SOFA: a modular yet efficient physical simulation architecture

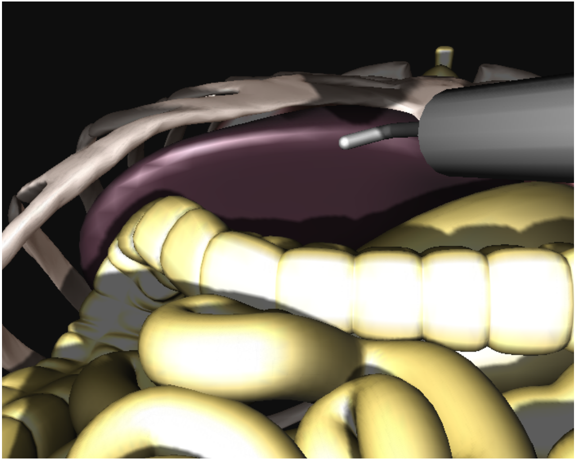
François Faure, INRIA

22 octobre 2013

Outline

- Motivation
- Simple bodies
- Layered objects using node hierarchies
  - Mappings
- Interacting objects
- Implementation
- Collision detection and response
- Parallelism
- Conclusion

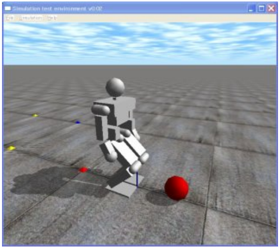
A complex physical simulation



Material, internal forces, contraints, contact detection and modeling, ODE solution, visualization, interaction, etc.

Open-Source Simulation Software

PhysX



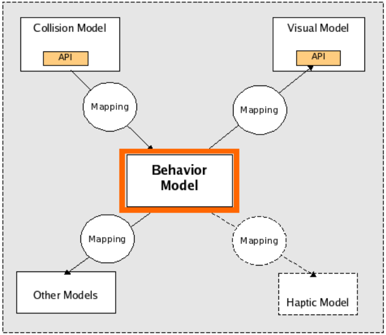
ODE

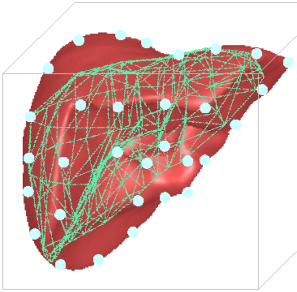
Bul

- Open-source libraries (ODE, Bullet, PhysX, etc.) provide :
  - limited number of material types
  - limited number of geometry types
  - no control on collision detection algorithms
  - no control on interaction modeling
  - few (if any) control of the numerical models and methods.
  - no control on the main loop
- We need much more !
  - models, algorithms, scheduling, visualization, etc.

A generic approach

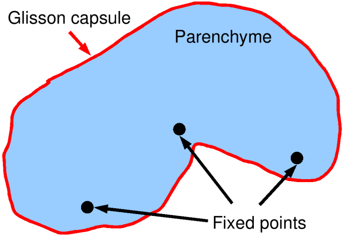
- Behavior model : all internal laws
- Others : interaction with the world
- Mappings : relations between the models (uni- or bi-directional)





Animation of a simple body

- a liver



- inside : soft material
- surface : stiffer material

A specialized program :

```
f  =  M*g
f  +=  F1(x,v)
f  +=  F2(x,v)
a  =  f/M
a  =  C(a)
v  +=  a * dt
x  +=  v * dt
display(x)
```

Outline

Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting objects

Implementation

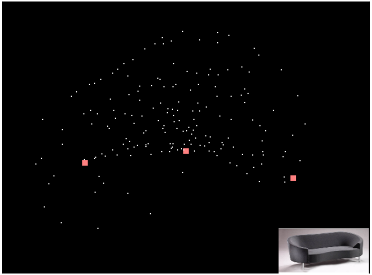
Collision detection and response

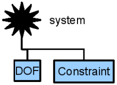
Parallelism

Conclusion

Components

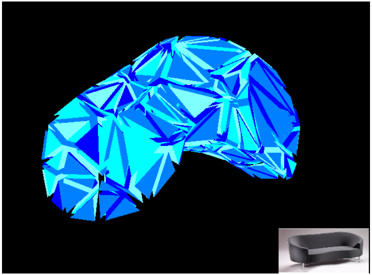
- state vectors (DOF) :
  x, v, a, f
- constraints : fixed points
  other : oscillator, collision plane, etc.

