Real-time simulation of contact and cutting of heterogeneous soft-tissues.

Hadrien Courtecuisse[a,b], Jérémie Allard[b], Pierre Kerfriden[a], Stéphane P. A. Bordas[a], Stéphane Cotin[b], Christian Duriez[b]

[a]*Institute of Mechanics and Advanced Materials, Cardiff University, UK*
[b]*SHACRA Project, INRIA, France*

Abstract

This paper presents a numerical method for interactive (real-time) simulations, which considerably improves the accuracy of the response of heterogeneous soft-tissue models undergoing contact, cutting and other topological changes. We provide an integrated methodology able to deal both with the ill-conditioning issues associated with material heterogeneities, contact boundary conditions which are one of the main sources of inaccuracies, and cutting which is one of the most challenging issues in interactive simulations. Our approach is based on an implicit time integration of a non-linear finite element model. To enable real-time computations, we propose a new preconditioning technique, based on an asynchronous update at low frequency. The preconditioner is not only used to improve the computation of the deformation of the tissues, but also to simulate the contact response of homogeneous and heterogeneous bodies with the same accuracy. We also address the problem of cutting the heterogeneous structures and propose a method to update the preconditioner according to the topological modifications. Finally, we apply our approach to three challenging demonstrators: i) a simulation of cataract surgery ii) a simulation of laparoscopic hepatectomy iii) a brain tumor surgery.

Keywords: Finite element method, Interactive, Cutting, Real-time, Contact, Preconditioner, GPU, Collision response, Topological modifications

1. Introduction

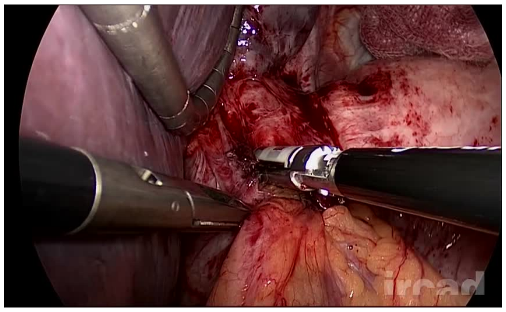
Interactive numerical simulation of surgical procedures, to provide realistic surgical simulators for the training of surgeons and to guide them during specific interventions, could open avenues leading to improved patient care and reduced risks.[1] The main requirement to be able to construct useful surgical simulators is to be able to simulate the mechanical response of organs. This requires generating geometrical and mechanical simplifications of these organs, adapted for simulations. The most widely used method to do so today is the finite element method (FEM).

Thus, building surgical simulators involves at least four major challenges:

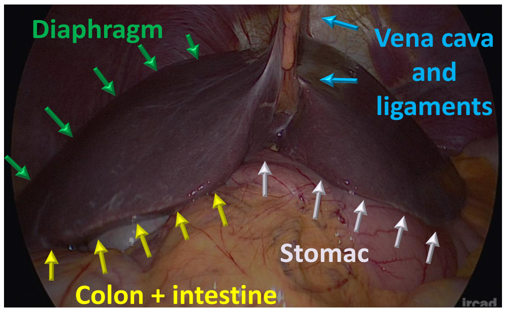
- it is difficult to characterize the material properties of living tissues, which are patient-specific and to predict their mechanical behavior;
- organ models must be acquired from medical images, and this process is still not automatic, requiring difficult segmentation and mesh generation. These meshes, which are geometrical simplifications of the organs are heavily constrained by the imposition of boundary conditions and the incompressibility of most tissues;
- significant numerical issues must be overcome, as most organs have an heterogeneous stiffness (which leads to ill-conditioned problems), are composed of multiple tissue types (which may lead to difficulties in defining boundary conditions) and are mostly incompressible (which leads to locking for most widely used finite elements);
- most importantly, the response must be computed in real-time to allow user interactions with the virtual body: pushing, prodding, palpation, needle insertion and cutting... In this sense, a simulation which would be mechanically realistic, but not interactive would not be fit for purpose and it can be claimed that lack of interactivity be the main error source.

Most of previous work has focused on producing accurate models for the deformations of soft tissues, but real-time simulations are still mainly composed of a single homogeneous organ with simple boundary conditions.

[1]Surgical simulators are meant here as computer-based tools used to simulate a surgeon's intervention in a virtual environment.



(a) Camera view during a hepatic surgery.



(b) Constraints from the environment.

Figure 1: Real and simulated hepatectomy in laparoscopy.

However, we believe that it is also fundamental to take into account the deformable environment of the tissues to obtain a realistic global behavior. For instance, even if during a hepatectomy the view of the camera is focused on a small part of the liver tissues (see fig 1) the human body is composed of multiple organs which play an important role in the resulting deformation and motion of the liver (breathing, contact with neighboring organs, etc.). Indeed, if we look closely at the boundary conditions of the liver: the upper part is compressed by the diaphragm on which it is attached through several ligaments including the falciform ligament. Moreover, the liver is in contact with the vena cava and the stomach and linked to it by the hepatogastric ligament. Therefore, when studying the motion and deformation of the liver, it cannot be considered as an isolated organ.

Beyond the importance of boundary conditions, our additional working hypothesis is that realistic surgical simulation also requires to interactively compute significant topological modifications of the organs, e.g. cutting. In this case, boundary conditions are also fundamental because stiff interactions with the tool (controlled by the user) as well as complex deformable–deformable interactions between the lips of the cut must be solved during the cut. Performing this at interactive rate is complicated, in particular with heterogeneous structures.

The contribution of this paper is to propose a consistent framework to address the aforementioned requirements for surgical simulations. This framework relies on an implicit time integration of the non-linear set of equations coming from the finite element models of the deformation. The core of the method is the use of an original preconditioning technique which is updated asynchronously at low frequency. This preconditioner reduces the convergence issues appearing when computing the deformation of nonlinear heterogeneous structures, and provides a very good estimate of the compliance[2] operator associated with the coupling between the solids in contact. We also extend the method to cut these heterogeneous organs at an interactive rate whilst resolving existing or new contact surfaces emanating from these topological modifications. Our solution is based on the Sherman Morison Formula to update the asynchronous preconditioner in case of topological modifications. Finally the generic nature of the method is demonstrated through three very different kinds of applications: a simulation of cataract surgery, a simulation of a hepatectomy using laparoscopy procedure, and a simulation of cerebral tumor removal.

The paper is organized as follows: In section 2, we briefly review the real-time simulation of deforming bodies. In section 3 we define the problem at hand and provide notations and key concepts necessary to build our method. Section 4 is dedicated to the description of the asynchronous preconditioner. In section 5 we propose to use the asynchronous preconditioner to solve the constraints associated with the multi-deformable-body contact problem under consideration. This section also provides details on a GPU implementation to achieve real-time results. In section 6, we extend the method to handle the topological modifications, whereas in section 7, we evaluate our method in terms of accuracy and computation time. Finally, in Section 8 we exercise the methodology in three practical problems involving complex heterogeneous structures in interaction.

2. Literature review

Biomechanical simulation with user interactions involves many challenges such as real-time computation of the deformation of soft tissues, collision detection, contact modeling, topological modification and haptic feedback (see Nealen et al. (2006); Payan (2012) for a broad survey).

2.1. Simulation of deformable bodies

The first methods proposed to simulate the deformation of soft tissues in real-time relied on mass-spring systems, e.g. Kühnapfel et al. (2000).

[2]inverse of the stiffness

Although such discrete methods are simple to implement and very fast, they are difficult to parameterize with material properties such as the Young's modulus. Moreover, they introduce anisotropy through the choice of the mesh which gives rise to stability and accuracy issues (node flipping, difficulty to preserve the volume).

Finite element methods (FEM) provide high bio-mechanical realism Zienkiewicz and Taylor (1991), mainly because the very complex non-linear behavior of soft-tissue is directly accounted for through constitutive relations. Real-time computations were first achieved for linear elastic material models (see Bro-Nielsen and Cotin (1996); Cotin et al. (1999) or James and Pai (1999)). In linear elasticity, precomputations (offline) can be used to accelerate the online simulations, which can drastically decrease computational expenses.[3] Additionally, the solution can be obtained with a single matrix inversion and no iterative process is required.

However, the small strain assumption is incorrect in practice, and produces erroneous results when the solids undergo large deformations.

Moreover, soft tissues do not behave linearly, i.e. the stress field is non-linearly related to the strain field, through the constitutive relation.

The *co-rotational* method is a very old method, which originated in continuum mechanics Freund (1970) and was introduced by Felippa (2000) within the field of numerical methods. In this formulation, the stiffness of each element is assumed linear within the local frame described by its rotated state, which enables to simulate geometric non-linearities (i.e. large displacements, large rotations but small strains). This enables to produce realistic simulations, while maintaining the algorithmic complexity at a minimum Felippa and Haugen (2005).

Later, other methods were proposed to model both geometric and material non-linearities. Material non-linearities are expressed through the non-linearity of the constitutive law relating the strain tensor to the stress tensor, characteristic of soft tissues. Real time models were recently proposed Comas et al. (2008); Joldes et al. (2009); Marchesseau et al. (2010), but these models remain in general complex and expensive, and the simulation of realistic boundary conditions such as interactions between deformable organs

[3]The use of precomputed solutions for highly non-linear problems is intensively pursued, e.g. in Niroomandi et al. (2008) for hyper elasticity, and Kerfriden et al. (2011); Kerfriden et al. (2012) for damage problems.

and surgical instruments is still an issue.

2.2. Time discretization

Explicit integration schemes are most widely adopted Miller et al. (2007); Taylor et al. (2008) in surgical simulation. The main advantage is that the solution process only involves the mass matrix, which can be lumped (diagonalised). The equations of motion are thus decoupled and each degree of freedom can be solved independently, making the solution process very fast, and inherently well-suited to parallelization Comas et al. (2008). The major drawback of explicit dynamics is the need to satisfy the Courant-Friedrichs-Lewy stability condition, which forces a strict upper bound on the time step used for integration. This means that the time step must be chosen to be less than a critical value, equal to the traversal time of an elastic wave through an element. Explicit methods are consequently particularly well-suited to very soft tissues such as the brain Joldes et al. (2009), but very small time steps (which prevent real-time computations) must be chosen for stiffer structures. Moreover, in explicit simulations, it is assumed that if the formulation is stable, it is accurate. This, however, does not guarantee that, at each time step, the residual vector is minimized, and hence, that the external and internal forces balance.

This is the major reason for developing implicit time integration techniques for real time simulations, which is one of our goals. Providing flexibility in the choice of the time step, even for very stiff objects Baraff and Witkin (1998), is required to achieve our aim: simulate user-controlled interactions between arbitrarily stiff anatomical structures or tools. Of course, the advantages of such methods come at the cost of having to solve a set of linear equations at each time step. Yet, this paper aims to show that implicit integration schemes can offer a reasonable tradeoff between robustness, stability, convergence and computation time, in particular when combined with a GPU implementation.

2.3. Solving the set of nonlinear equations

Using implicit integration, a non-linear set of equations must be solved at each load step. This set of equations is usually solved using an iterative method based on the Newton-Raphson method which solves the set of nonlinear equations through a sequence of solutions of linear equations.

The set of linear equations can either be solved by direct solvers or iterative solvers. Direct solvers provide the solution by computing the actual inverse of the system matrix Bro-Nielsen and Cotin (1996), or by creating a factorization that can then be used to compute the solution Barbič and James (2005).

These methods are often too costly to be applied at each iteration, and are often used in combination with an approach to reduce the number of degrees of freedom of the model, either using condensation on surface nodes Bro-Nielsen and Cotin (1996), or reduced-coordinate models Barbič and James (2005). Recently, Hecht et al. (2012) proposed a method to incrementally update a sparse Cholesky factorization. They obtain fast performance by making only partial changes to the simulation's linearized system matrices, but the method is closely related to the co-rotational formulation and cannot take into account topological modifications.

The second class of methods are iterative Saad (1996), and start from an initial estimate and iteratively refine it to approach the exact solution. One of the most popular methods is the Conjugate Gradient (CG) algorithm. Although in theory up to n iterations are necessary to achieve convergence for n equations in the system, in practice it is possible to stop the algorithm much sooner depending on the required accuracy. Parallel implementations on CPU are now well-mastered and optimized: see for example Parker and O'Brien (2009); Hermann et al. (2009) and start to appear on GPU Bolz et al. (2003); Buatois et al. (2009); Allard et al. (2011). Iterative solvers are often faster than direct methods, and require less memory storage, but they converge slowly for ill-conditioned problems, i.e. when the ratio of the largest and smallest eigenvalues is large. This is the case when solving linear systems of equations associated with the discretization of heterogeneous structures, as is the case in this paper.

Another intense area of research aims to improve the performance of the CG algorithm with the use of preconditioners to speed-up its convergence. In the context of interactive simulation, Baraff and Witkin (1998) proposed to use a diagonal inverse, often called a Jacobi preconditioner. More advanced preconditioners such as Cholesky factorizations have also been studied Hauth et al. (2003). However, the performance improvement remains limited since the preconditioner itself is expensive to compute. Domain decomposition preconditioners are popular, for example in multi-scale parallel simulations Dryja and Widlund (1989) and for extended (enriched) finite element methods Menk and Bordas (2011b). Multi-grid pre conditioners, e.g. Braess (1986) were developed for real-time simulations for example in Dick et al. (2011b) and to simulate cuts in deformable objects in Dick et al. (2011a) and for fracture problems in Hiriyur et al. (2012); Berger-Vergiat et al. (2012); Gerstenberger and Tuminaro (2012).

2.4. Simulation of the interactions

A key requirement for realistic surgical simulators is to treat contact between deformable-deformable, deformable-"stiff" and "stiff"-"stiff" objects. A common solution to deal with contact consists of using a penalty method, which modifies the variational principle and solves the contact condition only approximately. A didactic review of constraint enforcement in a variational context is provided in Hughes (2000) and a review of error estimates associated with this enforcement of variational inequalities in finite element methods can be found in the seminal paper Brezzi et al. (1977). In penalty methods, a contact force $\boldsymbol{f} = \alpha\delta\mathbf{n}$ is added at each contact point, where δ is a measure of the interpenetration, $\mathbf{n}$ is the contact normal and α is a stiffness factor known as penalty coefficient. The constraint is satisfied exactly *only* when α is infinite. The higher α the better the constraint is imposed. However, the higher the value of α, the worse the condition number of the system is, the higher the spurious oscillations in contact forces, and the smaller the time step. The selection of α is also problem-dependent, and depends particularly strongly on the stiffness ratio between the contacting objects. Consequently, the penalty method, although extremely simple is limited for the applications we have in mind, i.e. contacting heterogenous objects.

Lagrange multipliers or augmented Lagrangian techniques Hughes (2000); Renard (2012); Jean (1999); Jourdan et al. (1998); Wriggers and Panatiotopoulos (1999) are usually preferred to penalty methods to treat contact constraints accurately and robustly.

Nitsche's method Nitsche (1971) offers a consistent constraint imposition method which was used in the context of contact mechanics in Wriggers and Zavarise (2008) and frictional contact Heintz and Hansbo (2006).

