NUMERICAL METHODS FOR LARGE SCALE NON-SMOOTH MULTIBODY PROBLEMS
Alessandro Tasora

Research network

Radu Serban
Dan Negrut et al.

Mihai Anitescu

Alessandro Tasora, Dario Mangoni
Simone Benatti et al.

POLITECNICO MILANO

THE UNIVERSITY OF CHICAGO

UNIVERSITÀ DI PARMA
1. INTRODUCTION

Motivation of large scale multibody dynamics
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*

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
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
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
  - ...

Alessandro Tasora
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 dynamical simulation
  • *NOTE: 48’000 million of dollars of revenues in videogames, A relevant market for physical simulation software.*

Applications of multibody methods

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

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?

\[
\begin{bmatrix}
0_r \\
1_r
\end{bmatrix} = \begin{bmatrix}
0_{rx} & 0_{ry} & 0_{rz} \\
1_{rx} & 1_{ry} & 1_{rz}
\end{bmatrix}^T
\]

- *Affine linear transformation:*

\[
\begin{bmatrix}
1_r
\end{bmatrix} = [\tilde{\alpha} A] \begin{bmatrix}
0_r \\
1_d
\end{bmatrix}
\]
Rigid body motion

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

• Example (in 2D):

$$\begin{bmatrix} 1_{rx} \\ 1_{ry} \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 0_{rx} \\ 0_{ry} \end{bmatrix} + \begin{bmatrix} 1_{dOx} \\ 1_{dOy} \end{bmatrix}$$

• $[A]$ 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

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

Simple rotation, no translation:

$$\begin{bmatrix} 1_{rx} \\ 1_{ry} \\ 1_{rz} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 0_{rx} \\ 0_{ry} \\ 0_{rz} \end{bmatrix}$$

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

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

$$\begin{bmatrix} 0_{rx} \\ 0_{ry} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \begin{bmatrix} 1_{rx} \\ 1_{ry} \end{bmatrix}$$
Rigid body motion

• “Direct” transformation:

\[ \{^0 r\} = [^0 A] \{^1 r\} + \{^0 d\} \]

• “Inverse” transformation:

\[ \{^1 r\} = [^0 A]^{-1} (\{^0 r\} - \{^0 d\}) = [^0 A]^{-1} (\{^0 r\} - \{^0 d\}) = [^0 A]^{-1} \{^0 A\} (\{^0 r\} - \{^0 d\}) = [^1 A] (\{^0 r\} - \{^0 d\}) \]

Rigid body motion

• Each body should have 3 (translation \( d \)) + 3x3=9 (rotation \([^0 A]\)) coordinates, that is 12 scalars.

• Some would be redundant...

• Is it possible to make \([^0 A]\) dependant on only three coordinates? \([^0 A(a,b,c)] = f(a,b,c)\)
Rigid body motion

- Make \( ^0_{\text{i}}A \) dependant on three angles?

- Different options, depending on the sequence of 3 rotations!

- Ex:

\[
\{1^r\} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_1 & \sin \theta_1 \\ 0 & -\sin \theta_1 & \cos \theta_1 \end{bmatrix} \{^0_{\text{i}}r\}
\]

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

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

\[
\{3^r\} = \begin{bmatrix} 2^r & A & \{1^r\} \{^0_{\text{i}}r\} = \begin{bmatrix} 0 \end{bmatrix} \{^0_{\text{i}}r\}
\]

Rigid body motion

- Ex: make \( ^0_{\text{i}}A \) dependant on three ‘Eulero’ angles:

- But also:
  - ‘Cardano’ angles
  - ‘HPB’ angles
  - ‘XYZ’ angles,
  - etc..

- See also ‘Rodriguez parameters’

\[
\{3^r\} = \begin{bmatrix} 3^r & A & \{2^r\} \{^0_{\text{i}}r\} = \begin{bmatrix} 2^r \end{bmatrix} \{^0_{\text{i}}r\}
\]

Alessandro Tasora
Rigid body motion

- Example: sequence X-Y-Z

\[
[A] = [T]_y [T]_x [T]_z = \begin{bmatrix}
\xi_{y,z} & \xi_{z,z} + \xi_{x,z} \xi_{y,z} & \xi_{z,z} - \xi_{x,z} \xi_{y,z} \\
\xi_{x,z} - \xi_{x, y} \xi_{z,z} & \xi_{x,y} \xi_{z,z} + \xi_{x,z} & \xi_{x,y} \xi_{z,z} - \xi_{x, z} \xi_{y,z} \\
\xi_{x,y} & -\xi_{x,y} \xi_{z,z} & \xi_{x,y} \xi_{z,z} + \xi_{x, z} \xi_{y,z}
\end{bmatrix}
\]

- Example: sequence Y-Z-X

\[
[A] = [T]_z [T]_x [T]_y = \begin{bmatrix}
\xi_{x,z} & \xi_{y,z} - \xi_{x,z} \xi_{y,z} & \xi_{z,z} - \xi_{x,z} \xi_{y,z} \\
\xi_{y,z} + \xi_{x,y} \xi_{z,z} & \xi_{x,y} \xi_{z,z} + \xi_{x,z} & \xi_{x,y} \xi_{z,z} - \xi_{x, z} \xi_{y,z} \\
\xi_{x,y} & -\xi_{x,y} \xi_{z,z} & \xi_{x,y} \xi_{z,z} + \xi_{x, z} \xi_{y,z}
\end{bmatrix}
\]

- NOTE: viceversa, how to compute \(\zeta, \xi, \eta\) from \([A]\) ?

\[
\eta = \text{asin} \left(-{A_{1, 1} / \cos(\zeta)}\right) ,
\zeta = \text{acos} \left({A_{1, 2} / \cos(\xi)}\right) \rightarrow \text{singularity for } \zeta = \pi/2 + n \pi \quad !!! (\text{Same for all sets of 3 angles!})
\]

Rigid body motion

- Angular velocity

\[
\begin{bmatrix}
\frac{d\boldsymbol{r}_m}{dt} \\
\frac{d\boldsymbol{r}_j}{dt} \\
\frac{d\boldsymbol{r}_k}{dt}
\end{bmatrix} = \begin{bmatrix}
\dot{\omega}_z & 0 & -\omega_x \\
0 & \dot{\omega}_y & -\omega_z \\
-\omega_y & \omega_z & 0
\end{bmatrix} \begin{bmatrix}
\tau_m \\
\tau_j \\
\tau_k
\end{bmatrix}
\]

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

\[
[0, \omega] = [0, A] [\omega] [0, A]^T
\]
Rigid body motion

- Angular velocity, velocity

\[ \ddot{\mathbf{r}} = \omega \times \dot{\mathbf{r}} \]

\[ \{v^0 \} = [^0\omega] \{v^0_r \} \]

\[ \{v^1 \} = [^1\omega] \{v^1_r \} \]

\[ \{v^0 \} = [^0A] \{v^1 \} \]

\[ \{v^0_r \} = [^0A] \{v^1_r \} \]

\[ \{v^1 \} = [^1A]^T [^0\omega] [^0A] \{v^1_r \} \]

\[ \{s^0 \} = [^0A] \{s^1 \} + [^0A] \{s^1_s \} \]

\[ \{s^0 \} = [^0A] \{s^1 \} \]