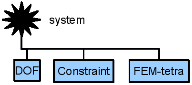




Components

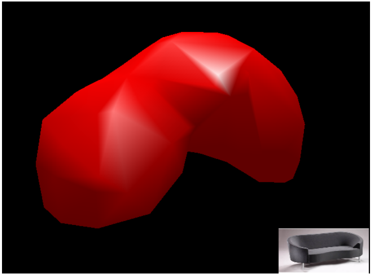
- state vectors (DOF) :
  x, v, a, f
- constraints : fixed points
- force field : tetrahedron FEM
  other : triangle FEM, springs, Lennard-Jones, SPH, etc.

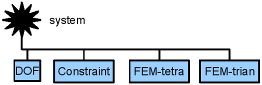




Components

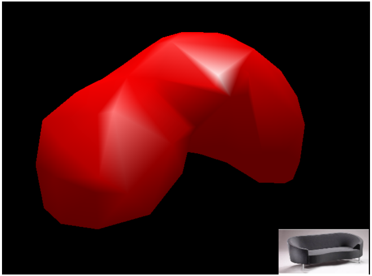
- state vectors (DOF) : x, v, a, f
- constraints : fixed points
- force field : tetrahedron FEM
- force field : triangle FEM

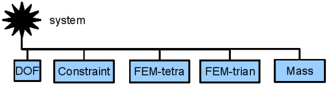




Components

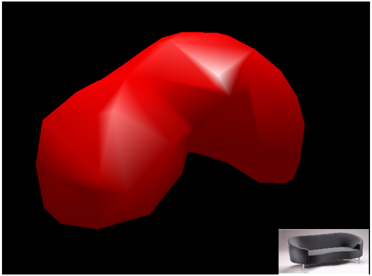
- state vectors (DOF) : x, v, a, f
- constraints : fixed points
- force field : tetrahedron FEM
- force field : triangle FEM
- mass : uniform
  other : diagonal, sparse symmetric matrix

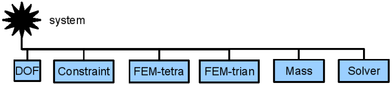




Components

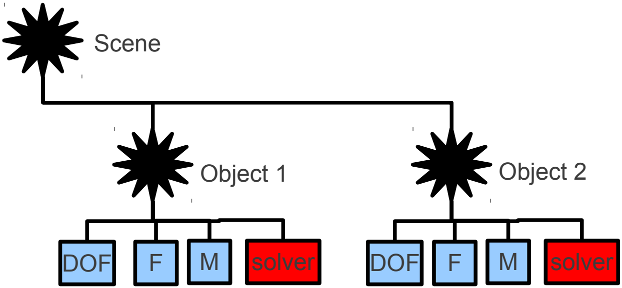
- state vectors (DOF) :
  x, v, a, f
- constraints : fixed points
- force field : tetrahedron FEM
- force field : triangle FEM
- mass : uniform
- ODE solver : explicit Euler
  other : Runge-Kutta, implicite Euler, static solution, etc.





