Real-Time FE Simulation for Large-Scale Problems Using Precondition-Based Contact Resolution and Isolated DOFs Constraints

Ziqiu Zeng, Stéphane Cotin, Hadrien Courtecuisse

Volume xx (200y), Number z, pp. 1–16

Real-time FE simulation for large-scale problems using precondition-based contact resolution and isolated DOFs constraints

Z. Zeng [1,2] S. Cotin [1,2] and H. Courtecuisse[1,2,3]

[1] ICube, University of Strasbourg, France
[2]MIMESIS research team, INRIA, France
[3]CNRS, France

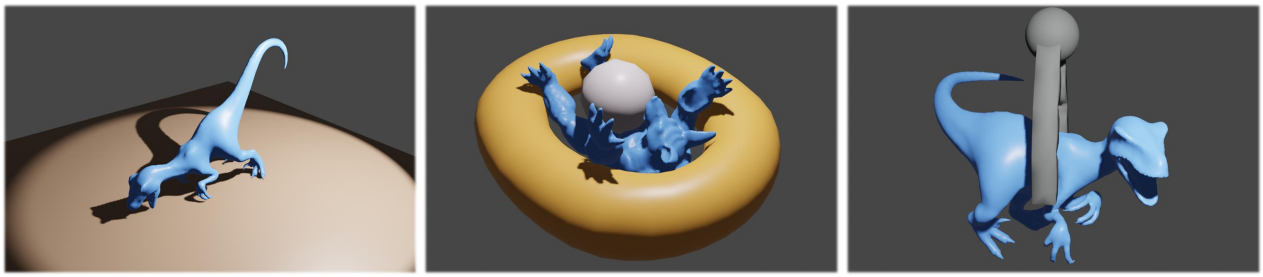

Figure 1: Dealing with complex interactions in multi-object systems usually requires distributing the contact forces through mechanical coupling. The constraint-based techniques can efficiently solve the contact problem in a constraint space but require assembling a compliance matrix, while this process is a schur-complement of the system matrix. When simulating the deformable models with detailed discretization, the large number of *mechanical DOFs* makes the computation of schur-complement very difficult with traditional solvers. In this paper, our new methods provide a fast process for the schur-complement while being capable of completing different challenges in various examples such as simple collision test (Left), complex interaction (Middle), and gripping task (Right).

Abstract

This paper presents a fast method to compute large-scale problems in real-time finite element simulations in the presence of contact and friction. The approach uses a precondition-based contact resolution that performs a Cholesky decomposition at low frequency. On exploiting the sparsity in assembled matrices, we propose a reduced and parallel computation scheme to address the expensive computation of the Schur-complement arisen by detailed mesh and accurate contact response. An efficient GPU-based solver is developed to parallelize the computation, making it possible to provide real-time simulations in the presence of coupled constraints for contact and friction response. In addition, the preconditioner is updated at low frequency, implying reuse of the factorized system. To benefit a further speedup, we propose a strategy to share the resolution information between consecutive time steps. We evaluate the performance of our method in different contact applications and compare it with typical approaches on CPU and GPU.

Keywords: Physics-based animation, Real-time Finite Element simulation, Constraint-based resolution, GPU-based parallelization.

1 Introduction

Interactive simulations received strong interest in many developing research fields such as surgical simulations and robotic simulations. One of the main requirements is to provide simulations of multi-object systems with complex interactions, such as contact and friction in real-time. Many works can be found in Computer Graphics to simulate deformable objects. As gold-standard methods in the study of physics-based simulations ([BSK20], [KE20]), advanced Finite Elements (FE) simulations are very popular in many applications due to their capacity to predict the complex behavior of deformable objects, providing relevant information for users in real-time. For this purpose, real-time FE simulations must meet the antagonistic requirements of accuracy and fast computation time at the same time.

The contact and friction response is always an essential issue in interactive simulations. A simple contact model using stiff penalty forces provides fast and approximate resolution but cannot guarantee to eliminate the interpenetration within a time step. In contrast, constraint-based approaches using Lagrange multipliers have proven to be robust and accurate for solving coupled contacts. However, for detailed deformable meshes, such approaches usually raise costly computations. To address this issue, parallelization strategies are usually employed to maintain the computational expense sufficiently low to account for user interactions. In this context, *general-purpose computing on graphics processing units* (GPGPU) has been widely studied as it provides access to massively parallel architecture with very low-cost memory transfers compared to distributed machines. Compared to the sequential computation on traditional central processing units (CPU), graphics processing units (GPU) operate at lower frequencies but process far more computation in parallel owing to massive parallel architecture, boosting performance for many compute-intensive tasks. Nowadays, more and more GPU-based approaches benefit significantly from the high performance of state-of-art GPUs with more than thousands of cores and ample memory space.

The contributions in the current paper focus on the boost of performance in contact simulations using the constraint-based technique. Using a precondition-based $\mathbf{LDL}^{\mathrm{T}}$ solver in constraint resolution, the method isolates the *DOFs* information in the constraint matrix. It performs a reduced computation in a constraint space named *isolated DOFs*. A GPU-based implementation is developed based on the *isolated DOFs* constraints, computing a parallel resolution. Moreover, we propose a strategy to share the computation results in constitutive time steps, furtherly reducing the computation costs.

The rest of this paper is organized as follows. After reviewing the related works Section 2, Section 3 presents the relevant model of contact and friction, as well as the resolution strategy used in this paper. Section 4 and Section 5 are dedicated to the new proposed approach to solve the contact equations, which is finally evaluated Section 6 in different simulation examples.

2 Related works

2.1 Physics-Based Models

In the context of interactive simulations, an important choice is the time integration scheme. Compared to the fast and simple explicit schemes [CTA*08], implicit schemes [Bar96] provide better control of the residual vector and hence that the external and internal forces are balanced at the end of the time steps. Although these advantages come at the cost of solving a set of linear equations at each time step, implicit integration schemes offer a reasonable trade-off between robustness, stability, convergence, and computation time, particularly when combined with a GPU implementation.

There is a considerable volume of works in the area of implicit simulation of deformable objects. Finite Element (FE) models provide a good understanding of the mechanisms involved in physiological or pathological cases, mainly because the soft-tissue behavior is directly explained through constitutive relations. However, reaching real-time performances is very challenging when dealing with deformable simulations. This problem is critical when using a complex FE model with a fine mesh discretization and nonlinear mechanical laws because of the high degrees of freedom and the integration of the nonlinear mechanics at each step. With the rapid growth of computational power, FE models have become compatible with real-time and interactivity. First limited to linear elastic models [BNC96], it was later extended to large displacements with the co-rotational formulation [Fel00]. [ACF12] proposes a parallel resolution on GPU for a co-rotational model without explicitly assembling the system matrix. In [KKB18] a novel approach is proposed for co-rotated FE simulations, splitting the deformation energy terms to a stretching part which can be solved efficiently by a pre-computed factorization and a volumetric part correction which is addressed approximately. This operator-splitting method shows considerably high performance. FE models are now used for the simulation of hyperelastic or viscoelastic materials in real-time [MHC*10]. However, although some validations of the behavior against real objects have been conducted, these models remain complex and expensive, and the simulation of realistic boundary conditions such as interactions between deformable objects is still an issue.

On the other hand, discrete methods such as Position-Based Dynamics (PBD) [MHHR07] provide simple, fast, and robust simulation. However, for these methods, the material properties can hardly be involved in their behaviors and the volume conservation cannot be guaranteed [BMO*14]. In order to bridge the gap between the PBD and the continuum mechanics, [BML*14] proposed projective dynamics that gives a good trade-off between the performance of PBD and the accuracy of continuum mechanics. This work is extended in many recent papers, such as [LBK17] for hyperelastic materials using a Quasi-Newton method to boost convergence, [WTB*21] for stable collision response between volumetric objects, and [LJBBD20] for frictional contacts. Despite being computationally efficient and robust, projective dynamics suffers from some drawbacks (e.g., the iteration number should be pre-adjusted to have correct behavior). Finally, meshless methods and Neural Networks are other strategies to model soft tissues in real-time. A detailed review of this topic goes far beyond the scope of this article, but a survey can be found in [ZZG18]. In many fields such as bio-mechanical simulations, FE methods still remain gold-standard approaches to simulate realistic behavior with real material parameters.

2.2 Contact and constraint-based resolution

In contact simulations, interactions in multi-objects systems usually cause discontinuity in the velocities. This can be handled either by the *event driven* scheme or by the *time stepping* scheme. The *event driven* scheme gives accurate results but is restricted with limited instantaneous contacts. In contrast, the *time stepping* scheme involves all contacts during a fixed time step [APS99].

In the *time stepping* scheme, the interactions between objects are detected in each time step (often at the beginning) by a collision detection process that defines potential constraint pairs for discretized systems. A typical collision detection method searches the closest distances between elements on the mesh surface. [AFC*10] presents a novel method to define constraints based on the volume

of interpenetration, using a GPU-based implementation to perform fast collision detection for complex meshes.

Once the contacts are defined, one can solve the dynamic system with different contact models that dominate the motion of interacting objects. As the earliest method, the penalty methods apply approximate forces on contact points depending on the interpenetration and a problem-dependent parameter named penalty value. A larger penalty value enforces a more limited interpenetration but also leads to a worse conditioned problem. Consequently, the methods only give an approximate solution and can hardly handle stable contacts. Being very simple to be implemented and fast, the penalty method is still very popular in many applications of fast simulations [BML*14], [LBK17], [KKB18].

On the other hand, the constraint-based methods using Lagrange multipliers ([Jea99], [Ren13]) solve the contact problem in a coupled way, addressing the limitation in the penalty methods. Such methods provide accurate and robust solutions in contact mechanics for large time steps, where interpenetration is entirely eliminated at the end of time steps. The Lagrange multipliers methods can be naturally used to impose displacement (bilateral constraints [GOM*06]), but interactions between multiple objects usually lead to define unilateral constraints. To address the problem, the methods are commonly formulated as a linear complementarity problem (LCP).

In the constraint resolution, different numerical methods can be used to address the LCPs in physics-based animations [Erl13]. Direct methods such as pivoting methods give exact resolution, but they are not computationally efficient. In contrast, iterative methods have been more widely applied in large-scale simulations, especially for those who require to perform real-time computations. Being simple to implement, projected Gauss-Seidel (PGS) ([DDKA06], [CA09], [MMC16]) is able to handle the friction response with the Coulomb's friction cone combined in the LCP formulation. However, the algorithm is not efficient for ill-conditioned problems due to the slow convergence. In order to solve this limitation, [MEM*19] proposes using a Newton method to solve the nonsmooth functions that are reformulated from complementarity problems. A complementarity preconditioner is used to boost the convergence of a conjugate residual (CR) algorithm for the constraint resolution. Although being very robust and efficient for interactive simulations, the method is introduced in a context with fewer degrees of freedom (*DOFs*) than constraints.

In physics-based animations with constrained dynamics, one efficient solution is to formulate the Schur-complement of the augmented system, allowing this way to solve the problem in the *constraint space* that usually has a much smaller size than the augmented system. The Schur-complement results in a compliance matrix (also called *Delasus operator*) in the *constraint space*. As discussed in [AE21], the computation of Schur-complement tends to be costly when dealing with soft-body since it implies to solve a linear system with multiple right-hand sides. In such system, the discretization of FE models determines the problem size (*mechanical DOFs*) and the constraints determines the right-hand sides. We would like to note here that although the geometrically motivated methods (PBD and projective dynamics) also process constraint resolution, they meet different challenges from the FE methods in the system resolution. Building the *Delasus operator* is not necessary for projective dynamics as the internal and external constraints are solved locally in each element then coupled in a global solver. But in FE simulations, the *Delasus operator* is important to couple contact forces in the resolution and enables stable simulations with complex interactions.