In graphics, methods used to solve contact equations can be classified into two categories which are numerically equivalent: *Quadratic Programming* (QP) methods and *Complementary Problem* methods that could be linear (LCP) or non-linear (NLCP). QP methods define the constraints directly into the mechanical system. They can be used to treat the inequality of the contact Redon et al. (2002); Pauly et al. (2004), and also to simulate friction using a discretized pyramidal cone Kaufman et al. (2008). However, these publications address the case of rigid bodies in contact, where the number of degree of freedom (DOF) is smaller than the number of contact constraint freedoms. Indeed, the resulting number of equations with QP methods is of the same order of magnitude as the number of DOFs of the interacting objects. Therefore, these methods are difficult to adapt to the simulation of the interaction between finely meshed deformable bodies in real-time.

An important advantage of (N)LCP methods is that the number of constraint equations is proportional to the number of contacts, which is often much smaller than the number of DOF in the context of deformable models. LCP can be used to simulate frictionless contact between deformable models Duriez et al. (2003) in real-time whereas NLCP methods can be used to simulate friction contacts with the exact friction cone Duriez et al. (2006). The main limitation of these methods is that the solution process involves the *compliance matrix*, which is the inverse of a large system composed of the mass, the damping and the stiffness of the deformable objects. Although the evaluation of this inverse in real-time is crucial to define the boundary conditions of the deformable structures, very few methods addressed this issue. Duriez et al. (2004) proposed to precompute the *compliance matrix*, but the solution is limited to linear elastic deformation. Otaduy et al. (2009), proposed to evaluate the compliance matrix using additional Gauss-Seidel iterations on the deformable models, but the method was not presented in a real-time context. Saupin et al. (2008) proposed a method, named *compliance warping*, which is dedicated to co-rotational models. It consists of pre-computing the *compliance matrix* from the rest position, and updating it using a local estimation of the nodal rotations. However, this approximation can become inaccurate for large deformations, and the method is limited to relatively coarse meshes. A prediction-correction scheme is introduced in Peterlik et al. (2011) to mitigate the inaccuracies and the method extends the formulation to haptic feedback with generic constraints between the deformable models.

To our knowledge, it was never demonstrated that any of the above methods are adequate to enforce contact constraints between geometrically complex deformable objects represented by moderately fine meshes (thousands of DOFs), both accurately and at interactive rates (30-50 frames per second). Nevertheless, this problem is particularly relevant for medical applications, since all the organs in a human body are subjected to interactions from other tissues, and resolving accurately these interactions between organs can be construed as more critical than the accuracy of the deformation of each individual organ itself. An important contribution of this article is to propose a generic solution to address this problem.

2.5. Simulation of topological modifications

When cutting through a finite element mesh, discontinuities in the displacement field must be introduced. This can be done through local remeshing. Several approaches were proposed to maintain a relatively good mesh quality Ganovelli and O'Sullivan (2001); Bielser et al. (2003); Molino et al. (2007); Sifakis et al. (2007). The main difficulty is to preserve the quality and the density of the mesh during the subdivision process, in order to avoid distorted elements, which lead to convergence difficulties during the simulation. Alternatives to generating finite element meshes, in particular to handle topological changes are extended finite element methods Nicolas et al. (1999). These methods allow cuts, material interfaces, and domain boundaries to be described independently of a background mesh, which may also be progressively adapted using a posteriori error estimators Bordas and Duflot (2007); Duflot and Bordas (2008); Bordas et al. (2008); Ródenas et al. (2008) or local heuristics Menk and Bordas (2011a). However, such methods are not yet developed in the real-time context, and the simulation of collisions and interactions remains unsolved, in particular because the geometry of the discontinuities is known implicitly.

Considering the mechanical aspect, handling topological changes essentially requires the update of the stiffness matrix. As most recent real-time deformation methods are based on non-linear models (geometrical and/or material), topological changes often add little overhead to the process. However, the treatment of the interactions in case of topological modifications remains an open problem. Few methods were proposed solutions to handle collision detection with the new triangles of the modified topology (where the quality and the density of the triangulation of the surface around the cut is difficult to control). Allard et al. (2010) proposed a GPU-based collision detection that provides intersection volumes without any pre-computation and regardless of the density of the mesh. On the other hand, the treatment of the interactions during the cut is also problematic. Indeed, although penalty-based solutions could be used with a limited overhead, we saw that these methods are not adapted to our purpose. Courtecuisse et al. (2011) proposed to extend the *compliance warping* technique using the Sherman Morrison formula to update the precomputed inverse in case of topological modifications. However, numerical errors accumulate over time and the method suffers from instabilities after a large number of modifications. In this paper, we propose a solution that enables to significantly decrease the numerical issues caused by the cuts, and to preserve the accuracy of the contact response during the cut.

3. Background

In this section we introduce some of the necessary background on which we build our method.

3.1. Deformable model and time-stepping implicit integration

Let us consider a generic dynamic deformable model. Equations used to model the dynamic behavior of bodies can be written within a synthetic formulation, given by Newton's second law:

$$\mathbb{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbb{P}(t) - \mathbb{F}\ (\mathbf{q}, \dot{\mathbf{q}}) + \mathbb{H}(\mathbf{q})^T\ \boldsymbol{\lambda} \tag{1}$$

where $\mathbf{q} \in \mathbb{R}^n$ is the vector of generalized degrees of freedom (here, the mesh node positions), $\mathbb{M}(\mathbf{q}) : \mathbb{R}^n \mapsto \mathcal{M}^{n\times n}$ is the inertia matrix. $\mathbb{F}$ represents internal forces applied to the simulated object depending on the current state and $\mathbb{P}$ gathers external forces. $\mathbb{H}(\mathbf{q})^T$ is a function that gives the constraint directions depending on the position of the objects, and $\boldsymbol{\lambda}$ is the associated the vector of Lagrange multipliers containing the constraint force intensities (see details below).

$\mathbb{M}(\mathbf{q})$ and $\mathbb{F}\ (\mathbf{q}, \dot{\mathbf{q}})$ are derived from the physics-based deformable model. In this paper, we use the *co-rotational* formulation as a tradeoff between accuracy and computation time. In this formulation, the stiffness stiffness of the material depends on the current rotation (and thus on the current positions), which results in a geometrically non-linear elastic formulation Felippa (2000). After discretization, the mass matrix noted $\mathbf{M}$ in the following, is considered as constant and lumped (we obtain a diagonal matrix).

Collision response on mechanical objects leads to discontinuities in the velocities of the colliding points. For such discontinuous events, the acceleration is not defined: the problem belongs to the field of non-smooth mechanics. To integrate the mechanics and the non-smooth events due to contact over time, we use a *time-stepping* method based on an implicit scheme: The time step is fixed and there is no limitation on the number of discontinuities that could take place during a time step (Anitescu et al. (1999)), but low-order integration schemes should be used. This could lead to excessive dissipation if the time step is too large but it provides stable simulations. This is particularly relevant for interactive simulations involving contact with virtual devices controlled by an operator. With or without haptic feedback, the motion of the user will not be completely constrained and could lead to excessive energy input in the simulation. This is why the stability and the robustness of these types of simulations are crucial.

Let's consider the time interval $[t_i, t_f]$ which length is $h = t_f - t_i$. We have:

$$\begin{aligned} \mathbf{M}(\dot{\mathbf{q}}_f - \dot{\mathbf{q}}_i) &= \int_{t_i}^{t_f} (\mathbb{P}(t) - \mathbb{F}(\mathbf{q}, \dot{\mathbf{q}}))\, dt \;+\; h\; \mathbb{H}(\mathbf{q})^T \boldsymbol{\lambda} \qquad (2) \\ \mathbf{q}_f &= \mathbf{q}_i + \int_{t_i}^{t_f} \dot{\mathbf{q}}\; dt \qquad (3) \end{aligned}$$

To evaluate integrals $\int_{t_i}^{t_f} (\mathbb{P}(t) - \mathbb{F}(\mathbf{q}, \dot{\mathbf{q}}, t))\, dt$ and $\int_{t_i}^{t_f} \dot{\mathbf{q}}\, dt$ we chose the following implicit Euler integration scheme:

$$\begin{aligned} \mathbf{M}(\dot{\mathbf{q}}_f - \dot{\mathbf{q}}_i) &= h\,(\mathbb{P}(t_f) - \mathbb{F}(\mathbf{q}_f, \dot{\mathbf{q}}_f)) + \; h\; \mathbb{H}(\mathbf{q})^T \boldsymbol{\lambda}_f \qquad (4) \\ \mathbf{q}_f &= \mathbf{q}_i + h\dot{\mathbf{q}}_f \qquad (5) \end{aligned}$$

$\mathbb{F}$ is a non-linear function, we apply a Taylor series expansion to $\mathbb{F}$ and make a first order approximation:

$$\mathbb{F}\;(\mathbf{q}_i + d\mathbf{q}, \dot{\mathbf{q}}_i + d\dot{\mathbf{q}}) = \boldsymbol{f}_i + \frac{\boldsymbol{\delta}\mathbb{F}}{\boldsymbol{\delta}\mathbf{q}} d\mathbf{q} + \frac{\boldsymbol{\delta}\mathbb{F}}{\boldsymbol{\delta}\dot{\mathbf{q}}} d\dot{\mathbf{q}} \qquad (6)$$

This linearization is actually the first iteration of the Newton-Raphson algorithm. This single iteration is done under the assumption of a temporal coherency of the mechanical behavior; it may lead to small numerical errors in the dynamic behavior, but these errors tend to decrease at equilibrium or with null velocity. After discretization $\mathbf{B} = \frac{\boldsymbol{\delta}\mathbb{F}}{\boldsymbol{\delta}\dot{\mathbf{q}}}$ and $\mathbf{K} = \frac{\boldsymbol{\delta}\mathbb{F}}{\boldsymbol{\delta}\mathbf{q}}$ are known respectively as the damping and stiffness matrices. Replacing (6) in (4) and using $d\mathbf{q} = \mathbf{q}_f - \mathbf{q}_i = h\dot{\mathbf{q}}_f$ and $d\dot{\mathbf{q}} = \dot{\mathbf{q}}_f - \dot{\mathbf{q}}_i$, we obtain:

$$\underbrace{\left(\mathbf{M} + h\mathbf{B} + h^2\mathbf{K}\right)}_{\mathbf{A}} \underbrace{d\dot{\mathbf{q}}}_{\mathbf{x}} = \underbrace{-h^2\mathbf{K}\dot{\mathbf{q}}_i - h\left(\boldsymbol{f}_i + \boldsymbol{p}_f\right)}_{\mathbf{b}} + h\; \mathbb{H}(\mathbf{q})^T \boldsymbol{\lambda}_f \qquad (7)$$

where $\boldsymbol{p}_f$ is the value of function $\mathbb{P}$ at time t_f. The only unknown values are the Lagrange multipliers $\boldsymbol{\lambda}$ but their computation is now detailed.

3.2. Contact and friction models

Before enforcing solving the contact between soft tissues or with surgical instruments (rigid or deformable), one needs to detect them. In our work, we use two types of algorithms: the first is based on proximity queries[4], and provides the minimal distances between mesh (even concave meshes) ; the second is based on detection of volume of interpenetration (details in Allard et al. (2010)). The first algorithm have the advantage to "anticipate" the contacts before they actually appears, the second algorithm needs unnatural interpenetration between models but provide much faster results on complex meshes thanks to GPU optimizations. For simplicity, in the following, we consider that we use the algorithm based on proximities detection.

Whatever method is used for detection, the collision response is based on the same model: Signorini's law. This model is defined for each *potential* contact point provided by the contact detection algorithm. It expresses that there is a complementarity relation along the direction $\mathbf{n}$ of contact[5] between the contact force $\boldsymbol{\lambda}_n$ and the distance $\boldsymbol{\delta}_n$ between the contacting object at the point of contact, that is:

$$\mathbf{0} \leq \boldsymbol{\delta}_\mathbf{n} \perp \boldsymbol{\lambda}_\mathbf{n} \geq \mathbf{0} \tag{8}$$

This model has several physical justifications including non interpenetration and no sticking force. Moreover the contact force vanishes if the points are not strictly in contact. Using Signorini's law, the contact space is along the normal and creates a frictionless response.

Coulomb's friction law describes the macroscopic behavior in the tangent contact space. In this law, the reaction force is included in a cone whose height and direction is given by the normal force. If the reaction force is strictly included inside the cone, objects stick together, otherwise, the reaction force is on the cone's border and objects are slipping along the tangential direction. In this last case, the friction force must be directed along the direction of motion.

$$\begin{array}{l} \dot{\boldsymbol{\delta}}_\mathbf{T} = \mathbf{0} \Rightarrow \|\boldsymbol{\lambda}_\mathbf{T}\| < \mu \, \|\boldsymbol{f}_\mathbf{n}\| \ \ \text{(stick)} \\ \dot{\boldsymbol{\delta}}_\mathbf{T} \neq \mathbf{0} \Rightarrow \boldsymbol{\lambda}_\mathbf{T} = -\mu \, \|\boldsymbol{\lambda}_\mathbf{n}\| \frac{\dot{\boldsymbol{\delta}}_\mathbf{T}}{\|\dot{\boldsymbol{\delta}}_\mathbf{T}\|} = -\mu \, \|\boldsymbol{\lambda}_\mathbf{n}\| \ \mathbf{T} \ \ \text{(slip)} \end{array} \tag{9}$$

[4]The implementation is available on SOFA using the component *LocalMinDistance*.

[5]Complementarity is noted $\perp$. It states that one of the two values $\boldsymbol{\delta}_n$ or $\boldsymbol{\lambda}_n$ must be null.

where μ is the friction parameter, and $\mathbf{T}$ is the tangential plane to the contact normal $\mathbf{n}$.

During 3D slipping motion (also called *dynamic friction*), the tangential direction is unknown. We only know that the tangential force and the tangential velocity are opposite along a direction that is to be found. It creates a non-linearity in addition to the complementarity state stick/slip. Signorini's law and Coulomb's law are also valid in a *multi-contact* case. However, to solve these laws at every contact point, we have to consider the coupling that exists between these contact points. This coupling comes from the intrinsic mechanical behavior of deformable objects.

3.3. Contact mapping

From collision or proximity detection, we have a set of potential contact spots $\alpha = 1...n_c$, which are defined on the surface of the deformable bodies (triangles, lines, or points). In order to transfer the contact informations to the Degrees of Freedom of the objects, we can build a mapping function $\mathbb{A}$ that links the positions in the contact space to the motion space (see Saupin et al. (2008) for details). For each contact point between two objects:

$$\boldsymbol{\delta}_\alpha = \mathbb{A}_\alpha(\mathbf{q}_1, t) - \mathbb{A}_\alpha(\mathbf{q}_2, t) \tag{10}$$

with $\mathbb{A}_\alpha(\mathbf{q}, t)$ the mapping function which depends on contact α and the positions $\mathbf{q}_1$ and $\mathbf{q}_2$ of the two colliding objects. To obtain a kinematic relation between the two spaces (contact, motion), we use a linearization of equation (10). If $\mathbb{H}_\alpha(\mathbf{q}) = \frac{\partial \mathbb{A}_\alpha}{\partial \mathbf{q}}$, we obtain, at time t for each contact:

$$\dot{\boldsymbol{\delta}}_\alpha(t) = \mathbb{H}_\alpha(\mathbf{q}_1)\dot{\mathbf{q}}_1(t) - \mathbb{H}_\alpha(\mathbf{q}_2)\dot{\mathbf{q}}_2(t) \tag{11}$$

where $\mathbb{H}$ is a non linear function where the dimension of the results (number of constraints) depends on its parameter $\mathbf{q}$. To simplify the solution process, we suppose that this matrix does not change during the contact response $\mathbf{H}^T\boldsymbol{\lambda}_f = \mathbb{H}(\mathbf{q})^T\boldsymbol{\lambda}_f$. In the following, this matrix is noted $\mathbf{H}$ to emphasize that it is constant during the time step.

