

# SOFA: a modular yet efficient physical simulation architecture

François Faure, INRIA

October 2012



Evasion



MOAIS

# Outline

## Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting bodies

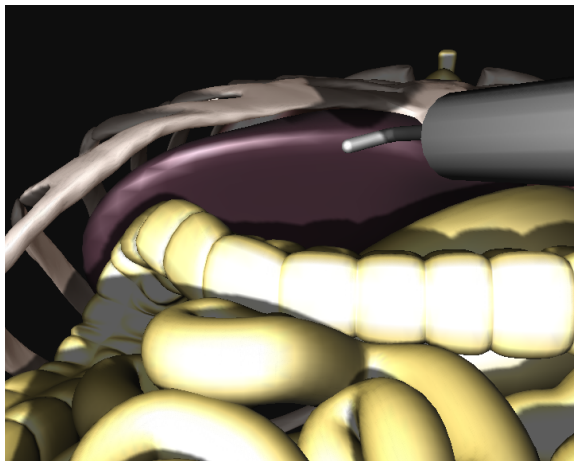
Data processing

Collision detection and response

Parallelism

Conclusion

# A complex physical simulation



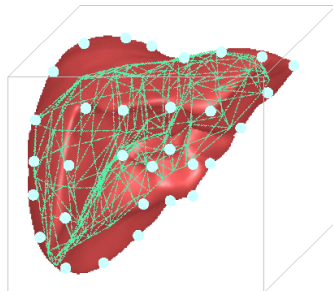
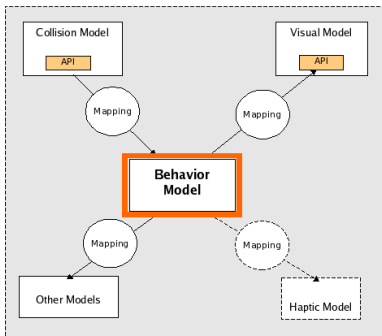
Material, internal forces, constraints, contact detection and modeling, ODE solution, visualization, interaction, etc.

# Simulation platforms

- ▶ Current platforms (ODE, Havok, 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.

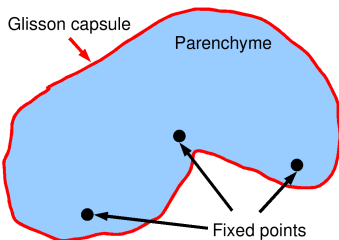
# A generic approach

- ▶ 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 bodies

Data processing

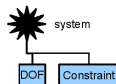
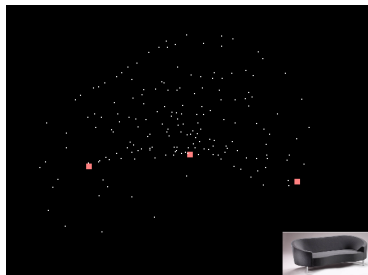
Collision detection and response

Parallelism

Conclusion

# Components

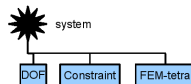
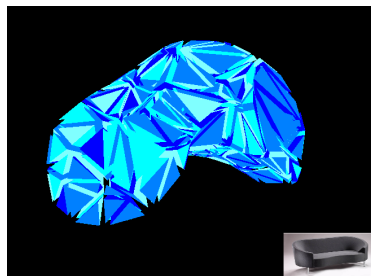
- ▶ state vectors (DOF) :  
 $x, v, a, f$
- ▶ constraints : fixed points  
*other : oscillator, collision plane, etc.*





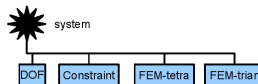
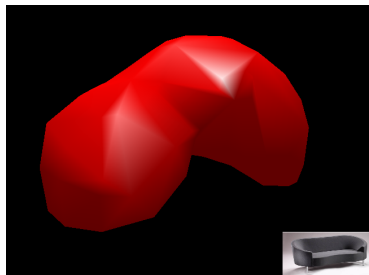
# Components

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



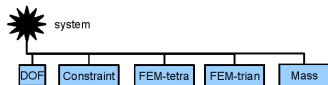
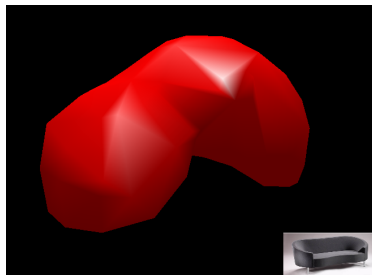
# Components

