Université de Strasbourg

UNIVERSITÉ DE STRASBOURG

msii ÉCOLE DOCTORALE

ÉCOLE DOCTORALE MSII 269

Mathématiques, Sciences de l'Information et de l'Ingénieur

Laboratoire des sciences de l'Ingénieur, de l'Informatique et de l'Imagerie

ICube UMR 7357

Thèse présentée par :

Ziqiu ZENG

soutenue le : **12 Septembre 2023**

pour obtenir le grade de : **Docteur de l'Université de Strasbourg**

Discipline/ Spécialité : **Informatique**

Towards real-time performance in large-scale physics-based simulations

THÈSE dirigée par :

Dr. Hadrien Courtecuisse	Chargé de Recherche CNRS, Strasbourg

MEMBRES DU JURY :

Rapporteurs :

Dr. Christian DURIEZ	Directeur de Recherche INRIA, Lille
Pr. Stéphane BORDAS	Professeur des Universités, Université du Luxembourg

Examinateurs :

Dr. Hyewon SEO	Directeur de Recherche CNRS, Strasbourg
Dr. Zhongkai ZHANG	Associate Professor CAIR-CAS, Hongkong

Université de Strasbourg

Ziqiu ZENG

iCUBE

Vers des performances en temps réel dans les simulations physiques à grande échelle

Résumé

Les simulations physiques ont suscité une attention considérable dans le domaine médical, en particulier dans le domaine des chirurgies virtuelles. Une préoccupation actuelle de la recherche dans ce domaine est centrée sur l'obtention de comportements physiques réalistes dans les simulations en temps réel d'objets déformables. Cependant, cela présente un défi car les simulations doivent concilier les exigences contradictoires de précision et de temps de calcul rapide simultanément. Bien que la finesse de la discrétisation soit préférée pour capturer des informations de forme détaillées, elle conduit souvent à des systèmes plus importants au prix d'une augmentation des coûts de calcul. L'objectif principal de ce manuscrit est d'améliorer les performances de calcul afin de permettre des simulations en temps réel à grande échelle. Pour atteindre cet objectif, notre travail englobe plusieurs méthodes visant à relever les défis de la résolution du système numérique dans différents domaines problématiques. Ces méthodes comprennent l'assemblage de matrices et les techniques de résolution parallèle pour simuler les déformations élastiques, l'assemblage de la matrice de conformité et les approches de résolution des contraintes pour simuler les contacts de friction, ainsi que l'optimisation de la qualité du maillage associée à un solveur efficace pour simuler les découpes. L'efficacité de ces avancées est évaluée dans diverses applications, permettant des simulations efficaces de la déformation, du contact et de la friction, ainsi que des coupes virtuelles. En conséquence, la réalisation de simulations physiques à grande échelle en temps réel devient réalisable.

Mots clés : Simulation physique, simulation en temps réel, méthode des éléments finis, simulation de contact, simulation de découpe, parallélisation sur le GPU.

Université de Strasbourg

Ziqiu ZENG

iCUBE

Towards real-time performance in large-scale physics-based simulations

Abstract

Physics-based simulations have garnered considerable attention in the medical field, particularly in the application of virtual surgeries. A current focus of research in this field is centered on achieving realistic physical behaviors in real-time simulations of deformable objects. However, this presents a challenge as simulations must fulfill the conflicting requirements of both accuracy and fast computation time simultaneously. While fine discretization is preferred for capturing detailed shape information, it often leads to larger systems with the cost of increased computational costs. The primary objective of this manuscript is to enhance computing performance to enable real-time simulations on a large scale. To address this objective, our work encompasses several methods aimed at overcoming challenges in numerical system resolution within various problem domains. These methods include matrix assembly and parallel solver techniques for simulating elastic deformations, compliance assembly and constraint resolution approaches for simulating frictional contacts, and mesh quality optimization combined with an efficient solver for simulating virtual cuts. The effectiveness of these advancements is evaluated across various applications, enabling efficient simulations of deformation, contact and friction, as well as virtual cutting. As a consequence, the attainment of large-scale physical simulations in real-time becomes feasible.

Key words: Physics-based simulation, real-time simulation, finite element method, contact simulation, cutting simulation, GPU-based parallelization.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to Dr. Hadrien Courtecuisse, my supervisor, for introducing me to the fascinating field of computer graphics. His unwavering support has been invaluable throughout the past four years of my research. I am truly thankful for the inspiration, discussions, and feedback he has provided, as his guidance has fueled my passion for research and significantly influenced the outcome of this thesis.

I would also like to extend my appreciation to Dr. Stéphane Cotin, Dr. Christian Duriez, and Dr. Florence Bertails-Descoubes for their valuable contributions. The insightful discussions and their vision have been instrumental in shaping my research work, and I am truly grateful for their invaluable feedback.

Furthermore, I want to acknowledge the valuable contributions of my colleagues in the MIMESIS/MLMS team, whose stimulating discussions have played a crucial role in my research. A special thanks goes to my friend Pedro Perrusi for his assistance during our preparation for the submission to *Eurographics*.

Lastly, I would like to express my sincere thanks to my parents in Hanchuan and my friends in Strasbourg for their unwavering support and understanding. Their encouragement has been crucial in helping me overcome challenges and maintain my motivation throughout the research process.

Thank you.

CONTENTS

CHAPTER 1

GENERAL INTRODUCTION

1.1 Physics-based simulations in medical applications

Surgical operations play a vital role in treating diverse medical conditions, from routine procedures to complex interventions. While they have significantly advanced medical practice, it's essential to note that the surgeries carry inherent risks and potential complications. To address these challenges, medical simulations have garnered significant attention due to their ability to provide a safe environment for learning and practicing complex surgical interventions. Using medical imaging techniques, it becomes possible to generate surgical models that accurately replicate a patient's anatomical structure in a digital format. Surgeons can utilize these virtual models to meticulously plan and rehearse complex procedures, enabling the identification of potential challenges and the optimization of surgical strategies before entering the operating room. Nowadays, virtual surgery involves a wide range of applications, including surgical planning, preoperative visualization, surgical training and education, as well as intraoperative guidance and assistance. These advancements have revolutionized the field of surgery by improving safety, efficiency, and overall patient care.

By employing principles from physics, mathematics, and computer science, medical simulations facilitate the simulation and visualization of biological systems, providing insights into intricate operations. Physics-based modeling forms the foundation of these simulations, involving the creation of virtual models that faithfully represent anatomical structures, biological systems, and their interactions. These interactions can vary across

applications, encompassing deformable solids (e.g., human tissues and organs), fluid dynamics (e.g., blood flow), and rigid objects (e.g., bones and surgical instruments). At the core of physics-based simulations lies the concept of using fundamental laws of physics to describe the behavior and interactions of biological systems. By integrating knowledge from multiple disciplines such as computer graphics and bio-mechanics, these simulations can accurately replicate the mechanical processes occurring within the human body.

The current focus in the field revolves around the potential to enhance medical simulations by integrating them more closely with the Operating Room (OR). This involves either using simulations for preoperative planning or incorporating them directly into the OR through visual assistance, registration, and augmented reality (AR). To achieve this, simulations must fulfill the conflicting requirements of both accuracy and fast computation time simultaneously. On one hand, the preference lies in employing complex models that can provide realistic predictions of the behavior of organs during surgery. On the other hand, real-time performance plays a crucial role in surgical simulations and training scenarios. These applications necessitate interactive and dynamic simulations capable of responding to user inputs and delivering immediate feedback. The primary challenges in real-time computing arise from various factors such as deformations, interactions, and topology modifications, among others.

Simulating the behavior of deformable solid is generally more challenging than that of rigid one due to the need to accurately represent elastic behavior with a higher dimension of degree of freedom (DOF). The most commonly employed method to replicate the behavior of deformable objects is the finite element method (FEM). This involves discretizing the object in the continuous space into finite elements, where a boundary value problem is formulated for each element. The connection structure between finite elements forms a mesh topology. By coupling all the local problems, the FEM produces a global system that can be solved using numerical methods. One of the challenges in computing is based upon the FE discretization. Typically, a finer discretization is preferred to accurately represent the shape and capture details. However, this leads to a larger number of DOFs and elements, resulting in expensive computational costs to solve the global system.

On the other hand, we need physical models for each finite element to correctly predict the deformations. In the fields of computer graphics and bio-mechanics, the physics-based modeling of deformable solids is typically derived from elastic models in continuum mechanics. Another challenge comes form the elastic model employed. Compared to straightforward linear elasticity models, hyperelastic models are preferred as they can accurately represent realistic behavior in response to large deformations. However, the

hyperelastic constitutive law leads to nonlinear formulations, requiring advanced numerical methods and extra costs to linearize the problem. Thus, finding the right balance between discretization, choice of elastic model, and real-time performance is an ongoing area of research and optimization.

Simulating interactions between objects presents another challenge, with frictional contact being a significant concern in the field of computer graphics. Several problems arise in this area: The first issue is how to model realistic frictional contacts. There are both simple and complex frictional models available, and the choice depends on specific application requirements. The second problem involves dealing with the discontinuities introduced by contact and friction. These non-smooth models actually transform the unconstrained problems in the deformation prediction into constrained ones, requiring complex solution methods to solve them. Furthermore, coupling different models within a multi-object system leads to the creation of an augmented system. In combination with nonlinear constraints, frictional contact problems typically result in large, nonlinear, and non-smooth problems, posing significant challenges for numerical resolution.

Our study also focus on simulating cutting operations, which encompass a diverse range of surgical interventions involving the precise removal or separation of tissues, structures, or organs for various purposes such as resection or dissection. The simulation of these cutting operations has significant interest, yet it also presents additional challenges. The act of cutting an object virtually introduces topological changes to the originally discretized mesh. Furthermore, the interaction between the surgical instrument and the object results in a cutting path that generally does not align smoothly with the existing discretization. As a result, additional modifications are required to align the cut surface with the cutting path. This can be achieved through various methods, such as subdividing the elements or snapping the closest existing surface. On the other hand, the topological changes caused by cutting have a direct impact on the numerical resolution of the system. Many of state-of-the-art solvers employ efficient solutions that rely on specific precomputations based on the topological structure. However, the cutting operation invalidates the precomputed data, posing significant challenges when utilizing these efficient solvers.

1.2 The open-source framework SOFA

Our research and development are based on the SOFA (Simulation Open Framework Architecture) Faure et al. (2012), which is an open-source framework targeted at real-time simulation, with an emphasis on medical simulations. Initially started in 2006 in Boston, the SOFA project was piloted by INRIA (Institut National de Recherche en Informatique

et en Automatique) engineers and researchers. Until 2023, SOFA has gathered about 15 years of research in physics simulation, involving diverse research topics such as soft robot control, endovascular simulations, needle insertion process (in minimally invasive operations), and virtual surgeries. Based on SOFA, many of works have been developed in different communities such as medical images, bio-mechanics, and computer graphics.

The fundamental basis of the SOFA framework is in C++. Compiling the C++ project generates various components that can be utilized in a scene graph, which is structured as a Direct Acyclic Graph (DAG). This scenario can be designed and organized flexibly using either an XML file or a Python file with the recently developed SofaPython plugin. Each component, also referred to as a node in the scene graph, serves a role in computational tasks such as algorithms, models, and tools. The interactions between the components are realized through specific data types and links defined within them.

The framework offers diverse types of physics-based simulations, including rigid solids, deformable solids using the FEM, and fluids using Smoothed Particle Hydrodynamics (SPH) and Eulerian formulation. SOFA's flexibility allows for a wide range of algorithms and models to be employed for deformable solids. SOFA enables using both the Euler explicit and the Euler implicit for time integration. The linearized system can be solved using either iterative solvers like Conjugate Gradient or direct solvers like Cholesky decomposition. Regarding contact problems, SOFA incorporates various discrete collision detection methods, such as proximity-based techniques and Layer Depth Image (LDI), to identify potential contacts. It provides the option to use penalty methods for fast contact responses and constraint-based techniques for accurate and stable solutions.

A notable feature of SOFA is its ability to represent multiple models for the same object. A general object can have different "states" for distinct purposes, such as mechanical displacements, collisions, and visualization. Importantly, each state can be defined flexibly to strike a balance between accuracy and real-time performance, based on specific requirements. For instance, a complex discretization can be applied to the collision model to achieve precise contact response, while a simpler mechanical model can be used to gain speedup from the computation of deformation behavior. Alternatively, this configuration can be inverted to prioritize accurate deformation prediction and provide a rough estimation of contact behavior. Finally, all the other models are connected to the mechanical model trough a mechanism called the "Mappings".

Another vital aspect of SOFA is its strong extensibility, which facilitates extensive research within the framework. This extensibility encompasses parallelization techniques on both CPU (using multithreading techniques) and GPU (via the CUDA API). Detailed explanations of these parallelization methods will be presented in the subsequent section.

1.3 Parallel programming

1.3.1 Multithreading

Multithreading is the capability of a program or operating system to execute multiple threads simultaneously within a single process. A thread represents a sequence of instructions that can be executed independently, enabling different parts of a program to be executed concurrently. This approach is widely used in simulation frameworks to enhance computational speed. Parallelization can be achieved at both the micro and macro levels, with a key requirement being the independence of parallel tasks:

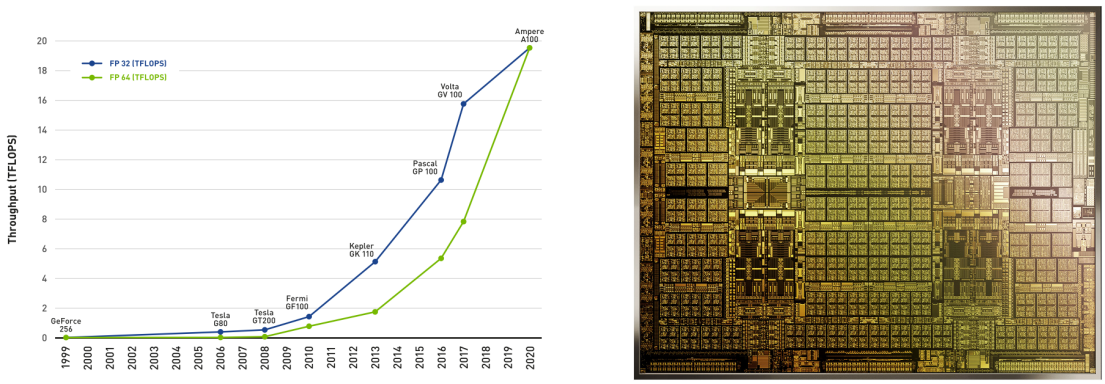
1. In the case of parallelization in massive independent unit tasks, tasks such as computing triangle normals or matrix-matrix multiplications can be parallelized using multithreading. This approach significantly reduces computation time compared to sequential execution using loops. Various C++ libraries, such as the Pardiso solver project for parallel Cholesky decomposition, provide support for multithreaded linear algebra operations.

2. Asynchronous programming also leverages multithreading between independent modules. For instance, solving the mechanical equilibrium without considering contact constraints and discrete collision detection at the beginning of a time step are independent tasks and could be executed simultaneously, since they only require mechanical states at the start of the time step. Recent research has explored advanced techniques such as asynchronous construction of preconditioners, allowing the main simulation loop to proceed without blocking. However, careful synchronization across sequential time steps is necessary for successful implementation of this technique.

To summarize, multithreading and asynchronous programming are valuable techniques that improve computational speed and performance by enabling concurrent execution of tasks.

1.3.2 GPU-based parallelization

Over the past two decades, the field of computer graphics has undergone a significant transformation, driven by the advancements in Graphics Processing Units (GPUs). Originally developed as specialized hardware for graphics rendering and display, GPUs have evolved into powerful parallel processing units that have revolutionized not only the gaming industry but also various scientific and engineering applications.



(a) **(b)**

Figure 1.1: Illustrations from the manuscript Dally et al. (2021): (a) The evolution of single GPU performance over the past 20 years. (b) The architecture of NVIDIA A100 GPU which was launched in November, 2020.

In particular, NVIDIA GPUs have gained prominence, with their CUDA (Compute Unified Device Architecture) cores playing a crucial role in parallel computing. The latest GPU architectures contains a remarkable number of CUDA cores, exceeding 10,000, enabling the concurrent processing of massively parallel tasks. This has led to the widespread use of General Purpose GPUs (GPGPUs) for accelerating computational workloads in fields such as Deep Learning (DL) that requires High-Performance Computing (HPC). Many methods in physics-based simulations also benefits from the boosted performance by GPU-based parallelization.

Similar to multithreading on CPUs, at the heart of GPU-based programming lies the concept of parallelism. However, implementing GPU-based methods presents challenges. GPUs excel in executing large-scale parallel computations but struggle with irregular or divergent control flow. Handling complex branching or conditional operations on GPUs can result in performance inefficiencies compared to CPUs. Additionally, GPUs have dedicated memory separate from the system's main memory, necessitating careful management of data transfer between CPU and GPU memory, which can introduce complexities and overhead. Debugging and profiling GPU code also pose difficulties, as GPUs execute parallel threads, requiring identification and resolution of issues like thread divergence, race conditions, or memory access conflicts. Debugging tools and profilers specific to GPU programming are available but often require a learning curve.

In our research, we utilize the CUDA Toolkit provided by NVIDIA, a robust development kit for implementing GPU-based methods. This toolkit includes a collection of CUDA-accelerated libraries optimized for GPU computing, such as cuBLAS and cuSPARSE for linear algebra tasks. Leveraging these libraries simplifies the development process by providing pre-optimized functions that enhance the performance of our applica-

tions.

1.4 Contributions

The primary aim of this manuscript is to enhance computing performance to enable real-time simulations on a large scale. We first study numerical solvers in real-time simulation frameworks. An efficient matrix-free iterative solver is available in SOFA, which is designed for linear elastic formulations and optimized for GPU-based implementation. However, implementing this method for generic hyperelastic materials in a uniform way proves to be challenging due to the divergence between different models. Conversely, the traditional solving strategy involves solving the system with the assembled matrix. Generic constitutive models can take advantage of GPU-based matrix operations, but the process of matrix assembly typically incurs an overhead cost that cannot be ignored. To address this issue, we introduce a fast matrix assembly method in our work Zeng and Courtecuisse (2023b), which is compatible with generic constitutive models. Our approach exploits the fact that system matrices are created in a deterministic way as long as the mesh topology remains constant. Using the sparsity pattern of the assembled system brings about significant optimizations on the assembly stage. As a result, developed techniques of GPU-based parallelization can be directly applied with the assembled system. Moreover, an asynchronous Cholesky precondition scheme is used to improve the convergence of the system solver. On this basis, a GPU-based Cholesky preconditioner is developed, significantly reducing the data transfer between the CPU/GPU during the solving stage. We evaluate the performance of our method with different mesh elements and hyperelastic models and compare it with typical approaches on the CPU and the GPU.

The second problem we address pertains to contact simulations, which involves solving a coupled multi-object system. Typically, this problem is transformed into constraint space by computing the Schur-complement of the system matrix. However, this process becomes extremely time-consuming when the system has large dimensions. In our work Zeng et al. (2022), we propose a fast method to handle large-scale FE simulations in the presence of contact and friction. Our approach employs a precondition-based contact resolution that performs Cholesky decomposition at a low frequency. By exploiting sparsity in assembled matrices, we propose a reduced and parallel computation scheme to address the expensive computation of the Schur-complement caused by detailed mesh and accurate contact response. An efficient GPU-based solver is developed to parallelize the computation, making it possible to provide real-time simulations in the presence of coupled constraints for contact and friction response. In addition, the preconditioner is

updated at low frequency, implying reuse of the factorized system. To benefit a further speedup, we propose a strategy to share the resolution information between consecutive time steps. We evaluate the performance of our method in various contact applications and compare it with typical approaches on both the CPU and GPU. The efficiency of the method has also been proven in our subsequent work on needle insertion simulations Martin et al. (2023).

Our subsequent work focuses on real-time virtual cutting. The mesh refinement methods, widely used in recent works, generate a large number of new DOFs, greatly increasing computational costs in the subsequent numerical solution. Instead, we employ a node-snapping strategy as the basis of our novel method to mitigate this issue. Using this strategy requires addressing several critical problems encountered in previous work, such as ill-shaped elements, topology changes, and energy preservation. With the solutions proposed, our cutting method involves using a vertices-snapping strategy to fit the boundary surface onto the cutting path while avoiding the generation of new elements: Using point cloud to model unscheduled cuts in real-time, our approach enables operating with users. We propose solving an elastic problem in order to minimize the volume change. To deal with cutting while deforming and to ensure energy conservation, the elastic problem is transferred into the reference state of the cutting object, and is solved in a second simulation. Efficient geometry operations are developed to handle the topological modifications during progressive cutting. Furthermore, we propose a modification to a fast matrix-free iterative solver on the GPU that can efficiently update pre-processed data used in GPU-based kernels, thus ensuring real-time performance in large-scale problems.

Our final study involves the potential works in contact and cutting problems. The first perspective is a a novel implicit scheme for resolving constraints in contact simulations Zeng and Courtecuisse (2023a). In our method, we propose a recursive corrective motion scheme where the constraint forces are corrected in Newton iterations. This scheme has the potential to handle the inconsistency of constraint directions at the beginning and the end of time steps. Furthermore, to keep the resolution process as efficient as possible, we propose a reformulation that provides efficient update of constraint matrices in the iterative scheme. Nevertheless, we have only evaluated the computing efficiency of the method, while the improvement of stability and behavior remains to be proven in the future work.

In the perspective of our cutting method, we propose enhancements to the fast assembly method Zeng and Courtecuisse (2023b) and the isolating mechanical DOFs method Zeng et al. (2022). By combining these solutions with the new virtual cutting approach, we overcome challenges related to the efficient update of pre-computed data in these methods, ensuring their efficiency even with topology changes.

In conclusion, our works contribute efficient solutions to simulating deformation, contact and friction, and virtual cutting. With the integration of GPU-based parallelization, our research enables real-time FE simulations for deformable solids, even with large-scale problems (up to 10,000 - 15,000 vertices), frictional contact, and cutting operations. This allows for interactive virtual surgeries with more detailed object shapes, providing operators with relevant information in real-time.

1.5 Overview

We present the outline of this manuscript: After reviewing the related works in Chapter 2, Chapter 3 presents the background for deformable solids simulations, followed by the fast matrix assembly strategy and the efficient GPU-based solution for the linear system in FE simulations. Chapter 4 and Chapter 5 are dedicated to the new proposed methods in contact problems and in virtual cutting, respectively. Finally, our works are concluded in Chapter 6 with the perspectives.

CHAPTER 2

STATE OF THE ART IN PHYSICS-BASED SIMULATIONS

2.1 Physics-based simulations of deformable solids

2.1.1 Time discretization

In the context of interactive simulations, an important choice is the time integration scheme. The explicit Euler methods have been widely used for medical simulations for soft materials Joldes et al. (2009). In this case, the solution only involves the (diagonal) mass matrix leading to very fast, simple to implement, and parallelizable solutions Comas et al. (2008). Unfortunately, user interactions may introduce sudden and stiff contacts at arbitrary location/frequency, which raises stability issues.

On the opposite, the implicit Euler methods are unconditionally stable, i.e., stable (but not necessarily accurate) for large time steps and arbitrary stiff materials Baraff (1996). Implicit schemes provide better control of the residual vector and hence that the external and internal forces are balanced at the end of the time steps. Although these advantages come at the cost of solving a set of linear equations at each time step, implicit integration schemes offer a reasonable trade-off between robustness, stability, convergence, and computation time, particularly when combined with a GPU implementation.

2.1.2 Deformation in solids

There is a considerable volume of works in the area of simulating deformable objects. Finite Element (FE) models using elastic/hyperelastic material provide a good understanding of the mechanisms involved in physiological or pathological cases, mainly because the soft-tissue behavior is directly explained through constitutive relations Courtecuisse et al. (2014). A common way to formulate the solid deformations is to search for the minimum conservative energy function $E(\mathbf{q})$ depending on the actual positions of vertices (*mechanical DOFs*) $\mathbf{q}$. However, dealing with such optimization problem in real-time is very challenging. This problem is critical when using a complex FE model with a fine mesh discretization and nonlinear mechanical laws because of the large dimension of DOFs and the integration of the nonlinear mechanics at each step.

A standard approach to minimize the conservative energy function in implicit scheme is the Newton-Raphson method (also known as Newton's method). Using the energy gradient ∇E (related to internal elastic forces) and the Hessian $\nabla^2 E$ (related to stiffness matrix), the Newton's method converges towards the minimized energy with a quadratic approximation in iterations. Consequently the Newton's method is very expensive in time consuming as it requires re-evaluating ∇E and $\nabla^2 E$ and resolve the linearized problem in each iteration. With the rapid growth of computational power and some simplifications such as setting with one Newton iteration, this approach, known as the classical force-based method, has become compatible with real-time and interactivity. First limited to linear elastic models Bro-Nielsen and Cotin (1996), it was later extended to large displacements with the co-rotational formulation Felippa (2000). Allard et al. (2012) proposes a parallel resolution on GPU for a co-rotational model without explicitly assembling the system matrix. In Kugelstadt et al. (2018) a novel approach is proposed for co-rotated FE simulations, splitting the deformation energy terms to a stretching part which can be solved efficiently by a pre-computed factorization and a volumetric part correction which is addressed approximately. This operator-splitting method shows considerably high performance. FE models are now used for the simulation of hyperelastic or viscoelastic materials in real-time Marchesseau et al. (2010). However, although some validations of the behavior against real objects have been conducted, these methods remain complex and expensive, and the simulation of realistic boundary conditions such as interactions between deformable objects is still an issue.

Departing from the classical force-based method, the geometrically motivated methods give an alternative solution for simple, robust and computationally efficient simulations. One of the successful works is the Position-Based Dynamics (PBD) Müller et al. (2007) that redresses the unconstrained deformation by iterative constraint solver, in or-

der maintain the undeformed shape. The main idea of PBD similar to those of the Shape Matching methods Müller et al. (2005); Rivers and James (2007). With a Gauss-Seidel type solver, the PBD keeps projecting the individual nonlinear constraints onto the global system iteratively. Consequently the material stiffness will depend on the iteration number and the fast method cannot correctly follow the elastic models in continuum mechanics.

Liu et al. (2013) uses Hookean springs to replace the inextensible springs in the PBD, and introduces a local-global alternating minimization solver that transform the nonlinear mechanics into coupling of a global linear system and localized nonlinear problems. By generalizing the mass-spring in the localized constraints to co-rotational models, the Projective Dynamics Bouaziz et al. (2014) successfully bridges the gap between the PBD and the continuum mechanics, providing a good trade-off between the performance of the PBD and the accuracy of continuum mechanics. This work is extended in Liu et al. (2017) for hyperelastic materials using a Quasi-Newton method to boost convergence. Recently, Trusty et al. (2022) models the elastic energy by mixed variational potential Reissner (1985) and uses the alternating solver to provide efficient and stable simulations for general elastic constitutive models.

Despite being computationally efficient and robust, the Projective Dynamics suffers from some drawbacks (e.g., the iteration number should be pre-adjusted to have correct behavior Ly et al. (2020)) and it dose not offer the same convention of parameters in continuum mechanics. Therefore the Projective Dynamics is not yet widely used in applied communities such as bio-mechanical simulations or soft robotics.

On the other hand, meshless methods and Neural Networks are other strategies to model soft tissues in real-time. A detailed review of this topic goes far beyond the scope of this article, but a survey can be found in Zhang et al. (2018). In this manuscript, our contributions focus on the improvement of real-time simulation methods in the community of SOFA Framework, which has an emphasis on bio-mechanical simulations and modeling of soft robotics. In these fields, the force-based FE methods still remain gold-standard approaches to simulate realistic behavior with real material parameters.

2.1.3 Linear problem solution

After an approximation of a first-order Taylor, which is equivalent to a single Newton iteration (see in Section 3.1.3), the nonlinear problem in the implicit scheme is linearized as a linear system $\mathbf{Ax} = \mathbf{b}$. The global system matrix $\mathbf{A}$ is a $n \times n$ symmetric and positive-definite matrix, where n is the number of mechanical DOFs. Generally, such a linear system can be solved either by a direct solver or an iterative solver. Direct solvers provide the exact solution by computing a factorization (for instance, the Cholesky factorization Barbič and

James (2005)) or a decomposition (QR decomposition), or eventually, the actual inverse of the system matrix Bro-Nielsen and Cotin (1996) (though not recommended for large matrices). Efficient libraries exist both on the CPU (Pardiso, MUMPS, Taucs) and GPU (cuSPARSE, MAGMA, AmgX). The direct solvers provides robust results but the computation cost trends to be expensive with increasing of n. For example, as one of the popular direct solvers, the Cholesky decomposition has a complexity of $\mathcal{O}(n^3)$, which becomes prohibitive for real-time performance in large-scale simulations. Therefore, the direct solvers are usually used in relatively small-scale problems.

On the other hand, iterative methods are usually preferred in FE simulations because they limit the number of iterations to compute an approximated solution and better control the time spent during the solving process. The most popular method is the Conjugate Gradient (CG) algorithm Saad (2003), because of the fast convergence and its simple implementation. Parallel implementations both on CPU Hermann et al. (2009); Parker and O'Brien (2009) and GPU Allard et al. (2011); Bolz et al. (2005); Buatois et al. (2009) were proposed. However, the convergence of iterative methods can be significantly impacted for ill-conditioned problems, i.e., when the ratio of the largest and smallest eigenvalues is large. The main issue to improve the CG is to gain speedup on sparse matrix-vector multiplication (*SpMV*) operations. As it is presented in Bell and Garland (2009), to accelerate the *SpMV* operations, many methods are explored to implement them on throughput-oriented processors such as GPU.

2.1.4 Matrix-free solvers and matrix assembly

Several methods rely on the fact that CG iterations can be performed without explicitly assembling the system matrix Martínez-Frutos et al. (2015); Müller et al. (2013). Matrix-free methods significantly reduce the memory bandwidth and are proven to be fast and stable. As an example, the method introduced in Allard et al. (2011) is designed for the co-rotational formulation and relies on specific cache optimization to compute rotation matrices directly on the GPU. However, the specific cache optimizations proposed for the rotation matrices do not extend to other types of material, such as hyperelastic laws.

Explicit assembly of global matrices is necessary for direct solvers to compute the factorization or decomposition of the system. The assembly step is usually less critical than the solving process itself, but it may become the bottleneck when combined with efficient solvers. There are several ways to construct sparse matrices; the most popular method is first to collect triplets (the row/column index and the value); then compress the triplets in a sparse format. A very efficient implementation is provided in the Eigen library. Recently Hiemstra et al. (2019) proposed a row by row assembling method for isogeometric linear

elasticity problems. To accelerate the assembling step and minimize memory transfers, several approaches proposed to assemble the matrix directly on the GPU Dziekonski et al. (2012); Fu et al. (2018); Zayer et al. (2017). However, specific GPU-based implementation of the assembling procedure is needed for each particular model.

2.1.5 Precondition techniques

Another intense area of research aims to improve the performance of the CG algorithm with the use of preconditioners to speed up its convergence. There are several typical preconditioners: diagonal matrix is simple to build but has limited effect Baraff and Witkin (1998); in contrast, precise ones such as incomplete Cholesky factorization are complex and costly to make but can significantly reduce the condition number Hauth et al. (2003).

For a typical synchronous preconditioner, the construction of the preconditioner has to be performed before the solving stage of each time integration, leading to additional computation costs. Some of the recent works aim to find a balance between the cost of applying the preconditioner and the effect of convergence boost, such as efficient preconditioners using the result of incomplete factorization Anzt et al. (2015) and inner Gauss-Seidel preconditioners Thomas et al. (2021).

On the other hand, the asynchronous preconditioners proposed in Courtecuisse et al. (2010a) exploit the continuity of the time line in physically-based simulations. Relying on the assumption that mechanical matrices undergo relatively small changes between consecutive time steps, the asynchronous preconditioning scheme processes the matrix factorization in a dedicated thread parallel to the main simulation loop and applies the factorization result as a preconditioner after a short delay. It enables access to a very efficient preconditioner with almost no overhead in the simulation loop. As a combination of a direct and iterative solver, the method requires explicitly assembling the matrix at a low frequency in the simulation loop to factorize the system in the dedicated thread. For both synchronous and asynchronous preconditioning schemes, applying the preconditioner requires processing the forward/backward substitution, leading to solving sparse triangular systems (STS). Parallelizing the solution of STS remains challenging in many applications. There are many works dedicated to improving the performance of STS solvers on the CPU Bradley (2016) and on the GPU Li and Zhang (2020); Picciau et al. (2017); Yamazaki et al. (2020). In Courtecuisse et al. (2014), a GPU-based asynchronous preconditioner was designed to solve the STS with multiple right-hand sides (RHS) in the contact problem. However, the method cannot efficiently exploit parallelization when dealing with a single RHS. Therefore, despite the asynchronous preconditioning scheme being introduced with a GPU-based CG implementation of the co-rotational model, ap-

plying the preconditioner was performed on the CPU, requiring data transfers between CPU/GPU for each iteration of the preconditioned CG.

2.2 Contact problem and coupled systems

2.2.1 Contact generation

In contact simulations, interactions in multi-objects systems usually cause discontinuity in the velocities. This can be handled either by the *event driven* scheme or by the *time stepping* scheme. The *event driven* scheme gives accurate results but is restricted with limited instantaneous contacts. In contrast, the *time stepping* scheme involves all contacts during a fixed time step Anitescu et al. (1999).

In the *time stepping* scheme, the interactions between objects are detected in each time step by a collision detection process that defines potential constraint pairs for discretized systems. The detection methods are generally classified as two types: the Discrete Collision Detection (DCD) and the Continuous Collision Detection (CCD). The DCD Macklin et al. (2019) searches for potential contact at fixed intervals, providing efficiency in computing performance. In contrast, the CCD Li et al. (2020) checks for collisions along the trajectory of potential contact objects, being necessary for high-speed objects such as bullets. Both of them could be either processed once at the time step beginning for performance, or be integrated into the solving iterations for accuracy and stability.

While applied in bio-mechanical simulations, the DCD methods are usually preferred since the fast-moving object is rare in application scenarios. As a straightforward method, the proximity-based detection Martin et al. (2023) searches for the closest distances between elements on the mesh boundary. The Bounding Volume Hierarchy (BVH) is a technique that is usually used to efficiently accelerating the detection process, where the BV types could be various: such as sphere Hubbard (1996), oriented bouning box (OBB) Zachmann (2002), and axis aligned bounding box (AABB) Larsson and Akenine-Möller (2001). The volume-based method computes the intersection between the bodies and defines the contacts based on the intersected volume. This method could be combined with the Layered Depth Images (LDI) technique, providing fast collision detection for complex meshes while using a GPU-based implementation Allard et al. (2010). The Signed Distance Fields (SDF) Xu and Barbič (2014) give an implicit function for the mesh shape, resulting in negative values if inside the object and positive values if outside (conventionally). However, since the function is static and usually pre-computed in initialization, using the SDF requires that at least one of the colliding object is rigid. There are also other recent methods on the collision detection. For example, Masterjohn et al. (2022) pro-

poses a new contact model that evaluates the contact with the surface of equal pressure between two intersected pressure fields. Using different collision detection method will not only impact the performance, but also the contact model and collision definition that will be discussed in the next section.

2.2.2 Contact and friction models

Once the contacts are defined, one can solve the dynamic system with different contact models that dominate the motion of interacting objects. As the earliest method, the penalty methods apply approximate forces on contact points depending on the interpenetration and a problem-dependent parameter named penalty value. A larger penalty value enforces a more limited interpenetration but also leads to a worse conditioned problem. Consequently, the methods only give an approximate solution and can hardly handle stable contacts. Being very simple to be implemented and fast, the penalty method is still very popular in many applications of fast simulations Kugelstadt et al. (2018).

On the other hand, the constraint-based methods using Lagrange multipliers (Jean (1999), Renard (2013)) solve the contact problem in a coupled way, addressing the limitation in the penalty methods. Such methods provide accurate and robust solutions in contact mechanics for large time steps, where interpenetration is entirely eliminated at the end of time steps. To address the problem, the methods are commonly formulated as a complementarity problem, which is based on the Signorini's law for non-penetration contact and the Coulomb's law for the friction. The Lagrange multipliers methods can be naturally used to impose displacement (bilateral constraints Galoppo et al. (2006)), but interactions between multiple objects usually lead to define unilateral constraints.

The PBD and the Projective Dynamics come with a straightforward way to handle the collision by simply adding penalty-based contact energy into the global system. Ly et al. (2020) introduces the Signorini-Coulomb law into the Projective Dynamics to satisfy the non-penetration and Couloumb constraints in the local solver. Lan et al. (2020) proposes a new method that reduces the detailed mesh as a skeleton-like model via the Medial Axis Transform (MAT) Li et al. (2016), and uses the Projective Dynamics to solve the reduced model. Using the MAT, this work implements efficient transform between the original shape and the reduced model, and an efficient collision detection between medial cones.

2.2.3 Numerical methods in complementarity problems

The force-based methods for the elastic deformation usually fit well with the constraint based techniques for stability and accuracy. However, using Signorini-Coulomb law will

introduce discontinuities into the dynamic equations, leading to a non-smooth formulation. In the constraint resolution, different numerical methods can be used to address the linear complementarity problem (LCP) in physics-based animations Erleben (2013). Direct methods such as pivoting methods give exact resolution, but they are not computationally efficient. In contrast, iterative methods have been more widely applied in large-scale simulations, especially for those who require to perform real-time computations. Being simple to implement, projected Gauss-Seidel (PGS) (Duriez et al. (2006), Courtecuisse and Allard (2009), Macklin et al. (2016)) is able to handle the friction response with the Coulomb's friction cone combined in the LCP formulation. However, the algorithm is not efficient for ill-conditioned problems due to the slow convergence. The non-smooth Newton methods reformulate the complementarity problem into a root search problem using nonlinear complementarity problem (NCP) functions. However, the NCP functions contains discontinuities in their derivatives, making it difficult to be integrated into the Newton's method. Macklin et al. (2019) proposes removing the discontinuities with a complementarity preconditioner for the NCP function. Moreover, in order to avoid recomputing and inverting the Hessian iteratively, the method approximates the system matrix by a diagonal matrix, with an evalutation of the momentum balance gradient. The Increment Potential Contact (IPC) Li et al. (2020) formulates the contact model as an unsigned distance function. Using designed barrier functions, the method transforms the constrained problem as an unconstrained optimization which could be solved by a Newton-like barrier solver.

2.2.4 Schur-complement in constraint-based methods

In physics-based animations with constrained dynamics, one efficient solution is to formulate the Schur-complement of the augmented system, allowing this way to solve the problem in the *constraint space* that usually has a much smaller size than the augmented system. The Schur-complement results in a compliance matrix (also called *Delasus operator*) in the *constraint space*. As discussed in Andrews and Erleben (2021), the computation of Schur-complement tends to be costly when dealing with soft-body since it implies to solve a linear system with multiple right-hand sides. In such system, the discretization of FE models determines the problem size (*mechanical DOFs*) and the constraints determines the right-hand sides. We would like to note here that although the geometrically motivated methods (PBD and projective dynamics) also process constraint resolution, they meet different challenges from the FE methods in the system resolution. Building the *Delasus operator* is not necessary for projective dynamics as the internal and external constraints are solved locally in each element then coupled in a global solver. But in FE

simulations, the *Delasus operator* is important to couple contact forces in the resolution and enables stable simulations with complex interactions.

Many methods have been proposed to efficiently process the Schur-complement in interactive FE simulations. Saupin et al. (2008b) presents a compliance warping to precompute a factorized system in initialization and to apply correction in online simulation according to the co-rotational formulation. Unfortunately, the method is restricted with small deformation. Schenk and Gärtner (2006) and Petra et al. (2014) present respectively a fast factorization method and a fast Schur-complement computation using an augmented factorization method. Both the two methods are parallelized in CPU threads and integrated into *Pardiso solver project*, giving a fast CPU-based resolution for linear systems and Schur-complement. Updating Cholesky factor is suitable for FE simulations since the deformations are usually limited in consecutive time steps. This technique becomes very efficient by reusing factorization on sub-meshes Herholz and Alexa (2018) and is extended to dimension addition cases (e.g., mesh cutting) in Herholz and Sorkine-Hornung (2020).

However, even with highly optimized methods, factorizing large-scale systems remains highly expensive and makes it hard to be applied in real-time simulations. These CPU-based methods suffer from a very costly factorization for detailed meshes, making it hard to perform real-time simulations. Since factorization remains a critical obstacle in real-time applications, Courtecuisse et al. (2010a) proposed an asynchronous method to compute a preconditioner for iterative method, moving the factorization of the system into a parallel thread. This work is extended to the contact simulation in Courtecuisse et al. (2014) to compute an approximate solution of Schur-complement, using a highly parallelized solver on GPU. The method enables real-time simulations with up to 2000 nodes with 300 constraints but the computation of the Schur-complement is dominant in the time integration.

2.3 Cutting simulations and topology change

A significant amount of research has been conducted in the scientific literature on the topic of virtual cutting. Virtual cutting methods can be broadly classified as either mesh-based or meshless methods, depending on the approach used to simulate deformations. Mesh-based methods, which utilize FE discretization, involve splitting the mesh and modifying its topology to accurately represent the shape of the object, consistent with the methods discussed in previous sections on deformation and contact. However, these methods face challenges in dealing with changes in topology and usually result in poorly

shaped elements after cutting. Additionally, the necessary operations involved in modifying the mesh incur extra computational costs, making it difficult to achieve real-time performance. The topology changes often lead to geometrical discontinuities such as cracks after virtual cuts. While the traditional FEM struggles to accurately represent such problems, the eXtended FEM (XFEM) Bansal et al. (2019); Peng et al. (2014) overcomes the limitations by enriching the standard finite element approximation with additional functions that can capture the singularities or discontinuities in the problem. However, using XFEM can be computationally expensive as the enrichment functions introduce additional degrees of freedom, increasing the complexity of the problem.

Meshless methods, on the other hand, have emerged as an alternative to mesh-based approaches and have been rapidly developing in recent years. These methods, such as those described in Jin et al. (2012); Nguyen et al. (2008); Zhang et al. (2014), employ a set of points to represent deformable objects without the need for mesh generation and refinement. Consequently, meshless methods overcome the difficulties associated with modifying mesh topology during cutting and efficiently model discontinuities by adapting the distribution of nodes and basis functions to the problem's characteristics. However, meshless methods also face challenges such as the lack of convergence and stability guarantees, difficulty in enforcing boundary conditions, and high computational costs when dealing with a large number of nodes.

In this manuscript, our focus is on mesh-based cutting in order to maintain consistency with the methods used in FE simulations. A comprehensive review of meshless methods in the context of virtual cutting is beyond the scope of this manuscript, but interested readers can refer to surveys conducted by Wang and Ma (2018); Wu et al. (2015) for more information.

