Compliant plugin

François Faure

2012

1 Constraint forces

1.1 Generalized constraints

This approach unifies soft and hard constraints, by providing constraints with compliance. Hard constraints are usually implemented using Lagrange multipliers λ in the following equation:

$$\begin{pmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{ext} \\ -\phi \end{pmatrix} \tag{1}$$

where $\mathbf{M}$ is the mass matrix (or, more generally, a dynamics matrix such as $\mathbf{M} - h^2\mathbf{K}$ used in implicit time integration), $\mathbf{J}$ is the Jacobian matrix of the constraint(s), $\mathbf{a}$ is the acceleration, $\mathbf{f}_{ext}$ is the net external force applied to the system, λ is the constraint force and ϕ is the constraint violation. λ and ϕ are vectors with as many entries as scalar constraints. The equation system is typically solved using a Schur complement to compute the constraint forces:

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\lambda = -\phi - \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_{ext} \tag{2}$$

and then the acceleration is computed as $\mathbf{a} = \mathbf{M}^{-1}(\mathbf{f}_{ext} + \mathbf{J}^T\lambda)$.

In the generalized approach, the constraint forces are considered proportional to the constraint violations:

$$\lambda = -\frac{1}{c}\left(\phi + d\dot{\phi}\right) \tag{3}$$

where the positive real number c is the compliance of the constraint and d is its damping ratio. Combined with a time discretization scheme, this leads to an equation system similar to (2) as shown in Section 2.

1.2 Force or constraint ?

ForceFields generally contribute the the top line of Equation 1, by accumulating force in the right-hand term. In implicit integration, their stiffness matrix is also accumulated to the left-hand side. When a ForceField has an invertible stiffness matrix, it can be handled as a constraint with non-null compliance, rather than a standard force. In this case, it contributes to the bottom of Equation 1 rather than to the top.

Due to matrix conditioning issues, and depending on which linear solver is used, it may be a good idea to process very stiff forces as low compliances rather than large stiffnesses. Note, however, that each force handled as a constraint increases the size of the equation system, since it adds lines to the bottom (and columns to the right) of the equation system. Conversely, it may be more efficient to handle soft forces as stiffnesses, since the size of the stiffness matrix depends on the number of independent DOFs, which does not increase along with the number of forces.

Note that geometric stiffness is not handled in the compliance view, which may result in instabilities in case of large displacements.

2 Time integration

Our integration scheme is:

$$\begin{aligned}\mathbf{v}_{n+1} &= \mathbf{v}_n + h\mathbf{M}^{-1}\left(\alpha\mathbf{f}_{n+1} + (1-\alpha)\mathbf{f}_n\right) &(4)\\ \mathbf{x}_{n+1} &= \mathbf{x}_n + h\left(\beta\mathbf{v}_{n+1} + (1-\beta)\mathbf{v}_n\right) &(5)\end{aligned}$$

where index n denotes current values while index $n+1$ denotes next values, α is the implicit velocity factor, and β is the implicit position factor. Let

$$\begin{aligned}\mathbf{\Delta v} = \mathbf{v}_{n+1} - \mathbf{v}_n &= h\mathbf{M}^{-1}\left(\alpha\mathbf{f}_{n+1} + (1-\alpha)\mathbf{f}_n\right) &(6)\\ \mathbf{\Delta x} = \mathbf{x}_{n+1} - \mathbf{x}_n &= h(v + \beta\mathbf{\Delta v}) &(7)\end{aligned}$$

be the velocity and position changes across the time step. We have not carefully studied the influence of these parameters, but it seems that $\alpha = 0.5$ and $\beta = 1$ corresponds to an **energy-conserving integration scheme**.