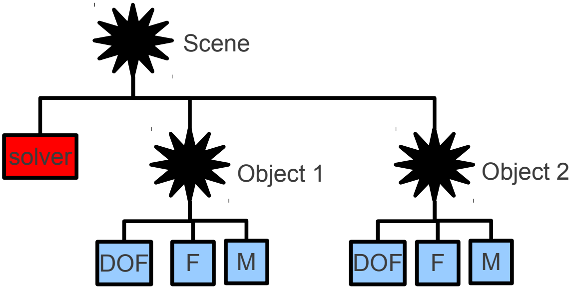
Multiple objects with their own solvers

Each object can be simulated using its own solver



Multiple objects with the same solver

A solver can drive an arbitrary number of objects of arbitrary types



Processing multiple objects using visitors

- The ODE solver sends visitors to apply operations
- The visitors traverse the scene and apply virtual methods to the components
- The methods read and write state vectors (identified by symbolic constants) in the DOF component
- Example : accumulate force
  - A `ResetForceVisitor` recursively traverses the nodes of the scene (only one node here)
    - All the `DOF` objects apply their `resetForce()` method
  - An `AccumulateForceVisitor` recursively traverses the nodes of the scene
    - All the `ForceField` objects apply their `addForce( Forces, const Positions, const Velocities )` method
  - the final value of f is weight + tetra fem force + trian fem force

Outline

Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting objects

Implementation

Collision detection and response

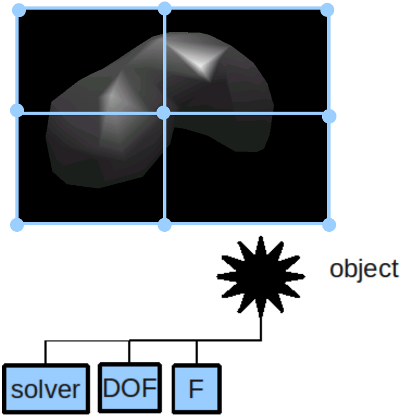
Parallelism

Conclusion

Layered object

Detailed geometry embedded in a coarse deformable grid

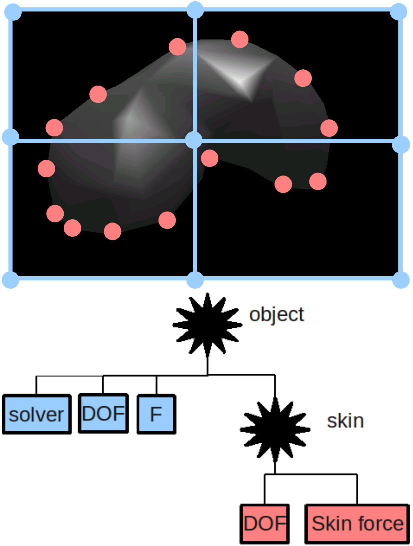
- independent DOFs (blue)



Layered object

Detailed geometry embedded in a coarse deformable grid

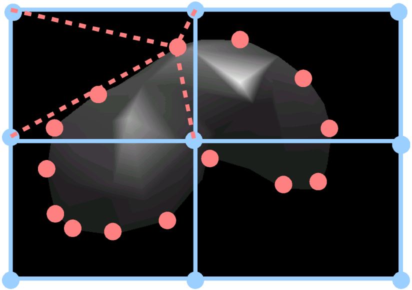
- independent DOFs (blue)
- skin vertices (salmon)

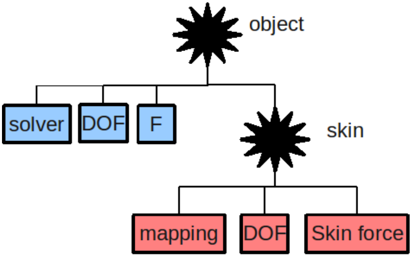


Layered object

Detailed geometry embedded in a coarse deformable grid

- independent DOFs (blue)
- skin vertices (salmon)
- mapping

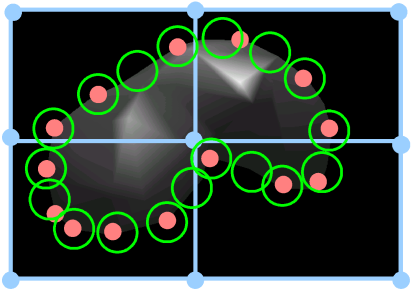


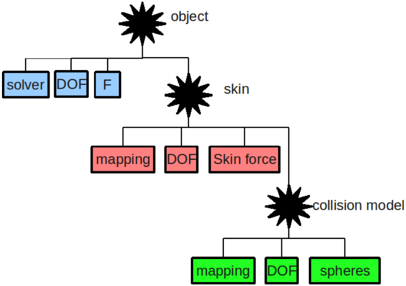


Layered object

Detailed geometry embedded in a coarse deformable grid

- independent DOFs (blue)
- skin vertices (salmon)
- mapping
- collision samples (green)
- collision mapping