Rigid body motion

- Velocity of a point on a moving frame

\[ \{v^0_{vP} \} = \{v^0_{vO} \} + 2 [^0A] [^0\omega] \{v^0_{sP} \} \]

\[ = \{v^0_{vO} \} + 2 [^0A] [^0\omega] [^0A] \{s^0_{sP} \} \]

\[ = \{v^0_{vO} \} + 2 [^0A] [^0\omega] \{s^1_{sP} \} \]
Rigid body motion

- Acceleration of a point on a moving frame

\[
\begin{align*}
\frac{d}{dt} \begin{bmatrix} \vec{v}_p \\ \vec{a}_p \end{bmatrix} &= \begin{bmatrix} \vec{v}_p \\ \vec{a}_p + \left[ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \\ \end{bmatrix} \right] \left[ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \\ \end{bmatrix} \right] \vec{a}_p \\ \end{align*}
\]

- Rotation in 3D not 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

\[ q = e_0 + i \cdot e_1 + j \cdot e_2 + k \cdot e_3 \]
\[ i^2 = j^2 = k^2 = ijk = -1 \]
\[ q = (s, v) \quad q^* = (s, -v) \]
\[ \|q\| = q^* \circ q = (e_0^2 + e_1^2 + e_2^2 + e_3^2) \]

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

• Sum:

\[ \bar{c} = \bar{a} \pm \bar{b} = \]
\[ = (a_0 + a_1 \cdot i + a_2 \cdot j + a_3 \cdot k) \pm (b_0 + b_1 \cdot i + b_2 \cdot j + b_3 \cdot k) = \]
\[ = (a_0 \pm b_0) \pm (a_1 \pm b_1) \cdot i \pm (a_2 \pm b_2) \cdot j \pm (a_3 \pm b_3) \cdot k \]

• Product:

\[ \bar{c} = \bar{a} \cdot \bar{b} = (s_a s_b - v_a \cdot v_b, s_a v_b^b + s_b v_a + v_a^b \times v_b) \]
\[ = (a_0 + a_1 \cdot i + a_2 \cdot j + a_3 \cdot k) \cdot (b_0 + b_1 \cdot i + b_2 \cdot j + b_3 \cdot k) = \]
\[ = (a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3) + \]
\[ + (a_0 b_1 + a_1 b_0 + a_2 b_3 - a_3 b_2) \cdot i + \]
\[ + (a_0 b_2 - a_1 b_3 + a_2 b_0 + a_3 b_1) \cdot j + \]
\[ + (a_0 b_3 + a_1 b_2 - a_2 b_1 + a_3 b_0) \cdot k \]
\[ \quad a (\bar{b} \cdot \bar{c}) = (a b) \cdot \bar{c} \quad a \neq b \neq c \]
Quaternions

- Conjugate:
  \[ q = (q_0 + q_1 i + q_2 j + q_3 k) \]
  \[ q^* = (q_0 - q_1 i - q_2 j - q_3 k) \]
  \[ (\bar{a}^*)^* = \bar{a} \]
  \[ (\bar{a} \bar{b})^* = b^* \bar{a}^* \]
  \[ (\bar{a} + \bar{b})^* = \bar{a}^* + \bar{b}^* \]
  \[ \bar{q} \bar{q}^* = (q_0^2 + q_1^2 + q_2^2 + q_3^2) \]
  \[ |\bar{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \]

- Inverse:
  \[ q^{-1} q = 1 \]
  \[ q^{-1} = \bar{q} \frac{1}{|\bar{q}|^2} \]
  \[ |\bar{q}| = 1 \implies q^{-1} = q^* \]

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: $|\bar{q}| = 1$.

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

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

• Inverse rotation:

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

Quaternions

• That is like rotation with matrix $[A]$:

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

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

$$\bar{v}' = [A(q)] \bar{v}$$

• Matrix $[A]$ as a function of a quaternion:

$$[A(q)] = \begin{bmatrix}
q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_3 q_0) & 2(q_1 q_3 + q_2 q_0) \\
2(q_1 q_2 + q_3 q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(-q_1 q_3 + q_2 q_0) \\
2(q_1 q_3 - q_2 q_0) & 2(q_1 q_3 + q_2 q_0) & q_0^2 - q_1^2 - q_2^2 + q_3^2
\end{bmatrix}$$

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

$$[A(q)] = [F(q)] [F(q)]^T \bar{v}$$
Quaternions

• Viceversa:

(note: no singularity!)

Algorithm 1: Calcolo quaternione \( q \) da matrice \([A]\)

\[
\begin{align*}
q_0 &= \cos \left( \frac{\phi}{2} \right) \\
q_1 &= u_x \sin \left( \frac{\phi}{2} \right) \\
q_2 &= u_y \sin \left( \frac{\phi}{2} \right) \\
q_3 &= u_z \sin \left( \frac{\phi}{2} \right)
\end{align*}
\]

Quaternions

• Quaternion function of angle and axis
# Quaternions

- Useful conversions

<table>
<thead>
<tr>
<th>Algebra dei quaternioni</th>
<th>Algebra matriciale</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{p}^r = \tilde{q} \tilde{p} \tilde{q}^*$, $\tilde{p} = (0, \vec{p})$</td>
<td>$\tilde{e}^r = [\tilde{A}] \tilde{e}$</td>
</tr>
<tr>
<td>$\tilde{p}^r = \tilde{q} \tilde{p} \tilde{q}^* + \tilde{q} \tilde{p} \tilde{q}^*$</td>
<td>$\tilde{e}^r = [\tilde{A}(\tilde{q})] \tilde{e} + [\tilde{A}(\tilde{q})] \tilde{\omega}$</td>
</tr>
<tr>
<td>$\tilde{p}^r = \tilde{q} \tilde{p} \tilde{q}^* + \tilde{q} \tilde{p} \tilde{q}^* + \tilde{q} \tilde{p} \tilde{q}^* + 2 \tilde{q} \tilde{p} \tilde{q}^* + 2 \tilde{q} \tilde{p} \tilde{q}^*$</td>
<td>$\tilde{A}(\tilde{q}) = [\tilde{A}(\tilde{q})] \tilde{\omega} + [\tilde{A}(\tilde{q})] \tilde{\omega}$</td>
</tr>
</tbody>
</table>

### 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 \( q(0) \).

\[
\begin{pmatrix}
mI & 0 \\
0 & J_c
\end{pmatrix}
\begin{pmatrix}
\dot{q} \\
\dot{\omega}
\end{pmatrix}
+
\begin{pmatrix}
0 \\
\omega \times J_c \omega
\end{pmatrix}
=
\begin{pmatrix}
f \\
\tau
\end{pmatrix}
\]

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

Example: **Newton-Euler equations**, single body:

- These can be obtained by developing, for instance, the Lagrange equations.

- Note that the unknowns are the linear accelerations and the angular accelerations: \( \begin{pmatrix} q \\ \omega \end{pmatrix} \).