3.4. Constraint-based solution

In the following, we present how the contact laws (8) and friction laws (9) are enforced while taking into account the dynamic equation (7) between 2 contacting objects. To resolve these laws, we use a Lagrange Multiplier approach and a single linearization by time step. For both interacting objects we applied equation (7):

$$\begin{array}{l} \mathbf{A}_1\mathbf{x}_1 = \mathbf{b}_1 + h\,\mathbf{H}_1^T\boldsymbol{\lambda} \\ \mathbf{A}_2\mathbf{x}_2 = \mathbf{b}_2 + h\,\mathbf{H}_2^T\boldsymbol{\lambda} \end{array} \tag{12}$$

In order to solve $\boldsymbol{\lambda}$ we follow the following steps.

Step 1: interacting objects are solved independently while setting $\boldsymbol{\lambda} = \mathbf{0}$. A set of independent linear systems of equations $\mathbf{Ax} = \mathbf{b}$ must then be solved for each object. For our purposes, this must be done in real-time and we use a GPU-based Conjugate Gradient algorithm (CG) as described in Allard et al. (2011). This method enables using meshes comprised of thousands of elements in real-time on standard architectures, but tends to converge very slowly for ill-conditioned matrices. This is particularly an issue for the simulation of heterogeneous materials with large phase contrast or for problems obtained from finite element meshes of non optimal quality (large local difference in element size). We propose in section 4 a novel method to simulate the heterogeneities in real-time. Finally, we obtain what we call the *free motion* $\mathbf{x}_1^{\text{free}}$ and $\mathbf{x}_2^{\text{free}}$ for each object which corresponds to $d\dot{\mathbf{q}}_1^{\text{free}}$ and $d\dot{\mathbf{q}}_2^{\text{free}}$. After integration, we obtain $\mathbf{q}_1^{\text{free}}$ and $\mathbf{q}_2^{\text{free}}$.

Step 2 : the constraint laws are linearized around the *free position* (i.e. we process a the collision detection between the *free position* of the objects, and we obtain the free interpenetration $\boldsymbol{\delta}^{\text{free}}$ and the associated $\mathbf{H}$, assumed constant during the time step):

$$\boldsymbol{\delta} = \underbrace{\mathbb{A}_\alpha(\mathbf{q}_1^{\text{free}}) - \mathbb{A}_\alpha(\mathbf{q}_2^{\text{free}})}_{\boldsymbol{\delta}^{\text{free}}} + h\,\mathbf{H}_1\,d\dot{\mathbf{q}}_1^{\text{cor}} + h\,\mathbf{H}_2\,d\dot{\mathbf{q}}_2^{\text{cor}} \tag{13}$$

with $d\dot{\mathbf{q}}_1^{\text{cor}}$ and $d\dot{\mathbf{q}}_2^{\text{cor}}$ being the unknown corrective motions when solving equation (12) with $\mathbf{b}_1 = \mathbf{b}_2 = \mathbf{0}$. When gathering equations (12) and (13), we have:

$$\boldsymbol{\delta} = \boldsymbol{\delta}^{\text{free}} + \underbrace{h^2\left[\mathbf{H}_1\mathbf{A}_1^{-1}\mathbf{H}_1^T + \mathbf{H}_2\mathbf{A}_2^{-1}\mathbf{H}_2^T\right]}_{\mathbf{W}}\boldsymbol{\lambda} \tag{14}$$

With respect of the Signorini's law (equation (8)), this equation describes a LCP (Linear Complementarity Problem). If it is combined with Coulomb's law (equation (9)), we obtain a NLCP (Non-linear complementarity problem). This equation implies to evaluate the inverse of large matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ (same dimension as the number of DOFs). We propose in section 5 a novel method to obtain an approximation in real-time.

Step 3 : We obtain the value of $\boldsymbol{\lambda}$ using a Gauss-Seidel algorithm dedicated to the NLCP created by contact and friction equations. Considering a contact α, among m instantaneous contacts, one can write the behavior of the model in contact space:

$$\underbrace{\boldsymbol{\delta}_\alpha - \mathbf{W}_{\alpha\alpha}\,\boldsymbol{\lambda}_\alpha}_{\text{unknown}} = \underbrace{\sum_{\beta=1}^{\alpha-1}\mathbf{W}_{\alpha\beta}\,\boldsymbol{\lambda}_\beta + \sum_{\beta=\alpha+1}^{m}\mathbf{W}_{\alpha\beta}\,\boldsymbol{\lambda}_\beta +}_{\text{frozen}}\ \boldsymbol{\delta}_\alpha^{\text{free}} \tag{15}$$

where $\mathbf{W}_{\alpha,\beta}$ is a compliance matrix that models the coupling between contact points α and β. For each contact α, this method solves the contact equations by considering the others contact points ($\alpha \neq \beta$) as "frozen". The new value of $\boldsymbol{\lambda}_\alpha$ is given by solving Signorini's law and the Coulomb's law on this contact (see Duriez et al. (2006) for details of implementation and performance).

Step 4 : When the value of $\boldsymbol{\lambda}$ is available, the corrective motion is computed:

$$\begin{aligned} \mathbf{q}_{1,t+h} &= \mathbf{q}_1^{\text{free}} + h\Delta\dot{\mathbf{q}}_1^{\text{cor}} \quad \text{with} \quad \Delta\dot{\mathbf{q}}_1^{\text{cor}} = \mathbf{A}_1^{-1}\mathbf{H}_1^T\boldsymbol{\lambda} \\ \mathbf{q}_{2,t+h} &= \mathbf{q}_2^{\text{free}} + h\Delta\dot{\mathbf{q}}_2^{\text{cor}} \quad \text{with} \quad \Delta\dot{\mathbf{q}}_2^{\text{cor}} = \mathbf{A}_2^{-1}\mathbf{H}_2^T\boldsymbol{\lambda} \end{aligned} \tag{16}$$

We finally obtain $\mathbf{q}_{1,t+h}$ and $\mathbf{q}_{2,t+h}$, the positions of object 1 and 2 that fulfils the contact and friction laws.

In this section, we identified two important difficulties to achieve real-time computations. (i) to compute the solution of the linear system defined in equation (7), in particular for heterogeneous objects. (ii) to compute the *compliance matrix* $\mathbf{W}$ in equation (14).

These two problems are addressed in the two following sections.

4. Asynchronous preconditioner

The condition number κ of the matrix $\mathbf{A}$, is the ratio of its largest and smallest eigenvalues. For heterogeneous objects or ill-structured meshes, the condition number κ is often high which raises convergence issues for the conjugate gradient algorithm used to solve equation (7). A common technique is to use a preconditioner to reduce the condition number, ensuring a faster convergence of the algorithm. By definition, a preconditioner is an approximation of the system matrix $\mathbf{A}$, which is less costly to invert. Solving equation (7) with a preconditioner $\mathbf{P}$ can be written:

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}, \quad \text{such that} \quad \kappa(\mathbf{P}^{-1}\mathbf{A}) < \kappa(\mathbf{A}) \tag{17}$$

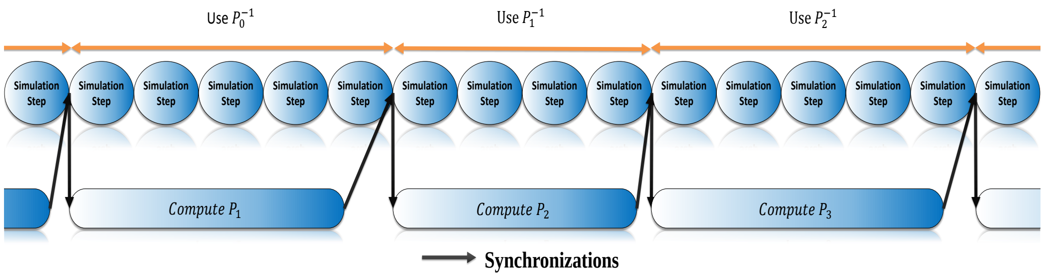


Figure 2: The preconditioner is updated asynchronously within a dedicated CPU thread. We use the last preconditioner available to advance the simulation so that the simulation never needs to wait for the computation of the current preconditioner to be complete.

In the real-time context, one strong limitation of this technique is the computational overhead added to the simulation: first, during the computation of the preconditioner itself, and second, during its use at each iteration of the CG (see Saad (2003) for details). Thus, the practical usefulness of preconditioners depends on the ability to strike a balance between the computational overheads and the time saved by decreasing the number of CG iterations. Several preconditioners can be used, from simple diagonal matrices Baraff and Witkin (1998) to precise but costly Cholesky factorizations.

We recently proposed a different approach Courtecuisse et al. (2010) that relies on the assumption that $\mathbf{A}$ undergoes small perturbations between two consecutive time steps. Indeed, if $\mathbf{P}_t = \mathbf{A}_t^{-1}$ is available at a specific time t, it may remain a "good enough" approximation for the following time steps. The preconditioner can then be updated at low frequency on a dedicated CPU thread, and the last preconditioner available can be used to advance the simulation (see fig 2). Therefore, the overhead in computing the preconditioner is removed from the simulation loop, which allows using more advanced and computationally costly preconditioners such as a factorization of the system[6]:

$$\mathbf{P} = \mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}} \tag{18}$$

where $\mathbf{D}$ is a diagonal matrix and $\mathbf{L}$ is a sparse lower-triangular matrix. In our application, we rely on $\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}$ factorization since it produce more stable results than a Cholesky factorization. The factorization is performed by

[6]Note that even if we compute an exact factorization of $\mathbf{A}_{t-h}$, the preconditioner remains an approximation since its computation is based on a previous configuration of the objects, and we use it with delay in the simulation.

the *cs_ sparse* library Davis (2006), using a single core on the CPU. Other libraries Toledo et al. (2003) and Schenk et al. (2008) propose parallel factorizations, but we found that *cs_ sparse* provides sufficiently fast updates, and a sequential factorization enables to save CPU cores to compute the preconditioners of other objects in parallel.

An important advantage of such a factorization is that the resulting $\mathbf{L}$ matrix remains sparse, which makes the application of the preconditioner faster within the CG. This operation consists in solving two Sparse Triangular Systems (STS):

$$\begin{aligned} \mathbf{y} =& \left(\mathbf{L}^{\mathrm{T}}\right)^{-1}\mathbf{b} \\ \mathbf{x} =& \mathbf{L}^{-1}\left(\mathbf{D}^{-1}\mathbf{y}\right) \end{aligned} \tag{19}$$

where $\mathbf{L}$ is stored in *Compressed Row Storage* (CRS) Barrett et al. (1994). Solving the STS is equivalent to performing a *Gauß elimination*, which is difficult to parallelize as it involves many dependencies. Therefore, the STS are solved on CPU[7] that can take advantage of caches and of the sparsity of the matrix, to efficiently solve the system.

For large systems (see section 7.2.2), the computation of the $\mathbf{LDL}^{\mathrm{T}}$ factorization can become prohibitively costly, and the resulting preconditioner can diverge from the actual system. However, we note that an important part of the error is associated with the rotations Saupin et al. (2008) which can vary quickly between time steps. In order to limit the divergence of the preconditioner, we estimate the nodal[8] rotations $\mathbf{R}_{t-h\to t}$ that were introduced since the last update of the preconditioner (i.e. between time $t-h$ and t). The most recent preconditioner $\mathbf{P}_{t-h}$ is then rotated with the current rotation matrix $\mathbf{R}_{t-h\to t}$ as follows:

$$\mathbf{P}_r = \mathbf{R}^{\mathrm{T}}{}_{t-h\to t}\ \left(\mathbf{L}_{t-h}\mathbf{D}_{t-h}\mathbf{L}^{\mathrm{T}}{}_{t-h}\right)\ \mathbf{R}_{t-h\to t} \tag{20}$$

where the "rotated preconditioner" $\mathbf{P}_r$ is less sensible to geometrical nonlinearities. Finally, the method enables to simulate the deformation of homogeneous as well as heterogeneous tissues in real-time.

[7] Using a GPU-based CG with a CPU-based preconditioner, implies to transfer the solution vector $\mathbf{b}$ between the CPU and the GPU at each iteration of the CG. But, since the preconditioner is usually a good approximation of the actual system, only a few iterations are necessary and the cost of such transfers remains limited.

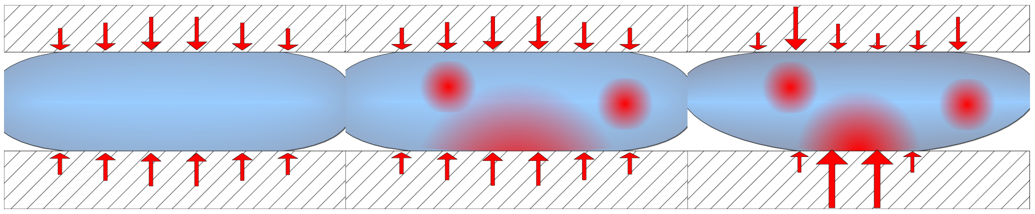
[8] Note that nodal rotations give an approximation of the co-rotaitonal formulation where rotations are computed per elements and sum in the global stiffness matrix. $\mathbf{R}_{t-h\to t}$ is a block diagonal matrix, easy to compute and easy to invert.

5. GPU-Based preconditioner for contact problems

Equation (14) requires the computation of $\mathbf{A}^{-1}$, which is a large matrix (same dimension as the number of DOF) and changes at each time step. Although computing this inverse in real-time is not possible, the resulting operator $\mathbf{W}$ plays an important role to enforce the constraints.

5.1. Compliance and mechanical coupling

In the following, the term "*mechanical coupling*" describes the coupling between contact constraints applied on two subsets of the boundary of a deformable body. This coupling occurs through the deformation of the body itself. Indeed, even if the contact points are only defined on the boundary of the deformable bodies, they are all influenced by each other through the stiffness of the material. Consider for example figure 3 where, even if the deformed shapes do not show any interpenetration, the behaviour computed in figure 3(b) is not acceptable, since stiff and soft parts deform in the same way. It shows that the contact force distribution is closely related to the underlying heterogeneity of the material, which is represented by $\mathbf{W}$.



(a) homogeneous object (b) heterogeneous object without coupling (c) heterogeneous object with coupling

Figure 3: Contact force distribution in different scenarii and using different approximation of the *mechanical coupling*. Contact forces are dependent on the stiffness areas (3(a) and 3(c)), but also on the *mechanical coupling* (3(b) and 3(c)).

