Concepts of SOFA

The SOFA team

2007

Abstract

This document contains discussions and references explaining the main concepts used within SOFA. Comments and suggestions are welcome on the SOFA devel mailing list.

1 Mappings

François Faure

1.1 Motivation

Different geometrical models can be used to model objects in contact. We organize them in a hierarchy of layers. An example is shown in figure 1, where a rigid object hits a shape embedded in deformable cells.

The state of a simulated system can be described by the values and time derivatives of its independent degrees of freedom (DOF) gathered in two vectors x_0 and v_0. The dynamics equation (Newton's law) relates the second time derivative a_0 of the DOF to the forces f_0 acting on them: $f_0 = Ma_0$, where M is a matrix modeling the mass of the system.

A geometrical model can be attached to the DOF for visualization or contact computation. Its DOF positions, velocities and associated forces are stored in vectors x_1, v_1 and f_1, respectively. They are not independent variables, since the positions and velocities are bound to the independent DOF. We say that the child geometrical model 1 is mapped from the parent model 0, using a kinematic operator which we call *mapping*:

$$\begin{aligned} x_1 &= \mathcal{J}_1(x_0) \\ v_1 &= J_1 v_0 \end{aligned}$$

Typical mappings include polygonal shapes attached to rigid bodies using local coordinates, or embedded in deformable cells using barycentric coordinates, as well as skin surrounding articulated bodies using vertex blending techniques. Matrix $J_1 = \frac{\partial x_1}{\partial x_0}$ encodes the linear relation between the DOF velocities and the shape velocities. Due to linearity, the same relation holds on elementary displacements dx. It also holds on accelerations, with an additional offset due to velocities when the position mapping J is nonlinear. In most cases, operators J and $\mathbf{J}$ are the same, but in the case of rigid bodies, J is nonlinear with respect to x_0 and it can not be written as a matrix. For surfaces embedded in deformable cells, matrix $\mathbf{J}$ contains the barycentric coordinates. For surfaces attached to rigid bodies, each row of the matrix encodes the usual relation $v = \dot{o} + \omega \times (x - o)$ for each vertex. Similarly, skins around articulated bodies involve, at each vertex, the weighted contributions of the rigid bodies.

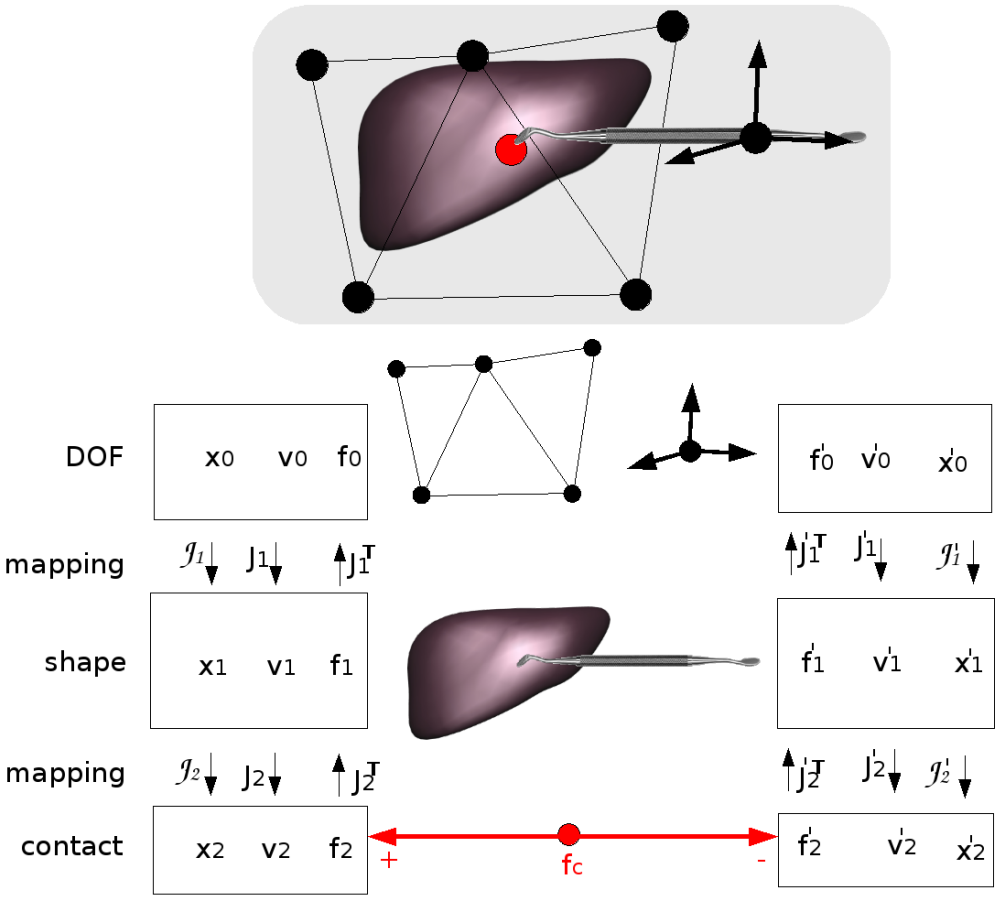

Figure 1: Mappings from the DOFs to a contact point. Top: two simulated objects in contact (red point). Bottom: hierarchy of geometrical layers. Positions and velocities are propagated top-down. The contact force f_c is accumulated in the contact layers. Forces are then propagated bottom-up.