- The gyroscopic term is null if \( \omega \) is parallel to one of the three principal axes of \( J \) tensor (i.e. \( \omega \) 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:

\[
\frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{q}_i} \right) - \frac{\partial \mathcal{L}}{\partial q_i} + \frac{\partial T}{\partial q_i} = \frac{\partial V}{\partial q_i} = 0, \quad \mathcal{L} = T - V.
\]

- Hamilton formulation:

\[
\dot{p}_i = \frac{\partial \mathcal{H}}{\partial \dot{q}_i}, \quad \dot{q}_i = -\frac{\partial \mathcal{H}}{\partial p_i}.
\]

\[\mathcal{H} = T + V, \quad \mathcal{H} = \sum p_i \dot{q}_i - \mathcal{L}\]

- Other variational principles:
  - Gauss least constraint principle
  - Jourdain principle
  - D’Alembert 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) \( \rightarrow \) 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 \((6 \times n_{\text{body}} + \text{constraint multipliers}) \rightarrow \text{DAE} \)

  • 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

\[
\begin{aligned}
\left\{ \begin{array}{l}
\frac{d}{dt} \left[ \frac{\partial \mathbf{E}_L}{\partial \dot{\mathbf{x}}} \right]^T - \left[ \frac{\partial \mathbf{E}_L}{\partial \dot{\mathbf{x}}} \right]^T + \left[ \mathbf{C}_z \right]^T \lambda = \dot{\mathbf{Q}} \\
\mathbf{C}(\mathbf{x}, t) = 0
\end{array} \right.
\end{aligned}
\]

  • \(\mathbf{C}(\mathbf{x}, t)\) is a vector of (nonlinear) equations, satisfied \(=0\) if constraint is ‘closed’
  • \(\lambda\) 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 simplest methods: index reduction:

\[
\begin{bmatrix}
\frac{d}{dt} \left[ \frac{\partial \mathbf{C}}{\partial \mathbf{x}} \right]^T \\
\mathbf{C}(\mathbf{x}, t) = 0
\end{bmatrix}^T + \left[ \mathbf{C}_x \right]^T \lambda = \dot{\mathbf{Q}}
\]

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

Trick: from a DAE…

…to a simpler ODE

Solving for unknowns

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

\[
\begin{bmatrix}
[M] & [C_x]^T \\
[C_x] & [0]
\end{bmatrix}
\begin{bmatrix}
\dot{\mathbf{x}} \\
\lambda
\end{bmatrix} = \begin{bmatrix}
\dot{\mathbf{Q}} + \mathbf{Q}_m \\
\mathbf{Q}_c
\end{bmatrix}
\]

Note...remember:

\[
\frac{1}{2} [\mathbf{G}_q (\mathbf{q})]^T [\mathbf{G}_q (\mathbf{q})] = \mathbf{1}
\]

\[
\mathbf{a}_l = \mathbf{G}_q (\mathbf{q}_c) \frac{\mathbf{q}}{2}
\]

\[
\mathbf{x}_c = \{ \mathbf{p}_{(0)}, \mathbf{a}_{(1)}, \cdots, \mathbf{p}_{(s)}, \mathbf{a}_{(s)} \}
\]

\[
\begin{bmatrix}
[M] & [C_x]^T \\
[C_x] & [0]
\end{bmatrix}
\begin{bmatrix}
\dot{\mathbf{x}}_c \\
\lambda
\end{bmatrix} = \begin{bmatrix}
\dot{\mathbf{Q}} + \mathbf{Q}_m \\
\mathbf{Q}_c
\end{bmatrix}
\]

\[
\mathbf{G}_q (\mathbf{q}_c) \frac{\mathbf{q}}{2}
\]

\[
\begin{bmatrix}
[I] & \frac{1}{2} [\mathbf{G}_q (\mathbf{q}_c)]_0 \\
\vdots & \vdots
\end{bmatrix}
\]

Alessandro Tasora
Solving for unknowns

• To keep symmetry, pre-multiply everything by \([T_q]^T\):

\[
\begin{bmatrix}
[I] & [C_y] & [0] & [I] \\
\end{bmatrix}
\begin{bmatrix}
\dot{x}_a \\
\dot{x}_b \\
\end{bmatrix}
= 
\begin{bmatrix}
\mathbf{Q} + \mathbf{Q}_u \\
\end{bmatrix}
\]

• 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:

  ![Diagram of constraint violation](image)

- Different approaches to solve the “constraint drifting”:

  - Solve DAE directly with a special method (ex. DASSL integrator)
    - Numerically intensive
    - May suffer ill-conditioning, esp. 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 $q''$)

Correct constraint position-violation
(re-map in quaternions $q$)

Correct constraint speed-violation
(re-map in quaternions $q'$)

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

Ex: Euler with simple constraint stabilization

$$
\begin{bmatrix}
\dot{M} & C_q^T \\
C_q & 0
\end{bmatrix}
\begin{bmatrix}
\nu^{l+1} \\
-\lambda^{l+1}
\end{bmatrix} =
\begin{bmatrix}
\dot{M} \nu^{l+1} + h f^l \\
\frac{C_q^T}{h} - C_t
\end{bmatrix}
$$

$q^{l+1} = q^l + h \nu^{l+1}$
Ex: implicit DAE solver: Euler implicit with constraints

\[
\begin{bmatrix}
\dot{M} - h^2 \nabla q f^{i+1} - h \nabla v f^{i+1} \\
C_q \\
0
\end{bmatrix}
\begin{bmatrix}
\Delta v^{i+1} \\
- \Delta \lambda^{i+1}
\end{bmatrix}
= \begin{bmatrix}
(v^{i} - v^{i+1}) \dot{M} + h f^{i+1} + h C_q^T \lambda^{i+1} \\
- C_q^{i+1} \hbar
\end{bmatrix}
\]

\[v_{n+1}^{i+1} = v_n^{i+1} + \Delta v^{i+1} \]
\[\lambda_{n+1}^{i+1} = \lambda_n^{i+1} + \Delta \lambda^{i+1} \]
\[q_{n+1}^{i+1} = q^i + h v_{n+1}^{i+1} \]

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*)

---

**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 \(\rightarrow\) DAEs, ODEs
  - **Non-smooth dynamics** with *set-valued* contact forces \(\rightarrow\) DVIs, MDIs, etc.
Why non-smooth dynamics?

- Most differential problems can be posed as equalities like:
  \[ \frac{dx}{dt} = f(x,t) \] → ODE, DAE, ok
- But some problems require inequalities or inclusions like
  \[ \frac{dx}{dt} \in f(x,t) \] → Differential Inclusion! (DI)

- Example: a flywheel with brake torque and applied torque (looks simple?!)
  \[ J \frac{d\omega}{dt} = M_f(\omega) + M_e(t) \] where
  \[ M_f = -M_{f\text{max}} \quad \text{if} \quad \omega > 0 \]
  \[ M_f = M_{f\text{max}} \quad \text{if} \quad \omega < 0 \]
  \[ -M_{f\text{max}} < M_f < M_{f\text{max}} \quad \text{for} \quad \omega = 0 \]
  - All ODE integrator would never stop in \( \omega = 0 \)!
    It would just ripple about \( \omega = 0 \)...
  - Reducing \( \Delta t \) 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!