In the context of explicit schemes matrix $\mathbf{W}$ would only be built from a diagonal mass matrix. If penalty methods are used, the force distribution would mainly depend on the geometrical interpenetration, not on the inhomogeneities. In both cases, it would lead to unrealistic configurations such as in figure 3(b), at least during transient states. Precomputing the inverse $\mathbf{A}_0$ at the initial step, and using it all along the simulation provides better results, but is limited to linear, small displacements models. Another approach adapted to the real-time context, is to use an approximation of $\mathbf{W}$. For instance, Saupin et al. (2008) proposed the *compliance warping* technique, that consists in updating the precomputed ${\mathbf{A}_0}^{-1}$ with the nodal rotations, but this solution remains inaccurate for large deformations, and requires storing a large dense matrix which makes the method unsuitable for fine meshes.

In this paper, we propose to reuse the asynchronous preconditioner of the previous section as an approximation of the compliance operator $\mathbf{W}$, and we detail our GPU-Based algorithm that allows real time computations. A fundamental improvement of our technique is its ability to keep large time steps even during contact enforcement. This point is particularly relevant given our assumption in equation (6) that a single iteration of the Newton-Raphson algorithm is performed at each time step (for the deformation as well as for the contact response).

5.2. *Optimized preconditioner for contacts*

We propose to use the asynchronous preconditioner computed in section 4 as an approximation of $\mathbf{A}^{-1}$:

$$\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^{\mathrm{T}} \approx \mathbf{H}\mathbf{P}^{-1}\mathbf{H}^{\mathrm{T}} \tag{21}$$

Indeed, since $\mathbf{P}$ represents a close approximation of the factorization of $\mathbf{A}$, we propose to use it to compute $\mathbf{W}$ in equation (14). For each interacting object, substituting equation (20) in (21) gives:

$$\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^{\mathrm{T}} \approx \mathbf{H}\left(\mathbf{R}\,\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\right)^{-1}\mathbf{H}^{\mathrm{T}} \tag{22}$$

The above equation requires computing the product of the inverse of the preconditioner with the Jacobian of contacts $\mathbf{H}$, which can be achieved by computing columns independently of $\mathbf{H}$:

$$\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}\,\mathbf{X}_i = \mathbf{H}_i \quad \Leftrightarrow \quad \mathbf{X}_i = \left(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}\right)^{-1}\mathbf{H}_i \tag{23}$$

where $\mathbf{X}$ gives the result of the inverse of the preconditioner times the Jacobian of the contact. Therefore, we obtain $\mathbf{W}$ within 4 steps as detail in algorithm 1. Step 1 and 3 are inexpensive because $\mathbf{H}$ and $\mathbf{J}$ are sparse matrices, $\mathbf{D}$ is a diagonal matrix, and $\mathbf{R}$ is a block-diagonal matrix. Step 4 involves the product of two dense matrices, which can be parallelized efficiently on GPU using the CUBLAS library. Step 2 requires the solution of a STS for each column of $\mathbf{J}$, composed of the lower triangular matrix $\mathbf{L}$. This operation remains the most expensive task and would quickly become too prohibitive if it was processed sequentially on a traditional CPU. Therefore, we propose to parallelize the computation of $\mathbf{S}$ on GPU.

```
1.  $\mathbf{J} = \mathbf{H}\,\mathbf{R}$                                   (rotate constraints)
2.  $\mathbf{S} = \mathbf{L}^{-1}\,\mathbf{J}^{\mathrm{T}}$                  (solve a single STS for each column of $\mathbf{J}$)
3.  $\mathbf{T} = \mathbf{D}^{-1}\,\mathbf{S}$                               (apply the diagonal)
4.  $\mathbf{W} = \mathbf{W} + \mathbf{S}^{\mathrm{T}}\,\mathbf{T}$          (sum the contributions)
```

Algorithm 1: Optimized algorithm to accumulate the contributions of a deformable on $\mathbf{W}$ using the asynchronous preconditioner (i.e. $\mathbf{W} = \mathbf{W} + \mathbf{H}\left(\mathbf{R}\,\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\right)^{-1}\mathbf{H}^{\mathrm{T}}$).

5.3. *Solution of multiple STS on GPU*

Implementation-wise, the simplest solution to solve the multiple STS on GPU is to use CUSPARSE library. Nevertheless, this library is optimized for solving large and very sparse systems of equations (see Naumov (2011)), but the range of size of the matrices compatible with the real-time constraint is still much smaller. Therefore, to decrease the computational time of this critical step, we propose a solution to parallelize the computation of $\mathbf{S}$ on GPU. Our solution is based on a two level parallelization strategy which is inspired from the method introduced in Courtecuisse and Allard (2009). The main difference is that the underlying $\mathbf{L}$ matrix is stored in a sparse format, which makes it difficult to load efficiently on the GPU processors.

5.3.1. *GPU-based parallelization*

The multiple right-hand side vectors stored in $\mathbf{J}$ can be computed independently from each other. Therefore, we assign the computation of each column of $\mathbf{S}$ to an independent multiprocessor on the GPU. Each group is therefore fully processed by a single processor (see fig. 4) which enables to use fast local synchronizations directly on the GPU. Then we use a second level of parallelism where each STS is solved with several threads. Indeed, a lot of data can potentially be treated in parallel during the solving process of each STS. This two level strategy fits the GPU architectures where local synchronizations within a group of threads are fast, whereas global synchronizations over multiple groups are much more costly.

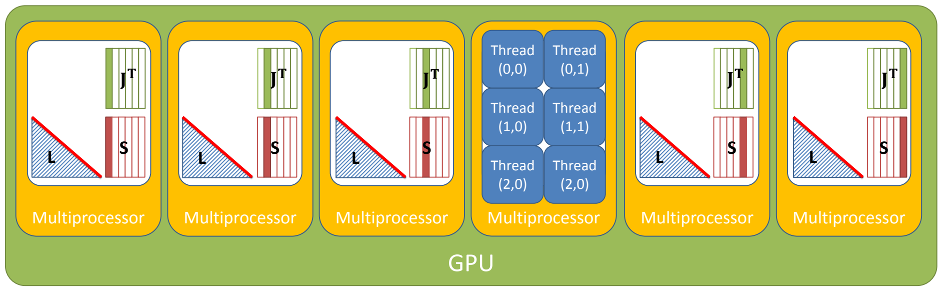


Figure 4: First level of parallelism achieved for solving a Sparse Triangular System with multiple right hand side vector on GPU.

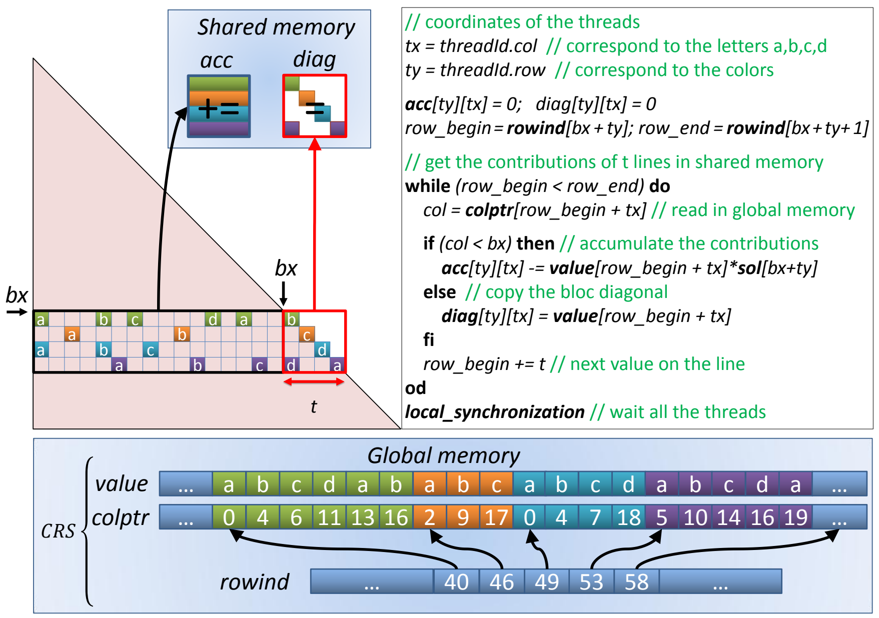
Nevertheless, as mentioned above, solving a STS involves a number of dependencies. For instance, for the lower triangular system, the computation of the solution $\mathbf{s}_j$ of a given row j, requires the result of all previous solutions $\mathbf{s}_i$ such as $i < j$:

$$\mathbf{s}_j = \mathbf{b}_j - \sum_{i=0}^{i<j} (\mathbf{s}_i \, \mathbf{L}_{j,i}) \tag{24}$$

where $\mathbf{b}$ is the solution vector and $\mathbf{s}$ is the unknown. Therefore, each row must be processed sequentially (i.e. a synchronization of each row is necessary). However, the partial contributions $\sum_{i=0}^{i<j} (\mathbf{s}_i \, \mathbf{L}_{k,i})$ could be processed simultaneously for any row k such as $k > j$. We propose a block-row[9] parallelization strategy, that is inspired from the block-column scheme introduced in Courtecuisse and Allard (2009): First we accumulate the contributions of the block off-diagonal in parallel; Then we solve the block diagonal sequentially (see fig. 5).

We use a group of $t \times t$ threads to process t rows simultaneously (each row is therefore treated by t threads in parallel). Since $\mathbf{L}$ is sparse, it is difficult to predict if a data is located on the bloc diagonal before actually reading it in global memory. To avoid reading twice the CRS matrix, we propose to use two buffers *acc* and *diag* stored in shared memory: *acc* is used to accumulate the contributions off-digonal whereas *diag* is used to copy the data located on the block-diagonal (see fig. 5). A first local synchronization is then used to ensure that the t rows are fully processed by all the threads.

[9]Block-row strategy instead of a block-column as in Courtecuisse and Allard (2009), because with a block-column solution, writing conflict in memory cannot be predicted due to the CRS format of $\mathbf{L}$.



```
// coordinates of the threads
tx = threadId.col  // correspond to the letters a,b,c,d
ty = threadId.row  // correspond to the colors

acc[ty][tx] = 0;   diag[ty][tx] = 0
row_begin = rowind[bx + ty]; row_end = rowind[bx + ty + 1]

// get the contributions of t lines in shared memory
while (row_begin < row_end) do
  col = colptr[row_begin + tx] // read in global memory
  if (col < bx) then // accumulate the contributions
    acc[ty][tx] -= value[row_begin + tx]*sol[bx+ty]
  else  // copy the bloc diagonal
    diag[ty][tx] = value[row_begin + tx]
  fi
  row_begin += t // next value on the line
od
local_synchronization // wait all the threads
```

Figure 5: Parallel accumulation of the contributions for solving a STS described by the CRS matrix. $t \times t$ threads (illustrated here with $t = 4$) are used such that t rows are processed simultaneously (colours). Each being accumulated by t threads in parallel (letters a,b,c,d).

Then, a parallel reduction (see Martin et al. (2012)) is processed to sum per row the contributions stored in *acc* (row 4 in algorithm 2), then a second local synchronization is necessary (row 5). Finally, the off-diagonal block is solved as a dense problem[10] using *diag* in shared memory (row 5).

In our experiments we use $t = 16$ and only 3 local synchronizations are necessary to solve 16 rows with 256 GPU threads. Although this implementation may be slower than a CPU-based solver for a single STS, our GPU-based strategy enables to solve the multiple right-hand side vectors simultaneously (i.e with the same cost as a single STS).

[10]The block-diagonal is solved using a column-based strategy to avoid the need of parallel reductions. This is possible since *diag* is stored in dense format.

```
1 $bx = 0$ **repeat**
2     $accumulate\_contributions(acc,diag)$ //see Fig. 5;
3     **local_synchronization**;
4     $cont[ty] = \sum_{i=0}^{t} acc[ty][i]$ //Parallel reduction;
5     **local_synchronization**;
6     $solve\_bloc\_diagonal(cont,diag)$ //see Courtecuisse and Allard (2009);
7     **local_synchronization**;
8     $bx = bx + t$ //We treat the next $t$ rows in parallel;
9 **until** $bx < dim$;
```
Algorithm 2: Algorithm used to solve a Sparse Triangular System with multiple right-hand side vectors on GPU.

6. Simulation of cutting

Simulation of cutting involves two main issues: First to re-mesh the FE structure while keeping the consistency of the mesh i.e. split correctly the domain in order to be able to separate the cut parts in the simulation, and avoid degenerated elements such as sharp or thin elements. The second issue is to update adequately the mechanical properties and the equation systems of the deformable model when the mesh is cut. In this paper, we only address this second aspect and we show how to update the asynchronous preconditioner according to the topological changes.

6.1. Topological modifications, cutting

The method presented in this paper could be used with any re-meshing algorithm, as far as the modifications remain local and only affect few elements per time step. In our simulations, we use a re-meshing algorithm similar to Mor and Kanade (2000), where, rather than reconstructing the overall mesh, we incrementally update it within 3 steps: First we remove the intersected elements from the current mesh; Second we subdivide the removed elements; Third, we add back the subdivided elements.

The subdivision process affects the *stiffness, mass* and *damping* matrices (see fig. 6), and the final linear system $\mathbf{A}$ (i.e. equations (7),(18) and (22)). Nevertheless, the conjugate gradient (CG) used to solve equation (7) only requires performing matrix vector products. Thus, $\mathbf{A}$ can be directly evaluated from the finite element (FE) mesh to instantly take into account the modifications in the solution. However, the preconditioner used in equations (18)

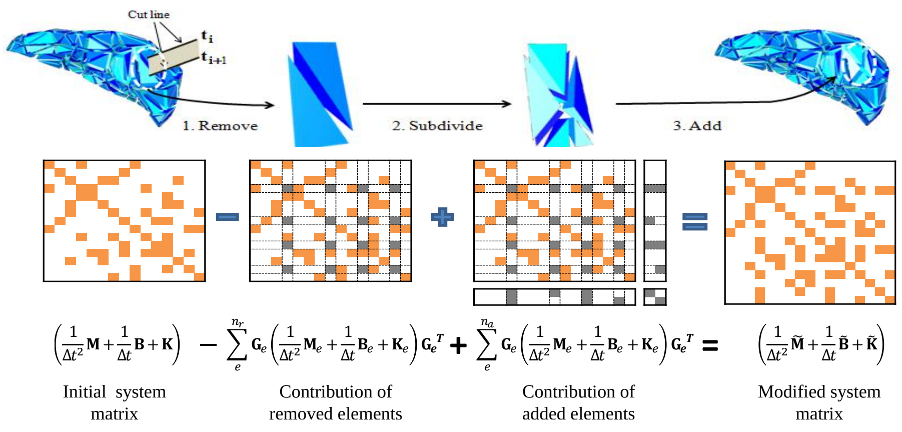

Cut line
t_i
t_{i+1}
1. Remove
2. Subdivide
3. Add
$\left(\frac{1}{\Delta t^2}\mathbf{M}+\frac{1}{\Delta t}\mathbf{B}+\mathbf{K}\right) - \sum_e^{n_r} \mathbf{G}_e\left(\frac{1}{\Delta t^2}\mathbf{M}_e+\frac{1}{\Delta t}\mathbf{B}_e+\mathbf{K}_e\right)\mathbf{G}_e^T + \sum_e^{n_a} \mathbf{G}_e\left(\frac{1}{\Delta t^2}\mathbf{M}_e+\frac{1}{\Delta t}\mathbf{B}_e+\mathbf{K}_e\right)\mathbf{G}_e^T = \left(\frac{1}{\Delta t^2}\tilde{\mathbf{M}}+\frac{1}{\Delta t}\tilde{\mathbf{B}}+\tilde{\mathbf{K}}\right)$
Initial system matrix
Contribution of removed elements
Contribution of added elements
Modified system matrix

Figure 6: Incremental update of the mesh structure for the cut.

and (22) is updated with delay, and the modifications significantly affect its efficacy. Moreover, contrary to equation (18) where the preconditioned CG ensures the convergence of the system, the preconditioner is directly used to build an approximation of the contact problem in equation (22). Therefore, the delay of the updates can lead to instabilities and inaccuracies, in particular when treating contact with the instruments and self-collisions between different parts of the cut.

