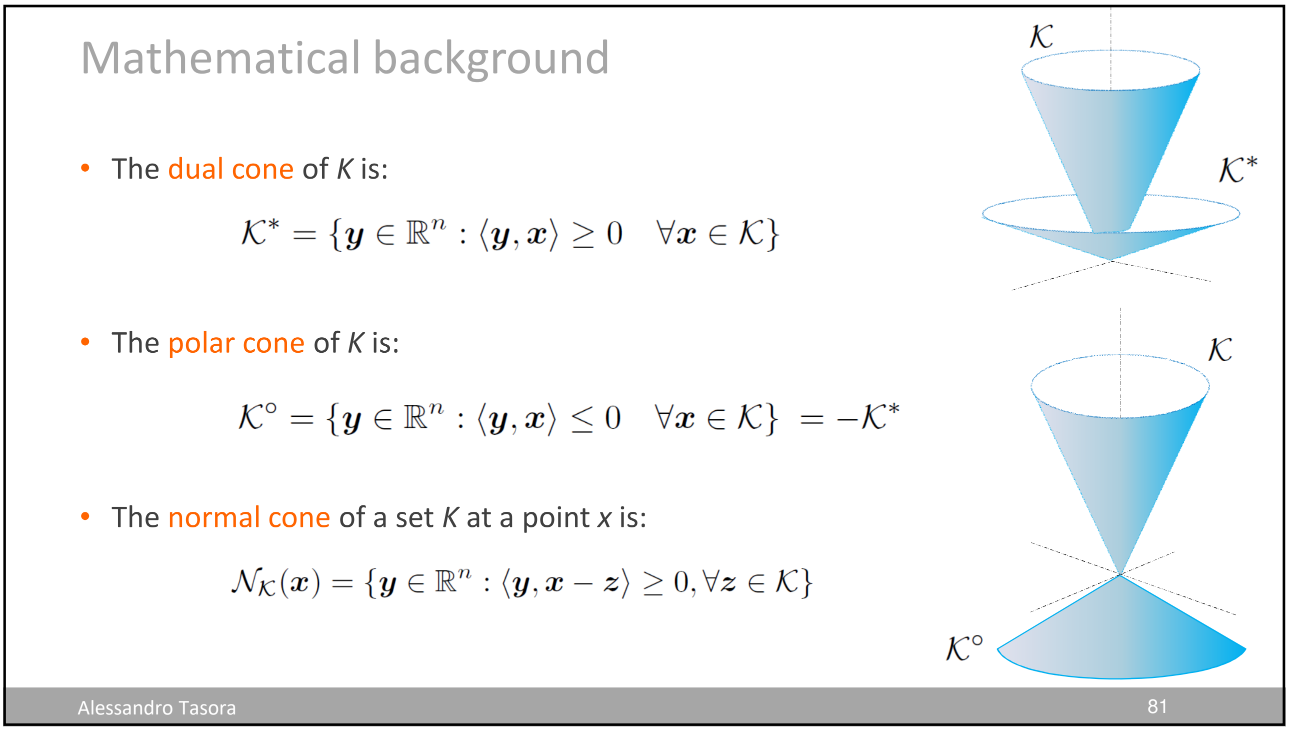
- Example of simulation where the non-smooth approach is a winner: a wrist watch escapement
  - Extremely stiff contacts
  - Extremely light parts

Example: ProjectChrono simulation of a Swiss escapement (A.Tasora)



A mathematician is a device for turning coffee into theorems.

Paul Erdős



Mathematical background

- The dual cone of *K* is:

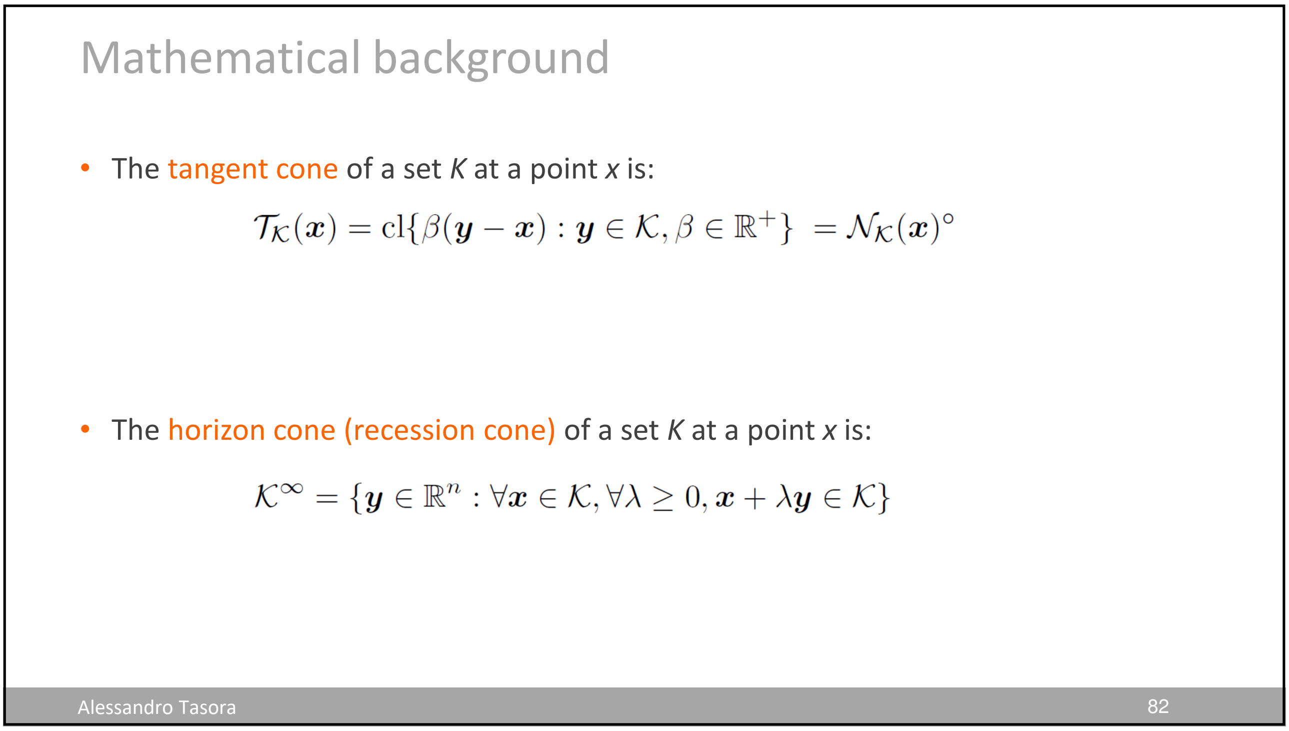
$$\mathcal{K}^* = \{\boldsymbol{y} \in \mathbb{R}^n : \langle \boldsymbol{y}, \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{x} \in \mathcal{K}\}$$

- The polar cone of *K* is:

$$\mathcal{K}^\circ = \{\boldsymbol{y} \in \mathbb{R}^n : \langle \boldsymbol{y}, \boldsymbol{x} \rangle \leq 0 \quad \forall \boldsymbol{x} \in \mathcal{K}\} \ = -\mathcal{K}^*$$

- The normal cone of a set *K* at a point *x* is:

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



Mathematical background

- The tangent cone of a set *K* at a point *x* is:

$$\mathcal{T}_{\mathcal{K}}(\boldsymbol{x}) = \mathrm{cl}\{\beta(\boldsymbol{y} - \boldsymbol{x}) : \boldsymbol{y} \in \mathcal{K}, \beta \in \mathbb{R}^+\} \ = \mathcal{N}_{\mathcal{K}}(\boldsymbol{x})^\circ$$

- The horizon cone (recession cone) of a set *K* at a point *x* is:

$$\mathcal{K}^\infty = \{\boldsymbol{y} \in \mathbb{R}^n : \forall \boldsymbol{x} \in \mathcal{K}, \forall \lambda \geq 0, \boldsymbol{x} + \lambda \boldsymbol{y} \in \mathcal{K}\}$$

Mathematical background

- The indicator function of a subset $\mathcal{A} \in \mathcal{E}$ is a scalar function:

$$I_{\mathcal{A}}(\boldsymbol{x}) = \begin{cases} \infty \text{ if } \boldsymbol{x} \in \mathcal{A} \\ 0 \text{ if } \boldsymbol{x} \notin \mathcal{A} \end{cases}$$

Mathematical background

- The subdifferential of a scalar, convex, possibly nondifferentiable function at *x* is:

$$\partial f(\boldsymbol{x}_0) = \{\boldsymbol{g} : f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + \langle \boldsymbol{g}, (\boldsymbol{x} - \boldsymbol{x}_0) \rangle \;\; \forall \boldsymbol{x} \in \mathcal{E}\}$$

- notes:
  - The normal cone is the subdifferential of the indicator function of *K*:

    $$\partial I_{\mathcal{K}}(\boldsymbol{x}) = \mathcal{N}_{\mathcal{K}}(\boldsymbol{x})$$

  - If *f* is differentiable,

    $$\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$$

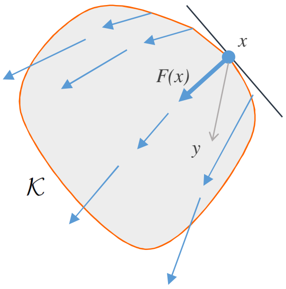
Mathematical background

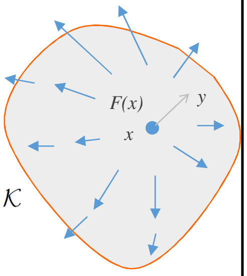
- Variational Inequality (VI):

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

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

(see Kinderleher and Stampacchia ,1980)





Alternative VI formulation:

$$\boldsymbol{x} \in \mathcal{K} \quad : \quad F(\boldsymbol{x}) \in \mathcal{N}_{\mathcal{K}}(\boldsymbol{x})$$

Mathematical background

- Linear Complementarity Problem (LCP):

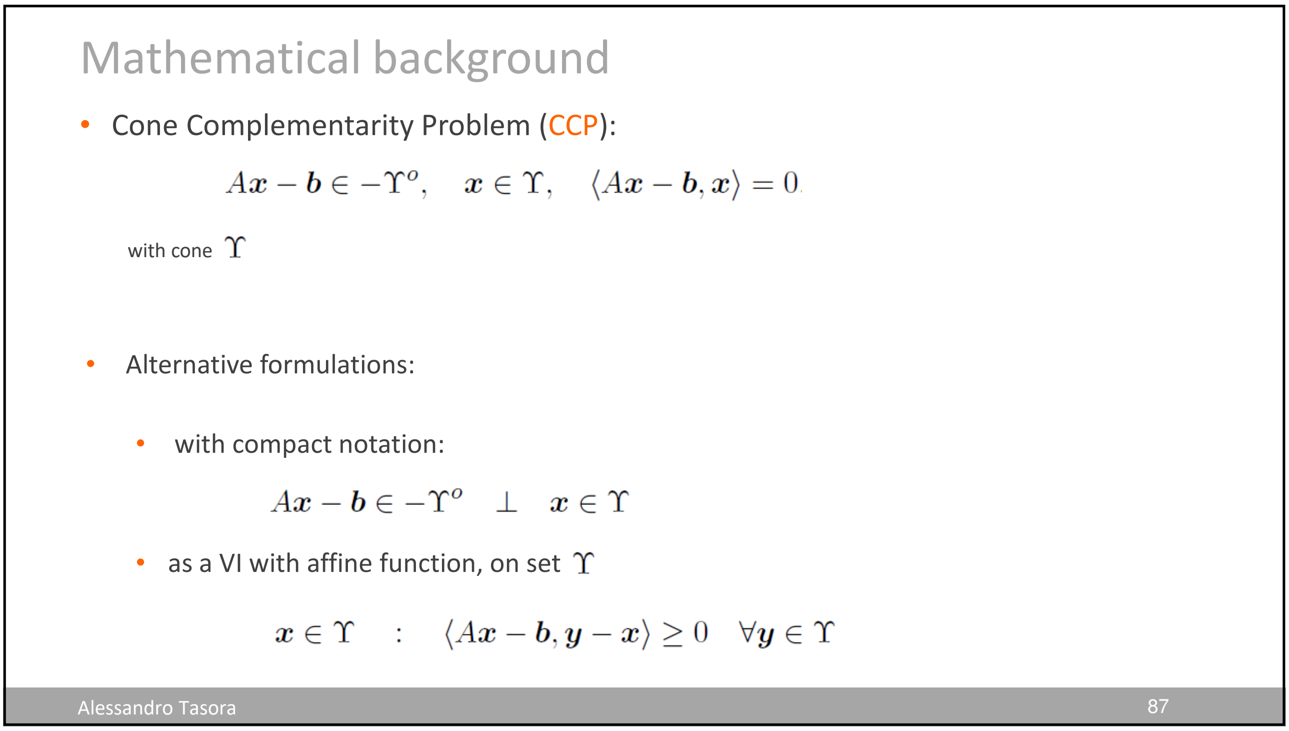
$$A\boldsymbol{x} - \boldsymbol{b} \geq \boldsymbol{0}, \quad \boldsymbol{x} \geq \boldsymbol{0}, \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{x} \rangle = 0.$$

- Alternative formulations:
  - with compact notation:

$$A\boldsymbol{x} - \boldsymbol{b} \geq \boldsymbol{0} \quad \perp \quad \boldsymbol{x} \geq \boldsymbol{0}$$

  - as a VI with affine function, on positive orthant

$$\boldsymbol{x} \in \mathbb{R}^n_+ \quad : \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{y} - \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \mathbb{R}^n_+$$



Mathematical background

- Cone Complementarity Problem (CCP):

$$A\boldsymbol{x} - \boldsymbol{b} \in -\Upsilon^o, \quad \boldsymbol{x} \in \Upsilon, \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{x} \rangle = 0.$$

with cone Υ

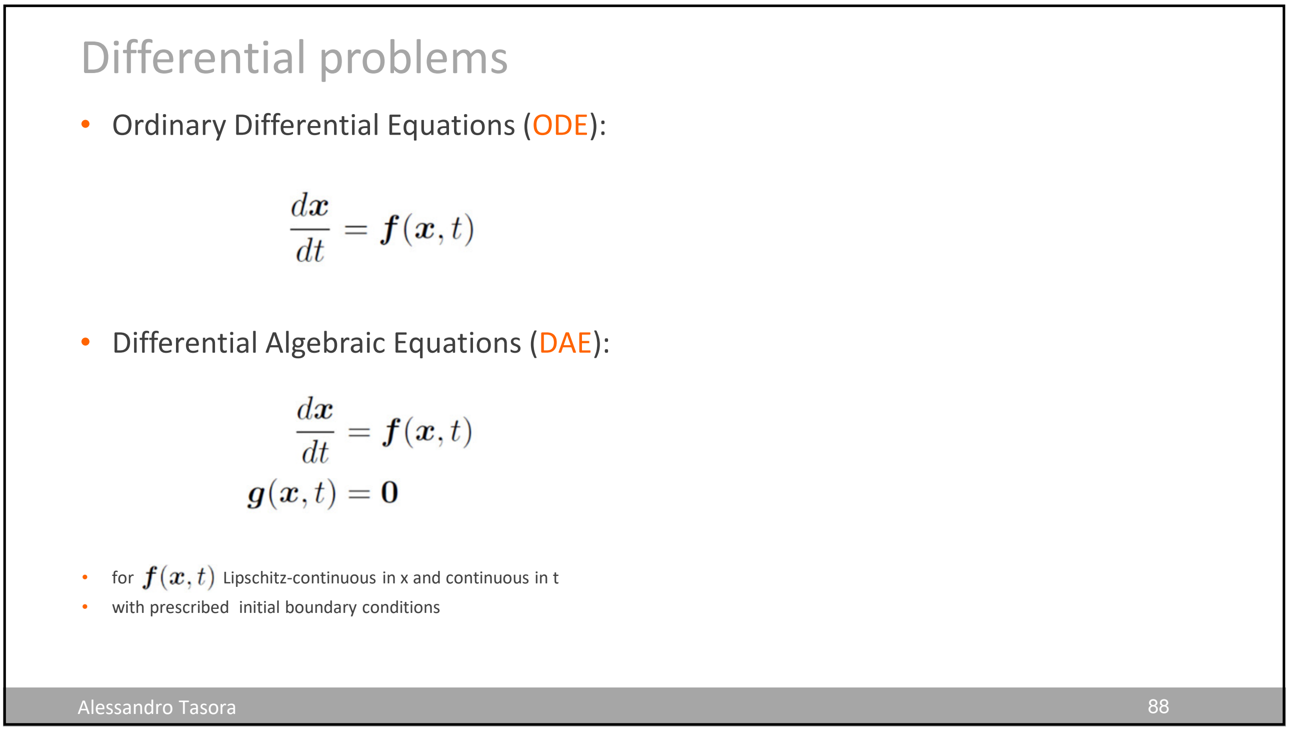
- Alternative formulations:
  - with compact notation:

$$A\boldsymbol{x} - \boldsymbol{b} \in -\Upsilon^o \quad \perp \quad \boldsymbol{x} \in \Upsilon$$

  - as a VI with affine function, on set Υ

$$\boldsymbol{x} \in \Upsilon \quad : \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{y} - \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \Upsilon$$

Alessandro Tasora 87



Differential problems

- Ordinary Differential Equations (ODE):

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t)$$

- Differential Algebraic Equations (DAE):

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t)$$
$$\boldsymbol{g}(\boldsymbol{x}, t) = \boldsymbol{0}$$

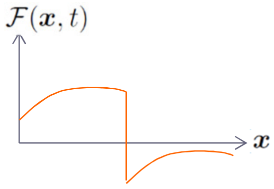
- for $\boldsymbol{f}(\boldsymbol{x}, t)$ Lipschitz-continuous in x and continuous in t
- with prescribed initial boundary conditions

Alessandro Tasora 88

Differential problems

- Differential Inclusions (DI):

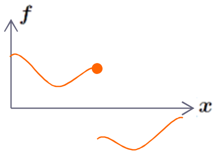
$$\frac{d\boldsymbol{x}}{dt} \in \mathcal{F}(\boldsymbol{x}, t)$$

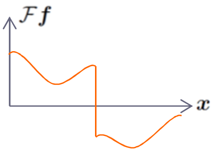


- with prescribed initial boundary conditions
- for set-valued $\mathcal{F}(\boldsymbol{x}, t)$
- closed, bounded and convex $\mathcal{F}(\boldsymbol{x}, t)$

- Example: Filippov Differential Inclusions for discontinuous $\boldsymbol{f}(\boldsymbol{x}, t)$

$$\frac{d\boldsymbol{x}}{dt} \in \mathcal{F}\boldsymbol{f}(\boldsymbol{x}, t) \quad \mathcal{F}f(\boldsymbol{x}, t) = \bigcap_{\eta>0} \bigcap_{N:\lambda_0(N)=0} \overline{co}\, \boldsymbol{f}(\boldsymbol{x} + \eta \boldsymbol{B}_1 \setminus N, t)$$





Differential problems

- Measure Differential Inclusions (MDI):

$$\frac{d\boldsymbol{v}}{dt} \in \mathcal{K}(\boldsymbol{q}, t)$$

- for set-valued $\mathcal{K}(\boldsymbol{q}, t)$
- closed, bounded and convex $\mathcal{K}(\boldsymbol{q}, t)$
- with function of bounded variation (BV), discontinuous

- Lebesgue decomposition of measure $d\boldsymbol{v} = \boldsymbol{\nu}_s + \boldsymbol{h}\lambda_0$

  - Singular part $\boldsymbol{\nu}_s$ → speed 'jumps'
  - Lebesgue measure λ_0 for continuous $\boldsymbol{h}(t) \in L^1(a,b)$ → classical 'acceleration'

- No acceleration in the classical sense!
  Relaxed acceleration, as a *distribution of vector-signed Borel measures*

Differential problems

- Measure Differential Inclusions (MDI):

$$\frac{d\boldsymbol{v}}{dt} \in \mathcal{K}(\boldsymbol{q}, t)$$

$$d\boldsymbol{v} = \boldsymbol{\nu}_s + \boldsymbol{h}\lambda_0$$

- Strong definition of solution:
  - $\boldsymbol{h}(t) \in \mathcal{K}(t)$ almost all t
  - Radon-Nikodym $d\boldsymbol{\nu}_s/|\boldsymbol{\nu}_s|(t) \in \mathcal{K}(t)_\infty$

Differential problems

- Measure Differential Inclusions (MDI):

$$\frac{d\boldsymbol{v}}{dt} \in \mathcal{K}(\boldsymbol{q}, t)$$

$$d\boldsymbol{v} = \boldsymbol{\nu}_s + \boldsymbol{h}\lambda_0$$

- Weak definition of solution: [Stewart]
  - $\dfrac{\int \phi(t) d\boldsymbol{\nu}(dt)}{\int \phi(t) dt} \in \overline{co} \bigcup_{\tau:\phi(\tau)\neq 0} \mathcal{K}(\tau)$

- Side note: MDI can solve the *Painlevé paradox* (1895)

Differential Variational Inequality

- Differential Variational Inequality (DVI)

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t)$$

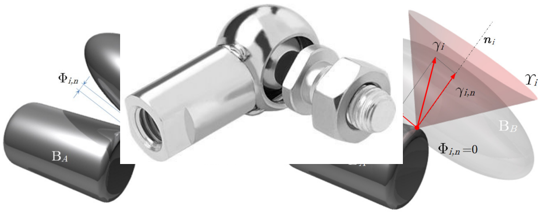
$$\boldsymbol{u} \in \mathsf{SOL}(\boldsymbol{F}, \mathcal{K}) \qquad\qquad \Xi(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0$$

With $\boldsymbol{u} \in \mathsf{SOL}(\boldsymbol{F}, \mathcal{K})$ as set of solutions to the VI $(\boldsymbol{F}, \mathcal{K})$

- Note that DVI with vector-signed measures are MDI: hard contacts lead to
  - velocities as BV functions, with Lebesgue decomposition $d\boldsymbol{v} = \boldsymbol{\nu}_s + \boldsymbol{h}\lambda_0$
  - accelerations in *distributional* generalized sense
- Note that DAE are a special case of DVI where $\mathcal{K} = \mathbb{R}^n$ and $\boldsymbol{F} = \boldsymbol{0}$

The DVI model

- Formulating Multibody Non-Smooth Contact Dynamics as a DVI:
  - Set $\mathcal{G}_{\mathcal{B}}$ of bilateral joints
  - Set $\mathcal{G}_{\mathcal{A}}$ of point contacts
  - External forces



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

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

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

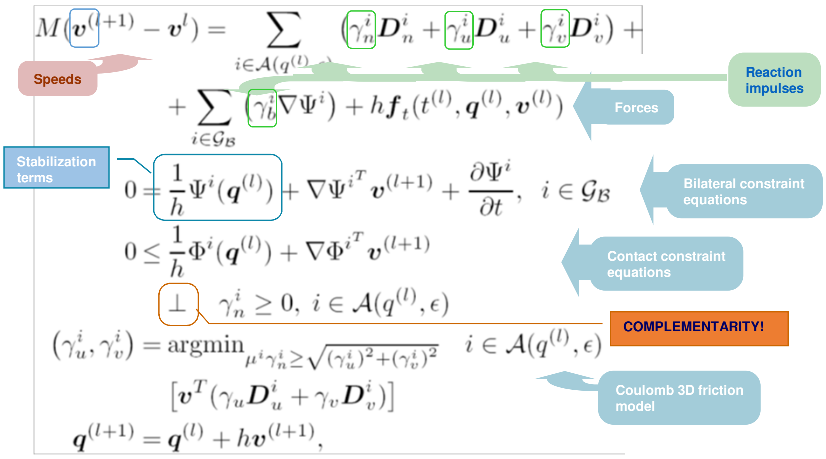
$$\widehat{\gamma}^i_{\mathcal{A}} \in \mathsf{SOL}\left(\Upsilon^i, F(t, q(t), v(t), \cdot)\right), \quad i \in \mathcal{G}_{\mathcal{A}}.$$

Bilateral constraint equations

Contact forces VIs

The DVI time-stepper: a VI

- Discretization of DVI leads to a VI problem with unknown speed jumps & impulses:



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

Speeds

Reaction impulses

Forces

Stabilization terms

Bilateral constraint equations

Contact constraint equations

COMPLEMENTARITY!

Coulomb 3D friction model

VI as a cone complementarity

- Aiming at a more compact formulation:

$$
\begin{aligned}
\boldsymbol{b}_\mathcal{A} &= \left\{\frac{1}{h}\Phi^{i_1}, 0, 0, \frac{1}{h}\Phi^{i_2}, 0, 0, \ldots, \frac{1}{h}\Phi^{i_{n_\mathcal{A}}}, 0, 0\right\} \\
\boldsymbol{\gamma}_\mathcal{A} &= \left\{\gamma_n^{i_1}, \gamma_u^{i_1}, \gamma_v^{i_1}, \gamma_n^{i_2}, \gamma_u^{i_2}, \gamma_v^{i_2}, \ldots, \gamma_n^{i_{n_\mathcal{A}}}, \gamma_u^{i_{n_\mathcal{A}}}, \gamma_v^{i_{n_\mathcal{A}}}\right\} \\
\boldsymbol{b}_\mathcal{B} &= \left\{\frac{1}{h}\Psi^1 + \frac{\partial\Psi^1}{\partial t}, \frac{1}{h}\Psi^2 + \frac{\partial\Psi^2}{\partial t}, \ldots, \frac{1}{h}\Psi^{n_\mathcal{B}} + \frac{\partial\Psi^{n_\mathcal{B}}}{\partial t}\right\} \\
\boldsymbol{\gamma}_\mathcal{B} &= \left\{\gamma_b^1, \gamma_b^2, \ldots, \gamma_b^{n_\mathcal{B}}\right\}
\end{aligned}
$$

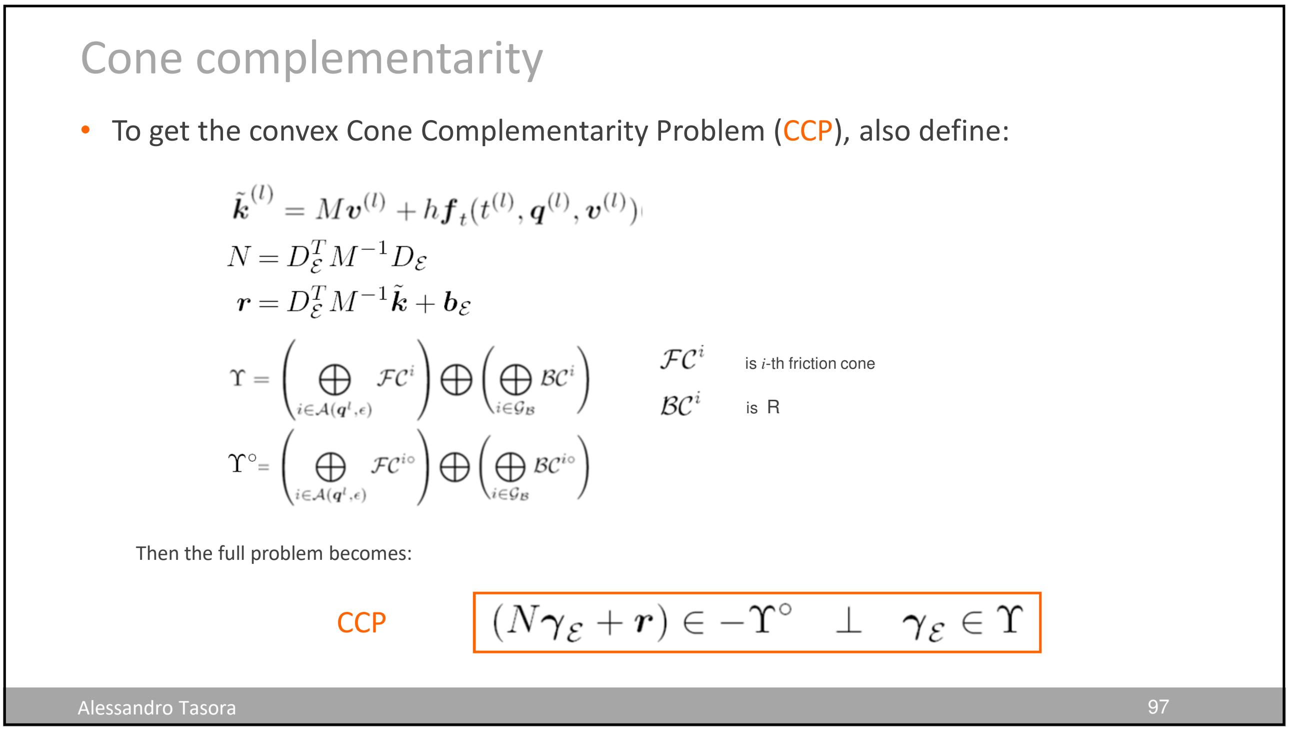
$$D_\mathcal{A} = \left[D^{i_1}|D^{i_2}|\ldots|D^{i_{n_\mathcal{A}}}\right], \quad i \in \mathcal{A}(\boldsymbol{q}^l, \epsilon) \qquad D^i = [\boldsymbol{D}_n^i|\boldsymbol{D}_u^i|\boldsymbol{D}_v^i]$$

$$D_\mathcal{B} = \left[\nabla\Psi^{i_1}|\nabla\Psi^{i_2}|\ldots|\nabla\Psi^{i_{n_\mathcal{B}}}\right], \quad i \in \mathcal{G}_\mathcal{B}$$

$$\boldsymbol{b}_\mathcal{E} \in \mathbb{R}^{n_\mathcal{E}} = \{\boldsymbol{b}_\mathcal{A}, \boldsymbol{b}_\mathcal{B}\}$$

$$\boldsymbol{\gamma}_\mathcal{E} \in \mathbb{R}^{n_\mathcal{E}} = \{\boldsymbol{\gamma}_\mathcal{A}, \boldsymbol{\gamma}_\mathcal{B}\}$$

$$D_\mathcal{E} = [D_\mathcal{A}|D_\mathcal{B}]$$



Cone complementarity

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

$$\tilde{\boldsymbol{k}}^{(l)} = M\boldsymbol{v}^{(l)} + h\boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)})$$

$$N = D_{\mathcal{E}}^T M^{-1} D_{\mathcal{E}}$$

$$\boldsymbol{r} = D_{\mathcal{E}}^T M^{-1} \tilde{\boldsymbol{k}} + \boldsymbol{b}_{\mathcal{E}}$$

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

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

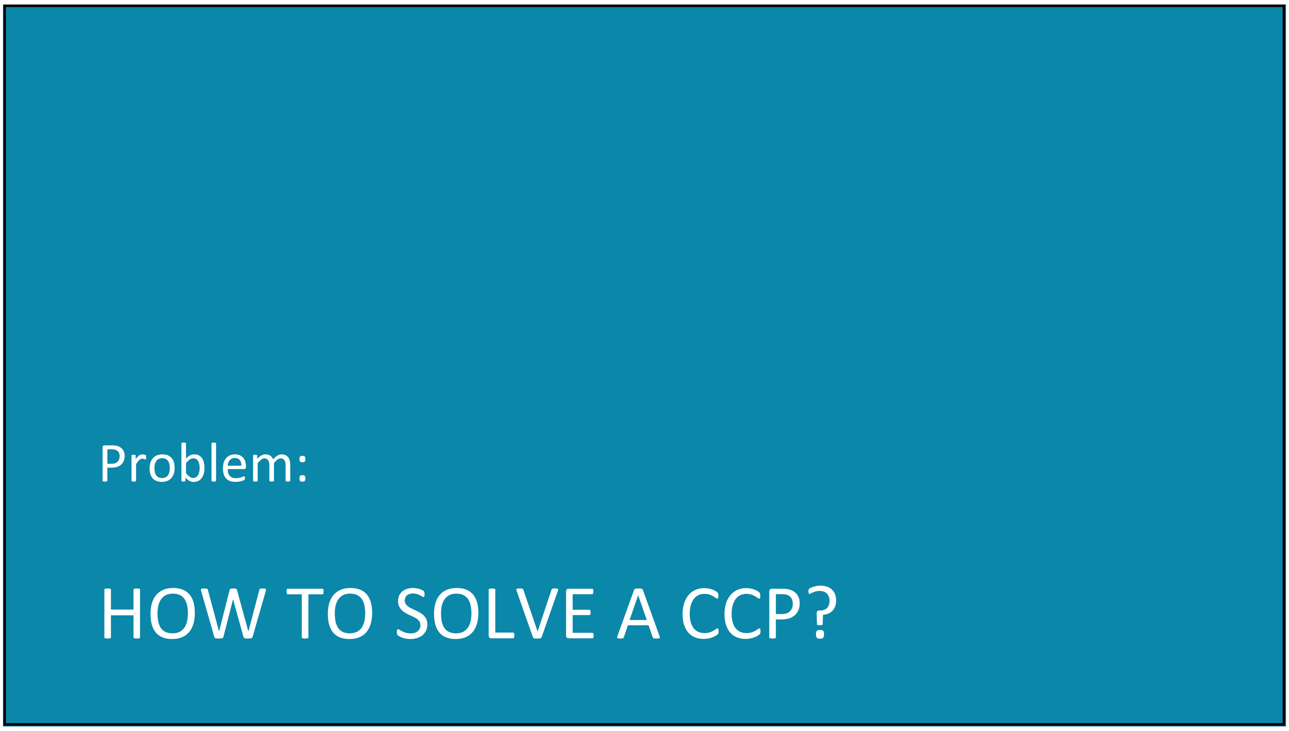
$\mathcal{FC}^i$ is i-th friction cone

$\mathcal{BC}^i$ is R

Then the full problem becomes:

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

Alessandro Tasora 97

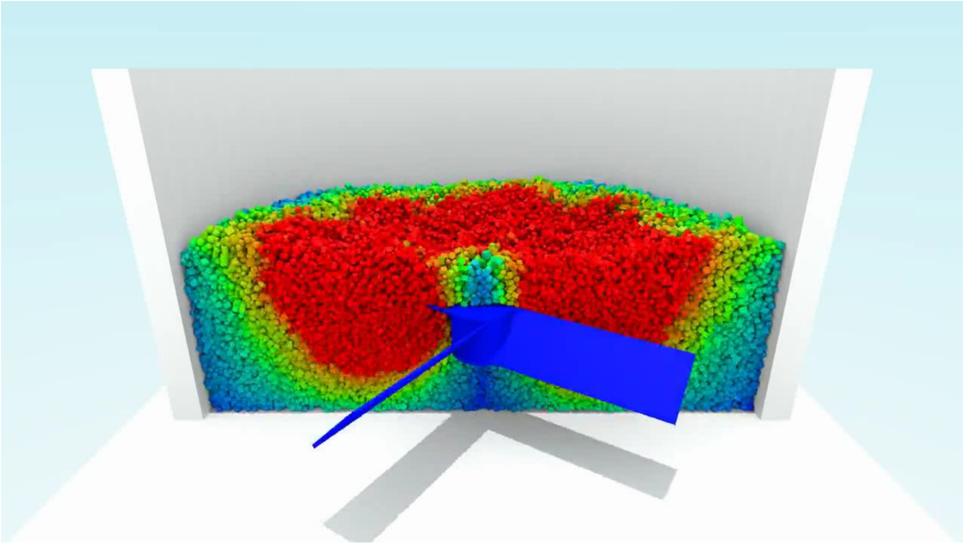


Problem:

HOW TO SOLVE A CCP?