- This could handle also \( \omega = 0 \) case, ex. brake sticking
- But now we have a differential inclusion \( \frac{d\omega}{dt} \in f(\omega,t) \).
  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:
  \[ K^* = \{ y \in \mathbb{R}^n : \langle y, x \rangle \geq 0 \quad \forall x \in K \} \]

- The **polar cone** of $K$ is:
  \[ K^\circ = \{ y \in \mathbb{R}^n : \langle y, x \rangle \leq 0 \quad \forall x \in K \} = -K^* \]

- The **normal cone** of a set $K$ at a point $x$ is:
  \[ N_K(x) = \{ y \in \mathbb{R}^n : \langle y, x - z \rangle \geq 0, \forall z \in K \} \]

Mathematical background

- The **tangent cone** of a set $K$ at a point $x$ is:
  \[ T_K(x) = \text{cl}\{ \beta(y - x) : y \in K, \beta \in \mathbb{R}^+ \} = N_K(x)^\circ \]

- The **horizon cone (recession cone)** of a set $K$ at a point $x$ is:
  \[ K^\infty = \{ y \in \mathbb{R}^n : \forall x \in K, \forall \lambda \geq 0, x + \lambda y \in K \} \]
Mathematical background

• The **indicator function** of a subset \( A \in \mathcal{E} \) is a scalar function:

\[
I_A(x) = \begin{cases} 
\infty & \text{if } x \in A \\
0 & \text{if } x \notin A 
\end{cases}
\]

- **notes:**
  - The normal cone is the subdifferential of the indicator function of \( K \):
    \[
    \partial I_K(x) = \mathcal{N}_K(x)
    \]
  - If \( f \) is differentiable,
    \[
    \partial f(x) = \{ \nabla f(x) \}
    \]
Mathematical background

• Variational Inequality (VI):

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

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

(see Kinderleher and Stampacchia, 1980)

Alternative VI formulation:

\[ x \in \mathcal{K} : F(x) \in \mathcal{N}_\mathcal{K}(x) \]

---

Mathematical background

• Linear Complementarity Problem (LCP):

\[Ax - b \geq 0, \quad x \geq 0, \quad \langle Ax - b, x \rangle = 0\]

• Alternative formulations:
  
  • with compact notation:
    \[Ax - b \geq 0 \quad \perp \quad x \geq 0\]
  
  • as a VI with affine function, on positive orthant
    \[x \in \mathbb{R}_+^n : \langle Ax - b, y - x \rangle \geq 0 \quad \forall y \in \mathbb{R}_+^n\]
Mathematical background

- Cone Complementarity Problem (CCP):
  \[ Ax - b \in -\mathcal{Y}^o, \quad x \in \mathcal{Y}, \quad \langle Ax - b, x \rangle = 0 \]
  with cone \( \mathcal{Y} \)

- Alternative formulations:
  - with compact notation:
    \[ Ax - b \in -\mathcal{Y}^o \quad \perp \quad x \in \mathcal{Y} \]
  - as a VI with affine function, on set \( \mathcal{Y} \)
    \[ x \in \mathcal{Y} : \quad \langle Ax - b, y - x \rangle \geq 0 \quad \forall y \in \mathcal{Y} \]

Differential problems

- Ordinary Differential Equations (ODE):
  \[ \frac{dx}{dt} = f(x, t) \]

- Differential Algebraic Equations (DAE):
  \[ \frac{dx}{dt} = f(x, t) \]
  \[ g(x, t) = 0 \]

  - for \( f(x, t) \) Lipschitz-continuous in \( x \) and continuous in \( t \)
  - with prescribed initial boundary conditions
Differential problems

- **Differential Inclusions (DI):**

  \[ \frac{dx}{dt} \in \mathcal{F}(x, t) \]

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

  - Example: Filippov Differential Inclusions for discontinuous \( f(x, t) \)

  \[ \frac{dx}{dt} \in \mathcal{F}f(x, t) \quad \mathcal{F}f(x, t) = \bigcap_{\eta > 0} \bigcap_{N>0} \bigcap_{\lambda_0(N) = 0} \partial f(x + \eta B_1 \setminus N, t) \]

- **Measure Differential Inclusions (MDI):**

  \[ \frac{dv}{dt} \in \mathcal{K}(q, t) \]

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

  - Lebesgue decomposition of measure \( dv = \nu_v + h\lambda_0 \)

    - Singular part \( \nu_v \)
    - Lebesgue measure \( \lambda_0 \) for continuous \( h(t) \in L^1(a, b) \)  
      \[ \text{speed ‘jumps’} \]
      \[ \text{classical ‘acceleration’} \]

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

• Measure Differential Inclusions (MDI):

\[
\frac{dv}{dt} \in K(q,t) \\
\dot{v} = \nu_s + h\lambda_0
\]

• Strong definition of solution:
  - \( h(t) \in K(t) \) almost all \( t \)
  - Radon-Nikodym

\[
\frac{d\nu_s}{|\nu_s|}(t) \in K(t)_\infty
\]

• Weak definition of solution: [Stewart]

\[
\frac{\int \phi(t)dv(dt)}{\int \phi(t)dt} \in \bigcap_{\tau:\phi(\tau)\neq\emptyset} K(\tau)
\]

• Side note: MDI can solve the Painlevé paradox (1895)
Differential Variational Inequality

- Differential Variational Inequality (DVI)

\[
\frac{dx}{dt} = f(x, u, t) \\
\Xi(x(0), x(T)) = 0
\]

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

- Note that DVI with vector-signed measures are MDI: hard contacts lead to
  - velocities as BV functions, with Lebesgue decomposition \( dv = \nu_x + h \lambda_0 \)
  - accelerations in distributional generalized sense

- Note that DAE are a special case of DVI where \( \mathcal{K} = \mathbb{R}^n \) and \( F = 0 \)

The DVI model

- Formulating Multibody Non-Smooth Contact Dynamics as a DVI:
  - Set \( G_B \) of bilateral joints
  - Set \( G_A \) of point contacts
  - External forces

\[
\dot{\theta} = \Gamma(q)v \\
M(q)\ddot{q} = \sum_{i \in G_B} \tilde{\psi}_i \xi_i + \sum_{i \in G_A} \tilde{\psi}_i \xi_i + f(q, v)
\]

\( \psi_i, \xi_i, \xi_i \in GB \) \( i \in GB \)
\( \psi_i \in \text{SOL}(F_i, F_i(q(t), v(t), \cdot), \cdot) \) \( i \in GA \)
The DVI time-stepper: a VI

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

\[
M(v^{t+1}) - v' = \sum_{i \in A(q^{(t)})} \left( \gamma_n^i D_n^i + \gamma_i^i D_{\alpha}^i + \gamma_\nu^i D_\nu^i \right) + \\
+ \sum_{i \in G_F} \left( \gamma_n^i \nabla \psi^i \right) + h f_s(t^{(t)}, q^{(t)}, v^{(t)})
\]

\[
0 \leq \frac{1}{h} \Phi^i(q^{(t)}) + \nabla \psi^i T v^{(t+1)} + \frac{\partial \psi^i}{\partial t} \quad i \in G_B
\]

\[
(\gamma_n^i, \gamma_i^i) = \text{argmin} \quad \rho \gamma_n^i \gamma_\nu^i \geq \sqrt{(\gamma_n^i)^2 + (\gamma_\nu^i)^2} \quad i \in A(q^{(t)}, \epsilon)
\]

\[
q^{(t+1)} = q^{(t)} + h v^{(t+1)}
\]