6.2. Low rank update of the "Asynchronus Preconditioner"

We propose to use the Sherman Morrison Formula (SMF) to compute the correction of the preconditioner due to the topological changes:

$$\begin{aligned}\widetilde{\mathbf{P}}^{-1} &= (\mathbf{P}+\mathbf{G}\mathbf{N}\mathbf{G}^{\mathrm{T}})^{-1}\\ &= \underbrace{\mathbf{P}^{-1}}_{\text{Last factorization}} - \underbrace{\mathbf{G}\mathbf{P}^{-1}\left(\mathbf{N}^{-1}+\mathbf{G}\mathbf{P}^{-1}\mathbf{G}^{\mathrm{T}}\right)^{-1}\mathbf{P}^{-1}\mathbf{G}^{\mathrm{T}}}_{\text{Correction due to the cut}}\end{aligned} \tag{25}$$

where $\widetilde{\mathbf{P}}$ is the modified preconditioner, $\mathbf{G}$ is a globalization matrix which maps the rows/columns to the global system and $\mathbf{N}$ is the perturbation of the preconditioner obtained as a difference between the modified system, and the last preconditioner:

$$\mathbf{N} = \mathbf{G}^{\mathrm{T}}\widetilde{\mathbf{P}}\mathbf{G} - \mathbf{G}^{\mathrm{T}}\begin{pmatrix}\mathbf{P} & 0\\ 0 & \mathbf{I}\end{pmatrix}\mathbf{G} \tag{26}$$

In order to keep a consistent formulation, $\mathbf{P}$ is padded with the identity for all the added nodes during the subdivision process. This approach assumes that the added degrees of freedom were present before the cut with a unitary mass but not attached to the mechanical system, and $\mathbf{N}$ corresponds to the correction of the padded system. $\mathbf{N}$ is fast to compute since it only involves subtractions of small matrices associated with the nodes affected by the cut.

An important advantage of updating the preconditioner is that it helps maintain the number of affected nodes by the perturbations minimum. Indeed, each new factorization implicitly contains all anterior modifications to the last update of the preconditioner. The SMF correction is therefore necessary only for the perturbation that appeared since the updates, and the correction only involves few affected nodes (see fig. 7).

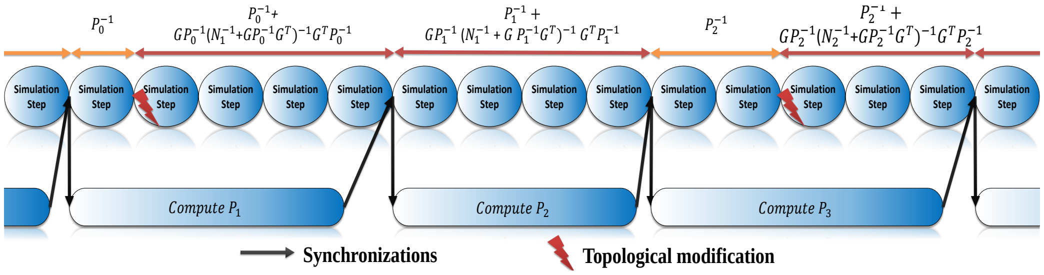

P_0^{-1}
$P_0^{-1}+ GP_0^{-1}(N_1^{-1}+GP_0^{-1}G^T)^{-1}G^TP_0^{-1}$
$P_1^{-1} + GP_1^{-1}(N_1^{-1}+GP_1^{-1}G^T)^{-1}G^TP_1^{-1}$
P_2^{-1}
$P_2^{-1} + GP_2^{-1}(N_2^{-1}+GP_2^{-1}G^T)^{-1}G^TP_2^{-1}$
Simulation Step
Compute P_1
Compute P_2
Compute P_3
Synchronizations
Topological modification

Figure 7: Correction of the preconditioner during topological modifications. When a modification is performed on the mesh, we first compute the correction of the current factorization. Then we compute the correction of the preconditioner which was being calculated at the time of the cut. After two consecutive updates without topological modification, the preconditioner does not need any additional correction.

Contrary to Courtecuisse et al. (2011), we do not store the dense inverse of $\mathbf{W}$. Instead we use the sparse $\mathbf{LDL}^{\mathrm{T}}$ factorization of $\mathbf{A}$. The SMF cannot be directly applied with such a factorization because it explicitly requires the inverse of $\mathbf{W}$. Thus, we proceed in two steps: First we compute the correction for the preconditioner, which is only necessary when a new topological modification is performed; Then we apply the correction until the next update of the preconditioner.

6.3. Computation of the correction

For each new topological modification, the correction is obtained by computing the two following matrices:

$$\mathbf{U} = \left(\mathbf{LDL}^{\mathrm{T}}\right)^{-1}\mathbf{G}^{\mathrm{T}} \tag{27}$$

$$\mathbf{Q} = \left(\mathbf{N}^{-1} + \mathbf{GU}\right)^{-1} \tag{28}$$

The computation of $\mathbf{U}$ involves the product of the inverse of the preconditioner with the globalization matrix $\mathbf{G}$. This implies to solve equations composed of the lower and upper triangular systems (see algorithm 3). Since, the CRS format used to store $\mathbf{L}$ prevents access to matrix $\mathbf{L}^{\mathrm{T}}$ by columns, and in order to use the same parallelization scheme as described in section 5.2, the transpose matrix $\mathbf{L}^{\mathrm{T}}$ is explicitly computed by the asynchronous thread after the factorization of the preconditioner.

```
1.  $\mathbf{S} = \mathbf{L}^{-1}\,\mathbf{G}^{\mathrm{T}}$    (solve the lower STS for each column of $\mathbf{G}$)
2.  $\mathbf{T} = \mathbf{D}^{-1}\,\mathbf{S}$    (apply the diagonal)
3.  $\mathbf{U} = \mathbf{L}^{-\mathrm{T}}\,\mathbf{T}$    (solve the upper STS for each column of $\mathbf{G}$)
```

Algorithm 3: Computation of $\mathbf{U}$ in the correction of the SMF.

Equation (28) requires to compute the inverse of two small matrices ($\mathbf{N}$ and $\mathbf{N}^{-1} + \mathbf{GU}$) which are performed on CPU. As a difference between two mechanical matrices, $\mathbf{N}$ is ill-conditioned. Its inverse is therefore ill-defined, but contrary to Courtecuisse et al. (2011), our method does not lead to the accumulation of subsequent round-off errors. Indeed, as soon as a new factorization is released the perturbation is include in the new preconditioner, and $\mathbf{N}$ is erased so that these numerical errors do not accumulate over time.

6.4. Application of the correction

Once $\mathbf{U}$ and $\mathbf{Q}$ are computed for a given modification, we use them to correct the preconditioner until the next update. However, the local rotations used in equation (20) and (22) vary at each time step, which prevents including them in the formulation of the correction. In order to use the rotations to improve the efficacy of the preconditioner, we apply them around the corrected formulation. Substituting (25), (27), (28) in (18) gives:

$$\widetilde{\mathbf{P}}^{-1} \approx \mathbf{R}\left(\mathbf{P}^{-1} - \mathbf{U}^{\mathrm{T}}\mathbf{Q}\mathbf{U}\right)\mathbf{R}^{\mathrm{T}}. \tag{29}$$

The rotations are therefore applied around the final corrected solution. To simplify notations we do not carry those rotations in the upcoming equations. For each iteration of the preconditioned CG, the application of the preconditioner is corrected:

$$\begin{aligned} \widetilde{\mathbf{P}}\mathbf{x} = \mathbf{b} \Leftrightarrow \mathbf{x} &= \left(\mathbf{P}^{-1} - \mathbf{U}^{\mathrm{T}}\,\mathbf{Q}\,\mathbf{U}\right)\mathbf{b} \\ \mathbf{x} &= \underbrace{\mathbf{P}^{-1}\mathbf{b}}_{\mathbf{x}^{\mathrm{a}}} - \underbrace{\mathbf{U}^{\mathrm{T}}\,\mathbf{Q}\,\mathbf{U}\,\mathbf{b}}_{\mathbf{x}^{\mathrm{c}}}, \end{aligned} \tag{30}$$

where $\mathbf{x}^{\mathrm{c}}$ is the correction of the solution, which involves 3 dense matrix–vector products, and are performed on GPU using the CUBLAS library. $\mathbf{x}^{\mathrm{a}}$ is equivalent to applying the preconditioner without correction as in equation (18), except that if nodes are added during the subdivision process, $\mathbf{P}$ is padded with the identity:

$$\begin{pmatrix} \mathbf{x}^{\mathrm{a}}_m \\ \mathbf{x}^{\mathrm{a}}_a \end{pmatrix} = \begin{pmatrix} \mathbf{P}^{-1} & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{b}_m \\ \mathbf{b}_a \end{pmatrix}, \tag{31}$$

where subscripts m and a correspond to the modified and added nodes, respectively.

For the contact response, the corrected *Delasus operator* $\mathbf{W}$ is obtained with:

$$\begin{aligned} \mathbf{H}\widetilde{\mathbf{P}}\mathbf{H}^{\mathrm{T}} &= \mathbf{H}\left(\mathbf{P}^{-1} - \mathbf{U}^{\mathrm{T}}\,\mathbf{Q}\,\mathbf{U}\right)\mathbf{H}^{\mathrm{T}} \\ &= \underbrace{\mathbf{H}\mathbf{P}^{-1}\mathbf{H}^{\mathrm{T}}}_{\mathbf{W}^{\mathrm{a}}} - \underbrace{\mathbf{H}\,\mathbf{U}^{\mathrm{T}}\,\mathbf{Q}\,\mathbf{U}\,\mathbf{H}^{\mathrm{T}}}_{\mathbf{W}^{\mathrm{c}}}, \end{aligned} \tag{32}$$

where $\mathbf{W}^{\mathrm{a}}$ is the asynchronous *Delasus operator*, and $\mathbf{W}^{\mathrm{c}}$ its correction. As in equation (31), $\mathbf{H}$ may involve constraints on the newly created degrees of freedom, so that we pad $\mathbf{P}$ with the identity matrix, and $\mathbf{W}^{\mathrm{a}}$ is computed by:

$$\mathbf{W}^{\mathrm{a}} = \begin{pmatrix} \mathbf{H}_m & \mathbf{H}_a \end{pmatrix} \begin{pmatrix} \mathbf{P}^{-1} & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{H}_m \\ \mathbf{H}_a \end{pmatrix} = \begin{pmatrix} \mathbf{H}_m\mathbf{P}^{-1}\mathbf{H}^{\mathrm{T}}{}_m & 0 \\ 0 & \mathbf{H}_a\mathbf{H}_a{}^{\mathrm{T}} \end{pmatrix} \tag{33}$$

$\mathbf{H}_m\mathbf{P}^{-1}\mathbf{H}^{\mathrm{T}}{}_m$ is solved using the GPU-based algorithm introduced section 5.2, whereas $\mathbf{H}_a\mathbf{H}_a{}^{\mathrm{T}}$ involves a sparse matrix product which is parallelized using the CUSPARSE library. Finally, $\mathbf{W}^{\mathrm{c}}$ is obtained by 3 small dense matrix products which are performed on GPU using the CUBLAS library.

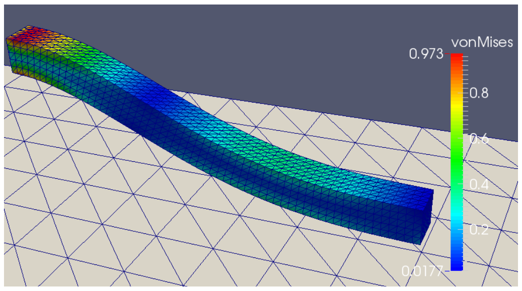
7. Results and evaluation

In this paper, we proposed a method to significantly improve the trade-off between accuracy and computation time of an interactive simulation. In the first part of this section, we compare the accuracy of our method versus standard algorithms. In the second part of this section we measure the performance of each aspects of our method.

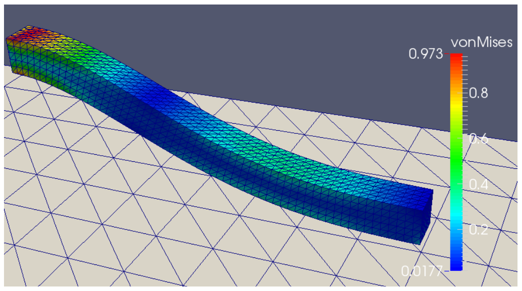
7.1. Accuracy and validation

7.1.1. Comparaison with ABAQUS

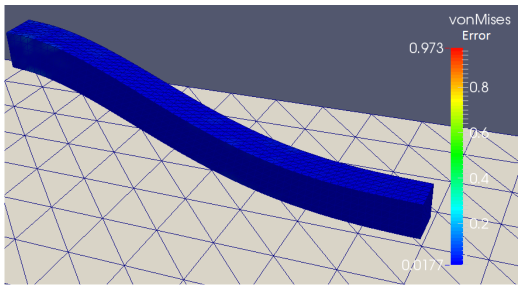
Some comparisons of algorithms available in SOFA with analytical solutions can be found in Nesme et al. (2005); Marchal et al. (2008). In order to provide a more thorough comparison for our method, in particular in the case of contacts, we produced a simulation involving a flexible beam attached at one extremity and deforming under gravity until the beam contacts with a plane. The dimension of the beam is $120 \times 10 \times 10$ mm, the mesh is composed of $10,779$ linear tetrahedral elements and $2,481$ nodes. The constitutive law is based on small strain with large displacement (co-rotational), the mass density is set at 0.0001 Kg/mm^3, the young modulus is set at 10 Mpa and the Poisson ration at 0.4.



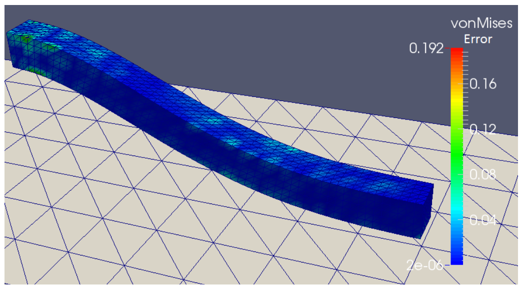
(a) Simulation with ABAQUS



(b) Simulation with SOFA.



(c) Difference of stresses between ABAQUS and SOFA



(d) Scaled difference of stresses between ABAQUS and SOFA

Figure 8: Comparison of our SOFA implementation of a co-rotational model in contact with the ABAQUS solution.