Example of DVI with large CCPs

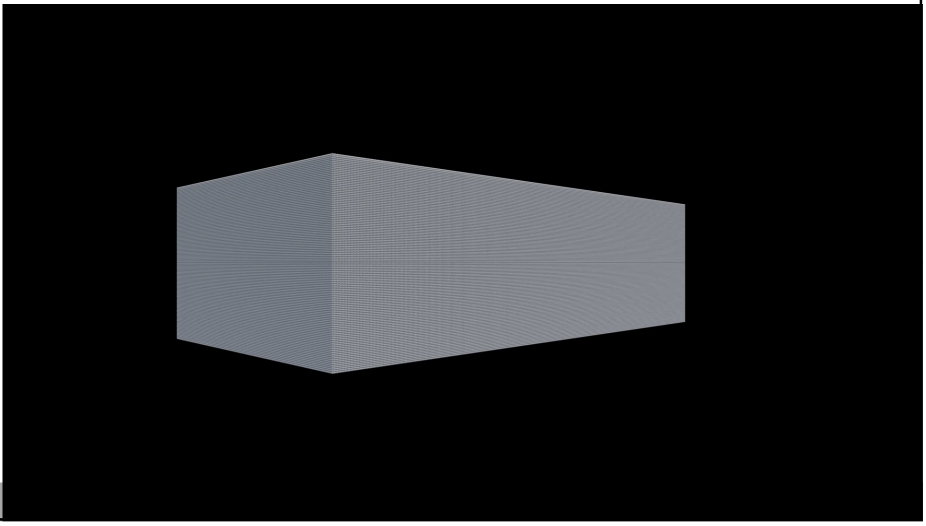
ProjectChrono benchmark (simulation and rendering by H. Mazhar 2015)



Example of DVI with large CCPs

- 10 millions of bodies
- 60 million of contacts

ProjectChrono – Chrono::Parallel benchmark (SBEL 2015)



Solve CCP using projected fixed-point iteration

- We outline a projected iteration that solves the Cone Complementarity Problem:

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

- This is a modified version of a SOR fixed point iteration [Mangasarian]

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

- 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 $\Pi_{\Upsilon} : \mathbb{R}^{n\varepsilon} \longrightarrow \mathbb{R}^{n\varepsilon}$

Solve CCP using projected fixed-point iteration

- 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 $(x^{r+1} - x^{r})^{T}\left((\lambda\omega B^{r})^{-1} + K^{r} - \frac{N}{2}\right)(x^{r+1} - x^{r}) \geq \beta \left\|x^{r+1} - x^{r}\right\|^{2}$.

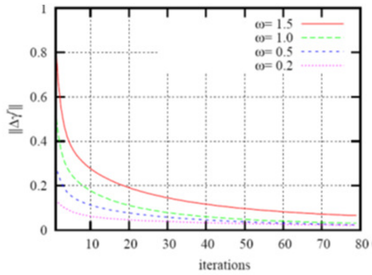
Always satisfied in multibody systems

Free choiche of the B^r matrix

Use • factor and the B^r matrix to adjust this

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

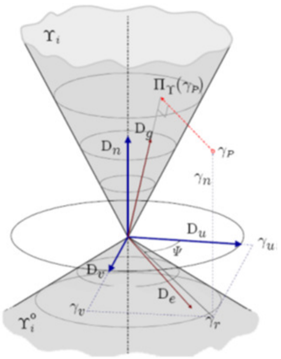
- The method produces a bounded sequence with an unique accumulation point.



Solve CCP using projected fixed-point iteration

- The projection operator must be non-extensive, i.e. lipschitzian with $\|f(a)-f(b)\| \leq \|a-b\|$
- *For each frictional contact constraint:*

$$\Pi_\Upsilon = \left\{ \Pi_{\Upsilon_1}(\boldsymbol{\gamma}_1)^T, \dots \Pi_{\Upsilon_{n_\mathcal{A}}}(\boldsymbol{\gamma}^{n_\mathcal{A}})^T, \Pi_b^1(\gamma_b^1), \dots, \Pi_b^{n_\mathcal{B}}(\gamma_b^{n_\mathcal{B}}) \right\}^T$$



- *For each bilateral constraint, simply do nothing.*
- The complete operator:

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

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

Solve CCP using projected fixed-point iteration

- Development of an efficient algorithm for fixed point iteration:

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

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

- *At each r-th iteration:*

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

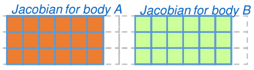
$$\gamma^{i,r+1} = \lambda \Pi_{\Upsilon^i}\left(\boldsymbol{\delta}^{i,r+1}\right) + (1-\lambda)\gamma^{i,r}$$

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

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

Jacobian for body A
Jacobian for body B

γ_b^i = 🟩 b_b^i = 🟪

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

Solve CCP using projected fixed-point iteration

- Even better, in incremental form:

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

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

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:*

$$\boldsymbol{\delta}^{i,r+1} = \left(\boldsymbol{\gamma}^{i,r} - \omega\eta^i \left(D^{i,T} \boldsymbol{v}^r + \boldsymbol{b}^i \right) \right);$$

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

$$\Delta\boldsymbol{\gamma}^{i,r+1} = \boldsymbol{\gamma}^{i,r+1} - \boldsymbol{\gamma}^{i,r} \ ;$$

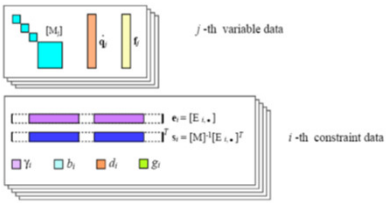
$$\boldsymbol{v} := \boldsymbol{v} + M^{-1} D^i \Delta\boldsymbol{\gamma}^{i,r+1}$$

Loop on all i-th constraints

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

Solve CCP using projected fixed-point iteration

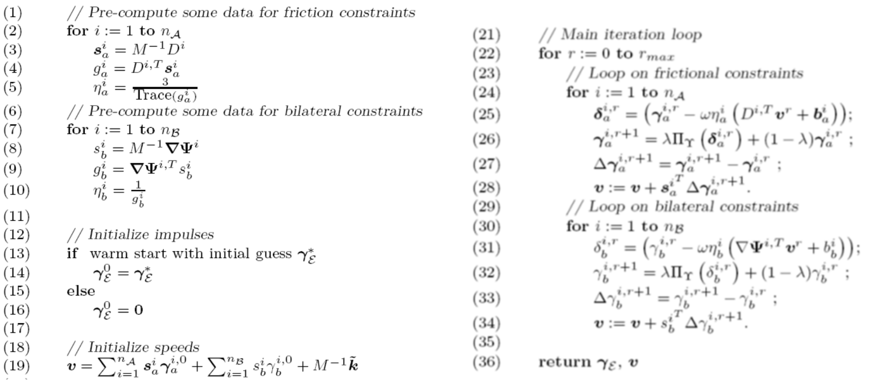
- Development of an efficient algorithm for fixed point iteration:



- *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*

Solve CCP using projected fixed-point iteration

Pseudocode:

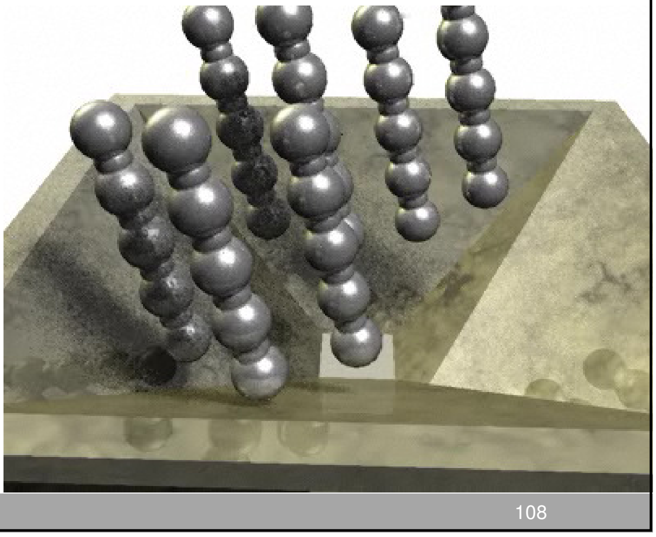


```
// Pre-compute some data for friction constraints
for $i := 1$ to $n_{\mathcal{A}}$
    $\boldsymbol{s}_a^i = M^{-1} D^i$
    $g_a^i = D^{i,T} \boldsymbol{s}_a^i$
    $\eta_a^i = \frac{3}{\mathrm{Trace}(g_a^i)}$
// Pre-compute some data for bilateral constraints
for $i := 1$ to $n_{\mathcal{B}}$
    $s_b^i = M^{-1} \boldsymbol{\nabla\Psi}^i$
    $g_b^i = \boldsymbol{\nabla\Psi}^{i,T} s_b^i$
    $\eta_b^i = \frac{1}{g_b^i}$

// Initialize impulses
if  warm start with initial guess $\boldsymbol{\gamma}_{\mathcal{E}}^*$
    $\boldsymbol{\gamma}_{\mathcal{E}}^0 = \boldsymbol{\gamma}_{\mathcal{E}}^*$
else
    $\boldsymbol{\gamma}_{\mathcal{E}}^0 = \mathbf{0}$

// Initialize speeds
$\boldsymbol{v} = \sum_{i=1}^{n_{\mathcal{A}}} \boldsymbol{s}_a^i \boldsymbol{\gamma}_a^{i,0} + \sum_{i=1}^{n_{\mathcal{B}}} s_b^i \gamma_b^{i,0} + M^{-1} \tilde{\boldsymbol{k}}$

// Main iteration loop
for $r := 0$ to $r_{max}$
    // Loop on frictional constraints
    for $i := 1$ to $n_{\mathcal{A}}$
        $\boldsymbol{\delta}_a^{i,r} = \left(\boldsymbol{\gamma}_a^{i,r} - \omega\eta_a^i \left(D^{i,T} \boldsymbol{v}^r + \boldsymbol{b}_a^i\right)\right);$
        $\boldsymbol{\gamma}_a^{i,r+1} = \lambda \Pi_{\Upsilon}\left(\boldsymbol{\delta}_a^{i,r}\right) + (1-\lambda)\boldsymbol{\gamma}_a^{i,r}$ ;
        $\Delta\boldsymbol{\gamma}_a^{i,r+1} = \boldsymbol{\gamma}_a^{i,r+1} - \boldsymbol{\gamma}_a^{i,r}$ ;
        $\boldsymbol{v} := \boldsymbol{v} + \boldsymbol{s}_a^{i^T} \Delta\boldsymbol{\gamma}_a^{i,r+1}.$
    // Loop on bilateral constraints
    for $i := 1$ to $n_{\mathcal{B}}$
        $\delta_b^{i,r} = \left(\gamma_b^{i,r} - \omega\eta_b^i \left(\nabla\boldsymbol{\Psi}^{i,T} \boldsymbol{v}^r + b_b^i\right)\right);$
        $\gamma_b^{i,r+1} = \lambda \Pi_{\Upsilon}\left(\delta_b^{i,r}\right) + (1-\lambda)\gamma_b^{i,r}$ ;
        $\Delta\gamma_b^{i,r+1} = \gamma_b^{i,r+1} - \gamma_b^{i,r}$ ;
        $\boldsymbol{v} := \boldsymbol{v} + s_b^{i^T} \Delta\gamma_b^{i,r+1}.$

return $\boldsymbol{\gamma}_{\mathcal{E}}$, $\boldsymbol{v}$
```

Examples

Test with

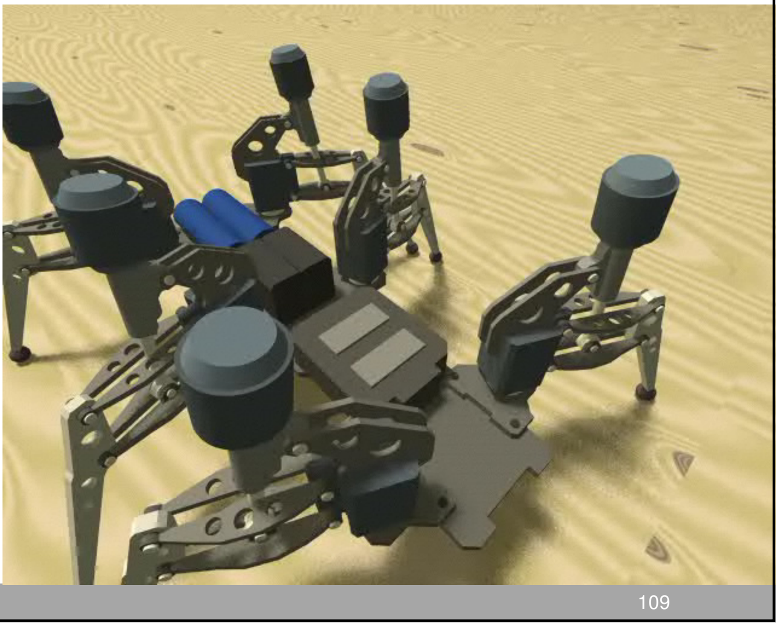
- Bilateral constraints: spherical joints between the balls
- Unilateral constraints: collisions + min/max rotation limits for balls
- No friction



Model

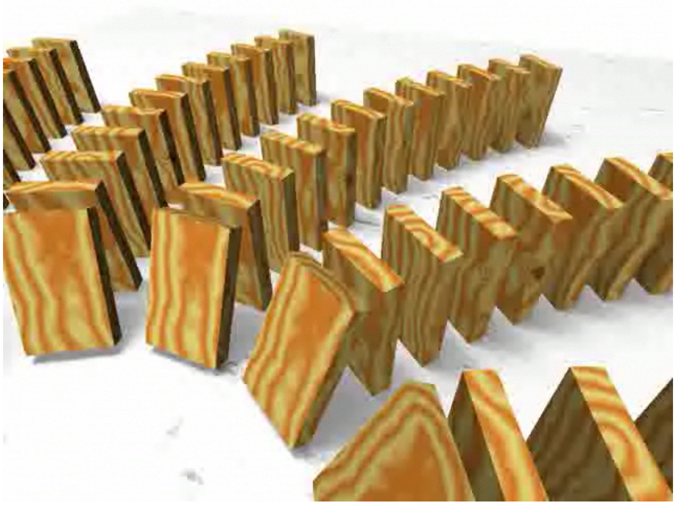
Test with:

- bilateral constraints
- motors
- contacts

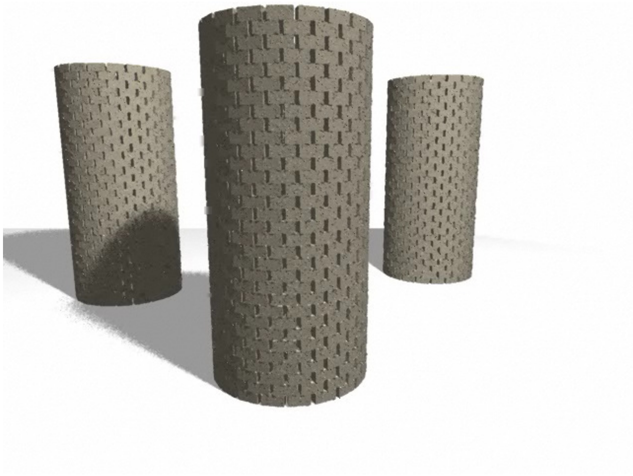


Simulation of a 6-legs robot with 42 joints, 38 rigid bodies, 12 motors, 6 contacts (A.Tasora)

Examples



Examples

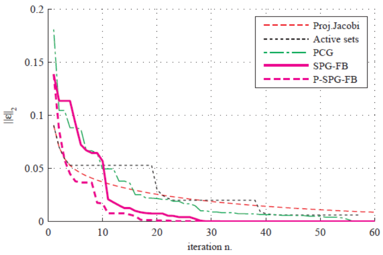


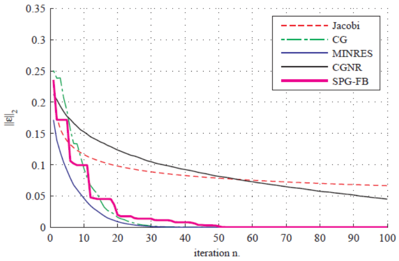
Better solver?

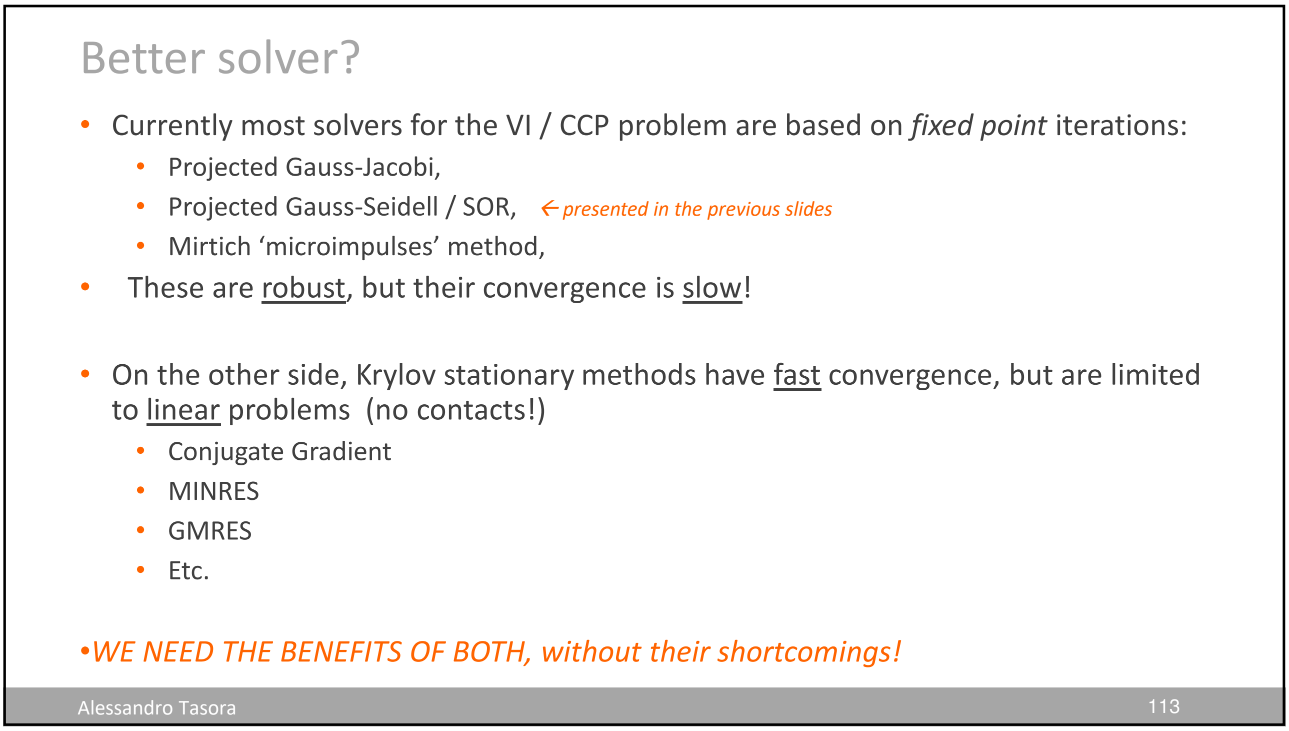
- The projected fixed point method has slow convergence!

- New methods under development

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





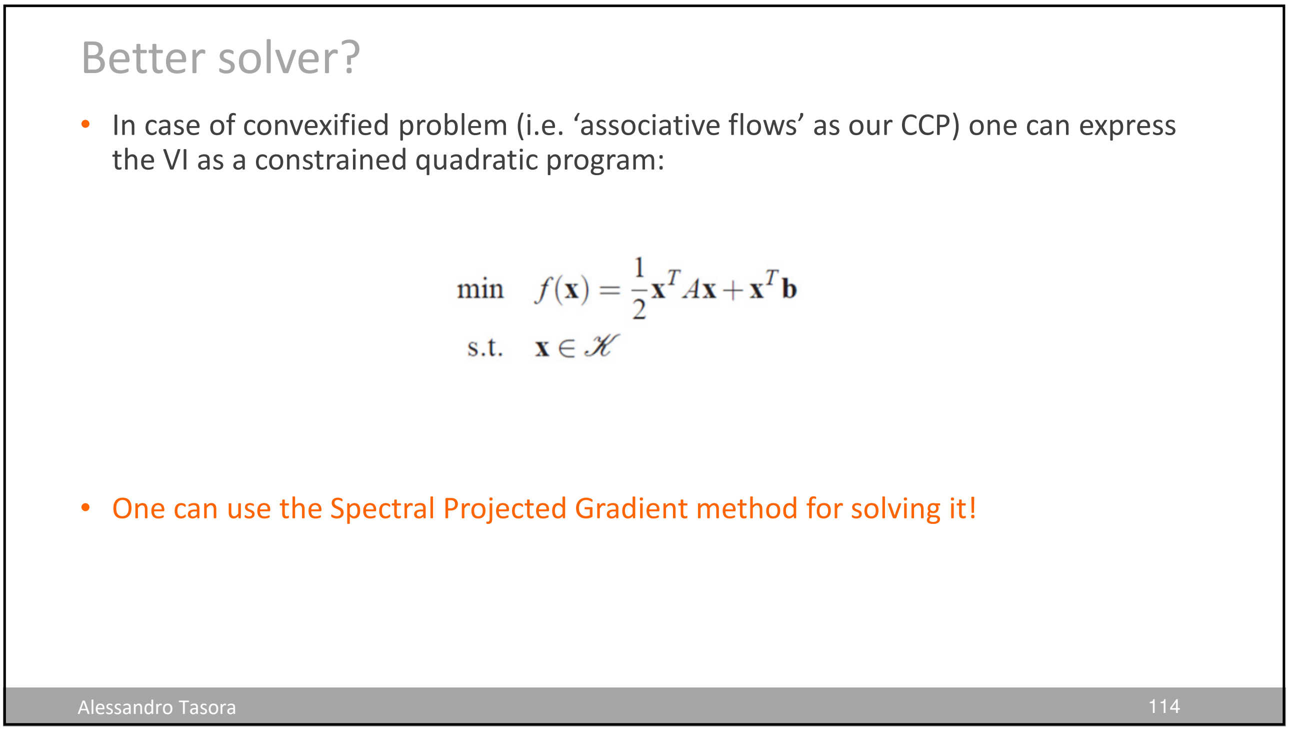


Better solver?

- 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 solver?

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

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

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

The P-SPG-FB first order method

Our P-SPG-FB algorithm:

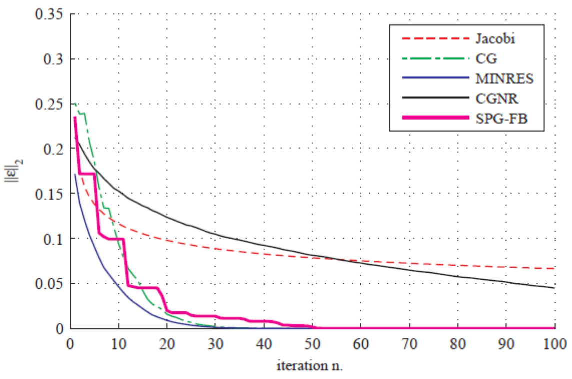
- Based on the SPG method
  - Extends Barzilai-Borwein spectral iteration
  - Uses GLL non-monotone line search
- Improvements:
  - Uses alternating step sizes
  - Uses diagonal preconditioning (with isothropic cone scaling) $P = \overline{\mathrm{diag}(A)}$
  - Supports premature termination with fall-back strategy (FB)
- Draws on three main computational primitives:
  - Matrix X vector multiplication
  - Vector inner product
  - Projection onto Lorentz cones

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

Alessandro Tasora

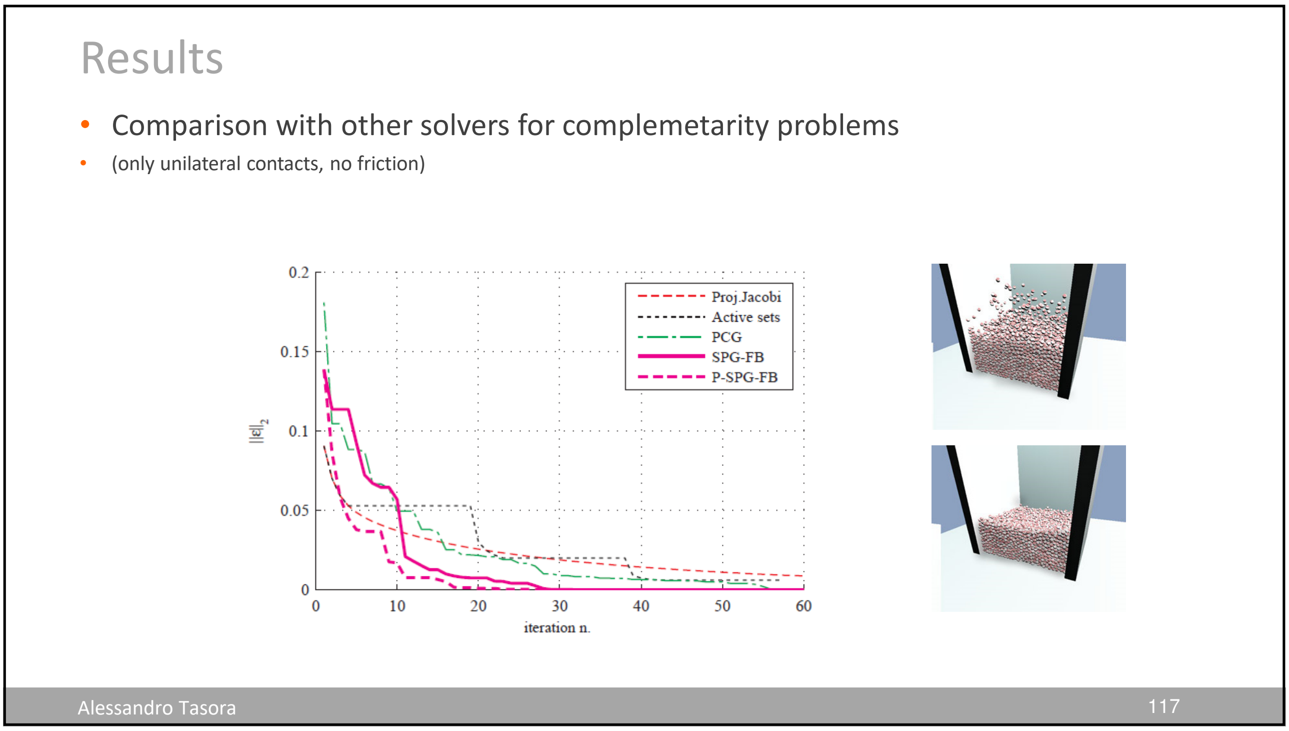
Results

- Comparison with other Krylov solvers for simple <u>linear</u> case
- (only bilateral constraints):



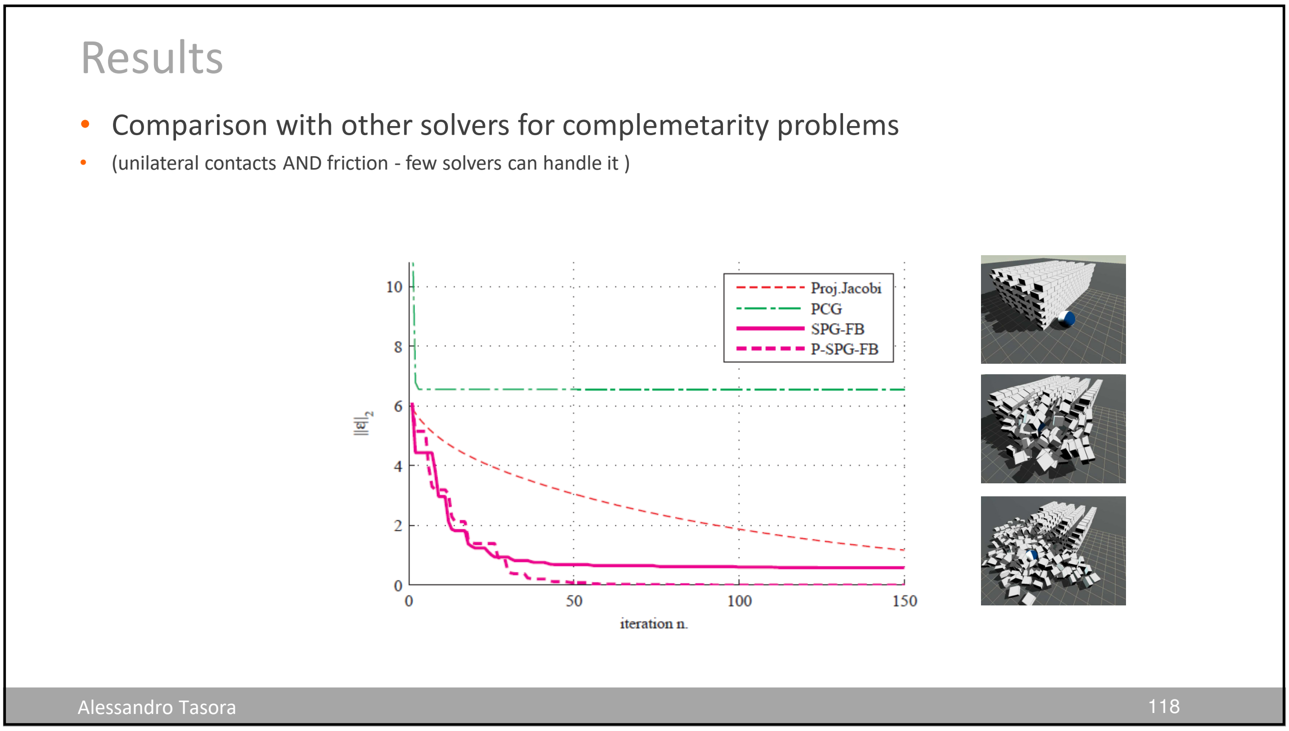
Alessandro Tasora

116



Results

- Comparison with other solvers for complemetarity problems
- (only unilateral contacts, no friction)

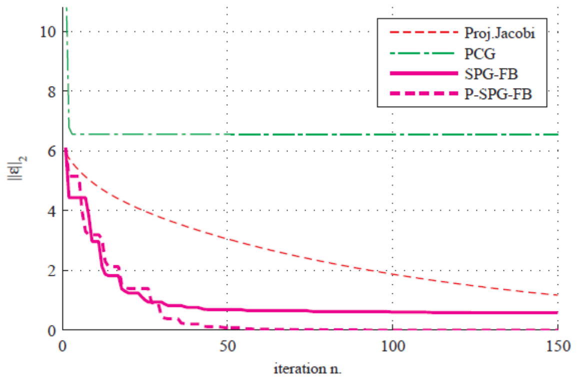


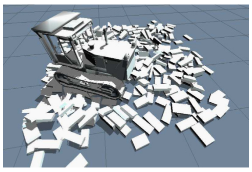
Results

- Comparison with other solvers for complemetarity problems
- (unilateral contacts AND friction - few solvers can handle it)

Results

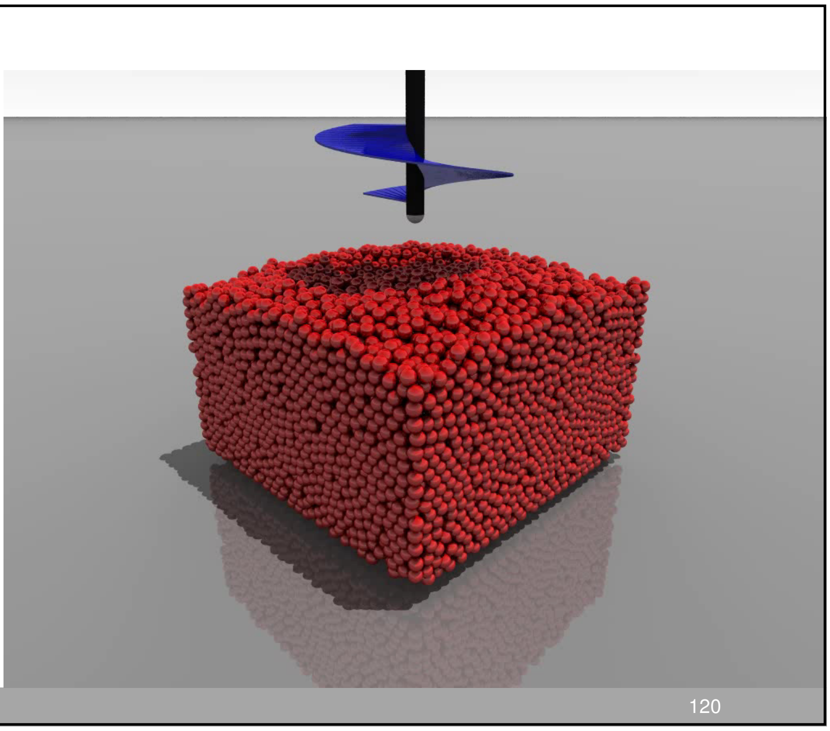
- Effect of preconditioning:





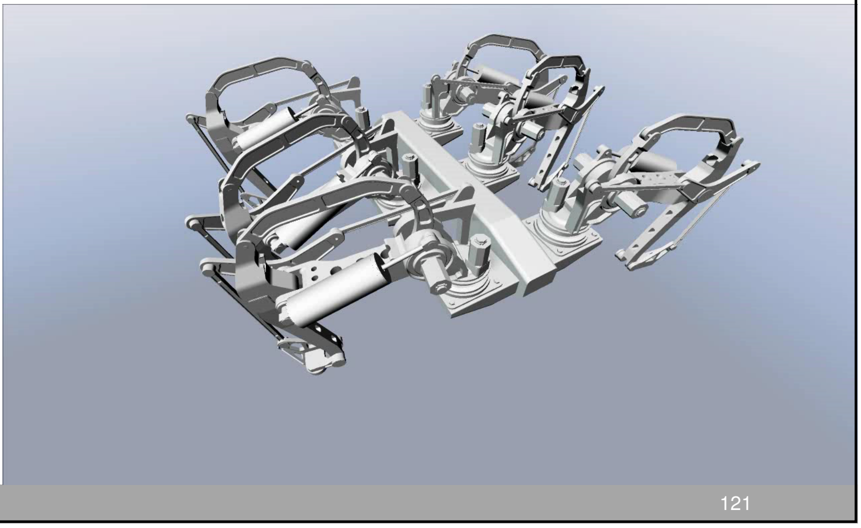
Example

ProjectChrono test, SBEL



Example

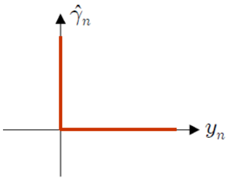
Walking robot with contacts and bilateral constraint



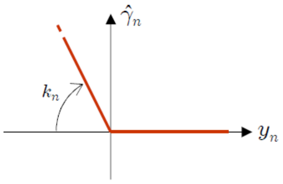
ProjectChrono test - A.Tasora, 2019

DVI advanced contact laws

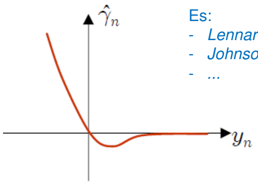
Rigid contact:



Compliant contact:

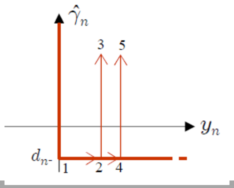


Nonlinear, with cohesion:



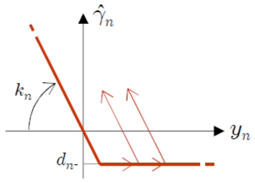
Es:
- *Lennard-Jones*
- *Johnson-Kendall-Roberts*
- ...