2.3.1 Mesh-based cutting methods

The meth-based cutting presents persistent challenges stemming from a range of factors, encompassing:

1. **Dealing with topology change** The principal hurdle in virtual cutting involves effectively managing alterations in the mesh topology required to divide the object into multiple parts. This necessitates techniques capable of accurately handling such changes.

2. **Ensuring cutting surface quality** It is crucial to maintain a high standard of quality for the cutting surface. Attaining a smooth and precise surface following the cut is imperative for accurate virtual simulations and visualizations. However, the inter-

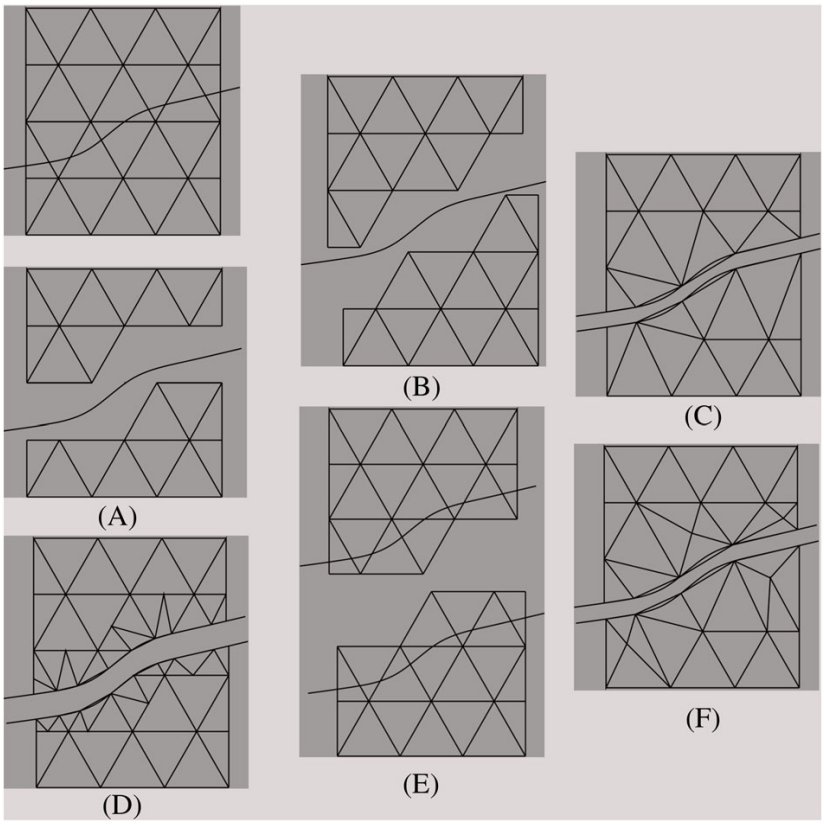


Figure 2.1: Illustrations for mesh-based methods from the survey Wang and Ma (2018) : (A) mesh deletion; (B) splitting along the existing surface; (C) node snapping; (D) mesh refinement; (E) virtual node algorithm (mesh duplication); (F) mesh refinement with node snapping

action between the surgical instrument and the object typically results in a cutting path that lacks smooth alignment with the existing discretization. Consequently, supplementary modifications are necessary to align the cut surface with the cutting path.

3. **Preserving mesh quality after cutting** Current supplementary techniques, such as subdivision or snapping strategies, are often employed to optimize the cut surface. However, these methods inevitably generate ill-shaped elements, leading to a degradation in the quality of the mesh. Preserving the overall quality of the mesh becomes a critical challenge, as degraded mesh quality can have adverse effects on subsequent numerical solutions, introducing ill-conditioned problems.

4. **Efficient numerical solution with topology change** Topology changes invalidate precomputed data in many state-of-the-art solvers, where efficiency heavily relies on these precomputations assuming an unchanged topology structure. As a result, achieving an efficient numerical solution in the presence of topology changes be-

comes an additional challenge that must be addressed.

In the context of real-time finite-element simulations of deformable solids, several methods can be used to address the topological process of tetrahedral meshes: Although the simple and efficient methods (element deletion Cotin et al. (2000); Forest et al. (2005) and splitting along existing faces Nienhuys and van der Stappen (2000)) ensure both the good quality of the mesh and real-time performance of the simulation, they cannot provide a cutting surface that fits onto the cutting path. Moreover, deleting elements will lead to the destruction of volume conservation in the physical system. Being popular in recent works of virtual cutting, the method of mesh subdivision Bielser et al. (1999); Burkhart et al. (2010); Li et al. (2021); Paulus et al. (2015a) refines the local elements on the cutting path, such that the cutting surface follows the path. However, the refined elements risk being ill-shaped, and the cutting process generates new elements continuously, increasing the computation cost for the numerical solver. On the other hand, the node snapping method Nienhuys and van der Stappe (2001); Serby et al. (2001) adjusts the cutting surface by moving the local vertices onto the cutting path. Similar to the refinement method, the snapping approach provides good quality of cutting surface and risks generating ill-shaped meshes. Furthermore, unlike the refinement method, it will not generate new elements, limiting the increase in computation cost. Nevertheless, few of recent works has developed the original node snapping method due to the difficulty of addressing the ill-shaped mesh and progressive simulation. A hybrid approach Steinemann et al. (2006) combines the methods of node snapping and element refinement. The virtual node algorithm Molino et al. (2004) duplicates the elements to cut and embeds the material surface into the duplicated element, allowing handling of the cutting surface while avoiding ill-shaped meshes. Sifakis et al. (2007) and Wang et al. (2014) extend the work to high-resolution material surface embedding and provide the possibility of arbitrarily cutting a tetrahedron.

2.3.2 Numerical solvers with topology change

Many state-of-the-art solvers that aim for real-time performance heavily depend on precomputed data that is determined by the original mesh's topological structure. Modifying the topology during virtual cutting necessitates efficient updates to the precomputed data, thereby posing a new challenge. In Section 2.1.3, we discussed both direct solvers and iterative solvers as viable options for solving the linearized problem in deformation simulations.

In the case of direct solvers used in static finite element simulations with a linear elasticity model, the inverse of the system matrix can be precomputed and stored. When

the cutting process alters the mesh topology, the Sherman-Morrison-Woodbury formula Hager (1989) can be employed to update the precomputed data. Authors such as Courtecuisse et al. (2014, 2010b) utilized this formula in an asynchronous preconditioner to update the compliance matrix after cutting. Another approach proposed by Yeung et al. (2016, 2020) involves quickly updating the system solution by computing the Schur complement of the stiffness matrix in an augmented system that accounts for changed or added DOFs. In Herholz and Alexa (2018), an efficient method was introduced to update the Cholesky factor by reusing factorization on sub-meshes, and this work was extended in Herholz and Sorkine-Hornung (2020) to handle changes in dimension and topology. In Projective Dynamics, a precomputed Cholesky factorization enables efficient system resolution. However, similar to the system matrix in typical force-based methods, topological changes during the cutting process necessitate refactorization, compromising real-time performance. To tackle this problem, Li et al. (2021) presents a novel algorithm to update the Cholesky factor in the cutting simulation using the Projective Dynamics and the element refinement technique. This method allows for efficient addition of arbitrary new DOFs in the Cholesky factor update, whereas Herholz and Sorkine-Hornung (2020) only permits copies of existing DOFs.

Simulating cuts with an iterative solver involves modifying the matrix pattern of the global system matrix. Efficient matrix assembly strategies, such as the fast matrix assembly method (one of our contributions in this manuscript), exploit the unchanged topology structure and provide efficient assembly when the topology remains unaltered. Conversely, matrix-free methods Allard et al. (2011) encounter challenges when dealing with topological modifications since their computational efficiency relies on extensive precomputation that is dependent on the mesh topology. In both methods, the cutting process invalidates the precomputed data, necessitating costly recomputation. Thus, when modifying the mesh topology, additional operations are required to effectively handle the changes in the precomputed data instead of recomputing it.

CHAPTER 3

PHYSICS-BASED SIMULATIONS OF DEFORMABLE SOLIDS

In this chapter we will discuss the background for deformable simulations using the implicit Euler integration in Section 3.1. This involves the typical formulations for elastic deformation derived from continuum mechanics, the dynamic equations in implicit scheme, and the matrix-free iterative solver. Then in the following sections we will clarify the challenges of parallelizing the numerical resolution and introduce our new proposed methods, including a fast assembly strategy and an efficient numerical solver in FE simulations.

3.1 From continuum mechanics to numerical simulations

3.1.1 Elastic deformation and constitutive laws

In computer graphics, the conventional definition for a deformation gradient $\mathbf{F}$ is described as a tensor that transforms the undeformed shape to the deformed shape Bargteil et al. (2020); Sifakis and Barbic (2012):

$$\mathbf{q} = \mathbf{F}\mathbf{q}^{\text{ref}} + \vec{T} \tag{3.1}$$

where we consider the deformed positions $\mathbf{q}$ and undeformed positions $\mathbf{q}^{\text{ref}}$ in a general finite element, and $\vec{T}$ refers to a rigid translation from the initial position.

Following Sifakis and Barbic (2012), we use the classical definition for hyperelastic materials in computer graphics (different from the definition in material mechanics): The material whose strain energy is independent from the prior deformation history. This property exposes a fact that the internal elastic forces $\mathscr{F}$ of hyperelastic materials are conservative, allowing for a formulation of the total strain energy $E(\mathbf{q})$ that is only determined by the final deformed shape.

The energy density $\Psi(\mathbf{q})$ measures strain energy per unit undeformed volume on an infinitesimal space Ω around the reference position $\mathbf{q}^{\text{ref}}$. Generally we expect that the energy density is independent from the rigid translation in Equation (3.1), implying that Ψ is expected to be a function of the deformation gradient, thereby $\Psi(\mathbf{F})$. This formula reveals the relation between $\mathbf{F}$ and Ψ , known as deformation-stress relation in constitutive models. There are several types of successful constitutive models that are designed for different behaviors and different applications. These models differ in the constitutive law and computing complexity, but share the same idea to define specific intermediate quantities derived form $\mathbf{F}$. In the following context we will discuss the definitions of deformation-stress relation in several typical models, varying from linear elasticity to high-order laws.

Linear elastic model The linear elasticity model, which serves as a fundamental constitutive model, employs the small strain tensor ξ as an intermediate variable. The small strain tensor is defined as follows:

$$\xi = \frac{1}{2}(\mathbf{F} + \mathbf{F}^{\mathrm{T}}) - \mathbf{I} \tag{3.2}$$

while the strain energy density is defined as:

$$\Psi(\mathbf{F}) = \mu||\xi||_{\mathrm{F}}^2 + \frac{\lambda}{2}\mathrm{tr}^2(\xi) = \mu||\frac{1}{2}(\mathbf{F} + \mathbf{F}^{\mathrm{T}}) - \mathbf{I}||_{\mathrm{F}}^2 + \frac{\lambda}{2}\mathrm{tr}^2(\frac{1}{2}(\mathbf{F} + \mathbf{F}^{\mathrm{T}}) - \mathbf{I}) \tag{3.3}$$

where μ and λ represent the Lamé coefficients, which can be derived from the Young's modulus E (indicating resistance to stretching) and the Poisson's ratio ν (indicating incompressibility).

The linear elasticity model exhibits several important characteristics. It is relatively straightforward to implement and computationally efficient since the nodal elastic forces (represented by $\nabla\Psi$) have a linear relationship with nodal positions. However, it is important to note that the small strain tensor approximation is only valid for small deformations. Furthermore, the model lacks rotational invariance, as a rigid rotation results in a nonzero strain tensor ξ.

Corotational model The corotated linear elasticity, also referred to as the corotational model, is introduced as a solution to the problem of rotational invariance in the linear model. By utilizing the Singular Value Decomposition (SVD), this model eliminates rotations from the deformation gradient. The SVD representation is given as:

$$\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}} \tag{3.4}$$

Here, $\mathbf{U}$ and $\mathbf{V}$ are rotation matrices, while Σ is a diagonal matrix containing the singular values of $\mathbf{F}$, representing the deformation gradient with rigid rotations removed.

Replacing the small strain tensor with the modified tensor measure $\Sigma - \mathbf{I}$, the strain energy density for corotated elasticity is defined as follows:

$$\Psi(\mathbf{F}) = \mu||\Sigma - \mathbf{I}||_{\mathrm{F}}^2 + \frac{\lambda}{2}\mathrm{tr}^2(\Sigma - \mathbf{I}) \tag{3.5}$$

The corotational model ensures rotational invariance in linear elasticity, albeit at the additional cost of performing the SVD. However, it still encounters the limitation that the reliable measure is only applicable to small-scale deformations.

St.Venant-Kirchhoff model The St.Venant-Kirchhoff model uses the *Green strain tensor*, defined as:

$$\mathbf{E} = \frac{1}{2}(\mathbf{F}\mathbf{F}^{\mathrm{T}} - \mathbf{I}) \tag{3.6}$$

It is important to note that the small strain tensor, denoted as ξ, employed in linear elasticity, is an approximation of the Green strain tensor for small deformations. In a similar fashion, the strain energy density for St.Venant-Kirchhoff elasticity can be derived by substituting ξ from Equation 3.3 with $\mathbf{E}$, yielding:

$$\Psi(\mathbf{F}) = \mu||\mathbf{E}||_{\mathrm{F}}^2 + \frac{\lambda}{2}\mathrm{tr}^2(\mathbf{E}) = \mu||\frac{1}{2}(\mathbf{F}\mathbf{F}^{\mathrm{T}} - \mathbf{I})||_{\mathrm{F}}^2 + \frac{\lambda}{2}\mathrm{tr}^2(\frac{1}{2}(\mathbf{F}\mathbf{F}^{\mathrm{T}} - \mathbf{I})) \tag{3.7}$$

The St.Venant-Kirchhoff model exhibits rotational invariance and demonstrates plausible material response in various scenarios involving large deformations, where linear elasticity would not be suitable. However, there are trade-offs associated with this model. The relationship between nodal elastic forces and nodal positions is no longer linear, necessitating advanced solvers for numerical resolution. Additionally, the model has limitations in terms of its resistance to extreme compression.

Neohookean model Instead of utilizing the strain tensor, the Neohookean elasticity model employs *isotropic invariants* as its intermediate variables. These invariants, denoted as I_1,

I_2, and I_3, are defined as follows:

$$I_1 = \sum_i \sigma_i^2 \qquad I_2 = \sum_i \sigma_i^4 \qquad I_3 = \prod_i \sigma_i^2 \tag{3.8}$$

where σ_i are singular values extracted from the SVD of $\mathbf{F}$ (as discussed in the corotated model).

The Neohookean elasticity model defines its strain energy density as follows:

$$\Psi(I_1, I_3) = \frac{\mu}{2}(I_1 - \log(I_3) - 3) + \frac{\lambda}{8}\log^2(I_3) \tag{3.9}$$

The Neohookean model has many notable properties on the behavior, including rotational invariance, a strong response to extreme compression, and the ability to prevent inverted configurations. However, this model involves a high-order nonlinear formulation, which requires significant computational resources for stress computations, making real-time computations challenging. Other nonlinear models, such as Mooney-Rivlin elasticity, also utilize isotropic invariants to formulate the energy density.

The divergence of formulations across different models poses a challenge when attempting to assembly the elastic tensors (elastic force ∇E and stiffness matrix $\nabla^2 E$) in a uniform way. An efficient method to assemble the stiffness matrix for the high-order models that utilize *isotropic invariants* in their formulations, can be observed in the implementation within SOFA. This method involves the utilization of the Multiplicative Jacobian Energy Decomposition (MJED) technique as described in the work Marchesseau et al. (2010).

3.1.2 Discretization in numerical simulations

In the discretized problem, the formulation in constitutive laws explicitly describes the deformation-stress relation, governing the behavior for each finite element. Since $\mathbf{F}$ is constant over the element, we have the total strain energy of this element as:

$$E = \int_V \Psi(\mathbf{F}) dV = V\Psi(\mathbf{F}) \tag{3.10}$$

with V the undeformed volume of the element. Following the definition in Equation (3.1), we can compute $\mathbf{F}$ with the currently deformed position $\mathbf{q}$ and the undeformed position $\mathbf{q}^{\text{ref}}$:

$$\mathbf{F} = \mathbf{D}_s(\mathbf{q})\mathbf{D}_m(\mathbf{q}^{\text{ref}})^{-1} \tag{3.11}$$

where $\mathbf{D}_s$ and $\mathbf{D}_m$ are tensors computed with $\mathbf{q}$ and $\mathbf{q}^{\text{ref}}$, respectively. For example, as the most widely-used discretization type in 3D problems, a tetrahedron element has $\mathbf{D}_s$ and

$\mathbf{D}_m$ like:

$$\begin{aligned}\mathbf{D}_s &= \begin{bmatrix} \mathbf{q}_{1x}-\mathbf{q}_{0x} & \mathbf{q}_{2x}-\mathbf{q}_{0x} & \mathbf{q}_{3x}-\mathbf{q}_{0x} \\ \mathbf{q}_{1y}-\mathbf{q}_{0y} & \mathbf{q}_{2y}-\mathbf{q}_{0y} & \mathbf{q}_{3y}-\mathbf{q}_{0y} \\ \mathbf{q}_{1z}-\mathbf{q}_{0z} & \mathbf{q}_{2z}-\mathbf{q}_{0z} & \mathbf{q}_{3z}-\mathbf{q}_{0z} \end{bmatrix} \\ \mathbf{D}_m &= \begin{bmatrix} \mathbf{q}^{\text{ref}}_{1x}-\mathbf{q}^{\text{ref}}_{0x} & \mathbf{q}^{\text{ref}}_{2x}-\mathbf{q}^{\text{ref}}_{0x} & \mathbf{q}^{\text{ref}}_{3x}-\mathbf{q}^{\text{ref}}_{0x} \\ \mathbf{q}^{\text{ref}}_{1y}-\mathbf{q}^{\text{ref}}_{0y} & \mathbf{q}^{\text{ref}}_{2y}-\mathbf{q}^{\text{ref}}_{0y} & \mathbf{q}^{\text{ref}}_{3y}-\mathbf{q}^{\text{ref}}_{0y} \\ \mathbf{q}^{\text{ref}}_{1z}-\mathbf{q}^{\text{ref}}_{0z} & \mathbf{q}^{\text{ref}}_{2z}-\mathbf{q}^{\text{ref}}_{0z} & \mathbf{q}^{\text{ref}}_{3z}-\mathbf{q}^{\text{ref}}_{0z} \end{bmatrix}\end{aligned} \tag{3.12}$$

where the indices represent the vertex index in the element and their position component on each axis (e.g., $\mathbf{q}_{1x}$ represents the position component on x-axis for the vertex 1). For the purpose of eliminating the rigid transitions, the tensors are computed with the intervals between vertices (instead of the positions), representing the "relative position" of the vertices. Since $\mathbf{D}_m$ only depends on the undeformed shape $\mathbf{q}^{\text{ref}}$ that is generally considered as constant, its inversion $\mathbf{D}_m(\mathbf{q}^{\text{ref}})^{-1}$ and the undeformed volume V are computed in the initialization process, and are considered as constant in the simulation loop.

Consequently, the total strain energy of the system can be computed with the position at the beginning of each time step. Integrating the time step requires minimizing the total energy in the system, including the strain energy and the kinematic energy:

$$\underset{\mathbf{q}}{\operatorname{argmin}}\, E_{\text{total}}(\mathbf{q}) = \underset{\mathbf{q}}{\operatorname{argmin}} \Big[\underbrace{\sum_k E_k(\mathbf{q})}_{\text{strain energy}} + \underbrace{\frac{1}{2}\dot{\mathbf{q}}^{\mathrm{T}}\mathbf{M}\dot{\mathbf{q}}}_{\text{kinematic energy}} \Big] \tag{3.13}$$

where k indicates the finite element indices, and $\mathbf{M}$ refers to as the diagonal mass matrix. Solving this optimization problem is equivalent to finding the zero derivative of the function:

$$(\nabla E)^{\mathrm{T}} + \mathbf{M}\ddot{\mathbf{q}} = \mathbf{0} \tag{3.14}$$

where the gradient of strain energy ∇E refers to the internal elastic forces. This formulation is actually equivalent to the Newton's second law (to be discussed in the next section) without external forces.

3.1.3 Implicit Euler integration

For each independent object in a multi-object system, a general description for the physical behavior can be expressed through the Newton's second law:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{e} - \mathbf{f} + \mathbf{c} \tag{3.15}$$

where the derivative of the velocity $\ddot{\mathbf{q}}$ is integrated with the external forces $\mathbf{e}$, the internal forces $\mathbf{f} = \mathscr{F}(\mathbf{q}, \dot{\mathbf{q}})$, and the constraint forces $\mathbf{c}$ (to be discussed in Chapter 4).

In the context of interactive simulations, an important choice is the time integration scheme. Indeed, explicit methods have been widely used for medical simulations Joldes et al. (2009).

$$\begin{aligned} \dot{\mathbf{q}}_{t+h} &= \dot{\mathbf{q}}_t + h\ddot{\mathbf{q}}_t \\ \mathbf{q}_{t+h} &= \mathbf{q}_t + h\dot{\mathbf{q}}_{t+h} \end{aligned} \tag{3.16}$$

where h is the length of time interval $[t, t+h]$. The unknown motion $\ddot{\mathbf{q}}_{t+h}$ depends on the mechanical states at the time step beginning.

$$\ddot{\mathbf{q}}_t = \mathbf{M}^{-1}[\mathbf{e} - \mathscr{F}(\mathbf{q}_t, \dot{\mathbf{q}}_t) + \mathbf{c}] \tag{3.17}$$

In this case, the solution only involves the (diagonal) mass matrix $\mathbf{M}^{-1}$ leading to very fast, simple to implement, and parallel solutions Comas et al. (2008). Unfortunately, user interactions may introduce sudden and stiff contacts at arbitrary location/frequency, which raises stability issues.

On the opposite, as discussed in Section 2.1.1, implicit schemes provide better control of the residual vector. To have a balance between different forces at the end of time steps, we choose to integrate the time step t with a backward Euler scheme:

$$\begin{aligned} \dot{\mathbf{q}}_{t+h} &= \dot{\mathbf{q}}_t + h\ddot{\mathbf{q}}_{t+h} \\ \mathbf{q}_{t+h} &= \mathbf{q}_t + h\dot{\mathbf{q}}_{t+h} \end{aligned} \tag{3.18}$$

which leads to an implicit differential–algebraic equation (DAE):

$$\ddot{\mathbf{q}}_{t+h} = \mathbf{M}^{-1}[\mathbf{e} - \mathscr{F}(\mathbf{q}_{t+h}, \dot{\mathbf{q}}_{t+h}) + \mathbf{c}] \tag{3.19}$$

For soft solids, the non-linear function of internal forces is linearized with a first-order Taylor expansion Baraff (1996):

$$\mathscr{F}(\mathbf{q}_{t+h}, \dot{\mathbf{q}}_{t+h}) = \mathscr{F}(\mathbf{q}_t, \dot{\mathbf{q}}_t) + \frac{\partial \mathscr{F}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \mathbf{q}} h\dot{\mathbf{q}}_{t+h} + \frac{\partial \mathscr{F}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}} h\ddot{\mathbf{q}}_{t+h} \tag{3.20}$$

This linearization corresponds to the first iteration of the Newton-Rapson method. The incomplete approximation may cause numerical errors of the dynamic behavior, but they lean towards decreasing at equilibrium.

In the practice of finite-element simulations, the partial derivative terms are expressed as matrices: $\frac{\partial \mathscr{F}}{\partial \dot{\mathbf{q}}}$ at $(\mathbf{q}_t, \dot{\mathbf{q}}_t)$ as a damping matrix $\mathbf{B}$, and $\frac{\partial \mathscr{F}}{\partial \mathbf{q}}$ at $(\mathbf{q}_t, \dot{\mathbf{q}}_t)$ as a stiffness matrix $\mathbf{K}$. The dynamic equation for soft solids finally results in a second order differential equation:

$$\underbrace{\left[\mathbf{M} + h\mathbf{B} + h^2\mathbf{K}\right]}_{\mathbf{A}} \Delta\dot{\mathbf{q}} = \underbrace{(h\mathbf{e}_t - h\mathbf{f}_t) - h^2\mathbf{K}\dot{\mathbf{q}}_t}_{\mathbf{b}} + h\mathbf{c} \tag{3.21}$$

with $\mathbf{f}_t = \mathscr{F}(\mathbf{q}_t, \dot{\mathbf{q}}_t)$. For both rigid and soft solids, we have a common formulation of a linear system $\mathbf{A}\Delta\dot{\mathbf{q}} = \mathbf{b} + h\mathbf{c}$ to be solved.

3.1.4 Matrix-free iterative solver

In FE simulations, the linear problem $\mathbf{Ax} = \mathbf{b}$ can be either solved with a direct solver or an iterative solver. The iterative solvers are usually preferred in real-time applications as it is easier to control the computation costs within a time step. An iterative solver (such as CG) that requires a matrix-vector multiplication operation in each iteration $\mathbf{z} = \mathbf{Ay}$. Generally, the CG algorithm requires assembling $\mathbf{A}$ before the iterations. Such assembly cost is usually less critical than the solving process itself, but it may become the bottleneck when combined with efficient solvers.

To parallelize the SpMV operations, the matrix-free method proposes not to assemble matrix $\mathbf{K}$, but instead to perform the calculations in parallel using elemental matrices $\mathbf{K}_e$:

$$\mathbf{z} = \mathbf{Ay} = a\mathbf{My} + b\mathbf{Ky} = a\mathbf{My} + b\sum \mathbf{G}\mathbf{K}_e\mathbf{y}_e\mathbf{G}^{\mathrm{T}} \tag{3.22}$$

where $\mathbf{G}$ refers to the sparse global matrice that distributes stiffness values throughout the global system, and $\mathbf{y}_e = \mathbf{G}^{\mathrm{T}}\mathbf{yG}$ refers to the corresponding local part of $\mathbf{y}$. The global problem is then transformed as resolving the local ones $\mathbf{K}_e\mathbf{y}_e$ in parallel and accumulate their contributions to the final force. However, different elements may share same vertices that determines the accumulation position in the result vector, leading to writing conflicts. Allard et al. (2011) provides efficient parallelization for this problem on GPUs. To efficiently parallelize the computation while avoiding writing conflicts, the resolution on the GPU is implemented by two steps, separated by a global synchronization:

As illustrated in Figure 3.1, a first kernel is used where each thread is associated with an element e. Each thread then computes the partial contribution of the force within element e using the element stiffness matrix $\mathbf{K}_e$. Since all elements have the same number

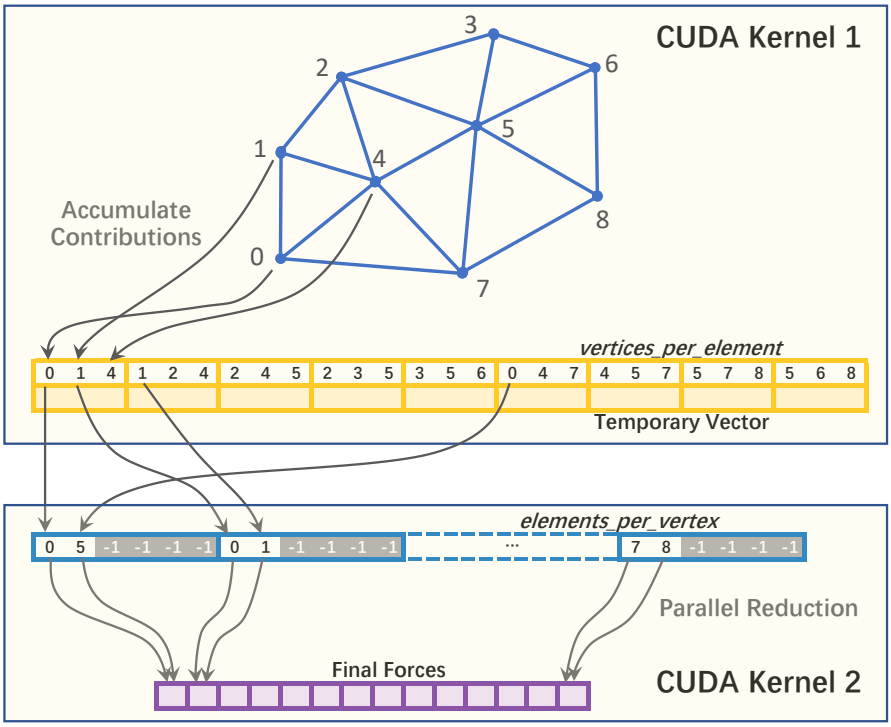


Figure 3.1: In each iteration of the matrix-free solver, accumulating the contributions to the final forces is implemented by two GPU-based kernels.

of degrees of freedom, vertex IDs are used to determine a unique memory address in a temporary array. A GPU structure *vertices_per_elements*, which contains the vertices connected to each element, is used to collect the contribution from each element into the temporary array.

After a global synchronization, a second kernel is used to sum up the partial contribution for all nodes that are connected to multiple elements. This requires knowing the address of all contributions from the same node in the temporary vector. This information acan be expressed as a deterministic mapping *elements_per_vertex* that defines the neighbor elements connected to each node. To facilitate the reading operation of *elements_per_vertex*, the solver allocates the same GPU memory size for each node, where the size is determined by the maximum number of elements connected to the same node in the mesh. Then the element indices are stored into *elements_per_vertex*, while the absent data is coded as -1 (see Figure 3.1). Consequently, this allows to launch a thread per vertex, since each thread knows the start position in *elements_per_vertex* to accumulate. The next step is to merge the partial contributions in each node, which requires handling the writing conflicts. Since the mapping *elements_per_vertex* groups the contributions by nodes, it will be straightforward to address the merging by parallel reduction operations. Finally, the contributions are accumulated into the final force.

3.2 Matrix assembly in finite element simulations

3.2.1 Fast Matrix assembly strategy

In order to solve the dynamic equation (3.21), the Conjugate Gradient algorithm is generally proposed to solve this problem since **A** is large, sparse, symmetric and positive definite. The highly efficient GPU-based solver Allard et al. (2011), discussed in 3.1.4 , offers a matrix-free approach without the need to assemble the system matrix **A**. However, implementing this method for generic hyperelastic materials proves to be highly invasive in the code, as the GPU-based algorithm is specifically designed and optimized for tetrahedron elements modeled with either a linear constitutive law or the co-rotational formulation. The divergence of formulations in different models (as discussed in Section 3.1) also exposes the challenge to parallelize the numerical resolution in a consistent manner. Conversely, the traditional solving strategy involves solving the system with the assembled matrix. Generic constitutive models can take advantage of GPU-based matrix operations, but the process of matrix assembly typically incurs an overhead cost that cannot be ignored.

To solve this issue, we propose a novel approach to matrix assembly that satisfies both efficiency and generality requirements. Our fast assembly method capitalizes on the fact that the same assembly procedure is called during each time integration. By constructing the system matrix in sequential order, we exploit the inherent topological structure, which guarantees a definitive sequence for filling elements into the matrix and determines the sparsity pattern of **A**. Consequently, we can establish a specific mapping from the filling element sequence to the final matrix pattern. This deterministic mapping replaces the time-consuming sorting of the initial filling sequence into the final sparse format, resulting in a substantial acceleration of the matrix assembly process.

Figure 3.2 provides an overview of the assembly procedure workflow of our method, consisting of the following steps:

1. **Collect data**: Gather mass and stiffness data for each element and store it in a triplet format, consisting of the row index, column index, and corresponding value (see illustraion in Figure 3.3). Using the triplet format to store primary data collected from the mesh is a common strategy in many assembly methods, including the implementation using Eigen's library.

2. **Build matrix pattern**: Arrange the collected triplet data in ascending order based on the row and column indices. This step becomes necessary only if structural modifications have been identified during the data collection phase.

3. **Compress**: Construct the system matrix in Compressed Sparse Row (CSR) format.

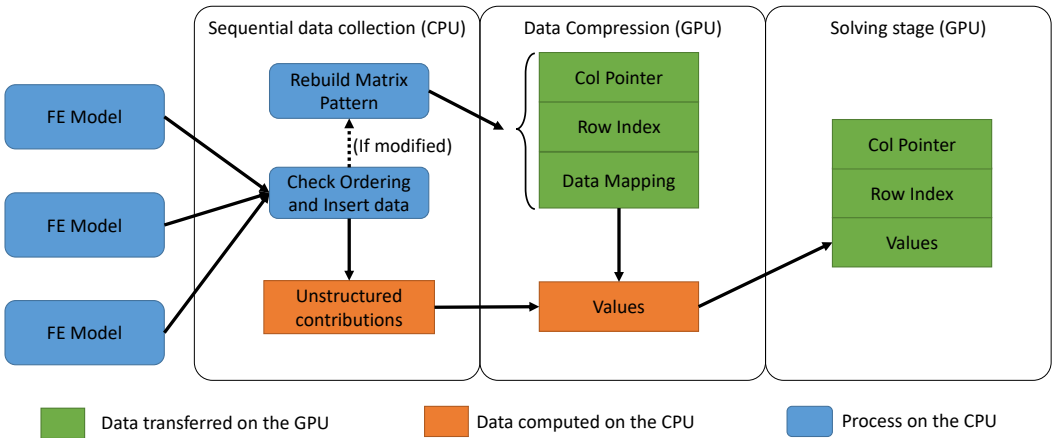


Figure 3.2: General workflow of the matrix assembly procedure. **Data Mapping** corresponds to the additional structure used to compress the unstructured contributions (see section 11) in CSR format. It is computed and sent on the GPU only once until no modifications of the fill ordering are detected during the collection phase.

The current matrix assembly method relies on the assumption that the topology remains invariant. Topological modifications are not addressed in this chapter, but the method remains generic since the topology modification only occurs in specific cases, such as cutting operations. A further discussion in Chapter 5 reveals the possibility to efficiently update the matrix pattern while cuts occur. Additionally, applying the asynchronous preconditioning in such case of sudden changes can be addressed with specific correction on the preconditioner Courtecuisse et al. (2014).

Collect data The matrices **M** and **K** appearing in equations (3.21) are constructed by summing the local contributions from each element into the corresponding global matrices. Initially, the values are stored in a collection of triplets, which is a data structure consisting of three variables: the row index, column index, and corresponding value. Since the triplet vector represents the original process of filling elements into the matrices, the sequence of row/column indices is unsorted and uncompressed[1], but remains definitive in each time integration step. Nevertheless, due to the frequency at which contributions are inserted into the triplet list, it is crucial to optimize this process as much as possible.

[1] meaning that a pair of row/column indices may appear multiple times when filling the matrices

Algorithme 1 : Procedure used to add value in the matrix.

```
1  Function Add(row, col, val):
2      if keepStruct & id < prevVal.size() &
         prevCol[id] = col & prevRow[id] = row then
3          prevVal[id] = val ;
4      end
5      else
6          keepStruct = false;
7          prevRow[id] = row ;
8          prevCol[id] = col ;
9          prevVal[id] = val ;
10     end
11     id=id+1;
```

The pseudocode for the **Add** function, responsible for inserting contributions into the matrices, is presented in Algorithm 1, and it is exposed to the FE models to incorporate their respective contributions. In the algorithm, the boolean variable **keepStruct** is employed to detect any modification in the filling order. The index variable **id** indicates the next writing address in the uncompressed arrays (**prevRow**, **prevCol**, and **prevVal**) that correspond to the list of triplets added in the previous time steps.

For each inserted value, the test performed in line 2 verifies the consistency of the pattern with respect to the previously constructed matrices. This test incurs an overhead but is necessary to identify any changes in the matrix structure. However, if no modifications are detected, only the value **val** is stored (line 3). This approach leverages the CPU cache and minimizes write operations.

Build matrix pattern Consider a generic matrix **X** that needs to be assembled, such as **M** and **K**. To construct the final CSR format for **X**, A method inspired by Eigen's library. This method involves computing the transpose of the matrix twice to facilitate value sorting.

To store the temporary matrices, we introduce a format called the "uncompressed structure," which bears similarities to the CSR format. Like the CSR, an arranged row pointer encodes the index in the arrays of column index and values that are unsorted and uncompressed (duplicate indices exist). We summarize the different stages of the assembly process in Table 3.1:

1. Initially, the temporary transposed matrix $\mathbf{X}^{\mathrm{T}}$ is built using the uncompressed structure. Prior to computing the transposed matrix, we count the number of values per row to allocate the necessary memory. Then, the data can be moved to their ap-

propriate locations within the allocated structure. With the predetermined matrix structure, the sequence of row indices can be arranged with a time complexity of $\mathcal{O}(2n)$, while the sequence of column indices within each row remains unsorted.

2. Similar to the previous step, $\mathbf{X}$ is built in the uncompressed structure by transposing $(\mathbf{X}^{\mathrm{T}})^{\mathrm{T}}$. The second transpose results in the initial matrix $\mathbf{X}$, where the values are sorted both by rows and columns, while the structure remains uncompressed.

3. Finally, the elements in the same position are merged, transforming the **uncompressed structure** into the CSR format.

Matrix	$\mathbf{X}$	$\mathbf{X}^{\mathrm{T}}$	$\mathbf{X}$	$\mathbf{X}$
Format	triplet set	uncompressed structure	uncompressed structure	CSR
Row	unsorted uncompressed	sorted compressed	sorted compressed	sorted compressed
Column & Values	unsorted uncompressed	unsorted uncompressed	sorted uncompressed	sorted compressed

Table 3.1: State of storage format at different stages in matrix assembly process

One notable distinction from Eigen's implementation is that the values of the transposed matrices are not directly stored in memory. This characteristic enables a strategy called *fast assembly* in Figure 3.4. Exploiting the assumption that the mesh topology remains unchanged, the filling order and the sparsity pattern (row pointer and column index arrays in CSR) can be reused. As long as the filling order remains unaltered during the collection stage, we propose constructing a mapping C from the initial triplet set to the CSR format, significantly enhancing the efficiency of building the value array. The main operation involves merging duplicated values in the triplet array that has been arranged using the deterministic mapping C. This operation must be performed at each time step, but it can be easily parallelized on both CPU and GPU since the addresses of values in the CSR format are known and unique. Importantly, parallelization can be accomplished without impacting the code of the constitutive models that generate the matrices, making our method highly efficient for any generic constitutive model. The deterministic mapping C can be reused as long as no modifications in the filling order are detected during the previous stage. However, if modifications are detected during the collection stage, a complete rebuild of the matrix pattern (referred to as *full assembly* in Figure 3.3) is performed. In this mode, the method exhibits similar performance to the default implementation of Eigen's library.

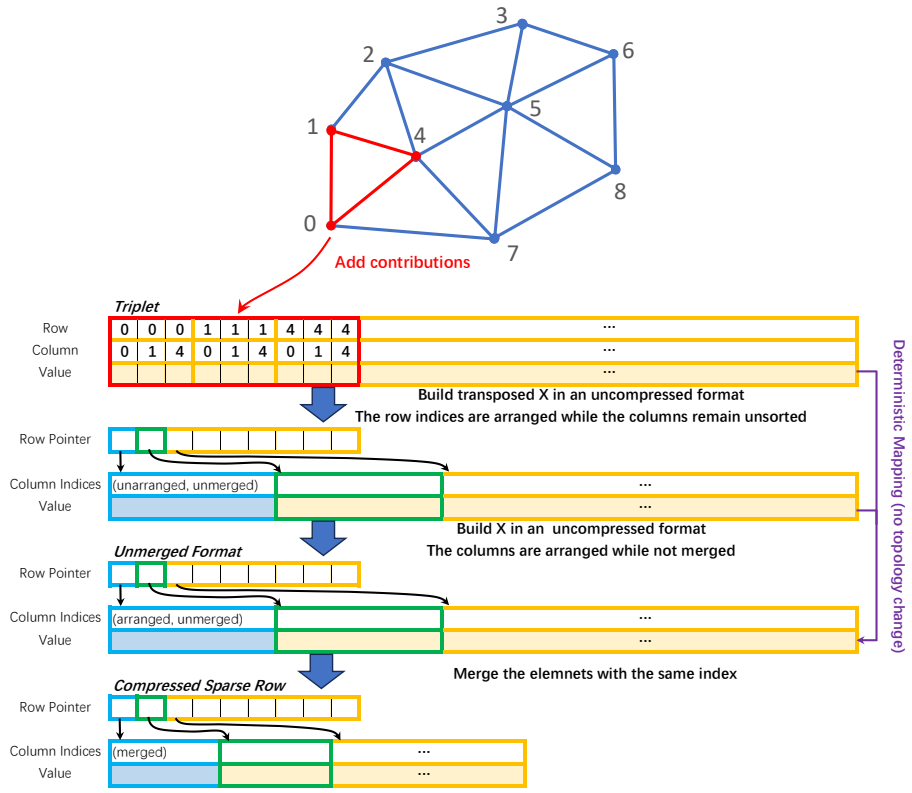


Figure 3.3: Full Assembly: build matrix pattern and mapping vector from triplets set

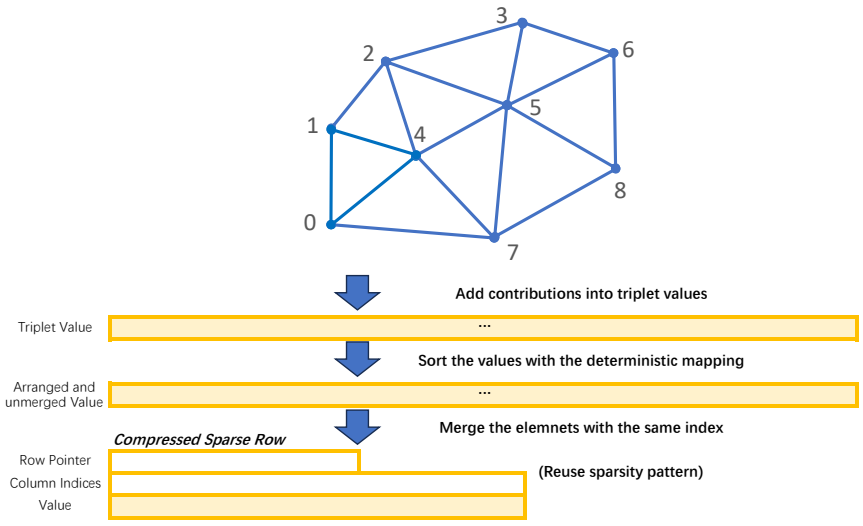


Figure 3.4: Fast Assembly: assemble the uncompressed format using the deterministic mapping.

In order to get the global matrix **A** and the vector **b** in equation (3.21), we may note that

both are generated from the sum of the same matrices **M** and **K** with various coefficients:

$$\begin{aligned} \mathbf{A} &= (1 + h\alpha)\mathbf{M} + h(h + \beta)\mathbf{K} \\ \mathbf{b} &= \mathbf{e}_t - \mathbf{f}_t - h\mathbf{K}\dot{\mathbf{q}}_t \end{aligned} \tag{3.23}$$

where the coefficients only depend on the time step and the Rayleigh damping constant during the entire simulation.