Many methods have been proposed to efficiently process the Schur-complement in interactive FE simulations. [SDCG08] presents a compliance warping to pre-compute a factorized system in initialization and to apply correction in online simulation according to the co-rotational formulation. Unfortunately, the method is restricted with small deformation. [SG06] and [PSLG14] present respectively a fast factorization method and a fast Schur-complement computation using an augmented factorization method. Both the two methods are parallelized in CPU threads and integrated into *Pardiso solver project*, giving a fast CPU-based resolution for linear systems and Schur-complement. Updating Cholesky factor is suitable for FE simulations since the deformations are usually limited in consecutive time steps. This technique becomes very efficient by reusing factorization on submeshes [HA18] and is extended to dimension addition cases (e.g., mesh cutting) in [HSH20]. However, even with highly optimized methods, factorizing large-scale systems remains highly expensive and makes it hard to be applied in real-time simulations. These CPU-based methods suffer from a very costly factorization for detailed meshes, making it hard to perform real-time simulations. Since factorization remains a critical obstacle in real-time applications, [CADC10] proposed an asynchronous method to compute a preconditioner for iterative method, moving the factorization of the system into a parallel thread. This work is extended to the contact simulation in [CAK*14] to compute an approximate solution of Schur-complement, using a highly parallelized solver on GPU. The method enables real-time simulations with up to 2000 nodes with 300 constraints but the computation of the Schur-complement is dominant in the time integration.

Based on the work in [CAK*14], the current paper aims to improve the performance of the Schur-complement in the simulations where contact and friction constraints are handled with precondition-based technique. Our work enables real-time simulation for large-scale problems with 10000 to 20000 nodes. We propose to reformulate the Schur-complement by isolating the *DOFs* information in the constraint matrix at the right-hand side. The transformed contact matrix and the sparse system matrix allow us to perform a significantly reduced computation, fitting into GPU-based parallel computation. To benefit from the nature of the asynchronous scheme, we also propose a strategy to reuse the computation results in consecutive time steps. All these methods finally result in a high-speed Schur-complement computation.

3 Background

The current method is based on a general background for deformable simulations using implicit integration, where a constraint-based approach solves the coupled contacts between multiple objects.

3.1 Implicit time integration

For any time step t, the general way to describe the physical behavior of a deformable objective problem can be expressed using Newton's second law:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{p} - \mathcal{F}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{c} \tag{1}$$

where $\mathbf{M}$ is the mass matrix, $\ddot{\mathbf{q}}$ the vector of the derivative of the velocity, $\mathbf{p}$ the external forces, and $\mathcal{F}(\mathbf{q},\dot{\mathbf{q}})$ the function represents the internal forces. $\mathbf{c}$ represents the contact and friction forces contribution. A backward Euler method is used to integrate the time step. The implicit scheme can be expressed as follows, where h is the length of time interval $[t, t+h]$:

$$\begin{aligned} \dot{\mathbf{q}}_{t+h} &= \dot{\mathbf{q}}_t + h\ddot{\mathbf{q}}_{t+h} \\ \mathbf{q}_{t+h} &= \mathbf{q}_t + h\dot{\mathbf{q}}_{t+h} \end{aligned} \tag{2}$$

As $\mathcal{F}(\mathbf{q},\dot{\mathbf{q}})$ is a non-linear function, a first-order Taylor expansion is performed to linearize the problem [Bar96]. This linearization corresponds to the first iteration of the Newton-Rapson method. The incomplete approximation may cause numerical errors of the dynamic behavior, but they lean towards decreasing at equilibrium. The internal forces are expanding as following:

$$\mathcal{F}(\mathbf{q}_{t+h},\dot{\mathbf{q}}_{t+h}) = \mathbf{f}_t + \frac{\partial\mathcal{F}(\mathbf{q},\dot{\mathbf{q}})}{\partial\mathbf{q}} h\dot{\mathbf{q}}_{t+h} + \frac{\partial\mathcal{F}(\mathbf{q},\dot{\mathbf{q}})}{\partial\dot{\mathbf{q}}} h\ddot{\mathbf{q}}_{t+h} \tag{3}$$

with $\mathbf{f}_t = \mathcal{F}(\mathbf{q}_t,\dot{\mathbf{q}}_t)$.

During a time integration, the force function is considered as constant and the partial derivative terms could be expressed as matrices: $\frac{\partial\mathcal{F}}{\partial\dot{\mathbf{q}}}$ at $(\mathbf{q}_t,\dot{\mathbf{q}}_t)$ is the damping matrix $\mathbf{B}$ and $\frac{\partial\mathcal{F}}{\partial\mathbf{q}}$ at $(\mathbf{q}_t,\dot{\mathbf{q}}_t)$ is the stiffness matrix $\mathbf{K}$. By integrating Equations (1), (2) and (3) we obtain the dynamic equation:

$$\underbrace{\left[\mathbf{M} + h\mathbf{B} + h^2\mathbf{K}\right]}_{\mathbf{A}} \underbrace{\Delta\dot{\mathbf{q}}_{t+h}}_{\mathbf{x}} = \underbrace{(h\mathbf{p}_t - h\mathbf{f}_t) - h^2\mathbf{K}\dot{\mathbf{q}}_t}_{\mathbf{b}} + h\mathbf{c} \tag{4}$$

with $\Delta\dot{\mathbf{q}} = h\ddot{\mathbf{q}}$.

3.2 Contact and friction

The response of contact and friction can be defined by Signorini's law and Coulomb's friction law, respectively. Signorini's law presents the complementarity relationship along the constraint direction n for each potential contact:

$$0 \le \delta_n \perp \lambda_n \ge 0 \tag{5}$$

where the δ_n is the distance of the contact points between the interacting objects and the λ_n is the constraint force along the constraint direction n. The model describes a non-interpenetration physical behavior where the constraint force is eliminated if the points are not in contact.

The friction response is completed by Coulomb's friction law that describes the behavior in tangent contact space:

$$\begin{aligned} \dot{\delta}_T = 0 &\Rightarrow ||\mathbf{f}_T|| < \mu||\mathbf{f}_n|| \quad (stick) \\ \dot{\delta}_T \neq 0 &\Rightarrow \mathbf{f}_T = -\mu||\mathbf{f}_n||\frac{\dot{\delta}_T}{||\dot{\delta}_T||} = -\mu||\mathbf{f}_n||T \quad (slip) \end{aligned} \tag{6}$$

where μ is the coefficient of friction and T is the motion direction in the tangential space. In a 3D problem with dynamic friction, each potential contact is along with two tangential directions. The model of Coulomb's friction law adds a non-linearity into the complementarity state in Signorini's law.

In order to associate the contact points (usually defined on the surface of the object with the Degrees of Freedom) at the beginning of the time step, a mapping function $\mathcal{J}(\mathbf{q})$ can be built to link the contacts and mechanical motion space. For two colliding objects 1 and 2, we have the interpenetration on contact pairs defined in collision detection:

$$\delta(t) = \mathcal{J}_1(\mathbf{q}_1(t)) - \mathcal{J}_2(\mathbf{q}_2(t)) \tag{7}$$

With the implicit integration (Equation (2)), the mapping functions are linearized with a first-order Taylor expansion:

$$\mathcal{J}(\mathbf{q}_{t+h}) \approx \mathcal{J}(\mathbf{q}_t) + h\frac{\partial\mathcal{J}(\mathbf{q})}{\partial\mathbf{q}}\dot{\mathbf{q}}_t + h\frac{\partial\mathcal{J}(\mathbf{q})}{\partial\mathbf{q}}\Delta\dot{\mathbf{q}} \tag{8}$$

Once the collision information is available, all the constraint equations are then evaluated along with the collision information that is assumed constant for the rest of the time step (see Figure 2). This leads to several simplifications: First of all, $\mathbf{H} \approx \frac{\partial\mathcal{J}(\mathbf{q})}{\partial\mathbf{q}}$ at t the beginning of each time step , known as the *constraint Jacobian*, can be defined, providing the constraint directions (blue and orange arrows in Figure 2). The dimension of $\mathbf{H}$ is $c \times n$, where c is the number of discretized constraints, and n is the number of DOFs. Similarly, the discretized violation of the constraints $\mathcal{J}(\mathbf{q}_t) \approx \mathbf{H}\mathbf{q}_t$ is evaluated along the same constraint directions. With these simplification and replacing (7) in (8) the violation of the constraint at the end of the step can be rewritten as:

$$\delta^{t+h} = \delta^t + h\mathbf{H}_1\Delta\dot{\mathbf{q}}_1 - h\mathbf{H}_2\Delta\dot{\mathbf{q}}_2 \tag{9}$$

with $\delta^t = (\mathbf{H}_1\mathbf{q}_1^t + h\mathbf{H}_1\dot{\mathbf{q}}_1^t) - (\mathbf{H}_2\mathbf{q}_2^t + h\mathbf{H}_2\dot{\mathbf{q}}_2^t)$ the interpenetration at the beginning of the time step. On the other hand, the contact Jacobian transforms the forces λ in the contact space to the contact forces $\mathbf{c}$ in the motion space:

$$\mathbf{c}_1 = {\mathbf{H}_1}^{\mathrm{T}}\lambda \quad \mathbf{c}_2 = -{\mathbf{H}_2}^{\mathrm{T}}\lambda \tag{10}$$

where λ is applied in two opposite ways for the two objects. We note that each constraint group involves contact constraint n and frictional constraint T. Consequently, $\mathbf{H}_n$ and $\mathbf{H}_T$ are grouped in $\mathbf{H}$; δ_n and δ_T are grouped in δ; λ_n and λ_T are grouped in λ. Combining Equations (4), (9) and (10), a Karush-Kuhn-Tucker (KKT) system is assembled as follows:

$$\begin{cases} \mathbf{A}_1\mathbf{x}_1 - h{\mathbf{H}_1}^{\mathrm{T}}\lambda = \mathbf{b}_1 & \text{(11a)} \\ \mathbf{A}_2\mathbf{x}_2 + h{\mathbf{H}_2}^{\mathrm{T}}\lambda = \mathbf{b}_2 & \text{(11b)} \\ h\mathbf{H}_1\mathbf{x}_1 - h\mathbf{H}_2\mathbf{x}_2 + \delta^t = \delta^{t+h} & \text{(11c)} \end{cases}$$

In addition, δ^{t+h} and λ at the end of time steps should satisfy the complementarity according to Signorini's law (Equation (5)) and Coulomb's law (Equation (6)).

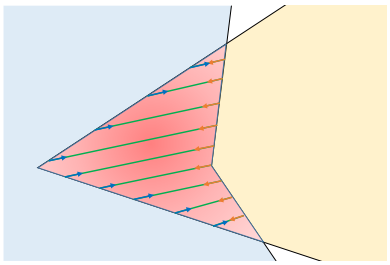

Figure 2: In practice, the evaluation and the linearization of the constraints equations are difficult. To simplify the solving process, collision detection is performed providing a set of discretized constraints between both objects (gree lines). The number of discretized constraints usually depends on the resolution of the collision mesh and/or the collision detection method itself (for instance, by filtering constraints afterward).