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



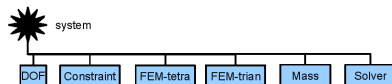
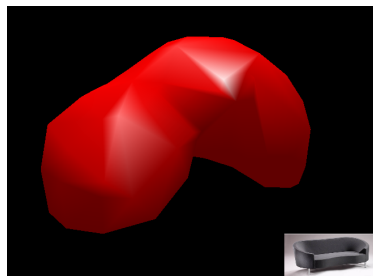
# Components

- ▶ 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, implicate Euler, static solution, etc.*



# Operations

- ▶ 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 bodies

Data processing

Collision detection and response

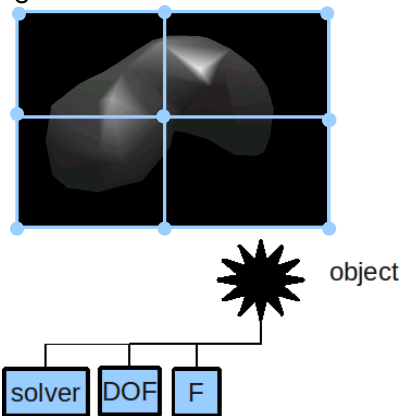
Parallelism

Conclusion

# Layered object

Object embedded in a deformable grid

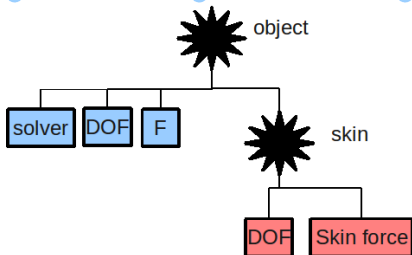
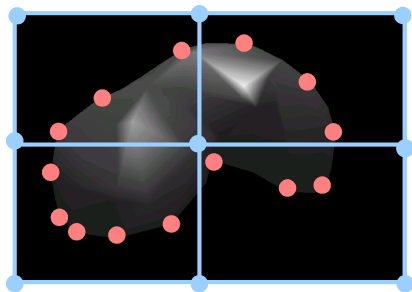
- independent DOFs (blue)



# Layered object

Object embedded in a deformable grid

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

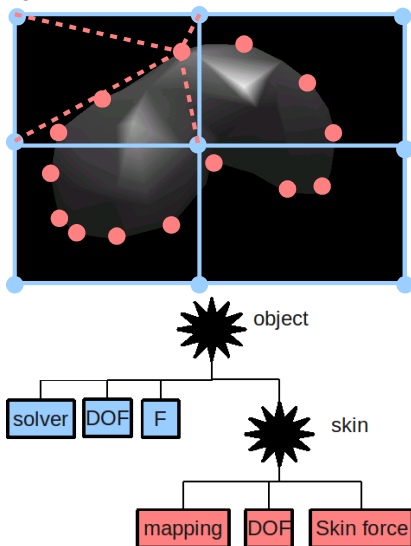




# Layered object

Object embedded in a deformable grid

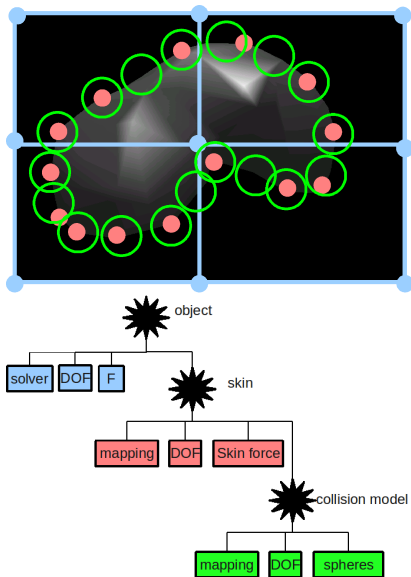
- ▶ independent DOFs (blue)
- ▶ skin vertices (salmon)
- ▶ mapping