VI as a cone complementarity

- Aiming at a more compact formulation:

\[
b_A = \left\{ \frac{1}{h} \Phi^i, 0, 0, \frac{1}{h} \Phi^i, 0, 0, \ldots, \frac{1}{h} \Phi^i, 0, 0 \right\}
\]

\[
\gamma_A = \left\{ \gamma_n^i, \gamma_i^i, \gamma_\nu^i, \gamma_n^i, \gamma_i^i, \gamma_\nu^i, \ldots, \gamma_n^i, \gamma_i^i, \gamma_\nu^i \right\}
\]

\[
b_B = \left\{ \frac{1}{h} \Phi^i, \frac{\partial \psi^i}{\partial t}, \frac{1}{h} \Phi^i, \frac{\partial \psi^i}{\partial t}, \ldots, \frac{1}{h} \Phi^i, \frac{\partial \psi^i}{\partial t} \right\}
\]

\[
\gamma_B = \left\{ \gamma_n^i, \gamma_\nu^i, \ldots, \gamma_n^i, \gamma_\nu^i \right\}
\]

\[
D_A = \left[ D^i \left| D^i \right| \ldots \left| D^i \right| \right], \quad i \in A(q^t, \epsilon) \quad D^i = \left[ D_n^i \left| D_\alpha^i \right| D_\nu^i \right]
\]

\[
D_B = \left[ \nabla \psi^i \left| \nabla \psi^i \right| \ldots \left| \nabla \psi^i \right| \right], \quad i \in G_B
\]

\[
b_C \in \mathbb{R}^{n_C} = \{ b_A, b_B \}
\]

\[
\gamma_C \in \mathbb{R}^{n_C} = \{ \gamma_A, \gamma_B \}
\]

\[
D_C = [D_A | D_B]
\]
Cone complementarity

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

\[
\begin{align*}
\tilde{k}^{(l)} &= Mv^{(l)} + h f_i(t^{(l)}, q^{(l)}, v^{(l)}) \\
N &= D^T_\varepsilon M^{-1} D_\varepsilon \\
r &= D^T_\varepsilon M^{-1} \tilde{k} + b_\varepsilon
\end{align*}
\]

\[
Y = \left( \bigoplus_{i \in A(q^{\prime}, \varepsilon)} FC_i \right) \oplus \left( \bigoplus_{i \in B \varepsilon} BC_i \right) \\
Y^\circ = \left( \bigoplus_{i \in A(q^{\prime}, \varepsilon)} FC_i^{\circ} \right) \oplus \left( \bigoplus_{i \in B \varepsilon} BC_i^{\circ} \right)
\]

Then the full problem becomes:

\[
\text{CCP} \quad (N\gamma_\varepsilon + r) \in -Y^\circ \quad \perp \gamma_\varepsilon \in Y
\]
Example of DVI with large CCPs

- 10 millions of bodies
- 60 million of contacts
Solve CCP using projected fixed-point iteration

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

\[ (N\gamma^r + r) \in -\gamma^o \quad \perp \quad \gamma^r \in \gamma \]

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

\[
\gamma^{r+1} = \lambda \Pi_T \left( \gamma^r - \omega B^r \left( N\gamma^r + r + K^r \left( \gamma^{r+1} - \gamma^r \right) \right) \right) + (1 - \lambda) \gamma^r
\]

- With matrices:

\[
B^r = \begin{bmatrix}
\eta_1 I_{n_1} & 0 & \cdots & 0 \\
0 & \eta_2 I_{n_2} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \eta_m I_{n_m}
\end{bmatrix}
\]

\[
K^r =:\begin{bmatrix}
0 & K_{12} & \cdots & K_{1n_m} \\
0 & 0 & \cdots & K_{2n_m} \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

- and a non-extensive orthogonal projection operator onto feasible set \( \Pi_T : \mathbb{R}^{n_E} \rightarrow \mathbb{R}^{n_E} \)

**ASSUMPTIONS**

- 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:
  \[
  \Pi_T = \left\{ \Pi_T(\gamma_1), \ldots, \Pi_{T,i}(\gamma_{n,i}), \Pi_{T,j}(\gamma_{n,j}) \right\}^T
  \]
- For each bilateral constraint, simply do nothing.
- The complete operator:
  \[
  \gamma_i = \begin{cases}
  \gamma_i, & \text{if } \gamma_1 < \mu \gamma_n \\
  \gamma_i < -\frac{1}{\mu}, & \text{if } \gamma_2 \\
  \gamma_i > \mu \gamma_n \wedge \gamma_3 > -\frac{1}{\mu}, & \text{if } \gamma_3 = \mu \gamma_2 + \gamma_n
  \end{cases}
  \]

Development of an efficient algorithm for fixed point iteration:

\[
\gamma^{i+1} = \Pi_T \left( \gamma^i - \omega \left( N \gamma^i + r + K^T \left( \gamma^{i+1} - \gamma^i \right) \right) \right) + (1 - \lambda) \gamma^i
\]

With \( N = D^T M^{-1} D \)

- At each \( i \)-th iteration:
  \[
  \begin{align*}
  \delta^{i+1} &= \omega T \left( \sum_{j=1}^{n_A} D_j g_j^A + \sum_{j=1}^{n_B} D_j g_j^B + \delta^i \right) + b^i \\
  \gamma^{i+1} &= \Pi_T \left( \delta^{i+1} + (1 - \lambda) \gamma^i \right)
  \end{align*}
  \]

If \( i \)-th is a contact constraint:

\[
\begin{align*}
D^i \gamma^i &= D^i \gamma^i + \frac{3}{\text{Trace}(g_a^i)} \\
\gamma_a^i &= \gamma_a^i \\
\eta_a^i &= \frac{1}{g_a^i}
\end{align*}
\]

If \( i \)-th is a scalar bilateral constraint

\[
\begin{align*}
D^i \gamma^i &= \nabla \psi \gamma^i \\
\gamma_b^i &= \gamma_b^i \\
\eta_b^i &= \frac{1}{g_b^i}
\end{align*}
\]
Solve CCP using projected fixed-point iteration

- Even better, in **incremental** form:

\[
\begin{align*}
\delta_{i,r+1} &= \gamma_{i,r} - \omega \eta_i \left( D_i \gamma_i + \sum_{j \neq i} D_j \gamma_{j,r} + k_i \right) + b_i^r \\
\gamma_{i,r+1} &= \lambda \Pi_i \left( \delta_{i,r+1} \right) + (1 - \lambda) \gamma_{i,r} \\
\end{align*}
\]

We know that: \( \mathbf{v} = M^{-1} \mathbf{D} \mathbf{y} + M^{-1} \mathbf{k} \) ...so we rewrite:

\[
\begin{align*}
\delta_{i,r+1} &= \left( \gamma_{i,r} - \omega \eta_i \left( D_i \mathbf{v} + b_i^r \right) \right) \\
\gamma_{i,r+1} &= \lambda \Pi_i \left( \delta_{i,r+1} \right) + (1 - \lambda) \gamma_{i,r} \\
\Delta \gamma_{i,r+1} &= \gamma_{i,r} - \gamma_{i,r} \\
\mathbf{v} &= \mathbf{v} + M^{-1} D_i \Delta \gamma_{i,r+1} \\
\end{align*}
\]