3.3 Constraint resolution

By eliminating the unknowns $\mathbf{x}_1$ and $\mathbf{x}_2$ in Equation (11c), we have:

$$\begin{aligned}\delta^{t+h} = \delta^t + h[\mathbf{H}_1 \underbrace{\mathbf{A}_1^{-1}\mathbf{b}_1}_{\mathbf{x}_1^{\text{free}}} - \mathbf{H}_2 \underbrace{\mathbf{A}_2^{-1}\mathbf{b}_2}_{\mathbf{x}_2^{\text{free}}}] \\ + h^2 \underbrace{[\mathbf{H}_1\mathbf{A}_1^{-1}\mathbf{H}_1{}^{\mathrm{T}} + \mathbf{H}_2\mathbf{A}_2^{-1}\mathbf{H}_2{}^{\mathrm{T}}]}_{\mathbf{W}}\lambda\end{aligned} \tag{12}$$

where $\mathbf{W}$ is the Schur-complement (also called *compliance matrix* or *delassus operator* in constrained dynamics) that project the system matrix $\mathbf{A}$ in the motion space to the contact space. We process a first step called *free motion* that computes the temporary motion $\mathbf{x}_{\text{free}}$, which mathematically corresponds to physics dynamics without considering the constraints of contact and friction:

$$\mathbf{x}^{\text{free}} = \mathbf{A}^{-1}\mathbf{b} \tag{13}$$

With implicit integration we may note that:

$$\delta^t + h[\mathbf{H}_1\mathbf{x}_1^{\text{free}} - \mathbf{H}_2\mathbf{x}_2^{\text{free}}] = \underbrace{[\mathbf{H}_1\mathbf{q}_1^{\text{free}} - \mathbf{H}_2\mathbf{q}_2^{\text{free}}]}_{\delta^{\text{free}}} \tag{14}$$

where the *free interpenetration* δ^{free} is computed directly with the *free position* $\mathbf{q}^{\text{free}}$ (integrated with $\mathbf{x}^{\text{free}}$). Once $\mathbf{W}$ and δ^{free} are assembled, a complementarity system with constraint dimension can be formulated:

$$\delta^{t+h} = \delta^{\text{free}} + h^2\mathbf{W}\lambda \tag{15}$$

The unknown λ is solved by a projected Gauss-Seidel algorithm [DDKA06] during the successive iterations (i):

$$\delta_\alpha - h^2\mathbf{W}_{\alpha\alpha}\lambda_\alpha^{(i)} = \sum_{\beta=1}^{\alpha-1} h^2\mathbf{W}_{\alpha\beta}\lambda_\beta^{(i)} + \sum_{\beta=\alpha+1}^{c} h^2\mathbf{W}_{\alpha\beta}\lambda_\beta^{(i-1)} + \delta_\alpha^{\text{free}} \tag{16}$$

where $\mathbf{W}_{\alpha\beta}$ is a local matrix of $\mathbf{W}$ that couples the contact α and β. The complementarity problem for each contact group α is solved in the local solution while following Signorini's law for unilateral contact response and Coulomb's law for frictional response. As a Gauss-Seidel-like algorithm, after solving each contact α, the correction on λ_α is immediately propagated to all the following contacts. In this way, the contact forces are coupled by the compliance matrix during the constraint resolution.

Once the λ is solved, a *corrective motion* is processed to integrate the final motion $\mathbf{x}_{t+h}$:

$$\begin{aligned}\mathbf{x}_1^{t+h} &= \mathbf{x}_1^{\text{free}} + h\mathbf{A}_1^{-1}\mathbf{H}_1{}^{\mathrm{T}}\lambda \\ \mathbf{x}_2^{t+h} &= \mathbf{x}_2^{\text{free}} - h\mathbf{A}_2^{-1}\mathbf{H}_2{}^{\mathrm{T}}\lambda\end{aligned} \tag{17}$$

3.4 Factorization

A primary challenge in real-time simulation is computing the Schur-complement in Equation (12). This involves the large system matrix $\mathbf{A}$ with a dimension size corresponding to the number of *mechanical DOFs*. Inverting such large a system is highly expensive, especially with multiple right-hand sides (RHS) in the contact Jacobian $\mathbf{H}$ with a size corresponding to the number of constraints. Processing an exact factorization for $\mathbf{A}$ in each time step can be used in small scale problems but becomes prohibitive when dealing with detailed soft bodies. To address the problem, many works are dedicated to find a good approximation of the factorized system, such as incomplete factorization ([SG06] implemented in *Pardiso solver project*), updating Cholesky factor ([HA18], [HSH20]) and asynchronous preconditioning strategy ([CADC10]). In the current paper, our main contribution relies on an hypothesis that the solver is able to obtain a good approximation of factorization. Our work is based on the asynchronous preconditioning strategy that releases the computing expense in the main simulation loop. Let $\mathbf{A}_t$ be the matrix built in a specific time t. Following [CADC10], a preconditioner $\mathbf{P}$ can be built from an asynchronous $\mathbf{LDL}^{\mathrm{T}}$ factorization:

$$\mathbf{P} = \mathbf{A}_t^{-1} = (\mathbf{LDL}^{\mathrm{T}})^{-1} \tag{18}$$

where $\mathbf{D}$ is a diagonal matrix and $\mathbf{L}$ is a sparse lower triangular matrix. The factorized matrices will be available after the factorization is done, usually several time steps after time t, and used as a preconditioner with the assumption that $\mathbf{P}$ remains a relatively good approximation to the inverse of the current matrix $\mathbf{A}_{t+n}$. According to [CADC10], the method is very efficient because the $\mathbf{LDL}^{\mathrm{T}}$ factorization requires only few simulation steps (usually $n < 5$) to update. The nested dissection algorithm is used to reduce the filling of the matrix pattern, recursively dividing the mesh into two parts with the nearly same number of vertices while keeping the divider part at a small scale [Geo73]. Consequently, $\mathbf{L}$ is reordered and partitioned into sub-domains with the indices given by the nested dissection algorithm.

In *free motion*, the resolution in Equation (13) is solved with a preconditioned Conjugate Gradient (CG) algorithm. In each CG iteration, the application of preconditioner $\mathbf{P}$ implies to solve sparse triangular systems (STS) with a vector at right-hand side:

$$\mathbf{z} = \mathbf{P}\mathbf{r} = \mathbf{L}^{-\mathrm{T}}\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{r} \tag{19}$$

where $\mathbf{r}$ is the vector of residual and $\mathbf{z}$ the residual applied with the preconditioner in CG. Usually, the preconditioner is very efficient because the preconditioner $\mathbf{P}$ remains a good approximation to the inverse of the current system matrix. In practice, only 2 to 5 preconditioned CG iterations are necessary to converge. However, this

step is difficult to parallelize on a GPU due to numerous data dependencies. As the triangular matrices are sparse, the solution can easily be implemented with a *Gauss elimination* on the CPU.

In *constraint resolution*, [CAK*14] presents a precondition-based approach for contact problems. The asynchronous preconditioner $\mathbf{P}$ is reused as a close approximation of the inverse of the factorization of the current system matrix $\mathbf{A}$. As a result, the compliance matrix is approximately built as:

$$\mathbf{W} = \sum \mathbf{H}\mathbf{A}^{-1}\mathbf{H}^{\mathrm{T}} \approx \sum \mathbf{H}\mathbf{P}\mathbf{H}^{\mathrm{T}} = \sum \mathbf{H}(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1}\mathbf{H}^{\mathrm{T}} \tag{20}$$

with the summation of contribution of all the contacting objects.

ALGORITHM 1: Approximate computation of the Schur-complement with the system factorized in asynchronous thread

1 $\mathbf{S} = \mathbf{L}^{-1}\mathbf{H}^{\mathrm{T}}$
2 $\mathbf{W} = \sum \mathbf{S}^{\mathrm{T}}\mathbf{D}^{-1}\mathbf{S}$

The contribution accumulation of each object on the compliance is processed in column independently with Algorithm 1: the first step is the resolution of multiple STS, which is usually the most expensive task and tends to be very costly while processed sequentially on CPU. [CAK*14] proposes an efficient GPU-based solution for multiple STS using a two-level parallelization: each right-hand side in $\mathbf{H}^{\mathrm{T}}$ is computed in parallel multiprocessors. Since the resolution of each triangular system involves numbers of dependencies, the left-hand side $\mathbf{L}$ is fully processed and the result $\mathbf{S}$ is stored in a dense matrix. The second step consists of matrix-matrix multiplications and can be efficiently processed on GPU. Despite the fact that the method already provides a significant speedup compared to the sequential computation on CPU, the building of contact compliance matrix still remains the most expensive process, taking a ratio of more than 70% in time integration. Accelerating this process is an important issue that will be addressed in the following sections 4 and 5. It has been proved in [CAK*14] that the asynchronous preconditioning strategy provides a good approximation to the actual $\mathbf{W}$ (assembled with $\mathbf{A}^{-1}$), allowing to efficiently couple the contact forces in constraint resolution.

3.5 Compliance assembly

An iterative method like Equation (16) can be performed either with explicitly assembled $\mathbf{W}$, or with an unbuilt form since processing each iteration only requires an operation of matrix-vector multiplication:

$$\mathbf{z} = \mathbf{W}\mathbf{y} = \sum (\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^{\mathrm{T}})\mathbf{y} = \sum \mathbf{H}\mathbf{A}^{-1}(\mathbf{H}^{\mathrm{T}}\mathbf{y}) \tag{21}$$

where $\mathbf{H}^{\mathrm{T}}\mathbf{y}$ leads to a vector, avoiding to apply the multiple right hand sides ($\mathbf{A}^{-1}\mathbf{H}^{\mathrm{T}}$). However, the possibility to not build explicitly $\mathbf{W}$ is popular as long as $\mathbf{A}^{-1}$ is sparse and easy to be computed. This is the case for instance for rigid objects with a diagonal matrix, or beam elements with a block-tridiagonal matrix where Thomas algorithm used in [XL18] can be used to invert the system. In these cases, the unbuilt version is known to be faster. However, this assumption does not apply to FE models with large unstructured matrices.

Following the context in Section 3.4, as long as an approximation of factorization can be fast obtained, the matrix-multiplication in unbuilt scheme (Equation (21)) actually requires solving a $\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}$ system:

$$\mathbf{z} = \sum \mathbf{H}\mathbf{A}^{-1}(\mathbf{H}^{\mathrm{T}}\mathbf{y}) \approx \sum \mathbf{H}(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1}(\mathbf{H}^{\mathrm{T}}\mathbf{y}) \tag{22}$$

Algorithm 2 implements the resolution of the $\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}$ system in Equation (22).

ALGORITHM 2: Unbuilt scheme: implementation of the STS resolution (Equation 22) in iterations of relaxation methods. $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are temporary vectors

1 $\mathbf{v}_1 = \mathbf{L}^{-1}(\mathbf{H}^{\mathrm{T}}\mathbf{y})$
2 $\mathbf{v}_2 = \mathbf{D}^{-1}\mathbf{v}_1$
3 $\mathbf{v}_3 = (\mathbf{L}^{\mathrm{T}})^{-1}\mathbf{v}_2$
4 $\mathbf{z} = \mathbf{H}\mathbf{v}_3$

This leads to extra cost in each iteration. When dealing with large scale problems, the $\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}$ resolution becomes costly. Such an addition operation in iterations will result in enormous extra computation cost. In Section 6.2.9 we will compare the cost of unbuilt scheme and assembling $\mathbf{W}$ in different cases.

4 Reformulating the Schur-complement

4.1 Exploit the sparsity of constraint Jacobian matrix

The *constraint Jacobian* matrix $\mathbf{H}$ describes how the contact constraints are applied to mechanical degrees of freedom (*DOFs*). Since the contacts are often limited in local areas, the size of constraint dimension $\boldsymbol{c}$ is usually far more smaller than the dimension of *mechanical DOFs* $\boldsymbol{n}$. Coupled with the fact that each constraint is linked to limited *mechanical DOFs*, $\mathbf{H}$ is very sparse in many cases. Based on this observation, we propose eliminating empty columns in $\mathbf{H}$ and formulating a "compressed" matrix $\hat{\mathbf{H}}$. The relation between the two matrices can be actually expressed by a matrix-matrix multiplication (see also Figure 3):

$$\mathbf{H} = \hat{\mathbf{H}}\bar{\mathbf{I}} \tag{23}$$

where $\bar{\mathbf{I}}$ is a "*partial identity*" matrix that is formulated with the indices of *non-zero* columns in $\mathbf{H}$. By formulating $\bar{\mathbf{I}}$ we actually isolate the indices information of *mechanical DOFs* that receive contributions from the constraints from $\mathbf{H}$. This generates a new dimension "*isolated DOFs*" (also abbreviated to "*isodof*"). The matrix $\bar{\mathbf{I}}$ is also called as "*isodof Jacobian*". Compared to the constraint dimension $\boldsymbol{c}$, the *isodof* dimension $\boldsymbol{k}$ may be larger or smaller. Now, with Equation (23), computing the Schur-complement in Equation (20) is reformulated as:

$$\mathbf{W} = \sum \hat{\mathbf{H}} \underbrace{\bar{\mathbf{I}}\mathbf{A}^{-1}\bar{\mathbf{I}}^{\mathrm{T}}}_{\bar{\mathbf{W}}} \hat{\mathbf{H}}^{\mathrm{T}} \tag{24}$$

where $\mathbf{W}$ is built with $\hat{\mathbf{H}}$ and $\bar{\mathbf{W}}$ that is called *"isodof compliance matrix"*.