One advantageous consequence of our approach is the ability to merge the computation of the right-hand side and left-hand side terms into a single procedure. This consolidation allows for the extraction of a substantial amount of data that is highly suitable for GPU architectures. Moreover, it benefits from cache optimization since the mapping C is accessed twice.

Once the vector of values is compressed, the CSR format can be directly employed within a parallel CG solver, either on the CPU or the GPU[2]. To achieve this, parallelization of the sparse matrix-vector product (*SpMV*) operation is required, which is straightforward when working with the assembled system. Numerous efficient implementations exist for this common operation on both CPUs and GPUs. In this study, we utilize the *SpMV* implementation provided by the CUSPARSE library developed by NVIDIA. This choice can significantly enhance the efficiency of the CG iterations, resulting in a noteworthy acceleration of the entire simulation without necessitating modifications to the code responsible for matrix generation.

3.2.2 Evaluation of matrix assembly strategy

3.2.2.1 Performances of matrix assembly

To evaluate the performance of the proposed matrix assembly strategy, the simulation tests are conducted in the open-source SOFA framework with a CPU Intel@ core i9-9900k at 3.60GHz and a GeForce RTX 2070 8 Gb.

Our matrix assembly strategy aims to reach a compromise between the computation cost and the versatility of the code by assembling the matrix **A** with low cost. This section compares the matrix building time between the current assembly method and the standard assembly method implemented in the Eigen library. The simulation tests for the assembly stage are executed with a group of deformable mesh representing the shape of a raptor with various mesh resolutions (see table 3.2).

[2]Note that if the compression is performed on the GPU, the vector of values is already available on that architecture. The row index and column pointer only need to be transferred if the mapping is modified.

Example	Raptor 1	Raptor 2	Raptor 3
Nodes	2996	4104	5992
Tetra	8418	12580	19409

Table 3.2: Number of nodes and tetrahedral elements of the meshes.

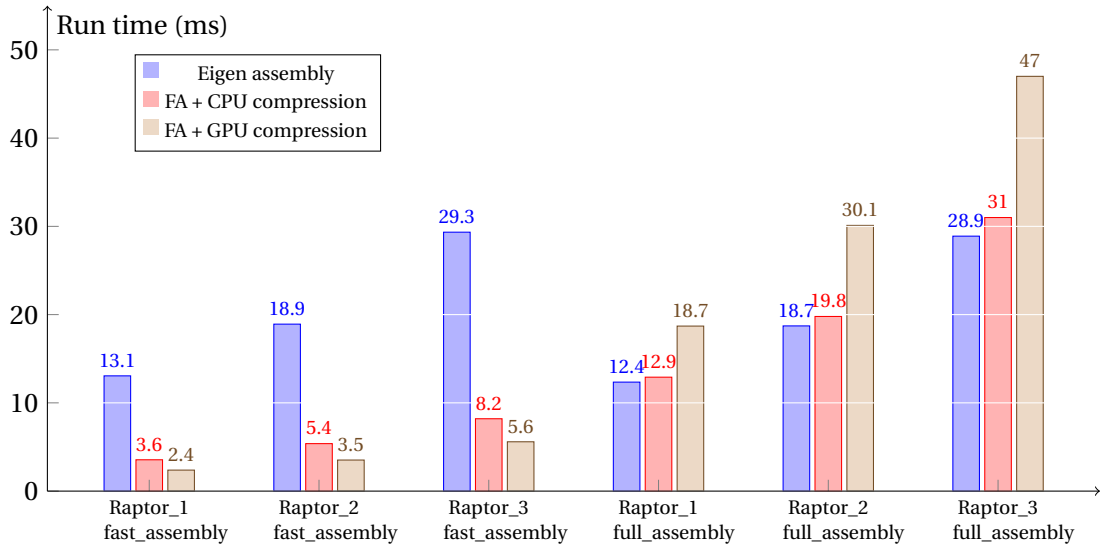


Figure 3.5: Computational costs for the matrix assembly in **fast_assembly** mode (operations excluding the re-computation of the compression mapping C) and **full_assembly** mode (when rebuilding the matrix pattern) introduced in Section 11. We compare the performance between the Eigen's library implementation (**Eigen assembly**) and our fast assembly strategy (**FA**) with the compression performed on the CPU (with 8 CPU threads) or on the GPU.

The figure 3.5 shows the performances of the assembling stage, including the accumulation of triplets and the compression to the CSR format but excluding the computation of the mapping C. With the exception of the first time step where the mapping is actually computed, it corresponds to the standard performances obtained during the entire simulation with the various assembly methods. Compared with the standard method using Eigen library, the current method on CPU reduces by 72% time cost of building on average. This cost reduction rises to 81% for the fast assembly method on the GPU. The compression on the GPU provides a speedup of between 2.7× to 3× with respect to the parallel implementation of the compression on the CPU using 8 threads.

If topological modifications are performed or if the filling order is modified, the matrix pattern needs to be rebuilt. In this case, the building cost, including the computation of the pattern, is measured in the figure 3.5. The time cost of the current method on CPU when the matrix pattern is rebuilt is slightly slower than the Eigen implementation, but it remains in the same order. The overhead is due to the additional computation of the

index vector mapping C providing the position of the triplets in the CSR format. However, the cost is balanced because the mapping can be reused for the next time steps. Indeed, reusing the mapping for only two consecutive time steps already provides an acceleration compared to the Eigen implementation. Since the computation of the mapping is performed on the CPU, the GPU-based compression suffers a slowdown due to data transfers between the CPU and the GPU.

3.2.2.2 Performances with the CG solver

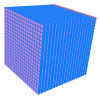
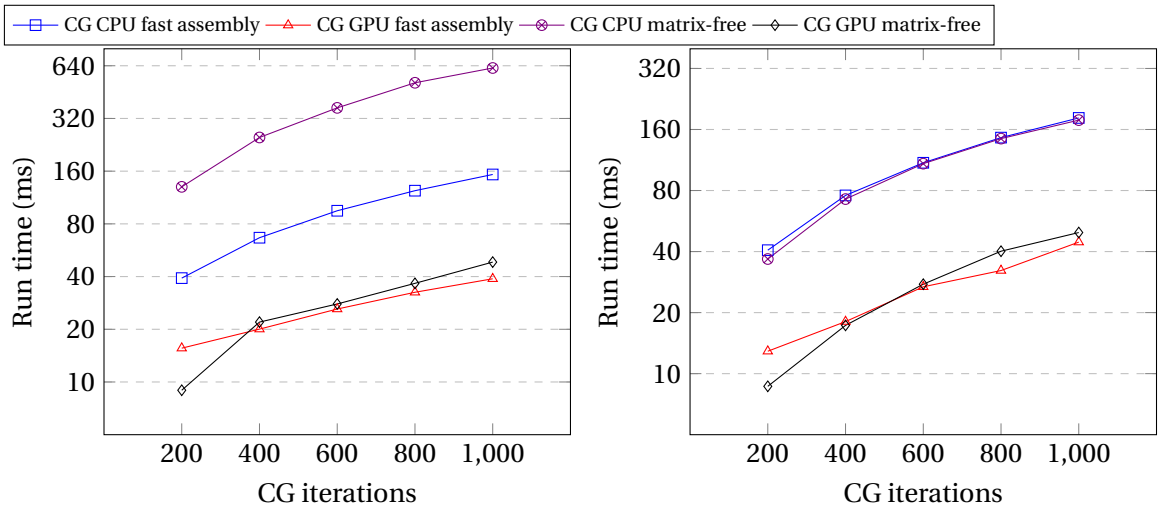
Example	Liver	Cloth	Cube	Raptor
Model	Co-rotational			Hyperelastic
Type	Tetrahera	Triangle	Hexahedra	Tetrahedra
Nodes	2660	4900	8000	2996
Nb.element	12328	9522	6859	8418
Mesh				

Table 3.3: Configurations of different scenario examples.

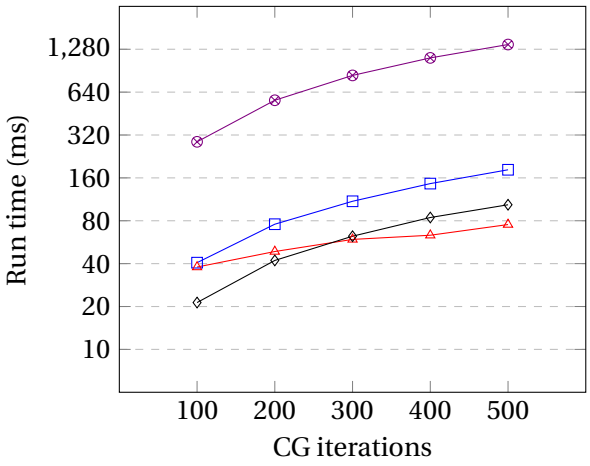
The performances of the global simulation are now compared in a complete simulation of a deformable body, including the time for the computation of the FE model, the assembling step and the solving process. Performances of the fast assembly method combined with a Conjugate Gradient solver (**CG GPU fast assembly**) is measured and compared with both a CPU-based matrix-free implementation of the Conjugate Gradient (**CG CPU matrix-free**) and with the method introduced in Allard et al. (2011) which includes a matrix-free GPU-based Conjugate Gradient (**CG GPU matrix-free**) for the tetrahedral co-rotational model.

In order to verify the generality of the proposed solution, the specific GPU-based implementation introduced in Allard et al. (2011) has been extended for other types of elements (triangles and hexahedron), requiring the development of specific code for each model on the GPU. In addition, the fast assembly method is also tested for hyperelastic material laws. However, since developing an efficient GPU parallelization is not trivial, the method is only compared with CPU-based matrix-free solvers. The scenarios are illustrated in Table 3.3.

For the scenarios in Figure 3.6, the run time increases linearly along with the number of iterations. The fast assembly method combined with the GPU-based CG is up to 16× faster than the standard CPU method implemented in SOFA and reaches the same



(a) Liver mesh (tetrahedron elements) **(b)** Cloth mesh (Triangular elements)

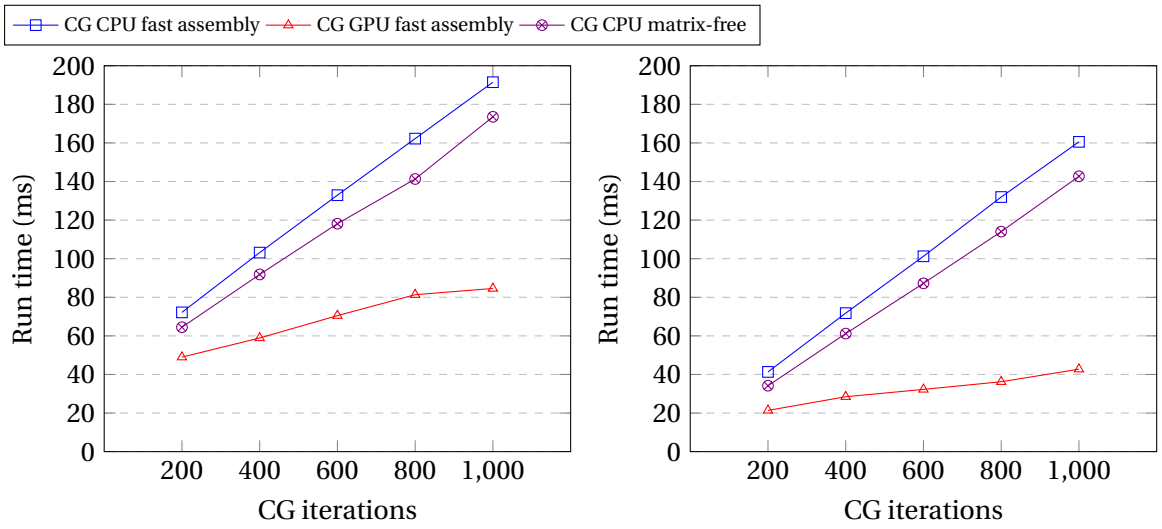


(c) Cube mesh (hexahedron elements)

Figure 3.6: Computation time of a single time step for different examples modeled with the co-rotational formulation for various (fixed) number of CG iterations. The figures share the same legend on the top. The details of different examples can be found in Table **??**. We note that the y axis is logarithmic in these figures.

computation cost level as the GPU-optimized method. The Fast Assembly method suffers a slowdown compared to the GPU matrix-free method with fewer iterations, but this case is inverted when the iteration increases. This is due to the fact that the fast assembly method takes time to build the matrix, but this overhead is compensated at each CG iteration since the parallel implementation of the *SpMV* operation is faster with the assembled matrix.

It's important to note that although performances are comparable to the GPU-based matrix-free implementation, the code of the co-rotational model is written for the CPU where optimizations are simply obtained by calling the **Add** function of the algorithm 1, which is completely transparent for the code and enforces the compatibility with the rest



(a) Mooney-Rivlin model **(b)** St-Venant-Kichhoff model

Figure 3.7: Computation time of a single simulation step with hyperelastic models implemented in SOFA for various (fixed) number of CG iterations per time step. The figures share the same legend on the top. The scenarios simulate a tetrahedron raptor mesh (see Table 3.3).

of the models implemented in the SOFA framework. In addition, for computers without GPU-compatible hardware, the *SpMV* operation can also be parallelized on the CPU. The method **CG CPU fast assembly** uses 8 threads to perform the matrix-vector product, which leads to a speedup of up to 4.13× compared to the sequential method (see the figure 3.6a).

In the figure 3.7a, the method is directly tested with the Mooney-Rivlin material using the implementation of the MJED Marchesseau et al. (2010) provided in SOFA, without any modification of the code. The main difference with the co-rotational formulation lies in the fact that the computation of the hyperelastic formulation is significantly slower, and thus the time spent in the assembling and solving processes is smaller. Therefore, the benefits of the CPU parallelization with 8 threads (**CG CPU fast assembly**) is balanced by the overhead of assembling the matrix compared to the matrix-free version (**CG CPU matrix-free**). However, the GPU-based internal parallelization of the assembling and solving process provides a speedup between 1.31× and 2.05×. This represents the fastest method for nonlinear materials available in SOFA because no specific GPU-based parallelization of the MJED method is available. The method is also tested with the St Venant-Kirchhoff model using the MJED implementation. Compared to the Mooney-Rivlin material, the model is less complex so that the computation of the hyperelastic formulation is less costly. In the figure 3.7b, the fast assembly method gains a speedup between 1.60× and 3.34× compared to the matrix-free method, which is the fastest current implementation

for the nonlinear model in SOFA.

3.3 Linear system resolution

3.3.1 Asynchronous preconditioner

Following the matrix assembly, GPU-based implementations, such as NVIDIA's cuSPARSE, can effectively parallelize the *SpMV* operations within the CG solver, leading to accelerated computation costs in each CG iteration. However, another crucial aspect of numerical solvers is the number of iterations required for convergence.

A common strategy to reduce the iteration count in an iterative solver is to employ preconditioning techniques. In linear algebra, a preconditioner $\mathbf{P}$ approximates (through various methods) the inverse of the system matrix $\mathbf{A}$ such that $\mathbf{PA}$ has a smaller condition number compared to $\mathbf{A}$. A problem with a smaller condition number necessitates fewer iterations to achieve convergence in finding the solution.

Preconditioners can range from simple to precise, depending on the ability to approximate $\mathbf{A}^{-1}$ Simple preconditioners, such as the Jacobi preconditioner that selects the diagonal elements of the system matrix ($\mathbf{P} = \text{diag}(\mathbf{A})^{-1}$), are easy to construct but offer limited effectiveness in reducing the condition number. On the other hand, precise preconditioners like the Cholesky preconditioner provide extremely accurate approximations to $\mathbf{A}$, significantly enhancing convergence. However, they also suffer from expensive construction overhead costs that block the simulation loop.

To address this challenge, an efficient approach was proposed by Courtecuisse et al. (2010a), utilizing multithreading techniques to achieve a precise preconditioner without blocking the main simulation loop. This strategy, known as the *asynchronous preconditioner*, is accomplished by executing the expensive construction process in a dedicated thread.

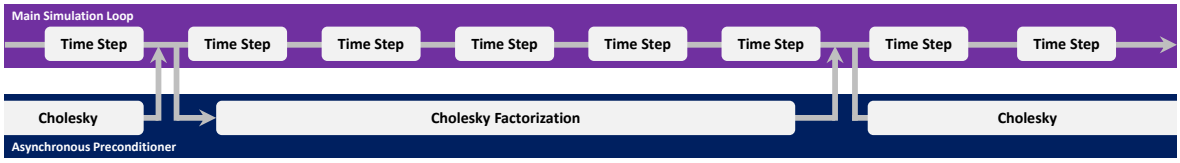


Figure 3.8: The scheme of the asynchronous preconditioner strategy.

Given the matrix $\mathbf{A}_t$ constructed at a specific time t, a preconditioner $\mathbf{P}$ can be derived through a parallel thread executing an $\mathbf{LDL}^{\mathrm{T}}$ factorization (a Cholesky-type decomposi-

tion):

$$\mathbf{P} = {\mathbf{A}_t}^{-1} = (\mathbf{LDL}^{\mathrm{T}})^{-1} \tag{3.24}$$

where the factorization yields the matrices **D**, a diagonal matrix, and **L**, a sparse lower triangular matrix, enabling the expression $\mathbf{P} = {\mathbf{A}_t}^{-1} = (\mathbf{LDL}^{\mathrm{T}})^{-1}$.

The factorized matrices become available after the completion of the factorization, typically several time steps after t. These matrices are then utilized as a preconditioner under the assumption that **P** remains a reasonably accurate approximation of the inverse of the current matrix $\mathbf{A}_{t+n}$. As described in Courtecuisse et al. (2010a), this method is highly efficient since the $\mathbf{LDL}^{\mathrm{T}}$ factorization only requires a few simulation steps (usually $n < 5$) for updating. The method proves to be highly efficient, typically requiring only 2 to 5 preconditioned CG iterations to achieve convergence with a predefined threshold of 10^{-9}.

3.3.2 Hybrid numerical solution

Considering the *SpMV* operations in CG solver with a $\mathbf{LDL}^{\mathrm{T}}$ preconditioner:

$$\mathbf{z} = \mathbf{PAr} = (\mathbf{LDL}^{\mathrm{T}})^{-1}\mathbf{Ar} \tag{3.25}$$

where **r** is the residual vector in each CG iteration, **z** is the preconditioned residual vector, and $\mathbf{LDL}^{\mathrm{T}}$ refer to as the approximation of the factorized **A**.

In Algorithm 2, the preconditioner is applied in the CG by two steps of STS.

Algorithme 2 : Applying a $\mathbf{LDL}^{\mathrm{T}}$ preconditioner in *SpMV* operations

1 $\mathbf{v} = \mathbf{Ar}$
2 $\mathbf{s} = \mathbf{L}^{-1}\mathbf{v}$
3 $\mathbf{z} = \mathbf{L}^{-\mathrm{T}}(\mathbf{D}^{-1}\mathbf{s})$

The forward and backward substitutions in *gaussian elimination* result in similar principle in solution. Therefore we focus on discussing the parallelization challenges associated with the forward substitution step for lower triangular systems. We recall the main obstacle for parallelizing a general lower triangular system $\mathbf{Ls} = \mathbf{r}$ is that the solution $\mathbf{r}_j$ of a given row j depends on all previous solutions $\mathbf{s}_i$:

$$\mathbf{s}_j = \mathbf{r}_j - \sum_{i=0}^{i<j} (\mathbf{s}_i \mathbf{L}_{j,i}) \tag{3.26}$$

Due to numerous data dependencies, applying the preconditioner (i.e, STSs in steps 2 and 3) remains on the CPU in current implementation, while the primary SpMV operations in

the CG algorithm are processed on the GPU. This hybrid solving strategy necessitates significant data transfers between the processors. During each CG iteration, the processors are required to send the residual vector $\mathbf{v}$ from the GPU to the CPU to apply the preconditioner and then send the resulting vector $\mathbf{z}$ back to the GPU. These data transfers, in addition to their associated costs, impose multiple synchronizations between the CPU and GPU, which reduce the efficiency of the preconditioner. To address this issue, our objective is to develop a GPU-based preconditioner that is at least as efficient as its CPU-based counterpart.

3.3.3 Cholesky-type preconditioner on GPUs

We propose a GPU-based Cholesky-type preconditioner, which is inspired by the solver in Courtecuisse et al. (2014) for STS with multiple right-hand sides. Since the method in Courtecuisse et al. (2014) was originally designed for multiple RHS, when applied in the problem with a single RHS, the level of parallelism for multiple RHS will be left unused.

We present a GPU-based Cholesky-type preconditioner, which is inspired by the solver proposed in Courtecuisse et al. (2014) for solving systems with multiple right-hand sides (RHS). While the method in Courtecuisse et al. (2014) was originally developed for multiple RHS problems, when applied to a problem with a single RHS, the potential for parallelism stemming from multiple RHS remains untapped. To leverage this untapped parallelism, we introduce the domain decomposition technique, specifically employing the nested dissection algorithm. The nested dissection algorithm serves to reduce the matrix pattern filling by recursively dividing the mesh into two parts, ensuring that each part contains approximately the same number of vertices while keeping the dividing part at a smaller scale George (1973). As a result, the lower triangular matrix $\mathbf{L}$ is reorganized and partitioned into sub-domains, with the indices of these sub-domains determined by the nested dissection algorithm. By incorporating the nested dissection technique, we effectively introduce additional parallelism in the context of a single RHS problem, thereby utilizing the computational resources of the GPU more efficiently.

The reordering algorithm partitions the graph as follows in a local view:

$$\underbrace{\begin{bmatrix} \hat{\mathbf{L}}_a & & \\ & \hat{\mathbf{L}}_b & \\ \mathbf{V}_a & \mathbf{V}_b & \tilde{\mathbf{L}}_c \end{bmatrix}}_{\hat{\mathbf{L}}_{(a,b,c)}} \begin{bmatrix} \mathbf{s}_a \\ \mathbf{s}_b \\ \mathbf{s}_c \end{bmatrix} = \begin{bmatrix} \mathbf{r}_a \\ \mathbf{r}_b \\ \mathbf{r}_c \end{bmatrix} \tag{3.27}$$

where the *diagonal domains* **a** and **b** can be solved independently and the reordering algorithm guarantees that the *separator* **c** (which requires the solution of **a** and **b**) is as small as possible. The partition and reordering are processed recursively on *diagonal domains* **a** and **b** until the block size is small enough.

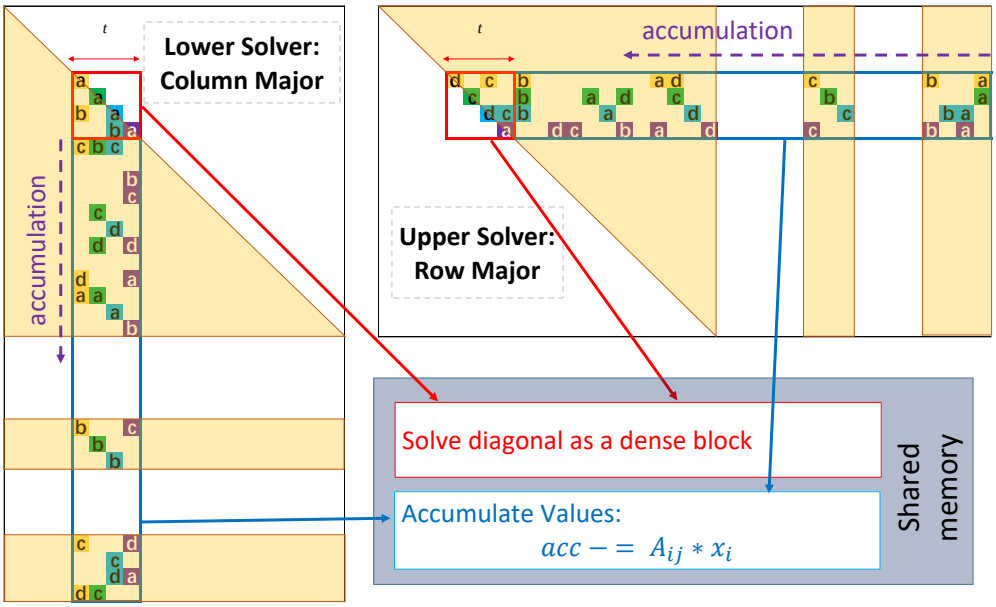


Figure 3.9: The solving stage for each subdomain is realized by GPU kernels, where contributions are accumulated in parallel. For the lower triangular system, the solution can be processed by column sequence (left), which pre-accumulates the data in higher levels, allowing sharing of the computation cost. On the other hand, when solving the upper triangular system, computation cost could be shared in lower levels, so the solution needs to be processed oppositely by row sequence (top-right).

According to the elimination tree, each subdomain identified in the lower triangular system $\mathbf{Ls} = \mathbf{r}$ is assigned a specific level of parallelization. The rule is that blocks with higher levels (left edge) require the solutions of lower-level blocks. Similarly, the upper triangular system problem $\mathbf{L}^{\mathrm{T}}\mathbf{s} = \mathbf{r}$ can be solved using the same method but with the computation sequence reversed, giving priority to blocks with higher levels, which exhibit fewer dependencies.

Within each level, we can employ the parallelization strategy presented in Courtecuisse et al. (2014) to solve each diagonal block ($\hat{\mathbf{L}}_a$, $\hat{\mathbf{L}}_b$) and the separator ($\mathbf{V}_a + \mathbf{V}_b + \tilde{\mathbf{L}}_c$) by processing rows in a sequential manner. This corresponds to the *Row Major* approach, where $t \times t$ threads are utilized to accumulate contributions, allowing for parallel processing of t rows simultaneously (in the current implementation $t = 16$). Due to the high dependencies, the diagonal part is treated separately as a dense matrix in shared

memory. A parallel reduction technique is then used to sum the contributions for each row, and finally, the $t \times t$ diagonal block is solved as a dense problem.

Alternatively, the *Column Major* approach is also feasible by pre-accumulating column contributions. In this case, the accumulation process of $\mathbf{V}_a$ and $\mathbf{V}_b$ is integrated into the kernels of $\hat{\mathbf{L}}_a$ and $\hat{\mathbf{L}}_b$, respectively, instead of solving the combined block ($\mathbf{V}_a + \mathbf{V}_b + \tilde{\mathbf{L}}_c$) in a single kernel. Since only the solution of $\hat{\mathbf{L}}_a$ (or $\hat{\mathbf{L}}_b$) is required, the accumulation of block $\mathbf{V}_a$ (or $\mathbf{V}_b$) can be performed in the same kernel. The part $\tilde{\mathbf{L}}_c$ can be solved as a diagonal block after the accumulation of $\mathbf{V}_a$ and $\mathbf{V}_b$. However, this pre-accumulation process may lead to data writing conflicts when multiple columns contribute to the same row simultaneously. To address this, the atomic add function provided by CUDA can handle the data conflicts automatically.

As depicted in Figure 3.9, to distribute the computational load in lower levels, the lower solver is implemented using the *Column Major*, while the upper system is solved using the *Row Major*. Our level-based parallelization strategy shares similarities with the approach in Yamazaki et al. (2020), but with several key differences:

1. Our solver utilizes the block-row parallelization strategy from Courtecuisse et al. (2014) to effectively exploit the GPU's parallel architecture (refer to Figure 3.9).

2. Our solver is optimized for FE simulations, where we continue to utilize the analysis results of parallelization levels until the matrix pattern changes.

3. Our solver benefits from the pre-accumulation technique, allowing for the sharing of computational costs in lower levels, resulting in improved efficiency (see Figure 3.9).

Data Transfer between processors In Section 3.3.4, we conducted a performance evaluation of our newly developed GPU-based preconditioner. The results, presented in Table 3.5, demonstrate that our method outperforms the CPU-based implementation across various examples. By replacing the CPU-based preconditioner with our GPU-based implementation, we achieve significant speedup in the solver and successfully address the data transfer issue between processors. As a result, our novel preconditioner enables the execution of a fully GPU-based preconditioned CG algorithm, requiring only a single scalar to be transferred from the GPU to the CPU at each iteration for convergence checking (refer to Figure 3.10).

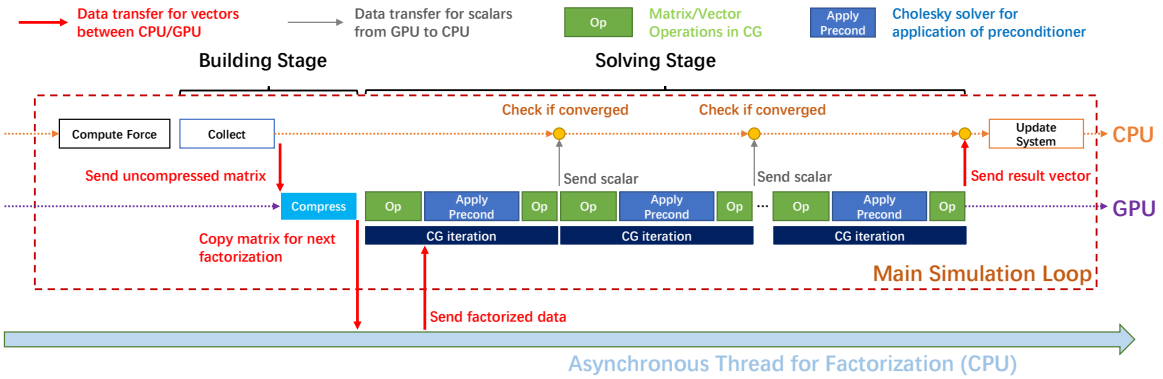


Figure 3.10: Workflow of the asynchronous preconditioning scheme. The **collect** phase is performed on the CPU with any generic implementation of FE models. The **Op** process corresponds to the necessary operation to perform one CG iteration. All of them are either *Spmv* or linear algebra operations on vectors that can be easily parallelized on the GPU. The application of preconditioner **Apply Precond** is also performed on the GPU, resulting in a preconditioned CG fully implemented on GPU. Only a scalar needs to be copied to the CPU in each iteration to check the convergence state.

3.3.4 Evaluation of preconditioned CG solver performance

The performances of different sparse $\mathbf{LDL}^{\mathrm{T}}$ solvers (including both the lower and upper triangular systems) are reported in the table 3.4. The proposed GPU-based parallelization relying on the nested dissection method (**GPU ND**) introduced in section 3.3.3 is $20.3-24.0\times$ faster than the GPU-based implementation provided in NVIDIA's cuSPARSE library. The main reasons lie in the fact that the cuSPARSE method requires performing the analysis of the data dependencies before actually solving the problem, and the parallelization strategies are optimal for much larger problems than the ones used in the context of real-time simulations. Such speedup compared to the golden-standard implementation (cuSPARSE library) is reported as maximally $5.8\times$ in Picciau et al. (2017) and $19.5\times$ in Yamazaki et al. (2020). The method is also compared with the GPU-based implementation proposed in Courtecuisse et al. (2014). As reported in this previous work, the GPU-based $\mathbf{LDL}^{\mathrm{T}}$ solver is $3\times$ slower than a sequential CPU implementation, whereas the (**GPU ND**) provides a speedup of $1.4-2\times$, enabling the possibility to solve the problem directly on the GPU.

The method is tested in complete simulations of deformable bodies with various constitutive laws (see Table 3.5). The tests are conducted with the same mesh group of raptors and solved with the asynchronous preconditioned CG. On average, the asynchronous preconditioner is updated every 2 to 4 simulation steps, which lead between 5 to 20 iterations (#**it**) according to different cases. Therefore, the asynchronous preconditioner

Mesh	Method	LDL solver	Lower	Upper
Raptor 1	CUSPARSE	13.46	3.33	2.48
	Courtecuisse et al. (2014) GPU	3.63	1.88	1.66
	CPU	1.13	0.52	0.58
	GPU ND	0.56	0.29	0.24
Raptor 2	CUSPARSE	22.77	5.63	3.91
	Courtecuisse et al. (2014) GPU	6.36	3.18	2.78
	CPU	1.97	0.90	1.04
	GPU ND	1.12	0.60	0.49
Raptor 3	CUSPARSE	44.96	11.02	6.97
	Courtecuisse et al. (2014) GPU	10.07	5.33	4.71
	CPU	3.94	1.77	2.14
	GPU ND	2.15	1.07	1.05

Table 3.4: Computation time (in ms) of various STS solvers

	#it	Method			Raptor 1	Raptor 2	Raptor 3
		Assembly	CG	Precond			
Corot	15	Fast Assembly	CPU	CPU	25.94	45.85	85.86
		Fast Assembly	**GPU**	**GPU**	**14.58**	**26.46**	**46.84**
		Matrix Free	CPU	CPU	43.18	65.65	118.33
		Matrix Free	GPU	CPU	31.00	48.52	80.50
MR	12	Fast Assembly	CPU	CPU	59.02	90.64	153.90
		Fast Assembly	**GPU**	**GPU**	**48.73**	**77.84**	**124.53**
		Matrix Free	CPU	CPU	56.76	89.76	152.11
SVK	8	Fast Assembly	CPU	CPU	23.16	37.96	67.03
		Fast Assembly	**GPU**	**GPU**	**16.60**	**26.46**	**42.86**
		Matrix Free	CPU	CPU	24.18	37.35	64.67

Table 3.5: Computation time (in ms) for various models: Corotational (**Corot**), Mooney-Rivelin (**MR**) and St-Venant-Kichhoff (**SVK**).

already provides a significant speedup with respect to the standard GPU-based CG. With the preconditioner, the matrix operations needed during the CG iterations are performed either with the fast assembly method or with the matrix-free method , either on the CPU

or the GPU when available. The preconditioner is explicitly built using the fast assembly method and applied on the CPU as done in Courtecuisse et al. (2014) or on the GPU with the method introduced in section 3.3.3.

The method **fast assembly + CG GPU + preconditioner GPU** is the fastest method and provides a speedup of between 1.7× and 2.1× for the co-rotational model compared to the solution proposed in Courtecuisse et al. (2014). An important advantage of the current solution is that the preconditioned CG is applied entirely on the GPU, without any need for data transfers or synchronizations between the CPU/GPU during the solving stage. In addition, since the matrix is assembled every time step, no additional overhead is introduced when the factorization needs to be recomputed.

As no GPU-based matrix-free method is implemented for hyperelastic models in SOFA, the comparison is made with the CG performed on the CPU. Although the computation cost of the hyperelastic formulation is significantly higher, the result of the proposed GPU version also provides a speedup from 1.15× to 1.22× for the Mooney-Rivlin material. For the St Venant-Kirchhoff material, where the model is more straightforward than the Mooney-Rivlin material, this speedup is raised from 1.41× to 1.51×.

3.4 Conclusion

In this chapter, we provided an overview of the background pertaining to physics-based simulation for deformable solids. By utilizing the constitutive laws derived from material mechanics, the simulation frameworks can effectively handle diverse scenarios with distinct requirements for deformation behavior and computational efficiency. However, the existence of various formulations presents challenges in parallelizing different hyperelastic models uniformly, particularly within matrix-free iterative solvers. Consequently, we propose a novel matrix assembly strategy that demonstrates high efficiency and accommodates different constitutive models. Our newly introduced approach achieves a significant acceleration when the topology structure remains unchanged, while maintaining construction costs at the same level as standard methods when rebuilding the matrix pattern becomes necessary. This expedited matrix assembly offers the potential for parallelization during the solving stage without requiring a specific parallel implementation of the constitutive model. Furthermore, we replace the CPU-based preconditioner with a newly developed GPU-based implementation during the solving stage. This enhancement notably reduces data transfer between the CPU and GPU, enabling the utilization of a fully GPU-based CG solver. Finally, we assess the effectiveness of our matrix assembly and parallelization strategy through various examples, encompassing different types of

elements and constitutive models. Additionally, our approach has been validated to be compatible with contact problems, which will be discussed in the subsequent chapter.

CHAPTER 4

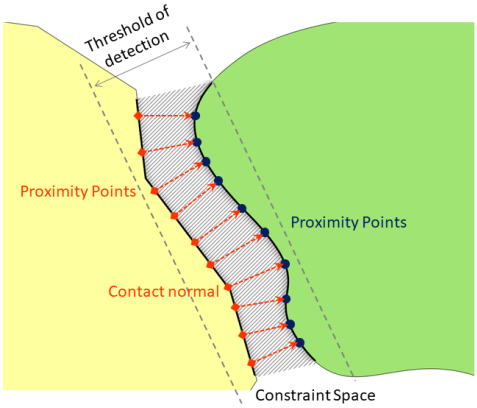
CONTACT PROBLEMS IN INTERACTIVE SIMULATIONS

Interactive simulations received strong interest in surgical simulations and robotic simulations. One of the main requirements is to provide simulations of multi-object systems with complex interactions, such as contact and friction in real-time. This chapter will begin with the introduction to the formulations in constraint-based contact resolution, which is the background of our contributions. Then efficient methods will be proposed to solve the challenging issues of computing the Schur-complement in constrained problem. Finally we evaluate the performance of our method in different contact applications and compare it with typical approaches on CPU and GPU.

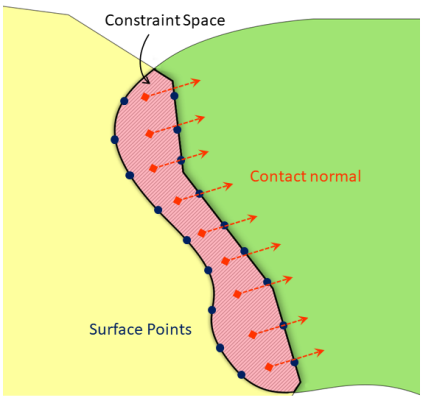
4.1 Formulating contact problem

4.1.1 Contact and friction model

As illustrated in Figure 4.1, a contact constraint involves a contact normal and a pair of proximity points that is referred to the objects in touch. The contact normal defines the direction of a non-interpenetration force to separate the objects. The pair of proximity points $\mathbf{p}$ represent the current state of the surfaces of the objects. In different scenarios, $\mathbf{p}$ can be directly the points on the object surface in a straightforward way, or the representative points using a geometric mapping to transform between $\mathbf{q}$ and $\mathbf{p}$ with the



(a) For typical method of discrete collision detection, a threshold is usually used to test if the proximity points are close enough. The evaluation of potential contact normal depends directly on the surface elements (positions and normals) at the beginning of time steps.



(b) For the image-based method of collision detection, the evaluation of contacts is usually based on penetrated volume. The evaluation of constraint normal depends on the boundary of the interpenetration area, which is computed by the positions of surface elements.

Figure 4.1: Constraint linearization with different types of collision detection: To simplify the solving process, collision detection is performed providing a set of discretized constraints between both objects (red arrows). Each contact constraint involves the proximity points and a direction of contact normal, which is used to apply the force to separate objects. According to different collision detection algorithms, the contact normal is dependent, directly or indirectly, upon the position of proximity points (red and black points on the surface of contacting bodies).

multi-model representation discussed in Section 1.2. We can generally define the relation between the *mechanical DOFs* $\mathbf{q}$ and the proximity positions $\mathbf{p}$ as a geometric mapping function $\mathcal{G}$:

$$\mathbf{p} = \mathcal{G}(\mathbf{q}) \tag{4.1}$$

The contacts between two objects are modeled as constraints, which are discretized and linearized through the collision detection process. As illustrated in Figure 4.1, to prevent interpretations, the distance between two solids 1 and 2 can be formulated as a gap function:

$$\boldsymbol{\delta}_n = \vec{\mathbf{n}}\,[\mathbf{p}_1 - \mathbf{p}_2] = \vec{\mathbf{n}}\,[\mathcal{G}_1(\mathbf{q}) - \mathcal{G}_2(\mathbf{q})] = \mathcal{H}_{n1}(\mathbf{q}) - \mathcal{H}_{n2}(\mathbf{q}) \tag{4.2}$$

where the interpretation δ_n is the distance measurement between the proximity positions $\mathcal{G}(\mathbf{q})$ projected on the contact normal $\vec{\mathbf{n}}$. The contact normal is the direction of a force $\mathbf{f}_n$ that separates the interpenetrating solids. The geometrical mapping function $\mathcal{G}(\mathbf{q})$ describes the mapping from the *mechanical DOFs* space to the proximity space. Integrating

$\vec{\mathbf{n}}$ into the geometric mapping function $\mathscr{G}(\mathbf{q})$ results in a new function $\mathscr{H}_n(\mathbf{q})$. Signorini's law presents the complementarity relationship along the constraint direction $\vec{\mathbf{n}}$ for each potential contact:

$$0 \leq \boldsymbol{\delta}_n \perp \boldsymbol{\lambda}_n \geq 0 \tag{4.3}$$

where the multiplier $\boldsymbol{\lambda}_n$ is the magnitude of the contact force $\vec{\boldsymbol{\lambda}}$ along $\vec{\mathbf{n}}$ as the constraint direction has been normalized.