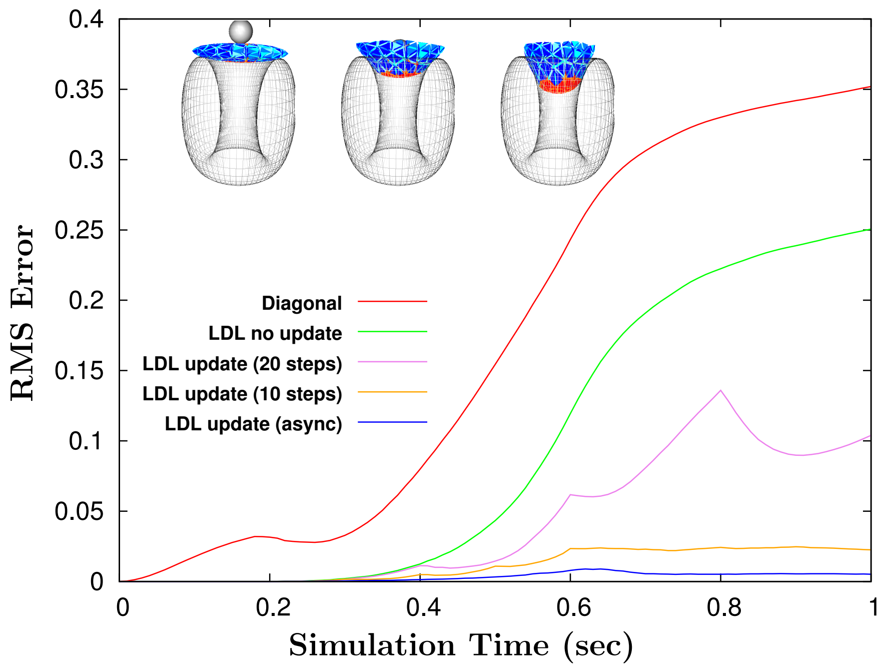
The same simulation was computed by both SOFA and ABAQUS (Simulia, Dassault Systèmes S.A.) using the same boundary conditions and assuming no friction during the contact. At equilibrium we computed the *Von Mises Stress* of the ABAQUS solution (see fig 8(a)) and of our SOFA implementation (see fig 8(b)). Then we measured the difference of the stresses computed by the two approaches. Results are presented in figure 8(c) (using the same scale as the initial stresses) and in figure 8(d) using a normalized scale, i.e. where red parts correspond to the main difference (to emphasize areas where even a small difference exists). Although a maximum error of 20% is found near the constrained part of the beam, the average stress error is equal to 0.017 (around 1.7%), which is almost negligible. Finally, the positions of the beams match perfectly, and the maximum distance between SOFA nodes and ABAQUS nodes, is less than 0.37 mm. However, ABAQUS simulation required more than 20 minutes to be computed whereas our SOFA simulation runs at 10 FPS, and require less than 1 minutes to reach the equilibrium.

7.1.2. Preconditioner for the contact response

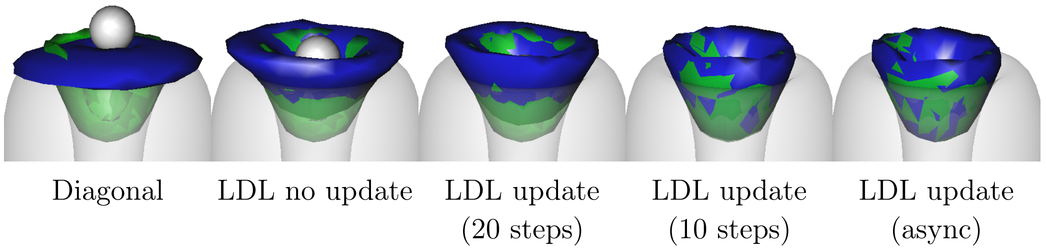
We now measure the error introduced when using our asynchronous preconditioner as an approximation of the compliance matrix defined in equation (14). We created a simulation involving an heterogeneous disk which is driven by a sphere through the center of a torus (see Figure 9). This is a very good test for measuring both the deformation process and the contact response. We first produced a reference simulation where the exact compliance matrix is computed by inverting $\mathbf{A}$ at each time step (which is obviously not real-time), and we measured the distances between the nodal positions obtained by using an "approximated compliance matrix" and the reference simulation.

Relying on the diagonal of $\mathbf{A}^{-1}$ to build $\mathbf{W}$ (**Diagonal**), as done by Dequidt et al. (2009) for the vessel wall compliance, the lens cannot be pushed into the cavity. Indeed, the forces exerted by the ball are located at the center (stiff part), while other forces are applied by the cylinder on the periphery (soft part) and hold the object up. Without coupling, the contact forces applied onto the stiff part are not transmitted to the periphery, and the lens remains in equilibrium without being deformed. The precomputed inverse of the compliance matrix (**LDL no update**), gives better results when the object is in a similar configuration as the rest position (i.e. in the state where the precomputed inverse has been computed). However, the rigidity is not properly evaluated in case of large deformations, and a large error is introduced in the behaviour.

When the preconditioner is updated periodically after a fixed number



(a) Root Mean Square error by using different approximation as compliance matrix.



(b) Simulated soft disk (blue) with reference (green) after 0.8 second of simulation.

Figure 9: Distance of nodal positions by using different approximations of the compliance matrix compared to a reference simulation. The reference simulation is obtained by computing the compliance matrix as the exact inverse of the system matrix every time step.

of time steps (**LDL update 20** and **LDL update 10**), the error tends to decrease with the frequency of the updates, and for the version where the preconditioner is updated every 10 time steps there is no visible error (see fig. 9(b)). Finally, using the asynchronous version (**LDL update async**), the preconditioner is updated on average every 5 time steps and the error introduced throughout the simulation is negligible.

7.2. Performance evaluation

7.2.1. Convergence rate and computation time

We produced a simulation where an heterogeneous beam composed of $6,500$ elements and $1,470$ nodes, is deform under gravity. We evaluated the computational time and the convergence rate of standard solvers to solve equation (7) (see fig. 10). In all our experiments we set the tolerance of the preconditioned CG to 10^{-7}, and we use a time step $h = 0.02$.

Direct solvers must process a complete factorization of the matrix at each time step, to take into account the non linearities of the model. This operation is expensive but it can be parallelized using optimized libraries such as Pardiso Schenk et al. (2008). However, due to the multiple dependencies and the relatively small size of the system, a parallel version with 2 threads (**Pardiso-2**) is only 1.5× faster than a sequential approach (**Pardiso-1**). With 4 threads (**Pardiso-4**) only 2.2× faster, and with 8 threads version (**Pardiso-8**) only 1.90× faster (and slower than **Pardiso-4**).

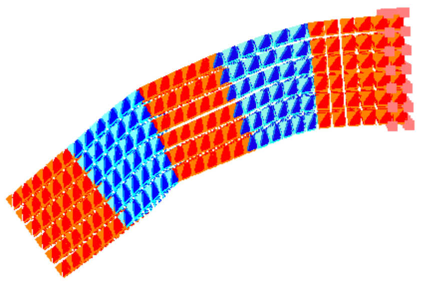
Standard iterative solvers (**standard**) require a large number of iterations, and thus a large computational time, due to the strong heterogeneity of the material. Indeed, although our GPU-based CG does not need to assemble the matrix and provides fast iterations, an average of 493 iterations are necessary to obtain a sufficient solution and the computation time is no longer compatible with real-time. The Jacobi preconditioner is too simple and does not manage to sufficiently reduce the number of iterations.

Pre-computing the $\mathbf{LDL}^{\mathrm{T}}$ factorization (**no update**) and using it throughout the simulation, enables to remove the overhead of the factorization while keeping a limited number of iterations. However in cases of large deformations, the actual stiffness of the material may be very different from the rest configuration, and a large number of iterations are necessary. Indeed, when the beam undergoes large deformations, we measured a maximum of 121 iterations necessary to achieve convergence, and the application time of the preconditioner was around 210 ms. Interestingly, applying the rotation around the preconditioner (*warping method*) helps to sensibly reduce the number of iteration with a limited overhead. Indeed, the unitary cost of a single iteration with the warping method is only 11% higher whereas it requires 3.5 times fewer iterations to converge.

When using our method where the $\mathbf{LDL}^{\mathrm{T}}$ factorization is updated asynchronously (**Async**), the factorization time of the preconditioner is still neg-

		Method	Iterations	Computational time (ms)		
				Inverse	Solving	Total
Direct Solvers		Pardiso-1	1	78.01	1.99	80.37
		Pardiso-2	1	50.81	1.27	52.42
		Pardiso-4	1	35.34	1.05	36.80
		Pardiso-8	1	40.89	1.70	43.05
Iterative Solvers	Stan-dard	CG	493.66	0.01	64.17	64.41
		Jacobi	314.28	14.20	53.43	67.85
	No update	**LDL**$^{\mathrm{T}}$	59.11	0.02	87.86	88.16
		LDL$^{\mathrm{T}}$ + warp	15.12	0.24	23.22	23.74
	Async	**LDL**$^{\mathrm{T}}$	9.16	0.03	14.44	15.27
		LDL$^{\mathrm{T}}$ + warp	6.85	0.29	11.29	12.56

(a) Average computational time for 200 simulation steps and convergence rate for different preconditioners. **Inverse** corresponds to the inversion of the diagonal matrix for the Jacobi preconditioner, and to the factorization of the system for **LDL**$^{\mathrm{T}}$ preconditioners. **Solving** is the time taken to solve the system. **Total** is the total time of a single time step.



(b) Heterogeneous beam falling under gravity. Red parts are 50× stiffer than bleu parts.

Figure 10: Simulation of an heterogeneous deformable object with different preconditioners.

ligible, and the number of iterations remains very low throughout the simulation. For this preconditioner, the (*warping method*) provides only limited improvements. Indeed, the preconditioner was updated on average every 8 time steps, and the rotations between two consecutive updates remained limited. The warping heuristic is therefore particularly beneficial if the preconditioner cannot be updated sufficiently fast (for larger systems for instance). Finally, the asynchronous solution is the only method which is compatible with real-time computations. Indeed, the simulation runs at 45 FPS, and 1 second in the simulation is simulated in 1 second.

7.2.2. Factorization of the preconditioner

To test the scalability of our method, we evaluate the computational time required to factorize the mechanical matrices with dimensions up to $10,000 \times 10,000$ (which corresponds to an object with $3,333$ nodes with 3 dofs per node). This operation is very expensive, but since we process it asynchronously it does not impact directly the performances of the simulation.

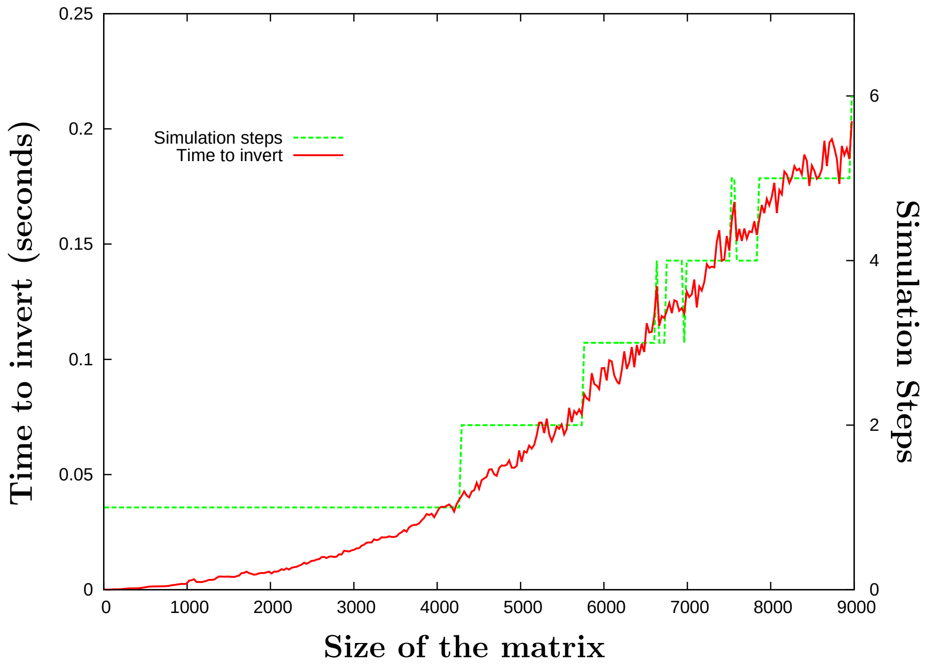


Figure 11: Time in (ms) to factorize the system depending on the dimension on the system. **Simulation steps** gives the number of simulation steps that are necessary to update the asynchronous preconditioner with a simulation running at 25 FPS.

We also evaluate the number of simulation steps necessary to update the preconditioner with a simulation runing asynchronously at 25 FPS (see fig. 11). Depending on the dimension of the matrix, the number of time steps necessary to update the factorization varies between 1 to 6. With our asynchronous version, this period can twice larger because a first period is necessary to compute the factorization, and then it will be used until the next update of the preconditioner. However, even for a $9,000 \times 9,000$ matrix, the maximum delay due to the update the preconditioner is less than 0.5 seconds.

7.2.3. Computation of the compliance

We now focus on the computation time (see fig. 12) for solving a Sparse Triangular System with multiple right hand side vectors. This operation is necessary to set the contact problem in equation (14) and (23).

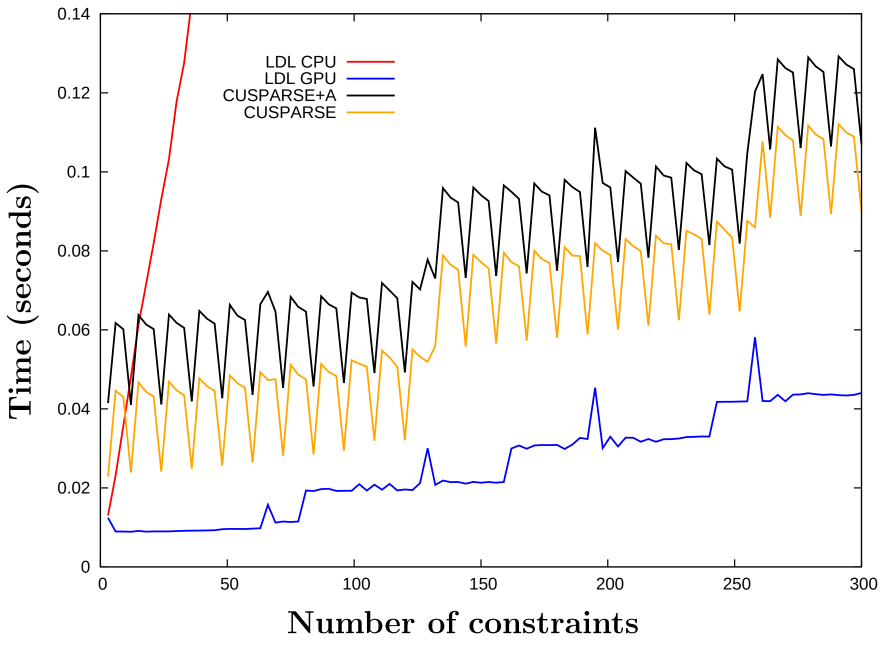


Figure 12: Computation time for solving a STS with multiple right-hand side vectors.