Rigid, with plastic cohesion

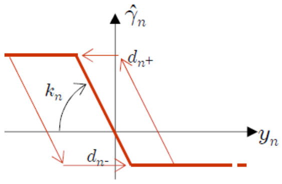


DVI advanced contact laws

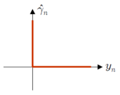
Compliant, plastic cohesion



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

$$\widehat{\gamma}^i_{\mathcal{A}} = \{\widehat{\gamma}^i_n, \widehat{\gamma}^i_u, \widehat{\gamma}^i_w\}^T$$

- Inclusion in yield surface:

$$\widehat{\gamma}^i_{\mathcal{A}} \in \widehat{\Upsilon}^i$$

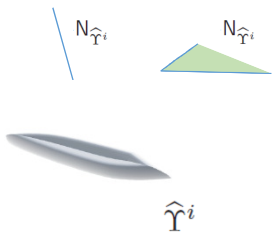
- Prandtl-Reuss-like assumption

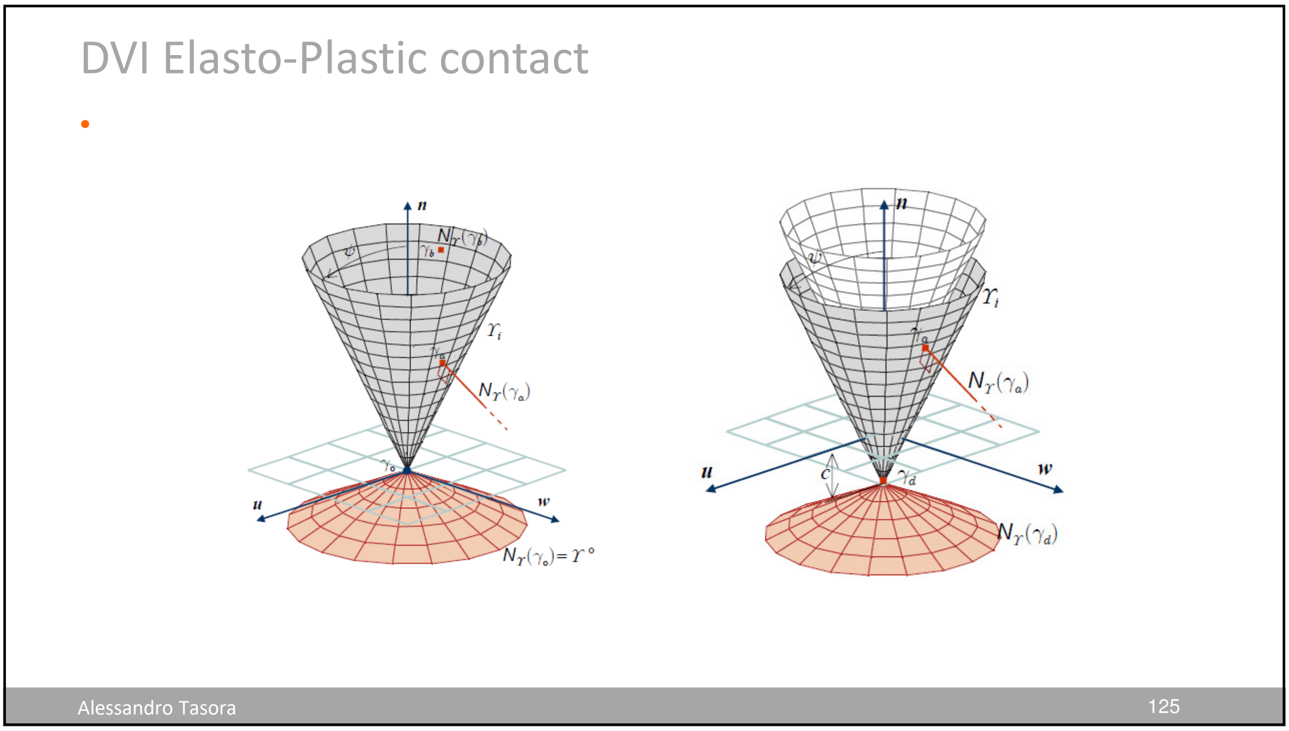
on displacements y

$$\boldsymbol{y}^i = \boldsymbol{y}^i_E + \boldsymbol{y}^i_P$$

- Associated flow assumption:
- *The increment to the plastic flow is orthogonal to the yield surface*

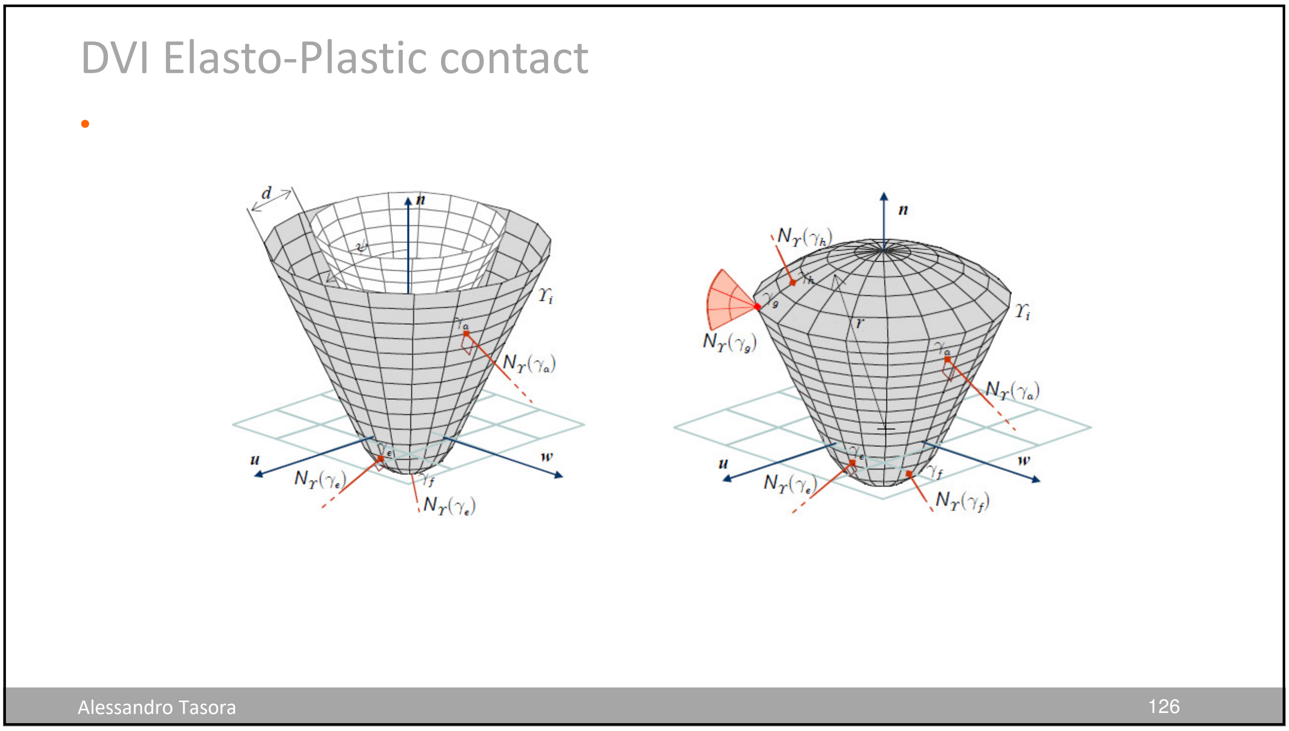
$$\dot{\boldsymbol{y}}^i_P \in -\mathsf{N}_{\widehat{\Upsilon}^i}(\widehat{\gamma}^i_{\mathcal{A}})$$


$\mathsf{N}_{\widehat{\Upsilon}^i}$
$\mathsf{N}_{\widehat{\Upsilon}^i}$
$\widehat{\Upsilon}^i$



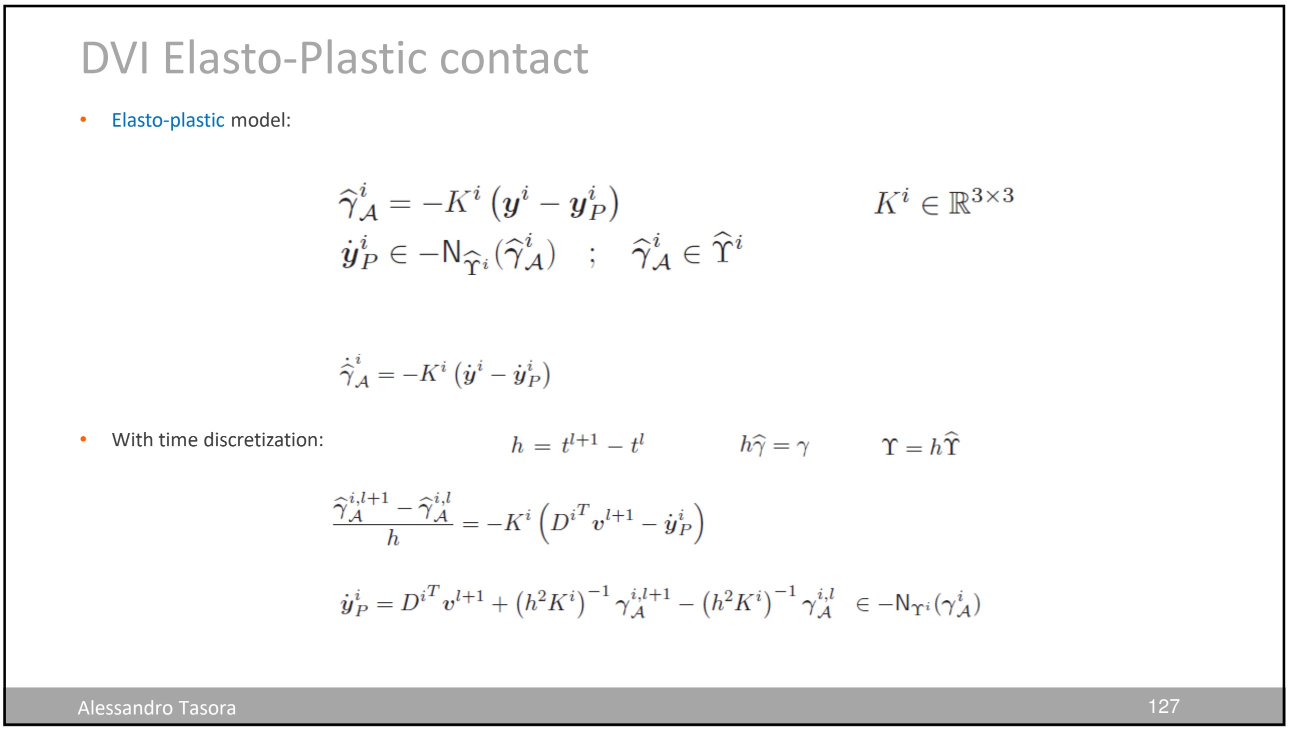
DVI Elasto-Plastic contact

-



DVI Elasto-Plastic contact

-



DVI Elasto-Plastic contact

- Elasto-plastic model:

$$\widehat{\gamma}^i_{\mathcal{A}} = -K^i\left(\boldsymbol{y}^i - \boldsymbol{y}^i_P\right) \qquad\qquad K^i \in \mathbb{R}^{3\times 3}$$

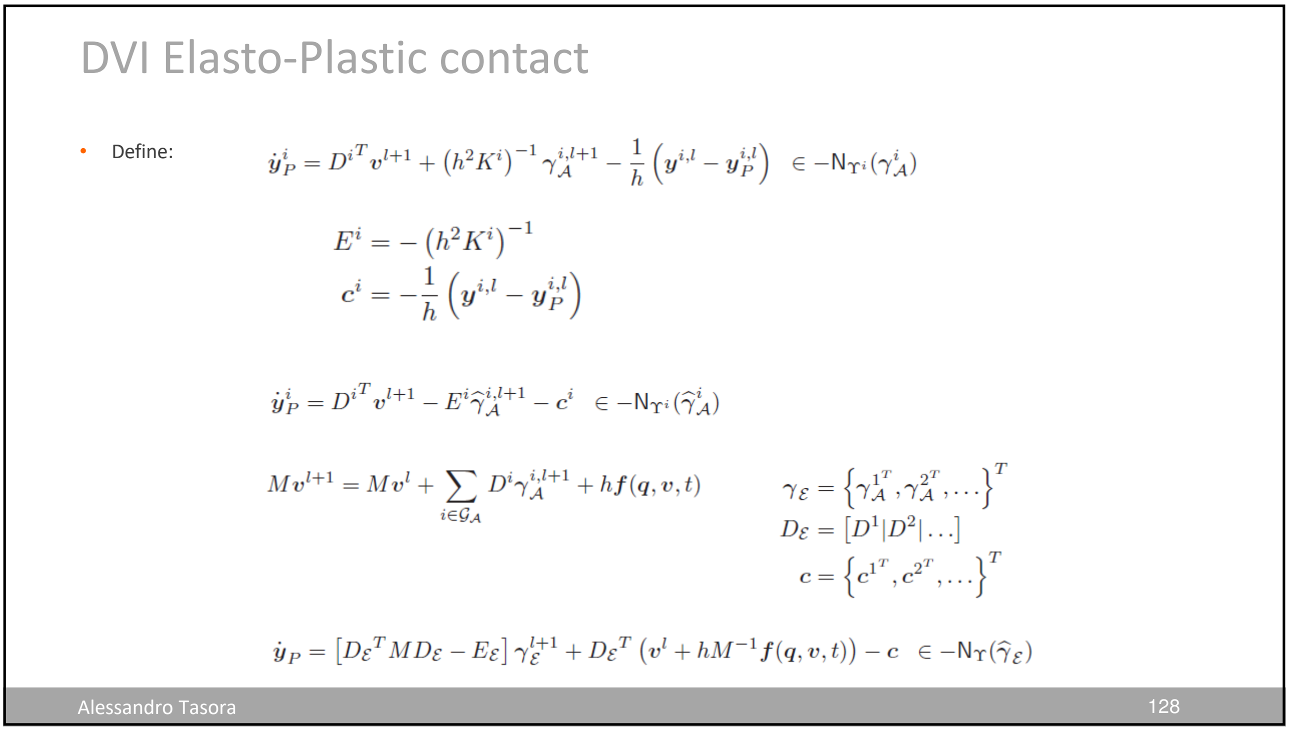
$$\dot{\boldsymbol{y}}^i_P \in -\mathsf{N}_{\widehat{\Upsilon}^i}(\widehat{\gamma}^i_{\mathcal{A}}) \quad ; \quad \widehat{\gamma}^i_{\mathcal{A}} \in \widehat{\Upsilon}^i$$

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

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

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

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



DVI Elasto-Plastic contact

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

$$E^i = -\left(h^2K^i\right)^{-1}$$

$$\boldsymbol{c}^i = -\frac{1}{h}\left(\boldsymbol{y}^{i,l} - \boldsymbol{y}^{i,l}_P\right)$$

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

$$M\boldsymbol{v}^{l+1} = M\boldsymbol{v}^l + \sum_{i\in\mathcal{G}_{\mathcal{A}}} D^i\gamma^{i,l+1}_{\mathcal{A}} + h\boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t) \qquad \begin{aligned} \gamma_{\mathcal{E}} &= \left\{\gamma^{1^T}_{\mathcal{A}}, \gamma^{2^T}_{\mathcal{A}}, \dots\right\}^T \\ D_{\mathcal{E}} &= \left[D^1|D^2|\dots\right] \\ \boldsymbol{c} &= \left\{\boldsymbol{c}^{1^T}, \boldsymbol{c}^{2^T}, \dots\right\}^T \end{aligned}$$

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

DVI Elasto-Plastic contact

- Posing:

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

- 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:

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

DVI Elasto-Plastic contact

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

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

$$\Updownarrow$$

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

$$\Updownarrow$$

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

DVI Visco-Elasto-Plastic contact

- By introducing also viscous damping, one gets the model

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

- Again one obtains a VI, this time with:

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

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

DVI Visco-Elasto-Plastic contact

- With Raleygh damping → simplification

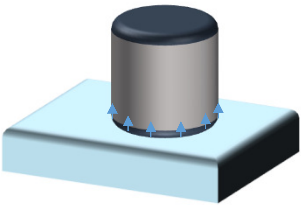
$$R^i = \alpha^i_K K^i$$

- Obtaining:

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

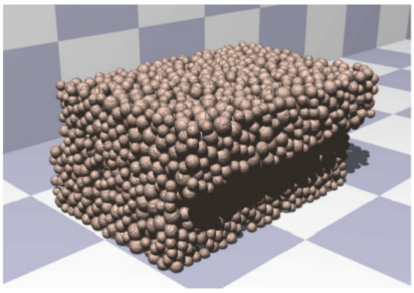
- NOTE
  the E term works as a Tykhonov regularization
  of the Schur complement

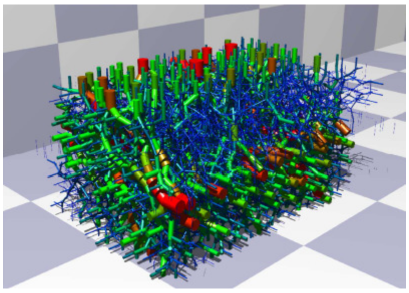
$$N = \left[D_{\mathcal{E}}{}^T M D_{\mathcal{E}} - E_{\mathcal{E}}\right]$$



Examples

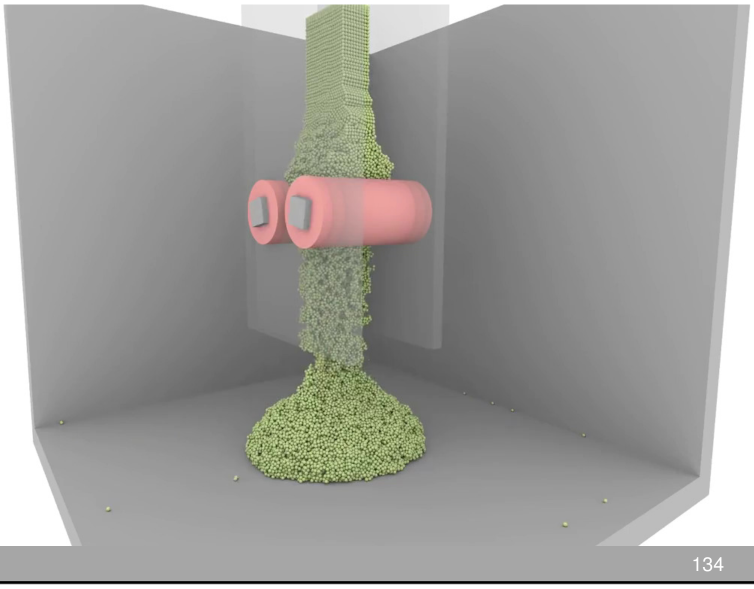
- Granular flows (shear test)



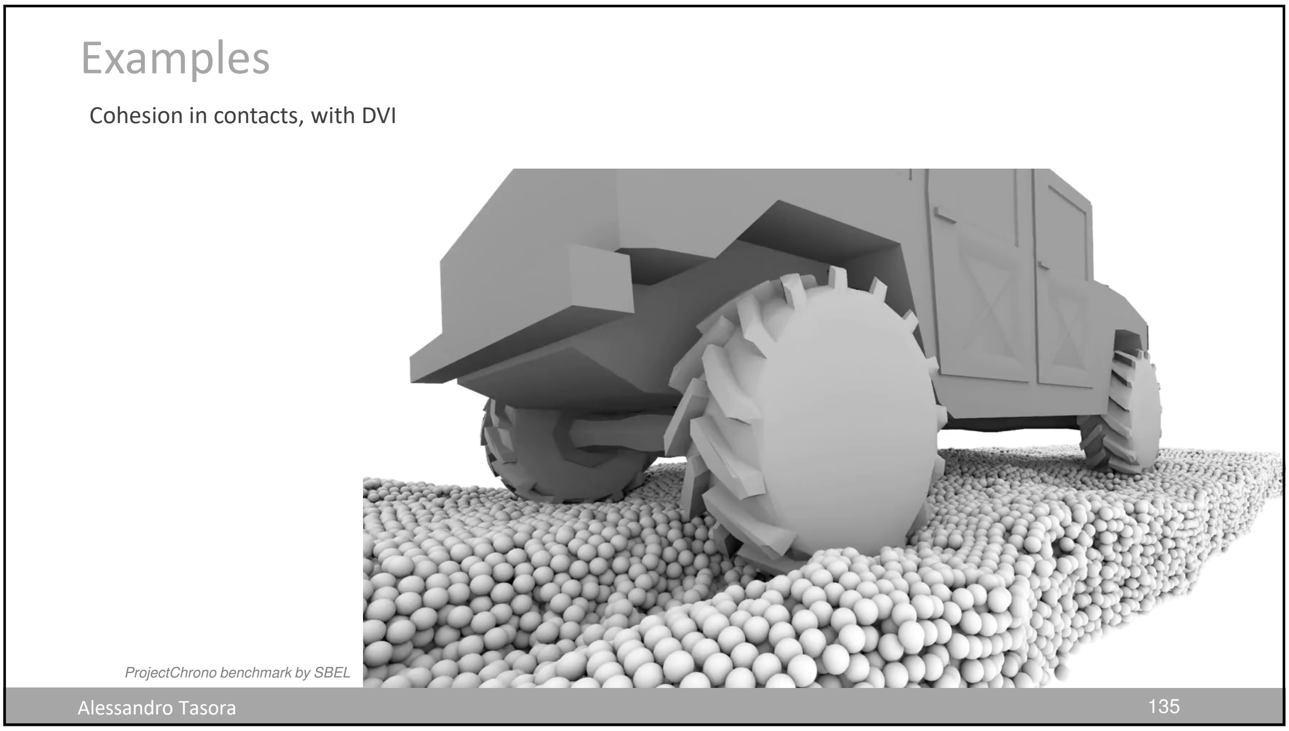


Examples

Cohesion in contacts, with DVI



Simulation by H.Mazhar, 2013



Examples

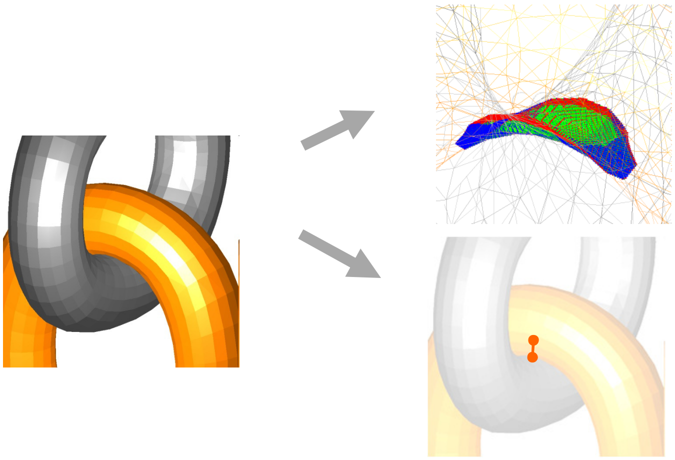
Cohesion in contacts, with DVI

ProjectChrono benchmark by SBEL

6. COLLISION DETECTION

Collision detection

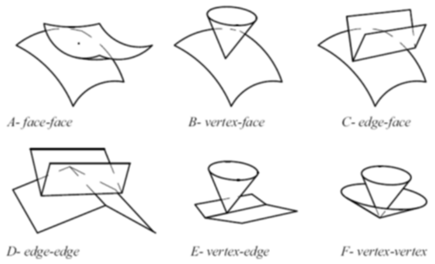
- Still one of the hardest problems of computational geometry
- Problem: find points or areas/volumes of contact between two shapes



Collision detection

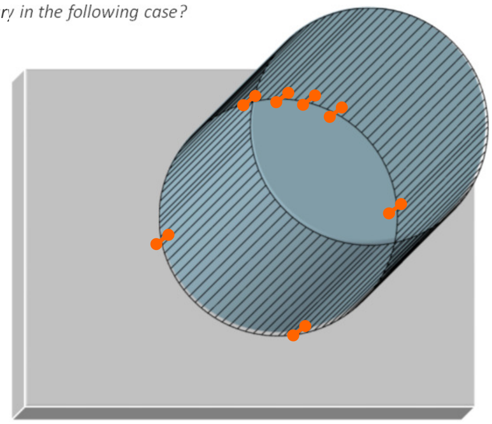
- Approaches based on areas/volumes fit better in stiffness-based contact models, are more related to physics, but..
- approaches based on points are much faster!

- Different sub-problems depending on shape’s topological entities:



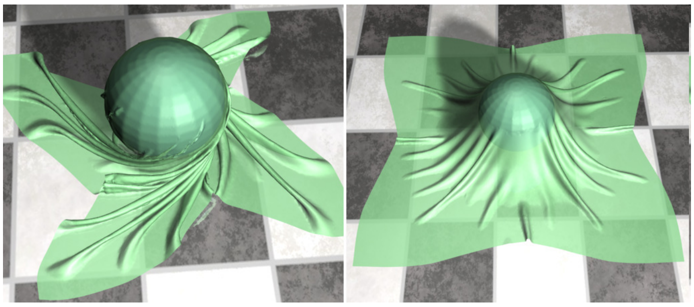
Collision detection

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



Collision detection

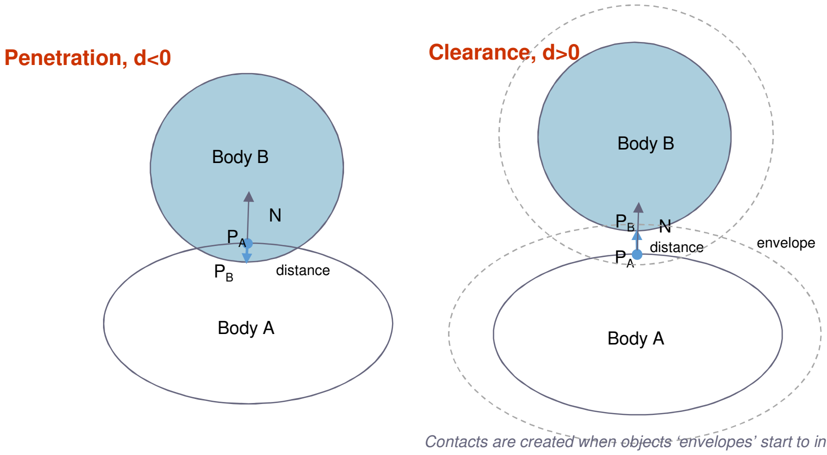
- Both point-based methods and area/volume methods can be used for deformable models
- Additional complication: deformable thin shells (may need CCD to avoid tangling – see later)



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

Collision detection

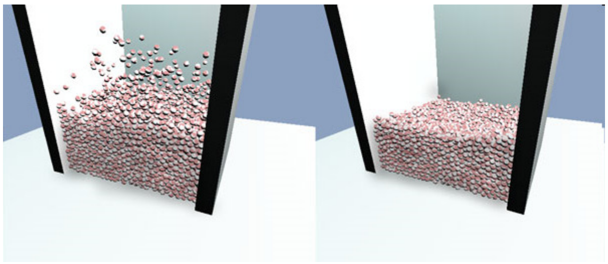
- We need: contact distance and normal between convex shapes
- Even potential contacts with distance>0 can be useful for the time integrator
- A tolerance (envelope) can be used to discard unlikely potential contacts



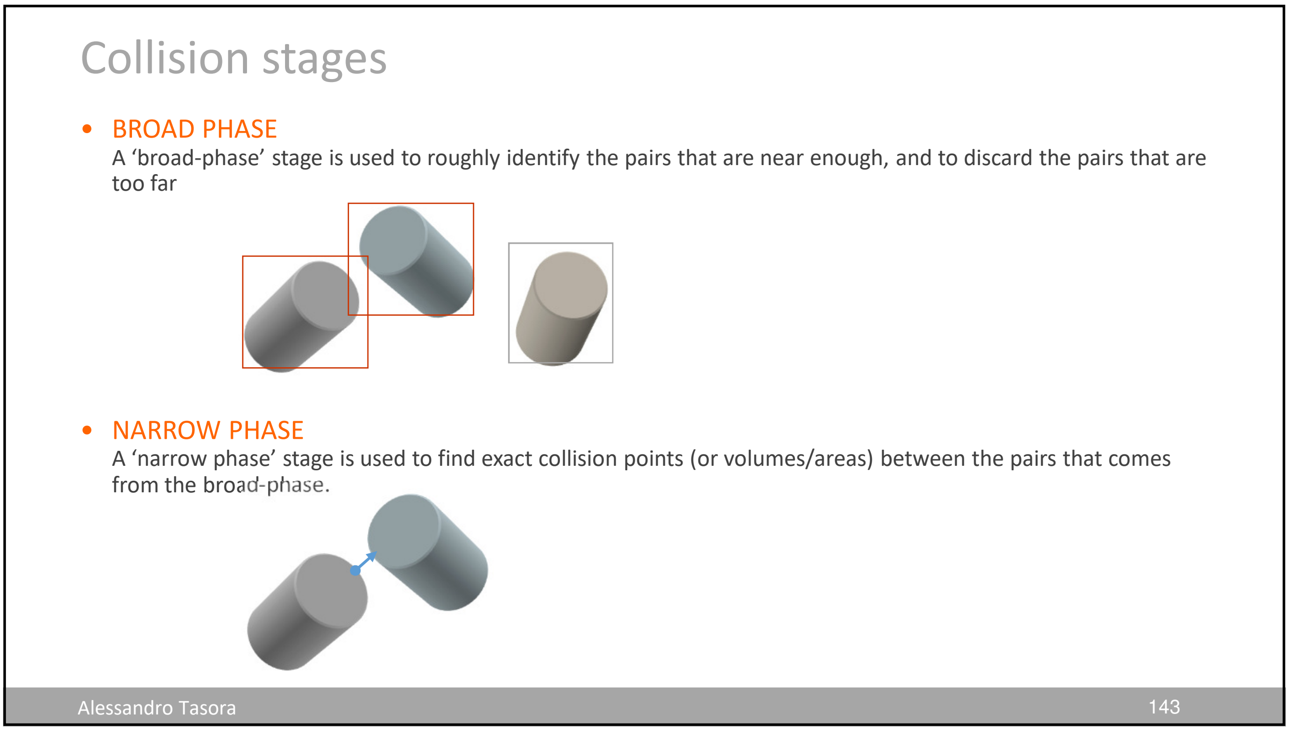
Contacts are created when objects 'envelopes' start to intersect

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!

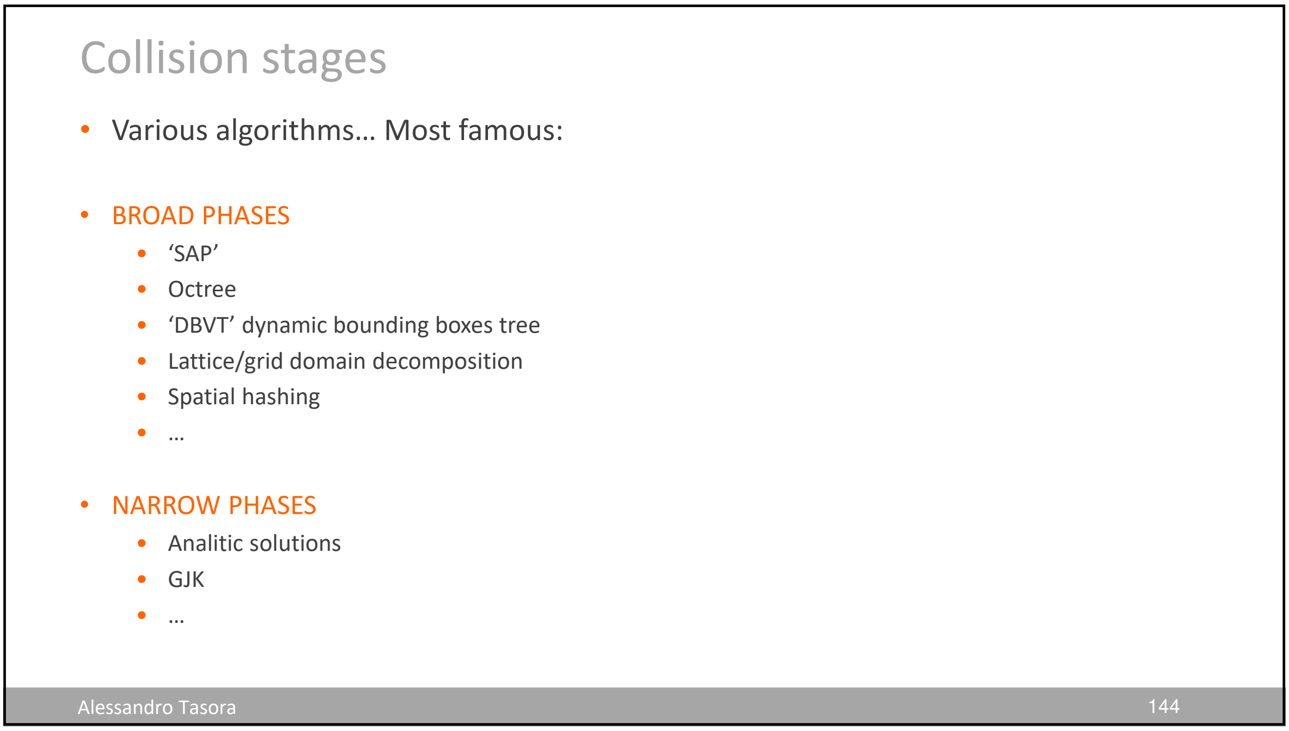


- Solution: check collision points between pairs of bodies that are 'near enough', using a preliminary filter to discard 'too far' pairs.
- This filter is called broad phase collision detection



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.



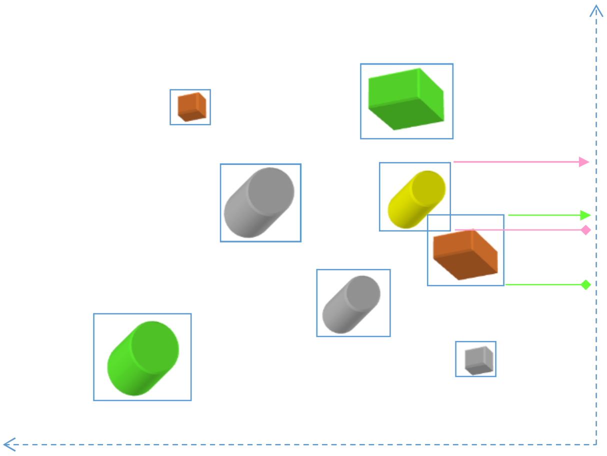
Collision stages

- Various algorithms… 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

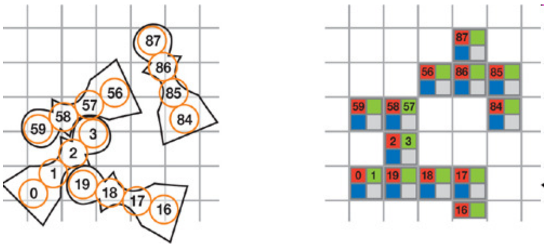
- SAP = ‘sweep-and-prune’
- Operates on AABB =
  Axis Aligned Bounding Boxes
- Basically, sorts X,Y,Z intervals
  of AABB and finds overlappings
- Optimization: use *quantized* AABB
- One of the most used
  and *fastest* broadphases!
- Not good for *deformable* objects



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!

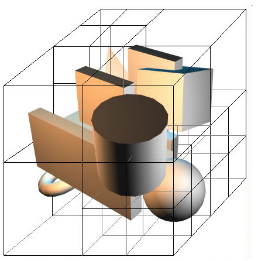


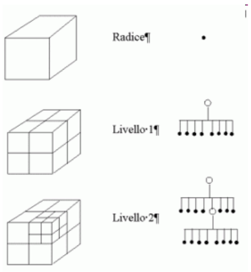
Broad phases

‘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.