Equation (4.3) only guarantees to separate the contacting objects. To model the friction response, the frictional constraints should be added along with the contact normal. In a 3D problem, a common frictional model complements each contact normal with two tangential directions $\vec{\mathbf{f}}$. When a contacting is validated ($\boldsymbol{\lambda}_n \geq 0$), following Coulomb's friction law, we have:

$$0 \leq \dot{\boldsymbol{\delta}}_f \perp \mu\boldsymbol{\lambda}_n - \boldsymbol{\lambda}_f \geq 0 \tag{4.4}$$

where $\boldsymbol{\delta}_f$ is the the relative velocity measurement projected on the tangential directions $\vec{\mathbf{f}}$, μ is the coefficient of friction, and $\boldsymbol{\lambda}_f$ is the magnitude of frictional force $\vec{\boldsymbol{\lambda}}_f$ along $\vec{\mathbf{f}}$. The friction model describes two states for the kinematic behavior: the contacting objects are stuck ($\boldsymbol{\delta}_f = 0$) while $\boldsymbol{\lambda}_f \leq \mu\boldsymbol{\lambda}_n$, and are slipping while $\boldsymbol{\lambda}_f$ achieves the maximum value $\mu\boldsymbol{\lambda}_n$:

$$\begin{aligned} \dot{\boldsymbol{\delta}}_f = 0 &\Rightarrow \boldsymbol{\lambda}_f < \mu\boldsymbol{\lambda}_n \quad (stick) \\ \dot{\boldsymbol{\delta}}_f \neq 0 &\Rightarrow \vec{\boldsymbol{\lambda}}_f = \boldsymbol{\lambda}_f \vec{\mathbf{f}} = -\mu\boldsymbol{\lambda}_n \vec{\mathbf{f}} = -\mu\boldsymbol{\lambda}_n \frac{\dot{\boldsymbol{\delta}}_f}{||\dot{\boldsymbol{\delta}}_f||} \quad (slip) \end{aligned} \tag{4.5}$$

In addition, $\boldsymbol{\delta}_f$ has a similar gap function to Equation (4.2):

$$\boldsymbol{\delta}_f = \vec{\mathbf{f}}\,[\mathbf{p}_1 - \mathbf{p}_2] = \vec{\mathbf{f}}\,[\mathscr{G}_1(\mathbf{q}) - \mathscr{G}_2(\mathbf{q})] = \mathscr{H}_{f1}(\mathbf{q}) - \mathscr{H}_{f2}(\mathbf{q}_2) \tag{4.6}$$

The governing equations including the complementarity relationships result in the following non-linear system:

$$\begin{cases} \mathbf{A}_1\Delta\dot{\mathbf{q}}_1 = \mathbf{b}_1 + h\mathbf{c}_1 & \text{(4.7a)} \\ \mathbf{A}_2\Delta\dot{\mathbf{q}}_2 = \mathbf{b}_2 + h\mathbf{c}_2 & \text{(4.7b)} \\ \boldsymbol{\delta}_n = \mathscr{H}_{n1}(\mathbf{q}) - \mathscr{H}_{n2}(\mathbf{q}) & \text{(4.7c)} \\ \boldsymbol{\delta}_f = \mathscr{H}_{f1}(\mathbf{q}) - \mathscr{H}_{f2}(\mathbf{q}) & \text{(4.7d)} \\ 0 \leq \boldsymbol{\delta}_n \perp \boldsymbol{\lambda}_n \geq 0 & \text{(4.7e)} \\ 0 \leq \dot{\boldsymbol{\delta}}_f \perp \mu\boldsymbol{\lambda}_n - \boldsymbol{\lambda}_f \geq 0 & \text{(4.7f)} \end{cases}$$

Since the contact normal constraint along $\vec{\mathbf{n}}$ and the frictional constraint along $\vec{\mathbf{f}}$

have the same mapping function in the gap functions, in practice, the constraints are grouped as constraint sets, where each one of them involves a normal constraint $\vec{\mathbf{n}}$ and two tangential constraints $\vec{\mathbf{f}}$. Consequently, the definitions of functions and vectors can be uniformed: $\mathscr{H}_n(\mathbf{q})$ and $\mathscr{H}_f(\mathbf{q})$ are uniformed as $\mathscr{H}(\mathbf{q})$; $\boldsymbol{\delta}_n$ and $\boldsymbol{\delta}_f$ are uniformed as $\boldsymbol{\delta}$; $\boldsymbol{\lambda}_n$ and $\boldsymbol{\lambda}_f$ are uniformed as $\boldsymbol{\lambda}$.

4.1.2 Constraint linearization

Following the previous section, the gap functions in Equation (4.2) and (4.6) are merged as:

$$\boldsymbol{\delta}(t) = \mathscr{H}_1(\mathbf{q}_1(t)) - \mathscr{H}_2(\mathbf{q}_2(t)) \tag{4.8}$$

With the implicit integration (Equation (3.18)), the mapping functions are linearized with a first-order Taylor expansion:

$$\mathscr{H}(\mathbf{q}^{t+h}) = \mathscr{H}(\mathbf{q}^t + h\dot{\mathbf{q}}^t + h\Delta\dot{\mathbf{q}}) = \mathscr{H}(\mathbf{q}^{\mathrm{e}} + h\Delta\dot{\mathbf{q}}) \approx \mathscr{H}(\mathbf{q}^{\mathrm{e}}) + h\frac{\partial\mathscr{H}(\mathbf{q})}{\partial\mathbf{q}}\Delta\dot{\mathbf{q}} \tag{4.9}$$

where $\mathbf{q}^{\mathrm{e}} = \mathbf{q}^t + h\dot{\mathbf{q}}^t$ represents the mechanical state after an explicit integration. Combining 4.9 and 4.8 we have the integration at the end of time step for the violation:

$$\begin{aligned}
\boldsymbol{\delta}^{t+h} &= \mathscr{H}_1(\mathbf{q}_1^{t+h}) - \mathscr{H}_2(\mathbf{q}_2^{t+h}) \\
&= \mathscr{H}_1(\mathbf{q}_1^{e}) - \mathscr{H}_2(\mathbf{q}_2^{e}) + h(\frac{\partial_1\mathscr{H}_1(\mathbf{q}_1)}{\partial\mathbf{q}_1}\Delta\dot{\mathbf{q}}_1 - \frac{\partial_2\mathscr{H}_2(\mathbf{q}_2)}{\partial\mathbf{q}_2}\Delta\dot{\mathbf{q}}_2) \\
&= \boldsymbol{\delta}^{e} + h(\frac{\partial_1\mathscr{H}_1(\mathbf{q}_1)}{\partial\mathbf{q}_1}\Delta\dot{\mathbf{q}}_1 - \frac{\partial_2\mathscr{H}_2(\mathbf{q}_2)}{\partial\mathbf{q}_2}\Delta\dot{\mathbf{q}}_2)
\end{aligned} \tag{4.10}$$

where $\boldsymbol{\delta}^{e} = \mathscr{H}_1(\mathbf{q}_1^{e}) - \mathscr{H}_2(\mathbf{q}_2^{e})$ represents the interpenetration after an explicit integration.

Once the potential contact information is available with the result of the collision detection procedure, all the constraint equations are then evaluated along with the collision information that is assumed constant for the rest of the time step (see Figure 4.2). This leads to an important simplification: $\mathbf{J} \approx \frac{\partial\mathscr{H}(\mathbf{q})}{\partial\mathbf{q}}$ at t the beginning of each time step , known as the *constraint Jacobian*, can be defined, providing the constraint directions (blue and orange arrows in Figure 4.2). The dimension of $\mathbf{J}$ is $c \times n$, where c is the number of discretized constraints, and n is the number of *mechanical DOFs*. With the simplification, the violation of the constraint at the end of the step can be rewritten as:

$$\boldsymbol{\delta}^{t+h} = \boldsymbol{\delta}^{e} + h\mathbf{J}_1\Delta\dot{\mathbf{q}}_1 - h\mathbf{J}_2\Delta\dot{\mathbf{q}}_2 \tag{4.11}$$

On the other hand, the *constraint Jacobian* applies the constraint forces into the mechan-

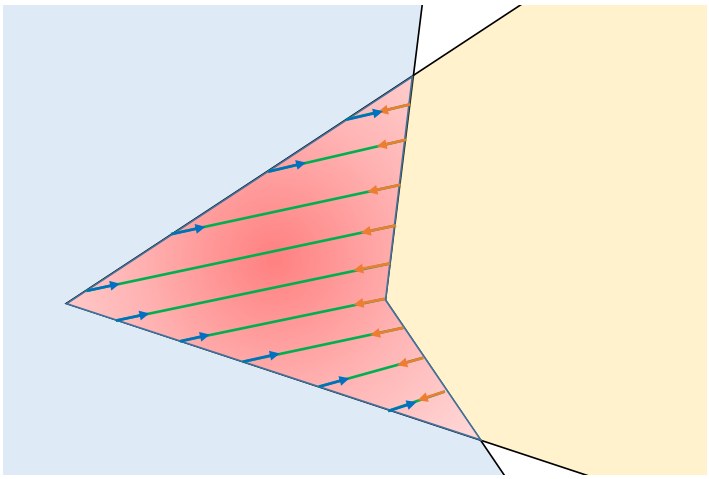


Figure 4.2: In practice, the evaluation and the linearization of the constraints equations are difficult. To simplify the solving process, collision detection is performed providing a set of discretized constraints between both objects (gree lines). The number of discretized constraints usually depends on the resolution of the collision mesh and/or the collision detection method itself (for instance, by filtering constraints afterward).

ical motion space:

$$\begin{aligned} \mathbf{c}_1 &= {\mathbf{J}_1}^{\mathrm{T}} \boldsymbol{\lambda} \\ \mathbf{c}_2 &= -{\mathbf{J}_2}^{\mathrm{T}} \boldsymbol{\lambda} \end{aligned} \tag{4.12}$$

where the forces are applied for the two objects in opposite directions ($\vec{\mathbf{n}}_1 = -\vec{\mathbf{n}}_2$, $\vec{\mathbf{f}}_1 = -\vec{\mathbf{f}}_2$).

With the constraint linearization, the Karush-Kuhn-Tucker (KKT) system in Equation (4.7) is assembled as follows:

$$\begin{cases} \mathbf{A}_1 \Delta\dot{\mathbf{q}}_1 - h{\mathbf{J}_1}^{\mathrm{T}} \boldsymbol{\lambda} = \mathbf{b}_1 & \text{(4.13a)} \\ \mathbf{A}_2 \Delta\dot{\mathbf{q}}_2 + h{\mathbf{J}_2}^{\mathrm{T}} \boldsymbol{\lambda} = \mathbf{b}_2 & \text{(4.13b)} \\ h\mathbf{J}_1 \Delta\dot{\mathbf{q}}_1 - h\mathbf{J}_2 \Delta\dot{\mathbf{q}}_2 + \boldsymbol{\delta}^e = \boldsymbol{\delta}^{t+h} & \text{(4.13c)} \end{cases}$$

In addition, $\boldsymbol{\delta}^{t+h}$ and $\boldsymbol{\lambda}$ at the end of time steps should satisfy the complementarity according to Signorini's law (Equation (4.3)) and Coulomb's law (Equation (4.4)).

4.1.3 Constraint resolution

By eliminating the unknowns $\mathbf{x}_1$ and $\mathbf{x}_2$ in Equation (4.7c), we have:

$$\boldsymbol{\delta}^{t+h} = \boldsymbol{\delta}^e + h[\mathbf{J}_1 \underbrace{\mathbf{A}_1^{-1}\mathbf{b}_1}_{\mathbf{x}_1^{\text{free}}} - \mathbf{J}_2 \underbrace{\mathbf{A}_2^{-1}\mathbf{b}_2}_{\mathbf{x}_2^{\text{free}}}] + h^2 \underbrace{[\mathbf{J}_1\mathbf{A}_1^{-1}{\mathbf{J}_1}^{\mathrm{T}} + \mathbf{J}_2\mathbf{A}_2^{-1}{\mathbf{J}_2}^{\mathrm{T}}]}_{\mathbf{W}} \boldsymbol{\lambda} \tag{4.14}$$

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

$$\mathbf{x}^{\text{free}} = \mathbf{A}^{-1}\mathbf{b} \tag{4.15}$$

With implicit integration we may note that:

$$\begin{aligned}
\boldsymbol{\delta}^e + h[\mathbf{J}_1\mathbf{x}_1^{\text{free}} - \mathbf{J}_2\mathbf{x}_2^{\text{free}}] &= \mathscr{H}_1(\mathbf{q}_1^e) - \mathscr{H}_2(\mathbf{q}_2^e) + h\left[\mathbf{J}_1\mathbf{x}_1^{\text{free}} - \mathbf{J}_2\mathbf{x}_2^{\text{free}}\right] \\
&= \left[\mathscr{H}_1(\mathbf{q}_1^e) + h\mathbf{J}_1\mathbf{x}_1^{\text{free}}\right] - \left[\mathscr{H}_2(\mathbf{q}_2^e) + h\mathbf{J}_2\mathbf{x}_2^{\text{free}}\right] \\
&= \mathscr{H}_1(\mathbf{q}_1^t + h\dot{\mathbf{q}}_1^t + h\mathbf{x}_1^{\text{free}}) - \mathscr{H}_2(\mathbf{q}_2^t + h\dot{\mathbf{q}}_2^t + h\mathbf{x}_2^{\text{free}}) \\
&= \mathscr{H}_1(\mathbf{q}_1^{\text{free}}) - \mathscr{H}_2(\mathbf{q}_2^{\text{free}}) \\
&= \boldsymbol{\delta}^{\text{free}}
\end{aligned} \tag{4.16}$$

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

$$\boldsymbol{\delta}^{t+h} = \boldsymbol{\delta}^{\text{free}} + h^2\mathbf{W}\boldsymbol{\lambda} \tag{4.17}$$

The unknown $\boldsymbol{\lambda}$ is solved by a projected Gauss-Seidel algorithm Duriez et al. (2006) during the successive iterations (i):

$$\delta_\alpha - h^2\mathbf{W}_{\alpha\alpha}\boldsymbol{\lambda}_\alpha^{(i)} = \sum_{\beta=1}^{\alpha-1} h^2\mathbf{W}_{\alpha\beta}\boldsymbol{\lambda}_\beta^{(i)} + \sum_{\beta=\alpha+1}^{c} h^2\mathbf{W}_{\alpha\beta}\boldsymbol{\lambda}_\beta^{(i-1)} + \delta_\alpha^{\text{free}} \tag{4.18}$$

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

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

$$\begin{aligned}
\mathbf{x}_1^{t+h} &= \mathbf{x}_1^{\text{free}} + h\mathbf{A}_1^{-1}{\mathbf{J}_1}^{\mathrm{T}}\boldsymbol{\lambda} \\
\mathbf{x}_2^{t+h} &= \mathbf{x}_2^{\text{free}} - h\mathbf{A}_2^{-1}{\mathbf{J}_2}^{\mathrm{T}}\boldsymbol{\lambda}
\end{aligned} \tag{4.19}$$

Finally, we summarize the time step integration with constraint resolution in Algorithm 3.

Algorithme 3 : Standard simulation loop scheme with constraint-based contacts

while *simulation* **do**
  collision_detection($\mathbf{q}^t$) ;
  constraint_linearization($\mathbf{p}^t$) ;
  foreach object **do**
    Assemble $\mathbf{A}$, $\mathbf{b}$, $\mathbf{J}$;
    $\Delta\dot{\mathbf{q}}^{\text{free}} = \mathbf{A}^{-1}\mathbf{b}$;
    $\mathbf{q}^{\text{free}} = \mathbf{q}^h + h(\dot{\mathbf{q}}^h + \Delta\dot{\mathbf{q}}^{\text{free}})$;
  end
  Compute $\boldsymbol{\delta}^{\text{free}}$ according to $\mathbf{q}^{\text{free}}$;
  $\mathbf{W} = \sum \mathbf{J}\mathbf{A}^{-1}\mathbf{J}^{\mathrm{T}}$;
  foreach $i \in$ PGS_iterations **do**
    foreach $j \in$ constraint_groups **do**
      $\boldsymbol{\delta}^c = \boldsymbol{\delta}^{\text{free}} + \mathbf{W}\boldsymbol{\lambda}^i$;
      $\boldsymbol{\lambda}^i_j = solve(\boldsymbol{\lambda}^i, \boldsymbol{\delta}^c, \mathbf{W})$;
    end
    $\epsilon = \frac{|\boldsymbol{\lambda}^i - \boldsymbol{\lambda}^{i-1}|}{|\boldsymbol{\lambda}^i|}$;
    if $\epsilon \leq$ PGS_error **then**
      break ;
    end
  end
  foreach object **do**
    $\Delta\dot{\mathbf{q}}^{t+h} = \Delta\dot{\mathbf{q}}^{\text{free}} + h\mathbf{A}^{-1}\mathbf{J}^{\mathrm{T}}\boldsymbol{\lambda}$;
    $\dot{\mathbf{q}}^{t+h} = \dot{\mathbf{q}}^h + \Delta\dot{\mathbf{q}}^{t+h}$;
    $\mathbf{q}^{t+h} = \mathbf{q}^h + h\dot{\mathbf{q}}^{t+h}$;
  end
end

4.2 Computing of compliance in contact problem

4.2.1 Compliance assembly

A primary challenge in real-time simulation is computing the Schur-complement in Equation (4.14). This involves the large system matrix $\mathbf{A}$ with a dimension size corresponding to the number of *mechanical DOFs*. Inverting such large a system is highly expensive, especially with multiple right-hand sides (RHS) in the contact Jacobian $\mathbf{J}$ with a size corresponding to the number of constraints. Processing an exact factorization for $\mathbf{A}$ in each time step can be used in small scale problems but becomes prohibitive when dealing with

detailed soft bodies. To address the problem, many works are dedicated to find a good approximation of the factorized system, such as incomplete factorization (Schenk and Gärtner (2006) implemented in *Pardiso solver project*), updating Cholesky factor (Herholz and Alexa (2018), Herholz and Sorkine-Hornung (2020)) and asynchronous preconditioning strategy (Courtecuisse et al. (2010a)). In the current manuscript, our main contribution relies on an hypothesis that the solver is able to obtain a good approximation of factorization. Our work is based on the asynchronous preconditioning strategy that releases the computing expense in the main simulation loop. The strategy was discussed in 3.3.1. In *free motion*, the resolution in Equation (4.15) is solved with a preconditioned CG algorithm, as discussed in Section 3.3. In *constraint resolution*, Courtecuisse et al. (2014) presents a precondition-based approach for contact problems. The asynchronous preconditioner $\mathbf{P}$ is reused as a close approximation of the inverse of the factorization of the current system matrix $\mathbf{A}$. As a result, the compliance matrix is approximately built as:

$$\mathbf{W} = \sum \mathbf{J}\mathbf{A}^{-1}\mathbf{J}^{\mathrm{T}} \approx \sum \mathbf{J}\mathbf{P}\mathbf{J}^{\mathrm{T}} = \sum \mathbf{J}(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1}\mathbf{J}^{\mathrm{T}} \tag{4.20}$$

with the summation of contribution of all the contacting objects.

Algorithme 4 : Approximate computation of the Schur-complement with the system factorized in asynchronous thread

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

The contribution accumulation of each object on the compliance is processed in column independently with Algorithm 4: the first step is the resolution of multiple STS, which is usually the most expensive task and tends to be very costly while processed sequentially on CPU. Courtecuisse et al. (2014) proposes an efficient GPU-based solution for multiple STS using a two-level parallelization: each right-hand side in $\mathbf{J}^{\mathrm{T}}$ is computed in parallel multiprocessors. Since the resolution of each triangular system involves numbers of dependencies, the left-hand side $\mathbf{L}$ is fully processed and the result $\mathbf{S}$ is stored in a dense matrix. The second step consists of matrix-matrix multiplications and can be efficiently processed on GPU. Despite the fact that the method already provides a significant speedup compared to the sequential computation on CPU, the building of contact compliance matrix still remains the most expensive process, taking a ratio of more than 70% in time integration. Accelerating this process is an important issue that will be addressed in the following sections 4.2.3 and 4.2.4. It has been proved in Courtecuisse et al. (2014) that the asynchronous preconditioning strategy provides a good approximation to the actual $\mathbf{W}$ (assembled with $\mathbf{A}^{-1}$), allowing to efficiently couple the contact forces in constraint

resolution.

4.2.2 Complementarity problem resolution

An iterative method like Equation (4.18) can be performed either with explicitly assembled $\mathbf{W}$, or with an unbuilt form since processing each iteration only requires an operation of matrix-vector multiplication:

$$\mathbf{z} = \mathbf{W}\mathbf{y} = \sum(\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^{\mathrm{T}})\mathbf{y} = \sum\mathbf{J}\mathbf{A}^{-1}(\mathbf{J}^{\mathrm{T}}\mathbf{y}) \tag{4.21}$$

where $\mathbf{J}^{\mathrm{T}}\mathbf{y}$ leads to a vector, avoiding to apply the multiple right hand sides $(\mathbf{A}^{-1}\mathbf{J}^{\mathrm{T}})$. However, the possibility to not build explicitly $\mathbf{W}$ is popular as long as $\mathbf{A}^{-1}$ is sparse and easy to be computed. This is the case for instance for rigid objects with a diagonal matrix, or beam elements with a block-tridiagonal matrix where Thomas algorithm used in Xu and Liu (2018) can be used to invert the system. In these cases, the unbuilt version is known to be faster. However, this assumption does not apply to FE models with large unstructured matrices.

Following the context in Section 4.2.1, as long as an approximation of factorization can be fast obtained, the matrix-multiplication in unbuilt scheme (Equation (4.21)) actually requires solving a $\mathbf{LDL}^{\mathrm{T}}$ system:

$$\mathbf{z} = \sum\mathbf{J}\mathbf{A}^{-1}(\mathbf{J}^{\mathrm{T}}\mathbf{y}) \approx \sum\mathbf{J}(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1}(\mathbf{J}^{\mathrm{T}}\mathbf{y}) \tag{4.22}$$

Algorithm 5 implements the resolution of the $\mathbf{LDL}^{\mathrm{T}}$ system in Equation (4.22).

Algorithme 5 : Unbuilt scheme: implementation of the STS resolution (Equation 4.22) in iterations of relaxation methods. $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are temporary vectors

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

This leads to extra cost in each iteration. When dealing with large scale problems, the $\mathbf{LDL}^{\mathrm{T}}$ resolution becomes costly. Such an addition operation in iterations will result in enormous extra computation cost. In Section 4.2.5.1 we will compare the cost of unbuilt scheme and assembling $\mathbf{W}$ in different cases.

4.2.3 Isolating mechanical DOFs: Reformulating the Schur-complement

4.2.3.1 Exploit the sparsity of constraint Jacobian matrix

The *constraint Jacobian* matrix $\mathbf{J}$ describes how the contact constraints are applied to mechanical degrees of freedom (*DOFs*). Since the contacts are often limited in local areas, the size of constraint dimension $\boldsymbol{c}$ is usually far more smaller than the dimension of *mechanical DOFs* $\boldsymbol{n}$. Coupled with the fact that each constraint is linked to limited *mechanical DOFs*, $\mathbf{J}$ is very sparse in many cases. Based on this observation, we propose eliminating empty columns in $\mathbf{J}$ and formulating a "compressed" matrix $\hat{\mathbf{H}}$. The relation between the two matrices can be actually expressed by a matrix-matrix multiplication (see also Figure 4.3):

$$\mathbf{J} = \hat{\mathbf{H}}\bar{\mathbf{I}} \tag{4.23}$$

where $\bar{\mathbf{I}}$ is a "*partial identity*" matrix that is formulated with the indices of *non-zero* columns in $\mathbf{J}$. By formulating $\bar{\mathbf{I}}$ we actually isolate the indices information of *mechanical DOFs* that

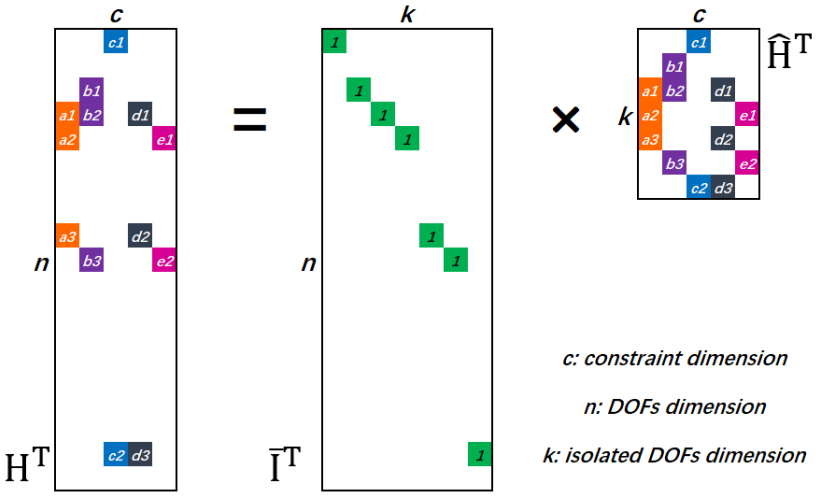


Figure 4.3: The *constraint Jacobian* $\mathbf{J}$ is usually very sparse and contains many empty columns. By eliminating these empty columns, we formulate a "compressed" matrix $\hat{\mathbf{H}}$ and a "*partial identity*" matrix $\bar{\mathbf{I}}$ that contains one element in each row that corresponds to a *non-zero* column in $\mathbf{J}$. The relation between matrices can be expressed by a matrix-matrix multiplication: $\mathbf{J} = \hat{\mathbf{H}}\bar{\mathbf{I}}$ (transposed format illustrated in the figure)

receive contributions from the constraints from $\mathbf{J}$. This generates a new dimension "*isolated DOFs*" (also abbreviated to "*isodof*"). The matrix $\bar{\mathbf{I}}$ is also called as "*isodof Jacobian*". Compared to the constraint dimension $\boldsymbol{c}$, the *isodof* dimension $\boldsymbol{k}$ may be larger or smaller. Now, with Equation (4.23), computing the Schur-complement in Equation (4.20) is refor-

mulated as:

$$\mathbf{W} = \sum \hat{\mathbf{H}} \underbrace{\bar{\mathbf{I}} \mathbf{A}^{-1} \bar{\mathbf{I}}^{\mathrm{T}}}_{\bar{\mathbf{W}}} \hat{\mathbf{H}}^{\mathrm{T}} \tag{4.24}$$

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

Following Equation (4.20), we propose to compute with the asynchronous preconditioner ($\mathbf{A} \approx \mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}$) in Algorithm 6. Step 1 is processed by analyzing the sparse pattern of $\mathbf{J}$; Step 2 involves solving multiple STS; Step 3 and Step 4 consist of matrix-matrix multiplications that can be efficiently computed on GPU. We underline that Algorithm 6 computes the same result as in Algorithm 4 (i.e., the resolution proposed in Courtecuisse et al. (2014)). The only approximation comes from using a delayed system that is factorized in an asynchronous thread.

Algorithme 6 : Approximate computation of the Schur-complement with the system factorized in asynchronous thread

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

4.2.3.2 STS resolution strategy

The step 2 of Algorithm 6 remains a difficult task to be parallelized on the GPU due to the data dependencies in triangular systems. However, the *isodof scheme* leads to a special resolution. Each right-hand side is no more the combination of various values (in $\mathbf{J}^{\mathrm{T}}$) but only contains one element with value "1" on a specific column (in $\bar{\mathbf{I}}^{\mathrm{T}}$). An important consequence is related to the fact that, for each column of $\bar{\mathbf{I}}^{\mathrm{T}}$, only a subset value needs to be computed, leading to a sparse resolution of the triangular system (see Figure 4.4). Therefore the density of $\bar{\mathbf{S}}$ is significantly reduced compared to $\mathbf{S}$ (i.e., dealing with $\mathbf{J}^{\mathrm{T}}$). In addition, the dependencies can formally be expressed with an elimination tree. Given that the matrix pattern of $\mathbf{L}$ only depends on the mesh topology, the elimination tree can be pre-computed, and the sparse matrix storage of the result is predictable for every column index, as long as the topology is not changed.

While processing the STS resolution, it can be either processed in the "row-major" or the "column-major" (see Figure 4.5). When assuming the solutions as dense vectors (as in Courtecuisse et al. (2014)), processing the "row-major" is more efficient because it does not cause data writing conflict in parallel resolution. However, as the structure of $\bar{\mathbf{S}}$ is very sparse, it accumulates zero contributions, causing a large amount of unnecessary computation cost. Although the "column-major" requires pre-accumulating the data on

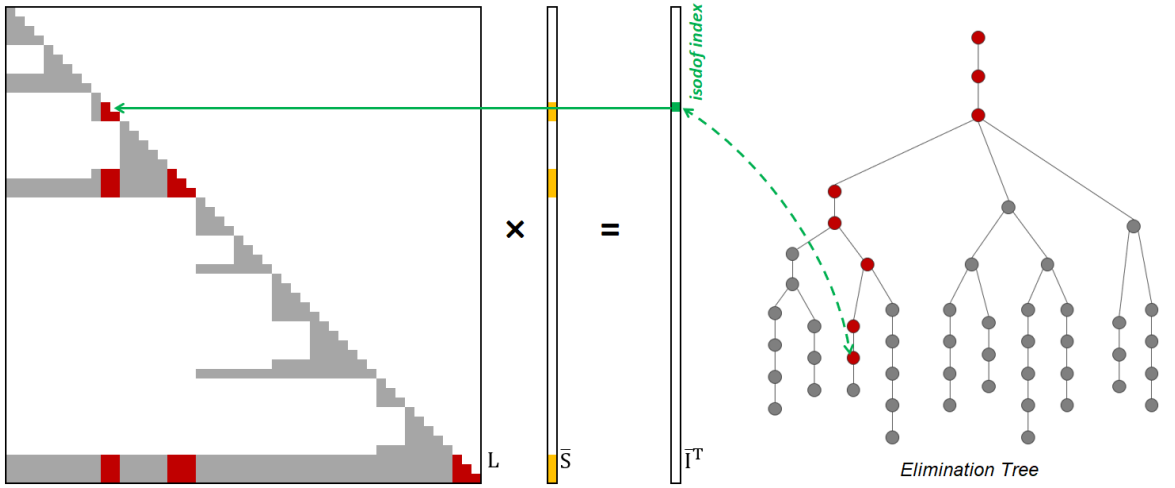


Figure 4.4: Resolution of one RHS in $\mathbf{L}\bar{\mathbf{S}} = \bar{\mathbf{I}}^{\mathrm{T}}$ (the elimination tree (top-right) helps visualize the structure of dependency in **L**): Each right-hand side contains one element with value "1" on a column index i, locating on a branch on the elimination tree. During the resolution, only this branch with index i and its parent branches need to be processed (red nodes on the elimination tree). Reflected on the matrix pattern, only the red elements in **L** need to be processed. This sparse resolution is very efficient compared to process the full matrix **L**.

the positions where the result is not yet solved (see below), it can naturally avoid this unnecessary accumulation.

4.2.3.3 GPU-based implementation

To implement Algorithm 6, Step 1 requires to build $\bar{\mathbf{I}}$ and $\hat{\mathbf{H}}$. According to the illustration in Figure 4.3, $\bar{\mathbf{I}}$ can be stored in a vector that contains the *non-zero* columns in **J**. As a "compressed reformulation", $\hat{\mathbf{H}}$ have the same data sequence of **J**. This means that, when stored in Compressed Row Sparse (CSR) format, they have the same vectors of *row index* and *values*, while the vector of *column index* of $\hat{\mathbf{H}}$ is built as "compressed indices".

The GPU-based implementation of STS resolution in Step 2 is inspired from the block-row parallelization strategy in Courtecuisse et al. (2014). Each right-hand side of $\bar{\mathbf{S}}$ is assigned to an independent multiprocessor since the multiple RHS are independent of each other (see Algorithm 7).

To perform the sparse resolution in Figure 4.4 for each right-hand side, we pre-compute offline the pattern to be processed for every index. Each pattern is composed of sub-domains, with their indices given by the reordering algorithm. As illustrated in Figure 4.6, within each sub-domain, we process t columns simultaneously by using a group of $t \times t$ threads. For each t columns, the block diagonal (*diag*) is firstly processed as a dense problem; then, the off-diagonal data is accumulated into the result. The parallel accumu-

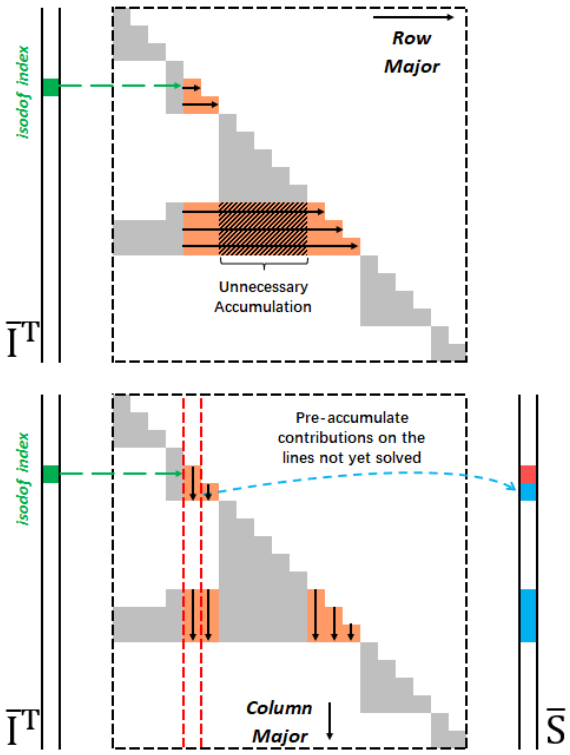


Figure 4.5: The STS can either be solved in the "row-major" (Top) or in the "column-major" (Bottom). In the "row-major", using the CSR format requires processing data in $\mathbf{L}$ continuously on each row, leading to many unnecessary accumulations to the result with zero contributions. Using the "column-major" scheme with CSC format can naturally address this problem. When dealing with a column i, the result on the line i is fully solved (red), but the rest lines remain unsolved (blue) and require to pre-accumulate the contributions on the lines: $res[j] = res[j] - res[i] * \mathbf{L}_{i,j}$.

lations may cause data writing conflicts, which can be handled with the *atomic* function.

As $\bar{\mathbf{S}}$ and $\hat{\mathbf{H}}$ are stored in the sparse format, once the STS resolution is processed, Step 3 and 4 in Algorithm 6 can be implemented with the following operations:

$$\mathbf{X}_1 = \mathbf{D}^{-1}\bar{\mathbf{S}} \quad (Diag - Sparse) \tag{4.25a}$$

$$\bar{\mathbf{W}} = \bar{\mathbf{S}}^T\mathbf{X}_1 \quad (Sparse - Sparse) \tag{4.25b}$$

$$\mathbf{X}_2 = \bar{\mathbf{W}}\hat{\mathbf{H}}^T \quad (Sparse - Dense) \tag{4.25c}$$

$$\mathbf{W} = \hat{\mathbf{H}}\mathbf{X}_2 \quad (Sparse - Dense) \tag{4.25d}$$

where Operation (4.25a) is a diagonal matrix-sparse matrix multiplication, resulting in a temporary matrix $\mathbf{X}_1$ with sparse format. Operation (4.25b) is a sparse matrix-sparse matrix multiplication, where the sparse structure of $\bar{\mathbf{S}}^T$ and $\mathbf{X}_1$ can be pre-computed. Operations(4.25c), and (4.25d) are sparse matrix-dense matrix multiplications (***SpMM***) with normal or transposed format, resulting in matrices with dense format. As we process the

Algorithme 7 : Algorithm to address Sparse Triangular System with multiple right-hand sides of *isolated DOFs* using column-major

Result : columns *res* in $\bar{\mathbf{S}}$ solved by multiprocessors in parallel

Initialization: compute sub-domains to be processed and their indices in offline : $subDomain, startInd, endInd$;
$i = 0$;
$bx = isodof$; // First sub-domain begins with *isodof index*
$end = endIndex[subDomain[i]]$;
while $i < subD.size$ **do**
  while $bx < end$ **do**
    copy_into_shared_memory(diag) ;
    local_synchronization ;
    solve_bloc_diagonal(diag,res) ; // see Courtecuisse et al. (2014)
    local_synchronization ;
    pre_accumulate_contributions(res) ; // Figure 4.5
    local_synchronization ;
    $bx = bx + t$; // next t columns
  end
  $i = i + 1$;
  $bx = startInd[subDomain[i]]$;
  $end = endInd[subDomain[i]]$; // next sub-domain
end

sparse computation, an important consequence is that the operations in Step 2 and 3 in Algorithm 6 are independent of the *mechanical DOF* dimension $\boldsymbol{n}$. As a result, the *isodof* method can be very efficient even with highly detailed mesh.

4.2.4 Reuse of solutions in consecutive time steps

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

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

$$\begin{aligned} \bar{\mathbf{S}}_t &= \mathbf{L}^{-1}\bar{\mathbf{I}}_t{}^{\mathrm{T}} \\ \bar{\mathbf{S}}_{t+i} &= \mathbf{L}^{-1}\bar{\mathbf{I}}_{t+i}{}^{\mathrm{T}} \end{aligned} \tag{4.26}$$

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

Moreover, the *mechanical DOFs* that are impacted by the contact constraints directly depend on the local mesh area where contact occurs. Figure 4.7 reveals that, in real-time

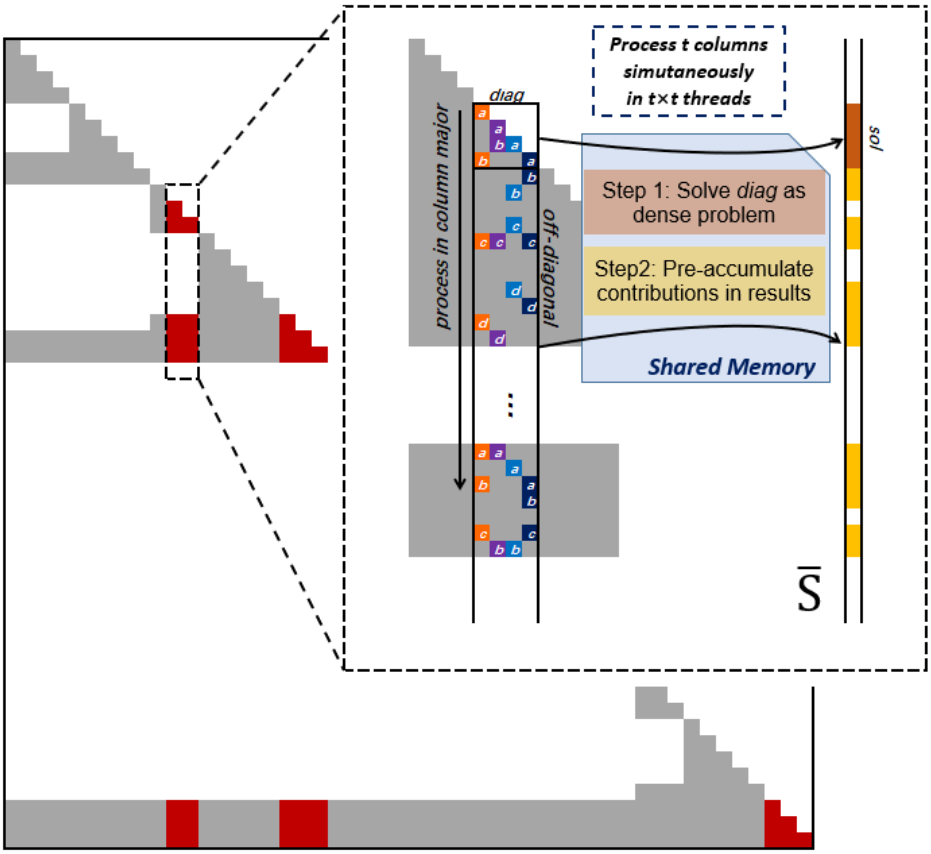


Figure 4.6: The STS with *isodof Jacobian* is solved in column-major. Each right-hand side is assigned to an independent multiprocessor, and for each one, $t \times t$ threads (represented by different colors) are used to process the resolution simultaneously. The *off-diagonal* contributions are pre-accumulated to the results in parallel threads.

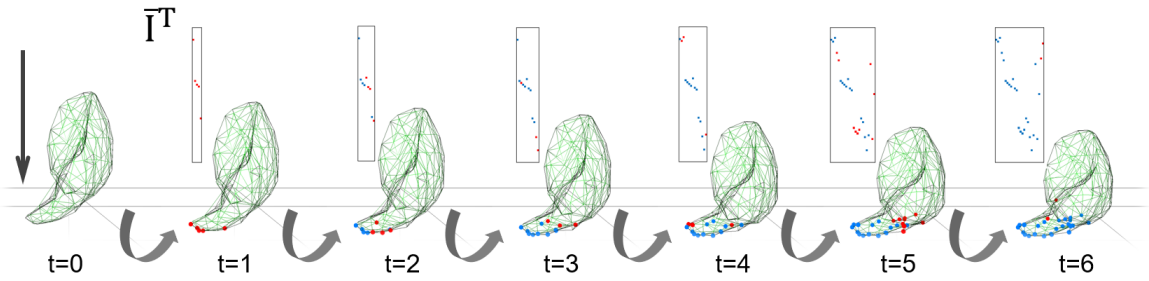


Figure 4.7: The evolution of contact space in a contact simulation between a deformable liver mesh and a rigid plane: The points on the mesh show the *isolated DOFs* that appear in the previous time steps (blue) and the new *isodofs* (red). It is revealed that the consecutive time steps usually shares same *isodofs*, which is relected on the *isodof Jacobian* matrix $\bar{\mathbf{I}}$.

simulations, consecutive time steps usually share a part of contact area, so as the *isodofs* impacted. Reflected on the matrices, the *isodof Jacobian* in consecutive time steps ($\bar{\mathbf{I}}_t{}^{\mathrm{T}}$ and $\bar{\mathbf{I}}_{t+i}{}^{\mathrm{T}}$) share a part of same right-hand sides. Consequently, while $\mathbf{L}$ is not updated, the

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

$$\bar{\mathbf{S}}_{\text{new}} = \mathbf{L}^{-1}\bar{\mathbf{I}}_{\text{new}}{}^{\mathrm{T}} \tag{4.27}$$

where $\bar{\mathbf{I}}_{\text{new}}$ consists of new *isodofs* that emerges in $\bar{\mathbf{I}}_{t+1}$. Compared to the standard *isodof scheme* presented in Section 4.2.3, we have a smaller dimension to deal with, implying a further speedup.

To benefit speedup from the *reuse scheme*, we propose a hybrid implementation illustrated in Figure 4.8:

1. **standard scheme** While a new factorization is done in the asynchronous thread, the solver follows the operations in Algorithm 6.

2. **reuse scheme** While the factorized system is not updated, the solver performs a "reuse scheme" (see the implementation below).