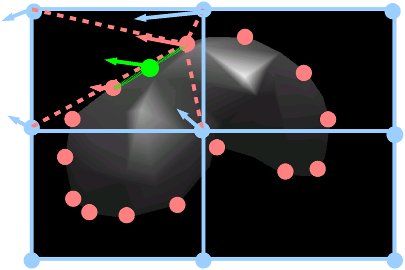




Layered object

Detailed geometry embedded in a coarse deformable grid

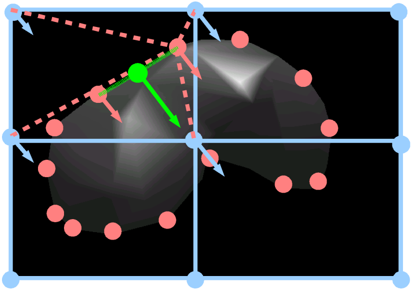
- independent DOFs (blue)
- skin vertices (salmon)
- mapping
- collision samples (green)
- collision mapping
- apply displacements
  1. $v_{skin} = J_{skin} v$
  2. $v_{collision} = J_{collision} v_{skin}$



Layered object

Detailed geometry embedded in a coarse deformable grid

- independent DOFs (blue)
- skin vertices (salmon)
- mapping
- collision samples (green)
- collision mapping
- apply displacements
  1. $v_{skin} = J_{skin} v$
  2. $v_{collision} = J_{collision} v_{skin}$
- apply forces
  1. $f_{skin} = J_{collision}^T f_{collision}$
  2. $f = J_{skin}^T f_{skin}$



More on mappings

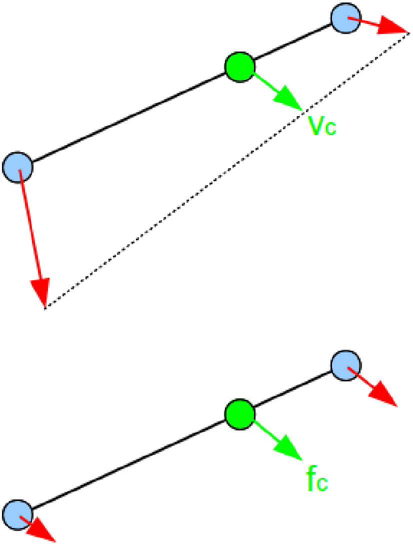
- Map a set of degrees of freedom (the parent) to another (the child).
- Typically used to attach a geometry to control points (but see Flexible and Compliant plugins).
- Child degrees of freedom (DOF) are not independent : their positions are totally defined by their parent's.
- Displacements are propagated top-down (parent to child) : $v_{child} = Jv_{parent}$
- Forces are accumulated bottom-up : $f_{parent} += J^T f_{child}$

The physics of mappings

Example : line mapping

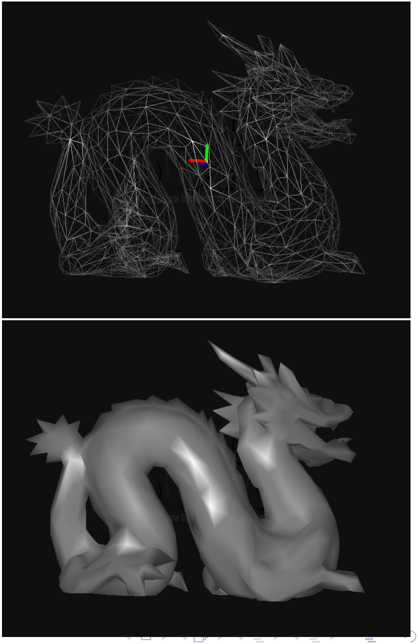
$$v_c = \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = Jv$$