This 'incremental' form has \( O(n) \) complexity!!!
Solve CCP using projected fixed-point iteration

Pseudocode:

```plaintext
(1) // Pre-compute some data for friction constraints
(2) for i := 1 to n_A
(3) \( \dot{\alpha}_i^f = M^{-1} P_i^f \)
(4) \( \alpha_i^{f,0} = D_i^{T} \dot{\alpha}_i^f \)
(5) \( \alpha_i^{f,0} = \frac{\alpha_i^{f,0}}{\| \alpha_i^{f,0} \|} \)
(6) // Pre-compute some data for bilateral constraints
(7) for i := 1 to n_B
(8) \( \dot{\alpha}_i^b = M^{-1} \dot{\phi}_i^b \)
(9) \( \alpha_i^{b,0} = \frac{\alpha_i^{b,0}}{\| \alpha_i^{b,0} \|} \)
(10) \( \alpha_i^{b,0} = \alpha_i^{b,0} \)
(11) // Initialize impulses
(12) if warm start with initial guess \( \gamma_i^w \)
(13) \( \gamma_i^w = \gamma_i^w \)
(14) else:
(15) \( \gamma_i^w = 0 \)
(16) \( \gamma_i^w = 0 \)
(17) // Initialize speeds
(18) \( v = \sum_{i=1}^{n_A} \alpha_i^{f,0} + \sum_{i=1}^{n_B} \dot{\alpha}_i^b \)
(19) \( \dot{v} = \sum_{i=1}^{n_A} \dot{\alpha}_i^f + \sum_{i=1}^{n_B} \dot{\phi}_i^b + M^{-1} k \)
(20) // Main iteration loop
(21) for \( r := 0 \) to \( r_{\text{max}} \)
(22) // Loop on frictional constraints
(23) for i := 1 to n_A
(24) \( \Delta \gamma_i^{f+1} = \frac{\gamma_i^{w+1} \pm \gamma_i^{r+1} \left( D_i^{T} \dot{w} + b_i^f \right)}{ \gamma_i^{w+1} - \gamma_i^{r+1} } \)
(25) \( \Delta \gamma_i^{f+1} = \Delta \gamma_i^{r+1} \gamma_i^{r+1} \)
(26) \( \Delta \gamma_i^{r+1} = \Delta \gamma_i^{r+1} \gamma_i^{r+1} \)
(27) \( \gamma_i^{w+1} = \gamma_i^{w+1} - \Delta \gamma_i^{r+1} \gamma_i^{r+1} \)
(28) \( \gamma_i^{r+1} = \gamma_i^{r+1} - \Delta \gamma_i^{r+1} \gamma_i^{r+1} \)
(29) \( \Delta \gamma_i^{b+1} = \frac{\gamma_i^{w+1} \pm \gamma_i^{r+1} \left( \dot{\phi}_i^b + b_i^b \right)}{ \gamma_i^{w+1} - \gamma_i^{r+1} } \)
(30) \( \Delta \gamma_i^{b+1} = \Delta \gamma_i^{b+1} \gamma_i^{r+1} \)
(31) \( \Delta \gamma_i^{b+1} = \Delta \gamma_i^{b+1} \gamma_i^{r+1} \)
(32) \( \gamma_i^{w+1} = \gamma_i^{w+1} - \Delta \gamma_i^{b+1} \gamma_i^{r+1} \)
(33) \( \gamma_i^{r+1} = \gamma_i^{r+1} - \Delta \gamma_i^{b+1} \gamma_i^{r+1} \)
(34) \( \gamma_i^{r+1} = \gamma_i^{r+1} \)
(35) \( \gamma_i^{r+1} = \gamma_i^{r+1} \)
(36) return \( \gamma_i^{w+1}, \gamma_i^{r+1} \)
```

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

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-Seidel / SOR, \(\leftarrow\) 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:

\[
\begin{align*}
\min & \quad f(x) = \frac{1}{2} x^T A x + x^T b \\
\text{s.t.} & \quad x \in \mathcal{X}
\end{align*}
\]