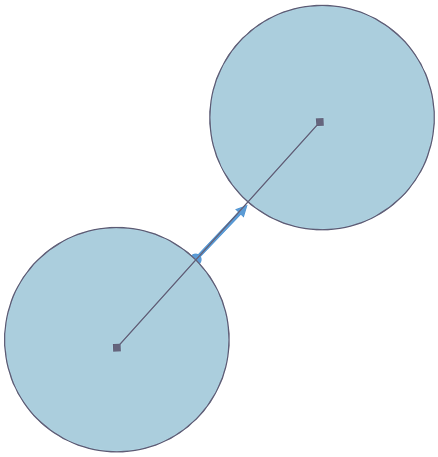


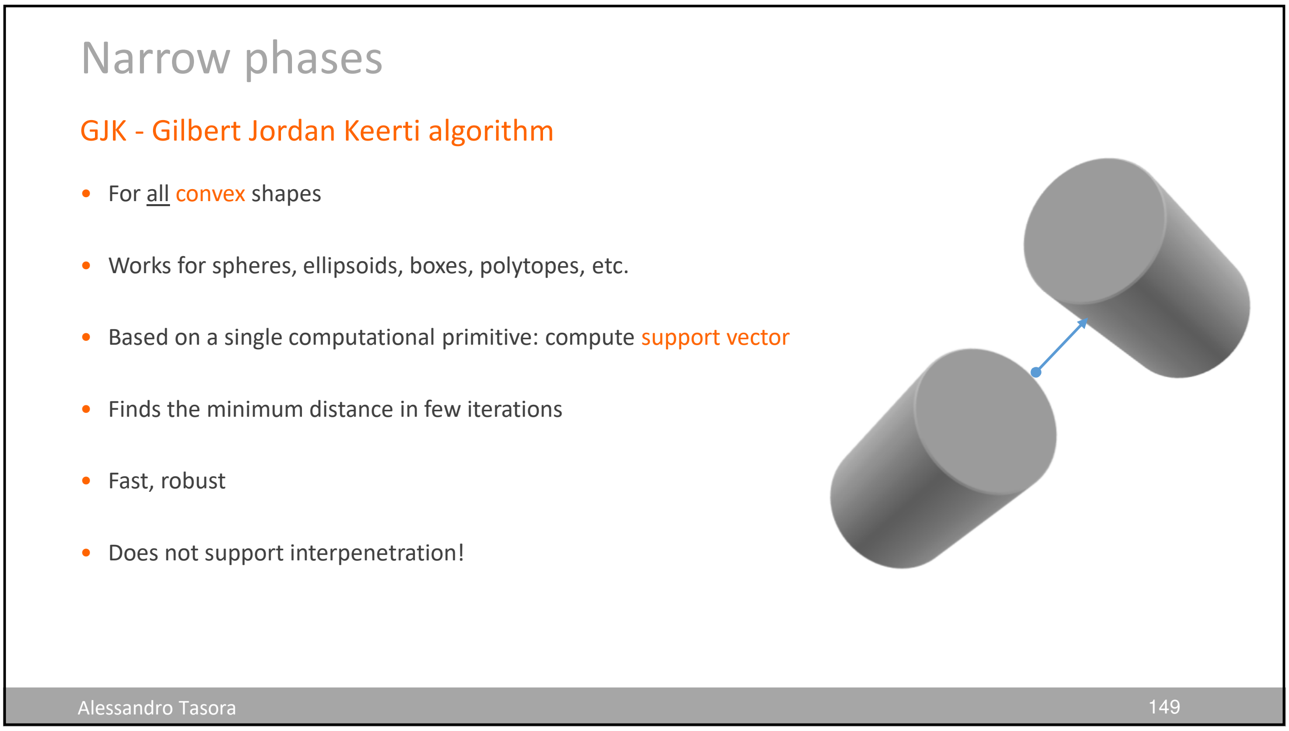
Narrow phases

Analytical solutions

- For limited number of primitives (es: sphere vs. sphere, sphere vs. plane)
- Fastest approach, but....
- Not always possible (es: analytical solution for ellipsoid vs.ellipsoid ?)
- The number of algorithms grows $O(n^2)$ with the number of primitives:

	Sphere	Cylinder	Cube	...
Sphere	Sphere-Sphere	Sphere-Cylinder	Sphere-Cube	...
Cylinder	Cylinder-Sphere	Cylinder-Cylinder	Cylinder-Cube	...
Cube	Cube-Sphere	Cube-Cylinder	Cube-Cube	...
....	...	...	...	...

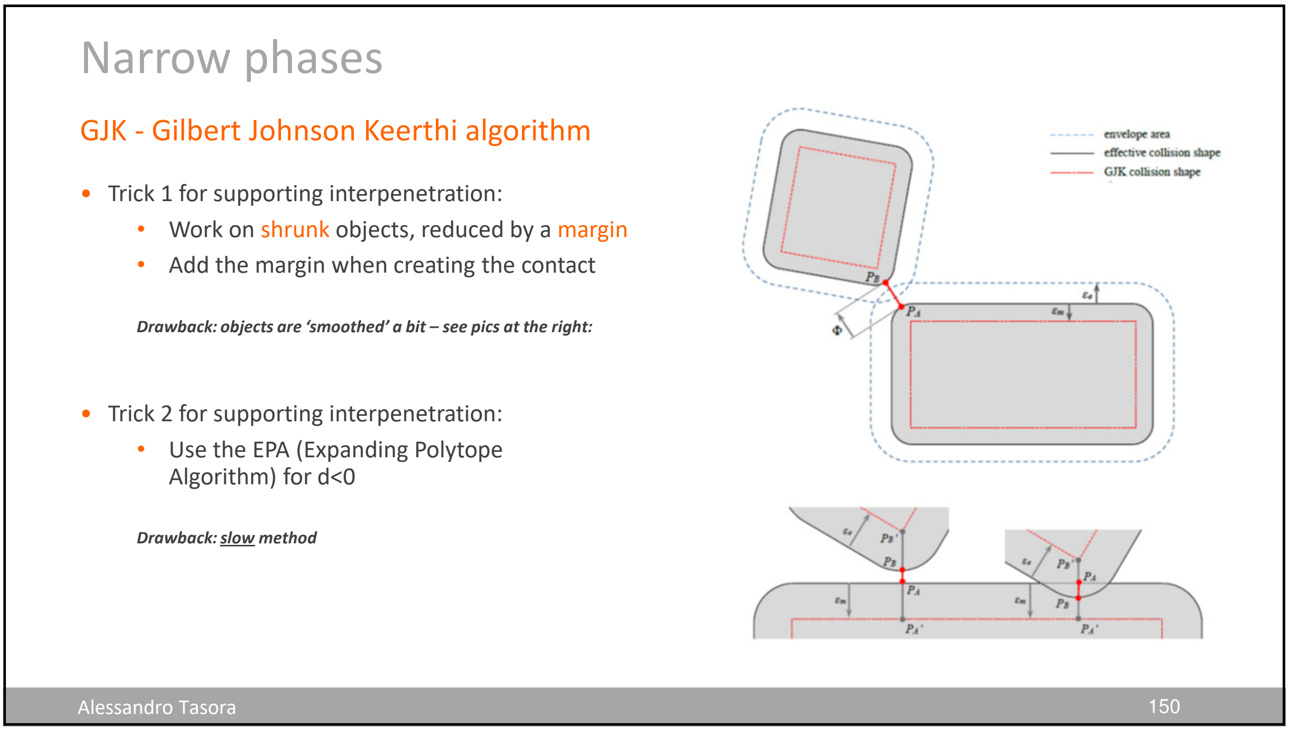




Narrow phases

GJK - Gilbert Jordan Keerti algorithm

- For all convex shapes
- Works for spheres, ellipsoids, boxes, polytopes, etc.
- Based on a single computational primitive: compute support vector
- Finds the minimum distance in few iterations
- Fast, robust
- Does not support interpenetration!



Narrow phases

GJK - Gilbert Johnson Keerthi algorithm

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

Drawback: objects are 'smoothed' a bit – see pics at the right:

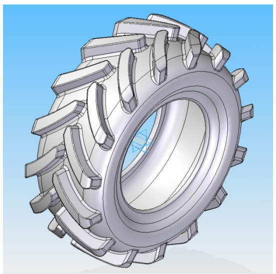
- Trick 2 for supporting interpenetration:
  - Use the EPA (Expanding Polytope Algorithm) for d<0

Drawback: slow method

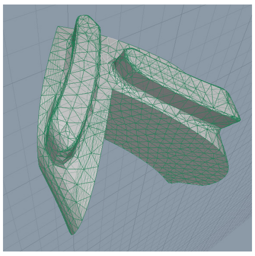
envelope area
effective collision shape
GJK collision shape
P_B
P_A
Φ
ε_e
ε_m
P_B'
P_A'

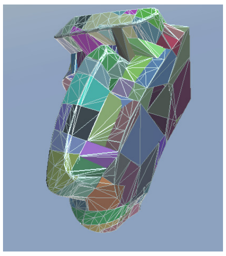
Narrow phases

GJK - Gilbert Johnson Keerthi algorithm



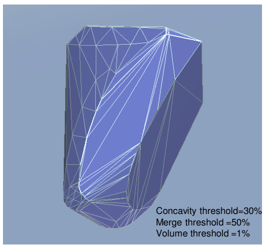
- What happens in case of concave shapes?
  - *Es. 'polygon soups',*
  - *meshes..*
- Possible solution: decompose concave shapes in many *convex* shapes, and process each one with GJK.

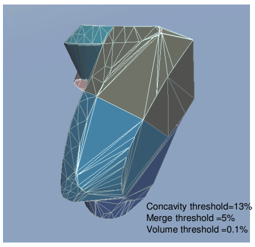


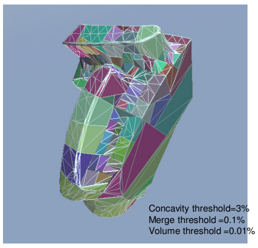


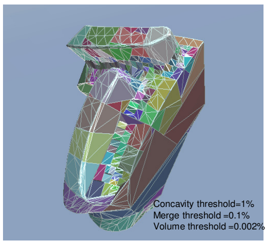
Narrow phases

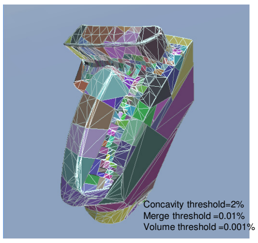
- Note: convex decomposition of concave shapes is not always easy…
- Sometimes, results are precise but not efficient, or viceversa.





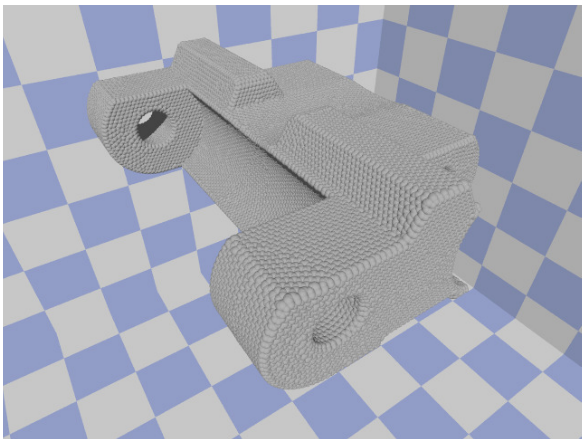






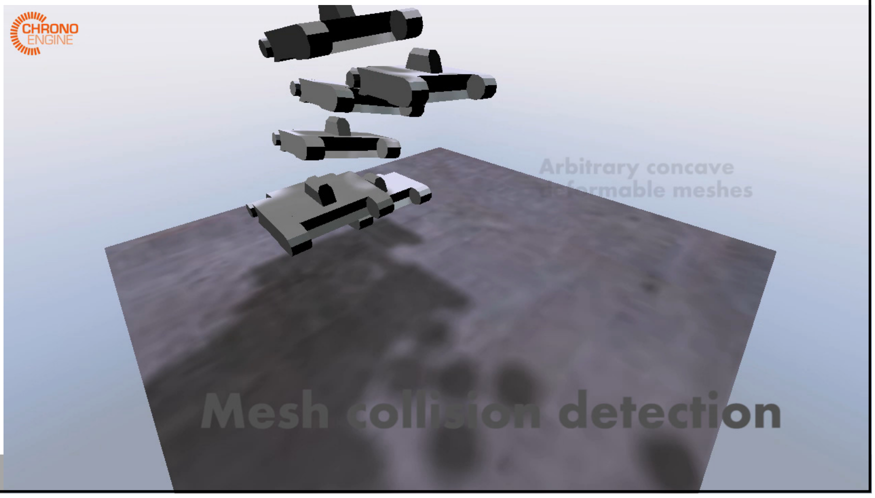
Narrow phases

- Another solution for concave shapes: spherical decomposition
- Lot of RAM is used
- Can work well with GPUs
- Issue: bumpy sliding



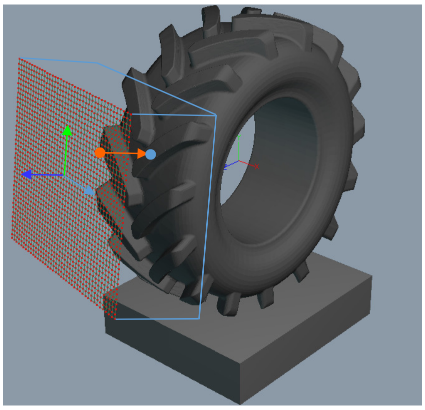
Narrow phases

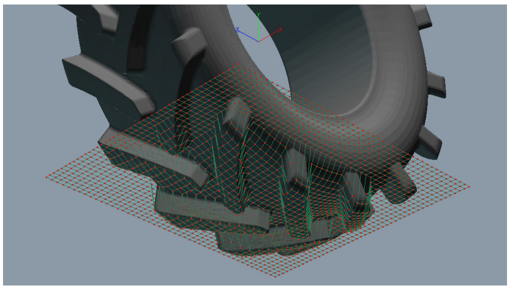
- Another solution for concave shapes: custom algorithm for triangle meshes
- Topological info (triangle connectivity) and watertight meshes needed for better robustness
- Implemented in ProjectChrono



Narrow phases

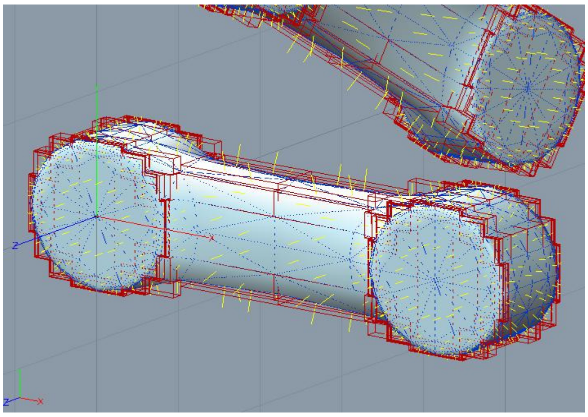
- In some special cases (ex. deformable soil) one can use simple workarounds:
  - Ex. raycasting methods
  - etc.

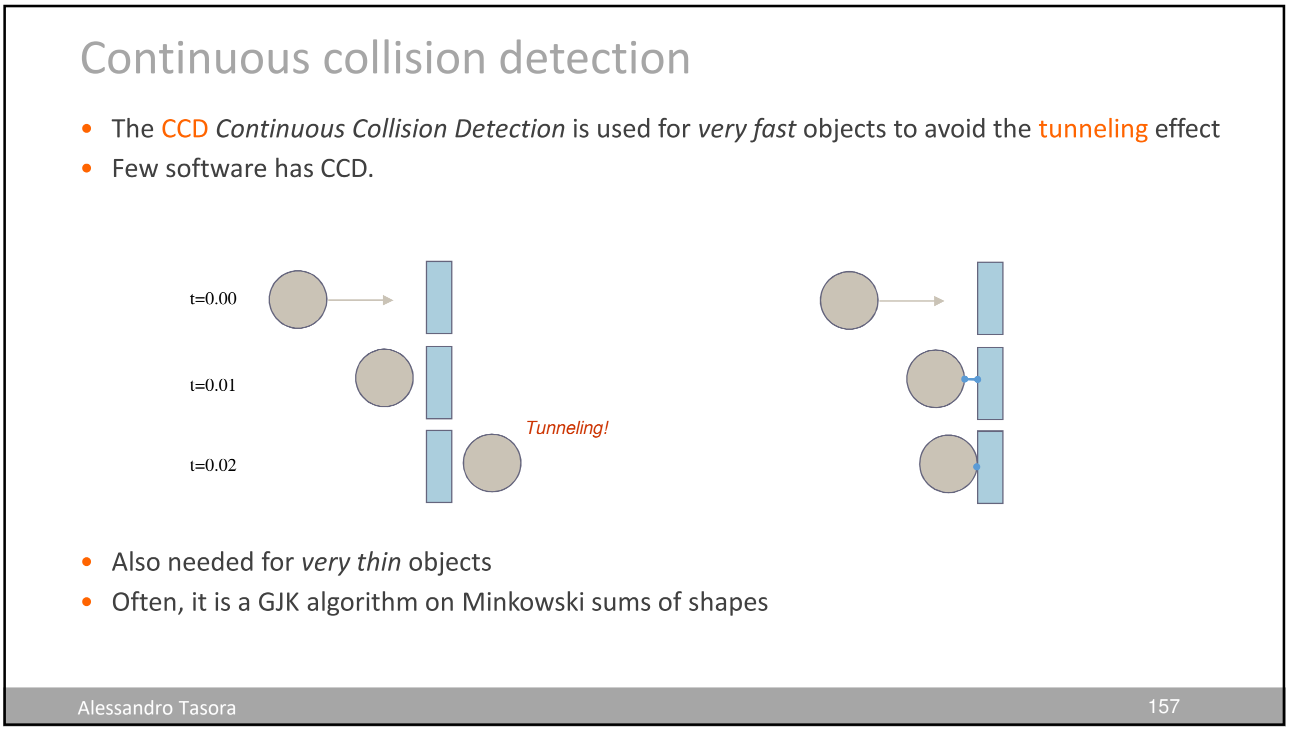




Middle phase

- If a shape is decomposed in many sub-shapes, the narrow-phase can still hit the $O(n^2)$ issue…
- Solution: use a …
- Middle phase
- Example:
- Uses BVh trees of AABB to manage objects with thousands of triangles or sub-convex shapes

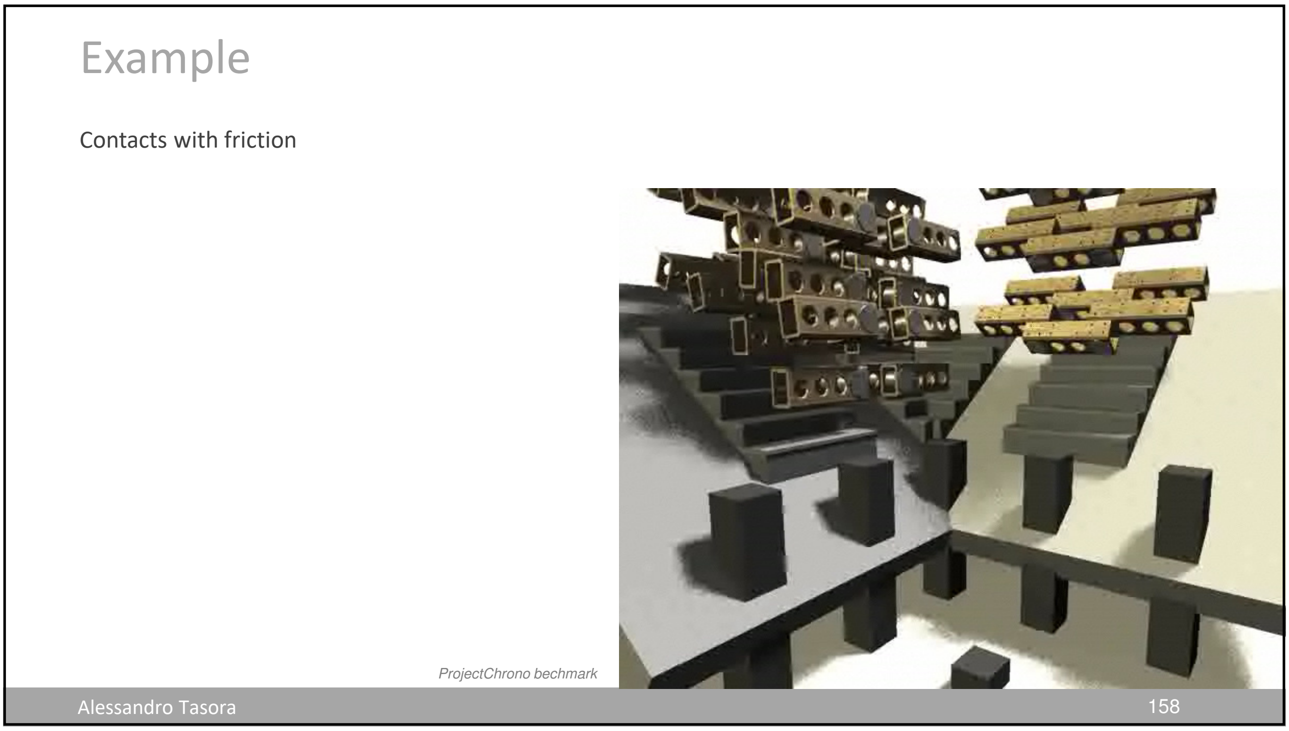




Continuous collision detection

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

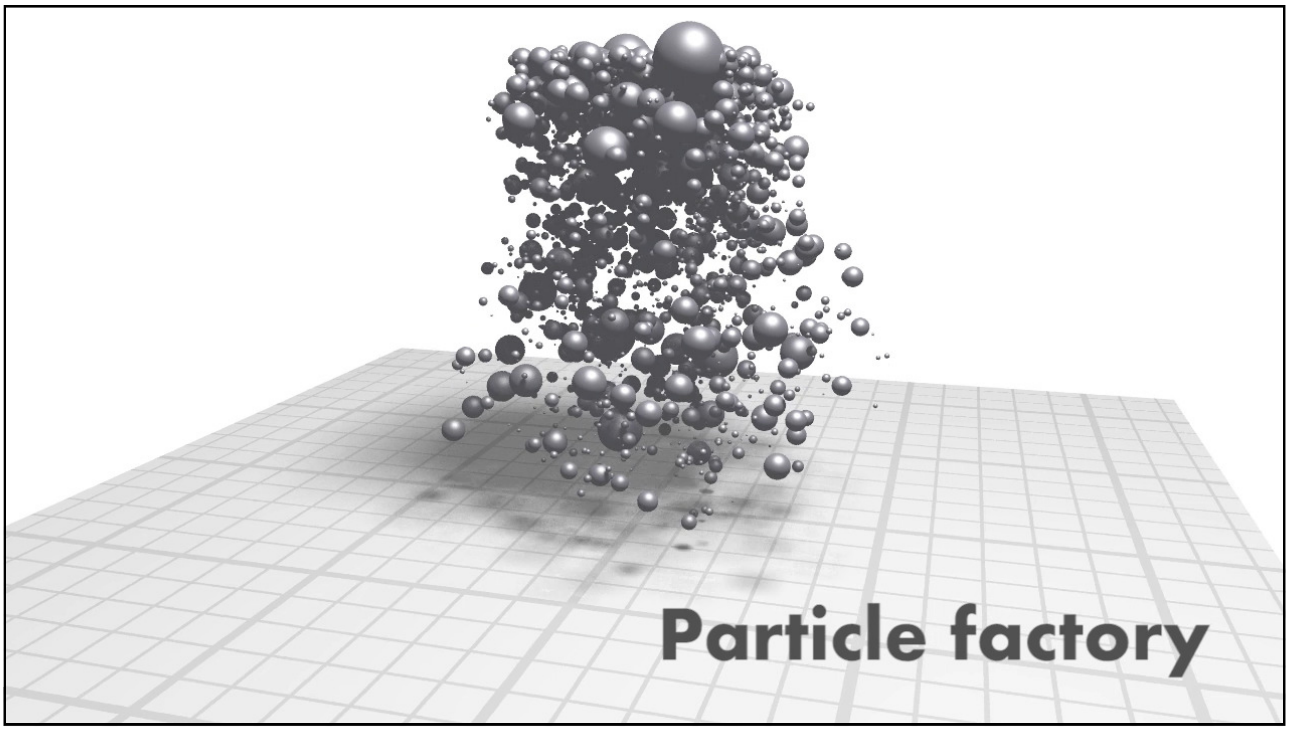
- Also needed for *very thin* objects
- Often, it is a GJK algorithm on Minkowski sums of shapes



Example

Contacts with friction

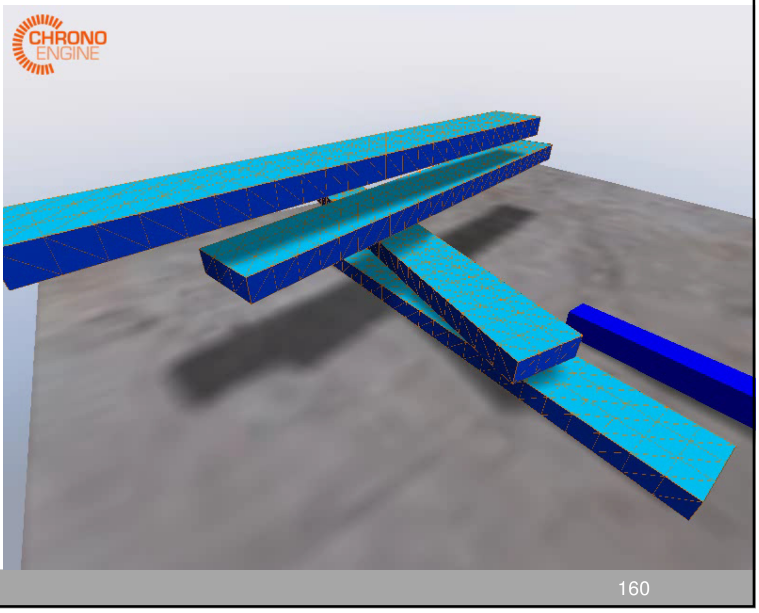
ProjectChrono bechmark



Particle factory

Example

Contacts between deformable parts (finite elements)



ProjectChrono benchmark, A.Tasora

Example: ProjectChrono benchmark (H.Mazhar)

7. AVAILABLE SOFTWARE

A list of multibody-related software tools

"This manual says what our product actually does,
no matter what the salesman may have told you it does"
In a graphic board manual, 1985.

Multibody software

- Classification by license:
  - Commercial
  - Open source

- Classification by architecture:
  - Stand-alone application with Graphical User Interface (GUI)
  - Only solver (batch processing)
  - As a plug-in for 3rd party CAD
  - As middleware (library)

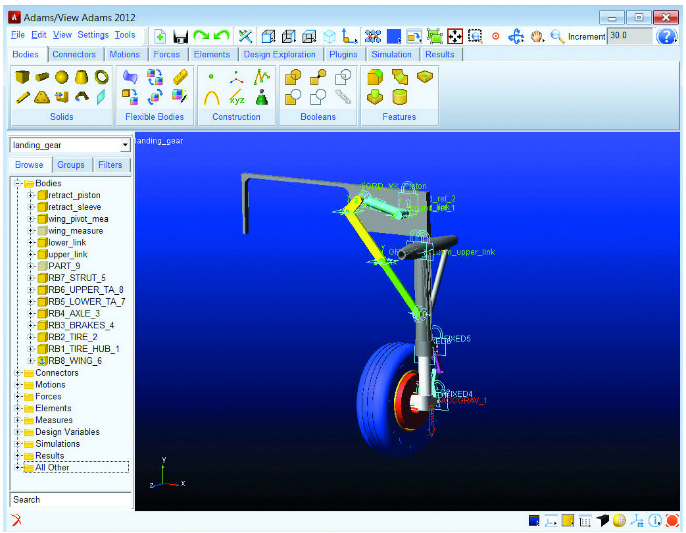
- Classification by purpose:
  - General purpose
  - Vertical (application-oriented)
  - Real-time
  - …

Multibody software

Notable 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

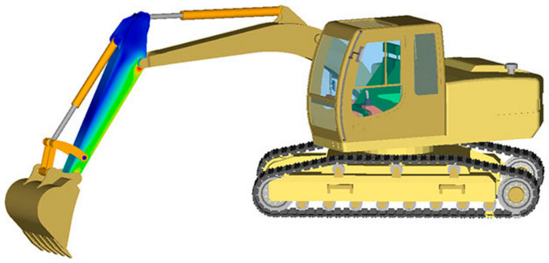


Multibody software

Notable commercial software (with GUI):

- ALTAIR MotionSolve
  - Similar to Adams
  - Integrated with other ALTAIR tools
  - Tools for automotive scenarios

Altair

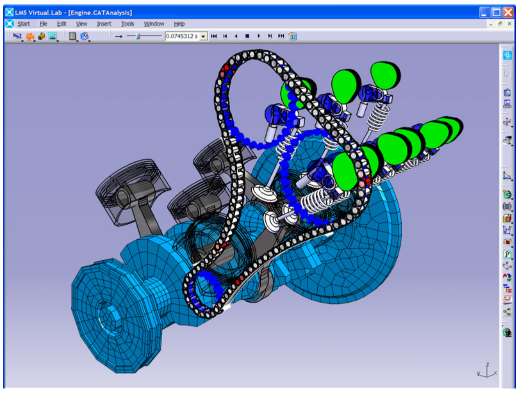


Multibody software

Notable 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

LMS® ENGINEERING INNOVATION

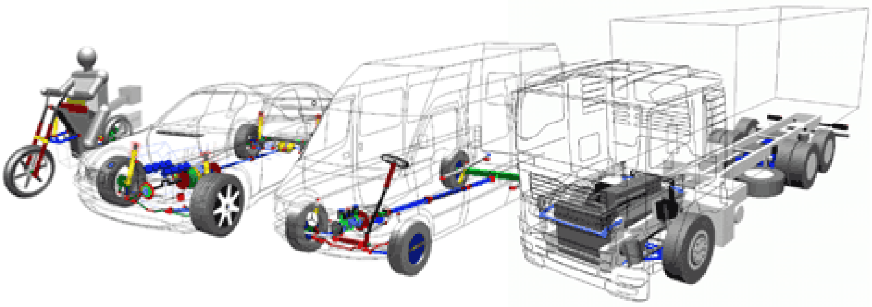


Multibody software

Notable 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

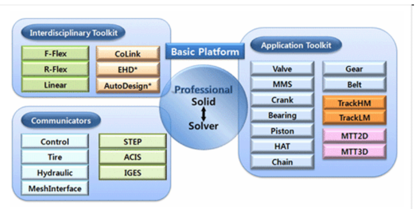
SIM PACK



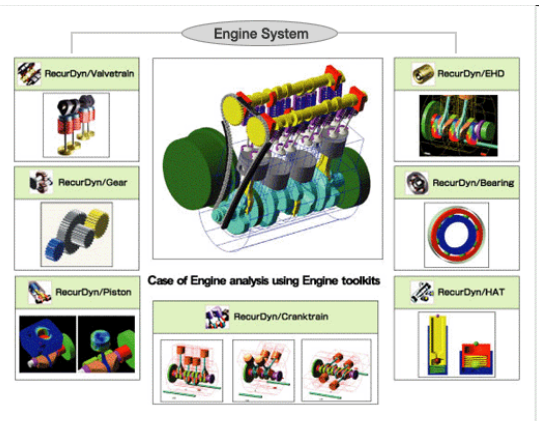
Multibody software

Notable 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'



Interdisciplinary Toolkit
F-Flex
R-Flex
Linear
CoLink
EHD*
AutoDesign*
Basic Platform
Professional
Solid
Solver
Application Toolkit
Valve
MMS
Crank
Bearing
Piston
HAT
Chain
Gear
Belt
TrackHM
TrackLM
MTT2D
MTT3D
Communicators
Control
Tire
Hydraulic
MeshInterface
STEP
ACIS
IGES

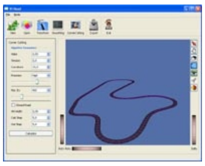


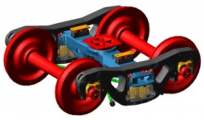
Engine System
RecurDyn/Valvetrain
RecurDyn/Gear
RecurDyn/Piston
RecurDyn/EHD
RecurDyn/Bearing
RecurDyn/HAT
Case of Engine analysis using Engine toolkits
RecurDyn/Cranktrain

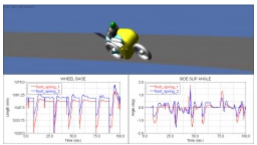
Multibody software

Special purpose commercial software – ex: vehicles

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



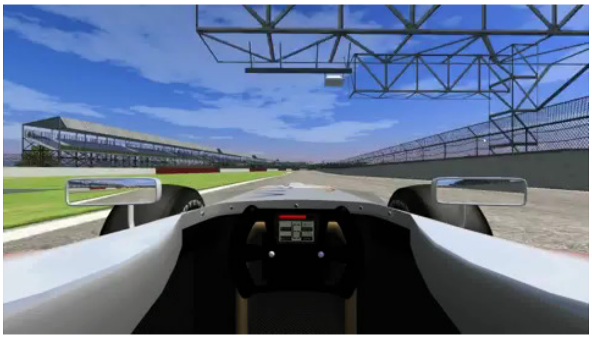




Multibody software

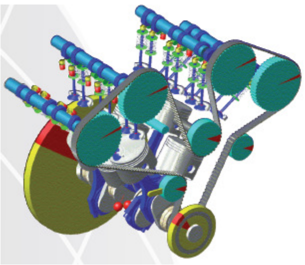
Special purpose commercial software – ex: vehicles

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

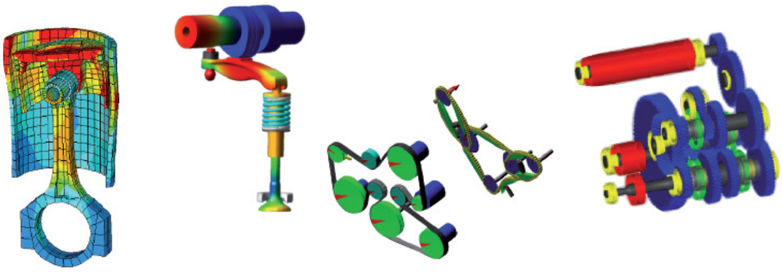


Multibody software

Special purpose commercial software – ex: vehicles



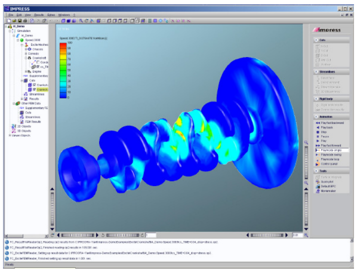
- VI-GRADE FEV VIRTUAL ENGINE
  - Crank train module
  - Timing Drive module
  - Valve train module
  - Gear drive module
  - Piston dynamics module

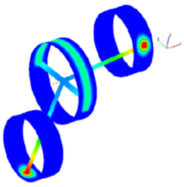


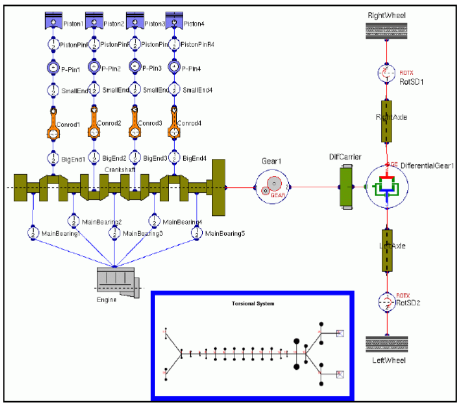
Multibody software

Special purpose commercial software – ex: vehicles

- AVL EXCITE



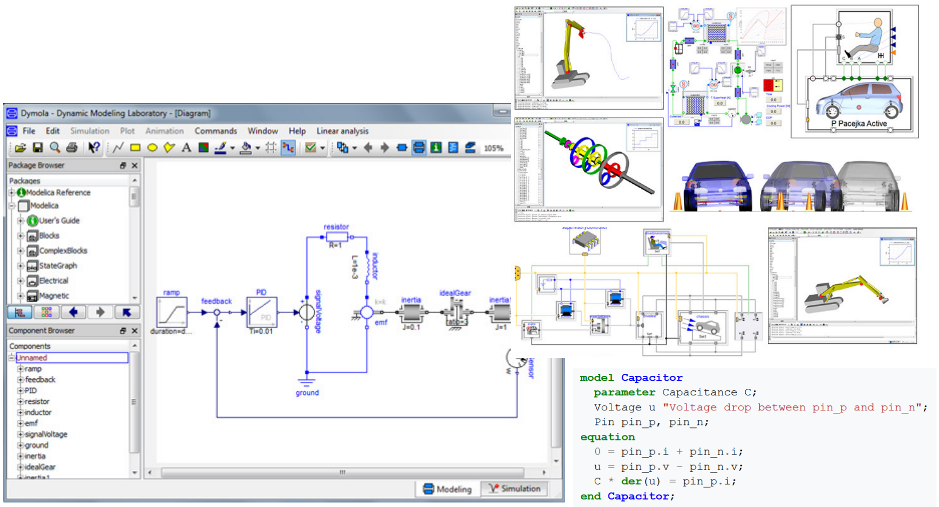




Multibody software

Model-based software (MODELICA language)