$$\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} f_c = J^T f_c$$



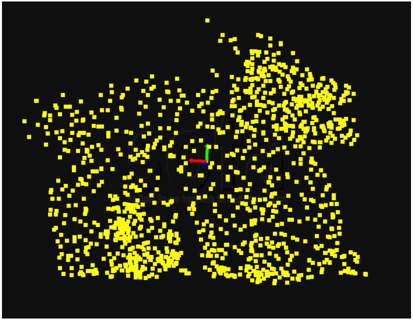
Examples of mappings

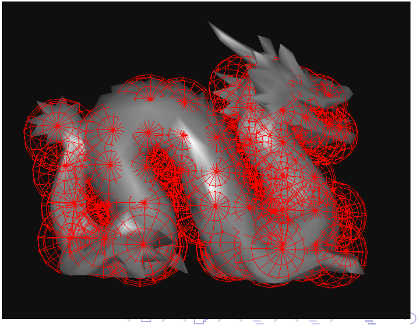
- RigidMapping can be used to attach points to a rigid body
  - to attach a visual model



Examples of mappings

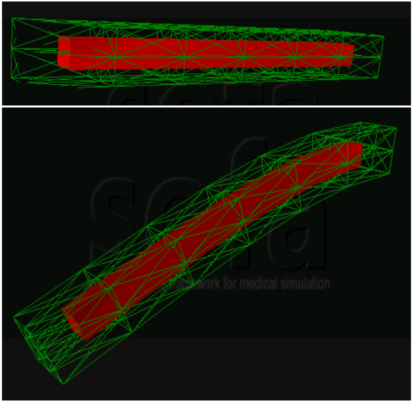
- RigidMapping can be used to attach points to a rigid body
  - to attach collision surfaces





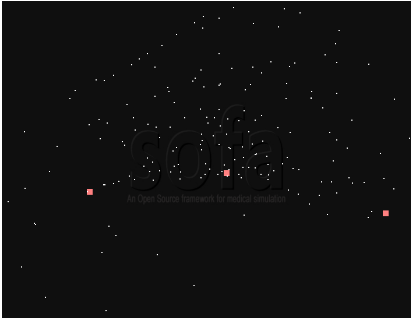
Examples of mappings

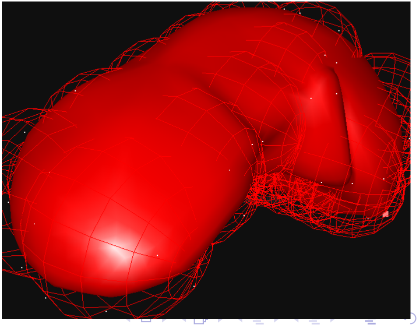
- RigidMapping can be used to attach points to a rigid body
- BarycentricMapping can be used to attach points to a deformable body
  - to attach a visual model



Examples of mappings

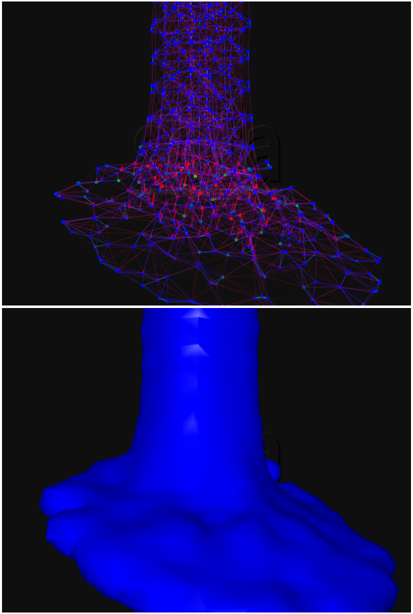
- RigidMapping can be used to attach points to a rigid body
- BarycentricMapping can be used to attach points to a deformable body
  - to attach collision surfaces