Following Equation (20), we propose to compute with the asynchronous preconditioner ($\mathbf{A} \approx \mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}$) in Algorithm 3. Step 1 is processed by analyzing the sparse pattern of $\mathbf{H}$; Step 2 involves solving multiple STS; Step 3 and Step 4 consist of matrix-matrix

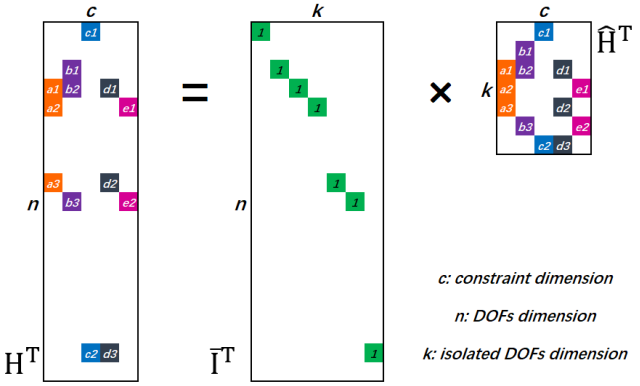

Figure 3: The *constraint Jacobian* $\mathbf{H}$ is usually very sparse and contains many empty columns. By eliminating these empty columns, we formulate a "compressed" matrix $\hat{\mathbf{H}}$ and a "*partial identity*" matrix $\bar{\mathbf{I}}$ that contains one element in each row that corresponds to a *non-zero* column in $\mathbf{H}$. The relation between matrices can be expressed by a matrix-matrix multiplication: $\mathbf{H} = \hat{\mathbf{H}}\bar{\mathbf{I}}$ (transposed format illustrated in the figure)

multiplications that can be efficiently computed on GPU. We underline that Algorithm 3 computes the same result as in Algorithm 1 (i.e., the resolution proposed in [CAK*14]). The only approximation comes from using a delayed system that is factorized in an asynchronous thread.

ALGORITHM 3: Approximate computation of the Schur-complement with the system factorized in asynchronous thread

1 Build $\bar{\mathbf{I}}$, $\hat{\mathbf{H}}$ from $\mathbf{H}$
2 $\bar{\mathbf{S}} = \mathbf{L}^{-1}\bar{\mathbf{I}}^{\mathrm{T}}$
3 $\bar{\mathbf{W}} = \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}$
4 $\mathbf{W} = \sum \hat{\mathbf{H}}\bar{\mathbf{W}}\hat{\mathbf{H}}^{\mathrm{T}}$

4.2 STS resolution strategy

The step 2 of Algorithm 3 remains a difficult task to be parallelized on the GPU due to the data dependencies in triangular systems. However, the *isodof scheme* leads to a special resolution. Each right-hand side is no more the combination of various values (in $\mathbf{H}^{\mathrm{T}}$) but only contains one element with value "1" on a specific column (in $\bar{\mathbf{I}}^{\mathrm{T}}$). An important consequence is related to the fact that, for each column of $\bar{\mathbf{I}}^{\mathrm{T}}$, only a subset value needs to be computed, leading to a sparse resolution of the triangular system (see Figure 4). Therefore the density of $\bar{\mathbf{S}}$ is significantly reduced compared to $\mathbf{S}$ (i.e., dealing with $\mathbf{H}^{\mathrm{T}}$). In addition, the dependencies can formally be expressed with an elimination tree. Given that the matrix pattern of $\mathbf{L}$ only depends on the mesh topology, the elimination tree can be pre-computed, and the sparse matrix storage of the result is predictable for every column index, as long as the topology is not changed.

While processing the STS resolution, it can be either processed in the "row-major" or the "column-major" (see Figure 5). When assuming the solutions as dense vectors (as in [CAK*14]), processing the "row-major" is more efficient because it does not cause data writing conflict in parallel resolution. However, as the structure of

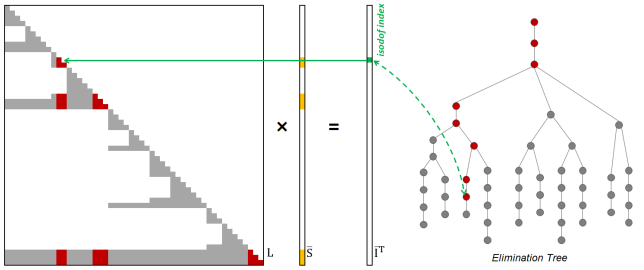

Figure 4: Resolution of one RHS in $\mathbf{L}\bar{\mathbf{S}} = \bar{\mathbf{I}}^{\mathrm{T}}$ (the elimination tree (top-right) helps visualize the structure of dependency in $\mathbf{L}$): Each right-hand side contains one element with value "1" on a column index i, locating on a branch on the elimination tree. During the resolution, only this branch with index i and its parent branches need to be processed (red nodes on the elimination tree). Reflected on the matrix pattern, only the red elements in $\mathbf{L}$ need to be processed. This sparse resolution is very efficient compared to process the full matrix $\mathbf{L}$.

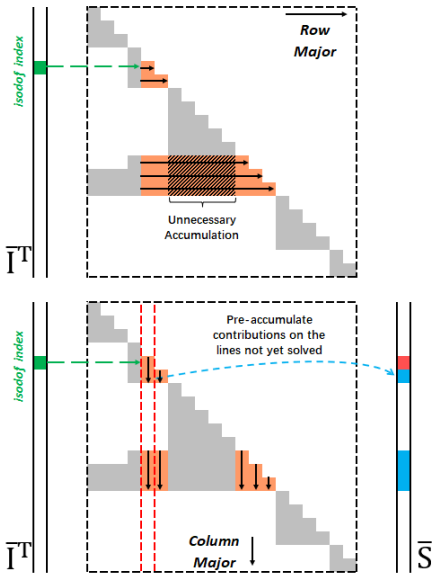

Figure 5: The STS can either be solved in the "row-major" (Top) or in the "column-major" (Bottom). In the "row-major", using the CSR format requires processing data in $\mathbf{L}$ continuously on each row, leading to many unnecessary accumulations to the result with zero contributions. Using the "column-major" scheme with CSC format can naturally address this problem. When dealing with a column i, the result on the line i is fully solved (red), but the rest lines remain unsolved (blue) and require to pre-accumulate the contributions on the lines: $res[j] = res[j] - res[i] * \mathbf{L}_{i,j}$.

$\bar{\mathbf{S}}$ is very sparse, it accumulates zero contributions, causing a large amount of unnecessary computation cost. Although the "column-major" requires pre-accumulating the data on the positions where the result is not yet solved (see below), it can naturally avoid this unnecessary accumulation.

4.3 GPU-based implementation

To implement Algorithm 3, Step 1 requires to build $\bar{\mathbf{I}}$ and $\hat{\mathbf{H}}$. According to the illustration in Figure 3, $\bar{\mathbf{I}}$ can be stored in a vector that contains the *non-zero* columns in $\mathbf{H}$. As a "compressed reformulation", $\hat{\mathbf{H}}$ have the same data sequence of $\mathbf{H}$. This means that, when stored in Compressed Row Sparse (CSR) format, they have the same vectors of *row index* and *values*, while the vector of *column index* of $\hat{\mathbf{H}}$ is built as "compressed indices".

The GPU-based implementation of STS resolution in Step 2 is inspired from the block-row parallelization strategy in [CAK*14]. Each right-hand side of $\bar{\mathbf{S}}$ is assigned to an independent multiprocessor since the multiple RHS are independent of each other (see Algorithm 4).

ALGORITHM 4: Algorithm to address Sparse Triangular System with multiple right-hand sides of *isolated DOFs* using column-major

```
Result: columns res in S̄ solved by multiprocessors in parallel
Initialization: compute sub-domains to be processed and their
  indices in offline : subDomain, startInd, endInd ;
i = 0 ;
bx = isodof ;          // First sub-domain begins with
  isodof index
end = endIndex[subDomain[i]] ;
while i < subD.size do
    while bx < end do
        copy_into_shared_memory(diag) ;
        local_synchronization ;
        solve_bloc_diagonal(diag,res) ;      // see [CAK*14]
        local_synchronization ;
        pre_accumulate_contributions(res) ;      // Figure 5
        local_synchronization ;
        bx = bx + t ;                        // next t columns
    end
    i = i + 1 ;
    bx = startInd[subDomain[i]] ;
    end = endInd[subDomain[i]] ;       // next sub-domain
end
```

To perform the sparse resolution in Figure 4 for each right-hand side, we pre-compute offline the pattern to be processed for every index. Each pattern is composed of sub-domains, with their indices given by the reordering algorithm. As illustrated in Figure 6, within each sub-domain, we process t columns simultaneously by using a group of $t \times t$ threads. For each t columns, the block diagonal (*diag*) is firstly processed as a dense problem; then, the off-diagonal data is accumulated into the result. The parallel accumulations may cause data writing conflicts, which can be handled with the *atomic* function.

As $\bar{\mathbf{S}}$ and $\hat{\mathbf{H}}$ are stored in the sparse format, once the STS resolution is processed, Step 3 and 4 in Algorithm 3 can be implemented with the following operations:

$$\mathbf{X}_1 = \mathbf{D}^{-1}\bar{\mathbf{S}} \quad (Diag - Sparse) \tag{25a}$$

$$\bar{\mathbf{W}} = \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{X}_1 \quad (Sparse - Sparse) \tag{25b}$$

$$\mathbf{X}_2 = \bar{\mathbf{W}}\hat{\mathbf{H}}^{\mathrm{T}} \quad (Sparse - Dense) \tag{25c}$$

$$\mathbf{W} = \hat{\mathbf{H}}\mathbf{X}_2 \quad (Sparse - Dense) \tag{25d}$$

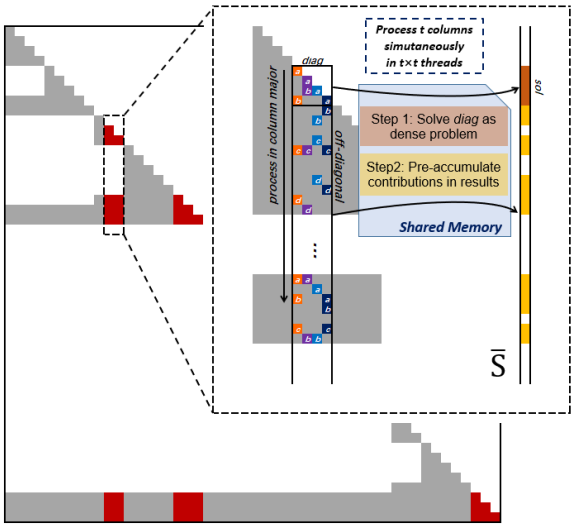

Figure 6: The STS with *isodof Jacobian* is solved in column-major. Each right-hand side is assigned to an independent multiprocessor, and for each one, $t \times t$ threads (represented by different colors) are used to process the resolution simultaneously. The *off-diagonal* contributions are pre-accumulated to the results in parallel threads.

where Operation (25a) is a diagonal matrix-sparse matrix multiplication, resulting in a temporary matrix $\mathbf{X}_1$ with sparse format. Operation (25b) is a sparse matrix-sparse matrix multiplication, where the sparse structure of $\bar{\mathbf{S}}^{\mathrm{T}}$ and $\mathbf{X}_1$ can be pre-computed. Operations(25c), and (25d) are sparse matrix-dense matrix multiplications (***SpMM***) with normal or transposed format, resulting in matrices with dense format. As we process the sparse computation, an important consequence is that the operations in Step 2 and 3 in Algorithm 3 are independent of the *mechanical DOF* dimension $\boldsymbol{n}$. As a result, the *isodof* method can be very efficient even with highly detailed mesh.