When shapes collide, additional geometry can be necessary to model the contact. For instance, when an edge intersects another one, a contact force is applied to the intersection points. These points are defined by their barycentric coordinates with respect to their edge vertices. Other relations can be used, depending on the kind of geometrical primitives in contact. This additional geometry requires another geometrical layer connected to the shape by a mapping, as illustrated in figure 1. We extend this approach to tree-like hierarchies of geometries, with the independent DOFs at the root. For instance, the independent DOF may have two children, one for collision using a coarse mesh, and the other for rendering using a finer mesh. The synchronization between these siblings is automatically guaranteed by their attachment to their common ancestor, the DOF layer.

Positions and velocities are propagated top-down in the hierarchy. Conversely, in order to take all the forces into account in the dynamics equation, the forces are propagated bottom-up, up to the independent DOFs, where Newton's law $f = ma$ is applied. Given forces f_c applied to a child model, the mapping computes and accumulates the equivalent forces f_p applied to its parent. Since equivalent forces must have the same power, the following relation holds:

$$v_p^T f_p = v_c^T f_c$$

The kinematic relation $v_c = J v_p$ allows us to rewrite the previous equation as

$$v_p^T f_p = v_p^T J^T f_c$$

Since this relation holds for any velocity v_p, the principle of virtual work allows us to simplify the previous equation to obtain:

$$f_p = J^T f_c \tag{1}$$

1.2 Mapping functions

As seen in section 1.1, the mappings propagate positions, velocities, displacements and accelerations top-down, and they propagate forces bottom-up. The top-down propagation methods are:

- `apply (const MechanicalParams*, MultiVecCoordId outPos, ConstMultiVecCoordId inPos )` for positions,
- `applyJ(const MechanicalParams*, MultiVecDerivId outVel, ConstMultiVecDerivId inVel )` for velocities and small displacements,
- `computeAccFromMapping(const MechanicalParams*, MultiVecDerivId outAcc, ConstMultiVecDeri inVel, ConstMultiVecDerivId inAcc )` for accelerations, taking into account velocity-dependent accelerations in nonlinear mappings.

The bottom-up propagation methods are:

- `applyJT(const MechanicalParams*, MultiVecDerivId inForce, ConstMultiVecDerivId outForce )` for child forces or changes of child forces,
- `applyDJT(const MechanicalParams*, MultiVecDerivId parentForce, ConstMultiVecDerivId childForce )` for changes of parent force due to a change of mapping with constant child force,
- `applyJT(const ConstraintParams*, MultiMatrixDerivId inConst, ConstMultiMatrixDerivId outConst )` for constraint Jacobians,

The name of the methods used to propagate velocities or small displacements top-down contain J, which denotes the kinematic matrix, while the names of the methods used to propagate forces or constraint Jacobians bottom-up contain JT, which denotes the transpose of the same. Method `applyJT(const MechanicalParams*, MultiVecDerivId inForce, ConstMultiVecDerivId outForce )` is used to accumulate forces from a child model to its parent. It performs a cumulative write (+=) since a model may have several children:

$$f_p += J^T f_c \tag{2}$$

Some differential equation solvers need compute the change of force df, given a change of position dx. The displacement dx, considered small, is propagated top-down using the linear operator `applyJ(const MechanicalParams*, MultiVecDerivId outVel, ConstMultiVecDerivId inVel )`, then the force changes are accumulated bottom-up. Differentiating eq.2, we get:

$$\delta f_p += J^T \delta f_c + \delta J^T f_c \tag{3}$$

Once the change of child force δf_c is computed (see the section on force fields), method `applyJT(const MechanicalParams*, MultiVecDerivId inForce, ConstMultiVecDerivId outForce )` is used to accumulate it in the parent, corresponding to the first term in the right of eq.3. Method `applyDJT(const MechanicalParams*, MultiVecDerivId parentForce, ConstMultiVecDerivId childForce )` is used to accumulate the second term, which is due to the change of matrix J due to a displacement. It is null in linear mappings, such as BarycentricMapping. This method queries the last displacement propagated and the child force using the MechanicalParams.

Constraints enforced using Lagrange multipliers are represented using linear equations. If a linear constraint on the child DOFS in expressed as $L_c v_c = a$, where L_c is the Jacobian of the constraint, then the equivalent constraint at the parent level is: $L_p v_p = a$, where $L_p = J^T L_c$. Method applyJT(const ConstraintParams*, MultiMatrixDerivId inConst, ConstMultiMatrixDerivId outConst) is used to compute L_p. Since the Jacobians are generally sparse, they are encoded in sparse matrices instead of the dense vectors used for forces.