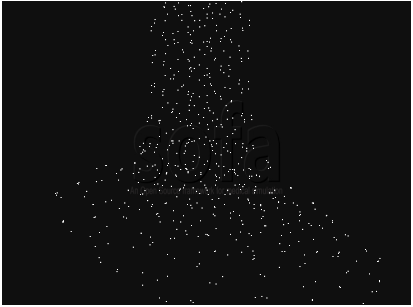
Examples of mappings

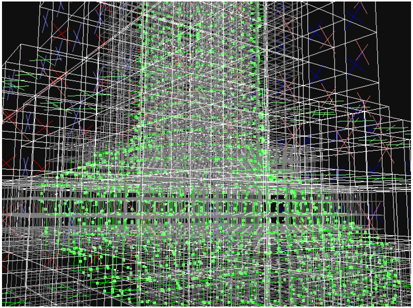
- RigidMapping can be used to attach points to a rigid body
- BarycentricMapping can be used to attach points to a deformable body
- More advanced mapping can be applied to fluids



Examples of mappings

- RigidMapping can be used to attach points to a rigid body
- BarycentricMapping can be used to attach points to a deformable body
- More advanced mapping can be applied to fluids





On the physical consistency of mappings

- Conservation of energy :
  Necessary condition : $v_{child} = Jv_{parent} \Rightarrow f_{parent} + = J^T f_{child}$
- Conservation of momentum :
  Mass is modeled at one level only. There is no transfer of momentum.
- Constraints on displacements (e.g. incompressibility, fixed points) are not easily applied *at the child level*

Outline

Motivation

Simple bodies

Layered objects using node hierarchies
    Mappings

Interacting objects

Implementation

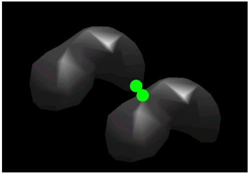
Collision detection and response

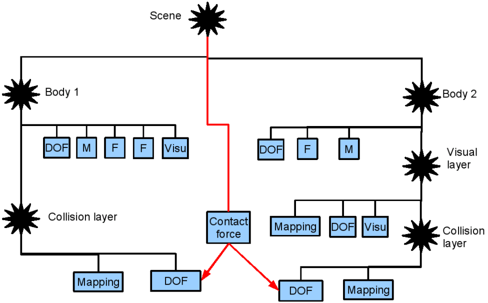
Parallelism

Conclusion

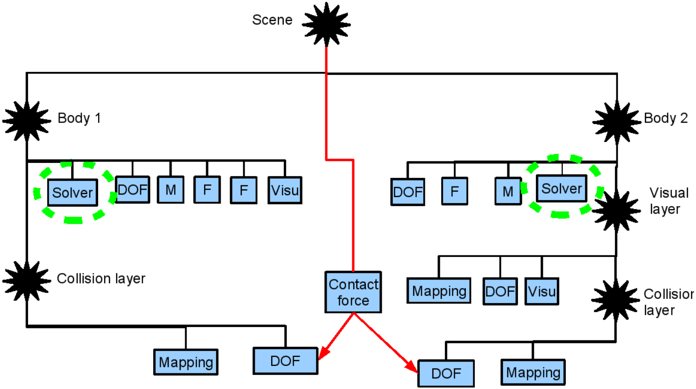
Two objects in contact

Example : 2-layer liver against 3-layer liver using a contact force.
Use extended trees (Directed Acyclic Graphs) to model trees with loops.