Solving the different STS on the GPU (**LDL GPU**) is much faster than solving sequentially each STS on the CPU (**LDL CPU**). Indeed, the computation time for the CPU version is linear according to the number of right-hand side vectors, whereas the GPU can process them in parallel. Therefore, for up to 78 constraints the computation time remains almost constant with our GPU implementation. Indeed, below this limit the GPU computing units are not fully utilized and the different STS are processed in parallel. Beyond this number, some GPU processors will compute several STS successively, and the computational time curve takes a staircase appearance. Nevertheless, the GPU processors are able to overlay waiting times, due to synchronizations and access in in memory, with computations for another STS. Thus, solving the system for 140 constraints is only 1.8 times slower than for 70 constraints.

We also compared our GPU implementation to the CUSPARSE library where an implementation of solving a STS with multiple right-hand side vector has recently been released in the procedure cusparseScsrsm_solve. This procedure is implemented in two steps: first an analysis of the sparsity of the matrix is process in order to determine the dependencies, then the analysis information is used to solve the STS (see Naumov (2011) for details). As the analysis is necessary only when the preconditioner is updated, we detailed separately the cost with (**CUSPARSE+A**) and without (**CUSPARSE**) the analysis. For 78 constraints our solution is 4.2 times faster when the analysis is not required (5.72 times with the analysis), and 2 times faster for 300 constraints (2.4 times with the analysis). Although the difference may be less for larger systems, we have optimal performance for the scenarios that are compatible with real-time constraint. Indeed, for 300 constraints the computation time to build $\mathbf{W}$ with our optimized approach is around 0.04 seconds which is a limit for interactive rates.

7.2.4. Topological Modifications

Finally, we evaluate the influence of the cut on the convergence of the standard **CG**, and using our method (**SMF**), on a simulation composed of a beam cut lengthwise and falling under gravity (see table 13). **Build** is the time to assemble $\mathbf{A}$: For the **CG**, the matrix is directly evaluated from the mesh structure at each iteration, whereas the preconditioner is updated on average every 4.30 simulation steps, each update requiring to fully assemble $\mathbf{A}$ to perform the factorization in equation (18). **Iterations** and **Solve** are respectively the number of iterations per time step, and the corresponding time to solve the system with a tolerance at 10^{-7}. The method significantly decreases the number of iterations, and provides an average speed-up of $2.6\times$ compared to the **CG**.

The main overhead of the **SMF** update is the computation $\mathbf{U}$ and $\mathbf{Q}$ (see fig. 14), but these operations are performed only when a new topological modification is detected (i.e. every 3.4 simulation steps on the beam example). The computation of $\mathbf{U}$ is the most expensive, and its GPU parallelization is the key point to enable real-time computation, whereas the

Nodes	Method	Affected Nodes	Build (ms)	Itera- tions	Solve (ms)	Total (ms)
1200	CG	NA	0.01	489.40	49.50	50.99
	SMF	46.39	7.30	6.59	10.41	20.25
1600	CG	NA	0.01	584	60.71	62.47
	SMF	52.50	9.00	7.26	15.70	27.57
2000	CG	NA	0.01	687.29	76.29	78.38
	SMF	54.75	10.24	7.43	21	34.40

Figure 13: Performances and convergence comparison.

inversion of $\mathbf{Q}$ is inexpensive for small perturbations but quickly becomes costly for large perturbations. In practice, the number of impacted nodes remains very small since the preconditioner is updated several times per second. Finally the application of the correction (i.e. applying $\mathbf{x}^c$ in equation (30) and $\mathbf{W}^c$ in (32)) is negligible since it represents less than 1% of the computation time of a time step.

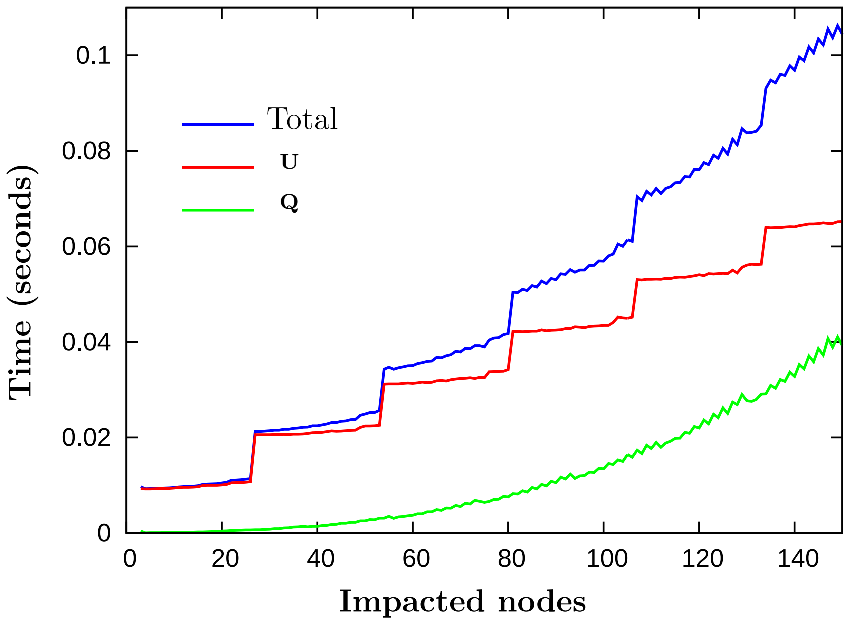


Figure 14: Overhead for the computation of the correction using the Sherman Morrison Formula.

8. Applications

We now demonstrate that our method is generic enough to address several kinds of simulation in a medical context. We use it to simulate a cataract surgery, an hepatectomy in laparoscopie, and a cerebral tumor removal. We show that our method can handle the requirements of such simulations in real time.

8.1. Application to cataract surgery

The cataract is an opacification of lens of the eye, which prevents the passage of light and results in partial or complete blindness. Millions of people are affected by this pathology, particularly in third world countries. A surgical treatment exists, which consists in extracting the diseased lens and replacing it with by an implant. The standard surgical procedure is known as *Phacoemulsification* Tsuneoka et al. (2002), where the lens is emulsified by an ultrasonic tool. However, Phacoemulsification requires advanced technology which is not available in many countries where the prevalence of cataract is highest, and hence many patients can simply not be treated.

Another surgical procedure known as *Manual Small Incision Cataract Surgery* (MSICS) Venkatesh et al. (2008) requires only basic technology and leads to quasi-identical results when performed by an experienced specialist. The cost of such surgical operations is significantly lower than phacoemulsification, but unfortunately only few surgeons in the world master this particular procedure. This technique requires a slightly larger incision (around 5 mm) so that the lens be extracted in a single piece. We believe that the requirements for such a simulator cannot be addressed by existing methods. Indeed, the eyeball as well as the lens must be simulated because they both undergo large deformations and high stresses. The heterogeneity of anatomical structures must also be taken into account since this heterogeneity has an impact on the success of the surgical operation. In particular, the nucleus of the lens is stiffer than its periphery which may require an adjustment of the size of the incision made to extract the lens.

We aim at simulating the extraction of the lens with the MSICS technique (see fig. 15). The incision and meshes of the organs has been generated offline using (CGAL) project. The lens is removed with the help of deformation of the eyeball, and friction with the instrument. The lens is modeled with 1113 nodes and 4862 tetrahedra, whereas the eye contains 1249 nodes and 3734 tetrahedra. The center of the lens is 5 times stiffer than the periphery,

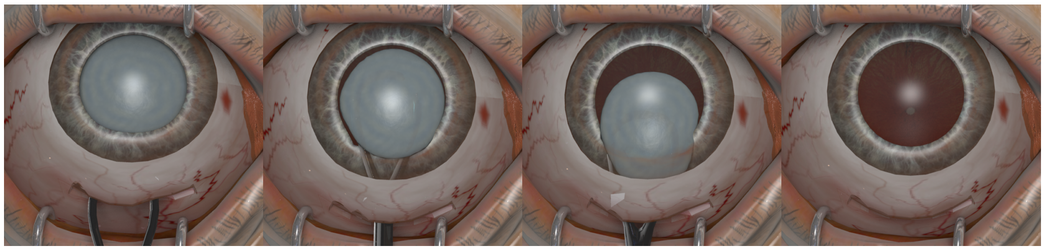


Figure 15: Simulation of the lens extraction with the MSICS technique.

and we used our asynchronous preconditioner to ensure the convergence of the CG. The preconditioned CG required an average of 11.6 iterations to converge to 10^{-5}, despite strong deformations and heterogeneity. Using our method, we managed to simulate this application in real-time, and the performance vary between 18 to 25 FPS. Within a single time step, the distribution of the computation time was: 40.56% for the free motion and 44.69% for the corrective motion.

8.2. Application to liver resection

In some cases, hepatectomy may be considered for the treatment of tumors localized inside the liver. Simulators of this procedure has already been developed in the past Bourquain et al. (2002); Lamadé et al. (2002). The originality of our approach is that our simulation is based on patient specific data. Indeed, the meshes of the organs are obtained from a semi-automatic segmentation of a CT (see Soler et al. (2001) for details). We simulate 5 deformable bodies in interactions (liver, stomach, colon, intestines and diaphragm). Each organ is composed of several hundred of nodes and thousand of elements with complex shapes composed of several thousand of triangles (see fig. 16). An important issue to produce this application concern the collision detection which is performed by the method introduced in Allard et al. (2010).

This application is simulated at a frequency of 25 FPS, including during cutting. The distribution of computing time in a time step is as follows: 27.77% for free movement, 11.01% for collision detection, 22.48% for the constraint motion. The preconditioned CG requires an average need of 5.82 iterations to converge. During cutting, the number of iterations was slightly higher but it quickly stabilizes when the cut stops. Finally, by taking into account the *mechanical coupling* between the contacts, we managed to pro-

	Nodes	Elements	Triangles
Liver	506	1607	3680
Stomach	306	756	594
Colon	347	819	702
Diaphragm	131	328	2162
Small Intestine	64	142	6192

Figure 16: dataset used for the simulation

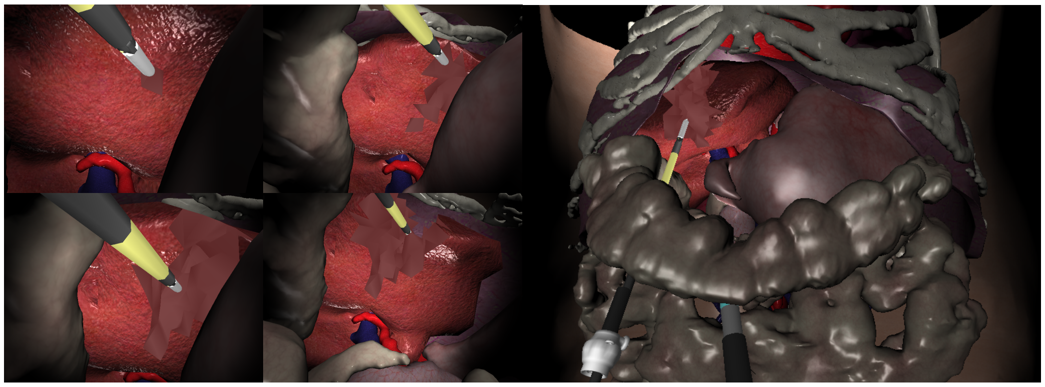


Figure 17: Simulation of an hepatectomy with haptic feedback.

duce a consistent haptic feedback. For instance, users can feel the stiffness of ribs behind the liver by applying contacts on the surface of the organ.

8.3. Application to brain tumor resection

Finally, we applied our method to the simulation of the resection of a brain tumour. The brain is modeled as a heterogeneous deformable body, composed of $1,734$ nodes and $7,680$ linear tetrahedral elements. The tumor is $20\times$ stiffer than the brain. During the simulation, the preconditioner is updated every 5.6 steps, and a new topological modification appears every 5.5 simulation steps, affecting 24 nodes. A total of 553 modifications are performed, and the method remains stable with an average of 5.70 iterations to solve the linear system. The collisions and self-collisions are correctly solved while processing the modifications, and cut parts can instantaneously be separated upon contact with the instrument. Finally, we achieve between 20 and 40 FPS and the method remains interactive.

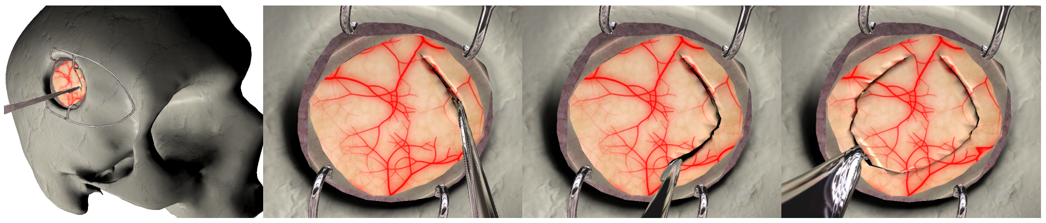


Figure 18: Real-time simulation of a brain tumor resection.

9. Conclusion

We presented a set of methods for the real-time simulation of deforming structures of complex shape relying on an implicit time integration method. The proposed paradigm relies on an asynchronous preconditioner that is updated at low frequency, and used to significantly reduce the number of iterations in the deformation solver, but also to set a contact problem which take into account the *mechanical coupling* between contacts. We also extended the approach to handle significant topological modifications at interactive rate. The method is particularly beneficial for heterogeneous structures, and enables to use large time steps to solve the contact problem, hence reducing significantly the computation time. We demonstrated the benefits of the proposed method through a simulation of a cataract surgery, but also a simulation of an hepatectomy based on patient specific data, and a simulation of a brain tumor removal involving topological modifications.

For future work, we plan to investigate algebraic model reduction techniques (using the proper orthogonal decomposition, or multi scale methods) to decrease the computational expense through pre computations, and to allow a finer description of organs. We also plan to investigate the use of enriched finite element methods in the real-time context to handle continus cut with the elements. Finally, we will also focus on the estimation of the spatial and temporal discretisation error and of the model error during our simulations. An analysis of the model error will require addressing more realistic models including hyper-elastic materials, where the mechanical matrices undergo 'faster' modifications than for the co-rotational case. We will also investigate the stability of the numerical schemes for nearly incompressible materials.

10. Acknowledgements

The authors would like to thank Mouhamadou Diallo and Igor Peterlik for their help and their assistance during the comparaison between SOFA and ABAQUS.

We also would like to thank the financial support of the European Research Council Starting Independent Research Grant (ERC Stg grant agreement No. 279578) entitled "Towards real time multiscale simulation of cutting in non-linear materials with applications to surgical simulation and computer guided surgery

References

Allard, J., Courtecuisse, H., Faure, F., 2011. Implicit fem solver on gpu for interactive deformation simulation. In: GPU Computing Gems Vol. 2. NVIDIA/Elsevier, to appear.

Allard, J., Faure, F., Courtecuisse, H., Falipou, F., Duriez, C., Kry, P. G., aug 2010. Volume contact constraints at arbitrary resolution. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010) 29 (3).

Anitescu, M., Potra, F., Stewart, D., 1999. Time-stepping for three-dimensional rigid body dynamics. Computer Methods in Applied Mechanics and Engineering, 183–197.

Baraff, D., Witkin, A., 1998. Large steps in cloth simulation. In: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. ACM, pp. 43–54.