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

$$\bar{\mathbf{I}} = \mathbf{P}_\pi \begin{bmatrix} \bar{\mathbf{I}}_{\text{old}} \\ \bar{\mathbf{I}}_{\text{new}} \end{bmatrix} \tag{4.28}$$

Following Algorithm 6 and Equation (4.28), $\bar{\mathbf{W}}$ in the current time step is built as:

$$\begin{aligned}
\bar{\mathbf{W}} &\approx \bar{\mathbf{I}}(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1}\bar{\mathbf{I}}^{\mathrm{T}} \\
&= \mathbf{P}_\pi \begin{bmatrix} \bar{\mathbf{I}}_{\text{old}} \\ \bar{\mathbf{I}}_{\text{new}} \end{bmatrix} (\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}})^{-1} \begin{bmatrix} \bar{\mathbf{I}}_{\text{old}}{}^{\mathrm{T}} & \bar{\mathbf{I}}_{\text{new}}{}^{\mathrm{T}} \end{bmatrix} \mathbf{P}_\pi{}^{\mathrm{T}} \\
&= \mathbf{P}_\pi \begin{bmatrix} \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}} & \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \\ \bar{\mathbf{S}}_{\text{new}}{}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}} & \bar{\mathbf{S}}_{\text{new}}{}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \end{bmatrix} \mathbf{P}_\pi{}^{\mathrm{T}} \\
&= \mathbf{P}_\pi \underbrace{\begin{bmatrix} \bar{\mathbf{W}}_{\text{old}} & \bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \\ (\bar{\mathbf{S}}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}})^{\mathrm{T}} & \bar{\mathbf{S}}_{\text{new}}{}^{\mathrm{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \end{bmatrix}}_{\bar{\mathbf{W}}_{\text{extend}}} \mathbf{P}_\pi{}^{\mathrm{T}}
\end{aligned} \tag{4.29}$$

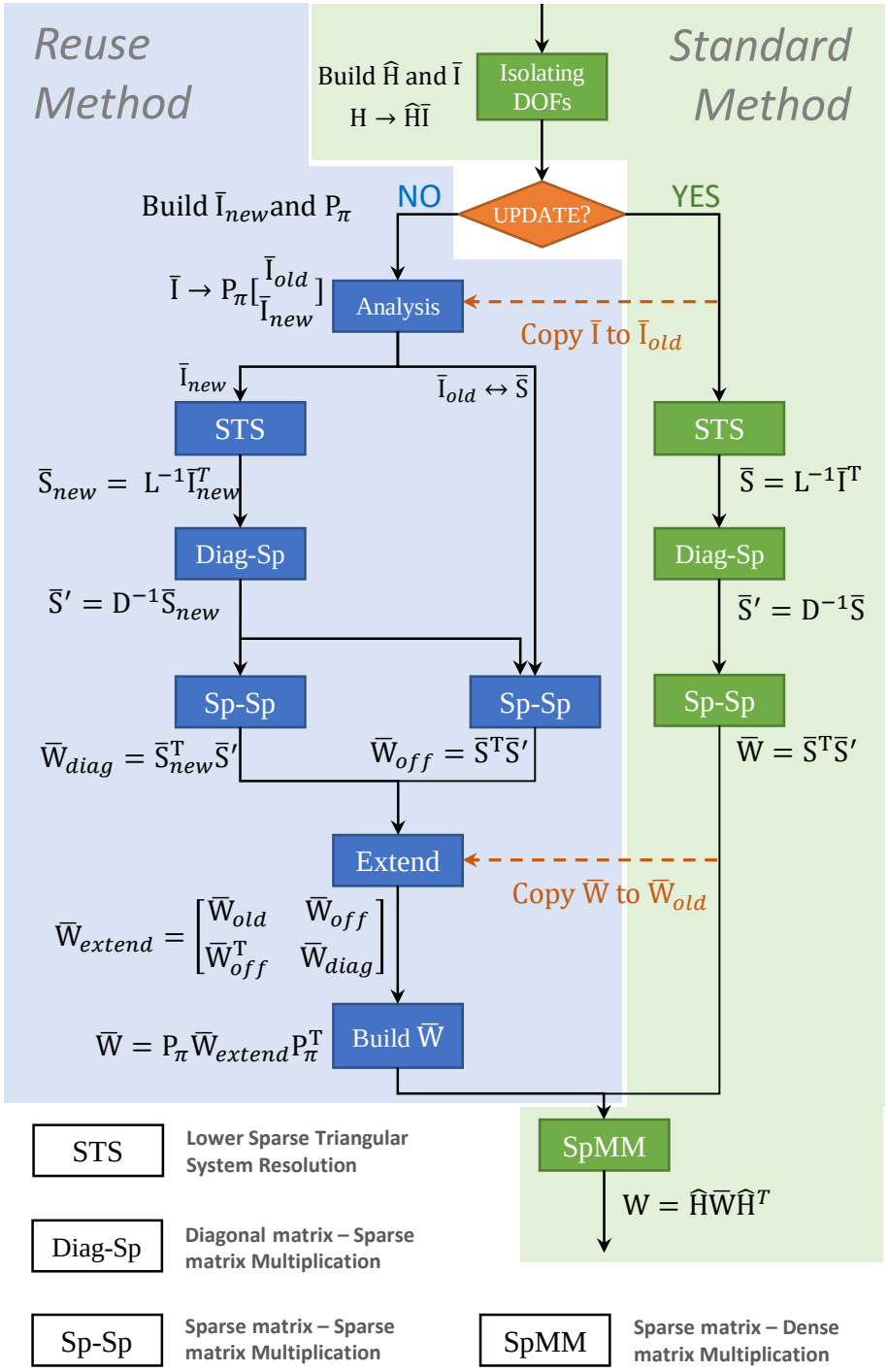


Figure 4.8: Scheme of standard/reuse scheme of *isodof* method: While a new factorization is done in the asynchronous thread, the solver performs a "standard scheme", storing the *isodof Jacobian* $\bar{\mathbf{I}}_{\text{old}}$, the STS solution $\bar{\mathbf{S}}$, and the *isodof delasus* $\bar{\mathbf{W}}_{\text{old}}$ in GPU memory. While the solver keeps using the same factorized system of the previous time steps, it performs a "reuse scheme".

with $\bar{\mathbf{S}} = \mathbf{L}^{-1}\bar{\mathbf{I}}_{\text{old}}$ and $\bar{\mathbf{S}}_{\text{new}} = \mathbf{L}^{-1}\bar{\mathbf{I}}_{\text{new}}$. The former has been computed in previous time steps and the latter is to be solved in the current time step. To finally build the current $\bar{\mathbf{W}}$, we have an "extended" matrix $\bar{\mathbf{W}}_{\text{extend}}$ to build, where the *diagonal* and the *off-diagonal* parts are formulated as:

$$\bar{\mathbf{W}}_{\text{diag}} = \bar{\mathbf{S}}_{\text{new}}{}^{\text{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \tag{4.30a}$$

$$\bar{\mathbf{W}}_{\text{off}} = \bar{\mathbf{S}}^{\text{T}}\mathbf{D}^{-1}\bar{\mathbf{S}}_{\text{new}} \tag{4.30b}$$

The detail implementation of the "reuse scheme" is illustrated in Figure 4.8. Although the number of operations is increased compared to the "standard scheme", the operations (i.e., STS resolution, ***SpMM***...) are very efficient since they usually have much smaller *isodof* dimension. Therefore, based on the *isodof* method presented in Section 4.2.3, the "reuse scheme" benefit a further speedup on performance from reducing the *isodof* dimension size. We still underline that the "reuse scheme" computes the same resolution as in Algorithm 4 and Algorithm 6.

4.2.5 Evaluation of computation cost in the Schur-complement

In this section we evaluate the computation cost of the *isodof* method presented in Section 4.2.3 as well as the *reuse isodof* method in Section 4.2.4. The simulation tests are conducted in the open-source SOFA framework with a CPU AMD@ Ryzen 9 5950X 16-Core at 3.40GHz with 32GB RAM, and a GPU GeForce RTX 3080 10GB.

The deformable meshes are modeled with the co-rotational formulation (although it should be compatible with other materials as proposed in Courtecuisse et al. (2015) for hyperelastic materials). Our methods are dedicated to assembling the *Delasus operator* and are compatible with various methods in the other steps. The free motion is solved with a preconditioned Conjugate Gradient (**PCG**), and the constraint resolution uses a projected Gauss-Seidel (**PGS**).

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

By simulating the simple collision between a deformable raptor mesh and a rigid plane (see Figure 4.12, Left), the tests are executed in conditions of various numbers of *mechanical DOFs* and contact constraints. We compare the performance of our methods of *isodof* / *Reuse isodof* to the method proposed in Courtecuisse et al. (2014) that currently

provides the fastest contact resolution in SOFA framework.

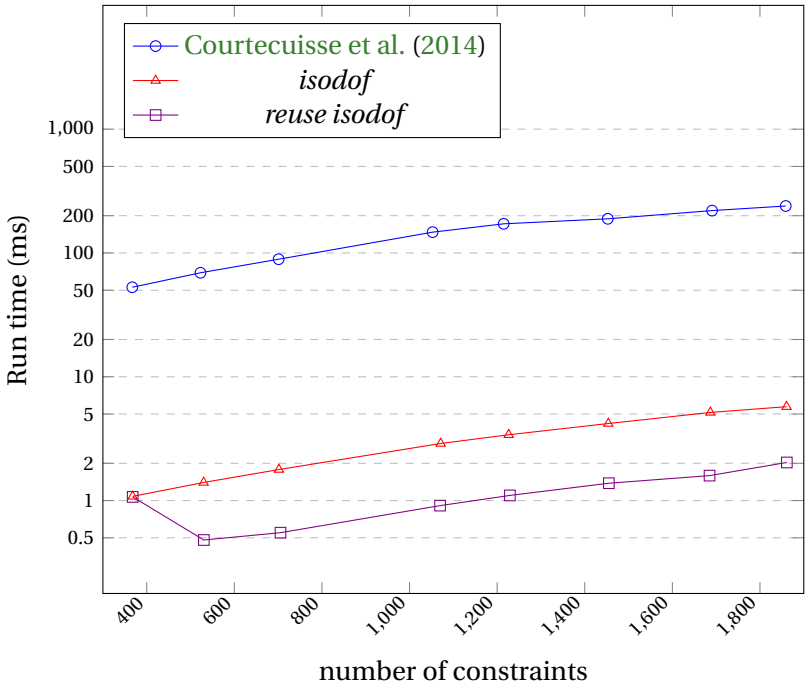


Figure 4.9: Computation cost of Schur-complement of different methods for various constraint number: contact simulation between a rigid plane and a deformable raptor mesh with 36069 *mechanical DOFs*. By changing the elasity parameters, the contact area is varied, leading to different constraint numbers. We note that the y axis is logarithmic in this figure.

In Figure 4.9, the *isodof* method shows an average speedup of 47.42× compared to Courtecuisse et al. (2014) and the *reuse isodof* shows a further speedup of 2.81× compared to the standard *isodof* and 133.31× compared to Courtecuisse et al. (2014). Our methods efficiently limit the computation cost: with 367 constraints (326 *mechanical DOFs* impacted) the *isodof* method takes 1.08ms; with 1860 constraints (1376 *mechanical DOFs* impacted) the *isodof* method takes 5.73ms, while the *reuse isodof* method takes 2.03ms by reusing 99.8% of the *isodofs*.

In Figure 4.10 we show the performances of our methods according to different mesh dimensions. The method in Courtecuisse et al. (2014) reveals a quadratic function according to the number of *mechanical DOFs* as the discretization of the mesh has an impact on the searching of contact pairs in the proximity collision detection. When the problem size is increased, Courtecuisse et al. (2014) (dense resolution) suffers from extremely large computation cost, while this cost is limited with our new methods (10.05ms for the *isodof* method and 2.32ms for the *reuse isodof* method). The speedup from Courtecuisse et al. (2014) to the *isodof* is averagely 50.55× and it is enlarged up to 133.57× for the *reuse isodof*. As expected, our method is poorly sensitive to the mesh dimension while the cost

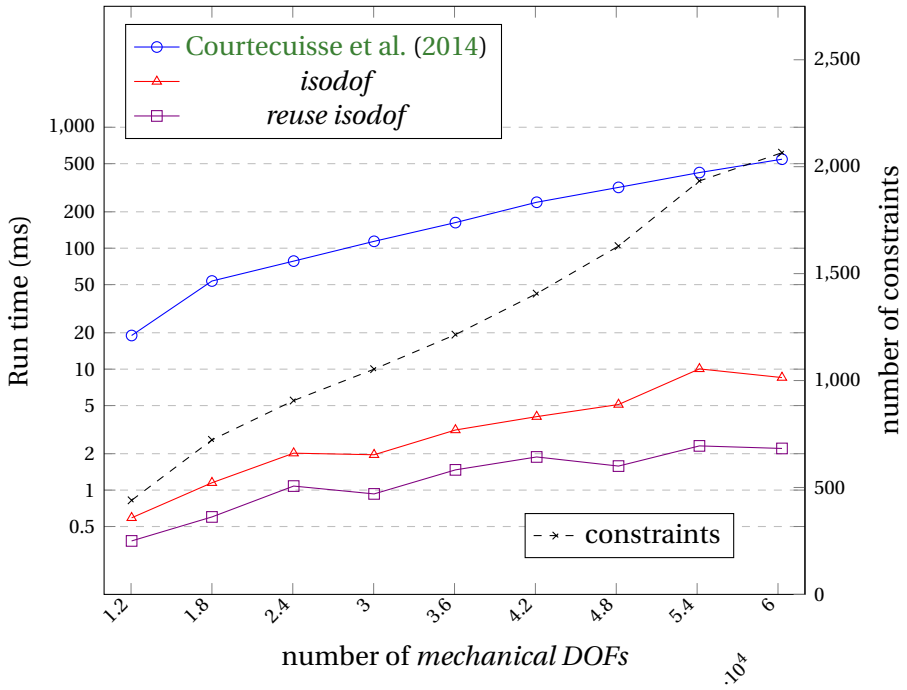


Figure 4.10: Computation cost of Schur-complement of different methods for various *mechanical DOFs*: contact simulation between a rigid plane and a deformable raptor mesh with various discretization. The mesh discretization has an impact on the contact constraints (dashed black line in the figure). We note that the y axis is logarithmic in this figure.

Method		Performance (in *ms*)						
Free Motion	Build W	Build + Fac.	Free Motion	Build **W**	Trans.	GS	Corr.	Time step
Pardiso-16		1129.72	37.7	57.49		57.46	26.32	1308.69
PCG + $\mathbf{LDL}^T$	Courtecuisse et al. (2014)		41.45(10#it)	554.15	27.09	48.06	3.60	674.35
PCG + $\mathbf{LDL}^T$	*isodof*		42.20(10#it)	13.53	26.97	47.55	3.83	134.08
PCG + $\mathbf{LDL}^T$	*reuse isodof*		42.45(10#it)	6.56	26.22	46.58	3.82	125.63

Table 4.1: Collision simulation between a rigid plane and a deformable raptor with 59 529 *mechanical DOFs* and 2250 contact constraints. Performance of various methods: system assembly + analysis for *Pardiso* + factorization (**Build + Fac.**), free motion resolution, Schur-complement (**Build W**), transfer **W** from GPU to CPU (**Trans.**), Gauss-Seidel (**GS**) for constraint resolution, corrective motion (**Corr.**) as well as the entire time step (**Step**). For the implementation in ***Pardiso*** (processed in 16 parallel threads), an augmented system with the *constraint Jacobian* is factorized, while the factorized system is used in the resolutions of the free motion, the Schur-complement, and the corrective motion.

of method in Courtecuisse et al. (2014) goes far beyond the real-time computation with large mesh dimensions.

In Table 4.1 we evaluate the entire time step for a real-time application. We compare our methods with the solvers in *Pardiso project*, which is a popularly used library for linear algebra due to its efficiency, especially while processing in parallel CPU threads. For

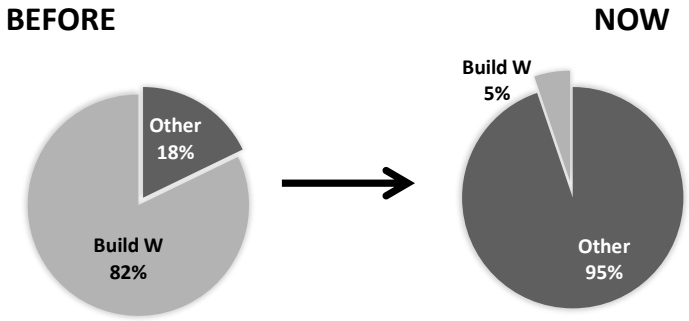


Figure 4.11: From Courtecuisse et al. (2014) to our methods, change of contribution of the Schur-complement in the entire time step

direct solvers in *Pardiso*, a factorization process is necessary before the free motion resolution and the Schur-complement. Although the *Pardiso* for the Schur-complement (i.e., the method in Petra et al. (2014)) is very fast, it requires a prerequisite augmented factorization. Computing the factorization depends directly on the dimension of *mechanical DOFs*, making it very difficult to achieve real-time computation for large-scale problems. On the other hand, the asynchronous preconditioner (i.e., the method in Courtecuisse et al. (2010a) (PCG) for the free motion, and the method in Courtecuisse et al. (2014) for the Schur-complement) removes the costly building and factorization stage out of the main simulation loop, showing a significant speedup compared to the direct solvers. Although this method gains significant speedup, it becomes prohibitive in large-scale contact problems in real-time applications (building **W** takes $554.15ms$ and 82.18% in a time step). With the *isodof* and *reusing isodof* methods, we succeed to limit the Schur-complement within a very low computation cost that is $13.53ms$ (10.09%) for *isodof* method and $6.56ms$ (5.22%) for *reuse isodof* method (see Figure 4.11)

4.2.5.1 Assembling compliance vs. unbuilt scheme

As discussed in Section 4.2.1, when using a relaxation method such as **PGS**, the unbuilt scheme can be an option to solve the problem. In Table 4.2, we compare the additional computational cost between the unbuilt scheme and the assembly scheme (building **W**). We collect the data from two of our examples in the application section 4.3.1 to compare the methods in both limited (*Pass Torus*) and rich contact (*Rich Contact*) cases. According to different contact cases, **PGS** will need several iterations to hundreds of iterations to converge. As illustrated in the table, for both limited/rich contact cases, the extra cost of applying $\mathbf{LDL}^{\mathrm{T}}$ in each iteration overcomes the overhead of building the compliance matrix after several **PGS** iterations. With ten iterations, building **W** with our methods is already more efficient than the unbuilt scheme. This gap tends to be extremely large when

the relaxation method needs hundreds of iterations to converge. We note that we use the same $\mathbf{LDL}^{\mathrm{T}}$ factorization with the same reordering technique for the different schemes in this test. For the unbuilt scheme, the $\mathbf{LDL}^{\mathrm{T}}$ resolution process in Algorithm 5 is optimized on GPU using a domain-decomposition technique in the forward/backward substitutions: The patterns of $\mathbf{L}/\mathbf{L}^{\mathrm{T}}$ matrices are partitioned into sub-blocks that correspond to the branches on the elimination tree (see Figure 4.4). We can parallelize the resolution of branches for those who have no data dependency between each other. Following the structure of the elimination tree, such parallel resolution can be processed recursively until the root.

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

Table 4.2: Comparison of the additional computation cost (in ms) between the assembly scheme and the unbuilt scheme. Since the choice of scheme will not impact the **PGS** performance, we can compare the **overhead/extra cost** in different schemes. The unbuilt scheme requires an extra cost of solving $\mathbf{LDL}^{\mathrm{T}}$ (see Algorithm 5) in each **PGS** iteration. Hence, the total extra cost scales linearly with the number of iterations. In contrast, the assembly scheme with our new method only requires a small overhead of building **W** and will not cause any extra cost in the iterations.

4.3 Applications

4.3.1 Applications of isolating mechanical DOFs in various scenarios

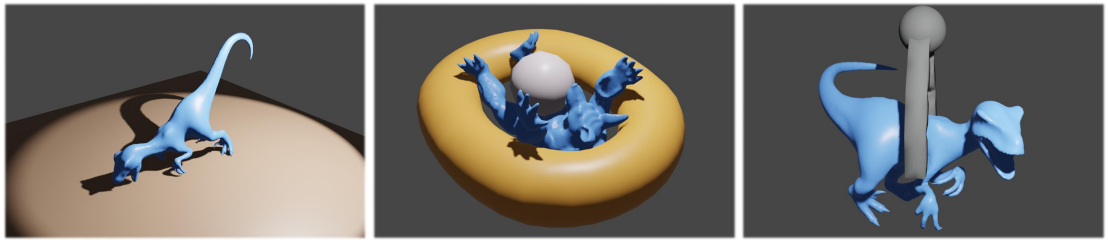


Figure 4.12: Our new methods provide a fast process for the schur-complement while being capable of completing different challenges in various examples such as simple collision test (Left), complex interaction (Middle), and gripping task (Right).

In this section, we apply our methods to different examples. To perform fast collision detection, we use the GPU-based method Allard et al. (2010) that relies on volume inter-

penetration. The tests are executed in the following examples, with various deformable meshes and challenges (detailed meshes, multi-objects, complex interactions, heterogeneous materials...).

4.3.1.1 Complex interaction: Pass Torus

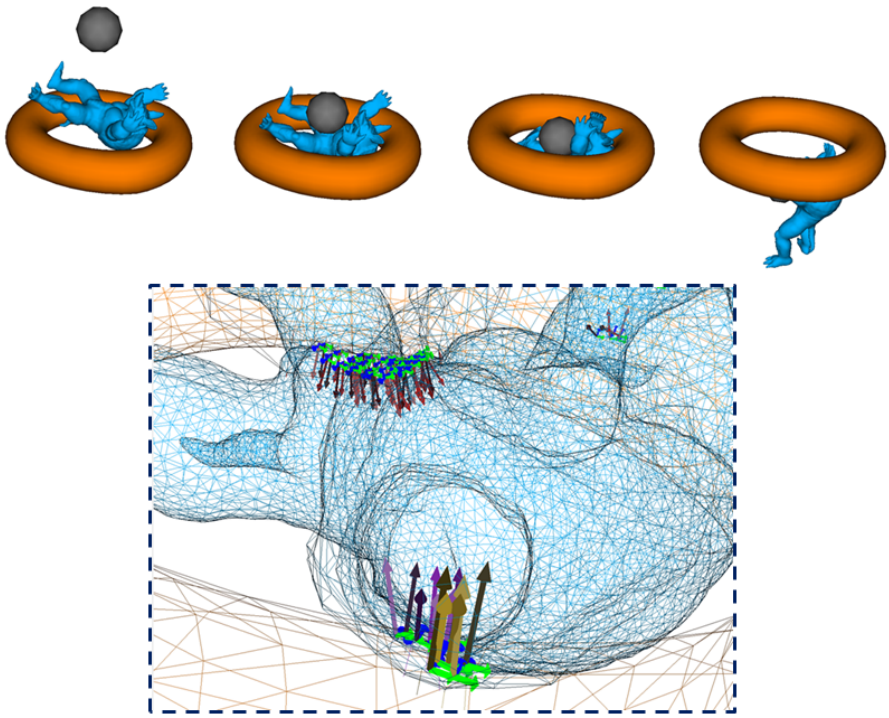


Figure 4.13: Pass a deformable armadillo through a torus. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

Figure 4.13 shows complex interactions between a rigid ico-sphere, a fixed rigid torus, and a deformable armadillo that has a detailed discretization with 31302 *mechanical DOFs*. The contact from the sphere is on the center of the armadillo, pushing it through the torus. On the other hand, the stiffness and the contacts on arms and legs generate the forces resisting against sphere's movement. It is, therefore, necessary to efficiently discretize contacts with mechanical coupling to compute and distribute the contact forces.

4.3.1.2 Multi-objects

Our methods are compatible with multi-object systems, such as a scenario of multi-torus (see Figure 4.14). In the test we simulate 10 deformable torus with 3357 *mechanical DOFs* for each one. Each deformable torus is contacting with the others and the fixed pillars.

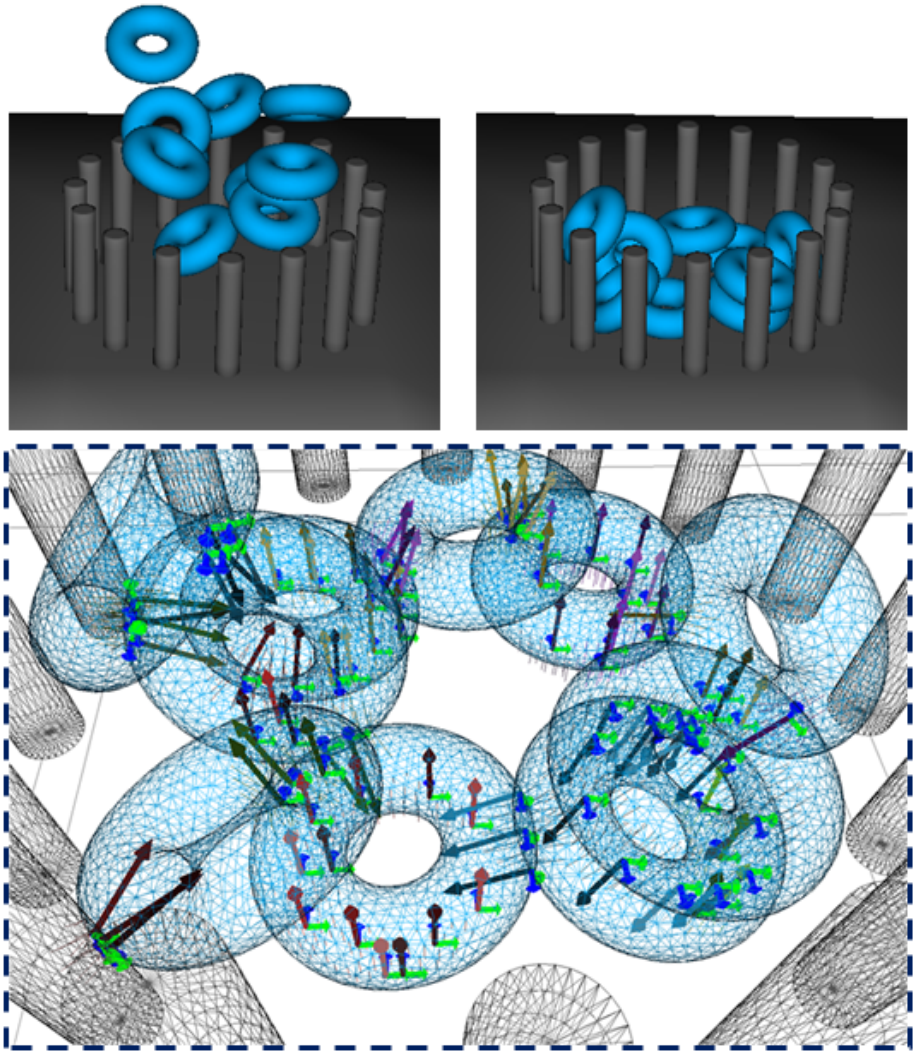


Figure 4.14: Collision between multiple deformable torus. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

Although we solve smaller mechanical problems in this scenario, the contact forces are transmitted among the objects, forming a complex multi-object system. Beyond the fact that the methods also provide speedup in multi-body contact simulations, building the *Delasus operator* is crucial to take into account the mechanical coupling of the system.

4.3.1.3 Dynamic contact: Rolling

To better evaluate the *reuse isodof* method, we design a "dynamic test" where the contacting area keeps shifting and numbers of new *isolated DOFs* appear in each time step (Figure

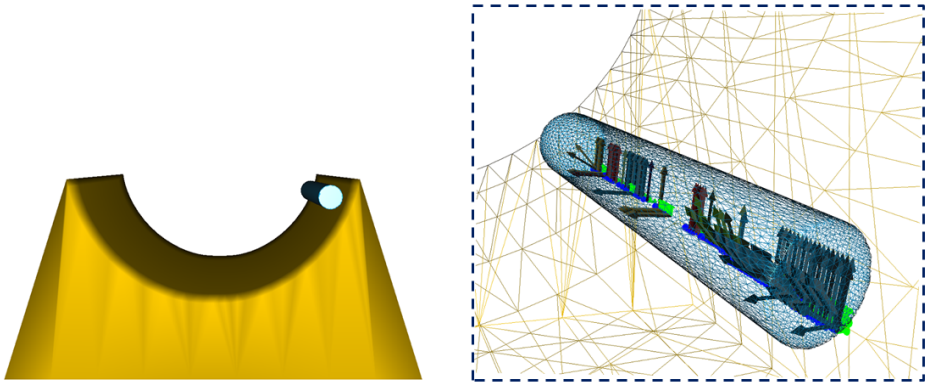


Figure 4.15: Rolling cylinder: a dynamic contact test for the *reuse isodof* method. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

4.15). Although the contact area keeps changing, more than 89% of *isodofs* are reused. The *reuse isodof* method has an additional speedup of 1.5× compared to the standard *isodof* method ($7.58ms \rightarrow 5.06ms$). In this case, the *reuse isodof* method still receives interest since the *isodofs* are efficiently reused between the consecutive time steps even when the contact area is constantly varying.

4.3.1.4 Heterogeneous material

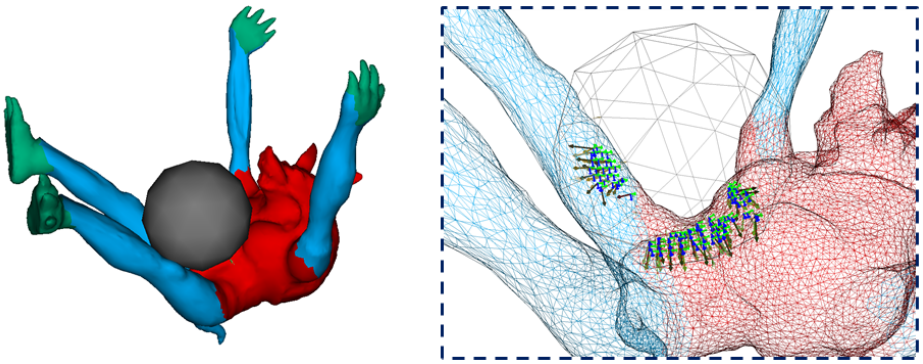


Figure 4.16: Heterogeneous material: the red parts are 10× stiffer than the blue parts, while the green parts are fixed. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

Our methods are compatible with heterogeneous materials. In Figure 4.16, we simulate the collision between a rigid ico-sphere and a deformable armadillo of heterogeneous material with 31302 *mechanical DOFs*. The main difficulty of this example is related to the fact that the sphere applies contact forces on stiffer parts (in red), whereas softer parts (arms and legs in blue) should deform more obviously. By formulating the *Delasus oper-*

ator, the contacts are solved with mechanical coupling and contact forces are efficiently distributed in heterogeneous material while enforcing fast computation time.

4.3.1.5 Needle Insertion

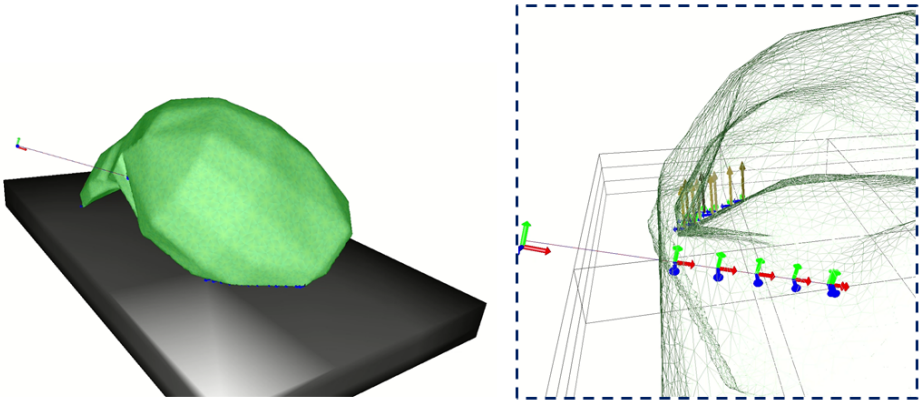


Figure 4.17: Needle insertion. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

Besides contact constraints (unilateral constraints), our methods are also compatible with other constraint types. We apply our methods in a scenario of needle insertion, which is a popular topic in medical simulations Adagolodjo et al. (2019). Figure 4.17 shows a process of inserting a needle into a deformable liver mesh (highly detailed, with 31566 *mechanical DOFs*). Our methods are compatible with the needle constraints (bilateral constraints) that only impact the *mechanical DOFs* nearby the insertion trajectory, making the *isodof* method very efficient. Moreover, the *reuse isodof* method can benefit a significant speedup from reusing the *isodofs* nearby the insertion trajectory.

4.3.1.6 Rich contact

We apply our methods in a scenario where a soft pad is covering on a rigid sphere. The contact area is very large and 1200-1300 contact constraints are generated while about 900 out of 6561 *mechanical DOFs* are impacted by the constraint. The ratio of constraints/*DOFs* in this scenario is about 0.2, which is significantly higher than other examples (less than 0.02). Our methods remain efficient in such rich contact cases: building $\mathbf{W}$ takes 2.03ms with the *reuse isodof* method, providing a speedup of 14.98 × compared to the method in Courtecuisse et al. (2014).

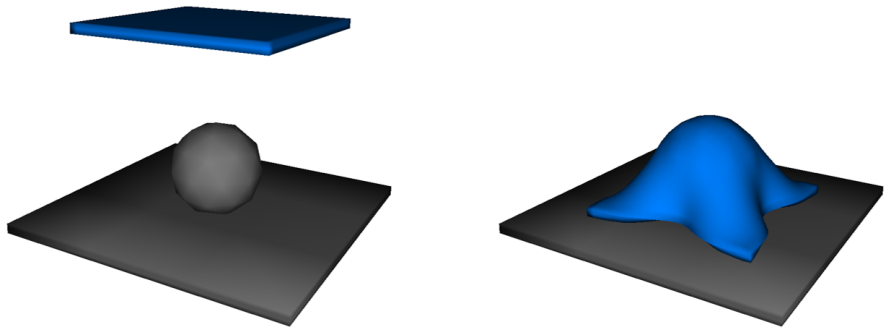


Figure 4.18: Rich contact.

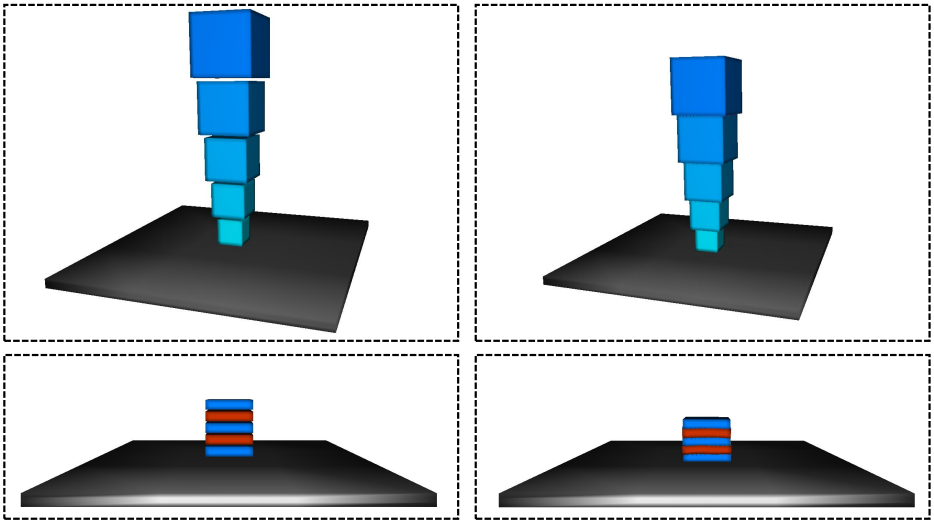


Figure 4.19: Stacking problems. Top: assembling **W** allows to couple the forces between stacking boxes with increasing masses which are represented by gradient blues. Bottom: assembling **W** is especially important for the stability in a heterogeneous stacking scenario with different stiffness (the blue pads are 10 × stiffer than the red pads).

4.3.1.7 Stacking problem

We apply our methods in a stacking problem, which is similar to the example in Macklin et al. (2019). In the first scenario, the stacking boxes are modeled with FEM and large young modulus to have a behavior near to rigid bodies. With increasing masses of the boxes, **PGS** can still handle the problem when the total mass ratio is of 256:1. When dealing with a mass ratio of 4096:1, the problem becomes very poorly conditioned and difficult to be solved with **PGS**. However, by assembling **W**, our methods allows to formulate a standard complementarity (linear system in Equation (4.17) combined with complementarity conditions in Equation (4.3) and (4.5)) discussed in Erleben (2013), making it flexible to be solved by different methods (e.g. pivoting methods to handle such a ill-conditioned

problem).

Besides the homogeneous stacking, our methods are also tested in a heterogeneous stacking problem. In the second scenario, the stacking soft pads are modeled with different stiffness. Such a test will be failed when the contact forces are not coupled (**W** is diagonal) as the system is extremely unstable, which is discussed in Andrews and Erleben (2021). In this test, the coupling of contact forces is very complex since the interaction on each contacting faces has impact on the other objects. Therefore, building the compliance matrix is very important to propagate the forces over different objects. Our methods allow to efficiently address this problem by fast building **W** that correctly couples the contact forces.

4.3.1.8 Gripping raptor

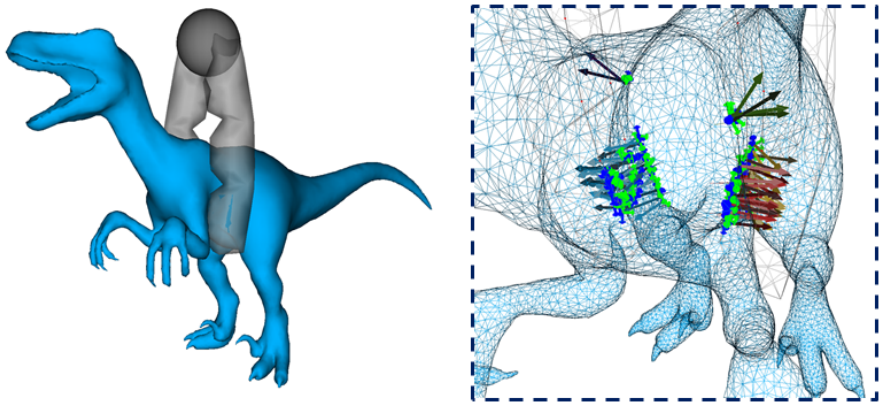


Figure 4.20: Grip a raptor with friction constraints. The zoom figure shows the detailed mesh discretization with arrows that represent the constraints of contact and friction.

We apply our methods on a gripping test. In a first scenario (see Figure 4.20), we use a soft gripper Duriez (2013) with two fingers to compress a deformable raptor mesh with 30033 *mechanical DOFs*. In a second scenario (see Figure 4.21), we fetch the raptor and deal with a pick-and-place task. This scenario implies comprehensive challenges: The deformable raptor is highly detailed, and the soft fingers are also deformable models (with 474 *mechanical DOFs* for each one). While gripping (compressing) the raptor, the fingers and the raptor are deformed to fit the contacting surface, generating numbers of contact constraints. Lifting, rotating, and moving the raptor by the fingers are complex operations where friction constraints are necessary. Moreover, the fingers, the raptor, the rigid plane, and the torus are grouped into a multi-object system, requiring efficient distribution of contact forces through mechanical coupling.

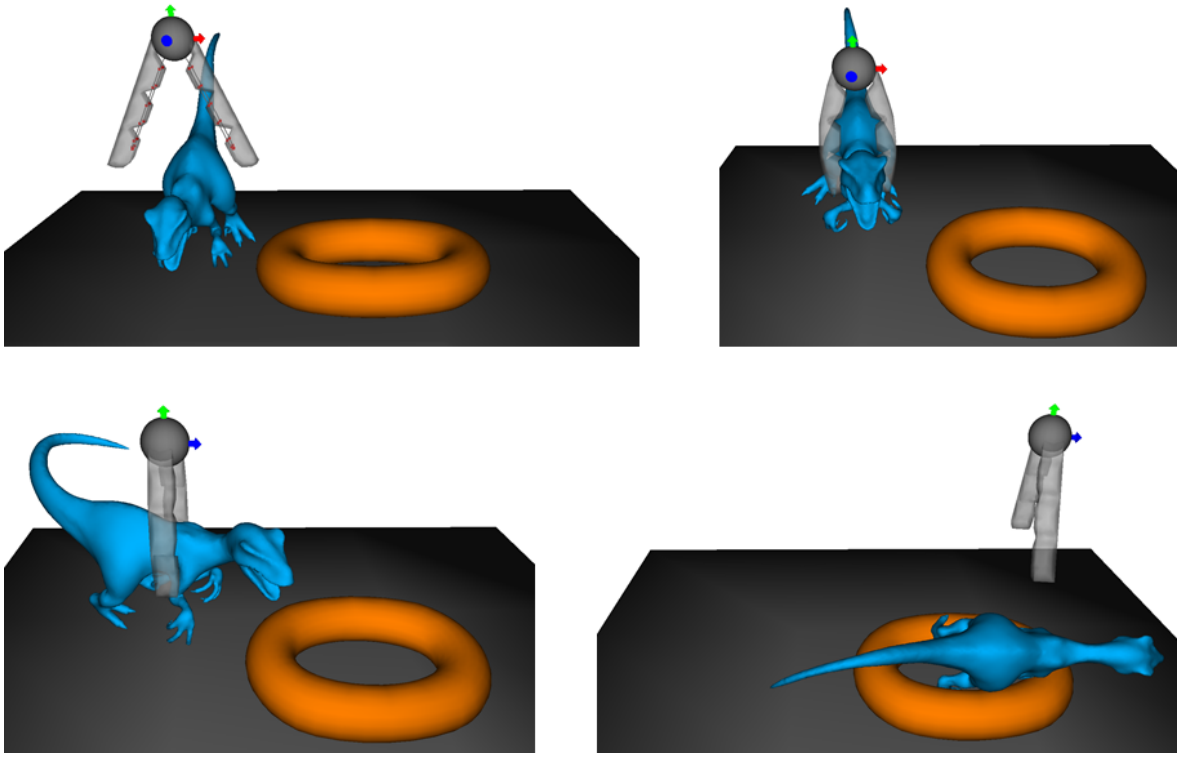


Figure 4.21: A pick-and-place task with Soft-Robot

4.3.1.9 Evaluation

In these scenarios, we meet various challenges: The complex interactions usually require to efficiently distribute the contact forces through the mechanical coupling, such as the cases in multi-object systems and in the problems with heterogeneous materials; The needle insertion operation necessitates simulating both the unilateral constraints for contacts and bilateral constraints for needle insertion at the same time; The pick-and-place task is even more challenging with different requirements. Moreover, all the scenarios simulate highly detailed meshes, raising large-scale problems. In this case, typical CPU-based or GPU-based approaches suffer from high computation costs to assemble the system $\mathbf{W}$ for the constraint resolution. However, our methods can complete the challenges with limited costs to build the compliance matrix. In Table 4.3 we evaluate the computation cost of the Schur-complement as well as its contribution in an entire time step. Although in different applications, our methods have different performances, we succeed in limiting the cost of building the *Delasus operator* within less than $10ms$ in all the examples. Consequently, with our methods, the Schur-complement process in constraint-based resolutions is no more a critical obstacle in real-time simulations.

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

Table 4.3: Performance (in ms) in various examples: we evaluate the computation cost of the Schur-complement (**Build W** with the *reuse isodof* method), its percentage in a time step, and the speedup compared to the method in Courtecuisse et al. (2014).

4.3.2 Isolating mechanical DOFs in needle insertion simulations

In this section we provide more details about the application of our *isolating mechanical DOFs* in needle-based procedures, which are beneficial for patients but are technically challenging to perform. Martin et al. (2023) proposes a new needle insertion approach, relying on the needle-tissue interaction model introduced in Duriez et al. (2009). The location of constraints is improved by prescribing the relative displacements of all the DOFs involved during the insertion. It results from the intersections between the needle and volume meshes. Compared to previous solutions, our approach improves the accuracy and generates a realistic haptic rendering, but a significantly larger number of constraints are generated. To limit the computational cost, the method is combined with the *isolating mechanical DOFs* approach Zeng et al. (2022) proposed in Section 4.2.3.

During needle insertion, the DOFs impacted by the constraints (known as *isolated DOFs*) in the previously inserted part remain in the current time step. Therefore, the improved *IsoDOFs* method, which reuses common results in consecutive time steps, is exploited to further accelerate the resolution.

To assess the improvement of constraint placement by the intersection process, a rigid needle was numerically inserted into a very soft beam-like gel (500 Pa) (see Fig. 4.22). A mesh of 50 nodes over 15 cm in the direction of the needle, against 6 nodes over 2 cm in the two other directions was used.