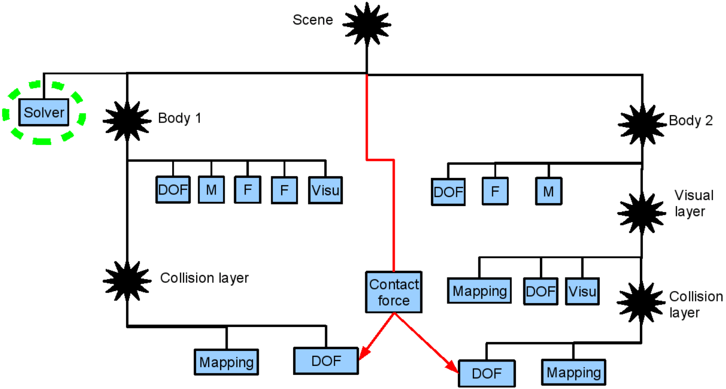


ODE solution of interacting objects



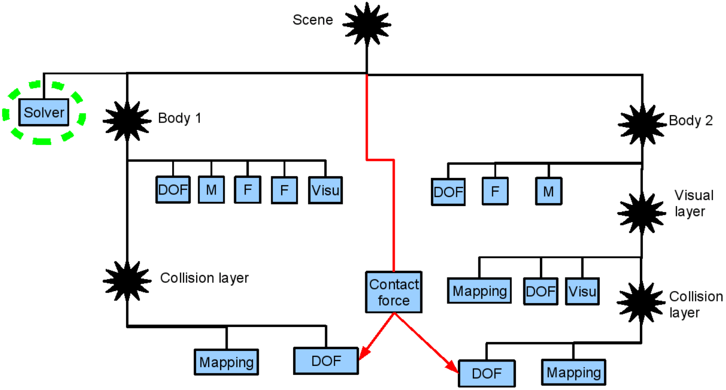
- Soft interactions : independent processing, no synchronization required

ODE solution of interacting objects



- Soft interactions : independent processing, no synchronization required
- Stiff interactions : unified implicit solution with linear solver, synchronized objects

ODE solution of interacting objects



- Soft interactions : independent processing, no synchronization required
- Stiff interactions : unified implicit solution with linear solver, synchronized objects
- Hard interaction constraints using Lagrange multipliers

Outline

Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting objects

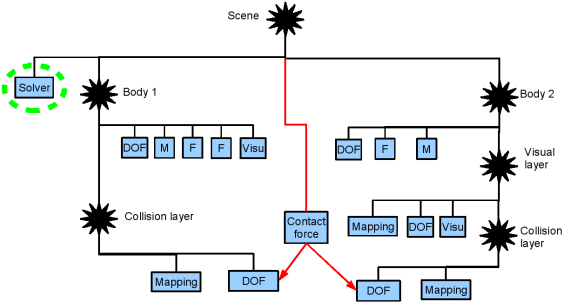
Implementation

Collision detection and response

Parallelism

Conclusion

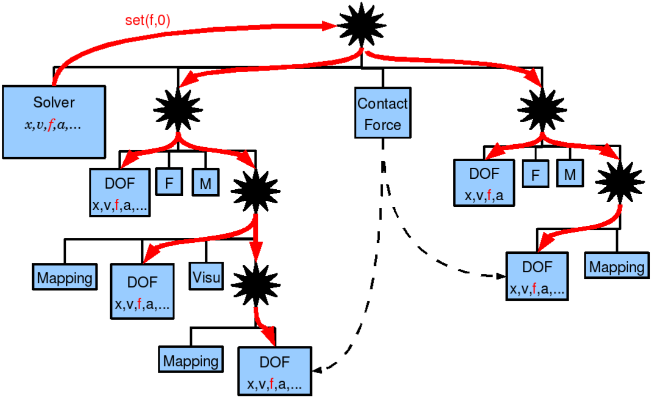
Actions implemented by Visitors



- No global state vector
- Operation = graph traversal + abstract methods + vector identificators

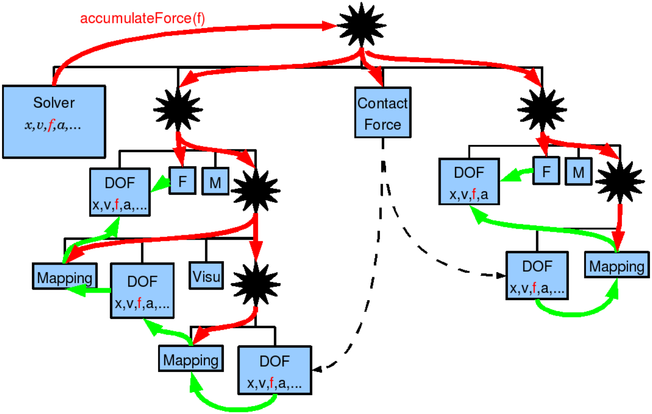
Example : clearing a global vector

- The solver triggers an action starting from its parent system and carrying the necessary symbolic information
- the action is propagated through the graph and calls the appropriate methods at each DOF node



