Compliant Plug-In

maxime.tournier@inria.fr
INRIA-DEMAR

and M. Nesme, B. Gilles, F. Faure

Plan

- Introduction
- Theory
- Implementation Notes
- Examples

Introduction

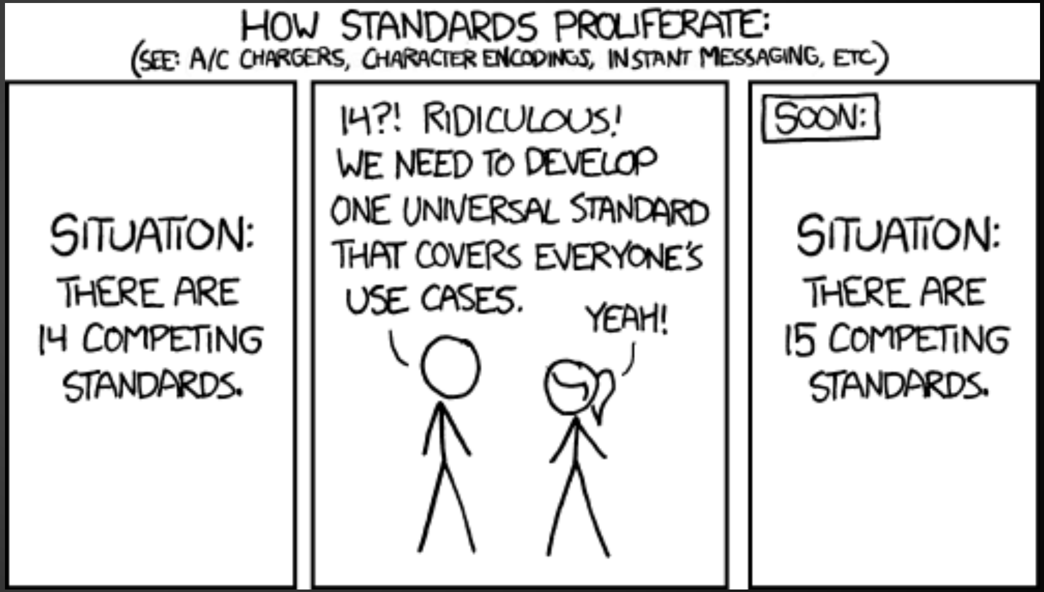
- Harmonize ForceFields/Constraints handling
  - (Re)use mappings as much as possible
- Holonomic constraint = infinite stiffness
  - = zero *compliance*
- Simplify constraint solvers, genericity

Context

- SoHuSim ANR
  - (Soft Human Simulation)
  - Human/environment mechanic interactions
- Constrained mechanical systems
  - Rigid/deformable objects
  - Constraints: bilateral/unilateral/friction/…
- Bi-phasic materials
  - Tendons, ligaments

Constraints in SOFA

- Projective
  - FixedConstraint
- LMConstraints
- constraintset
- … ?
- Compliant



(xkcd)

Philosophy

- Constraint = ForceField
  - Very (very) stiff !
- *Mappings* do most of the work
  - No PlaneConstraint, LineConstraint, ...
  - Instead: PlaneNormalMapping, LineNormalMapping, … + stiff ForceField at the end
- Factorize code whenever possible
  - in the ‘Flexible’ plug-in spirit

Very (very) stiff ?

- Stiffness matrix $K \rightarrow +\infty$
  - Numerically instable
- Compliance : $C = inv(K) \rightarrow 0$
- Formulate dynamics using compliance

Theory

Time integration

- Implicit linear velocity update:

$$Hv = c$$

- Typically:

$$H = M - hB - h^2 K$$
$$c = p + hf$$

Constraints: KKT systems

$$\begin{pmatrix} H & -J^T \\ -J & 0 \end{pmatrix} \begin{pmatrix} v \\ \lambda \end{pmatrix} = \begin{pmatrix} c \\ b \end{pmatrix}$$

- Holonomic constraint: $g(q) = 0$
- Gradients : $J^T = \nabla g(q)$
- Correction : $b = g(q)/dt$

Elasticity: mapped stiffness

- Mapping: *g(q)*
- Apply stiffness matrix K_g on mapped dofs *g (q)*
- Stiffness on *q* is:

$$K_q = J^T K_g J + \cancel{dJ^T K_g g(q)}$$

Elasticity: compliant KKT system

$$\begin{pmatrix} H & -J^T \\ -J & -C \end{pmatrix} \begin{pmatrix} v \\ \lambda \end{pmatrix} = \begin{pmatrix} c \\ b \end{pmatrix}$$

- Compliance: $C = -K_g / h^2$
    - [Servin06]
- Constraints: same system with $C = 0$

Schur Complement

- Assuming *H* is easily invertible:

$$\left(JH^{-1}J^T + C\right)\lambda = JH^{-1}c - b$$

- “Regularized” constraint system
  - Smaller than KKT system !
  - Positive (semi-)definite

In a nutshell

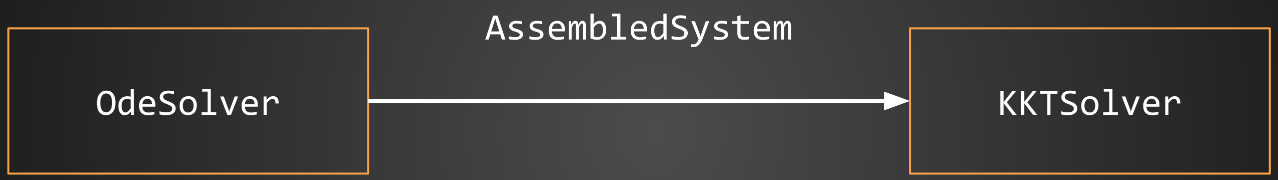
- Constraint = stiff ForceField
- Handle ForceFields as compliance
  - Easy transition towards very (very) stiff, $C \to 0$
- Numerical solves on KKT or Schur system

Implementation Notes

Compliance API

- `BaseForceField` additions:
  - `bool isCompliance()`
  - `BaseMatrix* getComplianceMatrix(...)`
- For now, complete KKT system assembly
  - Mapping J matrices <u>must</u> be provided
  - Will probably change eventually
- Not a lot to implement, don't worry

Solvers



- Integration scheme
- Constraint stabilization
- ...