5 Reuse of solutions in consecutive time steps

In this section, by exploiting the *isodof scheme* and the asynchronous preconditioning scheme, we present a "*reuse isodof scheme*" to benefit a further speedup.

Using the asynchronous preconditioning scheme implies that the solvers in the main simulation loop keep using the factorized system ($\mathbf{LDL}^{\mathrm{T}}$) until a new factorization is done. In this case, we have the STS resolution with the *isodof scheme* in two consecutive time steps t and $t+i$:

$$\begin{aligned} \bar{\mathbf{S}}_t &= \mathbf{L}^{-1}\bar{\mathbf{I}}_t{}^{\mathrm{T}} \\ \bar{\mathbf{S}}_{t+i} &= \mathbf{L}^{-1}\bar{\mathbf{I}}_{t+i}{}^{\mathrm{T}} \end{aligned} \tag{26}$$

where $t+i$ represents the consecutive time steps while the factorized system is not yet updated.

Moreover, the *mechanical DOFs* that are impacted by the contact constraints directly depend on the local mesh area where contact occurs. Figure 7 reveals that, in real-time simulations, consecutive time steps usually share a part of contact area, so as the *isodofs* impacted. Reflected on the matrices, the *isodof Jacobian* in consecutive time steps ($\bar{\mathbf{I}}_t{}^{\mathrm{T}}$ and $\bar{\mathbf{I}}_{t+i}{}^{\mathrm{T}}$) share a part of same right-hand

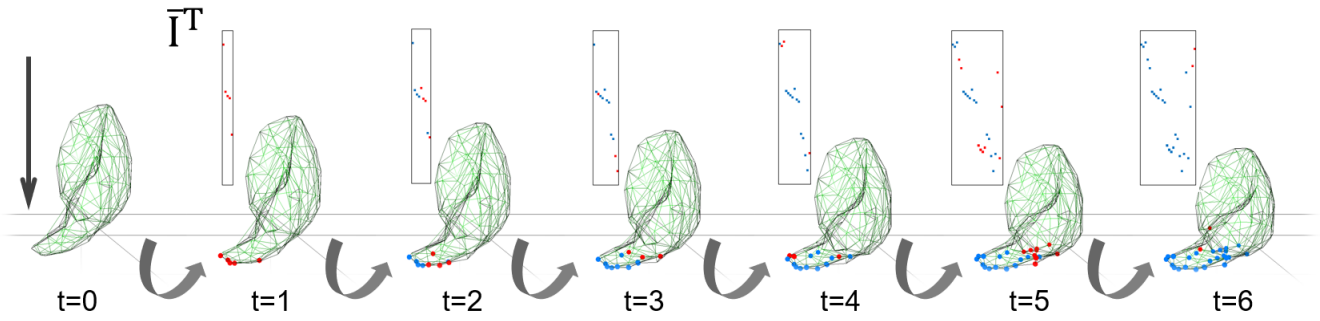

Figure 7: The evolution of contact space in a contact simulation between a deformable liver mesh and a rigid plane: The points on the mesh show the *isolated DOFs* that appear in the previous time steps (blue) and the new *isodofs* (red). It is revealed that the consecutive time steps usually shares same *isodofs*, which is relected on the *isodof Jacobian* matrix $\bar{\mathbf{I}}$.

sides. Consequently, while $\mathbf{L}$ is not updated, the results $\bar{\mathbf{S}}_{t+i}$ share the corresponding solutions with $\bar{\mathbf{S}}_t$, which are computed in previous time step t. Therefore, the following time step $t+i$ only needs to solve those that have not been shared:

$$\bar{\mathbf{S}}_{\text{new}} = \mathbf{L}^{-1}\bar{\mathbf{I}}_{\text{new}}{}^{\mathrm{T}} \tag{27}$$

where $\bar{\mathbf{I}}_{\text{new}}$ consists of new *isodofs* that emerges in $\bar{\mathbf{I}}_{t+1}$. Compared to the standard *isodof scheme* presented in Section 4, we have a smaller dimension to deal with, implying a further speedup.

To benefit speedup from the *reuse scheme*, we propose a hybrid implementation illustrated in Figure 8:

1. **standard scheme** While a new factorization is done in the asynchronous thread, the solver follows the operations in Algorithm 3.
2. **reuse scheme** While the factorized system is not updated, the solver performs a "reuse scheme" (see the implementation below).

In the "reuse scheme", a first step compares the current $\bar{\mathbf{I}}$ with the previous one $\bar{\mathbf{I}}_{\text{old}}$, outputting the new *isodofs* stored in a matrix $\bar{\mathbf{I}}_{\text{new}}$. To formulate the relation between $\bar{\mathbf{I}}_{\text{old}}$, $\bar{\mathbf{I}}_{\text{new}}$ and $\bar{\mathbf{I}}$, we use a function π, which is a actually partial permutation and can be represented by a matrix $\mathbf{P}_\pi$:

$$\bar{\mathbf{I}} = \mathbf{P}_\pi \begin{bmatrix} \bar{\mathbf{I}}_{\text{old}} \\ \bar{\mathbf{I}}_{\text{new}} \end{bmatrix} \tag{28}$$

Following Algorithm 3 and Equation (28), $\bar{\mathbf{W}}$ in the current time step is built as:

$$\begin{aligned} \bar{\mathbf{W}} &\approx \bar{\mathbf{I}}(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1}\bar{\mathbf{I}}^{\mathrm{T}} \\ &= \mathbf{P}_\pi \begin{bmatrix} \bar{\mathbf{I}}_{\text{old}} \\ \bar{\mathbf{I}}_{\text{new}} \end{bmatrix} (\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1} \begin{bmatrix} \bar{\mathbf{I}}_{\text{old}}{}^{\mathrm{T}} & \bar{\mathbf{I}}_{\text{new}}{}^{\mathrm{T}} \end{bmatrix} \mathbf{P}_\pi{}^{\mathrm{T}} \\ &= \mathbf{P}_\pi \begin{bmatrix} \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}} & \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \\ \bar{\mathbf{S}}_{\text{new}}{}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}} & \bar{\mathbf{S}}_{\text{new}}{}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \end{bmatrix} \mathbf{P}_\pi{}^{\mathrm{T}} \\ &= \mathbf{P}_\pi \underbrace{\begin{bmatrix} \bar{\mathbf{W}}_{\text{old}} & \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \\ (\bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}})^{\mathrm{T}} & \bar{\mathbf{S}}_{\text{new}}{}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \end{bmatrix}}_{\bar{\mathbf{W}}_{\text{extend}}} \mathbf{P}_\pi{}^{\mathrm{T}} \end{aligned} \tag{29}$$

with $\bar{\mathbf{S}} = \mathbf{L}^{-1}\bar{\mathbf{I}}_{\text{old}}$ and $\bar{\mathbf{S}}_{\text{new}} = \mathbf{L}^{-1}\bar{\mathbf{I}}_{\text{new}}$. The former has been com-

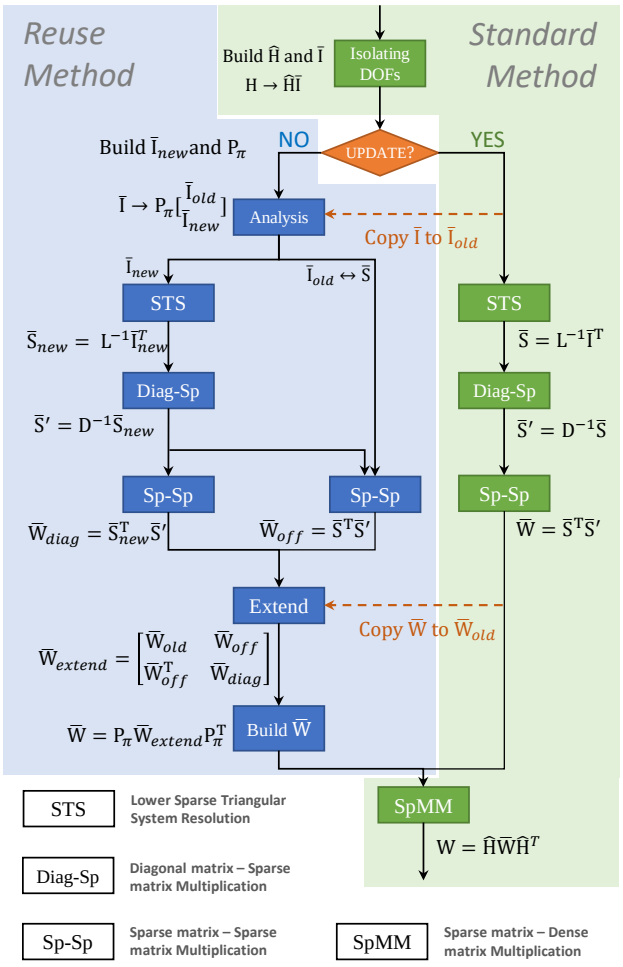

Figure 8: Scheme of standard/reuse scheme of *isodof* method: While a new factorization is done in the asynchronous thread, the solver performs a "standard scheme", storing the *isodof Jacobian* $\bar{\mathbf{I}}_{\text{old}}$, the STS solution $\bar{\mathbf{S}}$, and the *isodof delasus* $\bar{\mathbf{W}}_{\text{old}}$ in GPU memory. While the solver keeps using the same factorized system of the previous time steps, it performs a "reuse scheme".

puted in previous time steps and the latter is to be solved in the current time step. To finally build the current $\bar{\mathbf{W}}$, we have an "extended" matrix $\bar{\mathbf{W}}_{\text{extend}}$ to build, where the *diagonal* and the *off-diagonal* parts are formulated as:

$$\bar{\mathbf{W}}_{\text{diag}} = \bar{\mathbf{S}}_{\text{new}}{}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \tag{30a}$$

$$\bar{\mathbf{W}}_{\text{off}} = \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \tag{30b}$$

The detail implementation of the "reuse scheme" is illustrated in Figure 8. Although the number of operations is increased compared to the "standard scheme", the operations (i.e., STS resolution, ***SpMM***...) are very efficient since they usually have much smaller *isodof* dimension. Therefore, based on the *isodof* method presented in Section 4, the "reuse scheme" benefit a further speedup on performance from reducing the *isodof* dimension size. We still underline that the "reuse scheme" computes the same resolution as in Algorithm 1 and Algorithm 3.

6 Results

In this section we evaluate the computation cost of the *isodof* method presented in Section 4 as well as the *reuse isodof* method in Section 5. The simulation tests are conducted in the open-source SOFA framework with a CPU AMD@ Ryzen 9 5950X 16-Core at 3.40GHz with 32GB RAM, and a GPU GeForce RTX 3080 10GB.

The deformable meshes are modeled with the co-rotational formulation (although it should be compatible with other materials as proposed in [CADD15] for hyperelastic materials). Our methods are dedicated to assembling the *Delasus operator* and are compatible with various methods in the other steps. The free motion is solved with a preconditioned Conjugate Gradient (**PCG**), and the constraint resolution uses a projected Gauss-Seidel (**PGS**).

6.1 Evaluation of computation cost of the Schur-complement

In this section, we evaluate the performance of our methods in various conditions. We simulate the collision between a deformable raptor mesh and a rigid plane mesh, using a proximity-based method for the collision detection: The potential constraint pairs are defined by searching the closest elements between surface triangle meshes of contacting objects.