# Layered object

Object embedded in a deformable grid

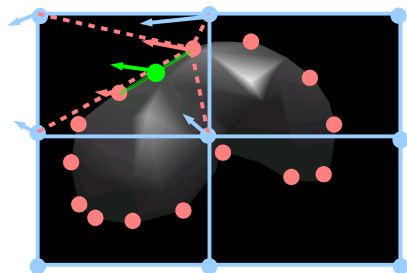
- ▶ independent DOFs (blue)
- ▶ skin vertices (salmon)
- ▶ mapping
- ▶ collision samples (green)
- ▶ collision mapping



# Layered object

Object embedded in a 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

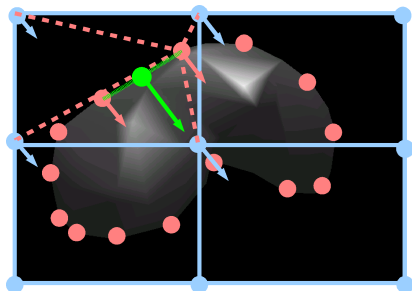
Object embedded in a deformable grid

- ▶ 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.
- ▶ 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 propagated bottom-up

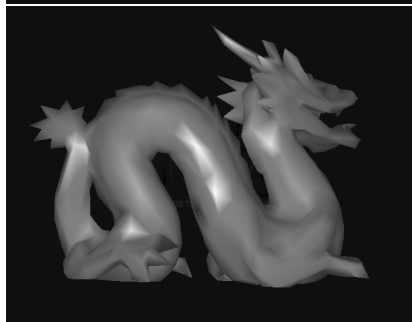
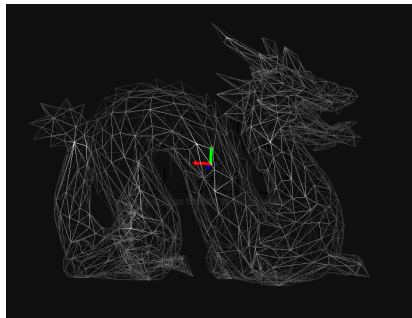
# The physics of mappings

Example : line mapping

$$\begin{aligned} v &= \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = Jv \\ \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} &= \begin{pmatrix} a \\ b \end{pmatrix} f = J^T f \end{aligned}$$

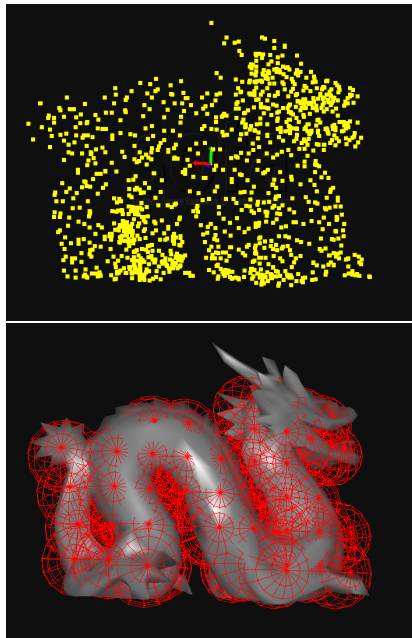
# Examples of mappings

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



# Examples of mappings

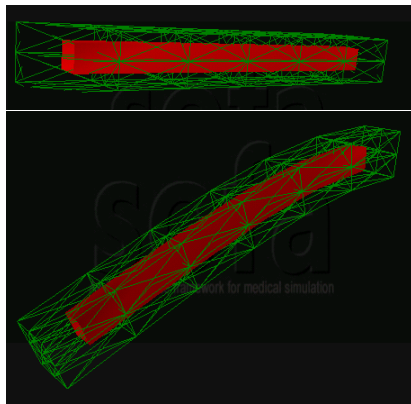
- ▶ RigidMapping can be used to attach points to a rigid body
  - ▶ to attach collision surfaces