- DYMOLA

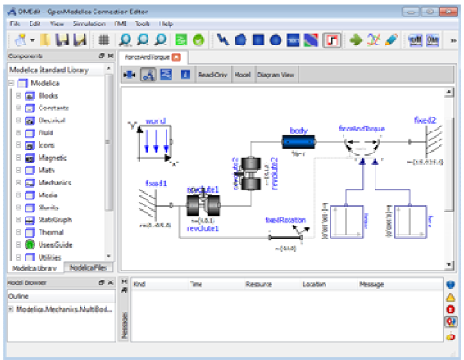


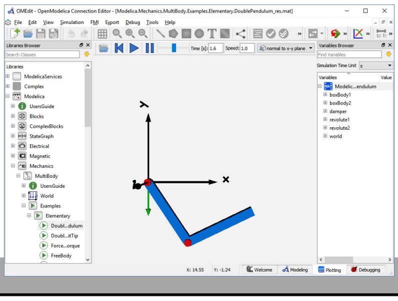
```
model Capacitor
  parameter Capacitance C;
  Voltage u "Voltage drop between pin_p and pin_n";
  Pin pin_p, pin_n;
equation
  0 = pin_p.i + pin_n.i;
  u = pin_p.v - pin_n.v;
  C * der(u) = pin_p.i;
end Capacitor;
```

Multibody software

Model-based software (MODELICA language)

- OpenModelica

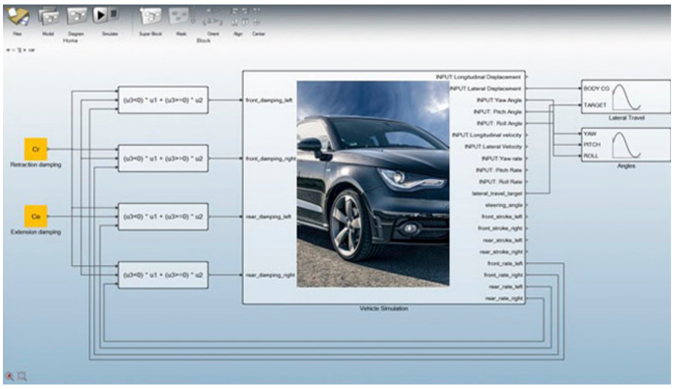




Multibody software

Model-based software (MODELICA language)

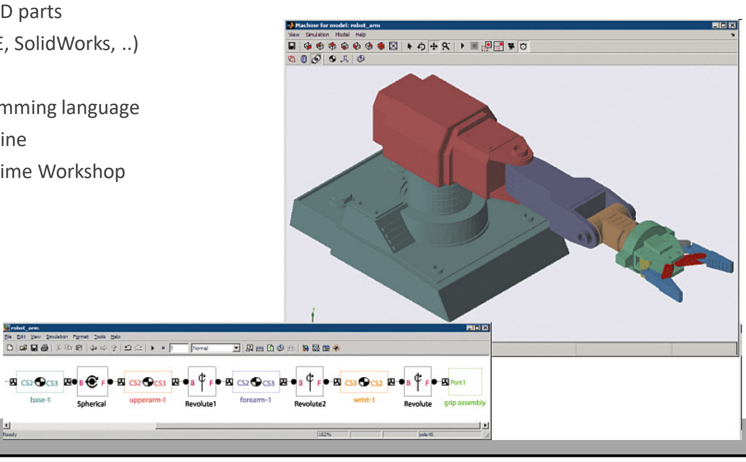
- Altair ACTIVATE



Multibody software

Model-based software (not using Modelica)

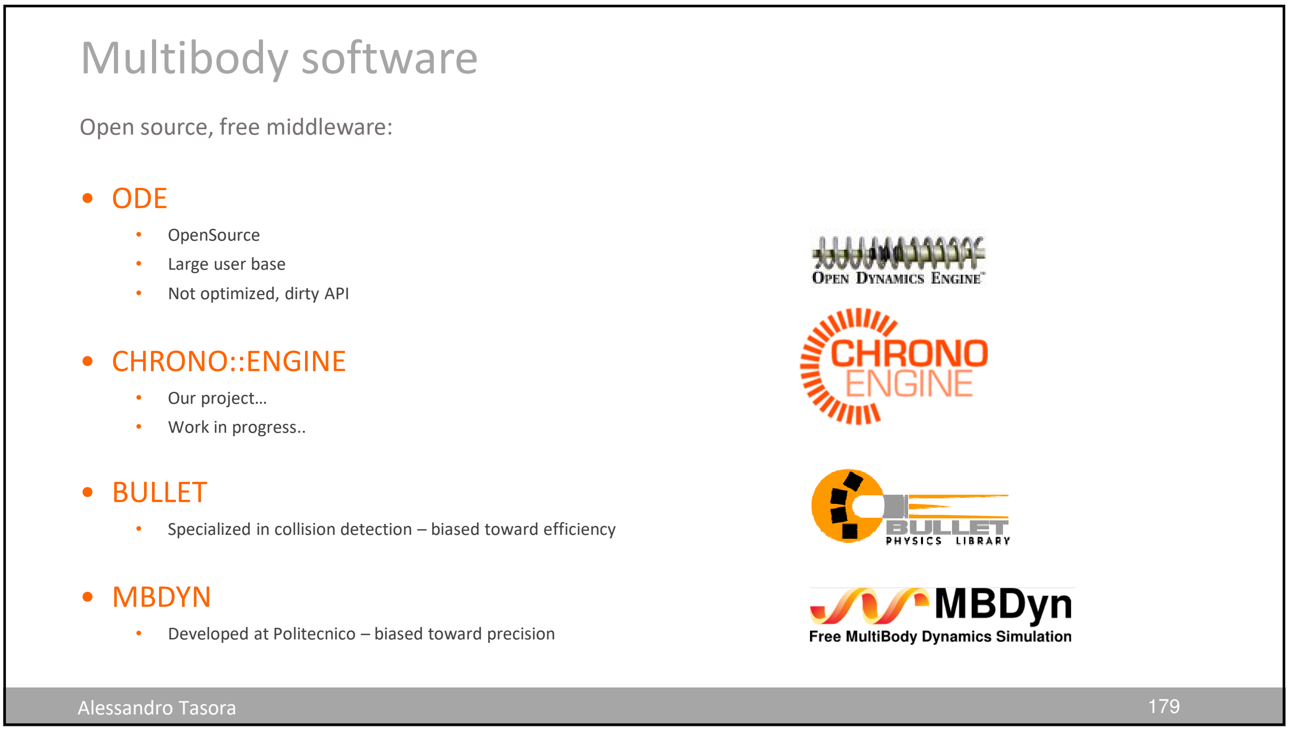
- SimScape - SimMechanics (MATLAB)
  - Based on Matlab + Simulink
  - No GUI for designing 3D parts
  - Import from CAD (ProE, SolidWorks, ..)
  - Slow simulation…
  - Expandable via programming language
  - Interfaces to SimDriveline
  - Export C code to RealTime Workshop



Multibody software

Proprietary middleware & APIs:

- HAVOK
  - 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, free middleware:

- ODE
  - OpenSource
  - Large user base
  - Not optimized, dirty API
- CHRONO::ENGINE
  - Our project…
  - Work in progress..
- BULLET
  - Specialized in collision detection – biased toward efficiency
- MBDYN
  - Developed at Politecnico – biased toward precision

OPEN DYNAMICS ENGINE

CHRONO ENGINE

BULLET PHYSICS LIBRARY

MBDyn

Free MultiBody Dynamics Simulation

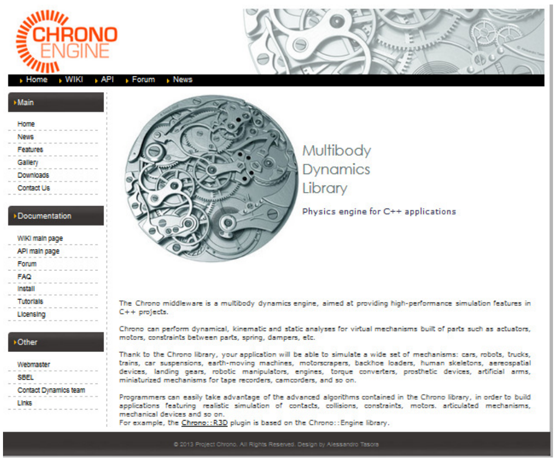
8.
PROJECT CHRONO

A tour into the software architecture of a middleware

> *"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to build bigger and better idiots. So far, the universe is winning."*
>
> Rick Cook

Multibody software

- Our ProjectChrono *middleware* project:



Multibody Dynamics Library

Physics engine for C++ applications

The Chrono middleware is a multibody dynamics engine, aimed at providing high-performance simulation features in C++ projects.

Chrono can perform dynamical, kinematic and static analyses for virtual mechanisms built of parts such as actuators, motors, constraints between parts, spring, dampers, etc.

Thank to the Chrono library, your application will be able to simulate a wide set of mechanisms: cars, robots, trucks, trains, car suspensions, earth-moving machines, motorscrapers, backhoe loaders, human skeletons, aereospatial devices, landing gears, robotic manipulators, engines, torque converters, prosthetic devices, artificial arms, miniaturized mechanisms for tape recorders, camcorders, and so on.

Programmers can easily take advantage of the advanced algorithms contained in the Chrono library, in order to build applications featuring realistic simulation of contacts, collisions, constraints, motors. articulated mechanisms, mechanical devices and so on.
For example, the Chrono::R3D plugin is based on the Chrono::Engine library.

© 2013 Project Chrono. All Rights Reserved. Design by Alessandro Tasora

- 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

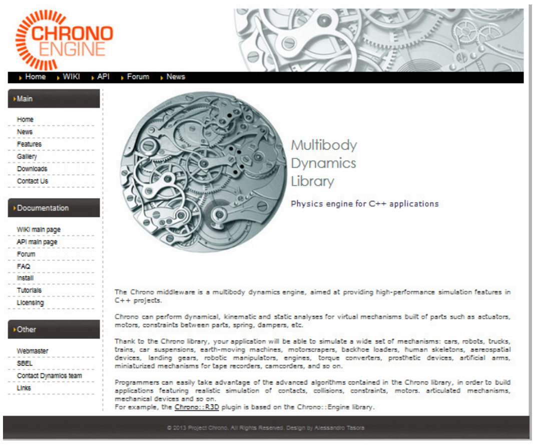
- Part of ProjectChrono: ← *very recent initiative, more to come...*

CHRONO ENGINE

CHRONO RENDER

CHRONO FLEX

CHRONO



Physics engine for C++ applications

The Chrono middleware is a multibody dynamics engine, aimed at providing high-performance simulation features in C++ projects.

Chrono can perform dynamical, kinematic and static analyses for virtual mechanisms built of parts such as actuators, motors, constraints between parts, spring, dampers, etc.

Thank to the Chrono library, your application will be able to simulate a wide set of mechanisms: cars, robots, trucks, trains, car suspensions, earth-moving machines, motorscrapers, backhoe loaders, human skeletons, aereospatial devices, landing gears, robotic manipulators, engines, torque converters, prosthetic devices, artificial arms, miniaturized mechanisms for tape recorders, camcorders, and so on.

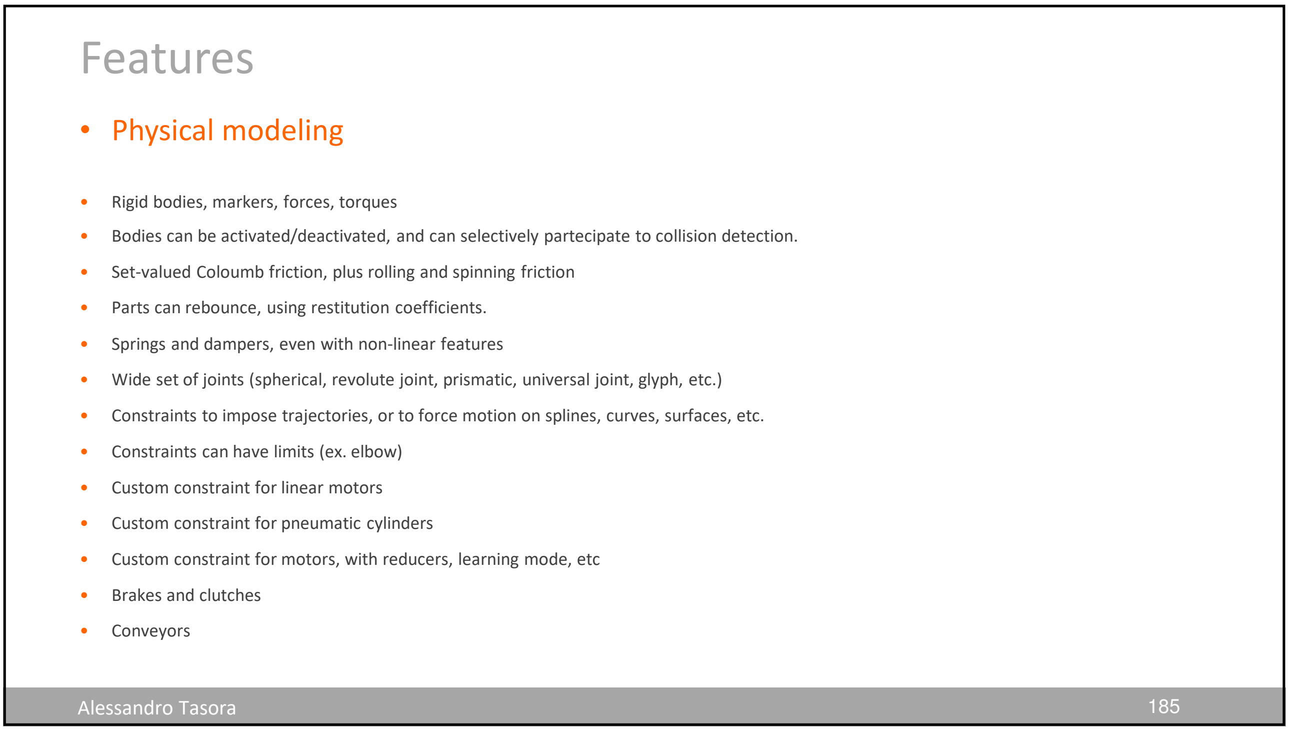
Programmers can easily take advantage of the advanced algorithms contained in the Chrono library, in order to build applications featuring realistic simulation of contacts, collisions, constraints, motors. articulated mechanisms, mechanical devices and so on.
For example, the Chrono::R3D plugin is based on the Chrono::Engine library.

© 2013 Project Chrono. All Rights Reserved. Design by Alessandro Tasora

Features

- Core features

- Platform independent
- C++11 compliant
- CMAKE build toolchain
- 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
- Class factory and archiving
- Smart pointers
- High resolution timers
- …

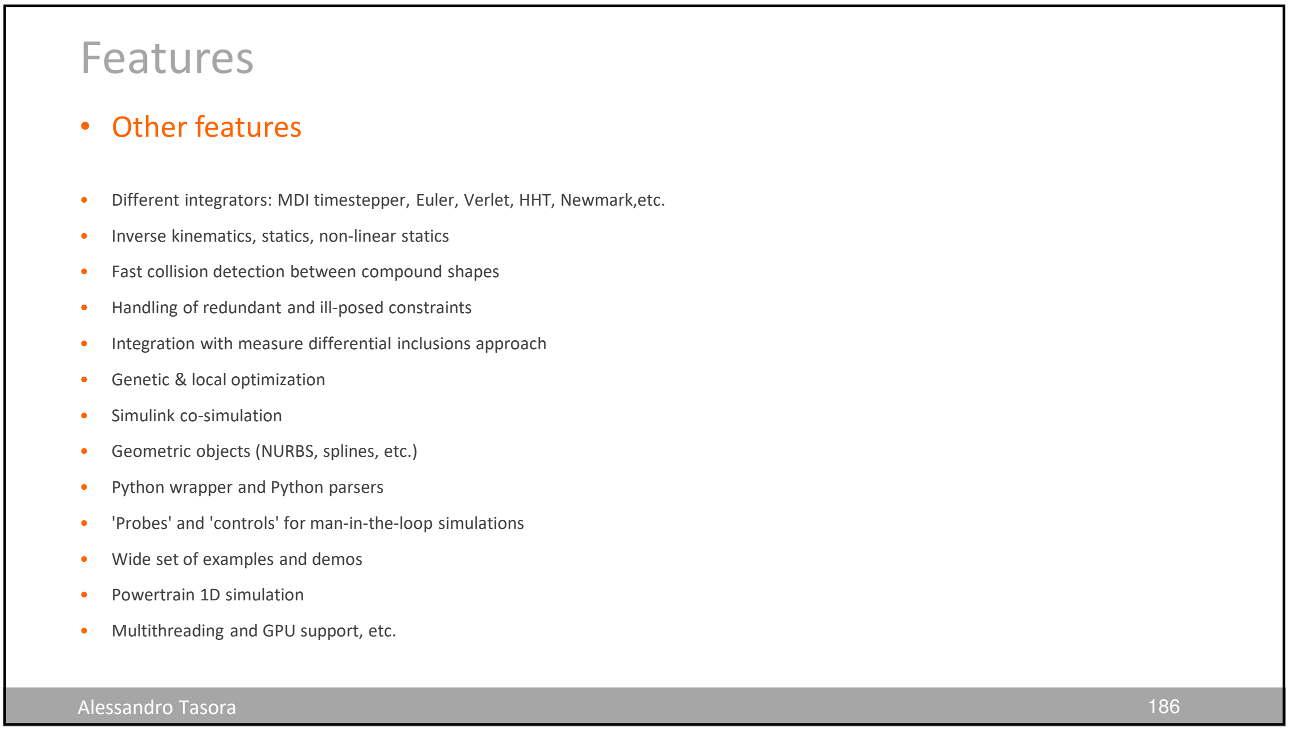


Features

- Physical modeling

- Rigid bodies, markers, forces, torques
- Bodies can be activated/deactivated, and can selectively partecipate to collision detection.
- Set-valued Coloumb friction, plus rolling and spinning friction
- Parts can rebounce, using 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)
- Custom constraint for linear motors
- Custom constraint for pneumatic cylinders
- Custom constraint for motors, with reducers, learning mode, etc
- Brakes and clutches
- Conveyors

Alessandro Tasora 185

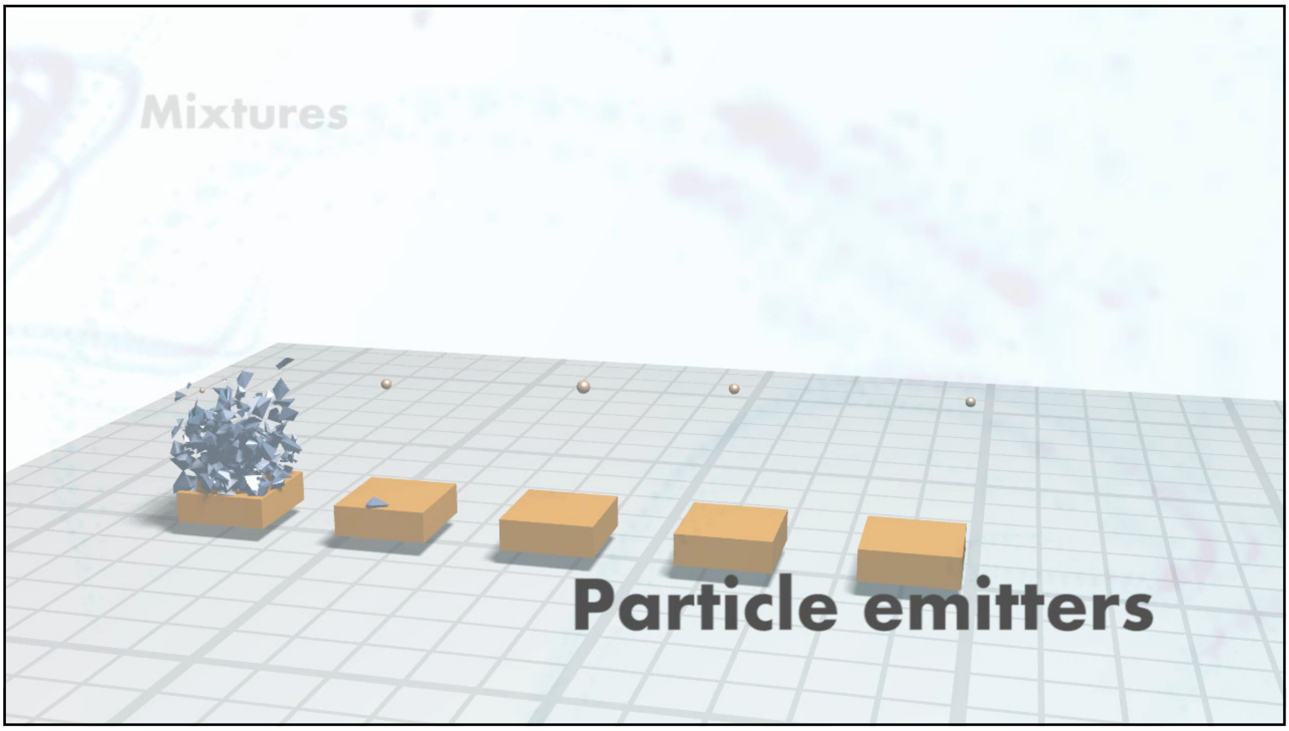


Features

- Other features

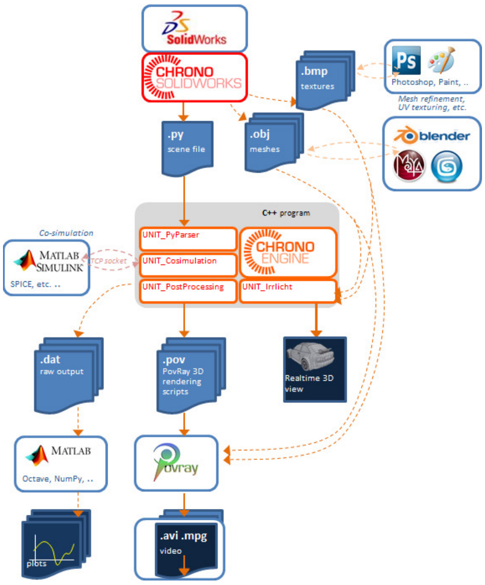
- Different integrators: MDI timestepper, Euler, Verlet, HHT, Newmark,etc.
- Inverse kinematics, statics, non-linear statics
- Fast collision detection between compound shapes
- Handling of redundant and ill-posed constraints
- Integration with measure differential inclusions approach
- Genetic & local optimization
- Simulink co-simulation
- Geometric objects (NURBS, splines, etc.)
- Python wrapper and Python parsers
- 'Probes' and 'controls' for man-in-the-loop simulations
- Wide set of examples and demos
- Powertrain 1D simulation
- Multithreading and GPU support, etc.

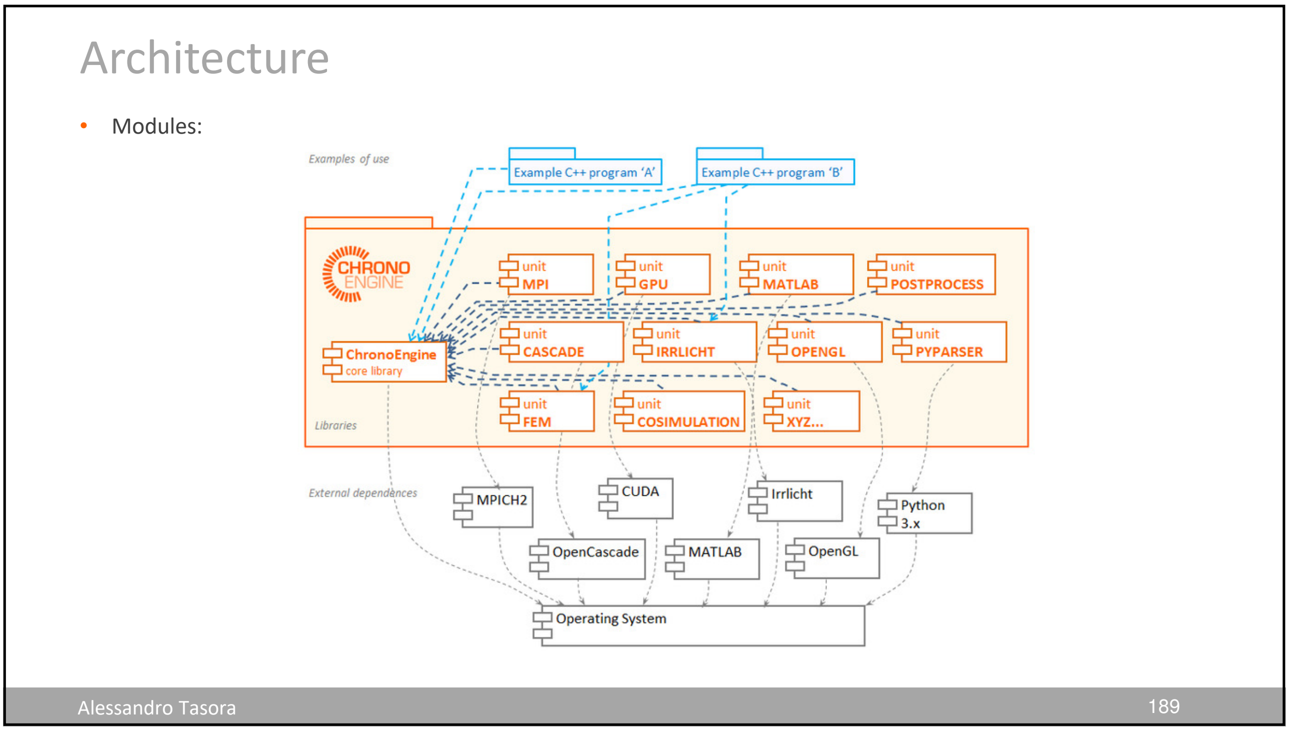
Alessandro Tasora 186



Architecture

- Workflow:

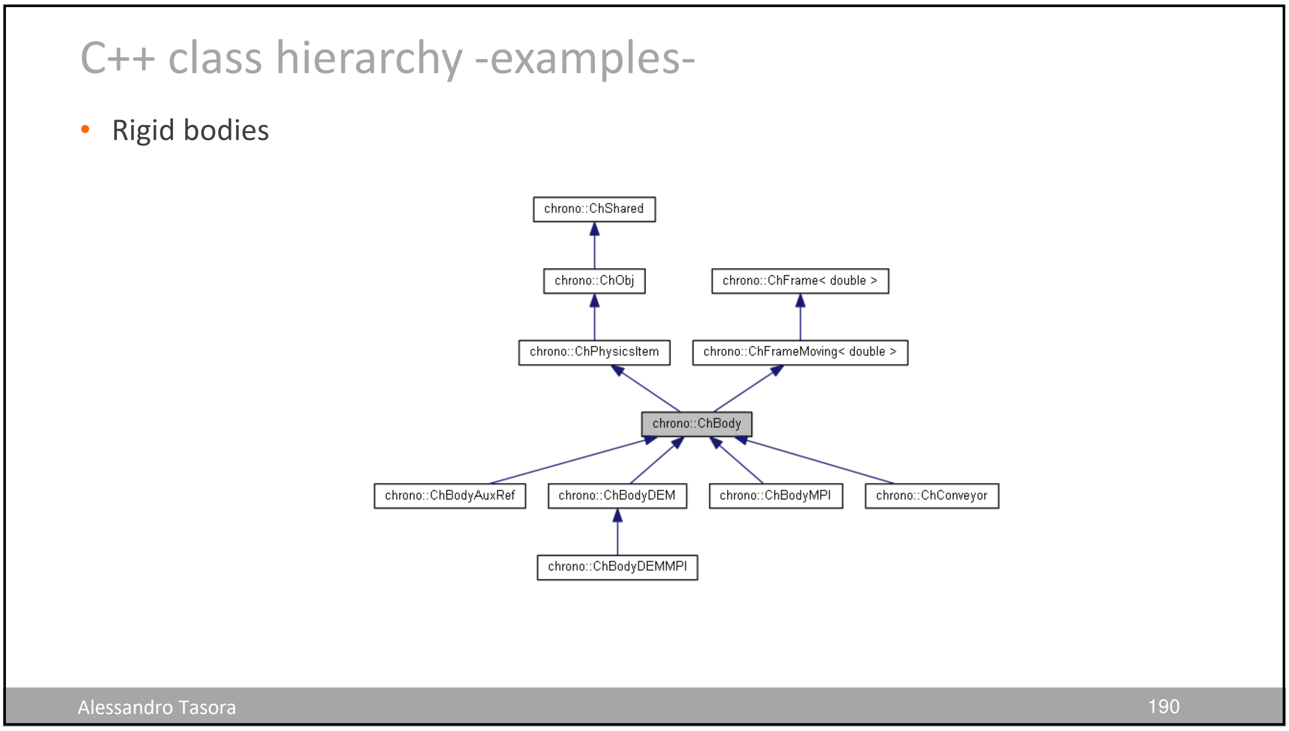




Architecture

- Modules:

Alessandro Tasora 189



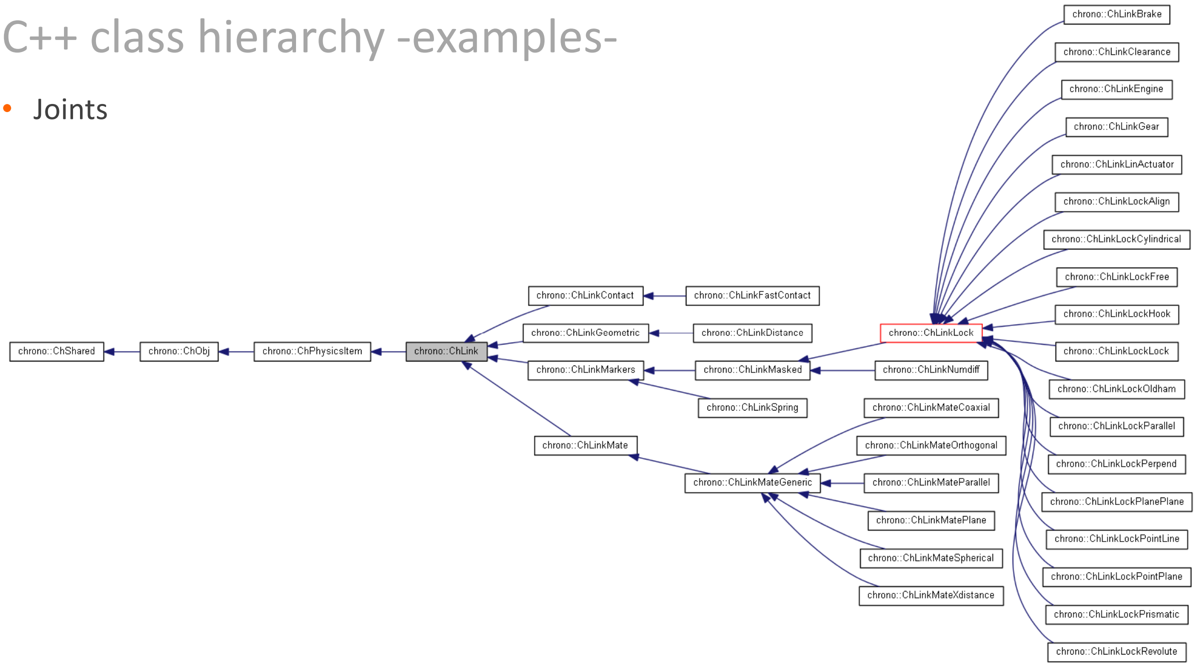
C++ class hierarchy -examples-

- Rigid bodies

Alessandro Tasora 190

C++ class hierarchy -examples-

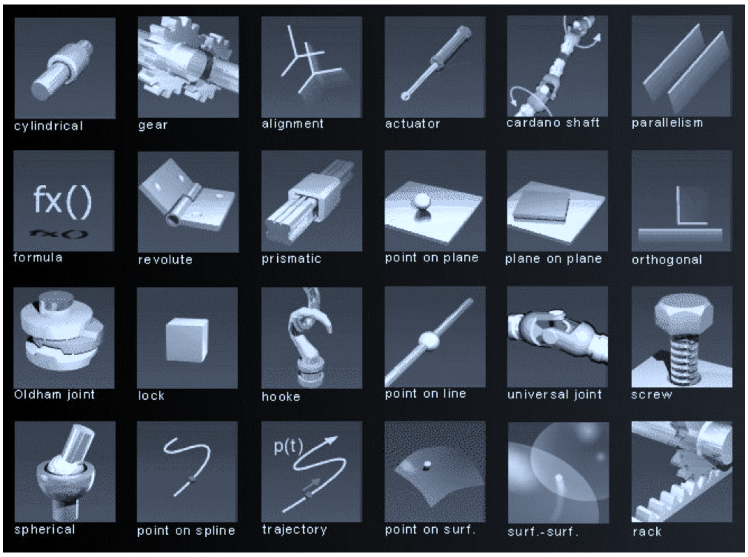
- Joints



chrono::ChShared
chrono::ChObj
chrono::ChPhysicsItem
chrono::ChLink
chrono::ChLinkContact
chrono::ChLinkFastContact
chrono::ChLinkGeometric
chrono::ChLinkDistance
chrono::ChLinkMarkers
chrono::ChLinkMasked
chrono::ChLinkNumdiff
chrono::ChLinkSpring
chrono::ChLinkMate
chrono::ChLinkMateGeneric
chrono::ChLinkMateCoaxial
chrono::ChLinkMateOrthogonal
chrono::ChLinkMateParallel
chrono::ChLinkMatePlane
chrono::ChLinkMateSpherical
chrono::ChLinkMateXdistance
chrono::ChLinkLock
chrono::ChLinkBrake
chrono::ChLinkClearance
chrono::ChLinkEngine
chrono::ChLinkGear
chrono::ChLinkLinActuator
chrono::ChLinkLockAlign
chrono::ChLinkLockCylindrical
chrono::ChLinkLockFree
chrono::ChLinkLockHook
chrono::ChLinkLockLock
chrono::ChLinkLockOldham
chrono::ChLinkLockParallel
chrono::ChLinkLockPerpend
chrono::ChLinkLockPlanePlane
chrono::ChLinkLockPointLine
chrono::ChLinkLockPointPlane
chrono::ChLinkLockPrismatic
chrono::ChLinkLockRevolute

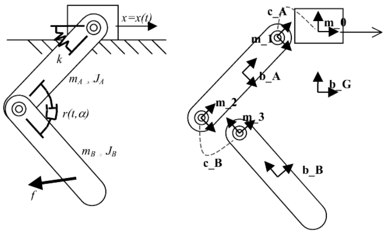
C++ class hierarchy -examples-

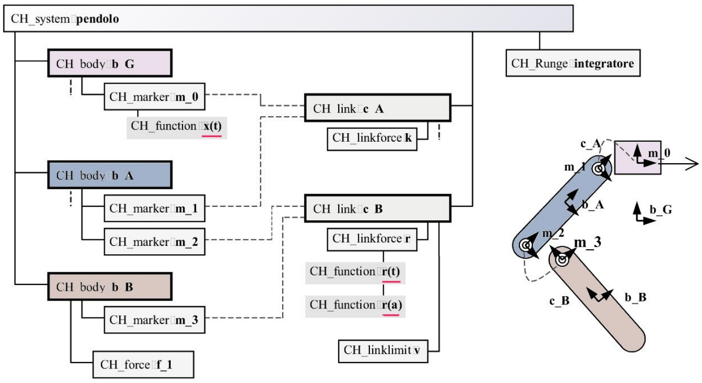
Some joint types in our Chrono::Engine software