Barbič, J., James, D. L., July 2005. Real-time subspace integration for st. venant-kirchhoff deformable models. ACM Trans. Graph. 24, 982–990.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., der Vorst, H. V., 1994. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM.

Berger-Vergiat, L., Waisman, H., Hiriyur, B., Tuminaro, R., Keyes, D., 2012. Inexact schwarz-algebraic multigrid preconditioners for crack problems modeled by extended finite element methods. International Journal for Numerical Methods in Engineering 90 (3), 311–328.

Bielser, D., Glardon, P., Teschner, M., Gross, M., 2003. A state machine for real-time cutting of tetrahedral meshes. In: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications. PG '03. IEEE Computer Society, Washington, DC, USA, pp. 377–.

Bolz, J., Farmer, I., Grinspun, E., Schröoder, P., 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Trans. Graph. 22 (3), 917–924.

Bordas, S., Duflot, M., 2007. Derivative recovery and a posteriori error estimate for extended finite elements. Computer Methods in Applied Mechanics and Engineering 196 (35), 3381–3399.

Bordas, S., Duflot, M., Le, P., 2008. A simple error estimator for extended finite elements. Communications in Numerical Methods in Engineering 24 (11), 961–971.

Bourquain, H., Schenk, A., Link, F., Preim, B., Prause, G., Peitgen, H., 2002. Hepavision2 - a software assistant for preoperative planning in living-related liver transplantation and oncologic liver surgery. Cancer Research, 1–6.

Braess, D., 1986. On the combination of the multigrid method and conjugate gradients. In: Multigrid methods II. Springer, pp. 52–64.

Brezzi, F., Hager, W. W., Raviart, P.-A., 1977. Error estimates for the finite element solution of variational inequalities. Numerische Mathematik 28 (4), 431–443.

Bro-Nielsen, M., Cotin, S., 1996. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. Comput. Graph. Forum 15 (3), 57–66.

Buatois, L., Caumon, G., Lévy, B., 2009. Concurrent number cruncher - a GPU implementation of a general sparse linear solver. Int. J. Parallel Emerg. Distrib. Syst. 24 (3), 205–223.

Comas, O., Taylor, Z., Allard, J., Ourselin, S., Cotin, S., Passenger, J., 2008. Efficient nonlinear FEM for soft tissue modelling and its GPU implementation within the open source framework SOFA. In: ISBMS. pp. 28–39.

Cotin, S., Delingette, H., Ayache, N., 1999. Real-time elastic deformations of soft tissues for surgery simulation. IEEE Transactions on Visualization and Computer Graphics 5 (1), 62–73.

Courtecuisse, H., Allard, J., jun 2009. Parallel dense gauss-seidel algorithm on many-core processors. In: High Performance Computation Conference (HPCC). IEEE CS Press.

Courtecuisse, H., Allard, J., Duriez, C., Cotin, S., nov 2010. Asynchronous preconditioners for efficient solving of non-linear deformations. In: Proceedings of Virtual Reality Interaction and Physical Simulation (VRIPHYS).

Courtecuisse, H., Jung, H., Allard, J., Duriez, C., Lee, D. Y., Cotin, S., 2011. Gpu-based real-time soft tissue deformation with cutting and haptic feedback. Progress in Biophysics and Molecular BiologySpecial Issue on Soft Tissue Modelling.

Davis, T. A., 2006. CSparse. Society for Industrial and Applied Mathematics, Philadephia, PA.

Dequidt, J., Duriez, C., Cotin, S., Kerrien, E., 2009. Towards interactive planning of coil embolization in brain aneurysms. In: Yang, G.-Z., Hawkes, D., Rueckert, D., Noble, A., Taylor, C. (Eds.), Medical Image Computing and Computer-Assisted Intervention, MICCAI 2009. Vol. 5761 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 377–385.

Dick, C., Georgii, J., Westermann, R., 2011a. A hexahedral multigrid approach for simulating cuts in deformable objects. Visualization and Computer Graphics, IEEE Transactions on 17 (11), 1663–1675.

Dick, C., Georgii, J., Westermann, R., 2011b. A real-time multigrid finite hexahedra method for elasticity simulation using cuda. Simulation Modelling Practice and Theory 19 (2), 801–816.

Dryja, M., Widlund, O. B., 1989. Towards a unified theory of domain decomposition algorithms for elliptic problems. New York University, Courant Institute of Mathematical Sciences, Division of Computer Science.

Duflot, M., Bordas, S., 2008. A posteriori error estimation for extended finite elements by an extended global recovery. International Journal for Numerical Methods in Engineering 76 (8), 1123–1138.

Duriez, C., Andriot, C., Kheddar, A., 2003. Interactive haptic for virtual prototyping of deformable objects: Snap-in tasks case. In: EUROHAPTICS. Citeseer.

Duriez, C., Andriot, C., Kheddar, A., 2004. Signorini's contact model for deformable objects in haptic simulations. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Duriez, C., Dubois, F., Kheddar, A., Andriot, C., 2006. Realistic haptic rendering of interacting deformable objects in virtual environments. IEEE Transactions on Visualization and Computer Graphics 12 (1), 36–47.

Felippa, C., Haugen, B., 2005. A unified formulation of small-strain corotational finite elements: I. theory. Computer Methods in Applied Mechanics and Engineering 194 (21-24), 2285 – 2335.

Felippa, C. A., 2000. A systematic approach to the element independent corotational dynamics of finite elements. Tech. Rep. CU-CAS-00-03, Center for Aerospace Structures.

Freund, L., 1970. Constitutive equations for elastic-plastic materials at finite strain. International Journal of Solids and Structures 6 (8), 1193–1209.

Ganovelli, F., O'Sullivan, C. A., 2001. Animating cuts with on-the-fly remeshing.

Gerstenberger, A., Tuminaro, R. S., 2012. An algebraic multigrid approach to solve xfem based fracture problems. Int. J. Numer. Meth. Eng.

Hauth, M., Etzmuß, O., Straßer, W., 2003. Analysis of numerical methods for the simulation of deformable models. The Visual Computer 19 (7-8), 581–600.

Hecht, F., Lee, Y. J., Shewchuk, J. R., O'Brien, J. F., Oct. 2012. Updated sparse cholesky factors for corotational elastodynamics. ACM Transactions on Graphics 31 (5), X:1–13, presented at SIGGRAPH 2012.

Heintz, P., Hansbo, P., 2006. Stabilized lagrange multiplier methods for bilateral elastic contact with friction. Computer methods in applied mechanics and engineering 195 (33), 4323–4333.

Hermann, E., Raffin, B., Faure, F., mar 2009. Interative physics simulation on multicore architectures. In: Proceedings of the 9th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'09).

Hiriyur, B., Tuminaro, R., Waisman, H., Boman, E., Keyes, D., 2012. A quasi-algebraic multigrid approach to fracture problems based on extended finite elements. SIAM Journal on Scientific Computing 34 (2), A603–A626.

Hughes, T., 2000. The finite element method: linear static and dynamic finite element analysis. Dover Publications.

James, D., Pai, D., 1999. Artdefo: Accurate real time deformable objects. In: 26th International Conference on Computer Graphics and Interactive Techniques. Proceedings of SIGGRAPH, ACM. pp. 65–72.

Jean, M., 1999. The non-smooth contact dynamics method. Computer Methods in Applied Mechanics and Engineering 177 (3-4), 235 – 257.

Joldes, G. R., Wittek, A., Miller, K., 2009. Suite of finite element algorithms for accurate computation of soft tissue deformation for surgical simulation. Medical Image Analysis 13 (6), 912 – 919, includes Special Section on Computational Biomechanics for Medicine.

Jourdan, F., Alart, P., Jean, M., 1998. A gauss-seidel like algorithm to solve frictional contact problems. Comp. Meth. in Appl. Mech. and Engin., 33–47.

Kaufman, D. M., Sueda, S., James, D. L., Pai, D. K., 2008. Staggered projections for frictional contact in multibody systems. ACM Transactions on Graphics 27 (5), 1–11.

Kerfriden, P., Gosselet, P., Adhikari, S., Bordas, S., 2011. Bridging proper orthogonal decomposition methods and augmented newton–krylov algorithms: an adaptive model order reduction for highly nonlinear mechanical problems. Computer Methods in Applied Mechanics and Engineering 200 (5), 850–866.

Kerfriden, P., Goury, O., Rabczuk, T., Bordas, S., 2012. A partitioned model order reduction approach to rationalise computational expenses in nonlinear fracture mechanics. Computer Methods in Applied Mechanics and Engineering.

Kühnapfel, U., Cakmak, H., Maaß, H., 2000. Endoscopic surgery training using virtual reality and deformable tissue simulation. Computers & Graphics 24 (5), 671–682.

Lamadé, W., Vetter, M., Hassenpflug, P., Thorn, M., Meinzer, H.-P., Herfarth, C., 2002. Navigation and image-guided hbp surgery: a review and preview. Journal of Hepato-Biliary-Pancreatic Surgery 9, 592–599, 10.1007/s005340200079.

Marchal, M., Allard, J., Duriez, C., Cotin, S., jul 2008. Towards a framework for assessing deformable models in medical simulation. In: Proceedings of ISBMS 2008. Springer, pp. 176–184.

Marchesseau, S., Heimann, T., Chatelin, S., Willinger, R., Delingette, H., sep 2010. Multiplicative jacobian energy decomposition method for fast porous visco-hyperelastic soft tissue model. MICCAI'10.

Martin, P., Ayuso, L., Torres, R., Gavilanes, A., 2012. Algorithmic strategies for optimizing the parallel reduction primitive in cuda. In: Smari, W. W., Zeljkovic, V. (Eds.), HPCS. IEEE, pp. 511–519.

Menk, A., Bordas, S., 2011a. Crack growth calculations in solder joints based on microstructural phenomena with x-fem. Computational Materials Science 50 (3), 1145–1156.

Menk, A., Bordas, S., 2011b. A robust preconditioning technique for the extended finite element method. International Journal for Numerical Methods in Engineering 85 (13), 1609–1632.

Miller, K., Joldes, G., Lance, D., Wittek, A., 2007. Total lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. Communications in numerical methods in engineering 23 (2), 121–134.

Molino, N., Bao, Z., Fedkiw, R., 2007. A virtual node algorithm for changing mesh topology during simulation. Proceeding of Eurographics, 73–80.

Mor, A. B., Kanade, T., 2000. Modifying soft tissue models: Progressive cutting with minimal new element creation. In: Proceedings of MICCAI 2000. pp. 598–607.

Naumov, M., 2011. Incomplete-lu and cholesky preconditioned iterative methods using cusparse and cublas.

Nealen, A., Mueller, M., Keiser, R., Boxerman, E., Carlson, M., 2006. Physically based deformable models in computer graphics. Comput. Graph. Forum 25.

Nesme, M., Marchal, M., Promayon, E., Chabanas, M., Payan, Y., Faure, F., 2005. Physically realistic interactive simulation for biological soft tissues. Recent Research Developments in Biomechanics 2, 1–22.

Nicolas, M., Dolbow, J., Belytschko, T., 1999. A finite element method for crack growth without remeshing. Int. J. Numer. Meth. Engng 46, 131–150.

Niroomandi, S., Alfaro, I., Cueto, E., Chinesta, F., 2008. Real-time deformable models of non-linear tissues by model reduction techniques. Computer Methods and Programs in Biomedicine 91 (3), 223–231.

Nitsche, J., 1971. Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind. In: Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg. Vol. 36. Springer, pp. 9–15.

NVIDIA Corporation, 2007a. CUBLAS library.

NVIDIA Corporation, 2007b. CUSPARSE library.

Otaduy, M. A., Tamstorf, R., Steinemann, D., Gross, M., 2009. Implicit contact handling for deformable objects. Computer Graphics Forum (Proceedings of Eurographics) 28 (2), 559–568.

Parker, E. G., O'Brien, J. F., aug 2009. Real-time deformaton and fracture in a game environment. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animaton. pp. 156–166.

Pauly, M., Pai, D. K., Guibas, L. J., 2004. Quasi-rigid objects in contact. In: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation. pp. 109–119.

Payan, Y., 2012. Soft tissue biomechanical modeling for computer assisted surgery 11.

Peterlik, I., Nouicer, M., Duriez, C., Cotin, S., Kheddar, A., Jul. 2011. Constraint-based haptic rendering of multirate compliant mechanisms. IEEE Trans. Haptics 4 (3), 175–187.

Redon, S., Kheddar, A., Coquillart, S., 2002. Gauss' least constraints principle and rigid body simulations. In: Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on. Vol. 1. IEEE, pp. 517–522.

Renard, Y., 2012. Generalized newton's methods for the approximation and resolution of frictional contact problems in elasticity.

Ródenas, J., González-Estrada, O., Tarancón, J., Fuenmayor, F., 2008. A recovery-type error estimator for the extended finite element method based on singular+ smooth stress field splitting. International Journal for Numerical Methods in Engineering 76 (4), 545–571.

Saad, Y., 1996. Iterative methods for sparse linear systems. Vol. 620. PWS publishing company Boston.

Saad, Y., 2003. Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Saupin, G., Duriez, C., Cotin, S., Grisoni, L., 2008. Efficient contact modeling using compliance warping. In: Computer Graphics International.

Schenk, O., Bollhöfer, M., Römer, R. A., 2008. On large scale diagonalization techniques for the Anderson model of localization. SIAM Review 50 (1), 91–112, sIGEST Paper.

Sifakis, E., Der, K. G., Fedkiw, R., 2007. Arbitrary cutting of deformable tetrahedralized objects. Proceeding of Eurographics, 73–80.

Soler, L., Delingette, H., Malandain, G., Montagnat, J., Ayache, N., Koehl, C., Dourthe, O., Malassagne, B., Smith, M., Mutter, D., Marescaux, J., 2001. Fully automatic anatomical, pathological, and functional segmentation from CT scans for hepatic surgery. Computer Aided Surgery 6 (3), 131–42.

Taylor, Z., Comas, O., Cheng, M., Passenger, J., Hawkes, D., Atkinson, D., Ourselin, S., Sep. 2008. Modelling anisotropic viscoelasticity for real-time soft tissue simulation. In: Proceedings of MICCAI 2008. pp. 703–710.

The CGAL Project, 2011. CGAL User and Reference Manual, 3.8 Edition. CGAL Editorial Board.

Toledo, S., Chen, D., Rotkin, V., feb 2003. Taucs: A library of sparse linear solvers version 2.2.

Tsuneoka, H., Shiba, T., Takahashi, Y., et al., 2002. Ultrasonic phacoemulsification using a 1.4 mm incision: clinical results. Journal of cataract and refractive surgery 28 (1), 81–86.

Venkatesh, R., Tan, C., Singh, G., Veena, K., Krishnan, K., Ravindran, R., 2008. Safety and efficacy of manual small incision cataract surgery for brunescent and black cataracts. Eye 23 (5), 1155–1157.

Wriggers, P., Panatiotopoulos, P., 1999. New developments in contact problems. No. 384. Springer.

Wriggers, P., Zavarise, G., 2008. A formulation for frictionless contact problems using a weak form introduced by nitsche. Computational Mechanics 41 (3), 407–420.

Zienkiewicz, O., Taylor, R., 1991. The Finite Element Method, 4th Edition. Vol. 1. McGraw-Hill.