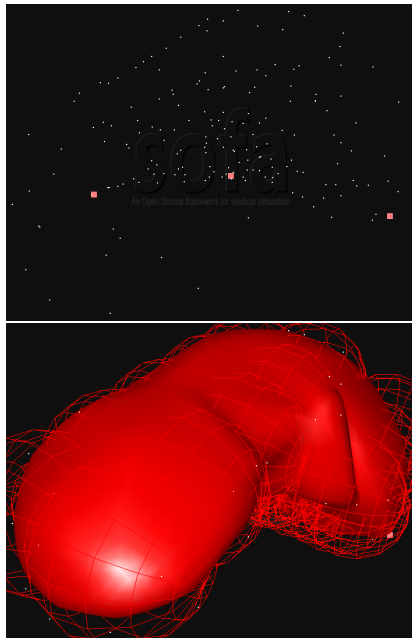
# Examples of mappings

- ▶ 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



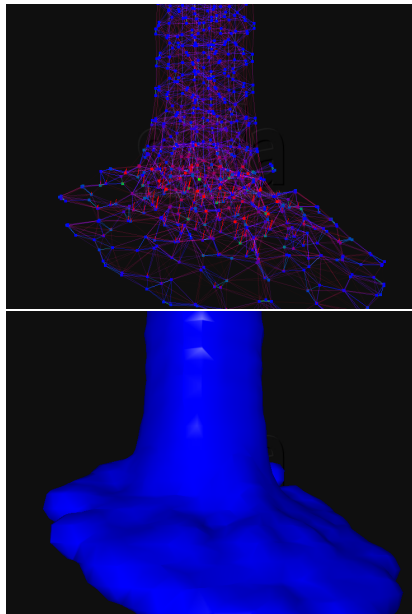
# Examples of mappings

- ▶ 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



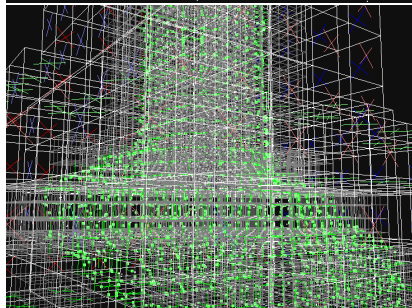
# Examples of mappings

- ▶ 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 bodies**

Data processing

Collision detection and response

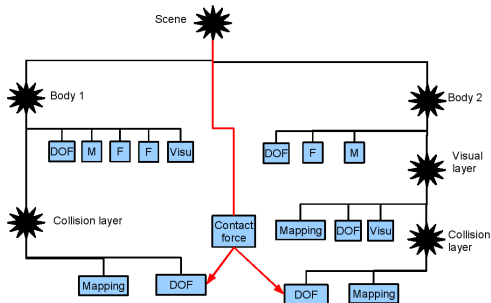
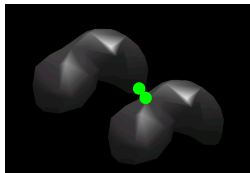
Parallelism

Conclusion

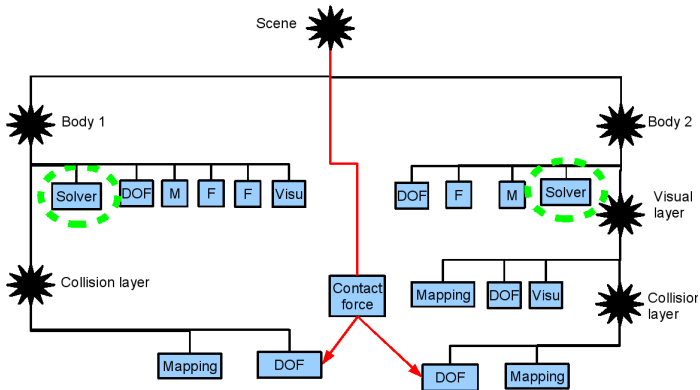
# Two bodies in contact

Example : 2-layer liver against 3-layer liver using a contact force.

Use extended trees (Directed Acyclic Graphs) to model trees with loops.