By simulating the simple collision between a deformable raptor mesh and a rigid plane (see Figure 1, Left), the tests are executed in conditions of various numbers of *mechanical DOFs* and contact constraints. We compare the performance of our methods of *isodof*/*Reuse isodof* to the method proposed in [CAK*14] that currently provides the fastest contact resolution in SOFA framework.

In Figure 9, the *isodof* method shows an average speedup of 47.42× compared to [CAK*14] and the *reuse isodof* shows a further speedup of 2.81× compared to the standard *isodof* and 133.31× compared to [CAK*14]. Our methods efficiently limit the computation cost: with 367 constraints (326 *mechanical DOFs* impacted) the *isodof* method takes 1.08*ms*; with 1860 constraints (1376 *mechanical DOFs* impacted) the *isodof* method takes 5.73*ms*, while the *reuse isodof* method takes 2.03*ms* by reusing 99.8% of the *isodofs*.

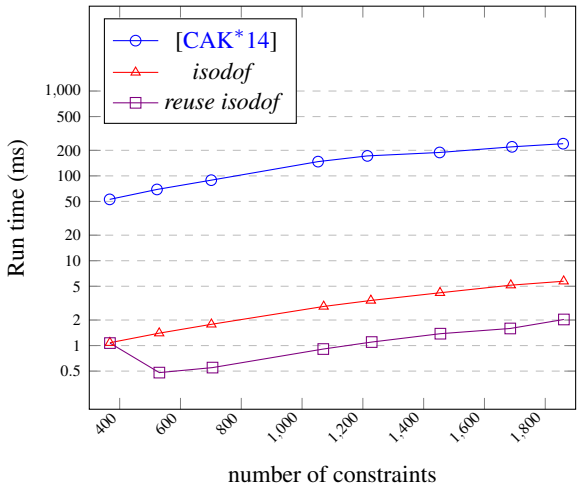

Figure 9: Computation cost of Schur-complement of different methods for various constraint number: contact simulation between a rigid plane and a deformable raptor mesh with 36069 *mechanical DOFs*. By changing the elasity parameters, the contact area is varied, leading to different constraint numbers. We note that the y axis is logarithmic in this figure.

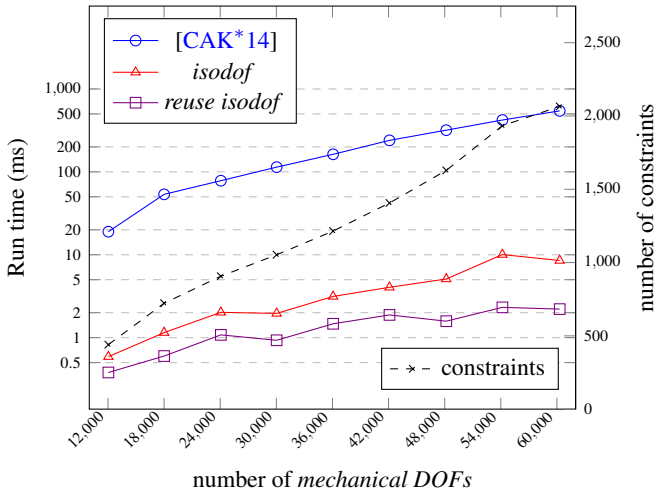

Figure 10: Computation cost of Schur-complement of different methods for various *mechanical DOFs*: contact simulation between a rigid plane and a deformable raptor mesh with various discretization. The mesh discretization has an impact on the contact constraints (dashed black line in the figure). We note that the y axis is logarithmic in this figure.

In Figure 10 we show the performances of our methods according to different mesh dimensions. The method in [CAK*14] reveals a quadratic function according to the number of *mechanical DOFs* as the discretization of the mesh has an impact on the searching of contact pairs in the proximity collision detection. When the problem size is increased, [CAK*14] (dense resolution) suffers from extremely large computation cost, while this cost is limited with our new methods (10.05*ms* for the *isodof* method and 2.32*ms* for the *reuse isodof* method). The speedup from [CAK*14] to the *isodof* is averagely 50.55× and it is enlarged up to 133.57× for the *reuse*

Method		Performance (in *ms*)						
Free Motion	Build W	Build + Fac.	Free Motion	Build **W**	Trans.	GS	Corr.	Time step
Pardiso-16		1129.72	37.7	57.49		57.46	26.32	1308.69
PCG + $\mathbf{LDL}^T$	[CAK*14]		41.45(10#it)	554.15	27.09	48.06	3.60	674.35
PCG + $\mathbf{LDL}^T$	*isodof*		42.20(10#it)	13.53	26.97	47.55	3.83	134.08
PCG + $\mathbf{LDL}^T$	*reuse isodof*		42.45(10#it)	6.56	26.22	46.58	3.82	125.63

Table 1: Collision simulation between a rigid plane and a deformable raptor with 59 529 *mechanical DOFs* and 2250 contact constraints. Performance of various methods: system assembly + analysis for *Pardiso* + factorization (**Build + Fac.**), free motion resolution, Schur-complement (**Build W**), transfer **W** from GPU to CPU (**Trans.**), Gauss-Seidel (**GS**) for constraint resolution, corrective motion (**Corr.**) as well as the entire time step (**Step**). For the implementation in *Pardiso* (processed in 16 parallel threads), an augmented system with the *constraint Jacobian* is factorized, while the factorized system is used in the resolutions of the free motion, the Schur-complement, and the corrective motion.

isodof. As expected, our method is poorly sensitive to the mesh dimension while the cost of method in [CAK*14] goes far beyond the real-time computation with large mesh dimensions.

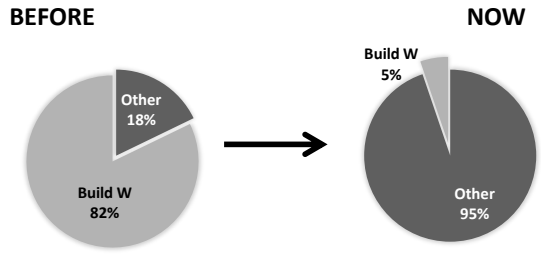

Figure 11: From [CAK*14] to our methods, change of contribution of the Schur-complement in the entire time step

In Table 1 we evaluate the entire time step for a real-time application. We compare our methods with the solvers in *Pardiso project*, which is a popularly used library for linear algebra due to its efficiency, especially while processing in parallel CPU threads. For direct solvers in *Pardiso*, a factorization process is necessary before the free motion resolution and the Schur-complement. Although the *Pardiso* for the Schur-complement (i.e., the method in [PSLG14]) is very fast, it requires a prerequisite augmented factorization. Computing the factorization depends directly on the dimension of *mechanical DOFs*, making it very difficult to achieve real-time computation for large-scale problems. On the other hand, the asynchronous preconditioner (i.e., the method in [CADC10] (PCG) for the free motion, and the method in [CAK*14] for the Schur-complement) removes the costly building and factorization stage out of the main simulation loop, showing a significant speedup compared to the direct solvers. Although this method gains significant speedup, it becomes prohibitive in large-scale contact problems in real-time applications (building **W** takes 554.15*ms* and 82.18% in a time step). With the *isodof* and *reusing isodof* methods, we succeed to limit the Schur-complement within a very low computation cost that is 13.53*ms* (10.09%) for *isodof* method and 6.56*ms* (5.22%) for *reuse isodof* method (see Figure 11)

6.2 Applications

In this section, we apply our methods to different examples. To perform fast collision detection, we use the GPU-based method [AFC*10] that relies on volume interpenetration. The tests are executed in the following examples, with various deformable meshes and challenges (detailed meshes, multi-objects, complex interactions, heterogeneous materials...).

6.2.1 Complex interaction: Pass Torus

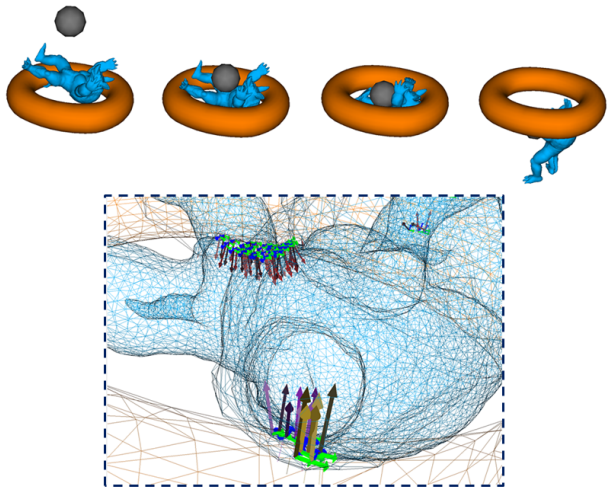

Figure 12: Pass a deformable armadillo through a torus. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

Figure 12 shows complex interactions between a rigid icosphere, a fixed rigid torus, and a deformable armadillo that has a detailed discretization with 31302 *mechanical DOFs*. The contact from the sphere is on the center of the armadillo, pushing it through the torus. On the other hand, the stiffness and the contacts on arms and legs generate the forces resisting against sphere's movement. It is, therefore, necessary to efficiently discretize contacts with mechanical coupling to compute and distribute the contact forces.

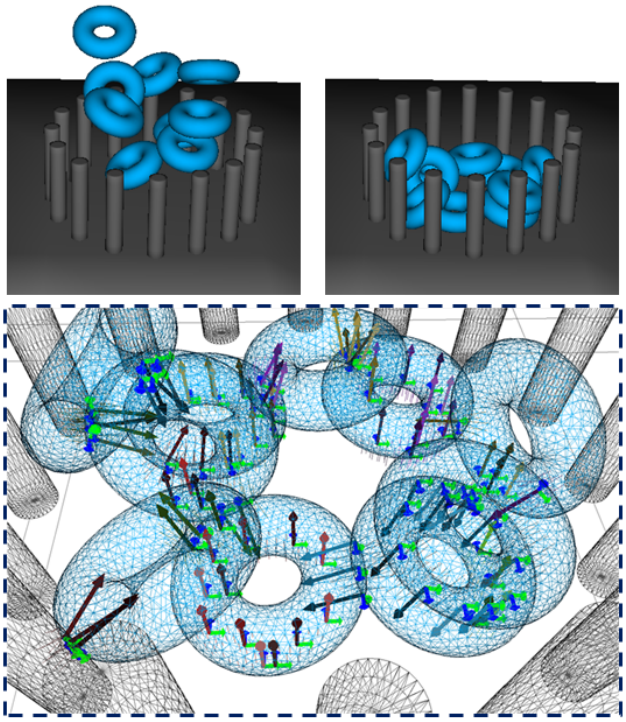

Figure 13: Collision between multiple deformable torus. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

6.2.2 Multi-objects

Our methods are compatible with multi-object systems, such as a scenario of multi-torus (see Figure 13). In the test we simulate 10 deformable torus with 3357 *mechanical DOFs* for each one. Each deformable torus is contacting with the others and the fixed pillars. Although we solve smaller mechanical problems in this scenario, the contact forces are transmitted among the objects, forming a complex multi-object system. Beyond the fact that the methods also provide speedup in multi-body contact simulations, building the *Delasus operator* is crucial to take into account the mechanical coupling of the system.

6.2.3 Dynamic contact: Rolling

To better evaluate the *reuse isodof* method, we design a "dynamic test" where the contacting area keeps shifting and numbers of new *isolated DOFs* appear in each time step (Figure 14). Although the contact area keeps changing, more than 89% of *isodofs* are reused. The *reuse isodof* method has an additional speedup of 1.5× compared to the standard *isodof* method ($7.58ms \rightarrow 5.06ms$). In this case, the *reuse isodof* method still receives interest since the *isodofs* are efficiently reused between the consecutive time steps even when the contact area is constantly varying.