Example : accumulating the forces

- The solver triggers the appropriate action
- the action is propagated through the graph and calls the appropriate (botom-up) methods at each Force and Mapping node



Efficient implicit integration

- Large time steps for stiff internal forces and interactions
- solve $(\alpha M + \beta h^2 K)\Delta v = h(f + hKv)$ Iteratively using a conjugate gradient solution

Actions :

- propagateDx
- computeDf
- vector operations
- dot product (only global value directly accessed by the solver)

System assembly in the Compliant plugin

Efficiency

- No global state vector
  - they are scattered over the DOF components
  - each DOF component can be based on its own types (e.g. Vec3, Frame, etc.)
  - symbolic values are used to represent global state vectors
- Action = graph traversal + global vector ids + call of abstract top-down and bottom up methods
  - Displacements are propagated top-down
  - Interactions forces are evaluated after displacement propagation
  - Forces are accumulated bottom-up
  - virtual functions applied to components

Outline

Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting objects

Implementation

Collision detection and response

Parallelism

Conclusion

Collision detection and response

CollisionPipeline component orchestrates specific components

- BroadPhase : bounding volume intersections
- NarrowPhase : geometric primitive intersections
- Reaction : what to do when collisions occur
- GroupManager : putting colliding objects under a common solver

Recent work uses the GPU

Outline

Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting objects

Implementation

Collision detection and response

Parallelism

Conclusion

Parallelism in time integration

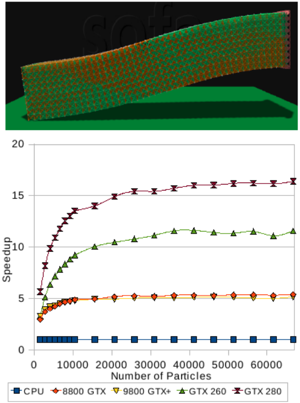
Different levels of parallelism :

- Low level : GPU implementations of components
- High level : task-based using data dependencies
- Thread-based using the Multithread plugin

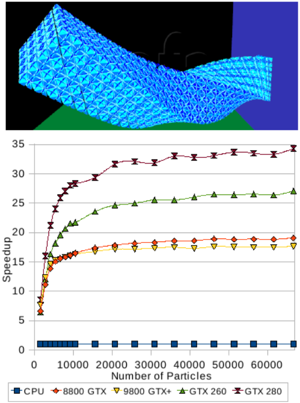
We can combine them !

GPU Parallelism

- StiffSpringForceField, TetrahedronFEMForceField, HexahedronFEMForceField are implemented on the GPU
- The DOF component makes data transfer transparent
- CPU and GPU components can be used simultaneously
- Nice speedups



(a) Mass-Spring



(b) Co-rotational FEM

Outline

Motivation

Simple bodies

Layered objects using node hierarchies

- Mappings

Interacting objects

Implementation

Collision detection and response

Parallelism

Conclusion

Conclusion - Features

High modularity :

- Abstract components : DOF, Force, Constraint, Solver, Topology, Mass, CollisionModel, VisualModel, etc.
- Multimodel simulations using mappings
- Explicit and implicit solvers, Lagrange multipliers

Efficiency :

- global vectors and matrices are avoided
- parallel implementations

Implementation :

- currently $> 750,000$ C++ lines
- Linux, MacOs, Windows

Ongoing work

- models and algorithms : better numerical solvers, cutting, haptics, Eulerian fluids...
- asynchronous simulation/rendering/haptic feedback
- multiphysics (electrical/mechanical)
- parallelism for everyone
- more documentation

www.sofa-framework.org