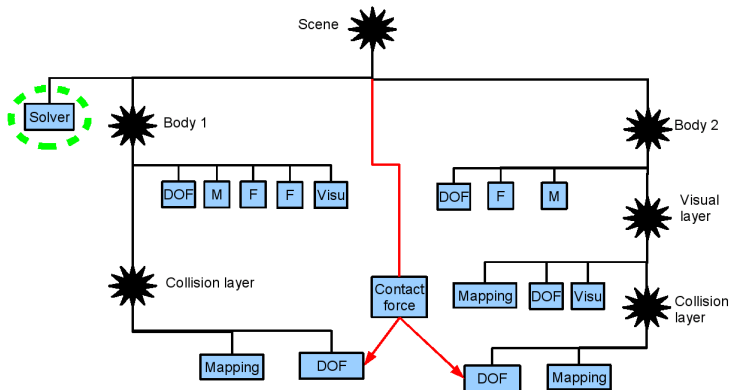
## ODE solution of interacting bodies



- ▶ Soft interactions : independent processing, no synchronization required

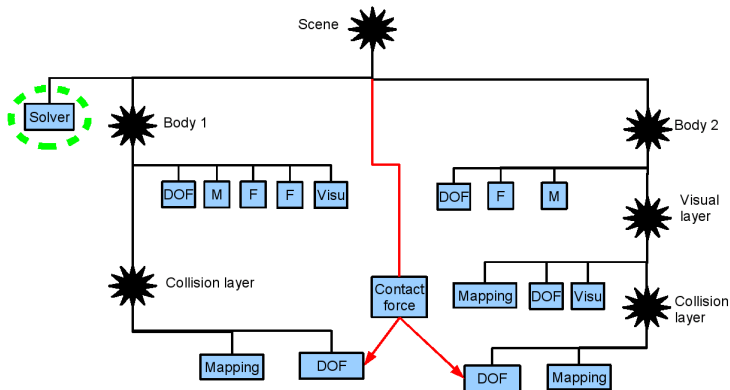


# ODE solution of interacting bodies



- ▶ Soft interactions : independent processing, no synchronization required
- ▶ Stiff interactions : unified implicit solution with Conjugate Gradient, synchronized objects

# ODE solution of interacting bodies



- ▶ Soft interactions : independent processing, no synchronization required
- ▶ Stiff interactions : unified implicit solution with Conjugate Gradient, synchronized objects
- ▶ Hard interaction constraints : work in progress

# Outline

Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting bodies

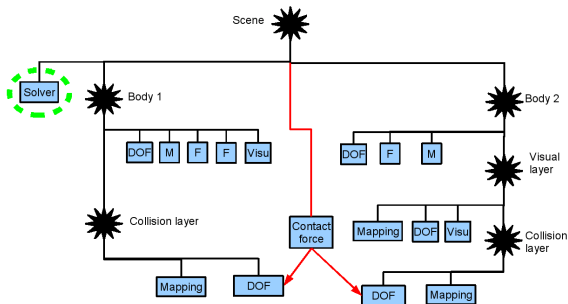
**Data processing**

Collision detection and response

Parallelism

Conclusion

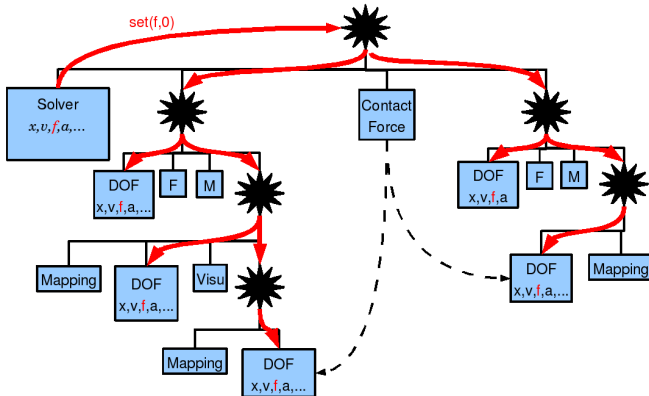
# Actions



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

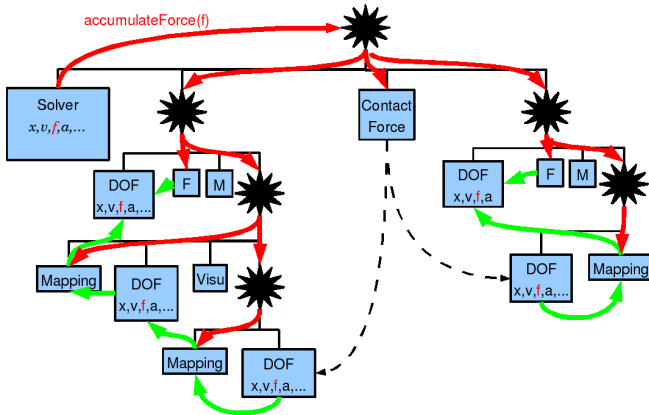
## Example : clearing a global vector

- ▶ 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 (bottom-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)