A needle was numerically inserted into a gel, following a defined curved trajectory. During the insertion, bilateral and friction constraints were generated from the intersection process. The time required to build **W** and the number of impacted DOFs were

evaluated (see Fig. 4.23). Comparisons were made between the *IsoDOFs* and *reuse IsoDOFs* methods, exposed in Zeng et al. (2022). Results show that only a minor portion of DOFs was newly constrained between consecutive time steps, thus decreasing the computation time required to build **W**: although our approach gives rise to a large number of constraints, the reuse of *IsoDOFs* allows the computation of **W** to be minimally impacted.

4.4 Conclusion

This chapter provides an introduction to the background of constraint resolution in FE simulations, followed by the presentation of novel methods aimed at efficiently address-

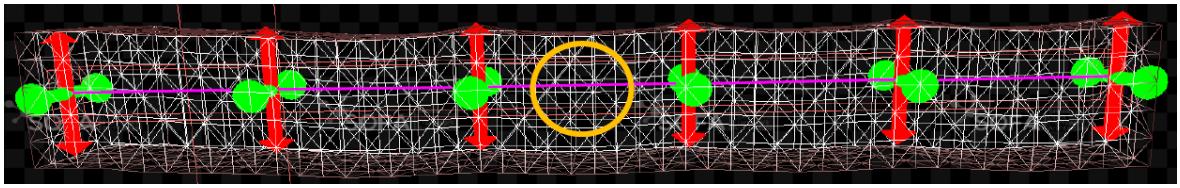


Figure 4.22: *Bilateral constraints (green and red arrows) were generated between the rigid needle (purple) and the volume. Gravity was applied to generate a motion of the unconstrained volume nodes, producing unrealistic needle-tissue relative motions as circled.*

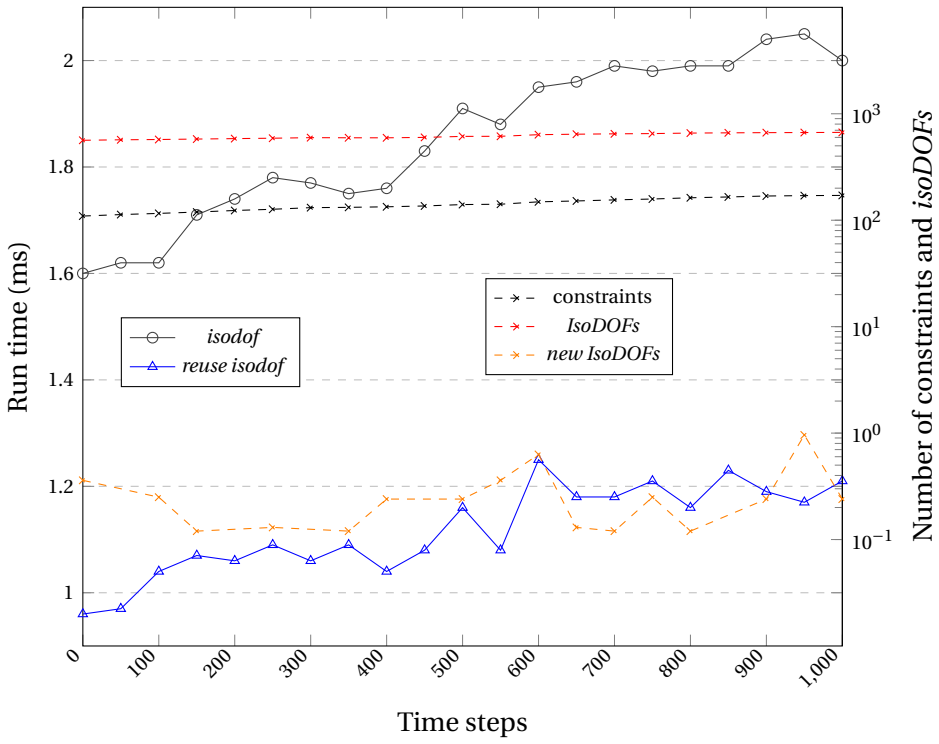


Figure 4.23: *Average computation time to build* **W** *(in ms) over 50 time steps from the reuse and standard IsoDOFs methods, during a needle insertion. Amounts of reused and total IsoDOFs are compared. Most IsoDOFs are reused between consecutive time steps.*

ing issues in contact simulations.

To overcome the computational cost associated with building the compliance matrix, we propose fast approaches based on precondition-based contact resolution. The *isodof* method reformulates the costly Schur-complement by isolating the mechanical DOFs in the constraint Jacobian matrix. This reformulation enables us to perform sparse resolution, which is suitable for parallelization on a GPU. The *reuse isodof* method further reduces the problem dimension, improving the efficiency of computing the Schur-complement. Even when the number of constraints and mechanical DOFs is significantly increased, our methods effectively limit the computational costs, eliminating the obstacle posed by the Schur-complement in real-time simulations (see Figure 4.11). Additionally, our methods yield mathematically equivalent results to those obtained by the method described in Courtecuisse et al. (2014), thereby enabling the computation of large-scale real-time simulations involving contact and friction.

Our work has limitations and suggests avenues for future research. For the *isodof* method, the fast computation achieved through our methods introduces a limitation for asynchronous preconditioners. While the computation cost in the main simulation loop is significantly reduced, the factorization in asynchronous threads remains computationally expensive. Consequently, the number of simulation steps required to update the asynchronous factorization increases considerably. Using a slowly updated matrix may lead to significant errors in the case of large deformations. One potential solution is to perform the asynchronous factorization on a GPU, which would involve conducting asynchronous multi-GPU computations.

Lastly, the *isodof* method relies on a precomputed structure based on the elimination tree. Any modification to the mesh topology would invalidate the precomputed data, rendering the method incompatible with cutting processes. In response to this challenge, we will discuss a new approach for simulating cutting operations in Chapter 5 and explore potential methods to extend the *isodof* method to accommodate topological changes in Chapter 6.

CHAPTER 5

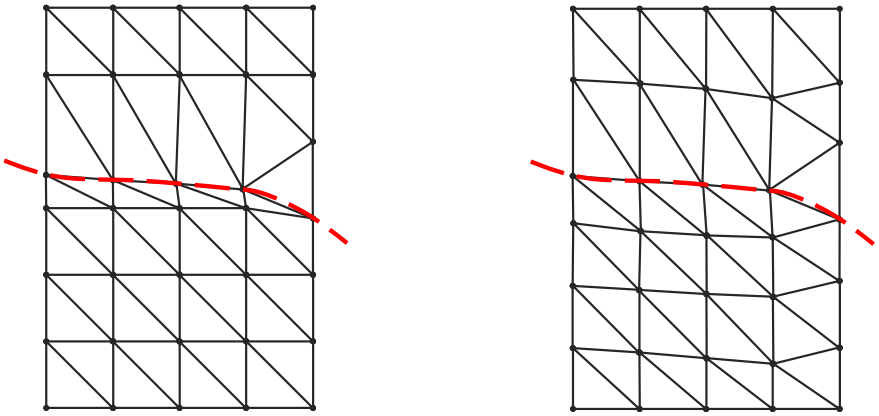
DYNAMIC CUTTING SIMULATIONS

Interactive simulations have the potential to revolutionize surgical training by enabling surgeons to practice various procedures. As one of the typical applications, cutting deformable solids is always challenging due to topological changes that introduce additional issues. Although simple techniques, such as element deletion and splitting along existing faces, are effective, they produce poor cutting surfaces. Mesh refinement is a popular alternative in recent studies as it provides better cutting surfaces and maintains reliable volume conservation. However, this method may generate ill-shaped elements and large amounts of new elements, invalidating real-time performance. Alternatively, the vertices-snapping approach offers the advantages of mesh refinement without generating new elements. Despite its potential benefits, this method was limited to static simulations due to issues related to energy conservation and topology changes.

This chapter aims to simulate progressive cutting with an advanced vertices-snapping method. Our method optimizes the mesh quality in a vritual FE system, and enables real-time peformance with efficient topology operations and fast numerical solver. According to the topological modification, we further propose an efficient strategy to update the stiffness matrices and the precomputed structure in matrix-free iterative solver. All these method finally result in an efficient cutting method with good mesh quality and smooth cutting surface.

5.1 Dynamic cutting simulation with mesh quality optimization

As discussed previously, we opt for the vertices-snapping strategy Nienhuys and van der Stappe (2001) as our preferred method for fitting the cutting boundary surface onto the cutting path. This approach avoids generating new elements, thereby minimizing additional computational costs in numerical solvers.



(a) Snapping the boundary **(b)** Elastic snapping

Figure 5.1: (a) The original snapping strategy risks of compressing the elements, leading to ill-shaped mesh. (b) We propose solving an elastic problem, where the equilibrium between the snapping forces and the elastic internal forces optimizes the mesh quality.

We propose solutions to address the major limitations of the original vertices-snapping strategy in dynamic simulations:

1. **Unscheduled cutting path** Working with an unscheduled cutting path requires a robust method that is able to deal with the potential perturbations in applications. To make sure a reliable and progressive cutting path, we propose generating the path mesh with point cloud and using polynomial fitting to smooth the path.

2. **Ill-shaped mesh** During cutting, the displacement of vertices may lead to ill-shaped elements, particularly when elements are significantly compressed. To overcome this issue, a previous study Serby et al. (2001) proposed a method for re-meshing the topology to improve mesh quality after each cut. However, this solution is primarily suitable for static simulations, and the re-meshing process adds extra computational costs to numerical solvers. In the case of dynamic simulations, we propose a modification to the conventional vertices-snapping strategy. Rather than

only snapping vertices near the cutting path, we suggest solving am elastic problem to optimize the mesh quality (see Figure 5.1).

3. **Energy conservation** In dynamic simulations where the objects are deformed, fixing vertices in their snapped positions may invalidate external/internal forces, leading to energy loss in the finite-element system. To address this issue, we propose snapping vertices in the reference shape such that the deformed object conforms to the cutting path while preserving existing external and internal forces.

4. **Separating boundary** In dynamic simulations, finding a boundary geometry that is accurately snapped onto the cutting path and effectively divides the local mesh into two parts can be challenging. Additionally, ensuring the connection of the boundaries between consecutive time steps is a crucial issue. To address these challenges, we propose a set of efficient geometry operations to search for the boundary. The details of the method are explained in the subsequent sections.

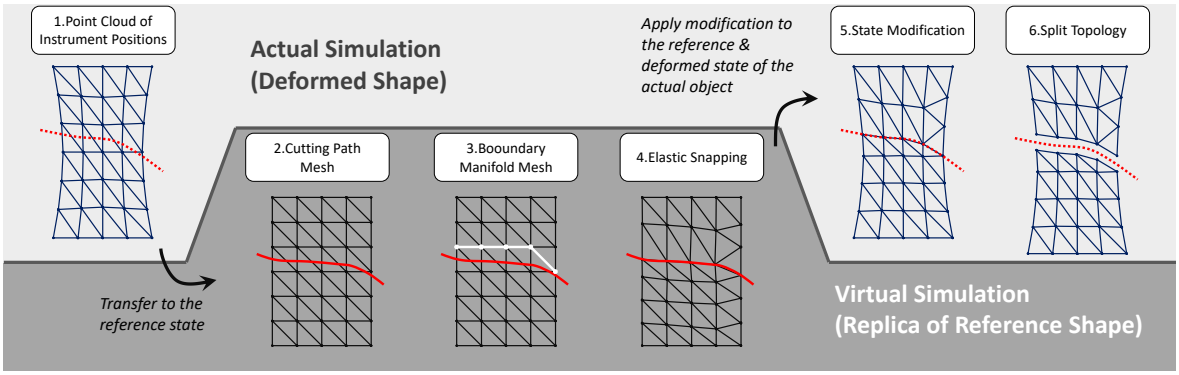


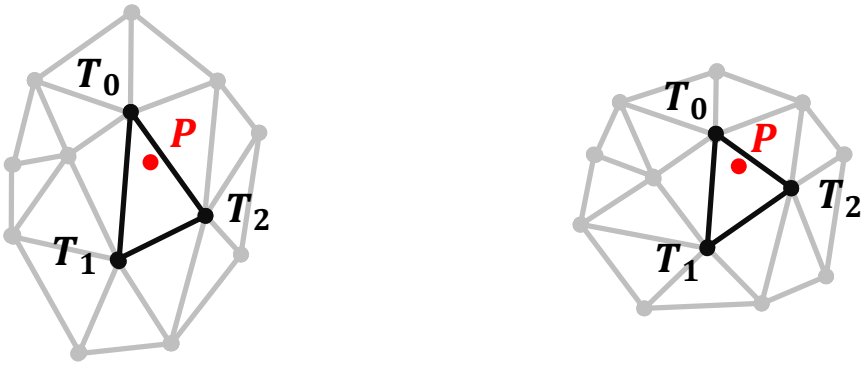
Figure 5.2: Workflow of the cutting method within a time step.

We propose a new method that integrates the above approaches to achieve the desired outcome. The workflow of the method is illustrated in Figure 5.2. At the beginning of each time step, we search for the interactions between the mesh and the instrument, generating a point cloud (with the instrument positions) in the deformed state. Subsequently, the point cloud is transferred into the reference state. After a polynomial fitting that reduces perturbations, we create a surface mesh by a triangulation algorithm, depicting the cutting path in the reference shape. The following step involves the identification of a boundary geometry consisting of pre-existing surfaces in the mesh that requires cutting, while ensuring that the boundary geometry is in close proximity to the cutting path. All these information is transferred into a second simulation, where the boundary vertices undergo constraint forces that leads their displacement onto the cutting path. The resultant displacement is then applied to the reference state of the actual physical object being

simulated. Finally, the method utilizes the boundary geometry to separate the local mesh into two distinct parts.

5.1.1 Cutting path

In this section, we propose a method for generating a cutting path in 3D problems. First, the positions of the cutting instrument are recorded in each time step, with a user-defined sampling distance to avoid excessive density. This collection of positions creates a point cloud representing the instrument's trajectory. In order to obtain a cutting path in the reference shape, the point cloud is transferred using barycentric coordinates (see Figure 5.3) between each point and its corresponding element in the mesh. For each point in the point cloud, we determine its corresponding element as the element that contains the point if the point lies within the mesh, or the nearest element if the point is outside the mesh.



(a) Deformed shape. **(b)** Reference shape.

Figure 5.3: We use the barycentric mapping to transfer the point cloud from the deformed shape to the reference shape: (a) In the deformed shape, we compute the barycentric mapping u, v, and w such that $\mathbf{q}(P) = u\mathbf{q}(T_0) + v\mathbf{q}(T_1) + w\mathbf{q}(T_2)$ while $u+v+w=1$. (b) In the reference shape, we use the barycentric coordinates to compute the relative position for the point: $\mathbf{q}_{\text{ref}}(P) = u\mathbf{q}_{\text{ref}}(T_0) + v\mathbf{q}_{\text{ref}}(T_1) + w\mathbf{q}_{\text{ref}}(T_2)$

Due to displacements and deformations, the transformed point cloud may not be as smooth as it is in the deformed shape. To address this, a polynomial fitting procedure using Moving-Least Squares (MLS) Alexa et al. (2003) is used to create a smooth plane in the reference state. Finally, a fast triangulation method Marton et al. (2009) is employed to generate a surface mesh that represents the cutting path. Both the polynomial fitting and triangulation methods are implemented in the Point Cloud Library (PCL). In the applications with interactions of user, the micro perturbations are usually inevitable due to unpredictable movement of the object and the instrument. Such as the slight shaking of hands that handle the cutting tool, or the motion of organ with respiration in the virtual

surgery. Therefore it is important to use MLS to efficiently smooth the cutting path. However, such smooth will lead to a drawback: In progressive cutting, the front end of the cutting path will be continuously updated with new polynomial fitting, leading to slight changes of the last path (see Figure 5.4). The effect of such change will be discussed in the next section.

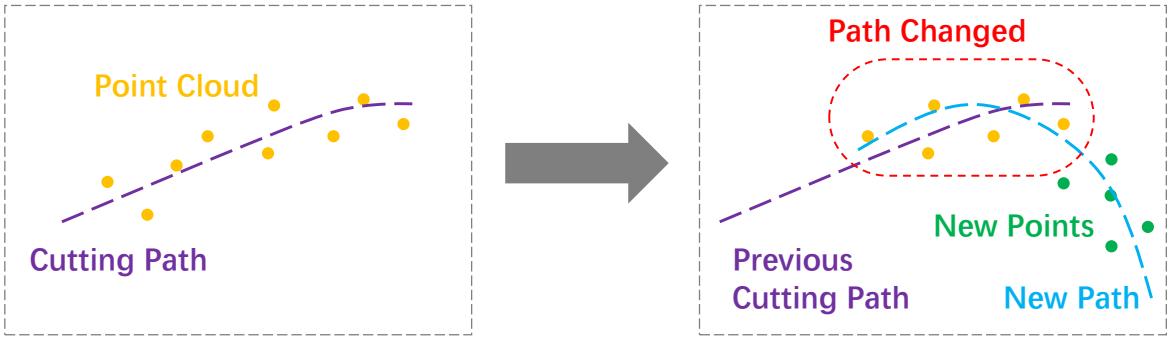
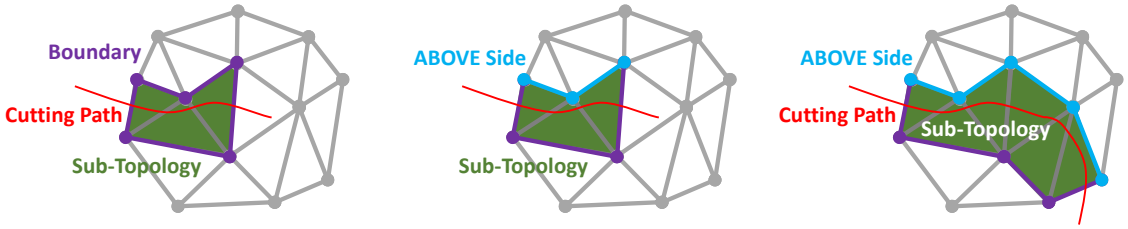


Figure 5.4: Moving Least Square (MLS) can efficiently smooth the cutting path, but will cause slight change in the last cutting path.

5.1.2 Boundary surface

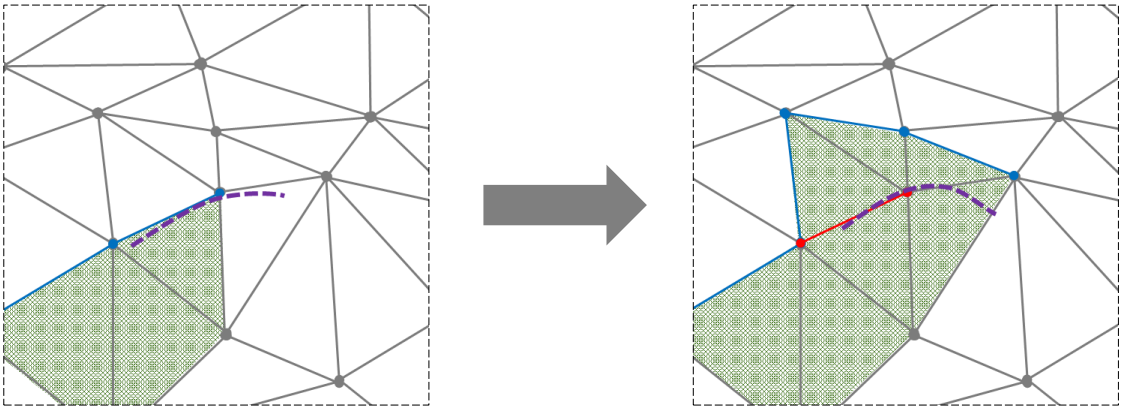


(a) **(b)** **(c)**

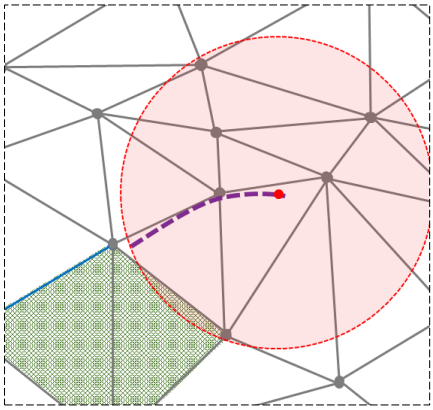
Figure 5.5: Our geometry operations to search for the boundary surface: (a) the interacted elements are grouped as a 3D-manifold mesh and its boundary forms a 2D-manifold mesh. (b) by selecting one side of the boundary, we obtain the boundary surface. (c) as the simulation is progressing, the subset is augmented and remains a 3D-manifold mesh. Therefore the boundary remains a 2D-manifold mesh.

After generating the cutting path mesh, the next step is to find out which vertices to be snapped. To ensure that the mesh is correctly cut, we search for a 2D-manifold mesh that consists of surfaces inside the original mesh. This 2D-manifold mesh, named as "boundary surface", will be snapped onto the cutting path and be used as splitting boundary that locally divide the mesh into two parts. As long as the cutting is progressing, the boundary surface found in a following time step should be able to connect to the previous one,

forming an enlarged 2D-manifold mesh and ensuring the progressive cutting. Finally, to minimize the vertices displacement caused by the snapping, the boundary surface should be as close as possible to the cutting path. With so many constraints, searching for the boundary surface is very difficult in progressive simulations.



(a) The detection of interactions is highly sensitive to the change of cutting path in Figure 5.4. This may cause topological error since a selected boundary cannot be unselected in subsequent time steps.



(b) We set a threshold to prevent the propagation of boundary manifold in the area where interactions remain being changed. The threshold is defined with the scale of the front end of cutting path for interaction detection.

Figure 5.6: Interaction change.

We propose a set of geometrical operations to search for the boundary surface (see the illustration in Figure 5.5): Tetrahedral elements that interact with the cutting path are computed, assembling a subset of the cutting mesh. Such subset of tetrahedron, calling "interacted sub-topology", can be considered as a 3D-manifold. The boundary of this sub-topology is divided into two parts **ABOVE**/**BELOW** by the cutting path. The boundary of a n dimension manifold mesh is a $n-1$ dimension manifold. Therefore, by selecting either **ABOVE** or **BELOW** side of the sub-topology's boundary, we obtain the desired bound-

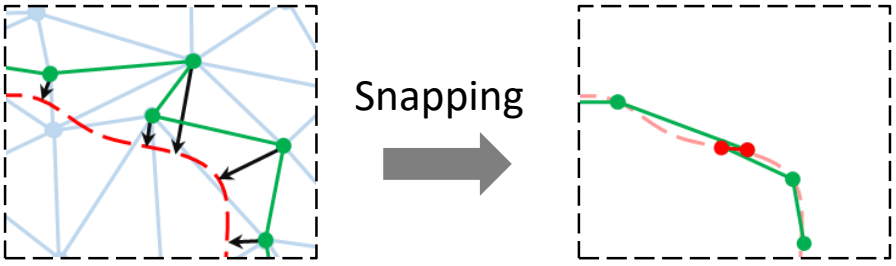
ary geometry that is guaranteed to be a 2D-manifold mesh. As the progressive simulation is going on, the new interacted tetrahedra will be connected to the previous subset (as long as the cutting path is continuous), forming a larger subset (see Figure 5.5c). Consequently the selected boundary is guaranteed to be a 2D-manifold mesh in close proximity to the cutting path, ensuring correct separation in progressive cutting.

As discussed in Section 5.1.1, the front end of the cutting path will be continuously updated with polynomial fitting and triangulation, in order to smooth the cutting path. Consequently the last cutting path will be slightly changed. As illustrated in Figure 5.6a, although the displacement is usually very slight in the physical space, the detection of interactions is highly sensitive to the change, as the detection output results only in 0 (not interacted) or 1 (interacted). To solve this problem, we set a delay with a threshold to exclude the latest topology operation (see Figure 5.6b). The threshold is defined with the scale of the front end of the cutting path, such that we prevent selecting the boundary surface from the sub-topology that still risk of changing.

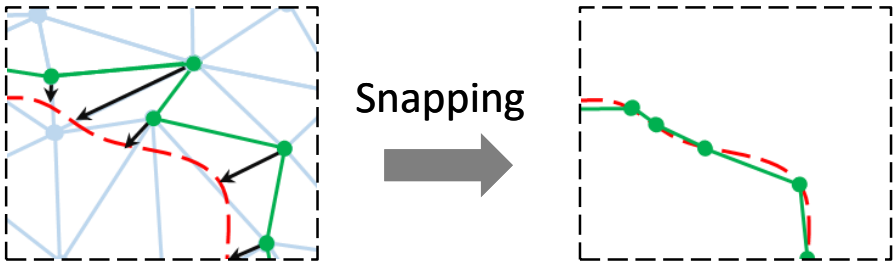
5.1.3 Vertices snapping

The subsequent step is to execute the process of snapping the boundary vertices onto the cutting path. A noteworthy enhancement from the original vertices-snapping method to our novel approach is the inclusion of neighborhood vertices while snapping the boundary vertices. This addition resolves the critical issue of creating ill-shaped meshes. To achieve this, we employ a second simulation wherein a finite element system is constructed with a replica of the reference mesh shape to be cut. This second simulation is called 'virtual simulation' while the cutting simulation is referred as 'actual simulation'. To maintain the surface point positions, the vertices on the mesh surface are subjected to a force field. Constraint forces are applied on the boundary vertices to move them onto the cutting path. Consequently, this solves an elastic problem, leading to the movement of all the vertices in the mesh. Upon one-step numerical resolution of the system, the finite element model attains equilibrium between the snapping forces and the internal forces. This equilibrium optimizes the distribution of element volumes, thereby minimizing the possibility of generating ill-shaped mesh.

Snapping constraints definition The first step is to define the direction of the constraints that will be applied to the boundary vertices. A straightforward strategy is to project a vertex to its closest position on the cutting path. This approach works well with most of meshes with good quality, but risk of compressing the elements in some arbitrary structures (see Figure 5.7a). Therefore we propose defining the constraint of a vertex with



(a) Projecting the vertices to their closest position on the cutting path may folds the boundary, leading to compressed elements.



(b) For each vertex, we use the average direction of the its neighbor edges in the sub-topology, preventing ill-shaped mesh.

Figure 5.7: Different strategies to define the constraint directions.

the average direction of its neighbor edges in the sub-topology. As illustrated in 5.7b, this strategy prevents folding the boundary, thus avoiding the compression of elements.

Elastic problem resolution The virtual simulation can be performed independently from the actual simulation, enabling us to employ a simplified constitutive law for the virtual object, even if the actual object employs complex models such as hyperelastic law. In the virtual problem, a finite element linear model is adequate, as the object is only deformed to fit the cutting path. Moreover, the boundary geometry defined in Section 5.1.2 is in proximity to the intended cutting path, avoiding significant deformation. In a similar formulation of the contact problem (see Section 4.1), employing the constraint-based technique in the virtual simulation necessitates solving a KKT system with bilateral constraints:

$$\begin{cases} \mathbf{A}_v \mathbf{x}_v - \mathbf{J}_v{}^{\mathrm{T}} \boldsymbol{\lambda} = \mathbf{b}_v \\ \mathbf{J}_v \mathbf{x}_v = \Delta \boldsymbol{\delta} \end{cases} \tag{5.1}$$

where $\mathbf{A}_v$, $\mathbf{x}_v$, and $\mathbf{b}_v$ refer to a FE system (see Equation (3.21)) used in the virtual simulation. The *constraint Jacobian* matrix, denoted as $\mathbf{J}_v$, defines how the constraints are applied to the degrees of freedom (DOFs) in the mechanical state. The Lagrange multiplier $\boldsymbol{\lambda}$ denotes the constraint force, and $\boldsymbol{\delta}_n$ signifies the constraint violation. An efficient so-

lution for the constrained system is to formulate the Schur-complement $\mathbf{W} = \sum \mathbf{J}_v \mathbf{A}_v^{-1} \mathbf{J}_v{}^{\mathrm{T}}$ and solve the problem in the constraint space. An important aspect of our virtual problem is that we use a linear finite element model for the static simulation. Thus the matrix $\mathbf{A}_v$ remains invariant. For small-scale problems, it is efficient to pre-compute the inverse of $\mathbf{A}_v$ and store the result in CPU memory to compute the Schur-complement Saupin et al. (2008a). However, for large-scale problems, the storage cost of the dense matrix $\mathbf{A}_v^{-1}$ with dimension $n \times n$ becomes prohibitively high.

In Chapter 4, we proposed a method that efficiently addresses the Schur-complement in large-scale constrained problems using the "*isolating mechanical DOFs* (isodof)" method Zeng et al. (2022). This approach exploits the sparsity in $\mathbf{J}$ and performs a reduced and parallelized computation on the GPU. Since each constraint in our virtual simulation affects a single vertex on the boundary geometry, the *constraint Jacobian* $\mathbf{J}_v$ is very sparse. The factorization of $\mathbf{A}_v$ can be precomputed and stored on the CPU with reasonable space costs. Furthermore, we can utilize the "*reuse isodof*" method from Zeng et al. (2022) to avoid recomputing previous constraints. Consequently, the constraint resolution is extremely fast in virtual snapping. More details on the "*isodof*" and "*reuse isodof*" methods can be found in Zeng et al. (2022).

After building the compliance matrix, the system is solved within the constraint space. We use the Projected Gauss-Seidel (PGS) method to solve the unknown forces $\boldsymbol{\lambda}$. In order to improve the stability in the constraint resolution, a strategy in constraint-based techniques is to set a maximum value for the snapping forces. In the PGS iterations, the constraint forces will be limited as the maximum value while exceeding the threshold. With the maximum value, the limited snapping forces may not guarantee to move the vertices onto the cutting path within a time step. Once $\boldsymbol{\lambda}$ is solved, a corrective motion is processed to integrate the motion in the virtual system. Similar to the first order linearization in equation (3.20), the virtual system actually performs only the first Newton-Raphson iteration within a time step, therefore it cannot guarantee the equilibrium at the end of time step. However, the dynamic virtual simulation is able to overcome this drawback since the constraint will be continuously applied on the boundary vertices in successive time steps. Therefore, each virtual simulation step can be considered as a Newton-Raphson iteration, while the progressive time steps continue approaching the equilibrium of the dynamic system.

5.1.4 State and topology correction

After the snapping process in the virtual simulation, the displacement result is applied back to the reference state of the actual simulation. To prevent generating large inter-

nal forces after snapping (particularly in stiff materials), we apply a correction to the deformed position by computing the inverted strain-stress with displacements in the reference shape. In the actual FE system, the displacement of vertices in the reference shape requires updating the reference stiffness matrices $\mathbf{K}_{e0}$. With the elastic equilibrium in the reference state, all the mesh vertices undergo more or less displacement in the actual object, requiring an update to the reference shape matrix for all elements. This update operation can be computationally expensive in large-scale systems. Nevertheless, as illustrated in Figure 5.8, not all the vertices are significantly displaced due to snapping. Thus, setting a user-defined snapping displacement threshold and filtering out the vertices that have undergone negligible displacement can significantly reduce the computational cost. We list three issues may reduce the mesh quality after cutting, within each time step:

1. The linearization with first Newton-Raphson iteration: equilibrium may not be reached within time step.

2. The maximum force that limits the snapping forces.

3. The snapping threshold that limits the displacements.

As discussed in Section 5.1.3, the virtual simulation keeps approaching the equilibrium in successive time steps. Consequently, the previous cut part remains possibility of evolution. We can efficiently deal with this change with the snapping threshold, since the vertices will be snapped only with sufficient displacement. In applications, the snapping threshold will effect both the performance and the quality of the cutting surface, which will be evaluated in Section 5.1.6.4.

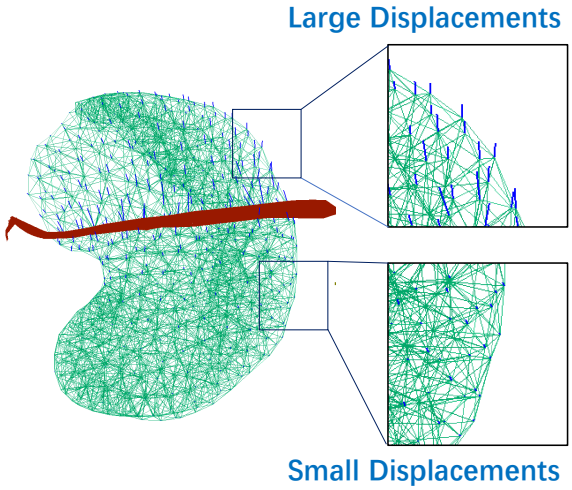


Figure 5.8: In the virtual simulation, the constraints that snaps the boundary to the cutting path (red plane) will effect all the vertices, leading to different scales of displacements (blue lines).

The last step of our cutting method is to process the topology change. From the perspective of topology, our method splits the mesh along its existing surfaces (the boundary surface defined in Section 5.1.2). In the cutting progress, the mesh of the boundary surface (containing triangles, edges, and vertices) is duplicated as two. The two sides of neighbor element of the boundary surface can be marked as side **A**/**B**. One side **A** is cho-

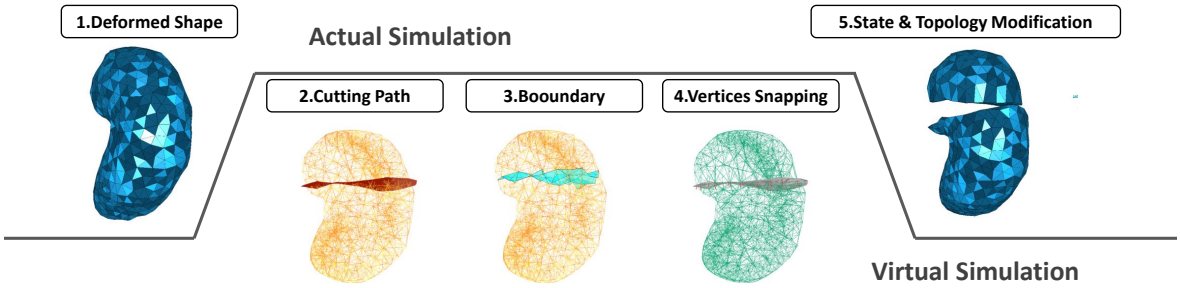


Figure 5.9: Workflow of the cutting method in 3D simulation. (View in SOFA Framework)

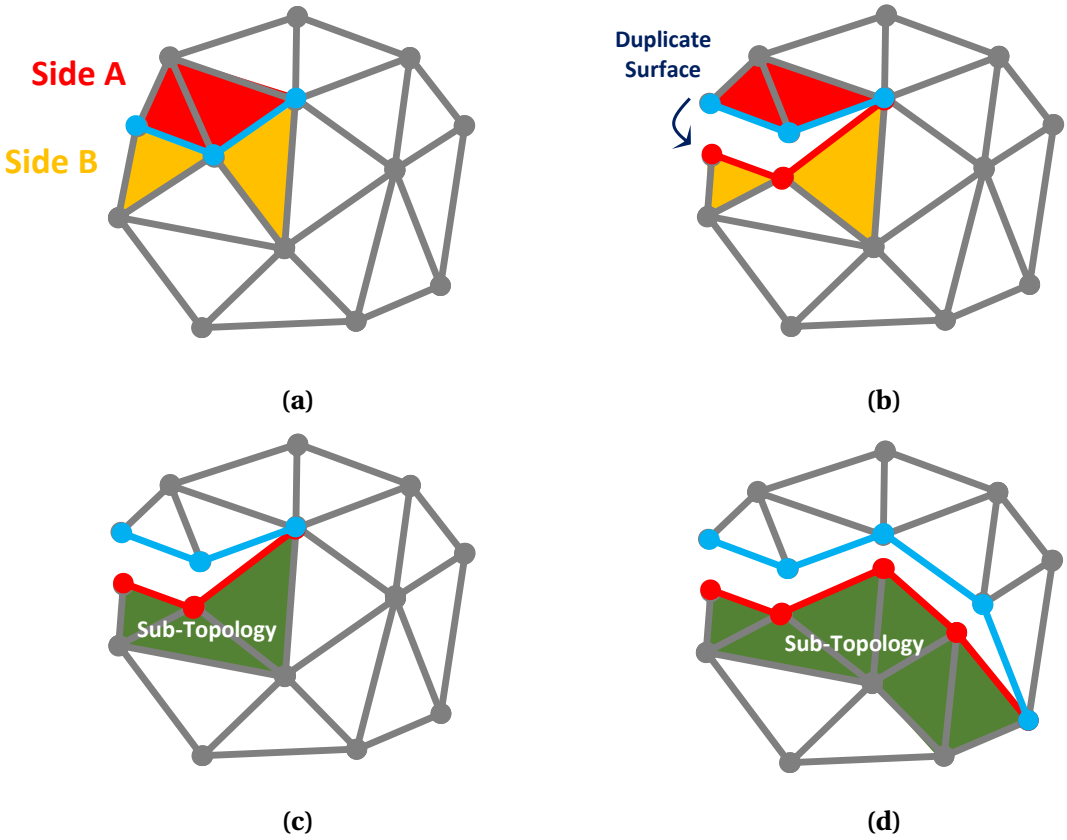


Figure 5.10: The method to split the mesh: (a) the boundary surface has two sides of neighbor elements. (b) we choose one side to contain the original boundary, and the other side to contain the duplicated boundary. (c) the criterion for selecting the side B is to choose neighbor elements from the sub-topology. (d) using the sub-topology helps to guarantee the topology consistency.

sen to contain the original boundary, and the other side **B** contains the copied boundary. The definition of sides should keep consistency in progressive cutting, such that the elements remains on the same side. The sub-topology in Section 5.1.2 can be used to define the sides. Since the boundary surface is the boundary of the sub-topology, choosing the elements in the sub-topology as the side that contains the copied boundary, guaranteeing the consistency of the side (see the illustration in Figure 5.10).

An important advantage of our vertices snapping method is that there is no addition or remove of original tetrahedron elements. The increases of new vertices, edges, and triangles are minimized since the mesh is split from the existing surface in the mesh. After the cutting progress, the numerical solvers in the actual simulation requires update with the topology change. How to efficiently update the solver is an issue and will be discussed in the next section.

5.1.5 Efficient numerical solver with topology change

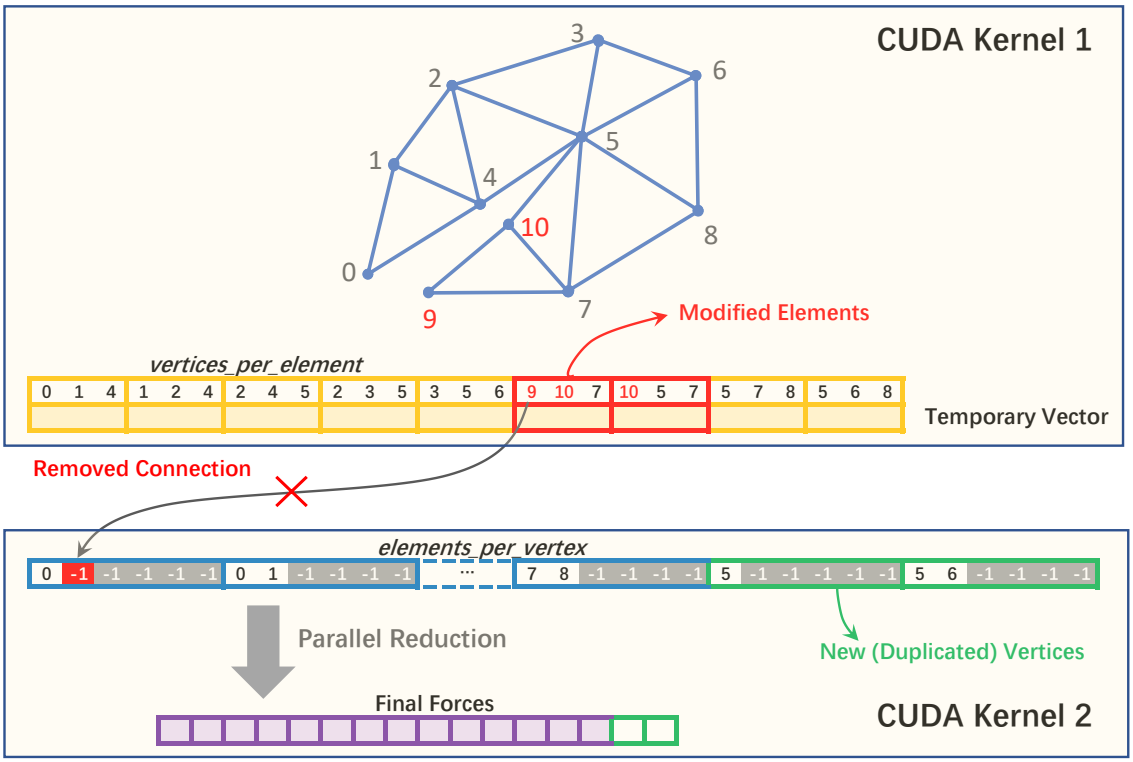


Figure 5.11: Modified matrix-free iterative solver: with the topology change information, we propose efficient modification to the precomputed GPU-based vectors.

The cutting process described in the current section produces vertex displacements in the reference shape and modifies the topology. As discussed in Section 3.1.4, the matrix-free method is an efficient approach for solving the dynamic system in the actual simu-

lation, but it necessitates a precomputed GPU-based structure. In Allard et al. (2011), the GPU structures *vertices_per_elements* and *elements_per_vertex* contain the information of neighbor vertices in elements, and neighbor elements in vertices respectively. This information can be directly obtained from the mesh topology, and is precomputed and stored in vectors that will be transferred to the GPU only once. The information remains valid as long as no topological change occurs in the simulation. In contrast, cutting operations will easily invalidate the information. But recomputing the GPU structures requires large computation cost. Therefore, efficiently updating the GPU structure is an important issue to make sure the real-time performance.

Based on our cutting method, we propose an approach that efficiently update the such structures without redoing the pre-computation. Since the cutting method splits the mesh along its existing surfaces, the topology change results in following modifications:

1. Duplicated boundary mesh: new triangles, edges, and vertices.

2. Modified elements: the indices of contained structure (triangles, edges, vertices) should be updated with the new (duplicated) indices.

Based on these changes, we propose modifications to the matrix-free iterative solver, as illustrated in Figure 5.11: The mapping *vertices_per_elements* should update the vertex indices for the modified elements. Consequently, some previous connections exist no longer, and we set the removed neighbor element in *elements per vertex* as -1. The structure for the new vertices is added at the end of *elements per vertex*, storing the indices of their neighbor elements. And the contributions will be finally accumulated into the final force that is augmented with new vertices. The modification information can be obtained from the topology change operations in Section 5.1.4. In progressive simulations, the local cutting usually impacts limited elements. Therefore updating the structures requires relatively small cost, compared to recomputing the structures. We evaluate the performance of our modified matrix-free solver in Section 5.1.6.4.