C++ transient database

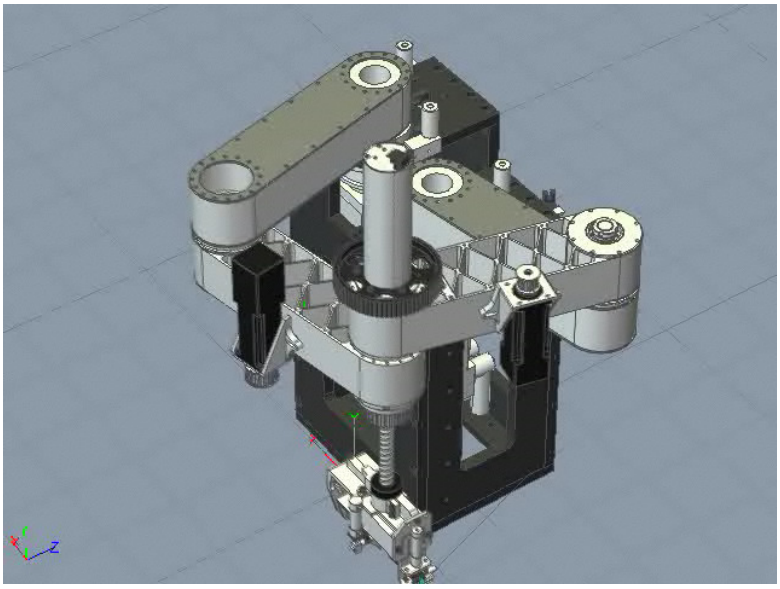
- Complex object hierarchy:
  smart shared pointers are used



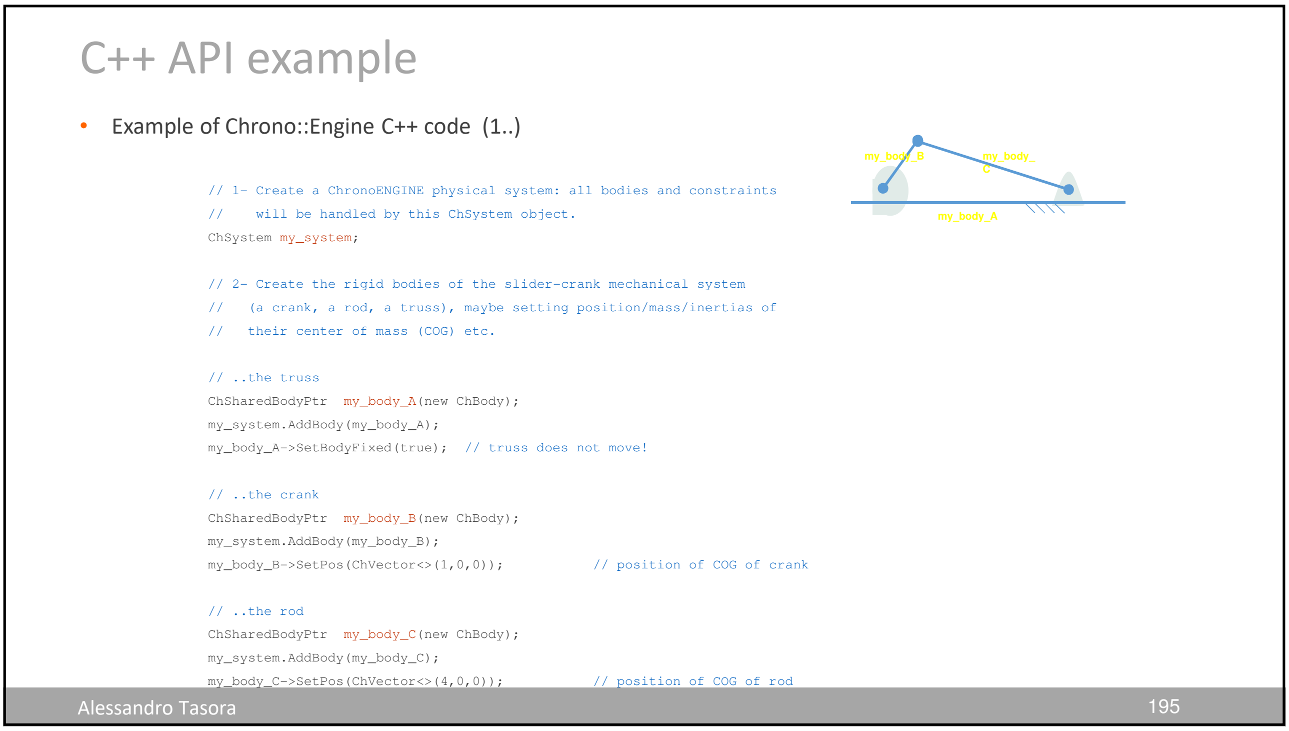


CH_system pendolo
CH_body b_G
CH_marker m_0
CH_function x(t)
CH_Runge integratore
CH_link c_A
CH_linkforce k
CH_body b_A
CH_marker m_1
CH_marker m_2
CH_link c_B
CH_linkforce r
CH_function r(t)
CH_function r(a)
CH_body b_B
CH_marker m_3
CH_force f_1
CH_linklimit v

Example



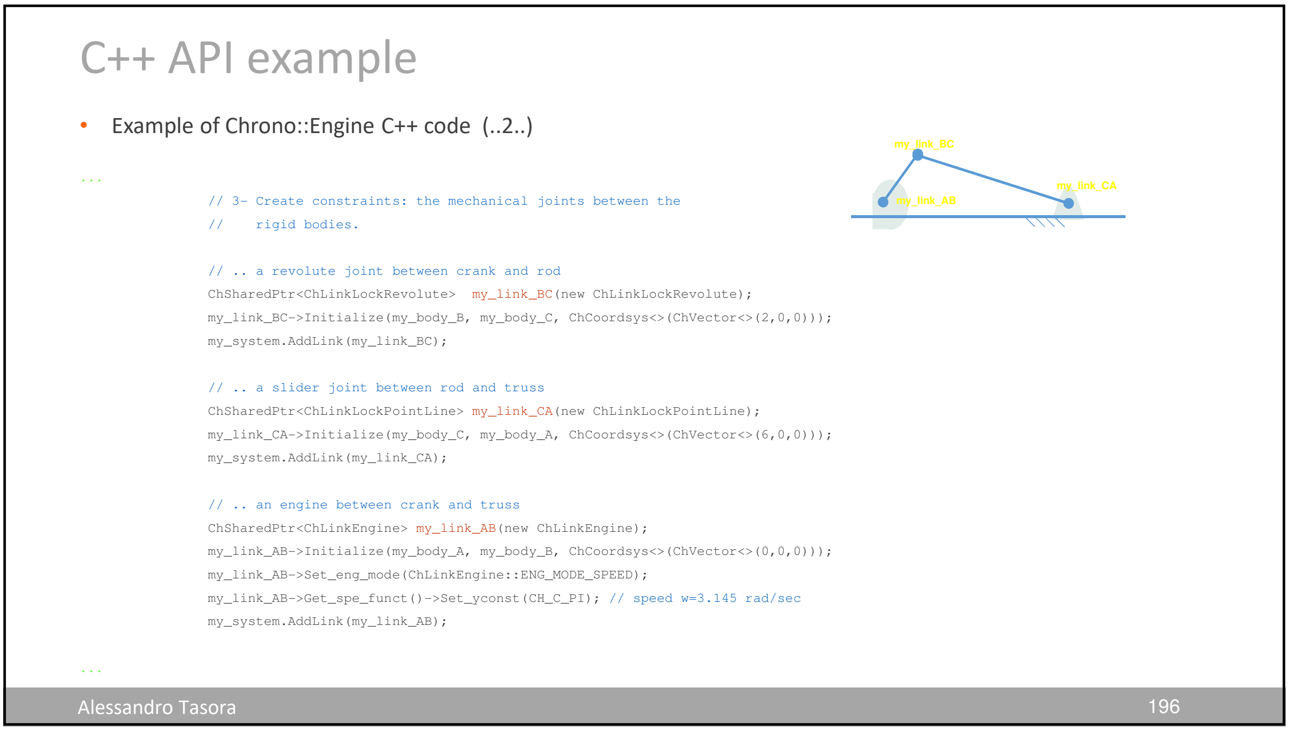
The GRANIT parallel-kinematics robot (Tasora, Righettini, Chatterton, 2007)



C++ API example

- 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
```



C++ API example

- 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);

...
```

C++ API example

- 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

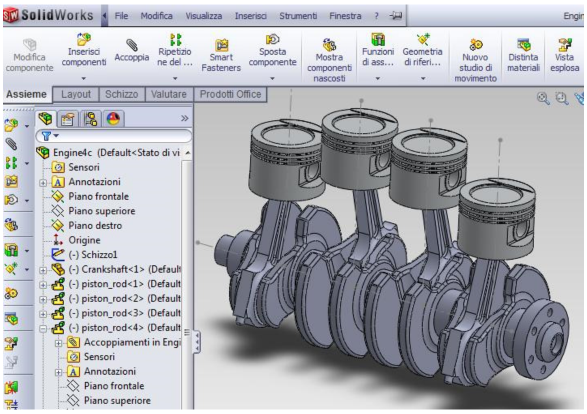
Demo_fourbar.exe

Demo_pendulum.exe

Demo_gears.exe

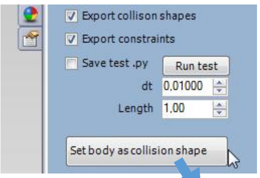
Chrono::SolidWorks

- 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

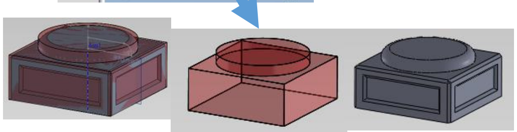


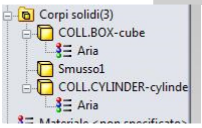
Chrono::SolidWorks

- Our Chrono::SolidWorks *add-in:*

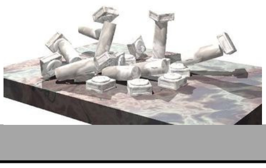


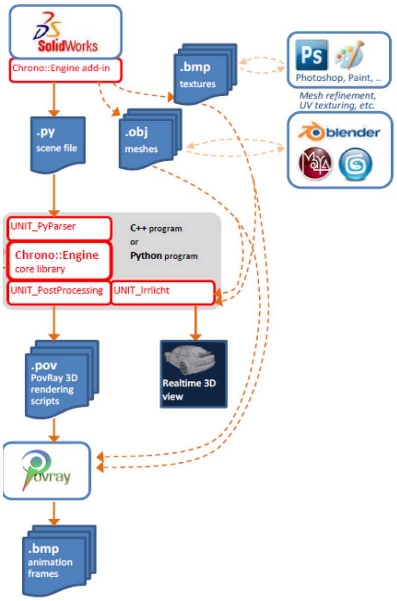
☑ Export collison shapes
☑ Export constraints
□ Save test .py Run test
dt 0,01000
Length 1,00
Set body as collision shape





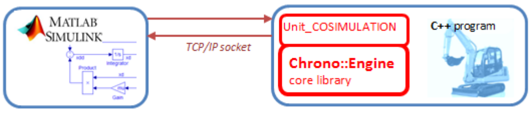
Corpi solidi(3)
COLL.BOX-cube
Aria
Smusso1
COLL.CYLINDER-cylinde
Aria
Materiale <non specificato>

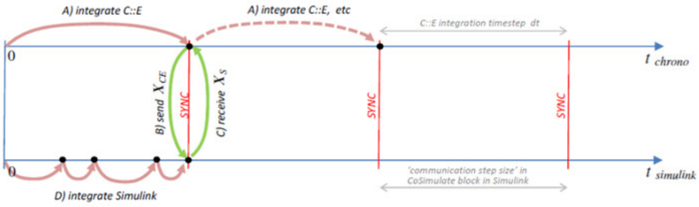




COSIMULATION module

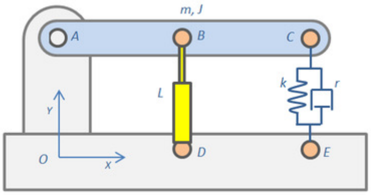
- The COSIMULATION module:

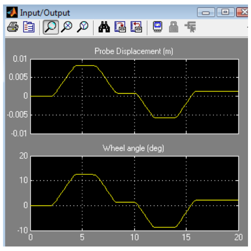


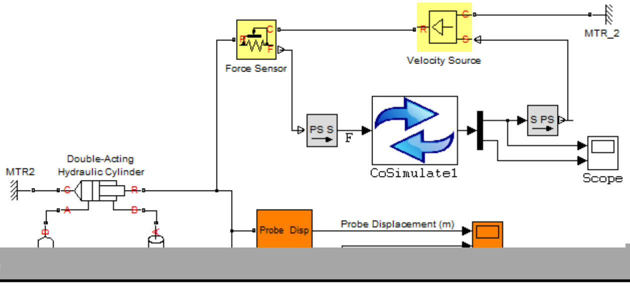


COSIMULATION module

- The COSIMULATION module:

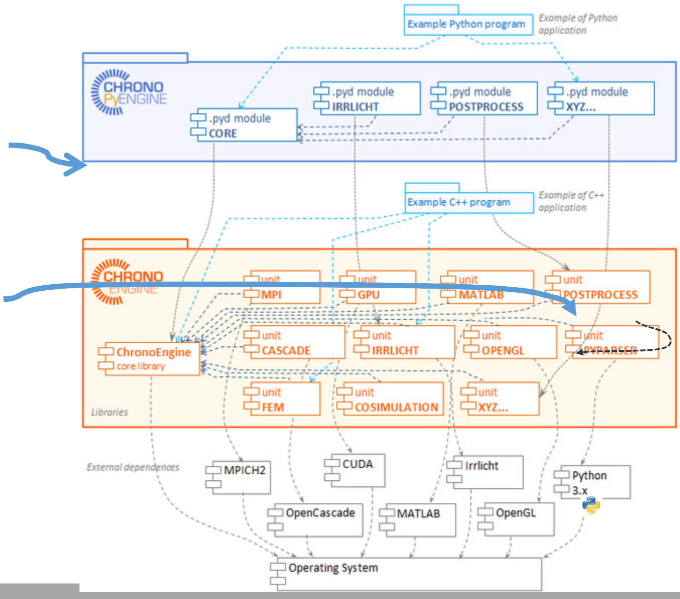






PYTHON module

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



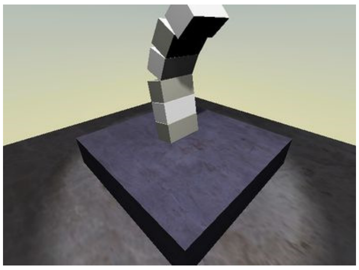
PYCHRONO

- PYCHRONO is the Python wrapper of Chrono:

Example:

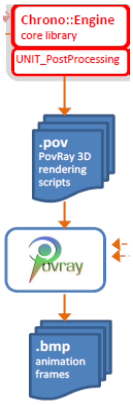
```
my_quat = chrono.ChQuaternionD(1,2,3,4)
my_qconjugate = ~my_quat
print ('quat. conjugate  =', my_qconjugate)
print ('quat. dot product=', my_qconjugate ^ my_quat)
print ('quat. product=',     my_qconjugate % my_quat)
ma = chrono.ChMatrixDynamicD(4,4)
ma.FillDiag(-2)
mb = chrono.ChMatrixDynamicD(4,4)
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);
```

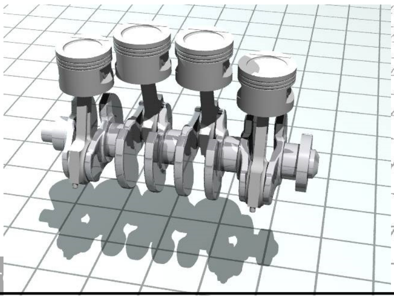
```
...
```

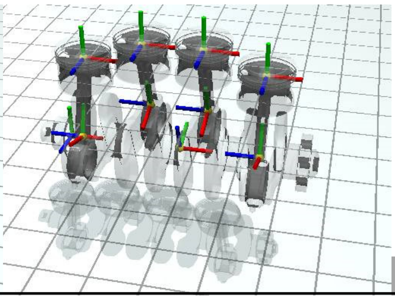


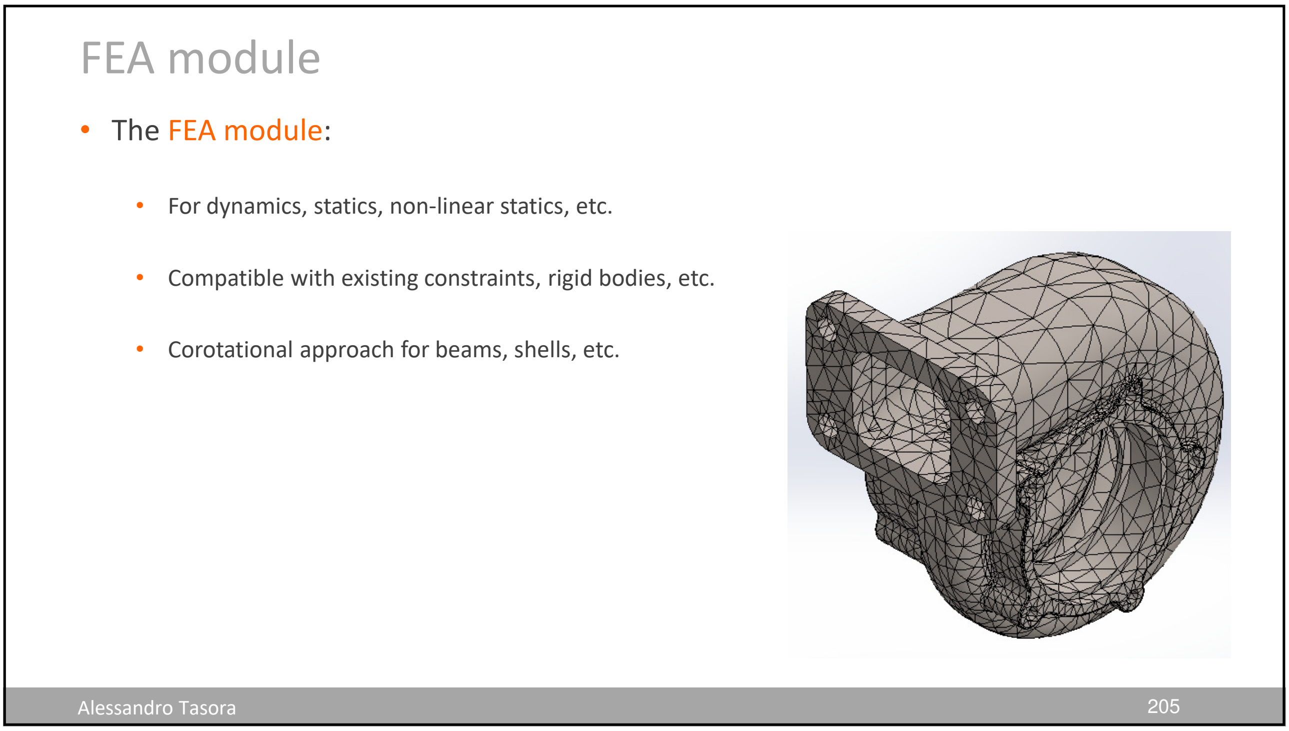
POSTPROCESSING module

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





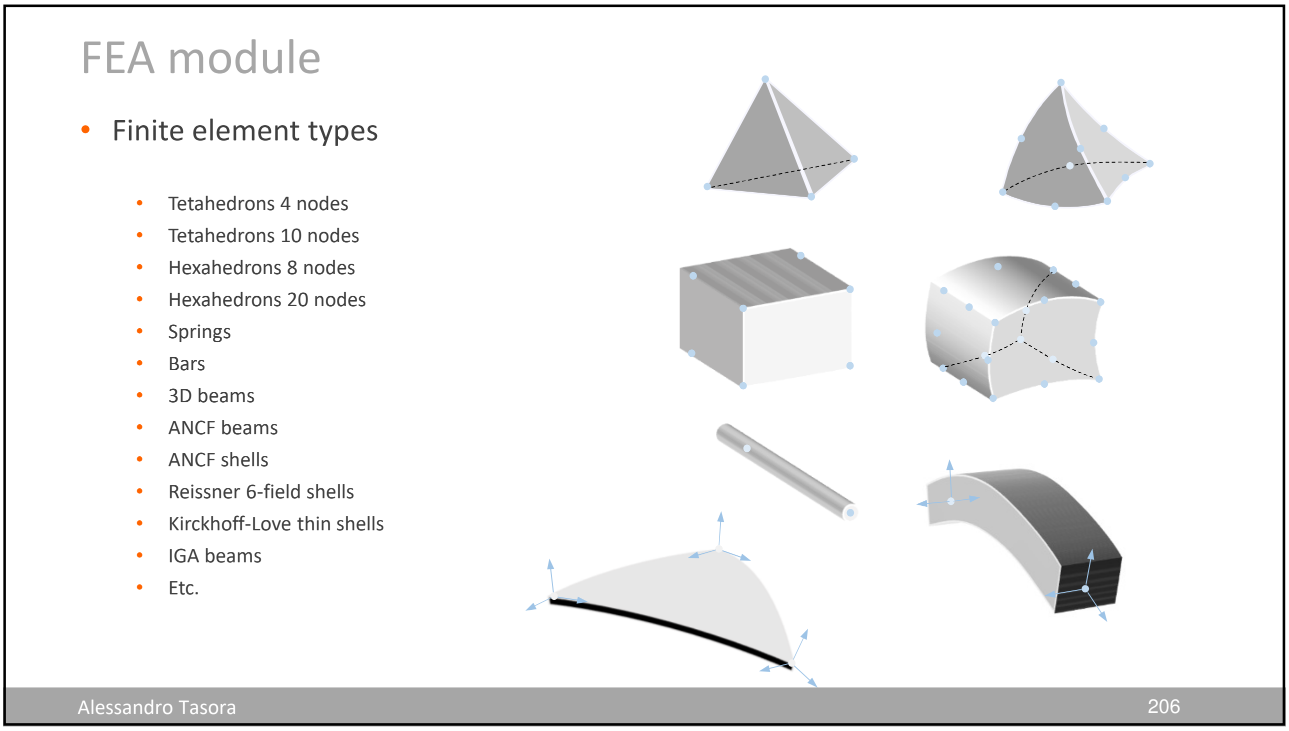




FEA module

- The FEA module:
  - For dynamics, statics, non-linear statics, etc.
  - Compatible with existing constraints, rigid bodies, etc.
  - Corotational approach for beams, shells, etc.

Alessandro Tasora 205



FEA module

- Finite element types
  - Tetahedrons 4 nodes
  - Tetahedrons 10 nodes
  - Hexahedrons 8 nodes
  - Hexahedrons 20 nodes
  - Springs
  - Bars
  - 3D beams
  - ANCF beams
  - ANCF shells
  - Reissner 6-field shells
  - Kirckhoff-Love thin shells
  - IGA beams
  - Etc.

Alessandro Tasora 206

FEA module

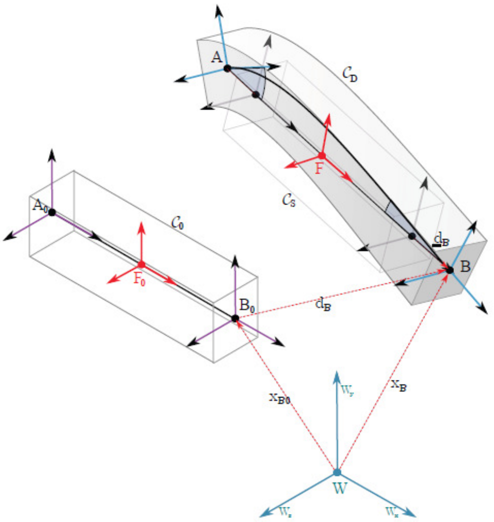
- The corotational approach for beam FE

- Locally, a 3D Eulero-Bernoulli beam..

$$\underline{\boldsymbol{f}}_{in} = \underline{\boldsymbol{K}}\underline{\boldsymbol{d}}$$

$$\underline{\boldsymbol{d}} \;=\; [\underline{\boldsymbol{d}}_A, \underline{\boldsymbol{\theta}}_A, \underline{\boldsymbol{d}}_B, \underline{\boldsymbol{\theta}}_B]$$

$$\underline{\boldsymbol{K}} = \begin{pmatrix}
k_u & 0 & 0 & 0 & 0 & 0 & -k_u & 0 & 0 & 0 & 0 & 0 \\
 & k_v & 0 & 0 & 0 & k_{v\psi} & 0 & -k_v & 0 & 0 & 0 & k_{v\psi} \\
 & & k_w & 0 & -k_{w\theta} & 0 & 0 & 0 & -k_w & 0 & -k_{w\theta} & 0 \\
 & & & k_\phi & 0 & 0 & 0 & 0 & 0 & -k\phi & 0 & 0 \\
 & & & & k_\theta & 0 & 0 & 0 & k_{w\theta} & 0 & k_{\theta\theta} & 0 \\
 & & & & & k_\psi & 0 & -k_{v\psi} & 0 & 0 & 0 & k_{\psi\psi} \\
 & & & & & & k_u & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & k_v & 0 & 0 & 0 & -k_{v\psi} \\
 & \text{Sym.} & & & & & & & k_w & 0 & k_{w\theta} & 0 \\
 & & & & & & & & & k_\phi & 0 & 0 \\
 & & & & & & & & & & k_\theta & 0 \\
 & & & & & & & & & & & k_\psi
\end{pmatrix}$$

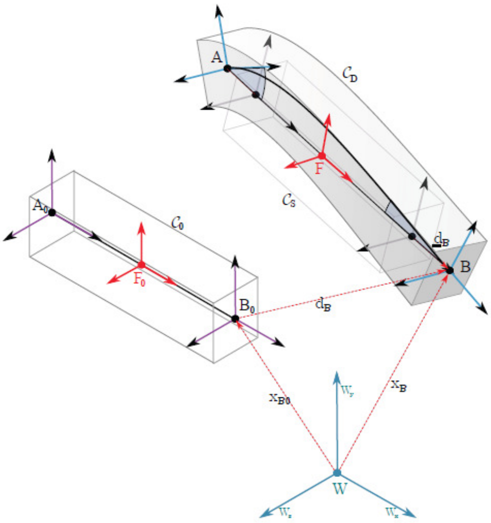


FEA module

- The corotational approach for beam FE

- ..mapped to global coordinates:

$$\boldsymbol{q} = [\boldsymbol{x}_1, \boldsymbol{\rho}_1, \boldsymbol{x}_2, \boldsymbol{\rho}_2, \ldots, \boldsymbol{x}_n, \boldsymbol{\rho}_n] \in \mathbb{R}^{(3+4)n}$$
$$\boldsymbol{v} = [\boldsymbol{v}_1, \boldsymbol{\omega}_1, \boldsymbol{v}_2, \boldsymbol{\omega}_2, \ldots, \boldsymbol{v}_n, \boldsymbol{\omega}_n] \in \mathbb{R}^{(3+3)n}$$



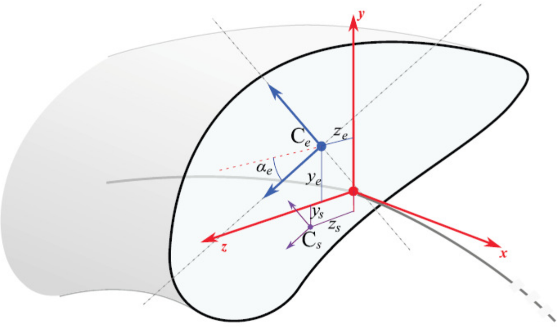
$$\boldsymbol{f}_{in} = \boldsymbol{R}_\diamond \underline{\boldsymbol{P}}^t \boldsymbol{H}^t \underline{\boldsymbol{f}}_{in}$$

$$\boldsymbol{K} = \qquad \boldsymbol{R}_\diamond \left(\underline{\boldsymbol{P}}^t \boldsymbol{H}^t \underline{\boldsymbol{K}} \boldsymbol{H} \underline{\boldsymbol{P}} - \underline{\boldsymbol{F}}_{nm} \underline{\boldsymbol{G}} - \underline{\boldsymbol{G}}^t \underline{\boldsymbol{F}}_n^t \underline{\boldsymbol{P}} + \underline{\boldsymbol{P}}^t \boldsymbol{L}_H \underline{\boldsymbol{P}} \right) \boldsymbol{R}_\diamond^t$$

FEA module

- The corotational approach for beam FE

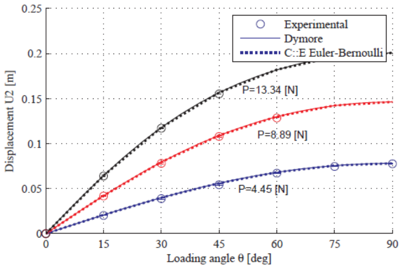
- Generic sections
- Offset in shear center
- Offset in elastic center
- Section rotation
- etc.

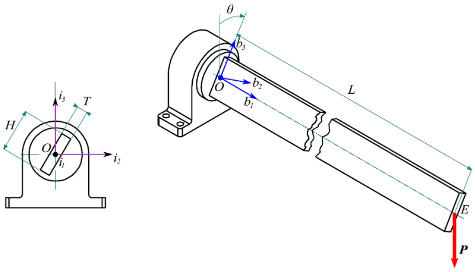


FEA unit

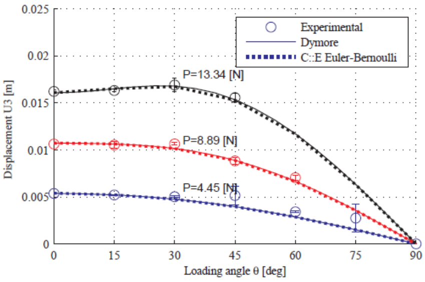
- The corotational approach for beam FE

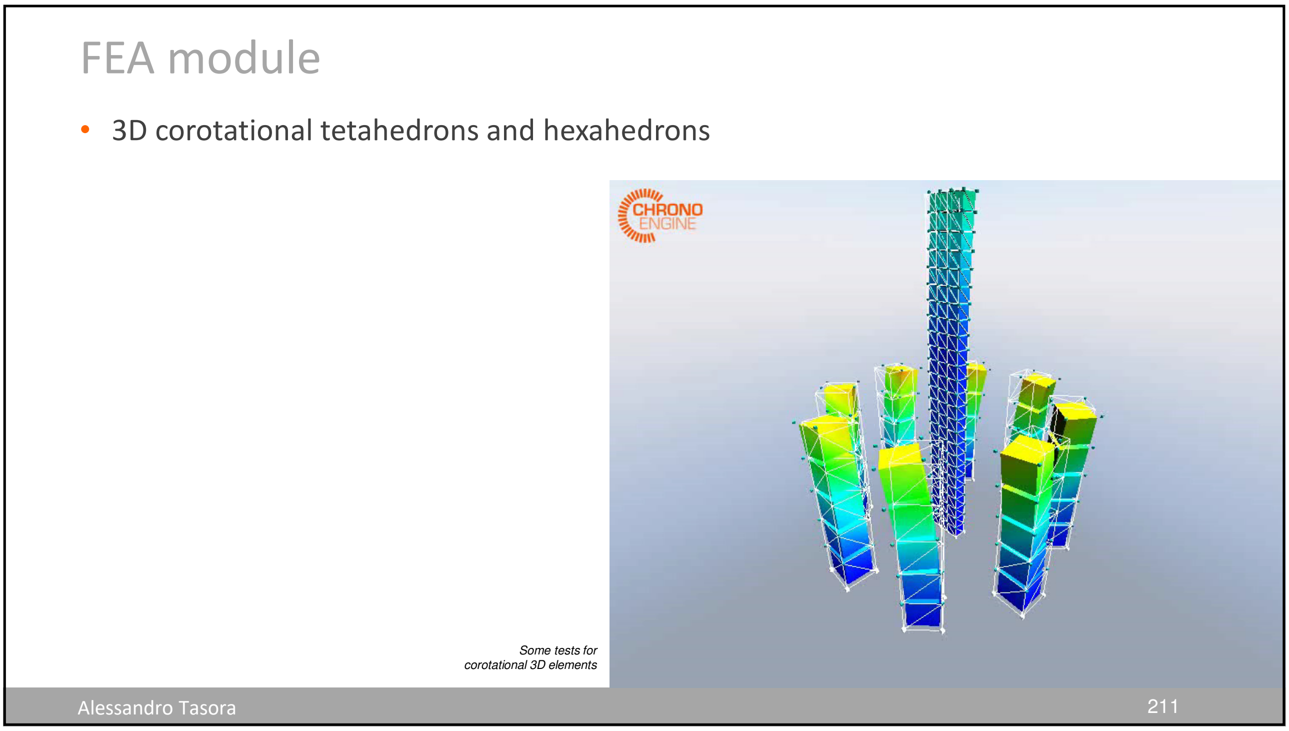
- Validation
  - Jeffcott rotor
  - Princeton beam
  - Lateral buckling
  - …





Example: the Princeton beam experiment, chordal and flapwise deflection

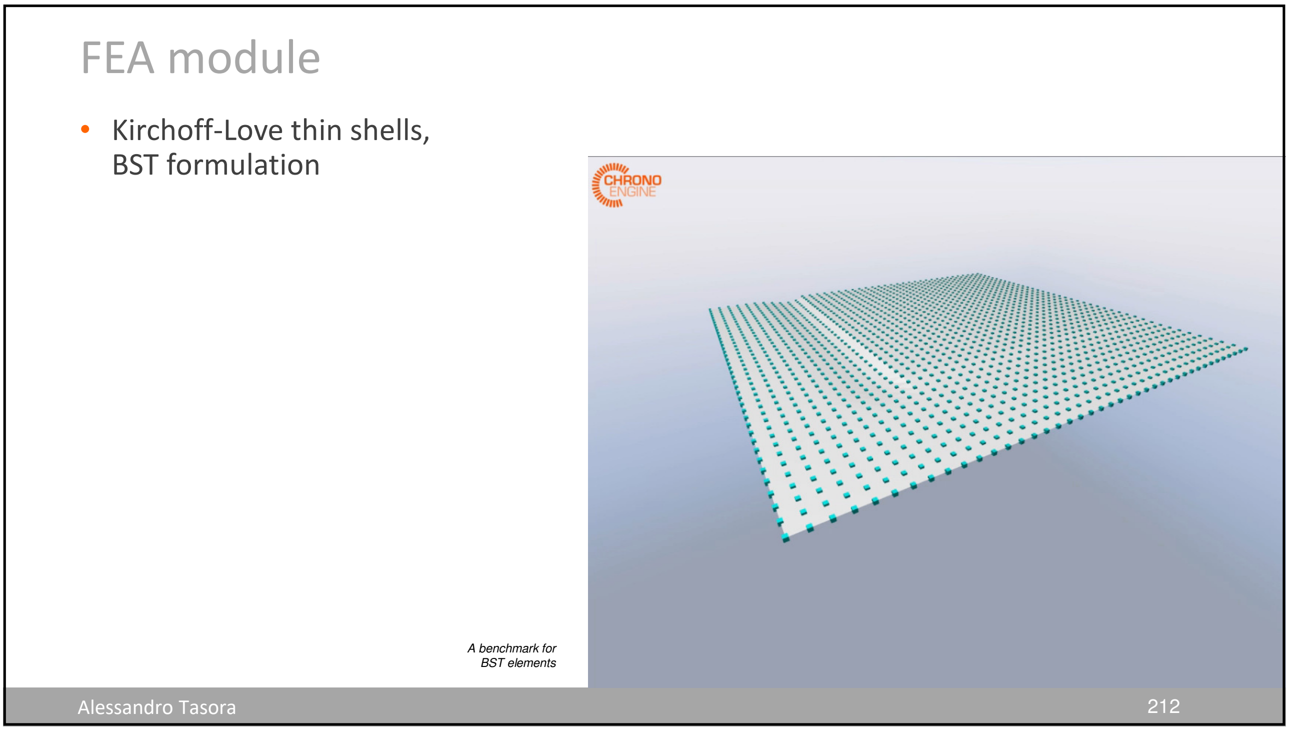




FEA module

- 3D corotational tetahedrons and hexahedrons

Some tests for corotational 3D elements



FEA module

- Kirchoff-Love thin shells, BST formulation

A benchmark for BST elements

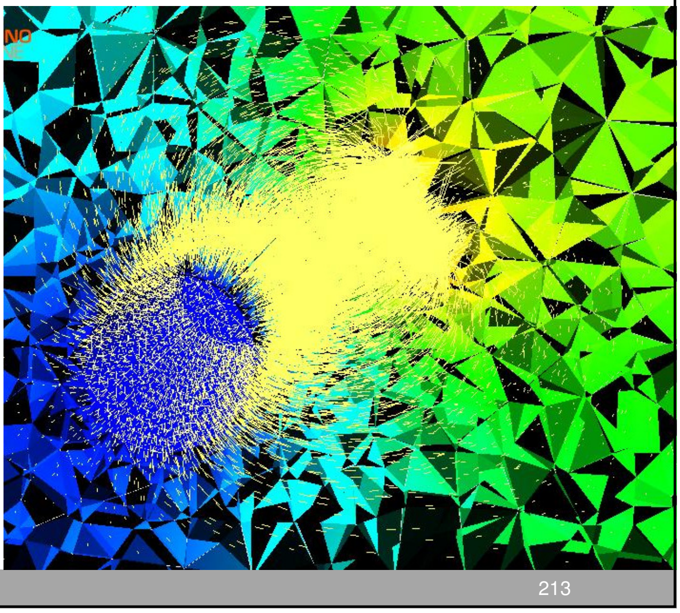
FEA module

- Other types of analysis

- Electrostatics

$$\nabla^2\varphi = -\frac{\rho_f}{\varepsilon}$$

$$\mathbf{E} = -\nabla\varphi$$



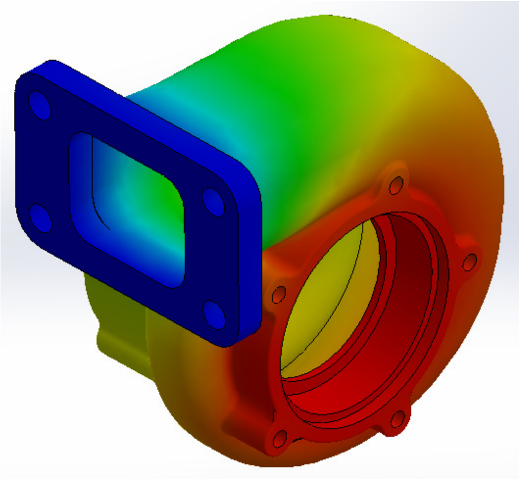
Example: Chrono::Engine solution for the E field between a 0kV cylinder and a 23kV plate