Ongoing work : building global vectors and matrices also

# 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 bodies

Data processing

**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 bodies

Data processing

Collision detection and response

**Parallelism**

Conclusion

# Parallelism in time integration

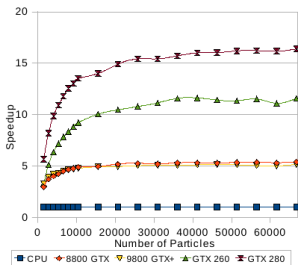
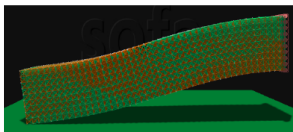
Two levels of parallelism :

- ▶ Low level : GPU implementations of components
- ▶ High level : task-based using data dependencies

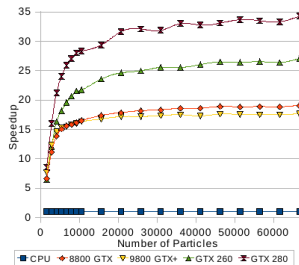
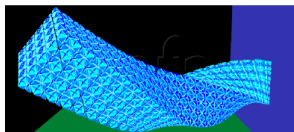
We can combine them !

# 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

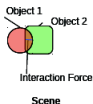


(b) Co-rotational FEM

# High-level Parallelism

Everton Hermann's Ph.D. thesis (2007-2010) :

- analyses solver operations

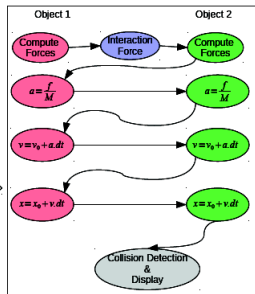


1: *Compute Force*  
2:  $a = f / M$   
3:  $v = v_0 + a \cdot dt$   
4:  $x = x_0 + v \cdot dt$   
5: *Collision Detection*  
6: *Display*

Animation Visitors (a)



Scene Graph (b)

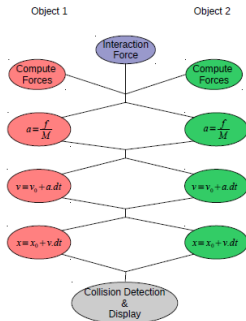


Time Integration Tasks (c)

# High-level Parallelism

Everton Hermann's Ph.D. thesis (2007-2010) :

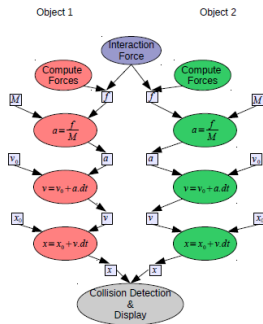
- analyses solver operations
- exploits scene graph branches



# High-level Parallelism

Everton Hermann's Ph.D. thesis (2007-2010) :

- ▶ analyses solver operations
- ▶ exploits scene graph branches
- ▶ removes spurious synchronizations using data dependency analysis [Athapascan]
- ▶ efficiency ranges for 50% to 90%





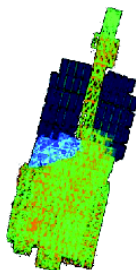
# High-level Parallelism

Everton Hermann's Ph.D. thesis (2007-2010) :

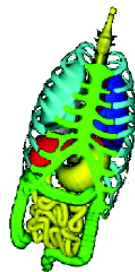
- ▶ analyses solver operations
- ▶ exploits scene graph branches
- ▶ removes spurious synchronizations using data dependency analysis [Athapascan]
- ▶ efficiency ranges for 50% to 90%



(a) Visual Model



(b) Behavior Model

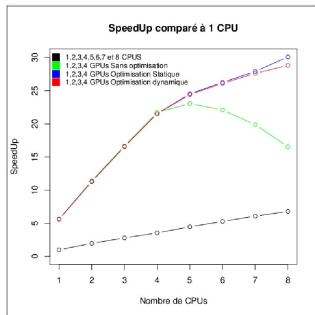


(c) Processor Distribution

# Hybrid Parallelism

Work in progress

- ▶ Use high level and low level simultaneously
- ▶ Dynamically balance between multiple CPUs and GPUs



# Outline

Motivation

Simple bodies

Layered objects using node hierarchies

Mappings

Interacting bodies

Data processing

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 450,000 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 (release 1.0 this year ?)

[www.sofa-framework.org](http://www.sofa-framework.org)