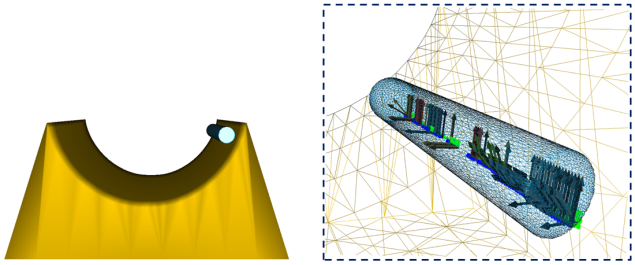

Figure 14: Rolling cylinder: a dynamic contact test for the *reuse isodof* method. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

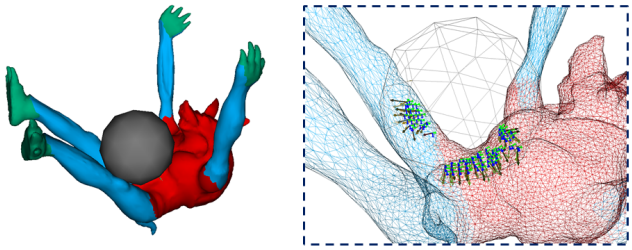

Figure 15: Heterogeneous material: the red parts are 10× stiffer than the blue parts, while the green parts are fixed. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

6.2.4 Heterogeneous material

Our methods are compatible with heterogeneous materials. In Figure 15, we simulate the collision between a rigid ico-sphere and a deformable armadillo of heterogeneous material with 31302 *mechanical DOFs*. The main difficulty of this example is related to the fact that the sphere applies contact forces on stiffer parts (in red), whereas softer parts (arms and legs in blue) should deform more obviously. By formulating the *Delasus operator*, the contacts are solved with mechanical coupling and contact forces are efficiently distributed in heterogeneous material while enforcing fast computation time.

6.2.5 Needle Insertion

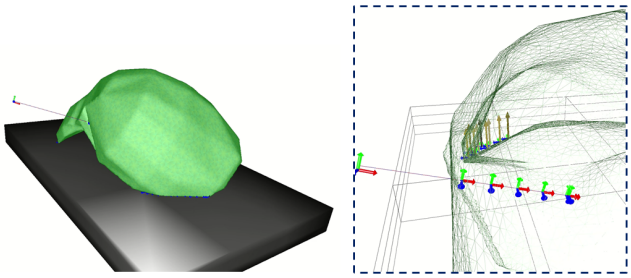

Figure 16: Needle insertion. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

Besides contact constraints (unilateral constraints), our methods

are also compatible with other constraint types. We apply our methods in a scenario of needle insertion, which is a popular topic in medical simulations [AGDC19]. Figure 16 shows a process of inserting a needle into a deformable liver mesh (highly detailed, with 31566 *mechanical DOFs*). Our methods are compatible with the needle constraints (bilateral constraints) that only impact the *mechanical DOFs* nearby the insertion trajectory, making the *isodof* method very efficient. Moreover, the *reuse isodof* method can benefit a significant speedup from reusing the *isodofs* nearby the insertion trajectory.

6.2.6 Rich contact

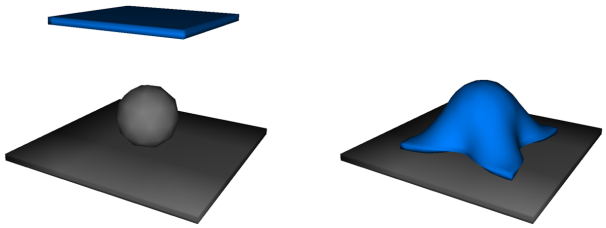

Figure 17: Rich contact.

We apply our methods in a scenario where a soft pad is covering on a rigid sphere. The contact area is very large and 1200-1300 contact constraints are generated while about 900 out of 6561 *mechanical DOFs* are impacted by the constraint. The ratio of constraints/*DOFs* in this scenario is about 0.2, which is significantly higher than other examples (less than 0.02). Our methods remain efficient in such rich contact cases: building **W** takes 2.03ms with the *reuse isodof* method, providing a speedup of 14.98 $\times$ compared to the method in [CAK*14].

6.2.7 Stacking problem

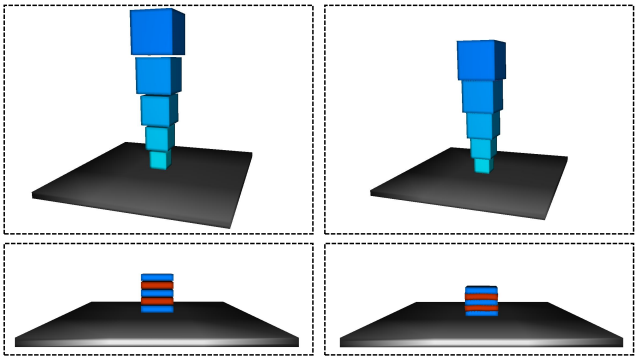

Figure 18: Stacking problems. Top: assembling **W** allows to couple the forces between stacking boxes with increasing masses which are represented by gradient blues. Bottom: assembling **W** is especially important for the stability in a heterogeneous stacking scenario with different stiffness (the blue pads are 10 $\times$ stiffer than the red pads).

We apply our methods in a stacking problem, which is similar to the example in [MEM*19]. In the first scenario, the stacking boxes are modeled with FEM and large young modulus to have a behavior near to rigid bodies. With increasing masses of the boxes, **PGS** can still handle the problem when the total mass ratio is of 256:1. When dealing with a mass ratio of 4096:1, the problem becomes very poorly conditioned and difficult to be solved with **PGS**. However, by assembling **W**, our methods allows to formulate a standard complementarity (linear system in Equation (15) combined with complementarity conditions in Equation (5) and (6)) discussed in [Erl13], making it flexible to be solved by different methods (e.g. pivoting methods to handle such a ill-conditioned problem).

Besides the homogeneous stacking, our methods are also tested in a heterogeneous stacking problem. In the second scenario, the stacking soft pads are modeled with different stiffness. Such a test will be failed when the contact forces are not coupled (**W** is diagonal) as the system is extremely unstable, which is discussed in [AE21]. In this test, the coupling of contact forces is very complex since the interaction on each contacting faces has impact on the other objects. Therefore, building the compliance matrix is very important to propagate the forces over different objects. Our methods allow to efficiently address this problem by fast building **W** that correctly couples the contact forces.

6.2.8 Gripping raptor

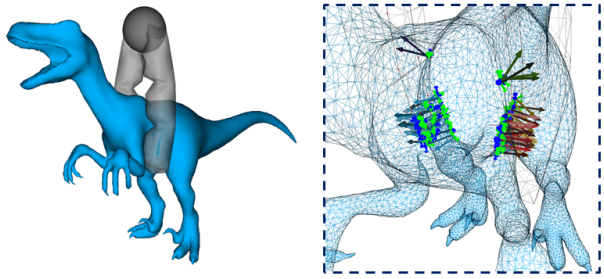

Figure 19: Grip a raptor with friction constraints. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

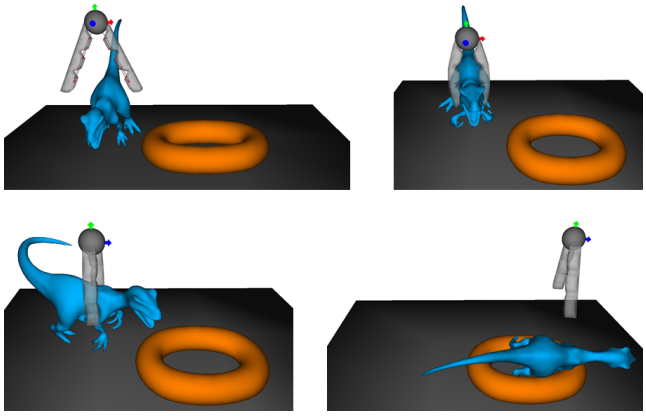

Figure 20: A pick-and-place task with Soft-Robot

Example	*Mecanical DOFs*	Constraints	Assembly scheme	Unbuilt scheme		
			Overhead (Build **W**)	Solving $\mathbf{LDL}^T$	PGS ite.	Extra cost
Pass Torus	31302	227.7	**2.19**	1.61	10	**16.1**
					200	**322.0**
Rich Contact	6561	1328.4	**2.03**	0.61	10	**6.1**
					200	**122.0**

Table 2: Comparison of the additional computation cost (in ms) between the assembly scheme and the unbuilt scheme. Since the choice of scheme will not impact the **PGS** performance, we can compare the **overhead/extra cost** in different schemes. The unbuilt scheme requires an extra cost of solving $\mathbf{LDL}^T$ (see Algorithm 2) in each **PGS** iteration. Hence, the total extra cost scales linearly with the number of iterations. In contrast, the assembly scheme with our new method only requires a small overhead of building **W** and will not cause any extra cost in the iterations.

We apply our methods on a gripping test. In a first scenario (see Figure 19), we use a soft gripper [Dur13] with two fingers to compress a deformable raptor mesh with 30033 *mechanical DOFs*. In a second scenario (see Figure 20), we fetch the raptor and deal with a pick-and-place task. This scenario implies comprehensive challenges: The deformable raptor is highly detailed, and the soft fingers are also deformable models (with 474 *mechanical DOFs* for each one). While gripping (compressing) the raptor, the fingers and the raptor are deformed to fit the contacting surface, generating numbers of contact constraints. Lifting, rotating, and moving the raptor by the fingers are complex operations where friction constraints are necessary. Moreover, the fingers, the raptor, the rigid plane, and the torus are grouped into a multi-object system, requiring efficient distribution of contact forces through mechanical coupling.

6.2.9 Assembling compliance vs. unbuilt scheme

As discussed in Section 3.5, when using a relaxation method such as **PGS**, the unbuilt scheme can be an option to solve the problem. In Table 2, we compare the additional computational cost between the unbuilt scheme and the assembly scheme (building **W**). We collect the data from two of our examples in the current section to compare the methods in both limited (*Pass Torus*) and rich contact (*Rich Contact*) cases. According to different contact cases, **PGS** will need several iterations to hundreds of iterations to converge. As illustrated in the table, for both limited/rich contact cases, the extra cost of applying $\mathbf{LDL}^T$ in each iteration overcomes the overhead of building the compliance matrix after several **PGS** iterations. With ten iterations, building **W** with our methods is already more efficient than the unbuilt scheme. This gap tends to be extremely large when the relaxation method needs hundreds of iterations to converge. We note that we use the same $\mathbf{LDL}^T$ factorization with the same reordering technique for the different schemes in this test. For the unbuilt scheme, the $\mathbf{LDL}^T$ resolution process in Algorithm 2 is optimized on GPU using a domain-decomposition technique in the forward/backward substitutions: The patterns of $\mathbf{L}/\mathbf{L}^T$ matrices are partitioned into sub-blocks that correspond to the branches on the elimination tree (see Figure 4). We can parallelize the resolution of branches for those who have no data dependency between each other. Following the structure of the elimination tree, such parallel resolution can be processed recursively until the root.

6.2.10 Evaluation

In these scenarios, we meet various challenges: The complex interactions usually require to efficiently distribute the contact forces through the mechanical coupling, such as the cases in multi-object systems and in the problems with heterogeneous materials; The needle insertion operation necessitates simulating both the unilateral constraints for contacts and bilateral constraints for needle insertion at the same time; The pick-and-place task is even more challenging with different requirements. Moreover, all the scenarios simulate highly detailed meshes, raising large-scale problems. In this case, typical CPU-based or GPU-based approaches suffer from high computation costs to assemble the system **W** for the constraint resolution. However, our methods can complete the challenges with limited costs to build the compliance matrix. In Table 3 we evaluate

Example	*Mecanical DOFs*	Constraints	Time Step	Build **W**	%	Speedup
Pass Torus	31302	227.7	109.2	2.19	2.01 %	12.89×
Multi-torus	3357 × 10	323.28	89.81	8.83	9.83 %	3.87×
Rolling	26082	537.99	36.56	5.06	13.84 %	14.27×
Hetero-Material	31302	270.96	156.17	2.14	1.37 %	19.84×
Needle Insertion	12555	96.84	25.56	3.29	12.87 %	1.86×
Rich Contact	6561	1235.40	24.95	2.03	8.14 %	14.98×
Stacking	30240	163.26	121.54	4.00	3.29 %	1.00×
Hetero-Stacking	2205	240.0	18.86	1.69	8.96 %	1.04×
Catch Raptor	30033	486.59	144.76	0.73	0.50 %	64.00×

Table 3: Performance (in ms) in various examples: we evaluate the computation cost of the Schur-complement (**Build W** with the *reuse isodof* method), its percentage in a time step, and the speedup compared to the method in [CAK*14].

the computation cost of the Schur-complement as well as its contribution in an entire time step. Although in different applications, our methods have different performances, we succeed in limiting the cost of building the *Delasus operator* within less than 10*ms* in all the examples. Consequently, with our methods, the Schur-complement process in constraint-based resolutions is no more a critical obstacle in real-time simulations.