The constraint violation ϕ and its Jacobian $\mathbf{J}$ are:

$$\begin{aligned}\mathbf{J} &= \frac{\partial\phi}{\partial\mathbf{x}} &(8)\\ \phi_{n+1} &\simeq \phi_n + \mathbf{J\Delta x} = \phi_n + h\dot{\phi} + \mathbf{J}h\beta\Delta\mathbf{v} &(9)\\ \dot{\phi}_{n+1} &\simeq \dot{\phi}_n + \mathbf{J\Delta v} &(10)\end{aligned}$$

The corresponding forces are:

$$\begin{aligned}\mathbf{f} &= \mathbf{f}_{ext} + \mathbf{J}^T\lambda &(11)\\ \lambda_i &= -\frac{1}{c_i}(\phi_i + d\dot{\phi}_i) &(12)\end{aligned}$$

where the subscript i denotes a scalar constraint.

The average constraint forces are computed using equations 12, 9 and 10:

$$\begin{aligned}\bar{\lambda}_i &= \alpha\lambda_{n+1} + (1-\alpha)\lambda_n\\ &= -\frac{1}{c_i}(\alpha\phi + \alpha h\dot{\phi} + \alpha h\beta\mathbf{J}\Delta\mathbf{v} + \alpha d\dot{\phi} + \alpha d\mathbf{J\Delta v} + (1-\alpha)\phi + (1-\alpha)d\dot{\phi})\\ &= -\frac{1}{c_i}(\phi + d\dot{\phi} + \alpha h\dot{\phi} + \alpha(h\beta + d)\mathbf{J\Delta v})\end{aligned}$$

We can rewrite the previous equation as:

$$\mathbf{J\Delta v} + \frac{1}{\alpha(h\beta+d)}\mathbf{C}\bar{\lambda} = -\frac{1}{\alpha(h\beta+d)}(\phi + (d+\alpha h)\dot{\phi}) \qquad (13)$$

where values without indices denote current values. The complete equation system is:

$$\begin{pmatrix} \frac{1}{h}\mathbf{PM} & -\mathbf{PJ}^T \\ \mathbf{J} & \frac{1}{l}\mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{\Delta v} \\ \bar{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{Pf}_{ext} \\ -\frac{1}{l}(\phi + (d+\alpha h)\dot{\phi}) \end{pmatrix} \qquad (14)$$

where $l = \alpha(h\beta + d)$ The system is singular due to matrix $\mathbf{P}$, however we can use $\mathbf{PM}^{-1}\mathbf{P}$ as inverse mass matrix to compute a Schur complement:

$$\begin{aligned}\left(h\mathbf{JPM}^{-1}\mathbf{PJ}^T + \tfrac{1}{l}\mathbf{C}\right)\bar{\lambda} &= -\tfrac{1}{l}\left(\phi + (d+h\alpha)\dot{\phi}\right) - h\mathbf{JM}^{-1}\mathbf{f}_{ext}\\ \mathbf{\Delta v} &= h\mathbf{PM}^{-1}(\mathbf{f}_{ext} + \mathbf{J}^T\bar{\lambda})\\ \mathbf{\Delta x} &= h(\mathbf{v} + \beta\mathbf{\Delta v})\end{aligned}$$

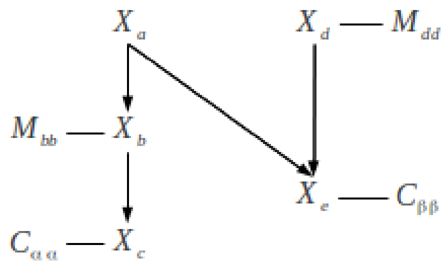

Figure 1: A mechanical system. The X represent mechanical states, while the arrows represent the kinematic hierarchy, and the plain lines represent components acting on the states.

3 Matrix assembly

The equation system, in its most general form, can be written as:

$$\begin{pmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \phi \end{pmatrix} \tag{15}$$

We assemble the 7 terms of equation 15 separately.

Figure 1 shows an example of mechanical system. The independent DOFs are X_a and X_d. State X_b is attached to X_a using a simple mapping, and a mass matrix M_{bb} is defined at this level. State X_c is attached to X_b using a simple mapping, and a compliance matrix $C_{\alpha\alpha}$ (possibly a deformation force) is applied to these DOFs. State X_e is attached to X_a and X_d at the same time, using a MultiMapping. A compliance matrix $C_{\beta\beta}$, possibly an interaction force, is applied to these DOFs, while a mass M_{dd} is applied to X_d.