FEA module

- Other types of analysis

- Thermal
  - steady state
  - transient

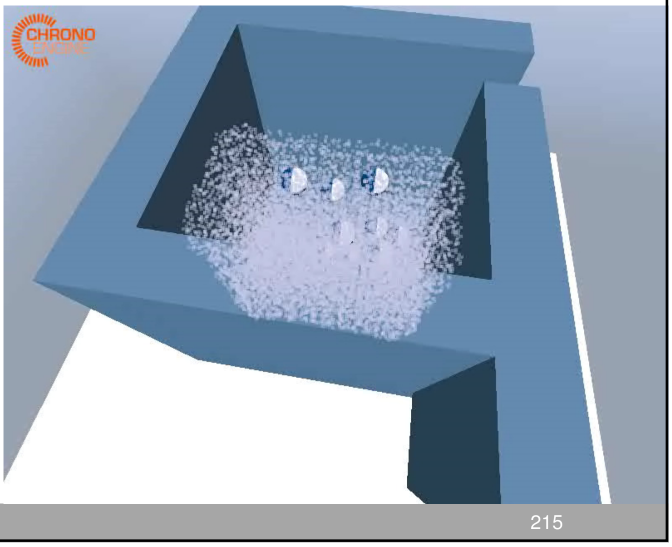
Example: turbo casing with Dirichlet boundary condition



$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + \frac{1}{c_p\rho}q$$

FLOW module

- SPH:



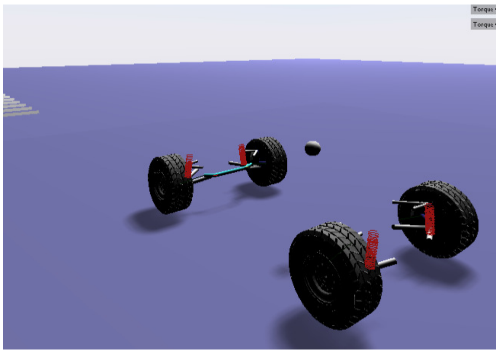
- *(work in progress)*

VEHICLE module

Gear: forward, n.gear: 1

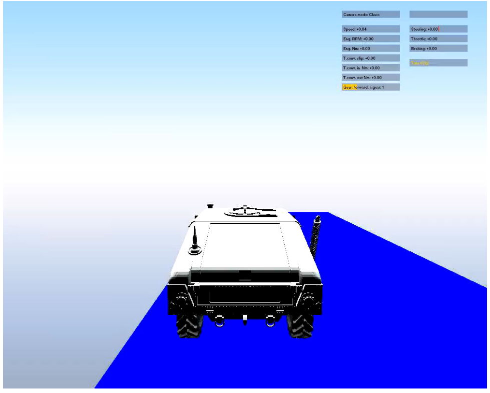
Torque wheel L: +0.00

Torque wheel R: +0.00

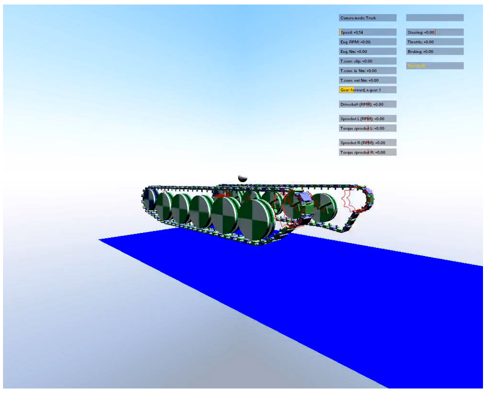


VEHICLE module

- templated vehicles: tracked, wheeled, multi-axle, etc.
- 1D power train, driveline & control
- granular soils, deformable tires (shells, multi-layered orthotropic materials, solid lugs)



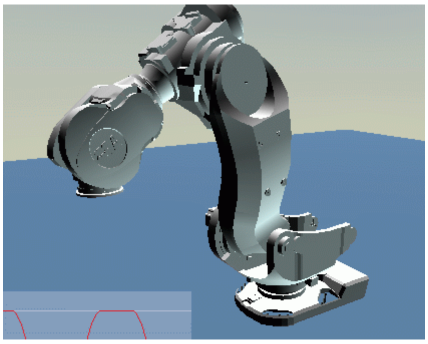
```
Camera mode: Chase

Speed: +0.04              Steering: +0.00
Eng. RPM: +0.00           Throttle: +0.00
Eng. Nm: +0.00            Braking: +0.00
T.conv. slip: +0.00
T.conv. in Nm: +0.00
T.conv. out Nm: +0.00
Gear: forward, n.gear: 1
```



```
Camera mode: Track

Speed: +0.54              Steering: +0.00
Eng. RPM: +0.00           Throttle: +0.00
Eng. Nm: +0.00            Braking: +0.00
T.conv. slip: +0.00
T.conv. in Nm: +0.00
T.conv. out Nm: +0.00
Gear: forward, n.gear: 1

Driveshaft (RPM): +0.00

Sprocket L (RPM): +0.00
Torque sprocket L: +0.00

Sprocket R (RPM): +0.00
Torque sprocket R: +0.00
```

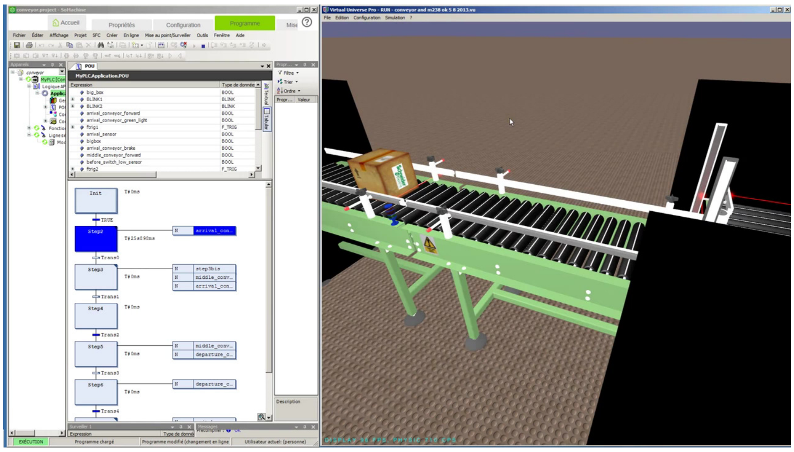
Other modules...

- CASCADE
- POSTPROCESSING
- MATLAB
- PARALLEL
- OPENGL
- IRRLICHT
- …



Embedding C::E in third party software

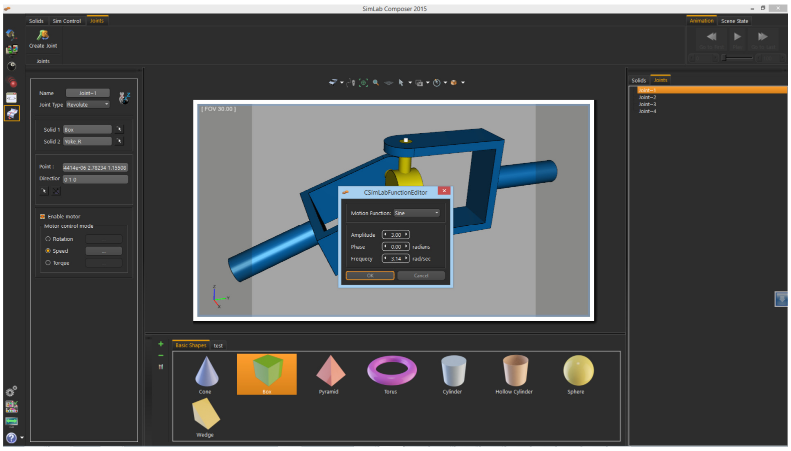
- Virtual Universe PRO
- *Company:* IRAI - France
- *Contact:* stephane.massart.irai@gmail.com



Embedding C::E in third party software

- SimLab Composer 2015
- *Company:* SimLab Soft - France
- *Contact:* Ashraf Sultan asultan@simlab-soft.com

Simlab Soft
3D Software Done Right

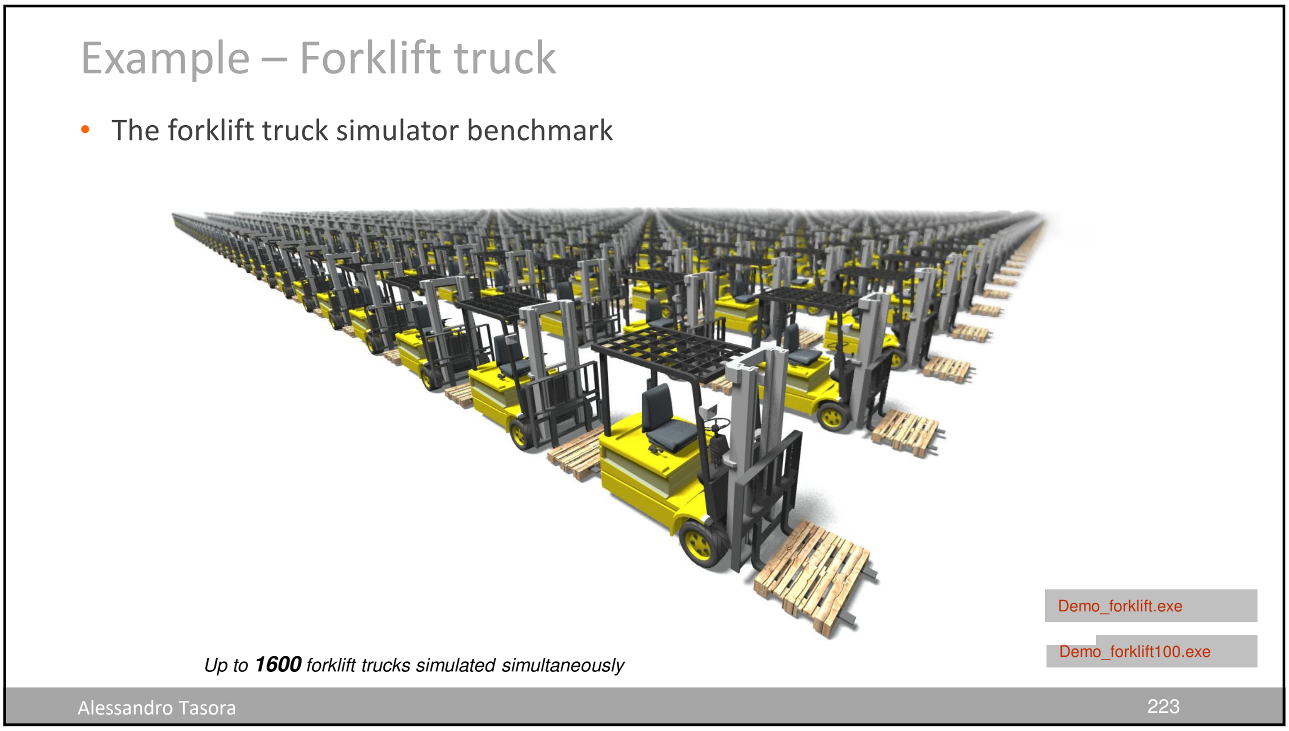


Name: Joint~1
Joint Type: Revolute
Solid 1: Box
Solid 2: Yoke_R
Point : :4414e-06 2.78234 1.15508
Direction: 0 1 0
☒ Enable motor
Motor control mode
○ Rotation
● Speed
○ Torque

Motion Function: Sine
Amplitude: 3.00
Phase: 0.00 radians
Frequecy: 3.14 rad/sec

9.
EXAMPLES AND APPLICATIONS

"For a list of all the ways technology has failed to improve the quality of life, please press three"
Alice Kahn



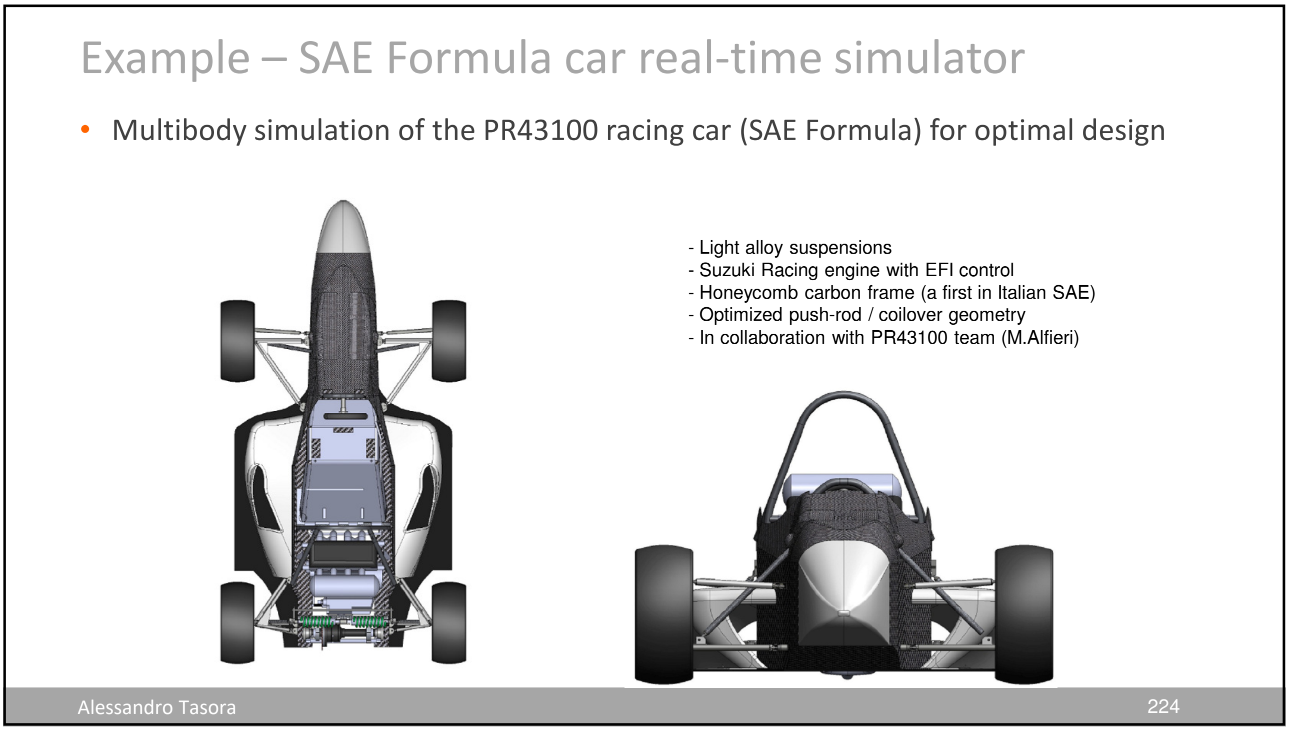
Example – Forklift truck

- The forklift truck simulator benchmark

Demo_forklift.exe

Demo_forklift100.exe

*Up to **1600** forklift trucks simulated simultaneously*



Example – SAE Formula car real-time simulator

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

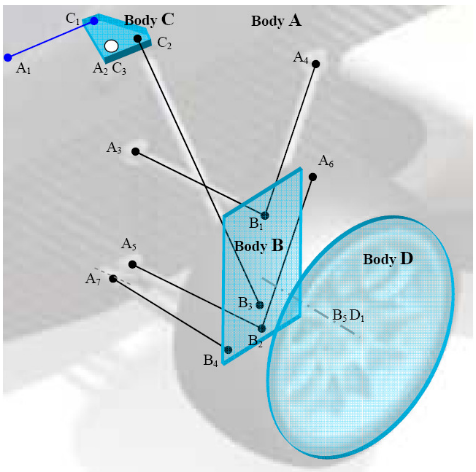
- Light alloy suspensions
- Suzuki Racing engine with EFI control
- Honeycomb carbon frame (a first in Italian SAE)
- Optimized push-rod / coilover geometry
- In collaboration with PR43100 team (M.Alfieri)

Example – SAE Formula car real-time simulator

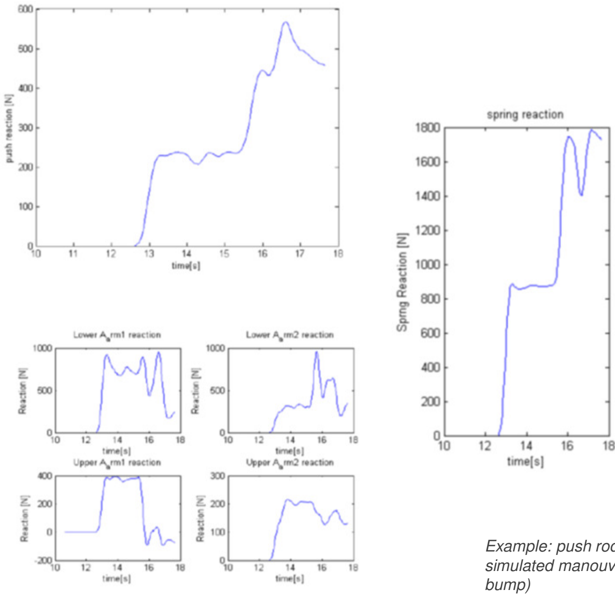
- **Special model based on 13 rigid bodies and 43 constraints**
- **Car model with 14 DOFs (78 DOFs unconstrained)**

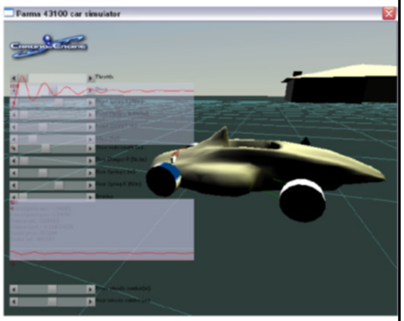
Bodies:

- 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





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

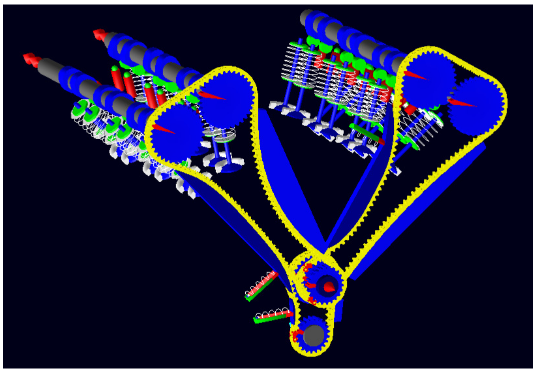
Example - Engines

- Simulation of high performance engines

collaboration with

F.Pulvirenti, C. Autore et al., Ferrari Auto

- Valve train & timing chain *with Adams + FEV*
- Mixed 3D-1D multibody engine model *with Chrono*
    - *dampers*
    - *Etc.*



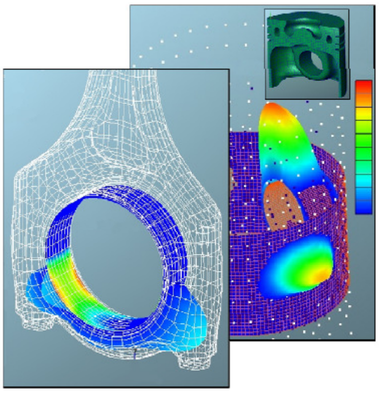
Example - Engines

- Simulation of high performance engines

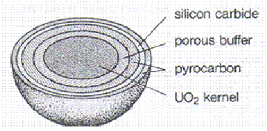
collaboration with

F.Pulvirenti, C. Autore et al., Ferrari Auto

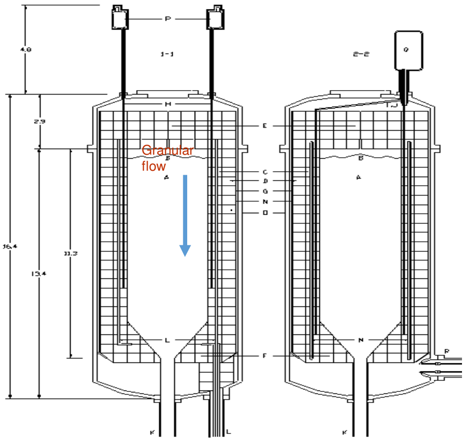
- Engine crank train, TEHD, etc. *with AVL Excite*
    - *wear prediction*
    - *oil temperature*
    - *etc.*



Example - Simulating the PBR nuclear reactor



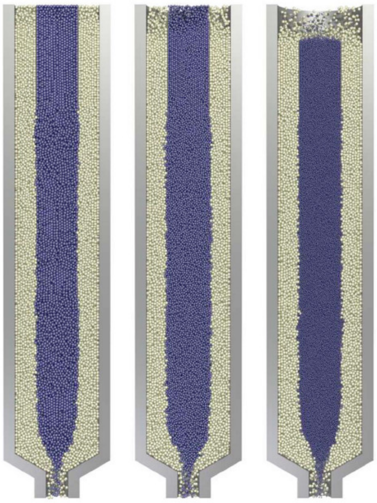
- The PBR nuclear reactor:
    - Fourth generation design
    - Inherently safe, by Doppler broadening of fission cross section
    - Helium cooled > 1000 °C
    - Can crack water (mass production of hydrogen)
    - Continuous cycling of 360’000 graphite spheres in a pebble bed



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

Example - Simulating the PBR nuclear reactor

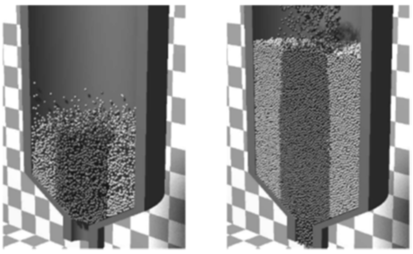
- The 360’000 spheres have different radii, % of actinides, etc.
- Most important: central spheres should have less Uranium/Thorium.
- Problem of bidisperse granular flow with dense packing.
- Previous attempts: DEM methods on supercomputers at Sandia Labs (but introducing compliance!)



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

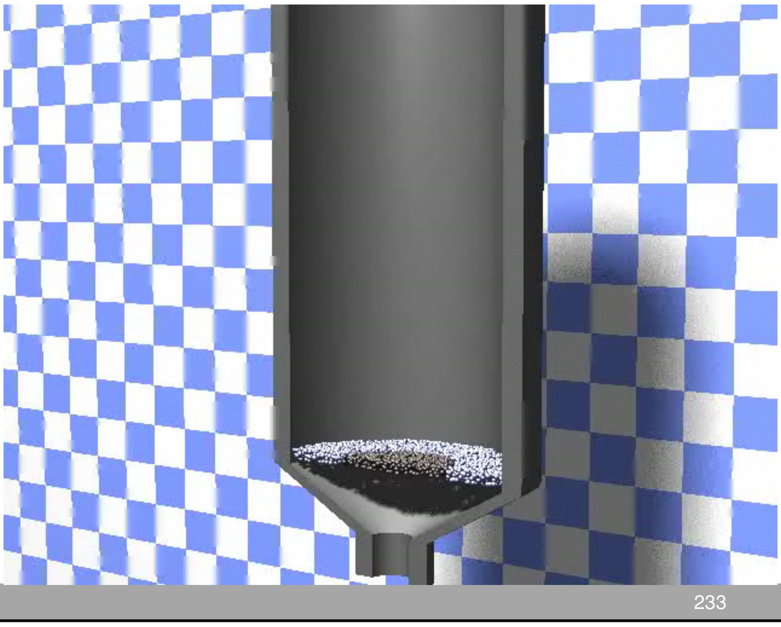
Example - Simulating the PBR nuclear reactor

- Our method can simulate systems with one million of frictional contacts:
  - *with rigid bodies (no fake springs-dashpots)*
  - *non-smooth DVI approach requires one day on a PC where a supercomputer required a week using smooth ODE.*



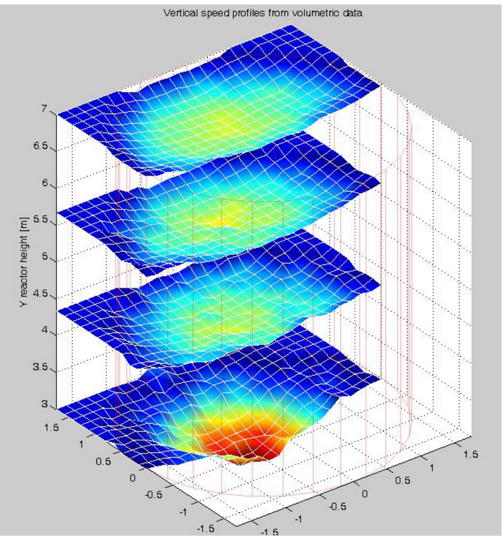
Example - Simulating the PBR nuclear reactor

- *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



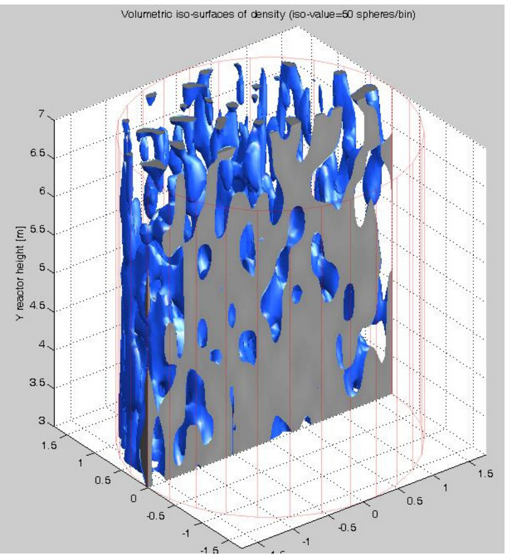
Example - Simulating the PBR nuclear reactor

- Example of results

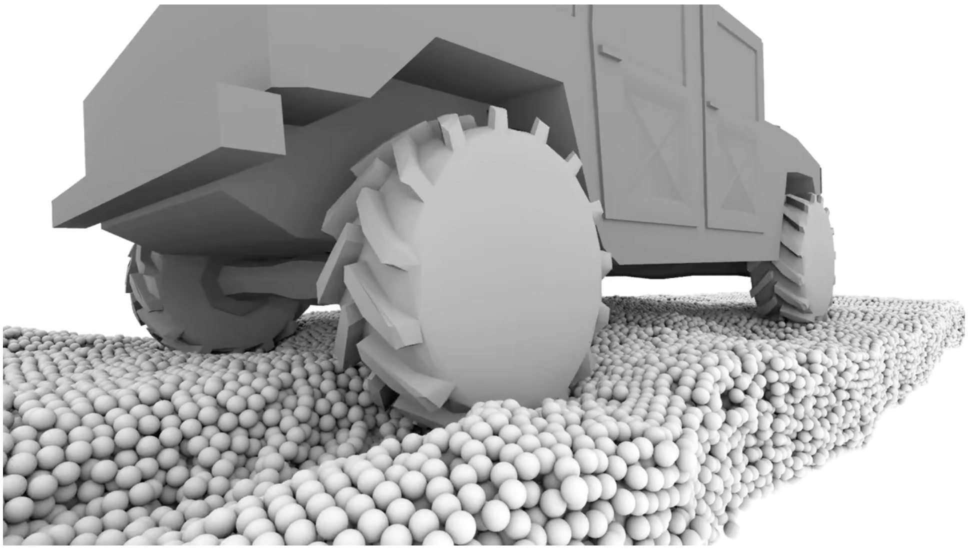


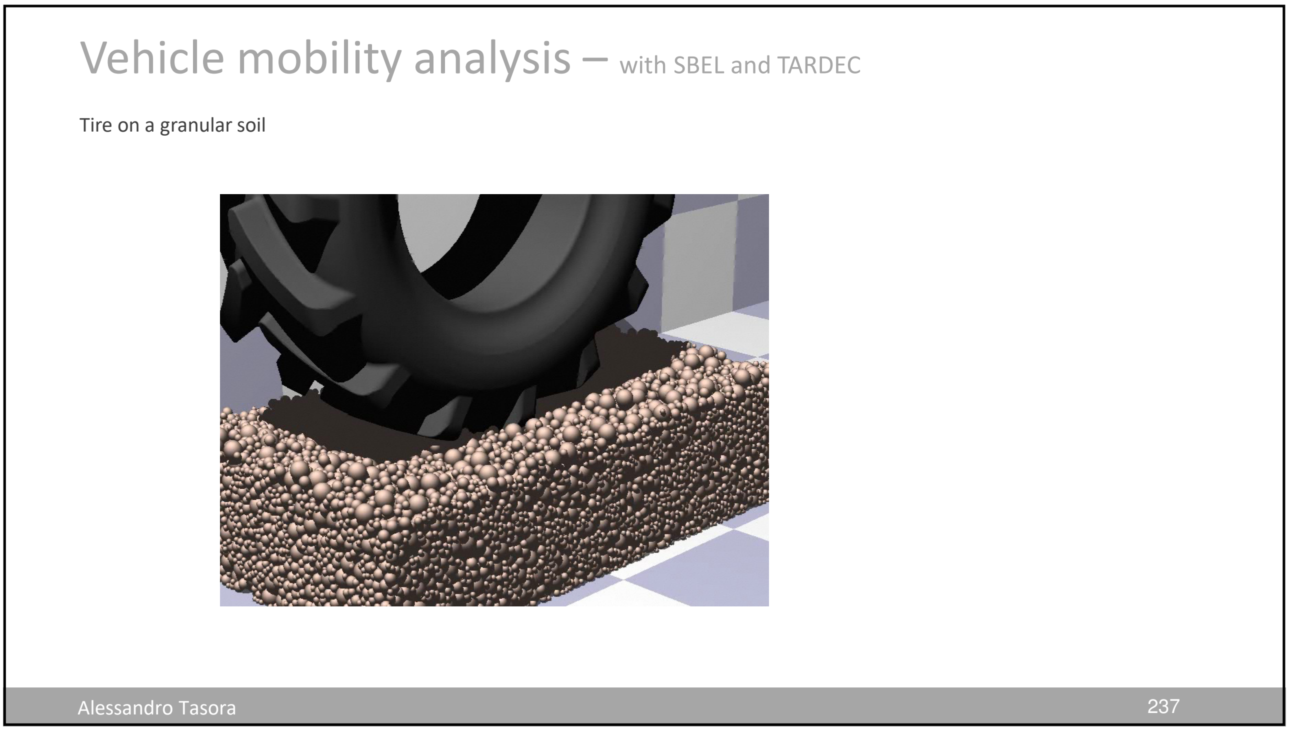
Example - Simulating the PBR nuclear reactor