• 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 isotropic cone scaling) \( P = \text{diag}(\mathcal{A}) \)
  • 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(\mathbf{A}, \mathbf{b}, \mathbf{x}_0, \mathbf{x}_f)
\begin{align*}
\mathbf{x}_0 &= \Pi_\mathcal{X}(\mathbf{x}_0), \quad \mathbf{x}_{2} = \mathbf{x}_0, \\
\mathbf{d}_0 &\in [\mathbb{B}_\text{max}, \mathbb{B}_\text{min}], \\
\mathbf{g}_0 &= -\mathbf{A}\mathbf{x}_0 + \mathbf{b}, \quad f(\mathbf{x}_0) = \frac{1}{2}\|\mathbf{g}_0\|_2^2 + \mathbf{w}_0, \quad \mathbf{w}_0 = 10^{29} \\
\text{for } j = 0 \text{ to } N_{\text{max}} \\
\mathbf{y}_j &= \mathbf{P}^{-1}\mathbf{d}_j, \\
\mathbf{d}_j &= \Pi_\mathcal{X}(\mathbf{x}_j - \alpha_{\text{FB}}\mathbf{d}_j) - \mathbf{x}_j \\
\text{if } (\mathbf{d}_j, \mathbf{g}_j) \geq 0 \\
\alpha_{\text{FB}} &= \Pi_\mathcal{X}(\mathbf{x}_j - \alpha_{\text{FB}}\mathbf{d}_j) - \mathbf{x}_j \\
\Delta_j &= 1 \\
\text{while line search} \\
\mathbf{x}_{j+1} &= \mathbf{x}_j + \Delta_j \mathbf{d}_j \\
\mathbf{g}_{j+1} &= -\mathbf{A}\mathbf{x}_{j+1} + \mathbf{b} \\
f_j &\equiv \frac{1}{2}\|\mathbf{g}_{j+1}\|_2^2 + \mathbf{w}_j, \quad \mathbf{w}_j = \frac{1}{2}\|\mathbf{g}_j\|_2^2 + \mathbf{w}_{j-1} \\
\text{if } f_j < f_{j+1} \\
p_j &\equiv \frac{1}{2}\|\mathbf{g}_j\|_2^2 + \mathbf{w}_{j-1} \\
\alpha(\mathbf{d}_j, \mathbf{g}_j) &\equiv \max_{0 < \alpha < 1} \frac{f(\mathbf{x}_j + \alpha \mathbf{d}_j)}{f_j} \\
\text{define } \lambda_{\text{new}} &\equiv \min \left\{ \lambda_{\text{old}}, \alpha(\mathbf{d}_j, \mathbf{g}_j) \right\} \\
\text{and} \\
\text{repeat line search} \\
\text{else} \\
\text{terminate line search} \\
\mathbf{y}_j &= \mathbf{x}_{j+1} - \mathbf{x}_j \\
\mathbf{g}_j &= \mathbf{g}_{j+1} - \mathbf{y}_j \\
\text{if } j \text{ is odd} \\
\alpha_{\text{FB}} &= \frac{\langle \mathbf{g}_j, \mathbf{y}_j \rangle}{\|\mathbf{y}_j\|_2^2} \\
\text{else} \\
\alpha_{\text{FB}} &= \min \left\{ \alpha_{\text{FB}}, \min \left\{ \lambda_{\text{old}}, \alpha(\mathbf{d}_j, \mathbf{g}_j) \right\} \right\} \\
w_j &= \|\mathbf{g}_j\|_2^2 + \min \{ \mathbf{w}_{j-1}, \|\mathbf{g}_j\|_2^2 \} \\
\mathbf{x}_{j+1} &= \mathbf{x}_j + \min \{ w_j, \mathbf{w}_{j-1} \} \\
\text{return } \mathbf{x}_{j+1}
\end{align*}
```

Results

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

![Graph showing comparison of different solvers](image.png)
Results

• Comparison with other solvers for complementarity problems
  • (only unilateral contacts, no friction)

Results

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

- Effect of preconditioning:

![Graph showing the effect of preconditioning](image)

Example

![Example image](image)
Example

Walking robot with contacts and bilateral constraint

DVI advanced contact laws

Rigid contact:

Compliant contact:

Nonlinear, with cohesion:

Rigid, with plastic cohesion
DVI advanced contact laws

- In general, DVI are useful for various reasons that are difficult to handle in DAE:
  - very stiff or rigid contacts $\rightarrow$ set valued force laws $\rightarrow$ VI
  - plasticity in contacts $\rightarrow$ yield surfaces $\rightarrow$ VI
  - friction $\rightarrow$ set valued force laws $\rightarrow$ VI

DVI Elasto-Plastic contact

- Contact forces
  \[
  \mathbf{\gamma}_A^i = \begin{pmatrix} \gamma_n^i, \gamma_u^i, \gamma_w^i \end{pmatrix}^T
  \]

- Inclusion in yield surface:
  \[
  \mathbf{\gamma}_A^i \in \mathbf{\Gamma}_f^i
  \]

- Prandtl-Reuss-like assumption on displacements $\mathbf{y}$
  \[
  \mathbf{y}^i = \mathbf{y}_E^i + \mathbf{y}_P^i
  \]

- Associated flow assumption:
  - The increment to the plastic flow is orthogonal to the yield surface
  \[
  \mathbf{y}_P^i \in -\mathbf{N}_{\mathbf{T}_i}(\mathbf{\gamma}_A^i)
  \]
DVI Elasto-Plastic contact

-
DVI Elasto-Plastic contact

- Elasto-plastic model:

\[
\begin{align*}
\dot{\gamma}_A^i &= -K^i \left(y^i - y_P^i\right) & K^i &\in \mathbb{R}^{3\times3} \\
\dot{y}_P^i &\in -N_{\gamma_i^k}(\dot{\gamma}_A^i) & \dot{\gamma}_A^i &\in \dot{\gamma}_i
\end{align*}
\]

\[
\dot{\gamma}_A^i = -K^i \left(y^i - y_P^i\right)
\]

- With time discretization:

\[
\begin{align*}
h &= t^{i+1} - t^i & h\dot{\gamma} &\equiv \gamma & \Upsilon &\equiv h\dot{\gamma}
\end{align*}
\]

\[
\frac{\gamma_A^{i+1} - \gamma_A^i}{h} = -K^i \left(D_i^T v^{i+1} - y_P^i\right)
\]

\[
\dot{y}_P^i = D_i^T v^{i+1} + (h^2 K^i)^{-1} \gamma_A^{i+1} - (h^2 K^i)^{-1} \gamma_A^i \in -N_{\gamma_i^k}(\gamma_i^k)
\]

---

DVI Elasto-Plastic contact

- Define:

\[
\begin{align*}
\dot{y}_P^i &= D_i^T v^{i+1} + (h^2 K^i)^{-1} \gamma_A^{i+1} - \frac{1}{h} \left(y^{i+1} - y_P^i\right) \in -N_{\gamma_i^k}(\gamma_i^k)
\end{align*}
\]

\[
\begin{align*}
E^i &= - (h^2 K^i)^{-1} & e^i &= - \frac{1}{h} \left(y^{i+1} - y_P^i\right)
\end{align*}
\]

\[
\begin{align*}
\dot{y}_P^i &= D_i^T v^{i+1} - E^i \gamma_A^{i+1} - e^i \in -N_{\gamma_i^k}(\gamma_i^k)
\end{align*}
\]

\[
\begin{align*}
M v^{i+1} &= M v^i + \sum_{i\in\mathcal{A}} D_i^T \gamma_A^{i+1} + h f(q,v,t) & \gamma_c &= \left\{\gamma_A, \gamma_A^T, \ldots\right\}^T \\
D_c &= \left[D_1^T | D_2^T | \ldots\right] & e_c &= \left[e_1^T, e_2^T, \ldots\right]^T
\end{align*}
\]

\[
\begin{align*}
\dot{y}_P &= [D_c^T M D_c - E_c] \gamma_c^{i+1} + D_c^T (v^i + h M^{-1} f(q,v,t)) - e_c \in -N_{\gamma_c^k}(\gamma_c)
\end{align*}
\]
DVI Elasto-Plastic contact

• Posing:

\[
N = [D\varepsilon^T M D\varepsilon - E]\varepsilon \\
\Rightarrow r = +D\varepsilon^T (e^l + hM^{-1} f(q, e^l, t)) - c
\]

• One finally gets the VI:

\[
N\gamma_{\varepsilon}^{l+1} + r \in -N\Gamma(\gamma_{\varepsilon}) \Rightarrow \gamma_{\varepsilon}^{l+1} \in \Gamma
\]

• That can be written also as the ‘classical’ VI:

\[
\gamma_{\varepsilon}^{l+1} \in \Gamma : \langle N\gamma_{\varepsilon}^{l+1} + r, z - \gamma_{\varepsilon}^{l+1} \rangle \geq 0 \forall z \in \Gamma
\]

DVI Elasto-Plastic contact

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

\[
\gamma_{\varepsilon}^{l+1} \in \Gamma : \langle N\gamma_{\varepsilon}^{l+1} + r, z - \gamma_{\varepsilon}^{l+1} \rangle \geq 0 \forall z \in \Gamma
\]
DVI Visco-Elasto-Plastic contact

- By introducing also viscous damping, one gets the model

\[
\gamma^i_A = -K^i (y^i - y^i_P) - R^i (\dot{y}^i - \dot{y}^i_P) \\
\dot{y}^i_P \in -N_{\gamma^i_A} (\gamma^i_A) ; \quad \gamma^i_A \in \hat{\gamma}^i
\]

- Again one obtains a VI, this time with:

\[
\begin{align*}
E^i &= -\left( h^2 K^i + h R^i \right)^{-1} \\
c^i &= -\left( h^2 K^i + h R^i \right)^{-1} \left( \gamma^i_A + h R^i (\dot{y}^i - \dot{y}^i_P) \right) \\
N &= [D\xi^T M D\xi - E\xi] \\
r &= +D\xi^T (v^i + hM^{-1} f(q, v, t)) - c
\end{align*}
\]

\[
\gamma^{i+1}_\xi \in \Upsilon : \quad \langle N \gamma^{i+1}_\xi + r, z - \gamma^{i+1}_\xi \rangle \geq 0 \quad \forall z \in \Upsilon
\]

DVI Visco-Elasto-Plastic contact

- With Raleygh damping \( \rightarrow \) simplification

\[
R^i = \alpha^i_K K^i
\]

- Obtaining:

\[
\begin{align*}
E^i &= -\frac{1}{h(h + \alpha^i_K)} K^{i-1} \\
c^i &= -\frac{1}{h + \alpha^i_K} (\dot{y}^i - \dot{y}^i_P)
\end{align*}
\]

- **NOTE**
  the \( E \) term works as a Tykhonov regularization of the Schur complement

\[
N = [D\xi^T M D\xi - E\xi]
\]
Examples

- Granular flows (shear test)

Examples

Cohesion in contacts, with DVI
Examples
Cohesion in contacts, with DVI

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)
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

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**
    - Analytic 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:

<table>
<thead>
<tr>
<th></th>
<th>Sphere</th>
<th>Cylinder</th>
<th>Cube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>Sphere-Sphere</td>
<td>Sphere-Cylinder</td>
<td>Sphere-Cube</td>
</tr>
<tr>
<td>Cylinder</td>
<td>Cylinder-Sphere</td>
<td>Cylinder-Cylinder</td>
<td>Cylinder-Cube</td>
</tr>
<tr>
<td>Cube</td>
<td>Cube-Sphere</td>
<td>Cube-Cylinder</td>
<td>Cube-Cube</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Narrow phases

GJK - Gilbert Jordan Keerthi 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!

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*
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 benchmark](ProjectChrono_benchmark)
Particle factory

Example

Contacts between deformable parts (finite elements)
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
  - Powerful 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...

- **ALTAIR MotionSolve**
  - Similar to Adams
  - Integrated with other ALTAIR tools
  - Tools for automotive scenarios
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
  - Suspension templates, etc.
  - Interfaced with CATIA

---

Multibody software

Notable commercial software (with GUI):

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

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’

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
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

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:
  - 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...

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 participate in collision detection.
  • Set-valued Coulomb friction, plus rolling and spinning friction
  • Parts can re-bounce, 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

Features

• **Other features**

  • Different integrators: MDI stepper, 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.
Architecture

- Workflow:
Architecture

- Modules:

C++ class hierarchy -examples-

- Rigid bodies
C++ class hierarchy -examples-

• Joints

Some joint types in our Chrono::Engine software
C++ transient database

- Complex object hierarchy: smart shared pointers are used

Example

The GRANIT parallel-kinematics robot (Tasora, Righettini, Chatterton, 2007)
C++ API example

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