The corresponding equation system has the block structure shown in the right of Figure 2. The main blocks of the equation system are highlighted in grey rectangles. The J matrices are the mapping matrices. The bottom row has two mappings, since the state X_e impacted by compliance β depends on two parent states.

The assembly of local matrices and vectors to the global matrices and vectors which compose the system is performed using shift matrices, as illustrated in Figure 3. Shift matrices J_{*0}, shown in the top of the figure, are composed of identity matrices (represented with a diagonal in a block) and null blocks. They can be used to implement the shifting of vector entry indices from local to global. The assembly is thus easily expressed as a sum of product of matrices, as illustrated in Figure 4. While shifting values in dense arrays using matrix products is not efficient (vector assembly is actually implemented using shifted copies), the sum of matrix products is a reasonable implementation of sparse matrix assembly.

4 Implementation

The virtual functions used for setting up the equation system are declared in BaseForceField.h and BaseMapping.h, and documented in “Experimental Compliance API” documentation blocks.

Figure 2: Block view of the equation 15 applied to the system of Figure 1, with non-null blocks highlighted in yellow.

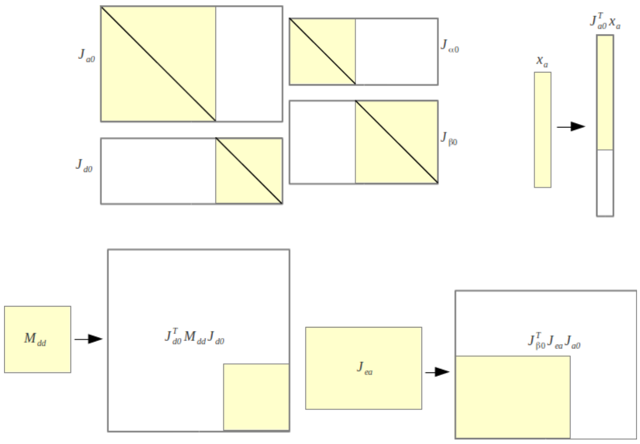

Figure 3: Shift matrices J_{*0} are used to place blocks within global vectors and matrices.

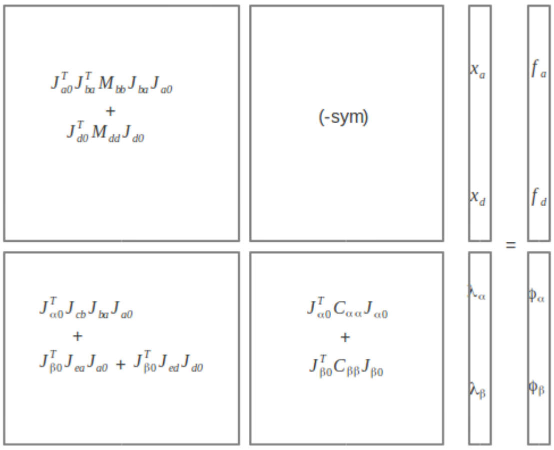

Figure 4: The assembly is performed by summing the global vectors and matrices obtained by shifting local vectors and matrices.

ForceFields may be handled as forces or as soft constraints, as explained in Section 1.2. The default behavior is to handle them as forces.

The BaseForceField functions are:

- `getComplianceMatrix` returns NULL if the ForceField is to be handled as a traditional force function, while it returns a pointer to a compliance matrix if it is to be handled as a constraint.
- `getStiffnessMatrix` is the complement of `getCompliantMatrix`. If one retuns NULL, then the other must return a pointer to a matrix.
- `writeConstraintValue` is used to write the constraint violation ϕ in Eq.3 when the ForceField is handled as a constraint.
- `getDampingRatio` is used in the constraint case, and corresponds to parameter d in Eq.3

The BaseMapping functions are:

- `getJs` returns a list of Jacobian matrices, one for each parent of the mapping. Typical mappings have only one parent, but MultiMappings have several ones.
- `getKs` returns a list of stiffness matrices, one for each parent. These correspond to geometric stiffness, i.e. change of forces due to mapping nonlinearity, like `addDJT` in the traditional API.