- Velocity / position correction
- Lagrange multipliers
- ...

Numerical solvers

- Direct
  - LDLT, … (courtesy of Eigen library)
- Krylov
  - CG, MINRES
- Matrix-splitting
  - (block) Gauss-Seidel

(More) numerical solvers

- Projected Gauss-Seidel
  - Unilateral/friction constraints
- In CompliantDev (ask us):
  - [Silcowitz-Hansen10]
    - Speedup PGS + Fletcher-Reeves
  - [Otaduy09]
    - Friction + deformable, PGS variant
  - [Kaufman08]
    - Staggered Projections

Compliant contacts

- Two contact responses:
  - Unilateral: `CompliantContact`
  - Coulomb friction: `FrictionCompliantContact`
  - (Only `CompliantContact` will remain)
- Simply tell the ContactManager to use them
  - To get compliant contacts

Stabilization

- Add these next to a compliant `ForceField`:
  - `odesolver::Stabilization`
- Ask `AssembledSolver` for a stabilization pass:
  - `stabilization=”true”`

Use Schur complement

- Add a `Response` component next to the `KKTSolver`
  - Abstraction of *inv(H)*
  - *e.g.* `DiagonalResponse`
  - Naming sucks (we know)
- Solvers will use optimized inverse

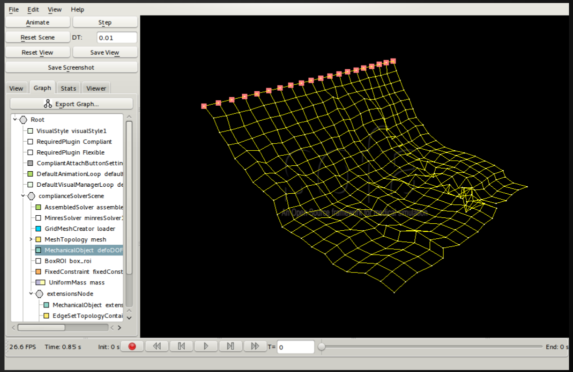
Python Library

- Simplifies scene graph creation *a lot*
  - For mere mortals
- Rigid bodies + most classic kinematic joints
- Typical examples:
  - Spherical joint + angular stiffness + damping
  - Angular limits

Examples

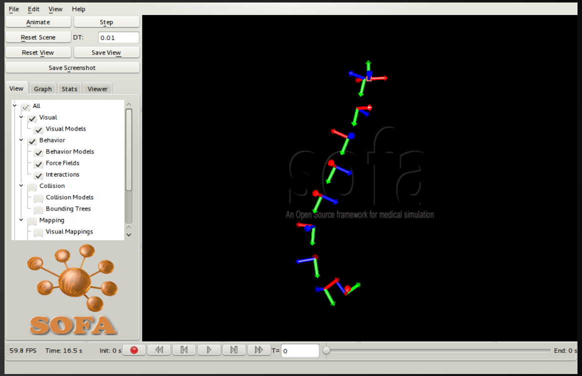
Cloths

- Start from a regular 2D mesh
- Mesh vertices are point masses
- Map relative distances along the mesh edges
- Apply `UniformCompliance` on relative distances



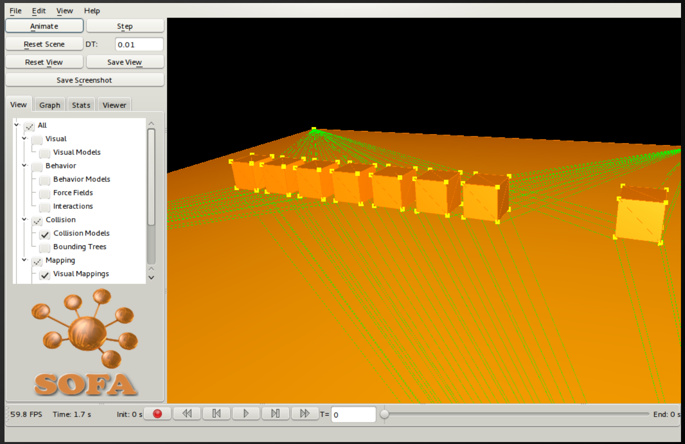
Articulated Chain

- Map parent/child rigid frames for the joint
- RigidJoint(Multi)Mapping
    - = joint dofs
- UniformCompliance on translation part
    - [+Stabilization] to avoid drift



Contacts

- Map contact point pairs, relative distances
- Apply `UniformCompliance`
  - `[+ Stabilization]`
  - `[+ Restitution]`
- Unilateral constraint:
  - `+ UnilateralConstraint`



Discussion

- Unified elasticity/constraint handling
- (re)use mappings whenever possible
- Minor modifications to existing code

Discussion

- Lots of components !
    - Options would only make it worse
    - No “default” configuration
    - Python helps *a lot*

- Assembly = slow, but:
    - Simplified solver implementation
    - Generic
    - “easy” parallelization

Final remarks

- Plays along nicely with the Flexible plug-in

- Currently under refactoring !
  - Cleaning !
  - Documentation ! yay !
  - Wait a couple of weeks before using :-)

- Python >> XML
  - Wrap gory details into a nice user API

Thank you !

maxime.tournier@inria.fr