```cpp
// 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 (CGO) 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 CG 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 CG of rod
```

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

```cpp
// 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->Set_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 (..2..)
C++ API example

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

```cpp
// 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) );
}
```

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:

![Image of Chrono::SolidWorks add-in features]

COSIMULATION module

- The COSIMULATION module:

![Image of COSIMULATION module features]
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**

is the Python wrapper of Chrono:

Example:

```python
my_quat = chrono.ChQuaternionD(1,2,3,4)
my_qconjugate = ~my_quat
print ('quat. conjugate =', my_qconjugate)
print ('quat. dot product=', my_qconjugate % my_quat)
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.

FEA module

* Finite element types
  * Tetrahedrons 4 nodes
  * Tetrahedrons 10 nodes
  * Hexahedrons 8 nodes
  * Hexahedrons 20 nodes
  * Springs
  * Bars
  * 3D beams
  * ANCF beams
  * ANCF shells
  * Reissner 6-field shells
  * Kirchhoff-Love thin shells
  * IGA beams
  * Etc.
FEA module

• The corotational approach for beam FE

• Locally, a 3D Euler-Bernoulli beam...

\[ f_{\text{in}} = K d \]

\[ d = [d_A, \theta_A, d_B, \theta_B] \]

\[ K = \begin{pmatrix}
\end{pmatrix} \]

FEA module

• The corotational approach for beam FE

• ...mapped to global coordinates:

\[ q = [x_1, \rho_1, x_2, \rho_2, \ldots, x_n, \rho_n] \in \mathbb{R}^{(3+4)n} \]

\[ v = [v_1, \omega_1, v_2, \omega_2, \ldots, v_n, \omega_n] \in \mathbb{R}^{(3+3)n} \]

\[ f_{\text{in}} = R_o P^H H^t f_{\text{in}} \]

\[ K = R_o (P^H K HP - F_{\text{in}} G - G^t E^t P + P^t L H P) R_o^{t} \]
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
    - ...
FEA module

- 3D corotational tetrahedrons and hexahedrons

FEA module

- Kirchoff-Love thin shells, BST formulation
FEA module

• Other types of analysis

• Electrostatics

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

$$E = -\nabla \varphi$$

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

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

Example: Turbo casing with Dirichlet boundary condition
FLOW module

- SPH:

(work in progress)

VEHICLE module
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)

Other modules...

- CASCADE
- POSTPROCESSING
- MATLAB
- PARALLEL
- OPENGL
- IRRLICHT
- ...

Alessandro Tasora
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
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

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.Afferi)
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: push rod and spring forces during a simulated maneuver (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

- Valve train & timing chain
  with Adams + FEV

- Mixed 3D-1D multibody engine model
  with Chrono
  
  - dampers
  - Etc.

collaboration with
F. Pulvirenti, C. Autore et al.,
Ferrari Auto
Example - Engines

• Simulation of high performance engines

• 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
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!)

---

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
Vehicle mobility analysis – with SBEL and TARDEC

Example: SCM fast model for plastic soil, with adaptive mesh refinement
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.
Processing of waste material

- Conveyor belts, hoppers, ...

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

Processing of waste material

- Separating materials in waste processing plants:
Space

- Simulation of aggregation of small bodies

![Image of simulations](image)

*ProjectChrono simulation by F. Ferrari, Politecnico di Milano · JPL*

---

**Space**

*The Mars rover on a granular soil, simulated with ProjectChrono*

![Image of Mars rover](image)

*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

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

- The dome of Brunelleschi

Seismic engineering

Example: vault collapse
Vehicle dynamics

Modelica-based real-time vehicle simulator
(in collaboration with Altair)

10.
FUTURE CHALLENGES
“I think there’s a world market for about 5 computers.”
J. Watson, Chairman of the Board, IBM, 1948

GPU stream supercomputing

- GPU, Graphical Processor Units = “stream processors” already used in hi-end gfx boards for pixel shading in real-time OpenGL 3D views.
- 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 !!!
"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 buldozer 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

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.
→ MPI is used to handle the node-level parallelism
• ...each computer fitted with one or multiple GPU boards
→ 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.

---

**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
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
• 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
Reference papers

- Anitescu, M. & Tasora, A.
  An iterative approach for cone complementarity problems for nonsmooth dynamics
  *Computational Optimization and Applications*, 2010, 47(2), 207-236

- 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

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