7 Conclusion and future work

In this paper, we presented a fast approach for constraint resolution in large-scale FE simulations. Our methods are founded on precondition-based contact resolution. The *isodof* method reformulates the Schur-complement by isolating the *mechanical DOFs* in the *constraint Jacobian* matrix. This reformulation allows us to perform a sparse resolution that is also suitable for parallelizing on GPU. The *reuse isodof* method further decreases the problem dimension size, making it more efficient to compute the Schur-compelement. Even in the case where number of constraints and number of *mechanical DOFs* are highly raised, our methods are capable of limiting the computation costs, making the Schur-complement process no more an obstacle in the real-time simulations (see Figure 11). Furthermore, our methods mathematically compute the same result with the method in [CAK*14]. As a result, we enable the possibility to compute large-scale real-time simulations in the presence of contact and friction.

In our experimentation set, the simulations are only tested with co-rotational models that rely on a GPU-based matrix-free solver [ACF11]. However, such GPU implementation is not available for other hyperelastic models in SOFA. A CPU-based implementation for hyperelastic models becomes extremely prohibitive for large-scale problems we are addressing. Therefore, we did not show the results with these models because the model will be dominant in terms of computation time (while our methods are compatible with hyperelastic models, as demonstrated in [CADD15]). Parallelizing such models is not in the scope of the paper. But one of our future works is to implement such a GPU-based implicit solver and to enable real-time computation for hyperelastic models.

The fast computation brings about a limitation for asynchronous preconditioners: The computation cost in the main simulation loop is significantly reduced; In contrast, the factorization in asynchronous thread remains very costly; the number of simulation steps that are necessary to update the asynchronous factorization is considerably increased. Using a slowly updated matrix may cause significant errors in case of large deformations. An efficient solution may be to process the asynchronous factorization on GPU, which requires performing asynchronous multi-GPU computation.

Acknowledgement: This work was supported by French National Research Agency (ANR) within the project SPERRY ANR-18-CE33-0007.).

References

[ACF11] ALLARD J., COURTECUISSE H., FAURE F.: Implicit FEM and fluid coupling on GPU for interactive multiphysics simulation. In *ACM SIGGRAPH 2011 Talks, SIGGRAPH'11* (New York, New York, USA, 2011), ACM Press, p. 1. 15

[ACF12] ALLARD J. A., COURTECUISSE H., FAURE F.: Implicit fem Solver on GPU for Interactive Deformation Simulation. In *GPU Computing Gems Jade Edition*. NVIDIA/Elsevier, 2012, pp. 281–294. 2

[AE21] ANDREWS S., ERLEBEN K.: Contact and friction simulation for computer graphics. *ACM SIGGRAPH 2021 Courses, SIGGRAPH 2021* (8 2021). 3, 13

[AFC*10] ALLARD J., FAURE F., COURTECUISSE H., FALIPOU F., DURIEZ C., KRY P. G.: Volume contact constraints at arbitrary resolution. *ACM SIGGRAPH 2010 Papers, SIGGRAPH 2010 C*, 3 (jul 2010), 1–10. 2, 11

[AGDC19] ADAGOLODJO Y., GOFFIN L., DE MATHELIN M., COURTECUISSE H.: Robotic Insertion of Flexible Needle in Deformable Structures Using Inverse Finite-Element Simulation. *IEEE Transactions on Robotics 35*, 3 (2019), 697–708. 13

[APS99] ANITESCU M., POTRA F. A., STEWART D. E.: Time-stepping for three-dimensional rigid body dynamics. *Computer Methods in Applied Mechanics and Engineering 177* (7 1999), 183–197. 2

[Bar96] BARAFF D.: Linear-time dynamics using Lagrange multipliers. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996* (1996), ACM, pp. 137–146. 2, 4

[BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics. *ACM Transactions on Graphics (TOG) 33* (7 2014). 2, 3

[BMO*14] BENDER J., MÜLLER M., OTADUY M. A., TESCHNER M., MACKLIN M.: A survey on position-based simulation methods in computer graphics. *Computer Graphics Forum 33*, 6 (2014), 228–251. 2

[BNC96] BRO-NIELSEN M., COTIN S.: Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum 15* (1996), 57–66. 2

[BSK20] BARGTEIL A. W., SHINAR T., KRY P. G.: An introduction to physics-based animation. *SIGGRAPH Asia 2020 Courses, SA 2020* (11 2020). 1

[CA09] COURTECUISSE H., ALLARD J.: Parallel dense gauss-seidel algorithm on many-core processors. In *2009 11th IEEE International Conference on High Performance Computing and Communications, HPCC 2009* (2009), IEEE, pp. 139–147. 3

[CADC10] COURTECUISSE H., ALLARD J., DURIEZ C., COTIN S.: Asynchronous preconditioners for efficient solving of non-linear deformations. In *VRIPHYS 2010 - 7th Workshop on Virtual Reality Interactions and Physical Simulations* (nov 2010), pp. 59–68. 3, 5, 11

[CADD15] COURTECUISSE H., ADAGOLODJO Y., DELINGETTE H., DURIEZ C.: Haptic rendering of hyperelastic models with friction. *IEEE International Conference on Intelligent Robots and Systems 2015-Decem* (sep 2015), 591–596. 10, 15

[CAK*14] COURTECUISSE H., ALLARD J., KERFRIDEN P., BORDAS S. P., COTIN S., DURIEZ C.: Real-time simulation of contact and cutting of heterogeneous soft-tissues. *Medical Image Analysis 18*, 2 (2014), 394–410. 3, 6, 7, 8, 10, 11, 13, 14, 15

[CTA*08] COMAS O., TAYLOR Z. A., ALLARD J., OURSELIN S., COTIN S., PASSENGER J.: Efficient Nonlinear FEM for Soft Tissue Modelling and Its GPU Implementation within the Open Source Framework SOFA. In *Biomedical Simulation* (2008), vol. 5104 LNCS, pp. 28–39. 2

[DDKA06] DURIEZ C., DUBOIS F., KHEDDAR A., ANDRIOT C.: Realistic haptic rendering of interacting deformable objects in virtual environments. *IEEE Transactions on Visualization and Computer Graphics 12*, 1 (apr 2006), 36–47. 3, 5

[Dur13] DURIEZ C.: Control of elastic soft robots based on real-time finite element method. *Proceedings - IEEE International Conference on Robotics and Automation* (2013), 3982–3987. 14

[Erl13] ERLEBEN K.: Numerical methods for linear complementarity problems in physics-based animation. *ACM SIGGRAPH 2013 Courses, SIGGRAPH 2013*, February (2013). 3, 13

[Fel00] FELIPPA C.: *A systematic approach to the element-independent corotational dynamics of finite elements*. Tech. Rep. January, College Of Engineeringuniversity Of Colorado, 2000. 2

[Geo73] GEORGE A.: Nested Dissection of a Regular Finite Element Mesh. *SIAM Journal on Numerical Analysis 10*, 2 (apr 1973), 345–363. 5

[GOM*06] GALOPPO N., OTADUY M. A., MECKLENBURG P., GROSS M., LIN M. C.: Fast simulation of deformable models in contact using dynamic deformation textures. *Computer Animation, Conference Proceedings 02-04-September-2006* (2006), 73–82. 3

[HA18] HERHOLZ P., ALEXA M.: Factor once: Reusing Cholesky factorizations on sub-meshes. *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018 37*, 6 (jan 2018), 1–9. 3, 5

[HSH20] HERHOLZ P., SORKINE-HORNUNG O.: Sparse cholesky updates for interactive mesh parameterization. *ACM Transactions on Graphics 39*, 6 (2020). 3, 5

[Jea99] JEAN M.: The non-smooth contact dynamics method. *Computer Methods in Applied Mechanics and Engineering 177*, 3-4 (1999), 235–257. 3

[KE20] KIM T., EBERLE D.: Dynamic deformables: Implementation and production practicalities. *ACM SIGGRAPH 2020 Courses, SIGGRAPH 2020* (2020). 1

[KKB18] KUGELSTADT T., KOSCHIER D., BENDER J.: Fast Corotated FEM using Operator Splitting. *Computer Graphics Forum 37*, 8 (2018), 149–160. 2, 3

[LBK17] LIU T., BOUAZIZ S., KAVAN L.: Quasi-Newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics 36*, 3 (2017). 2, 3

[LJBBD20] LY M., JOUVE J., BOISSIEUX L., BERTAILS-DESCOUBES F.: Projective dynamics with dry frictional contact. *ACM Transactions on Graphics (TOG) 39* (7 2020). 2

[MEM*19] MACKLIN M., ERLEBEN K., MÜLLER M., CHENTANEZ N., JESCHKE S., MAKOVIYCHUK V.: Non-smooth Newton methods for deformable multi-body dynamics. *ACM Transactions on Graphics 38*, 5 (oct 2019). 3, 13

[MHC*10] MARCHESSEAU S., HEIMANN T., CHATELIN S., WILLINGER R., DELINGETTE H.: Multiplicative Jacobian Energy Decomposition Method for Fast Porous Visco-Hyperelastic Soft Tissue Model. In *Lecture notes in computer science*, vol. 6361. Springer, 2010, pp. 235–242. 2

[MHHR07] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. *Journal of Visual Communication and Image Representation 18* (4 2007), 109–118. 2

[MMC16] MACKLIN M., MÜLLER M., CHENTANEZ N.: Xpbd: Position-based simulation of compliant constrained dynamics. *Proceedings - Motion in Games 2016: 9th International Conference on Motion in Games, MIG 2016* (10 2016), 49–54. 3

[PSLG14] PETRA C. G., SCHENK O., LUBIN M., GÄERTNER K.: An augmented incomplete factorization approach for computing the schur complement in stochastic optimization. *SIAM Journal on Scientific Computing 36*, 2 (2014), C139–C162. 3, 11

[Ren13] RENARD Y.: Generalized Newton's methods for the approximation and resolution of frictional contact problems in elasticity. *Computer Methods in Applied Mechanics and Engineering 256* (2013), 38–55. 3

[SDCG08] SAUPIN G., DURIEZ C., COTIN S., GRISONI L.: Efficient Contact Modeling using Compliance Warping. In *Computer graphics international* (2008). 3

[SG06] SCHENK O., GÄRTNER K.: On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis 23* (2006), 158–179. 3, 5

[WTB*21] WANG Q., TAO Y., BRANDT E., CUTTING C., SIFAKIS E.: Optimized Processing of Localized Collisions in Projective Dynamics. *Computer Graphics Forum 40*, 6 (2021), 382–393. 2

[XL18] XU L., LIU Q.: Real-time inextensible surgical thread simulation. *International Journal of Computer Assisted Radiology and Surgery 13* (7 2018), 1019–1035. 6

[ZZG18] ZHANG J., ZHONG Y., GU C.: Deformable models for surgical simulation: A survey. *IEEE Reviews in Biomedical Engineering 11* (11 2018), 143–164. 2