5.1.6 Evaluation of cutting method

In this section we evaluate the behavior of our cutting method and the computing performance of the numerical solver. The simulation tests are conducted in the open-source SOFA framework with a CPU AMD@ Ryzen 9 5950X 16-Core at 3.40GHz with 32GB RAM, and a GPU GeForce RTX 3080 10GB.

The deformable meshes are modeled with the co-rotational formulation, while the cutting method should be compatible with hyperelastic materials (as discussed in Section

5.1). The numerical solver is based on the matrix-free solver Allard et al. (2011) which is a specific implementation for co-rotational models.

5.1.6.1 Evaluation of mesh quality

In this section, we evaluate the mesh quality of our cutting method. To compare with the related work, we simulate a similar cutting to the example in Paulus et al. (2015b). As illustrated in Figure 5.12, we set a similar cutting path to progressively cut a deformable beam into two pieces. As our method is compatible in simulations with different systems scales, the shape is meshed with various discretizations. The number of elements and the initial number of vertices in each experiment set can be found in Table 5.1. Using existing manifold to split the mesh, our propose method is not suitable for meshes with extremely simple topology, as the geometric operations risk of selecting the object surface as splitting boundary. Therefore, instead of reproducing the exact same example in Paulus et al. (2015b) where the shortest beam edge only contains two layers of elements, we test our method with larger systems (765 – 9471 vertices) to guarantee correct geometric splitting. The mesh refinement methods are more suitable for the problems with extremely simple topology.

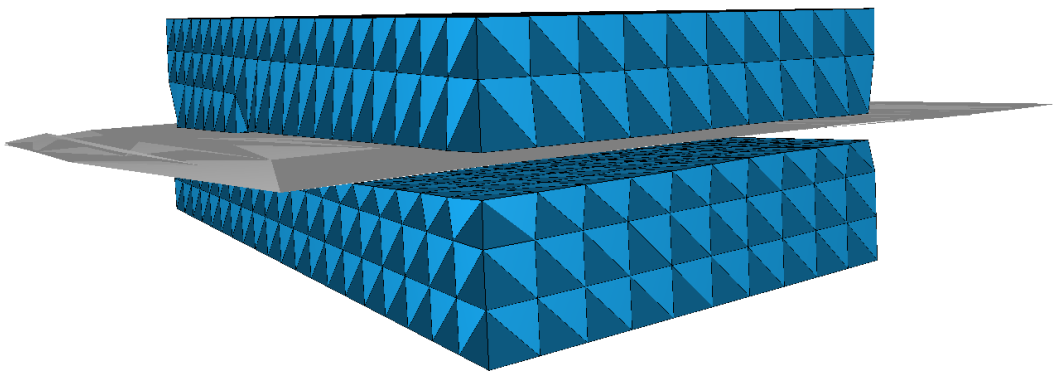


Figure 5.12: A deformed beam with size of 1:2:4 is progressively cut.

The scenario in Figure 5.12 takes approximately 60 time steps to perform a progressive cut from the beginning to the full cut. For different mesh discretizations, the evolution of the vertices number is illustrated in Figure 5.13. With the initial vertices number from 765 to 9471, our proposed method increases from 9.11% to 20.00% of the initial vertices number. According to the report in Paulus et al. (2015b), the methods Paulus et al. (2015b)

Example		Actual	Virtual					Actual			All
		Point	Cutting	Boundary	Vertices Snapping			Update		CG	
Vertices	Elements	Cloud	Path	Manifold	Delasus	PGS	Corr	State+Topo	Stiffness+Map		
765	3072	0.47	0.58	0.71	0.70	0.64	0.60	0.26	0.43	0.84	7.01
1386	6000	0.72	0.58	1.22	1.01	1.63	0.89	0.47	0.70	0.66	10.03
2275	10368	0.92	0.58	2.00	1.63	3.71	1.38	0.68	0.85	0.90	15.46
3480	16464	1.31	0.59	3.13	2.61	7.15	2.02	1.19	1.42	0.75	24.06
5049	24576	1.50	0.60	4.48	3.77	12.36	2.83	1.88	1.60	0.78	34.80
7030	34992	2.18	0.60	6.34	5.70	21.23	4.38	2.71	2.24	1.03	53.22
9471	48000	2.97	0.60	8.66	8.44	33.66	6.11	4.90	2.56	1.07	77.59

Table 5.1: Performance (in ms) of cutting a deformable beam (see Figure 5.12) with different discretizations. Different steps within a time step: **Actual**: process in the actual simulation; **Virtual**: process in the virtual simulation; **Point Cloud**: generating the point cloud; **Cutting Path**: generating the cutting path using MLS and fast triangulation; **Boundary Manifold**: searching for the boundary surface; **Delasus**: building the compliance matrix in the constrained system; **PGS**: Projected Gauss-Seidel method in the constraint resolution; **Corr**: constraint correction in the virtual simulation; **Update State+Topo**: modifying the state and topology in the actual simulation after snapping; **Update Stiffness+Map** updating the stiffness matrix (snapping threshold = 0.01) and the GPU-based mapping for the matrix-free solver; **CG**: matrix-free Conjugate Gradient solver in the actual simulation.

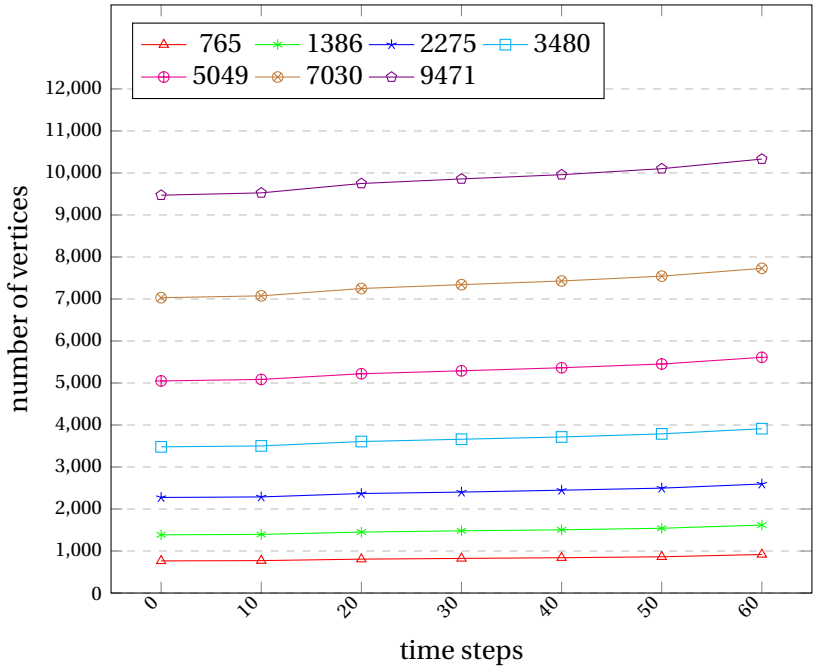


Figure 5.13: During progressive cutting, the evolution of vertices number in the beam to cut. The deformable beam is discretisized from 765 vertices to 9471 vertices (initial number of vertices).

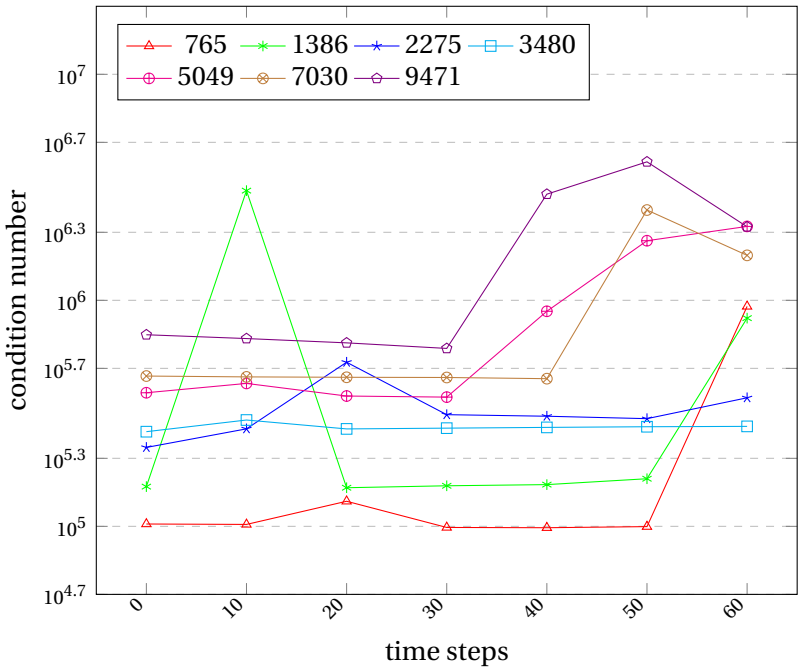


Figure 5.14: During progressive cutting, the evolution of condition number of the system matrix of the beam to cut. The experiment set is the same to Figure 5.13.

and Bielser et al. (1999) generate more than 200% and 400% of the initial vertices number, respectively. The result is consistent with the discussion in Section 5.1.4: Our method actually splits the mesh from its existing surface, minimizing the generation of new vertices. On the other hand, within the same experiments, we evaluate the mesh quality by measuring the condition number of the system matrix of the object to cut. The condition number gives an indication how difficult to solve a numerical system. A higher condition number implies requiring more iterations to address the problem by iterative solvers. Figure 5.14 illustrates the evolution of the condition number of the system. With different mesh discretizations, the condition numbers are increasing to 1.13× to 20.40× of their original values. While, according to the report in Paulus et al. (2015b), the method Bielser et al. (1999) quickly leads to ill-conditioned systems with $10^7\times$ of their original values. The best results of the method Paulus et al. (2015b) are $10^4\times$ of the initial value and less than 10× of the initial value with different thresholds ϵ, which is a parameter defined in their manuscript for snapping close boundary. Our result is comparable to the best result in Paulus et al. (2015b). As discussed in Section 5.1.3, the virtual simulation searches for an equilibrium between the snapping forces and the internal stiffness forces. This actually optimizes the redistribution of element sizes and the numerical quality of mesh.

5.1.6.2 Evaluation of boundary surface

In this section, we evaluate the computing performance of our cutting method and the advanced numerical solver. While keep using the experiment set in the previous section, we add more discretization sets to show the performance in high resolution cases.

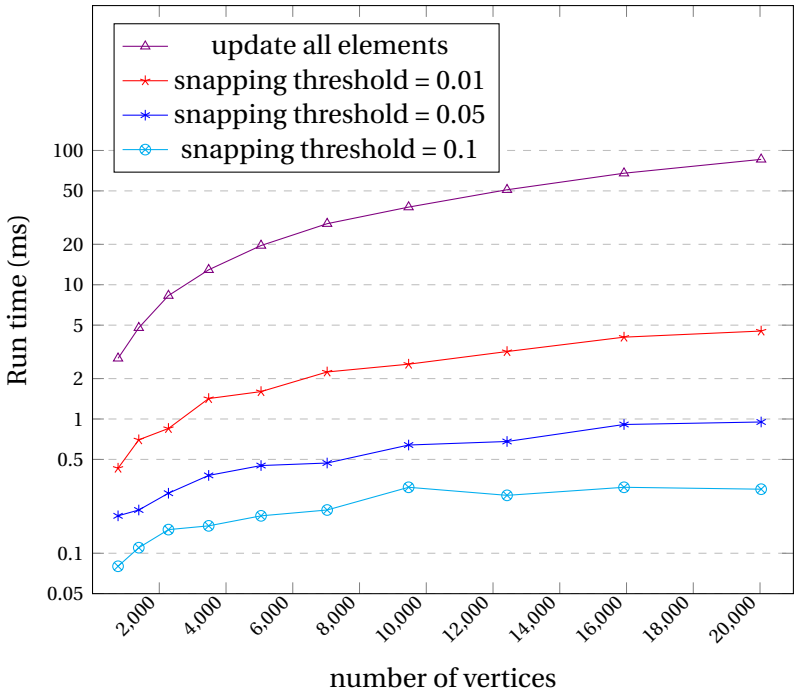
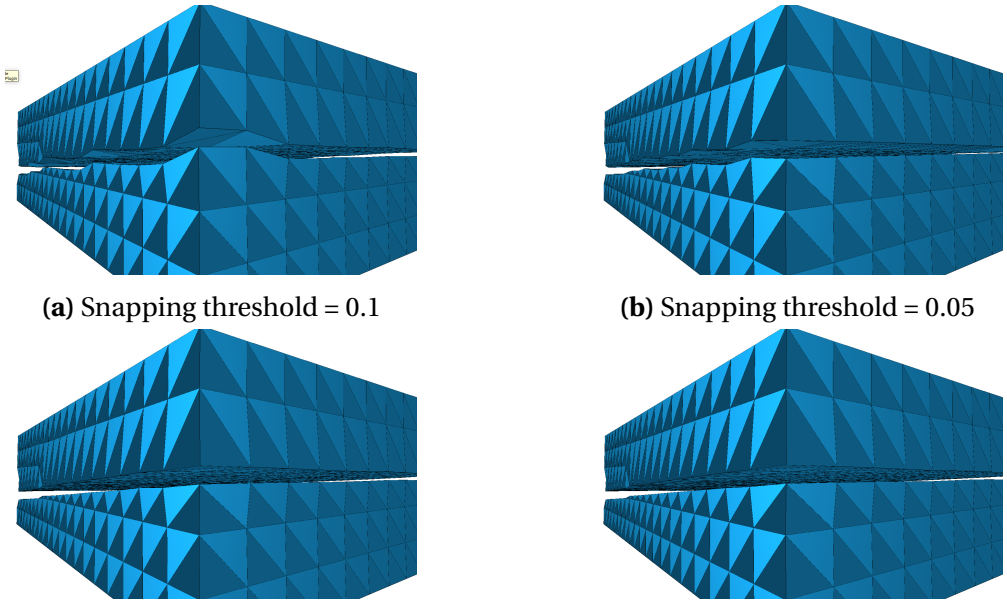


Figure 5.15: Performance (in ms) in updating initial stiffness matrices with different thresholds of the snapping displacement.

In Section 5.1.4, we have discussed the effect of setting the maximum snapping displacement. In Figure 5.15, we measure the effect of the snapping threshold to the computing performance in updating the initial stiffness matrices. While updating all the elements becomes prohibitive in large-scale problems, controlling the snapping threshold can efficiently reduce the computation cost. A higher threshold results in more efficient computing, but suffers from a worse cutting surface. In Figure 5.16 we show the surface quality with different thresholds. In this scenario, setting the snapping threshold as 0.01 is a good compromise since it ensures both the good cutting surface and the real-time performance.

5.1.6.3 Evaluation of numerical solver

In this section, we evaluate the method of updating the matrix-free solver presented in Section 5.1.5. In Figure 5.17 we compare the performance between updating the full GPU-based mappings *vertices_per_elements* and *elements per vertex*. While process the



(a) Snapping threshold = 0.1 **(b)** Snapping threshold = 0.05

(c) Snapping threshold = 0.01 **(d)** Snap all vertices

Figure 5.16: The cutting surface with different snapping displacement. The beam size is $2 \times 4 \times 8$.

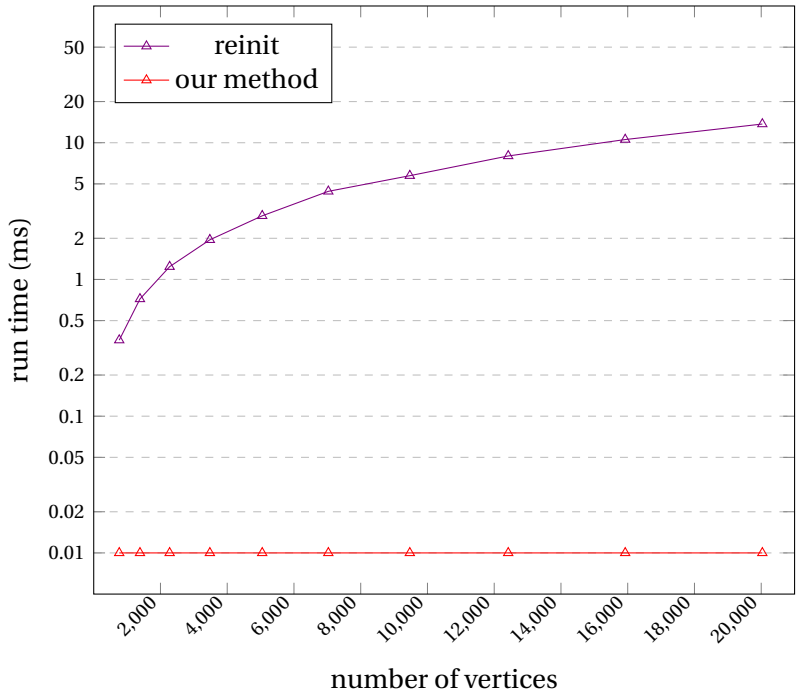


Figure 5.17: Performance (in ms) of updating GPU-based structure. The values less than 0.01 are set as 0.01 due to the measure precision.

full vectors becomes costly as the system scale increases, our method remains highly efficient, requiring negligible computing cost to update the GPU-based structures.

5.1.6.4 Evaluation of computing performance

To have an overview of the performance of our cutting method and numerical solver, we present in Table 5.1 the detail computing costs in different steps within a time step. The cutting method is generally very efficient, while the iterative solver PGS in virtual constraint resolution becomes dominant as the system scale increases. This is because the cutting path crosses most of the elements in the scenario in Figure 5.12, generating a large number of snapping constraints. However, this is not always the fact. In many other cases where the cutting path interacts with local areas, the PGS will be more efficient with limited constraints.

5.2 Applications

5.2.1 Progressive cutting while the object is deforming

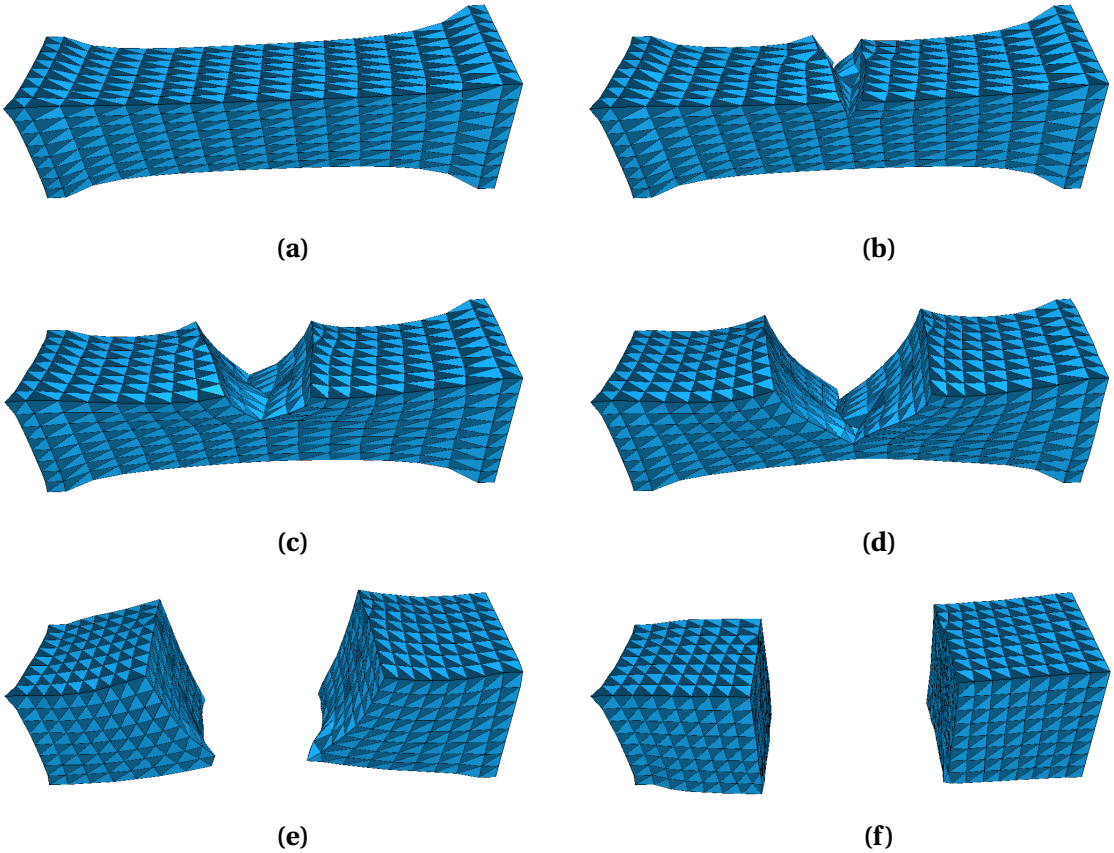
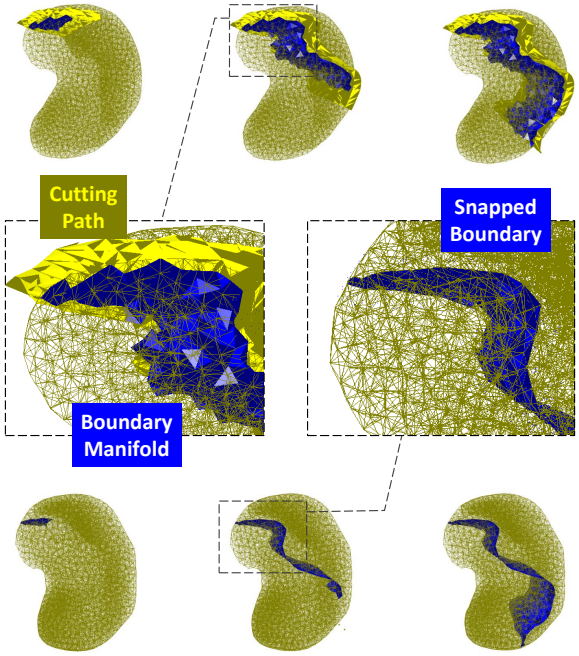


Figure 5.18: Modeling a cut while deforming.

In order to visualize that our method is able to conserve the system energy, in Figure 5.18 we model a cut while the object is deforming. A soft beam is initially stretching while

the both ends are fixed (Figure 5.18a). As discussed in Section 5.1, our method address the problem of energy conservation by snapping the vertices in the reference shape instead of the deformed shape. The displacements in the reference shape will not invalidate the internal and external forces. As illustrated in Figures from 5.18b to 5.18e, the elastic internal forces keep tensions to pull the shape back to its initial (reference) shape during the progressive cutting. After the beam is fully cut, the system is able to be restored as the reference shape (see Figure 5.18f).

5.2.2 Arbitrary cutting path and irregular mesh



(a) In the virtual simulation. **Top**: our method searches for the boundary manifold mesh (blue) in proximity to the cutting path (yellow). **Bottom**: after the vertices snapping, the manifold is snapped onto the cutting path, generating smooth boundary for splitting the mesh.

(b) The mesh is split in the actual simulation (View in SOFA Framework).

Figure 5.19: Modeling a cut with arbitrary cutting path.

Our method is compatible with arbitrary cutting path and irregular object shape. In Figure 5.19 we model a progressive cut where the cutting instrument is handled by a user.

Our geometric operations in Section 5.1.2 is able to efficiently find the boundary manifold mesh in proximity of the unscheduled cutting path. In the virtual simulation, the manifold mesh is snapped onto the cutting path and used as splitting boundary to divide the mesh into two parts. Consequently our proposed method results in a cutting with smooth splitting surface, while preventing generating new elements.

5.3 Conclusion

In this chapter we presented a novel cutting method based on the vertices-snapping strategy. While inheriting the advantages smooth cutting surface and limited new generalized vertices, our method is able to optimize the mesh quality by simulating a virtual FE model. To have correct splitting of the mesh, we implemented a set of efficient geometry operations to search for the boundary manifold. Moreover, according to the properties in the cutting method, we propose a modification to the matrix-free iterative solver to efficiently deal with the topological change. Finally, all these methods result in progressive cutting with high computing efficiency, good mesh quality, and smooth cutting surface. Our proposed method holds great potential for extending our previous works. We will discuss the details of these potential works in the next chapter.

CHAPTER 6

PERSPECTIVE AND CONCLUSION

6.1 Perspective: implicit constraint resolution

In Chapter 4, we have discussed the constraint resolution with a Gauss-Seidel solver. The current integration loop (see Algorithm 3) reveals a problem where the initial evaluation of constraint directions is not guaranteed to be consistent with the state at the end of the time step. To address this issue, we introduce a novel implicit scheme for constraint resolution in this section. Instead of using the standard motion correction scheme, we propose an iterative method where constraint forces are corrected in Newton iterations. This scheme allows for recursive updating of constraint directions, leading to more accurate contact and friction response.

6.1.1 Recursive corrective motion

For both discrete and continuous collision detection algorithms, the constraints will be linearized only once each time step. Nevertheless, it cannot be guaranteed that the initial evaluation of the constraint directions is always consistent with the state at the end of the time step. Figure 6.1 gives an example: in a grasping scenario, while acting a moving/rotation, the constraint directions undergo large deviations from the initial guess.

To address this problem, we propose a recursive motion correction scheme: Instead of using a single corrective motion as in Equation (4.19), a iterative correction is applied:

$$\Delta\dot{\mathbf{q}}^{t+h} = \Delta\dot{\mathbf{q}}^{\text{free}} + (\Delta\dot{\mathbf{q}}_1^{\text{cor}} + \Delta\dot{\mathbf{q}}_2^{\text{cor}} + \cdots + \Delta\dot{\mathbf{q}}_n^{\text{cor}}) \tag{6.1}$$

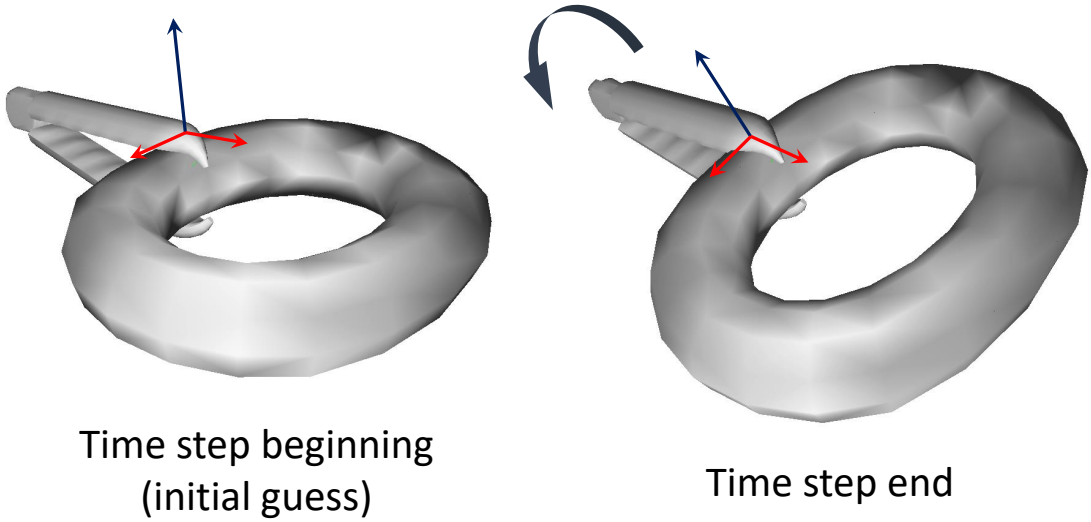


Figure 6.1: Grasping a solid while applying a fast rotation will case a large deviation of constraint directions from the beginning to the end of a time step. Such a deviation may cause inaccurate contact responses, instability of contact forces, and eventually failed simulations.

where n interactions are performed, and the corrective motion of a given iteration k is computed as following:

$$\Delta\dot{\mathbf{q}}_k^{\text{cor}} = h\mathbf{A}_k^{-1}\mathbf{J}_k{}^{\mathrm{T}}\boldsymbol{\lambda}_k \tag{6.2}$$

where $\mathbf{A}_k$ and $\mathbf{J}_k$ are reevaluated in each iteration according to the current mechanical state ($\mathbf{q}_k$). For the integration between two successive iterations k and $k+1$ in Equation (6.1), we have:

$$\mathbf{q}_{k+1} - \mathbf{q}_k = (\mathbf{q}^{\text{free}} + h\Delta\dot{\mathbf{q}}_{k+1}) - (\mathbf{q}^{\text{free}} + h\Delta\dot{\mathbf{q}}_k) = h\Delta\dot{\mathbf{q}}_{k+1}^{\text{cor}} \tag{6.3}$$

Such a recursive scheme is a Newton-Raphson method that provides a more accurate correction. However, performing such a scheme requires repeating the compliance assembly, the constraint resolution, the corrective motion, and the time integration in recursive iterations. These additional processes multiply the computational cost, making the system resolution very inefficient.

6.1.2 Efficient update of constraint tensors

In this section, we propose a strategy that is able to efficiently process the iterations in Equation (6.1). In the recursive motion correction scheme (Equation (6.1)), the system matrix $\mathbf{A}$ is actually the mass matrix $\mathbf{M}$ for a rigid object. Therefore $\mathbf{A}_k = \mathbf{A}_0 = \mathbf{M}$, $k \in \mathbb{R}^n$.

Algorithme 8 : Recursive correction scheme with standard rebuilding of the constraint matrices

foreach $k \in Newton\_iterations$ **do**
  Linearize contact constraints with the proximity positions $\mathbf{p}_k$, then update $\mathbf{J}_k$ with the constraint directions.
  Update compliance matrix: $\mathbf{W}_k = \sum \mathbf{J}_k \mathbf{A}_k^{-1} {\mathbf{J}_k}^{\mathrm{T}}$
  Compute constraint forces within local PGS: $\boldsymbol{\lambda} = \mathbf{W}^{-1}\boldsymbol{\delta}$
  Compute corrective motion: $\Delta\dot{\mathbf{q}}_k^{\mathrm{cor}} = h\mathbf{A}_k^{-1}{\mathbf{J}_k}^{\mathrm{T}}\boldsymbol{\lambda}_k$
  Integrate corrective motion: $\dot{\mathbf{q}}_k = \dot{\mathbf{q}}_{k-1} + \Delta\dot{\mathbf{q}}_k^{\mathrm{cor}}$, $\mathbf{q}_k = \mathbf{q}_0 + h\dot{\mathbf{q}}_k$
  Update the proximity positions: $\mathbf{p}_k = \mathscr{G}(\mathbf{q}_k)$
end

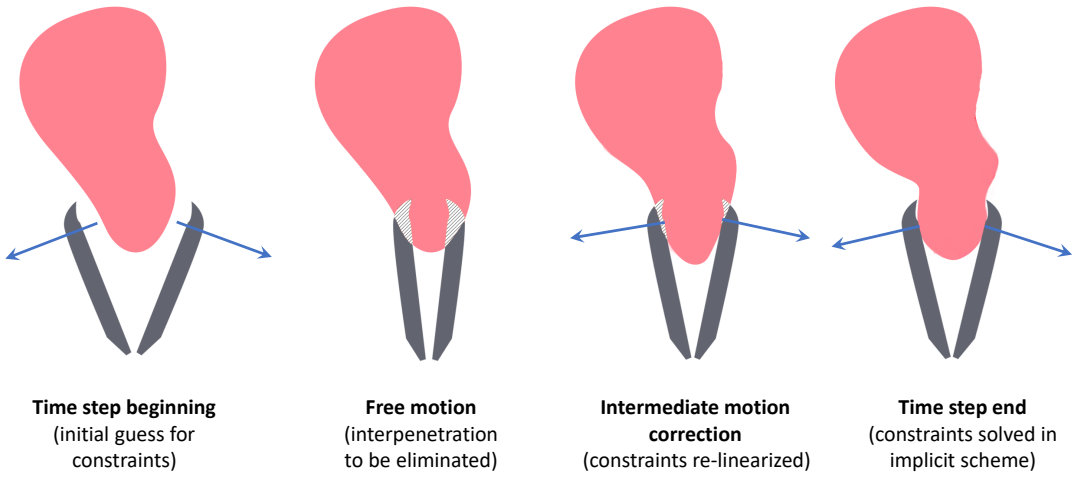


Figure 6.2: Our method solves the constraints in an implicit scheme: The constraints are linearized at the beginning of each time step using discrete collision detection. Contact and friction responses are computed to eliminate the interpenetration between solids in the free motion. The constraint resolution is performed in a recursive corrective motion scheme, where contact forces are computed in each iteration by a local solver. Using the contact forces, the boundary conditions of colliding solids and the constraint matrices are updated in the next iteration to compute new motion corrections. In this way, the constraint directions are re-linearized recursively until the interpenetration is eliminated.

On the other hand, the corrective motion usually causes small deformation in motion corrections. Relying on this hypothesis, we have an approximation for $\mathbf{A}_k$ in the iterations: $\mathbf{A}_k = \mathbf{A}_0$, $k \in \mathbb{R}^n$. Following Equation (6.1) and (6.2), we have:

$$\Delta\dot{\mathbf{q}}^{t+h} = \Delta\dot{\mathbf{q}}^{\mathrm{free}} + h\mathbf{A}^{-1}({\mathbf{J}_0}^{\mathrm{T}}\boldsymbol{\lambda}_0 + {\mathbf{J}_1}^{\mathrm{T}}\boldsymbol{\lambda}_1 + \cdot{\mathbf{J}_n}^{\mathrm{T}}\boldsymbol{\lambda}_n) \tag{6.4}$$

We recall the definition of the contact Jacobian $\mathbf{J}$. For a given constraint with a direc-

tion $\vec{\mathbf{c}}$, the contribution in $\mathbf{J}$ is expressed as:

$$\mathbf{J}_{(\vec{\mathbf{c}})} = \frac{\partial \mathscr{H}(\mathbf{q})}{\partial \mathbf{q}} = \frac{\partial (\vec{\mathbf{c}}\, \mathscr{G}(\mathbf{q}))}{\partial \mathbf{q}} = \vec{\mathbf{c}} \frac{\partial \mathscr{G}(\mathbf{q})}{\partial \mathbf{q}} \tag{6.5}$$

as in practice, the constraint direction $\vec{\mathbf{c}}$ is independent of the mechanical state after a constraint linearization.

With Equation (6.5), we propose to assembly a Jacobian matrix $\mathbf{H}$ for the Jacobian of the geometric mapping $\frac{\partial \mathscr{G}(\mathbf{q})}{\partial \mathbf{q}}$, and a block-diagonal matrix $\mathbf{C}$ to store the constraint directions:

$$\mathbf{C} = \begin{bmatrix} \vec{\mathbf{c}}_1 & & & \\ & \vec{\mathbf{c}}_2 & & \\ & & \ddots & \\ & & & \vec{\mathbf{c}}_c \end{bmatrix} \tag{6.6}$$

where $\vec{\mathbf{c}} = \begin{bmatrix} \vec{\mathbf{n}} & \vec{\mathbf{f}}_{(1)} & \vec{\mathbf{f}}_{(2)} \end{bmatrix}$ groups the normal and frictional constraints on a shared proximity point. The relation between the two matrices can be actually expressed by a matrix-matrix multiplication:

$$\mathbf{J} = \mathbf{C}\mathbf{H} \tag{6.7}$$

With Equation (6.7), a standard compliance assembly (Equation (4.14)) can be then reformulated as:

$$\mathbf{W} = \sum \mathbf{J}\mathbf{A}^{-1}\mathbf{J}^{\mathrm{T}} = \sum \mathbf{C} \underbrace{\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^{\mathrm{T}}}_{\mathbf{W_H}} \mathbf{C}^{\mathrm{T}} \tag{6.8}$$

where $\mathbf{W}$ can be built with $\mathbf{J}$ and $\mathbf{W_H}$. The geometrical mapping $\mathscr{G}(\mathbf{q})$ usually undergoes a slight change in the recursive corrective motion scheme. Based on this hypothesis, $\mathbf{H}$ and $\mathbf{W_H}$ are considered invariant during a time step.

Figure 4.1 illustrates that the linearization of contact constraints will based on the proximity positions $\mathbf{p}$. With a Taylor expansion, $\mathbf{p}$ in the recursive motion correction scheme can be as follows:

$$\mathbf{p}_{k+1} = \mathscr{G}(\mathbf{q}_{k+1}) \approx \mathscr{G}(\mathbf{q}_k) + \frac{\partial \mathscr{G}(\mathbf{q})}{\partial \mathbf{q}} (\mathbf{q}_{k+1} - \mathbf{q}_k) \tag{6.9}$$

where k and $k+1$ represents two successive iterations in the recursive motion correction scheme. With Equation (6.3) and $\frac{\partial \mathscr{G}(\mathbf{q})}{\partial \mathbf{q}}$ linearized as $\mathbf{H}$, we continue the development in Equation (6.9):

$$\mathbf{p}_{k+1} \approx \mathscr{G}(\mathbf{q}_k) + \frac{\partial \mathscr{G}(\mathbf{q})}{\partial \mathbf{q}} (\mathbf{q}_{k+1} - \mathbf{q}_k) = \mathbf{p}_k + h\mathbf{H}\Delta\dot{\mathbf{q}}^{\mathrm{cor}}_{k+1} \tag{6.10}$$

Combining Equation (6.2), (6.10), (6.7) and (6.8), we have:

$$\begin{aligned}\mathbf{p}_{k+1} &\approx \mathbf{p}_k + h\mathbf{H}\Delta\dot{\mathbf{q}}^{\text{cor}}_{k+1} \\ &= \mathbf{p}_k + h\mathbf{H}(h\mathbf{A}^{-1}{\mathbf{J}_k}^{\mathrm{T}}\boldsymbol{\lambda}_k) \\ &= \mathbf{p}_k + h^2\mathbf{H}\mathbf{A}^{-1}\mathbf{H}^{\mathrm{T}}{\mathbf{C}_k}^{\mathrm{T}}\boldsymbol{\lambda}_k \\ &= \mathbf{p}_k + h^2\mathbf{W}_{\mathbf{H}}{\mathbf{C}_k}^{\mathrm{T}}\boldsymbol{\lambda}_k\end{aligned} \tag{6.11}$$

Now with Equation (6.8) and (6.11), once $\mathbf{W}_{\mathbf{H}}$ is built, a recursive scheme can be performed in Algorithm 9.

Algorithme 9 : Recursive correction scheme with fast update of the constraint matrices

```
foreach k ∈ Newton_iterations do
    constraint_linearization(p_k).
    Update C_k with linearized constraint directions.
    Update compliance matrix: W_k = Σ C_k W_H C_k^T
    Compute constraint forces within local PGS: λ_k = W_k^{-1} δ
    Update the proximity positions: p_k = p_{k-1} + h^2 W_H C_k^T λ_k
end
```

Compared with the strategy of rebuilding the matrices with $\mathbf{J}$, the computation is greatly simplified in the new scheme. Instead of inverting the system matrix $\mathbf{A}$, the new scheme only needs matrix-matrix multiplications to update the compliance matrix $\mathbf{W}$, and a matrix-vector multiplication to update the boundary state (proximity positions $\mathbf{p}$). As the basic operations in linear algebraic, the matrix multiplications have highly efficient implementations, especially with parallelization on the GPU.

6.1.3 Evaluation of performance in implicit constraint solver

In this section, we evaluate the computation cost of the method presented in Section 6.1.2 that provides a fast update of constraint matrices in the recursive correction scheme proposed in Section 6.1.1. The simulation tests are conducted in the open-source SOFA framework with a CPU AMD@ Ryzen 9 5950X 16-Core at 3.40GHz with 32GB RAM and a GPU GeForce RTX 3080 10GB.

We keep using the precondition-based technique in Courtecuisse et al. (2014) to assemble the compliance matrix $\mathbf{W}$. As illustrated in Algorithm 9, a recursive corrective motion scheme helps to improve the constraint correction in an iterative way, which performs Newton-Rapshon iterations. Within each Newton iteration, a local **PGS** is performed to solve the complementarity problem in contact and friction responses, giving

DOFs	Cst.	Method	Build $\mathbf{W_H}$	Update $\mathbf{W}$	PGS	Corr.	Newton ite.	Total	Update
18003	308.19	rebuilding		19.89 ms	3.76 ms	0.82 ms	24.47 ms	122.39 ms	84.60 %
		fast update	19.26 ms	0.59 ms		0.08 ms	4.43 ms	41.45 ms	15.09 %
24066	365.97	rebuilding		32.21 ms	4.84 ms	1.19 ms	38.24 ms	191.21 ms	87.34 %
		fast update	31.27 ms	0.78 ms		0.07 ms	5.69 ms	59.72 ms	14.94 %
30033	410.58	rebuilding		45.47 ms	5.87 ms	1.47 ms	52.81 ms	264.07 ms	88.88 %
		fast update	45.16 ms	0.98 ms		0.06 ms	6.91 ms	79.73 ms	15.04 %
36069	482.25	rebuilding		64.32 ms	7.91 ms	1.87 ms	74.10 ms	370.52 ms	89.32 %
		fast update	63.73 ms	1.29 ms		0.07 ms	9.27 ms	110.10 ms	14.67 %

Table 6.1: Comparison of performance between rebuilding the matrices and the fast update scheme for different numbers of *mechanical DOFs* (*DOFs*) and constraints (Cst.): For the fast update scheme, building the mapping compliance matrix (Build $\mathbf{W_H}$) is performed once each time step. The following processes are performed once in each Newton iteration: updating the compliance matrix (Rebuild $\mathbf{W}$), processing the local PGS (PGS), and computing the corrective motion (Corr.). Then the cost of each Newton iteration (Newton ite.) and the whole recursive corrective motion scheme (Total) are compared. Finally, the proportion of the update constraint matrix process in the Newton iteration (Update) is illustrated.

temporary contact forces. The matrix operations in Algorithm 9 are realized with typical GPU-based implementation in **cuBLAS** and **cuSPARSE** libraries.

Such contact forces are used to compute the position of the surface elements, which are further used to define the constraint directions in the next iteration. In Table 6.1, we compare the computational cost between the rebuilding scheme and the fast update scheme in In a simple contact scenario, a recursive corrective motion scheme with 5 Newton iterations is performed. Each Newton iteration is accompanied by a local constraint resolution with 30 **PGS** iterations. In the rebuilding scheme (see Algorithm 8), updating constraint matrices takes massive computation cost: Rebuilding the compliance matrix $\mathbf{W}$ and applying the constraint correction with traditional way (see Equation (4.19)) require to invert the system matrix $\mathbf{A}$. The extra cost by these updating processes is enormous, taking 84-90 % of each Newton iteration. On the other hand, in the fast updating scheme (see Algorithm 9), the computation of the mapping compliance matrix $\mathbf{W_H}$ is outside of the recursive scheme and is performed only once in each time step. Our method allows fast rebuilding of the compliance matrix (by matrix-matrix multiplications) and efficiently computing the correction on proximity positions (by matrix-vector multiplication). The matrix multiplication operations are very suitable to be parallelized on GPU. Moreover, the extra updating cost takes 14-15 % of each Newton iteration. Our method is 6.97 × faster than the rebuilding scheme for each Newton iteration. For the cost of the whole Newton scheme, including the overhead of building $\mathbf{W_H}$, our method benefits a

speedup of 3.20 × compared to the rebuilding scheme.