- Example of results



Vehicle mobility analysis – with SBEL and TARDEC

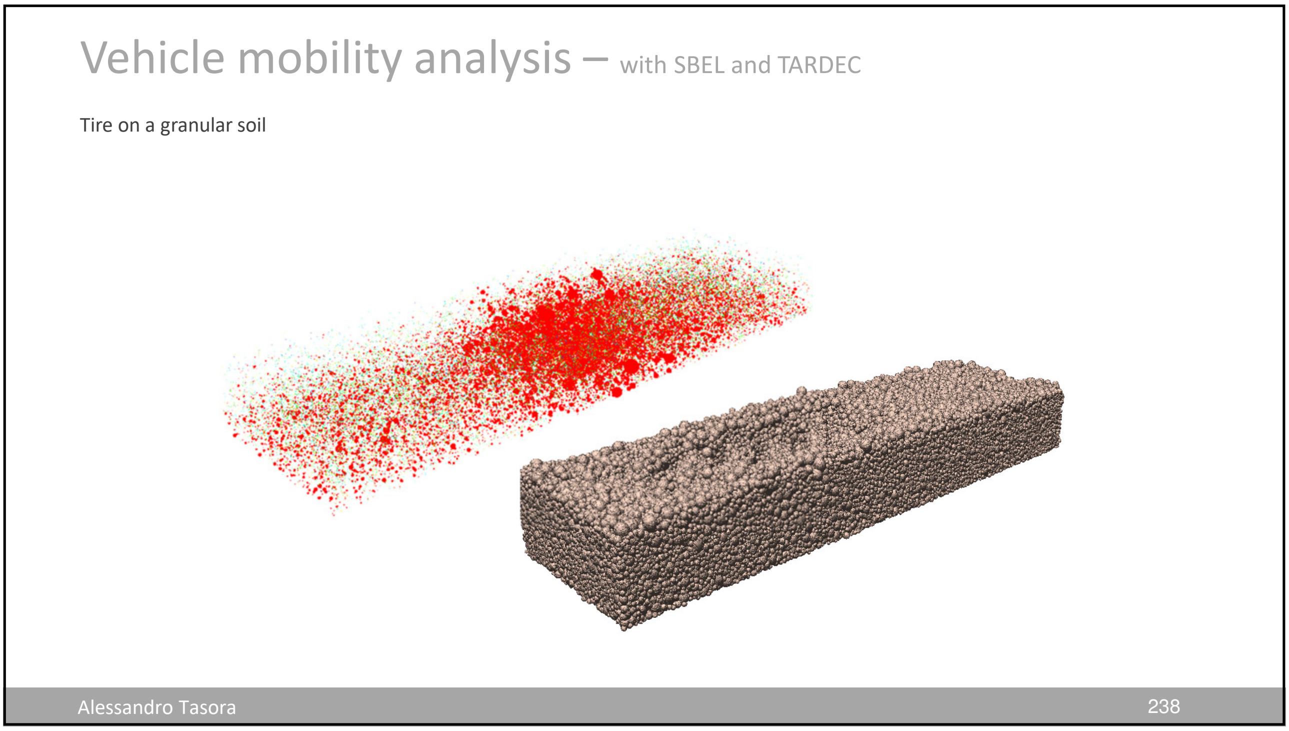




Vehicle mobility analysis – with SBEL and TARDEC

Tire on a granular soil

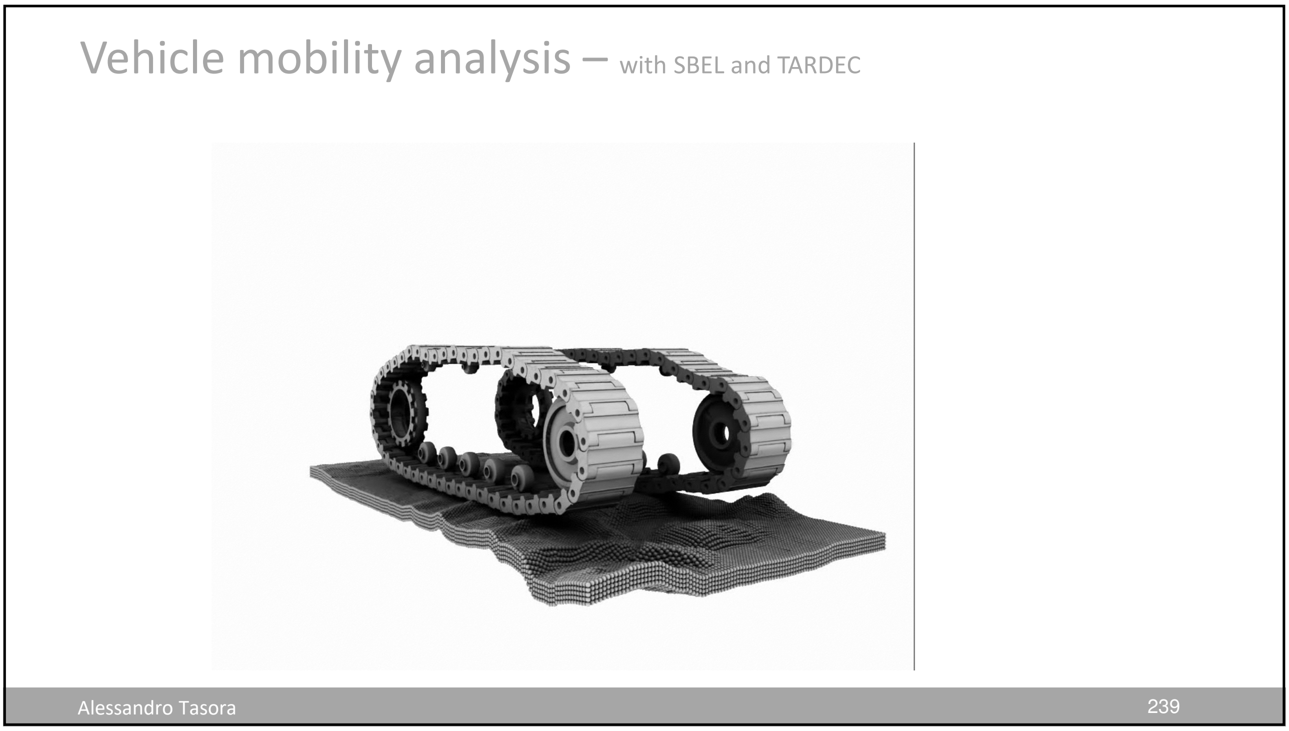
Alessandro Tasora 237



Vehicle mobility analysis – with SBEL and TARDEC

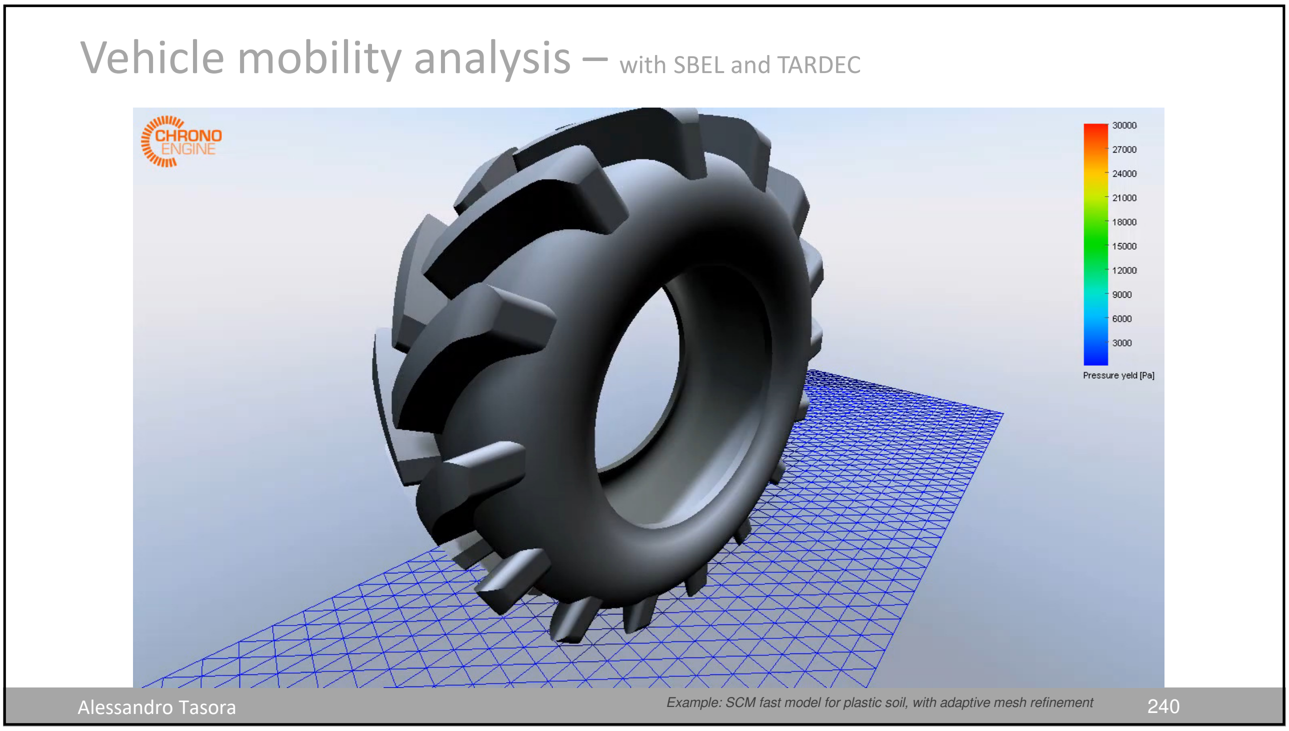
Tire on a granular soil

Alessandro Tasora 238



Vehicle mobility analysis – with SBEL and TARDEC

Alessandro Tasora 239

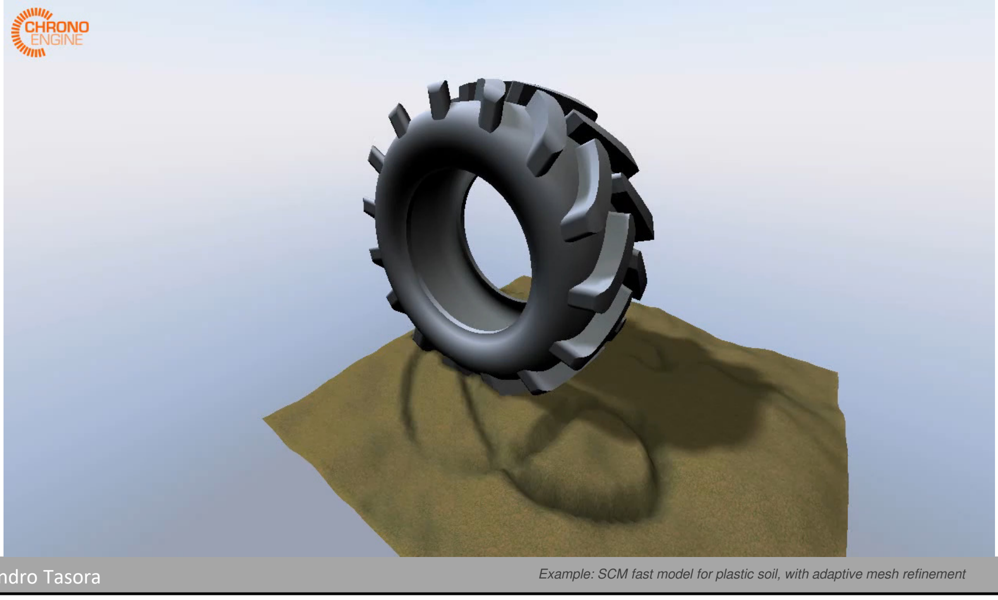


Vehicle mobility analysis – with SBEL and TARDEC

Example: SCM fast model for plastic soil, with adaptive mesh refinement

Alessandro Tasora 240

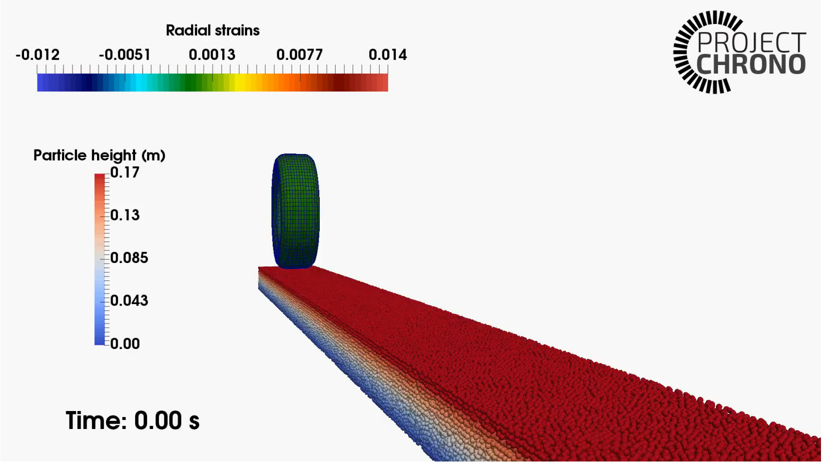
Vehicle mobility analysis – with SBEL and TARDEC



Example: SCM fast model for plastic soil, with adaptive mesh refinement

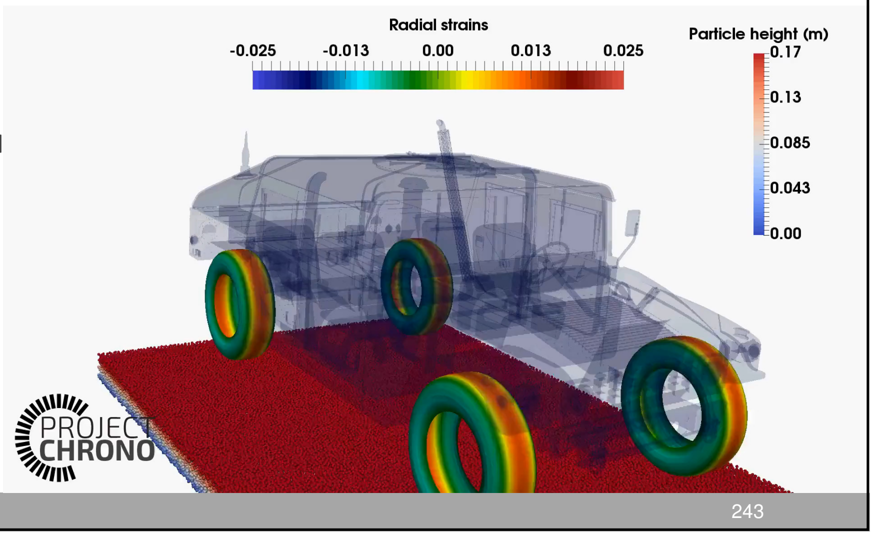
Tire-ground interaction

In collaboration with Dan Negrut, Radu Serban (University of Wisconsin), Hiroyuki Sugiyama (University of Iowa) et al.



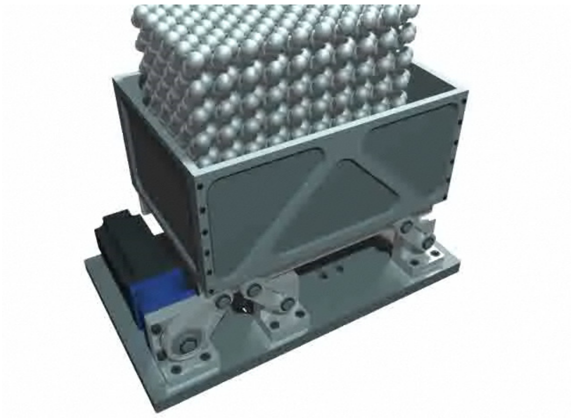
Tire-ground interaction

- FEA:
  - ANCF shells for tires
  - Multi-layer material
- Hybrid integration:
  - Granular soil with DVI
  - Tires with HHT



Particulated flows in industry

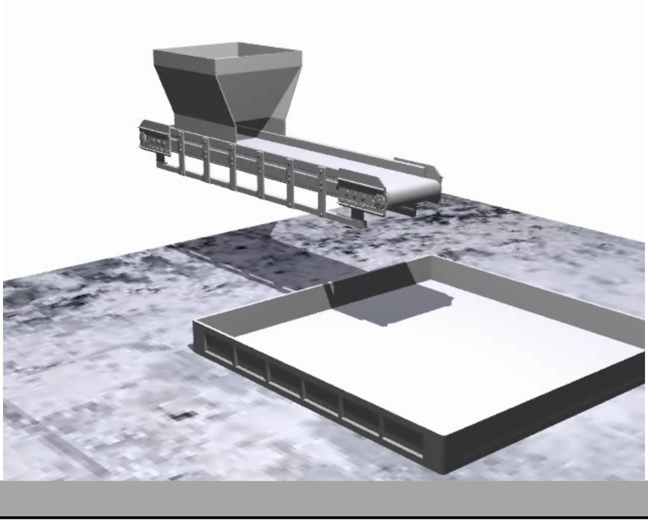
- Part feeders, size segregation devices, etc.



Example: size segregation device: about 2000 interacting objects

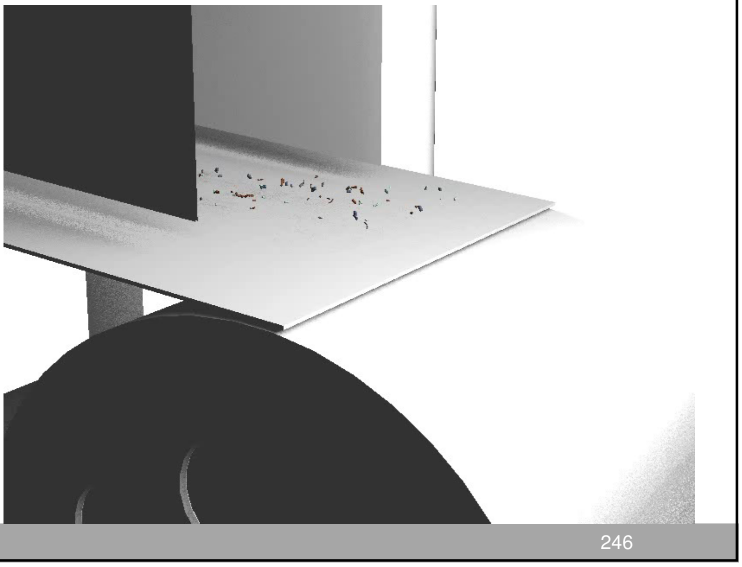
Processing of waste material

- Conveyor belts, hoppers, ...



Processing of waste material

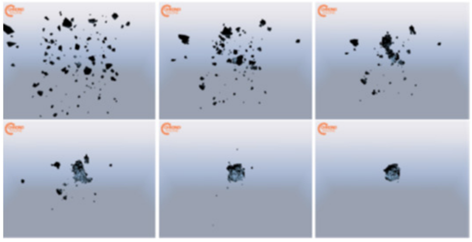
- Separating materials in waste processing plants:



Example of CES device simulated with ProjectChrono software (A.Tasora, I. Critelli 2014)

Space

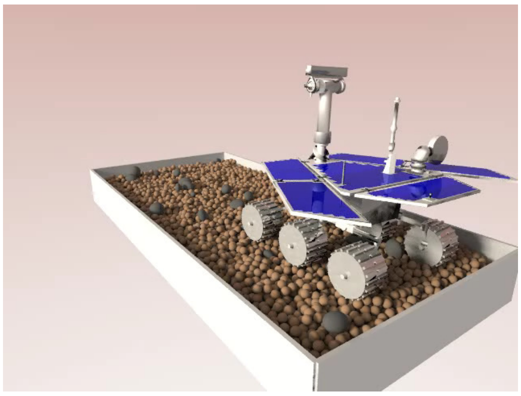
- Simulation of aggregation of small bodies



ProjectChrono simulation by F.Ferrari, Politecnico di Milano - JPL

Space

The Mars rover on a granular soil, simulated with ProjectChrono

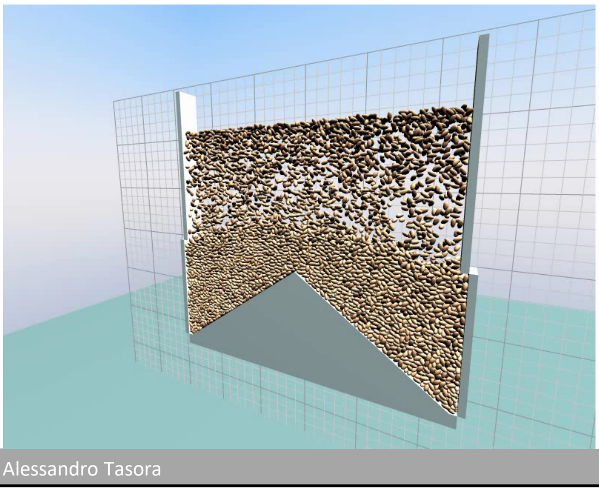


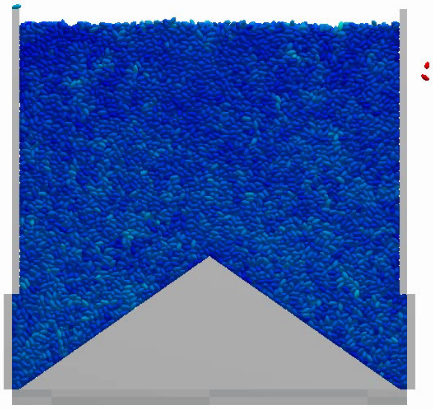
In collaboration with D.Negrut (USA) and SBEL labs *[test]*

Granular flows

Simulation of the lateral discharge of inverted-V silos:

ProjectChrono simulations by A.Tasora, 2018

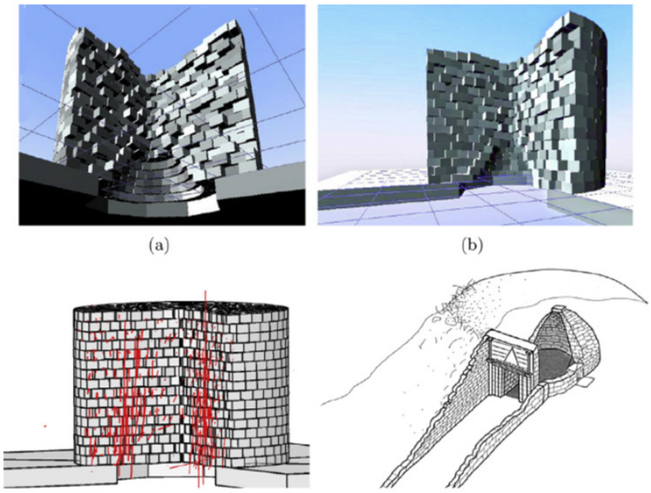




Masonry structures

In collaboration with Gianni C. Royer (University of Parma) and Valentina Beatini (University of Kayseri)

- The Non-Smooth dynamic approach can help studying ancient buildings
- Better insight in cases where traditional methods (ex. thrust line) cannot be used

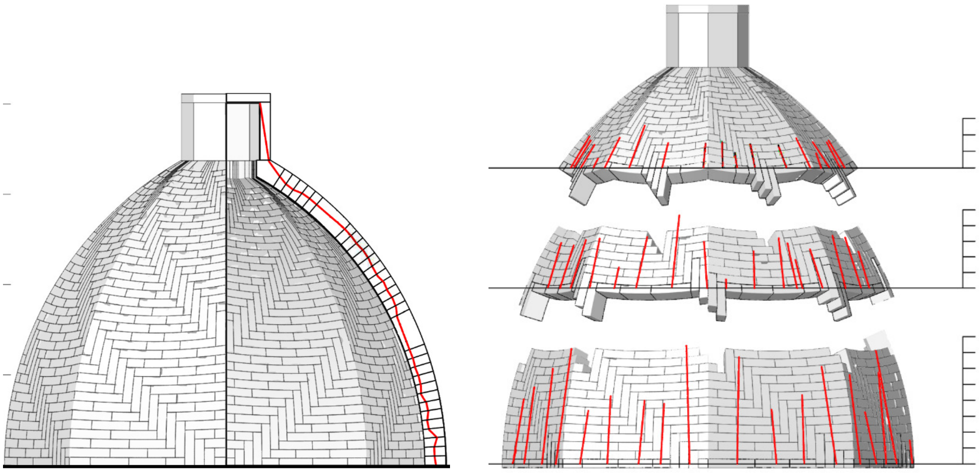


(a) (b)

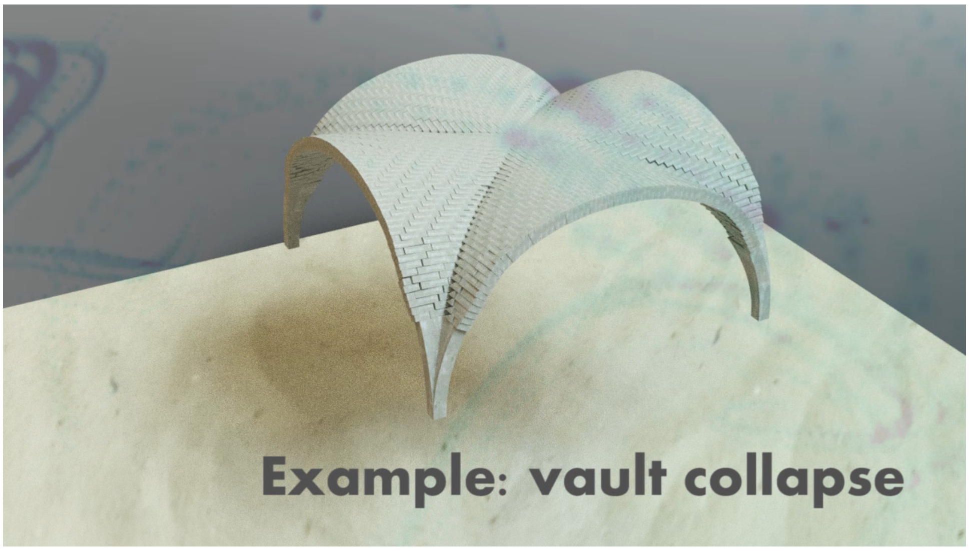
Tomb of Clytemnestra, Mycenae, c. 1500 b.C.

Masonry structures

- The dome of Brunelleschi



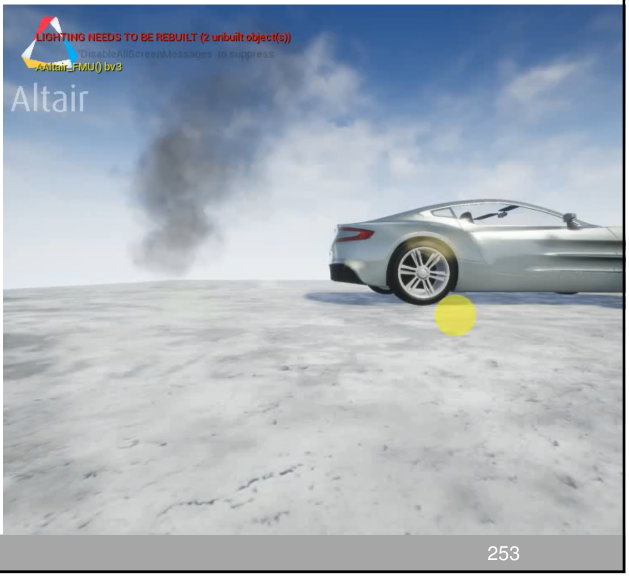
Seismic engineering



Vehicle dynamics

Modelica-based real-time vehicle simulator

(in collaboration with Altair)



LIGHTING NEEDS TO BE REBUILT (2 unbuilt object(s))

'DisableAllScreenMessages' to suppress

AAltair_FMU() bv3

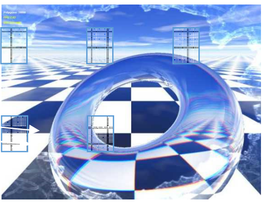
10. FUTURE CHALLENGES

“I think there’s a world market for about 5 computers.”
J. Watson, Chairman of the Board, IBM, 1948

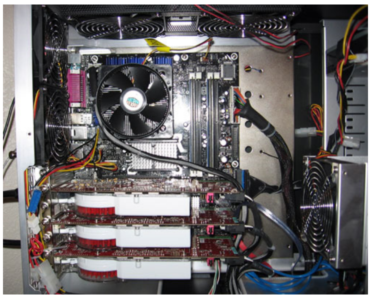
ΑΧΙΛΛΕΥΣ

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)

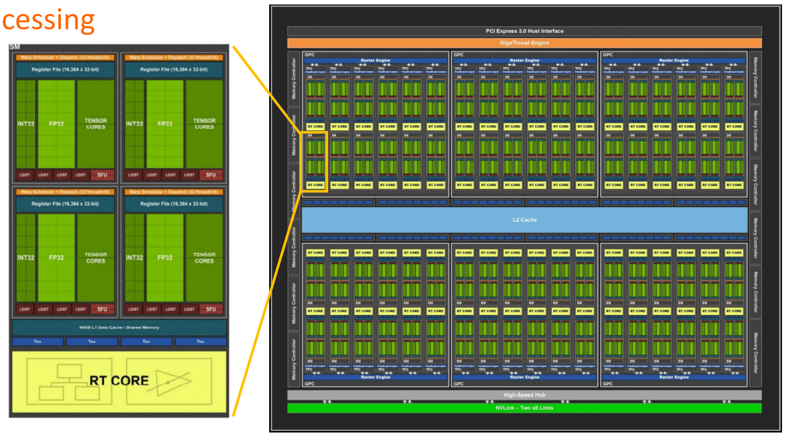


Note: multiple GPU? Yes!

(ex: 4x256=1024 stream processors)

GPU parallel computing

- Exploit GPU parallel processing

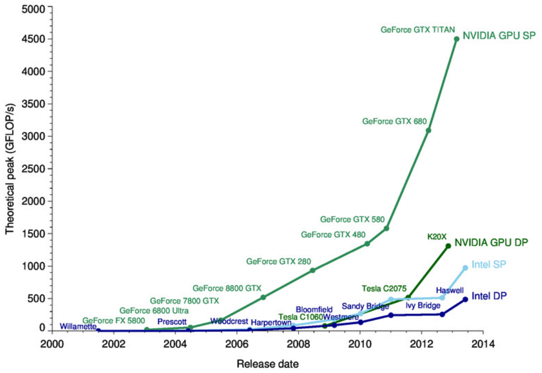


- Current NVIDIA GPU boards feature thousands of multiprocessors (cores), allowing more than 10 TFlop on a desktop system.
- Beware of
  - data transfer bottlenecks PC<->GPU
  - not always easy translation of serial C++ algos to parallel CUDA algos

GPU parallel computing

- Performance: > 4 TFLOP with recent GPU processors !!!

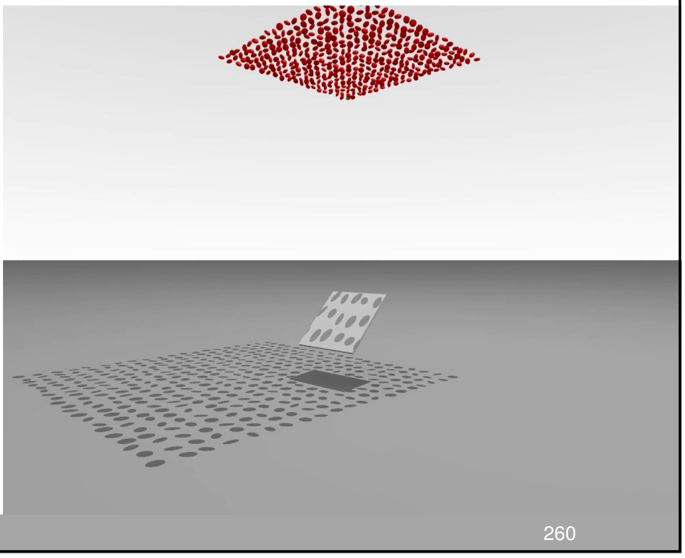
A single GeForce™ GPU board in 2012 was as powerful as four ASCI-RED 1996 supercomputers!



"Computers in the future may have only 1,000 vacuum tubes and perhaps only weigh 1 1/2 tons"
Popular Mechanics, 1949

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.

***> 10’000’000** 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

Supercomputing

Ex: MIRA supercomputer at Argonne National Labs

- 10-petaFLOPS
- 786,432 processors
- power: 3.9 MW

Heterogeneous parallelism

A solution for *very large* multibody problems:

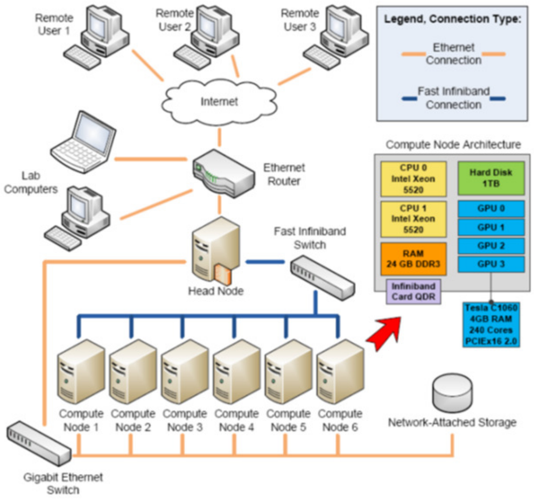
- use a cluster of computing nodes connected with Infiniband..
- ..each computer fitted with one or multiple GPU boards

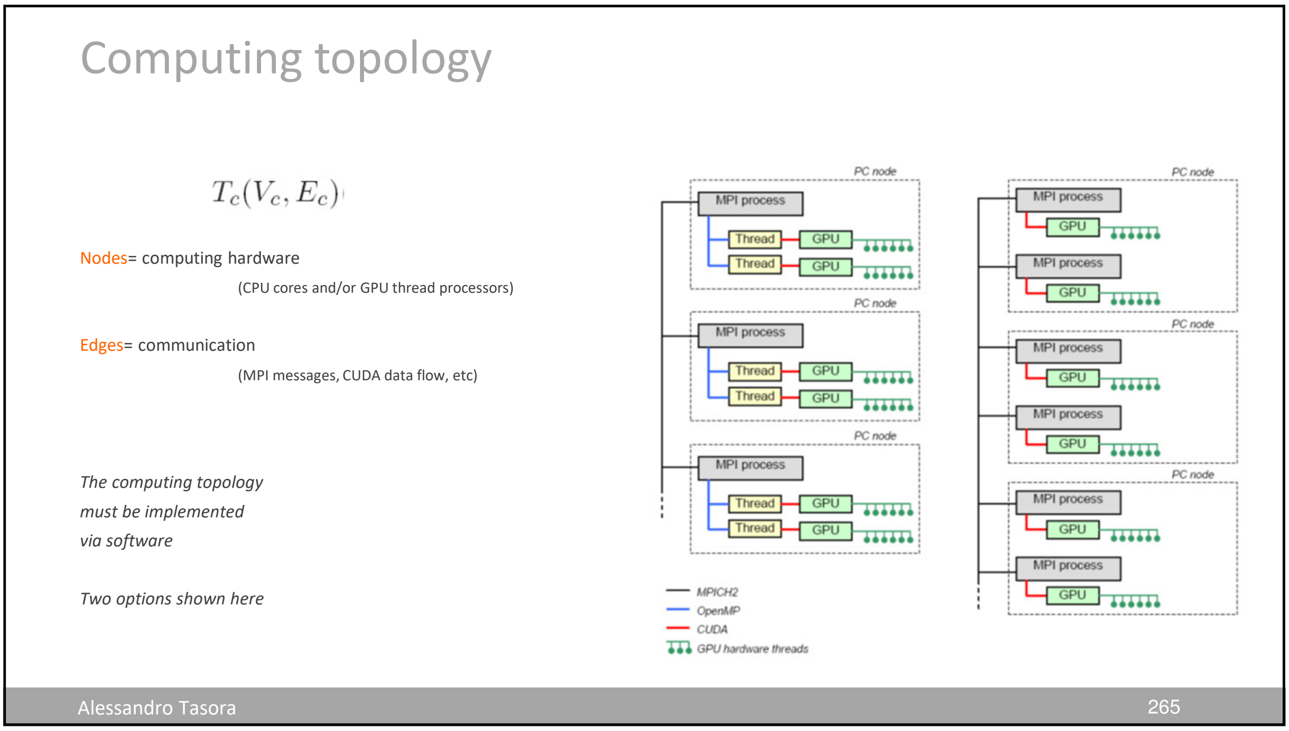
→ ***MPI*** *is used to handle the node-level parallelism*

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

Heterogeneous parallelism

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





Computing topology

$T_c(V_c, E_c)$

Nodes= computing hardware

(CPU cores and/or GPU thread processors)

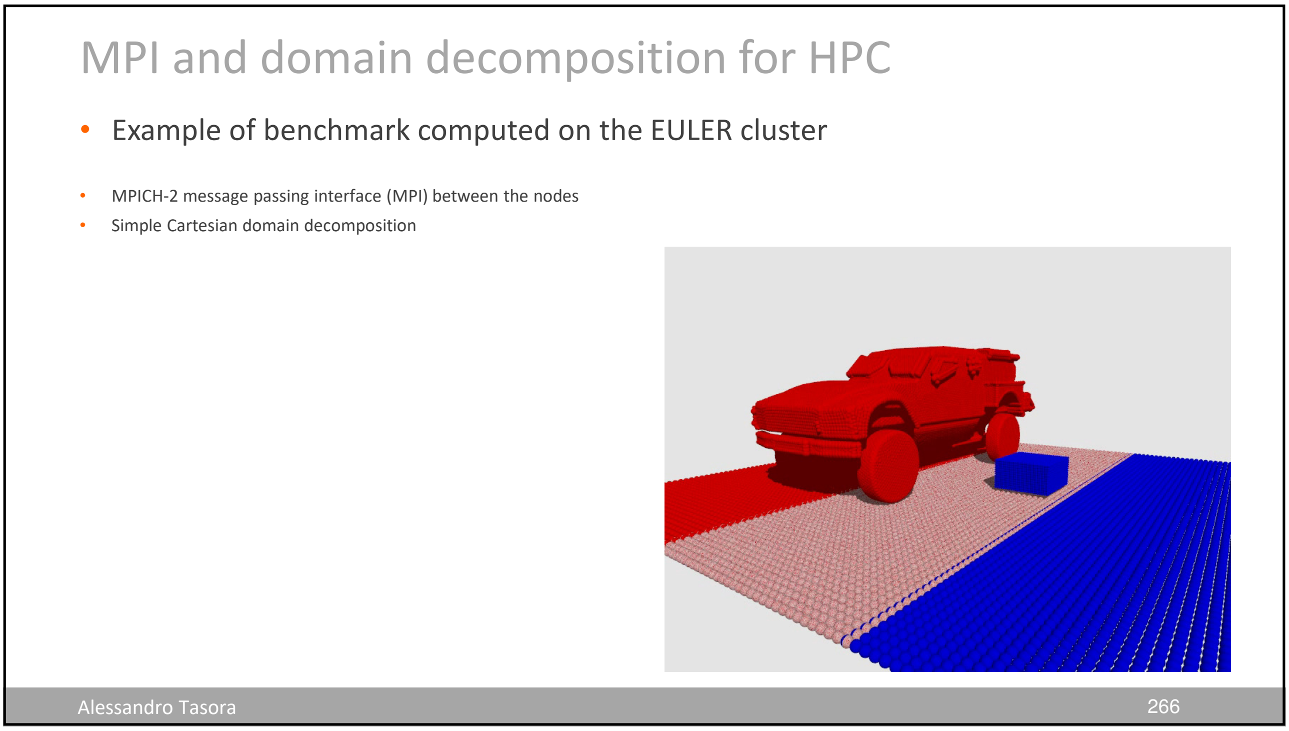
Edges= communication

(MPI messages, CUDA data flow, etc)

The computing topology must be implemented via software

Two options shown here

Alessandro Tasora 265



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

Alessandro Tasora 266

THANKS

Any question?

Contacts:

alessandro.tasora@unipr.it

http://projectchrono.org

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

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**