Besides the computational performance, our method is straightforward since only matrix operations are required in the recursive scheme. Moreover, the implementation of the method remains flexible. In fact, Algorithm 8 could be applied in different ways. Since the update process is very efficient, it is possible to update the constraint directions in each PGS iteration, or even to carry out the update after the resolution of each constraint, which is similar to the resolution scheme in the PBD Müller et al. (2007).

Our method enables the handling of inconsistencies in constraint directions between the beginning and end of time steps while maintaining an efficient resolution process. However, we have only evaluated the computing efficiency of the method, while the improvement of stability and behavior remains to be proven. Our future work is therefore to find more applications for the recursive corrective motion scheme with the fast update method.

6.2 Perspective: enhancements of virtual cutting

Owning to the mesh modification property employed in our cutting method, we have successfully enhanced the matrix-free iterative solver. By exploiting this same property, we present several potential advancements for existing approaches in this section.

6.2.1 Fast matrix assembly approach with topology change

In Section 3.2.1, we introduced a fast matrix assembly technique that relies on a deterministic mapping associated with the mesh topology. However, changes in the topology invalidate the mapping, necessitating a full matrix assembly to reconstruct the matrix pattern. As a result, our matrix assembly method encounters reduced efficiency during cut events, despite being as efficient as Eigen's implementation.

To address this, we propose an enhancement to the fast matrix assembly, integrated with the topology change in our cutting method discussed in Section 5.1. The upgrade follows a similar principle to the matrix-free iterative solver (discussed in Section 5.1.5). Splitting the mesh along existing surfaces results in a duplicated boundary and the modifications of the element indices. Based on the information obtained from the cutting process outcome, the modified connections (represented by pairs of row-column indices) can be identified in both the triplet set and the unmerged format. With this information, we can update the deterministic mapping. However, modifying the mapping can still be inefficient as it may require rearranging the mapping vector. In contrast, we propose

retaining the use of the deterministic mapping and performing a specific modification during the merge stage.

As depicted in Figure 6.3, we first gather the modified connections in the unmerged format and construct the CSR format for the newly generated *DOFs* at the end of the previous CSR matrix. Subsequently, the original portion of the CSR matrix can be built using the fast assembly strategy by clearing the values of all the modified elements.

Our proposed enhancement avoids the need to rebuild the deterministic mapping, employing minimal modifications during the merge process. Since local cutting typically affects only a limited number of elements in progressive simulations, the construction of CSR for new *DOFs* is highly efficient. Consequently, the fast assembly strategy can be extended to be compatible with cut events.

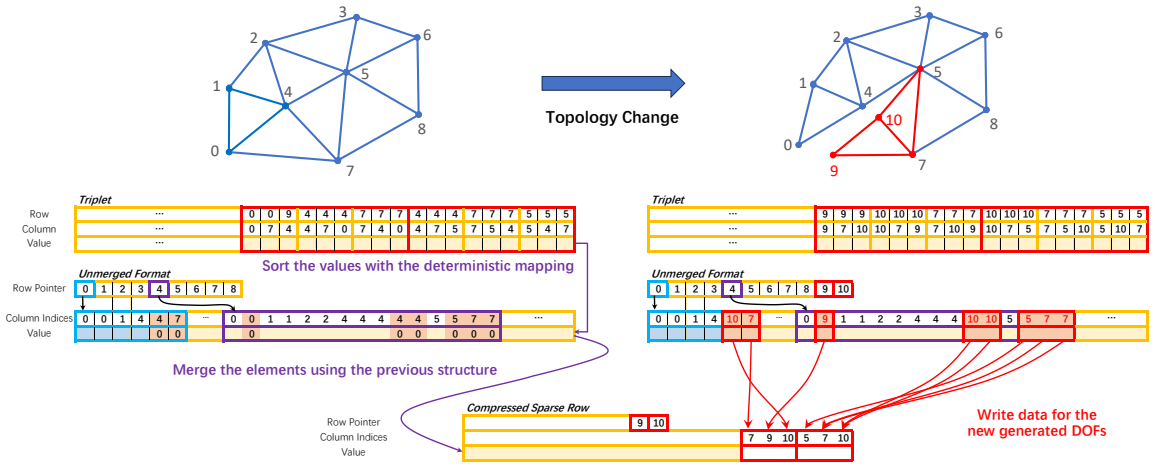


Figure 6.3: At the right: we collect the modified elements and build the CSR structure at the end of the CSR matrix. At the left: we clear the values of the modified elements and assemble the original part of the CSR matrix with the previous deterministic mapping.

6.2.2 Asynchronous cutting strategy

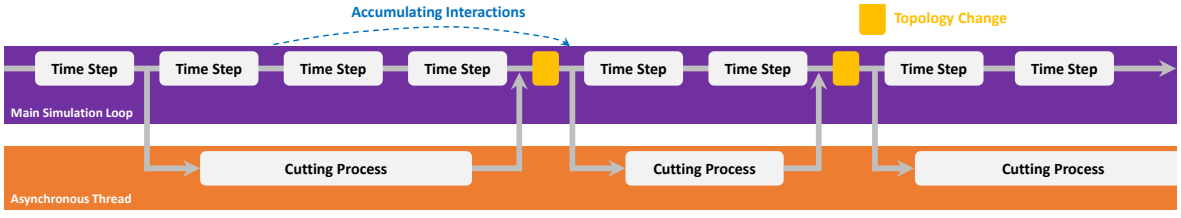


Figure 6.4: Executing the cutting process in an asynchronous thread.

The second potential avenue of our work involves implementing a the majority of our cutting method within an asynchronous thread. As indicated by the performance results

in Table 5.1, our method incurs substantial computational cost (73.8%) during virtual simulation in case of large-scale problems (10,000 vertices), while the actual simulation remains efficient.

We draw inspiration from the asynchronous preconditioner technique (see Section 3.3.1) and propose the following idea: after generating the point cloud, the remaining steps of the cutting process in the virtual simulation are independent of the main simulation loop. Therefore, we can move this computationally expensive process to a dedicated thread. Similar to the strategy employed in the asynchronous preconditioner, we do not synchronize the actual and virtual simulations within the current time step. Instead, we continue running the main simulation loop until the asynchronous thread returns the result of the virtual cutting. The resulting topology and state modifications are then processed in the actual simulation.

While this strategy offers high efficiency, it also presents a challenge. While the virtual simulation is running in the asynchronous thread, the actual system may continue to encounter interactions in subsequent time steps. However, these interactions cannot be immediately addressed because the asynchronous thread is still occupied by the previous cutting operation. To address this, we propose accumulating the results of the interactions until the next synchronization event. The generated point cloud is preserved and will be utilized after the next topology change. As a result, the modifications to the mesh and reference state will be updated at a lower frequency, determined by the efficiency of the virtual simulation.

The schematic representation of the proposed scheme is illustrated in Figure 6.4. Implementing this method introduces another challenge: how to combine this approach with the asynchronous preconditioner to reap the benefits of both asynchronous techniques. The potential solution to this challenge will be discussed in the subsequent section.

6.2.3 Updating contact compliance with topology change

The final potential extension of our cutting method aims to make it compatible with the methods in contact problem discussed in Chapter 4. In this context, the asynchronous preconditioning scheme proves highly efficient by moving the computationally expensive Cholesky factorization out of the main simulation loop to compute the Schur complement. o combine both the asynchronous cutting and asynchronous preconditioning, a primary idea is to set them into two dedicated threads. However, implementing this idea presents the challenge of synchronizing the threads with the main simulation loop. To address this, we define the following requirements in order of priority (from high to

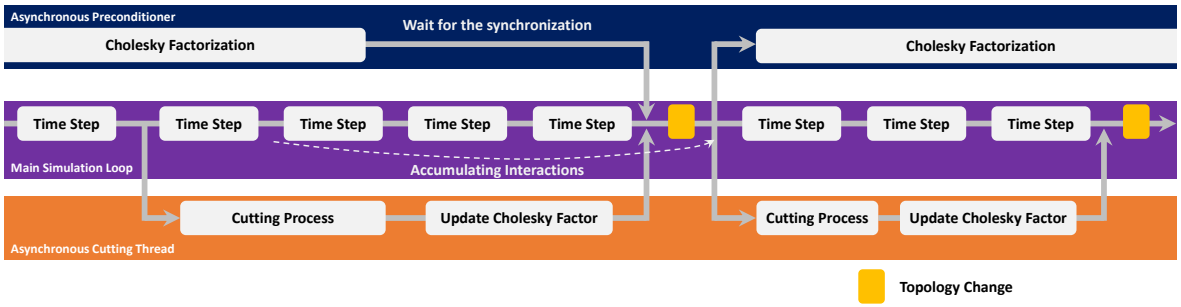


Figure 6.5: Asynchronous scheme, combining the precondition technique and the cutting method.

low):

1. The currently used Cholesky factor must be updated with topology changes.

2. The main simulation loop should be as efficient as possible.

3. The delay of the currently used Cholesky factor (from the start of its factorization) should be minimized.

In conjunction with the asynchronous cutting scheme proposed in Section 6.2.2, we propose an advanced scheme that combines both asynchronous methods, as illustrated in Figure 6.5: To fulfill the first and second requirements, we suggest incorporating a Cholesky factor update process after the cutting process to avoid blocking the main simulation loop. This involves transferring the current Cholesky factor to the dedicated thread at the beginning of the cutting process. To meet the third requirement, the full Cholesky decomposition is synchronized with the next topology change, and we determine the choice of the Cholesky factor to be transferred to the cutting thread as follows:

1. If a new full Cholesky factorization is completed just before the topology change, the cutting thread will utilize the newly factorized data.

2. Otherwise, the cutting thread will use the Cholesky factor updated by the previous cut.

As a result, the Cholesky factor used in the main simulation loop is ensured to be updated as frequently as possible in terms of data (when a new factorization is performed) and topology structure (otherwise).

Another challenge lies in efficiently updating the Cholesky factor with topology changes. The algorithm proposed in Herholz and Sorkine-Hornung (2020) specifically addresses

updating the symbolic factorization with duplication of *DOFs*, which aligns with the results of our cutting method. Therefore, a potential approach is to implement the algorithm from Herholz and Sorkine-Hornung (2020) to locally update the Cholesky factor, which is significantly more efficient than performing a full factorization. Additionally, it will be straightforward to extend the *isodof* method (discussed in Section 4.2.3) to the proposed scheme since the precomputed data in the method corresponds to the branches for each *isodof index* in the elimination tree. Updating the symbolic factorization will directly yield the updated elimination tree, thereby enabling compatibility of the *isodof* method with cut events.

6.3 Conclusion

In this manuscript, we have presented an overview of physical simulations and discussed the current state of the art in the field, focusing on topics such as simulating solid deformation, contact problems, coupled systems, and virtual cutting.

In Chapter 3, we provided a comprehensive background on physics-based simulations for deformable solids, along with the challenges associated with parallelizing different hyperelastic models uniformly. To address this challenge, we introduced a novel matrix assembly strategy Zeng and Courtecuisse (2023b) that offers high efficiency and accommodates various constitutive models. Additionally, we replaced the CPU-based preconditioner with a GPU-based implementation during the solving stage, reducing data transfer between the CPU and GPU and enabling a fully GPU-based CG solver. Chapter 4 focused on the background and challenges in the context of contact problems. To tackle the computational cost associated with building the compliance matrix, we proposed the efficient *isodof* method and the enhanced *reuse isodof* method Zeng et al. (2022). These methods involve a reformulation of the contact *Jacobian* matrix and efficient GPU-based implementation. In Chapter 5, we presented a novel cutting method that enables progressive cutting with high computational efficiency, good mesh quality, and smooth cutting surfaces. By leveraging the mesh modification property employed in the cutting method, we successfully enhanced the matrix-free iterative solver.

Finally in the current chapter we introduced several potential approaches that remain further researches on their implementations and applications. For the contact problem, We introduced a recursive motion correction scheme Zeng and Courtecuisse (2023a) to address inconsistencies in constraint directions between the initial guess and the end of time steps, as well as an efficient updating process for constraint matrices during iterations. In a similar principle of the upgraded matrix-free iterative solver, we proposed a

potential advancements for fast matrix assembly. We also discussed a novel asynchronous cutting scheme and an enhancement for the *isodof* method to enable compatibility with cut events and topology changes. The proposed solutions give the possibility to address the major drawback of the approaches in the previous chapters by effectively handling topology changes.

Although our contributions involves various problems such as linear system resolution, contact problems, coupled systems, and virtual cutting, our overarching objective in these works is to enhance computing performance in large-scale simulations. Through the proposed methods and experimentation results, our contributions enable real-time simulations on a large scale (up to 10,000 -15,000 vertices), with particular emphasis on contact problems and virtual cutting. We hope that our proposed approaches will benefit other researchers and engineers in the field of medical simulations and soft robotics, and that one day they may be applied in surgical simulations, providing surgeons with more detailed relevant information in real time.

BIBLIOGRAPHY

Adagolodjo, Y., Goffin, L., De Mathelin, M., and Courtecuisse, H. (2019). Robotic Insertion of Flexible Needle in Deformable Structures Using Inverse Finite-Element Simulation. *IEEE Transactions on Robotics*, 35(3):697–708.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. (2003). Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15.

Allard, J., Courtecuisse, H., and Faure, F. (2011). Implicit FEM and fluid coupling on GPU for interactive multiphysics simulation. In *ACM SIGGRAPH 2011 Talks, SIGGRAPH'11*, page 1, New York, New York, USA. ACM Press.

Allard, J., Courtecuisse, H., and Faure, F. (2012). Chapter 21 - implicit fem solver on gpu for interactive deformation simulation. In mei W. Hwu, W., editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 281–294. Morgan Kaufmann, Boston.

Allard, J., Faure, F., Courtecuisse, H., Falipou, F., Duriez, C., and Kry, P. G. (2010). Volume contact constraints at arbitrary resolution. *ACM SIGGRAPH 2010 Papers, SIGGRAPH 2010*, C(3):1–10.

Andrews, S. and Erleben, K. (2021). Contact and friction simulation for computer graphics. *ACM SIGGRAPH 2021 Courses, SIGGRAPH 2021*.

Anitescu, M., Potra, F. A., and Stewart, D. E. (1999). Time-stepping for three-dimensional rigid body dynamics. *Computer Methods in Applied Mechanics and Engineering*, 177:183–197.

Anzt, H., Chow, E., and Dongarra, J. (2015). Iterative sparse triangular solves for preconditioning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9233:650–661.

Bansal, M., Singh, I., Mishra, B., and Bordas, S. (2019). A parallel and efficient multi-split xfem for 3-d analysis of heterogeneous materials. *Computer Methods in Applied Mechanics and Engineering*, 347:365–401.

Baraff, D. (1996). Linear-time dynamics using Lagrange multipliers. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996*, pages 137–146. ACM.

Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998*, pages 43–54.

Barbič, J. and James, D. (2005). Real-time subspace integration for St.Venant-Kirchhoff deformable models. *ACM Transactions on Graphics*, 24(3):982–990.

Bargteil, A. W., Shinar, T., and Kry, P. G. (2020). An introduction to physics-based animation. In *SIGGRAPH Asia 2020 Courses*, SA '20, New York, NY, USA. Association for Computing Machinery.

Bell, N. and Garland, M. (2009). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09*, page 1, New York, New York, USA. ACM Press.

Bielser, D., Maiwald, V. A., and Gross, M. H. (1999). Interactive cuts through 3-dimensional soft tissue. *Computer Graphics Forum*, 18(3):31–38.

Bolz, J., Farmer, I., Grinspun, E., and Schröder, P. (2005). Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *ACM SIGGRAPH 2005 Courses, SIGGRAPH 2005*.

Bouaziz, S., Martin, S., Liu, T., Kavan, L., and Pauly, M. (2014). Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4).

Bradley, A. M. (2016). A hybrid multithreaded direct sparse triangular solver. *2016 Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing*, pages 13–22.

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

Buatois, L., Caumon, G., and Lévy, B. (2009). Concurrent number cruncher: a GPU implementation of a general sparse linear solver. *International Journal of Parallel, Emergent and Distributed Systems*, 24(3):205–223.

Burkhart, D., Hamann, B., and Umlauf, G. (2010). Adaptive and feature-preserving subdivision for high-quality tetrahedral meshes. *Computer Graphics Forum*, 29:117–127.

Comas, O., Taylor, Z. A., Allard, J., Ourselin, S., Cotin, S., and Passenger, J. (2008). Efficient Nonlinear FEM for Soft Tissue Modelling and Its GPU Implementation within the Open Source Framework SOFA. In *Biomedical Simulation*, volume 5104 LNCS, pages 28–39.

Cotin, S., Delingette, H., and Ayache, N. (2000). Hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. *Visual Computer*, 16(8):437–452.

Courtecuisse, H., Adagolodjo, Y., Delingette, H., and Duriez, C. (2015). Haptic rendering of hyperelastic models with friction. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem:591–596.

Courtecuisse, H. and Allard, J. (2009). Parallel dense gauss-seidel algorithm on many-core processors. In *2009 11th IEEE International Conference on High Performance Computing and Communications, HPCC 2009*, pages 139–147. IEEE.

Courtecuisse, H., Allard, J., Duriez, C., and Cotin, S. (2010a). Asynchronous preconditioners for efficient solving of non-linear deformations. In *VRIPHYS 2010 - 7th Workshop on Virtual Reality Interactions and Physical Simulations*, pages 59–68.

Courtecuisse, H., Allard, J., Kerfriden, P., Bordas, S. P., Cotin, S., and Duriez, C. (2014). Real-time simulation of contact and cutting of heterogeneous soft-tissues. *Medical Image Analysis*, 18(2):394–410.

Courtecuisse, H., Jung, H., Allard, J., Duriez, C., Lee, D. Y., and Cotin, S. (2010b). GPU-based real-time soft tissue deformation with cutting and haptic feedback. *Progress in Biophysics and Molecular Biology*, 103(2-3):159–168.

Dally, W. J., Keckler, S. W., and Kirk, D. B. (2021). Evolution of the graphics processing unit (gpu). *IEEE Micro*, 41(6):42–51.

Duriez, C. (2013). Control of elastic soft robots based on real-time finite element method. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3982–3987.

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

Duriez, C., Guébert, C., Marchal, M., Cotin, S., and Grisoni, L. (2009). Interactive simulation of flexible needle insertions based on constraint models. *Lecture Notes in Computer Science*, 5762 LNCS(PART 2):291–299.

Dziekonski, A., Sypek, P., Lamecki, A., and Mrozowski, M. (2012). Finite element matrix generation on a GPU. In *Progress in Electromagnetics Research*, volume 128, pages 249–265. Electromagnetics Academy.

Erleben, K. (2013). Numerical methods for linear complementarity problems in physics-based animation. *ACM SIGGRAPH 2013 Courses, SIGGRAPH 2013*, (February).

Faure, F., Duriez, C., Delingette, H., Allard, J., Gilles, B., Marchesseau, S., Talbot, H., Courtecuisse, H., Bousquet, G., Peterlik, I., and Cotin, S. (2012). *SOFA: A Multi-Model Framework for Interactive Physical Simulation*, volume 11.

Felippa, C. (2000). A systematic approach to the element-independent corotational dynamics of finite elements. Technical Report January, College Of Engineeringuniversity Of Colorado.

Forest, C., Delingette, H., and Ayache, N. (2005). Removing tetrahedra from manifold tetrahedralisation: application to real-time surgical simulation. *Medical Image Analysis*, 9:113–122.

Fu, M., Kuntz, A., Webster, R. J., and Alterovitz, R. (2018). Safe Motion Planning for Steerable Needles Using Cost Maps Automatically Extracted from Pulmonary Images. *IEEE International Conference on Intelligent Robots and Systems*, pages 4942–4949.

Galoppo, N., Otaduy, M. A., Mecklenburg, P., Gross, M., and Lin, M. C. (2006). Fast simulation of deformable models in contact using dynamic deformation textures. *Computer Animation, Conference Proceedings*, 02-04-September-2006:73–82.

George, A. (1973). Nested Dissection of a Regular Finite Element Mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363.

Hager, W. W. (1989). Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239.

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

Herholz, P. and Alexa, M. (2018). Factor once: Reusing Cholesky factorizations on submeshes. *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*, 37(6):1–9.

Herholz, P. and Sorkine-Hornung, O. (2020). Sparse cholesky updates for interactive mesh parameterization. *ACM Transactions on Graphics*, 39(6).

Hermann, E., Raffin, B., and Faure, F. (2009). Interactive Physical Simulation on Multicore Architectures. In *Time*, pages 1–9.

Hiemstra, R. R., Sangalli, G., Tani, M., Calabrò, F., and Hughes, T. J. (2019). Fast formation and assembly of finite element matrices with application to isogeometric linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 355:234–260.

Hubbard, P. M. (1996). Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph.*, 15(3):179–210.

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

Jin, X., Joldes, G., Miller, K., Yang, K., and Wittek, A. (2012). Meshless algorithm for soft tissue cutting in surgical simulation. *Computer methods in biomechanics and biomedical engineering*, 17.

Joldes, G. R., Wittek, A., and 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.

Kugelstadt, T., Koschier, D., and Bender, J. (2018). Fast Corotated FEM using Operator Splitting. *Computer Graphics Forum*, 37(8):149–160.

Lan, L., Luo, R., Fratarcangeli, M., Xu, W., Wang, H., Guo, X., Yao, J., and Yang, Y. (2020). Medial elastics: Efficient and collision-ready deformation via medial axis transform. *ACM Trans. Graph.*, 39(3).

Larsson, T. and Akenine-Möller, T. (2001). Collision detection for continuously deforming bodies. In *Eurographics*.

Li, J., Liu, T., Kavan, L., and Chen, B. (2021). Interactive cutting and tearing in projective dynamics with progressive cholesky updates. *ACM Transactions on Graphics (TOG)*, 40.

Li, M., Ferguson, Z., Schneider, T., Langlois, T., Zorin, D., Panozzo, D., Jiang, C., and Kaufman, D. M. (2020). Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Trans. Graph. (SIGGRAPH)*, 39(4).

Li, P., Wang, B., Sun, F., Guo, X., Zhang, C., and Wang, W. (2016). Q-mat: Computing medial axis transform by quadratic error minimization. *ACM Trans. Graph.*, 35(1).

Li, R. and Zhang, C. (2020). Efficient parallel implementations of sparse triangular solves for gpu architectures. *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing*, pages 106–117.

Liu, T., Bargteil, A. W., O'Brien, J. F., and Kavan, L. (2013). Fast simulation of mass-spring systems. *ACM Trans. Graph.*, 32(6).

Liu, T., Bouaziz, S., and Kavan, L. (2017). Quasi-Newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics*, 36(3).

Ly, M., Jouve, J., Boissieux, L., and Bertails-Descoubes, F. (2020). Projective dynamics with dry frictional contact. *ACM Trans. Graph.*, 39(4).

Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S., and Makoviychuk, V. (2019). Non-smooth Newton methods for deformable multi-body dynamics. *ACM Transactions on Graphics*, 38(5).

Macklin, M., Müller, M., and Chentanez, N. (2016). Xpbd: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, MIG '16, page 49–54, New York, NY, USA. Association for Computing Machinery.

Marchesseau, S., Heimann, T., Chatelin, S., Willinger, R., and Delingette, H. (2010). Multiplicative Jacobian Energy Decomposition Method for Fast Porous Visco-Hyperelastic Soft Tissue Model. In *Lecture notes in computer science*, volume 6361, pages 235–242. Springer.

Martin, C., Zeng, Z., and Courtecuisse, H. (2023). Efficient Needle Insertion Simulation using Hybrid Constraint Solver and Isolated DOFs. In Babaei, V. and Skouras, M., editors, *Eurographics 2023 - Short Papers*. The Eurographics Association.

Martínez-Frutos, J., Martínez-Castejón, P. J., and Herrero-Pérez, D. (2015). Fine-grained GPU implementation of assembly-free iterative solver for finite element problems. *Computers and Structures*, 157:9–18.

Marton, Z. C., Rusu, R. B., and Beetz, M. (2009). On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan.

Masterjohn, J., Guoy, D., Shepherd, J., and Castro, A. (2022). Velocity level approximation of pressure field contact patches. *IEEE Robotics and Automation Letters*, 7(4):11593–11600.

Molino, N., Bao, Z., and Fedkiw, R. (2004). A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics (TOG)*, 23:385–392.

Müller, E., Guo, X., Scheichl, R., and Shi, S. (2013). Matrix-free gpu implementation of a preconditioned conjugate gradient solver for anisotropic elliptic pdes. *Computing and Visualization in Science*, 16(2):41–58.

Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. (2007). Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118.

Müller, M., Heidelberger, B., Teschner, M., and Gross, M. (2005). Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478.

Nguyen, V. P., Rabczuk, T., Bordas, S., and Duflot, M. (2008). Meshless methods: A review and computer implementation aspects. *Mathematics and Computers in Simulation*, 79(3):763–813.

Nienhuys, H. W. and van der Stappe, A. F. (2001). A surgery simulation supporting cuts and finite element deformation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2208:145–152.

Nienhuys, H.-W. and van der Stappen, A. (2000). Combining finite element deformation with cutting for surgery simulations. In *Eurographics*.

Parker, E. G. and O'Brien, J. F. (2009). Real-time deformation and fracture in a game environment. In *Computer Animation, Conference Proceedings*, pages 165–175.

Paulus, C. J., Untereiner, L., Courtecuisse, H., Cotin, S., and Cazier, D. (2015a). Virtual cutting of deformable objects based on efficient topological operations. *Visual Computer*, 31:831–841.

Paulus, C. J., Untereiner, L., Courtecuisse, H., Cotin, S., and Cazier, D. (2015b). Virtual cutting of deformable objects based on efficient topological operations. *Visual Computer*, 31(6-8):831–841.

Peng, X., Kulasegaram, S., Bordas, S., and Wu, S. (2014). An extended finite element method (xfem) for linear elastic fracture with smooth nodal stress. *Engineering Fracture Mechanics*.

Petra, C. G., Schenk, O., Lubin, M., and Gäertner, K. (2014). An augmented incomplete factorization approach for computing the schur complement in stochastic optimization. *SIAM Journal on Scientific Computing*, 36(2):C139–C162.

Picciau, A., Inggs, G. E., Wickerson, J., Kerrigan, E. C., and Constantinides, G. A. (2017). Balancing locality and concurrency: Solving sparse triangular systems on gpus. *Proceedings - 23rd IEEE International Conference on High Performance Computing, HiPC 2016*, pages 183–192.

Reissner, E. (1985). On mixed variational formulations in finite elasticity. *Acta Mechanica*, 56:117–125.

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

Rivers, A. R. and James, D. L. (2007). Fastlsm: Fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph.*, 26(3):82–es.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, society fo edition.

Saupin, G., Duriez, C., and Cotin, S. (2008a). Contact model for haptic medical simulations. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5104 LNCS, pages 157–165, Berlin, Heidelberg. Springer-Verlag.

Saupin, G., Duriez, C., Cotin, S., and Grisoni, L. (2008b). Efficient Contact Modeling using Compliance Warping. In *Computer graphics international*.

Schenk, O. and Gärtner, K. (2006). On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158–179.

Serby, D., Harders, M., and Székely, G. (2001). A new approach to cutting into finite element models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2208:425–433.

Sifakis, E. and Barbic, J. (2012). Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, New York, NY, USA. Association for Computing Machinery.

Sifakis, E., Der, K. G., and Fedkiw, R. (2007). Arbitrary Cutting of Deformable Tetrahedralized Objects. In Metaxas, D. and Popovic, J., editors, *Eurographics/SIGGRAPH Symposium on Computer Animation*. The Eurographics Association.

Steinemann, D., Harders, M., Gross, M., and Szekely, G. (2006). Hybrid cutting of deformable solids. *Proceedings - IEEE Virtual Reality*, 2006:35–42.

Thomas, S., Yamazaki, I., Berger-Vergiat, L., Kelley, B., Hu, J., Mullowney, P., Rajamanickam, S., and Katarzyna´swirydowicz, K. K. (2021). Two-stage gauss–seidel preconditioners and smoothers for krylov solvers on a gpu cluster.

Trusty, T., Kaufman, D., and Levin, D. I. (2022). Mixed variational finite elements for implicit simulation of deformables. In *SIGGRAPH Asia 2022 Conference Papers*, SA '22, New York, NY, USA. Association for Computing Machinery.

Wang, M. and Ma, Y. (2018). A review of virtual cutting methods and technology in deformable objects. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 14:e1923.

Wang, Y., Jiang, C., Schroeder, C., and Teran, J. (2014). An Adaptive Virtual Node Algorithm with Robust Mesh Cutting. In Koltun, V. and Sifakis, E., editors, *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*. The Eurographics Association.

Wu, J., Westermann, R., and Dick, C. (2015). A survey of physically based simulation of cuts in deformable bodies. *Computer Graphics Forum*, 34:161–187.

Xu, H. and Barbič, J. (2014). Signed distance fields for polygon soup meshes. *Graphics Interface 2014*.

Xu, L. and Liu, Q. (2018). Real-time inextensible surgical thread simulation. *International Journal of Computer Assisted Radiology and Surgery*, 13:1019–1035.

Yamazaki, I., Rajamanickam, S., and Ellingwood, N. (2020). Performance portable supernode-based sparse triangular solver for manycore architectures. *ACM International Conference Proceeding Series*.

Yeung, Y. H., Crouch, J., and Pothen, A. (2016). Interactively cutting and constraining vertices in meshes using augmented matrices. *ACM Transactions on Graphics*, 35.

Yeung, Y. H., Pothen, A., and Crouch, J. (2020). Amps: Real-time mesh cutting with augmented matrices for surgical simulations. *Numerical Linear Algebra with Applications*, 27:e2323.

Zachmann, G. (2002). Minimal hierarchical collision detection. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '02, page 121–128, New York, NY, USA. Association for Computing Machinery.

Zayer, R., Steinberger, M., and Seidel, H. P. (2017). Sparse matrix assembly on the GPU through multiplication patterns. In *2017 IEEE High Performance Extreme Computing Conference, HPEC 2017*.

Zeng, Z., Cotin, S., and Courtecuisse, H. (2022). Real-time fe simulation for large-scale problems using precondition-based contact resolution and isolated dofs constraints. *Computer Graphics Forum*, 41:418–434.

Zeng, Z. and Courtecuisse, H. (2023a). An efficient implicit constraint resolution scheme for interactive fe simulations.

Zeng, Z. and Courtecuisse, H. (2023b). Efficient parallelization strategy for real-time fe simulations.

Zhang, G., Wittek, A., Joldes, G., Jin, X., and Miller, K. (2014). A three-dimensional nonlinear meshfree algorithm for simulating mechanical responses of soft tissue. *Engineering Analysis with Boundary Elements*, 42:60–66. Advances on Meshfree and other Mesh reduction methods.

Zhang, J., Zhong, Y., and Gu, C. (2018). Deformable models for surgical simulation: A survey. *IEEE Reviews in Biomedical Engineering*, 11:143–164.

LIST OF FIGURES

List of Tables

Vers des performances en temps réel dans les simulations physiques à grande échelle

Les simulations médicales ont suscité un intérêt considérable car elles offrent un environnement sûr pour l'apprentissage et la pratique d'interventions complexes, permettant ainsi de répondre aux préoccupations éthiques liées à l'expérimentation humaine directe. Pour atteindre des simulations interactives, elles doivent satisfaire simultanément aux exigences contradictoires de précision et de rapidité de calcul. D'une part, la préférence est d'utiliser des modèles complexes capables de fournir des prédictions réalistes sur le comportement complexe des organes pendant la chirurgie. D'autre part, les performances en temps réel sont nécessaires dans les simulations chirurgicales et les scénarios de formation. Ces applications nécessitent des simulations interactives et dynamiques capables de répondre rapidement aux entrées de l'utilisateur et de fournir des retours immédiats. Les principaux défis de l'informatique en temps réel découlent de divers facteurs tels que les déformations, les interactions et les modifications de topologie. L'objective principal de la thèse est d'améliorer les performances de calcul pour permettre des simulations en temps réel à grande échelle.

Nous étudions d'abord les solveurs numériques dans les cadres de simulation en temps réel, en nous concentrant spécifiquement sur le solveur itératif sans assembler la matrice du système. La méthode le plus efficace dans SOFA, qui est optimisée pour une implémentation basée sur GPU, est conçu pour l'élasticité linéaire ou la formulation co-rotationnelle. Cependant, les modèles hyperélastiques nécessitent un assemblage de matrices, ce qui entraîne des coûts supplémentaires lors de l'utilisation d'implémentations typiques comme la bibliothèque Eigen. Pour résoudre ce problème, nous présentons une méthode rapide d'assemblage de matrices, compatible avec des modèles constitutifs génériques.

Notre approche exploite le fait que les matrices du système sont créées de manière déterministe tant que la topologie du maillage reste constante. L'utilisation du schéma de parcimonie de la matrice assemblée apporte des optimisations significatives à l'étape d'assemblage. En conséquence, les techniques développées de parallélisation basée sur GPU peuvent être directement appliquées avec le système assemblé. De plus, un schéma de préconditionnement de Cholesky asynchrone est utilisé pour améliorer la convergence du solveur du système. Sur cette base, un préconditionneur de Cholesky basé sur GPU est développé, réduisant considérablement les transferts de données entre le CPU et le GPU pendant l'étape de résolution. Nous évaluons les performances de notre méthode avec différents éléments de maillage et modèles hyperélastiques, et nous les comparons avec des approches typiques sur le CPU et le GPU.

Le deuxième problème que nous abordons concerne les simulations de contact, qui impliquent la résolution d'un système multi-objet couplé. Typiquement, ce problème est transformé en espace de contrainte, et le problème de complémentarité est ensuite résolu. Cependant, le calcul du complément de Schur de la matrice du système, qui est nécessaire pour cette approche, devient extrêmement chronophage lorsque le système a de grandes dimensions. Dans notre travail, nous proposons une méthode rapide pour gérer les simulations EF à grande échelle en présence de contact et de frottement. Notre approche utilise une résolution de contact basée sur un préconditionnement qui effectue une décomposition de Cholesky à basse fréquence.

En exploitant la parcimonie des matrices assemblées, nous proposons un schéma de calcul réduit et parallèle pour traiter le calcul coûteux du complément de Schur causé par un maillage détaillé et une réponse de contact précise. Un solveur efficace basé sur GPU est développé pour paralléliser le calcul, ce qui permet de réaliser des simulations en temps réel en présence de contraintes couplées pour la réponse au contact et au frottement. De plus, le préconditionneur est mis à jour à basse fréquence, ce qui implique la réutilisation du système factorisé. Pour accélérer davantage le processus, nous proposons une stratégie permettant de partager les informations de résolution entre les pas de temps consécutifs. Nous évaluons les performances de notre méthode dans diverses applications de contact et nous les comparons avec des approches typiques sur le CPU et le GPU.

Notre travail ultérieur présente une nouvelle méthode implicite pour résoudre les contraintes dans les simulations impliquant des contacts avec frottement. Au lieu d'utiliser le schéma de correction de mouvement standard, nous proposons une méthode itérative dans laquelle les forces de contrainte sont corrigées dans des itérations de Newton. Dans ce schéma, nous sommes capables de mettre à jour les directions des contraintes de manière récursive, ce qui permet une réponse au contact et au frottement plus précise. Cependant, la mise à jour des matrices de contraintes entraîne des coûts de calcul massifs.

Pour résoudre ce problème, nous proposons de séparer la direction des contraintes et la correspondance géométrique dans la matrice jacobienne de contact et de reformuler le complément de Schur de la matrice du système. Lorsqu'elle est combinée à une parallélisation basée sur GPU, cette reformulation fournit un processus de mise à jour très efficace pour les matrices de contraintes dans le schéma itératif de correction du mouvement récursif. Notre méthode permet de gérer l'incohérence des directions de contraintes au début et à la fin des pas de temps. En même temps, le processus de résolution est maintenu aussi efficace que possible. Nous évaluons les performances de notre schéma de mise à jour rapide dans une simulation de contact et nous les comparons avec le schéma de mise à jour standard.

Notre dernière étude se concentre sur la découpe virtuelle en temps réel. Les méthodes de raffinement de maillage, largement utilisées dans les travaux récents, génèrent un grand nombre de nouveaux degrés de liberté, ce qui augmente considérablement les coûts de calcul dans la solution numérique subséquente. Au lieu de cela, nous utilisons une stratégie de "node-snapping" comme base de notre nouvelle méthode pour atténuer ce problème. L'utilisation de cette stratégie nécessite de résoudre plusieurs problèmes critiques rencontrés dans les travaux précédents, tels que les éléments mal formés, les changements de topologie et la préservation de l'énergie.

Avec les solutions proposées, notre méthode de découpe consiste à utiliser une stratégie de "vertices-snapping" pour ajuster la surface frontière sur le chemin de découpe tout en évitant la génération de nouveaux éléments : En utilisant un nuage de points pour modéliser les découpes non programmées en temps réel, notre approche permet de travailler avec les utilisateurs. Nous proposons de résoudre un problème élastique afin de minimiser le changement de volume. Pour gérer la découpe tout en déformant et pour garantir la conservation de l'énergie, le problème élastique est transféré à l'état de référence de l'objet découpé et est résolu dans une deuxième simulation. Des opérations géométriques efficaces sont développées pour gérer les modifications topologiques lors de la découpe progressive. De plus, nous proposons une modification d'un solveur itératif de style ‘matrix-free’ rapide sur le GPU qui peut mettre à jour efficacement les données prétraitées utilisées dans les noyaux basés sur le GPU, garantissant ainsi des performances en temps réel pour les problèmes à grande échelle.

Dans la perspective de nos travaux, nous proposons des solutions pour surmonter les limitations de la méthode d'assemblage rapide et de la méthode "isolating mechanical DOFs" (reformulation du complément de Schur). En combinant ces solutions avec la nouvelle approche de découpe virtuelle, nous surmontons les défis liés à la mise à jour efficace des données précalculées dans ces méthodes, ce qui leur permet de fonctionner efficacement même avec des changements de topologie.

En conclusion, nos travaux apportent des solutions efficaces pour la simulation de la déformation, du contact et du frottement, ainsi que pour la découpe virtuelle. Grâce à l'intégration de la parallélisation basée sur GPU, notre recherche permet des simulations EF en temps réel pour les solides déformables, même avec des problèmes à grande échelle (10 000 - 15 000 nœuds), des contacts avec frottement et des opérations de découpe. Cela permet des interventions chirurgicales virtuelles interactives avec des formes d'objets plus détaillées, fournissant aux opérateurs des informations pertinentes en temps réel.

Liste des articles publiés:

1. Real-Time FE Simulation for Large-Scale Problems Using Precondition-Based Contact Resolution and Isolated DOFs Constraints, Z. Zeng et S. Cotin et H. Courtecuisse, *Computer Graphics Forum*, DOI : 10.1111/CGF.14563
2. Efficient Needle Insertion Simulation using Hybrid Constraint Solver and Isolated DOFs, Martin, Claire et Ziqiu Zeng et Hadrien Courtecuisse, *Eurographics 2023* - Short Papers, DOI : 10.2312/egs.20231003

Liste des manuscrits pré-publication ou des articles en phase de révision :

1. Efficient parallelization strategy for real-time FE simulations, Ziqiu Zeng et Hadrien Courtecuisse, DOI : https://doi.org/10.48550/arXiv.2306.05893
2. An efficient implicit constraint resolution scheme for interactive FE simulations, Ziqiu Zeng et Hadrien Courtecuisse, DOI : https://doi.org/10.48550/arXiv.2306.06946
3. Dynamic cutting simulation using elastic snapping for mesh quality optimization, Ziqiu Zeng et Hadrien Courtecuisse, soumis à *Computer Graphics Forum*

Université de Strasbourg

Ziqiu ZENG

iCUBE

Towards real-time performance in large-scale physics-based simulations

Abstract

Physics-based simulations have garnered considerable attention in the medical field, particularly in the application of virtual surgeries. A current focus of research in this field is centered on achieving realistic physical behaviors in real-time simulations of deformable objects. However, this presents a challenge as simulations must fulfill the conflicting requirements of both accuracy and fast computation time simultaneously. While fine discretization is preferred for capturing detailed shape information, it often leads to larger systems with the cost of increased computational costs. The primary objective of this manuscript is to enhance computing performance to enable real-time simulations on a large scale. To address this objective, our work encompasses several methods aimed at overcoming challenges in numerical system resolution within various problem domains. These methods include matrix assembly and parallel solver techniques for simulating elastic deformations, compliance assembly and constraint resolution approaches for simulating frictional contacts, and mesh quality optimization combined with an efficient solver for simulating virtual cuts. The effectiveness of these advancements is evaluated across various applications, enabling efficient simulations of deformation, contact and friction, as well as virtual cutting. As a consequence, the attainment of large-scale physical simulations in real-time becomes feasible.

Key words: Physics-based simulation, real-time simulation, finite element method, contact simulation, cutting simulation, GPU-based parallelization.

Université de Strasbourg

Ziqiu ZENG

iCUBE

Vers des performances en temps réel dans les simulations physiques à grande échelle

Résumé

Les simulations physiques ont suscité une attention considérable dans le domaine médical, en particulier dans le domaine des chirurgies virtuelles. Une préoccupation actuelle de la recherche dans ce domaine est centrée sur l'obtention de comportements physiques réalistes dans les simulations en temps réel d'objets déformables. Cependant, cela présente un défi car les simulations doivent concilier les exigences contradictoires de précision et de temps de calcul rapide simultanément. Bien que la finesse de la discrétisation soit préférée pour capturer des informations de forme détaillées, elle conduit souvent à des systèmes plus importants au prix d'une augmentation des coûts de calcul. L'objectif principal de ce manuscrit est d'améliorer les performances de calcul afin de permettre des simulations en temps réel à grande échelle. Pour atteindre cet objectif, notre travail englobe plusieurs méthodes visant à relever les défis de la résolution du système numérique dans différents domaines problématiques. Ces méthodes comprennent l'assemblage de matrices et les techniques de résolution parallèle pour simuler les déformations élastiques, l'assemblage de la matrice de conformité et les approches de résolution des contraintes pour simuler les contacts de friction, ainsi que l'optimisation de la qualité du maillage associée à un solveur efficace pour simuler les découpes. L'efficacité de ces avancées est évaluée dans diverses applications, permettant des simulations efficaces de la déformation, du contact et de la friction, ainsi que des coupes virtuelles. En conséquence, la réalisation de simulations physiques à grande échelle en temps réel devient réalisable.

Mots clés : Simulation physique, simulation en temps réel, méthode des éléments finis, simulation de contact, simulation de découpe, parallélisation sur le GPU.